Machine Learning Using Python

Title: Wireless Sound Control

Abstract

A hand gesture recognition system is a good way to communicate without words. It's natural, innovative & modern so it has wide applications in human computer interaction and sign language. The intention of this implementation is to discuss a novel approach of hand gesture recognition based on detection of some shape-based features. The setup consists of a single camera to capture the gesture formed by the user, which is then taken as input to the system.

The goal of gesture recognition is to create a system that can identify specific human gestures and use them to convey information for device control. By implementing real-time gesture recognition, a user can control a computer volume by doing a specific gesture in front of a video camera linked to a computer. In this tutorial, we use the OpenCV and MediaPipe libraries to create a gesture-controlled volume changing system. This saves not just time but also allows more intuitive control

Introduction

MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. This cross-platform Framework works on Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano.

MediaPipe Toolkit comprises the Framework and the Solutions.

Solutions are open-source pre-built examples based on a specific pre-trained TensorFlow or TFLite model.

MediaPipe Solutions are built on top of the Framework. Currently, it provides sixteen solutions as listed below.

1. Face Detection
2. Face Mesh
3. Iris
4. Hands
5. Pose
6. Holistic
7. Selfie Segmentation
8. Hair Segmentation
9. Object Detection
10. Box Tracking
11. Instant Motion Tracking
12. Objectron
13. KNIFT
14. AutoFlip
15. MediaSequence

16. YouTube 8M

The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. For example, it can form the basis for sign language understanding and hand gesture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality. While coming naturally to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other (e.g. finger/palm occlusions and hand shakes) and lack high contrast patterns.

MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference, our method achieves real-time performance on a mobile phone, and even scales to multiple hands. We hope that providing this hand perception functionality to the wider research and development community will result in an emergence of creative use cases, stimulating new applications and new research avenues.

ML Pipeline

MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints. This strategy is similar to that employed in our MediaPipe Face Mesh solution, which uses a face detector together with a face landmark model.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in our pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand.

The pipeline is implemented as a MediaPipe graph that uses a hand landmark tracking subgraph from the hand landmark module, and renders using a dedicated hand renderer subgraph. The hand landmark tracking subgraph internally uses a hand landmark subgraph from the same module and a palm detection subgraph from the palm detection module.

Note: To visualize a graph, copy the graph and paste it into MediaPipe Visualizer. For more information on how to visualize its associated subgraphs, please see visualizer documentation.

Models

Palm Detection Model

To detect initial hand locations, we designed a single-shot detector model optimized for mobile real-time uses in a manner similar to the face detection model in MediaPipe Face Mesh. Detecting hands is a decidedly complex task: our lite model and full model have to work across a variety of hand sizes with a large scale span (~20x) relative to the image frame and be able to detect occluded and self-

occluded hands. Whereas faces have high contrast patterns, e.g., in the eye and mouth region, the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localization.

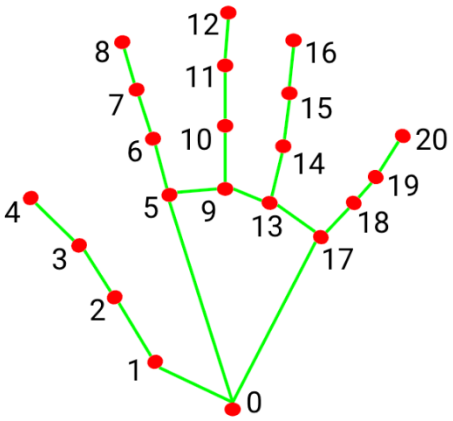
Our method addresses the above challenges using different strategies. First, we train a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology) ignoring other aspect ratios, and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects (similar to the RetinaNet approach). Lastly, we minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance.

With the above techniques, we achieve an average precision of 95.7% in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22%.

Hand Landmark Model

After the palm detection over the whole image our subsequent hand landmark model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, we have manually annotated ~30K real-world images with 21 3D coordinates, as shown below (we take Z-value from image depth map, if it exists per corresponding coordinate). To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, we also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Fig 2. 21 hand landmarks.

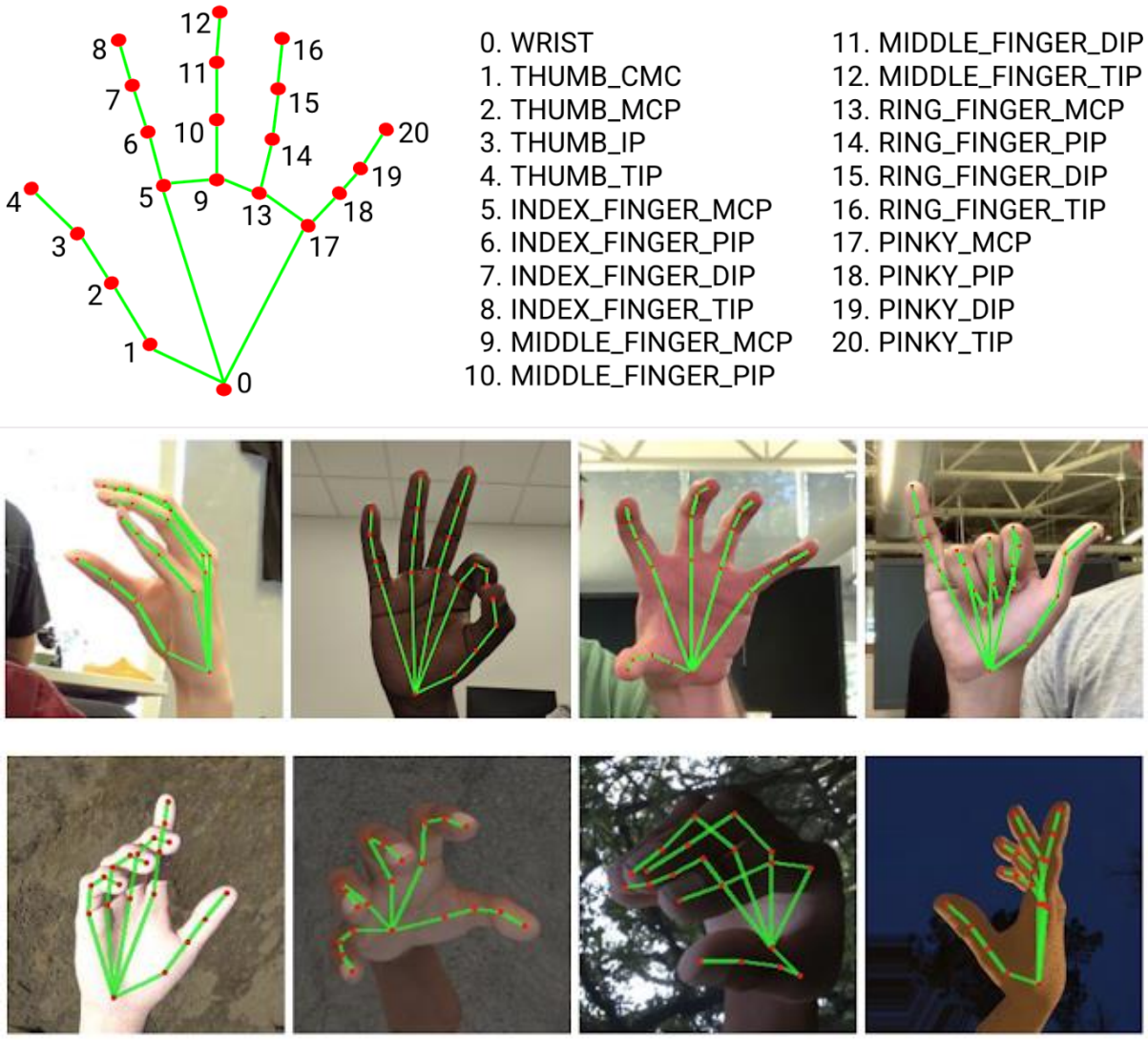


Fig 3. Top: Aligned hand crops passed to the tracking network with ground truth annotation. Bottom: Rendered synthetic hand images with ground truth annotation.

Solution APIs

Configuration Options

Naming style and availability may differ slightly across platforms/languages.

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 robustness of the solution, at the expense of a higher latency. Ignored if static image mode is `true`, where hand detection simply runs on every image. Default to `0.5`.

Output

Naming style may differ slightly across platforms/languages.

MULTI_HAND_LANDMARKS

Collection of detected/tracked hands, where each hand is represented as a list of 21 hand landmarks and each landmark is composed of `x`, `y` and `z`. `x` and `y` are normalized to `[0.0, 1.0]` by the image width and height respectively. `z` represents the landmark depth with the depth at the wrist being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of `z` uses roughly the same scale as `x`.

MULTI_HAND_WORLD_LANDMARKS

Collection of detected/tracked hands, where each hand is represented as a list of 21 hand landmarks in world coordinates. Each landmark is composed of `x`, `y` and `z`: real-world 3D coordinates in meters with the origin at the hand's approximate geometric center.

MULTI_HANDEDNESS

Collection of handedness of the detected/tracked hands (i.e. is it a left or right hand). Each hand is composed of `label` and `score`. `label` is a string of value either `"Left"` or `"Right"`. `score` is the estimated probability of the predicted handedness and is always greater than or equal to `0.5` (and the opposite handedness has an estimated probability of `1 - score`).

Note that handedness is determined assuming the input image is mirrored, i.e., taken with a front-facing/selfie camera with images flipped horizontally. If it is not the case, please swap the handedness output in the application.

Python Solution API

Please first follow general instructions to install MediaPipe Python package, then learn more in the companion Python Colab and the usage example below.

Volume Control Module

Python Core Audio Windows Library, working for both Python2 and Python3.

Install

Latest stable release:

```
pip install pycaw
```

Development branch:

```
pip install https://github.com/AndreMiras/pycaw/archive/develop.zip
```

System requirements:

```
choco install visualcpp-build-tools
```

Usage

```
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(
    IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volume.GetMute()
volume.GetMasterVolumeLevel()
volume.GetVolumeRange()
volume.SetMasterVolumeLevel(-20.0, None)
```

Code:

Hand_Tracking_module.py

```
import cv2
import mediapipe as mp
import time


class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionConfidence=1,
trackConfidence=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionConfidence = detectionConfidence
        self.trackConfidence = trackConfidence

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands,
                                         self.detectionConfidence,
self.trackConfidence)
        self.mpDraw = mp.solutions.drawing_utils

    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

    def findPosition(self, img, handNo=0, draw=True):

        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)
                # print(id, cx, cy)
                lmList.append([id, cx, cy])


        return lmList


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, f'FPS: {int(fps)}', (10, 70), cv2.FONT_HERSHEY_PLAIN,
3,
                    (255, 0, 255), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)
```

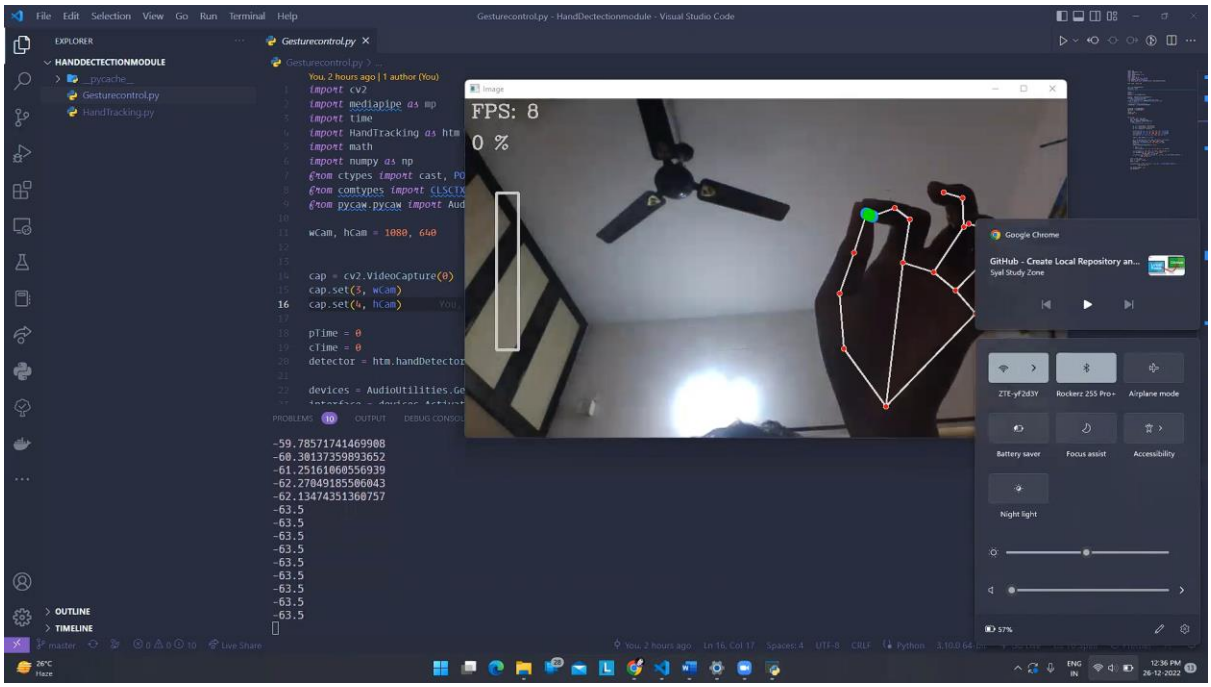
```
if __name__ == "__main__":
    main()
```

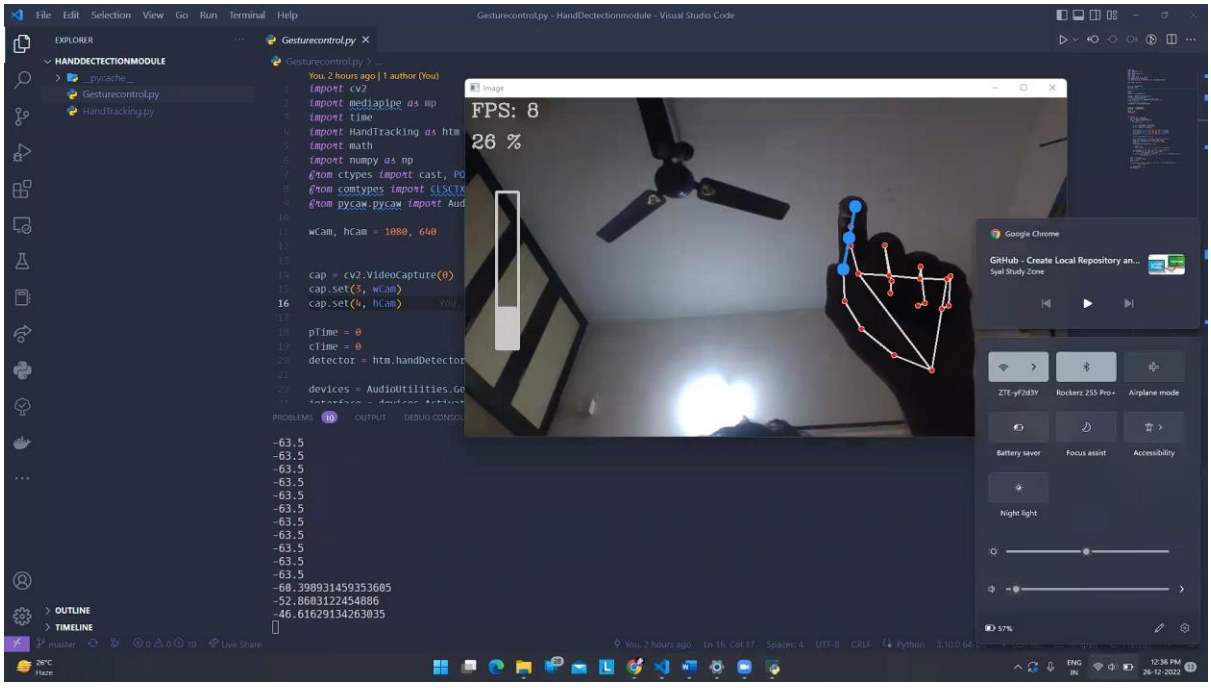
Gesture_Control.py

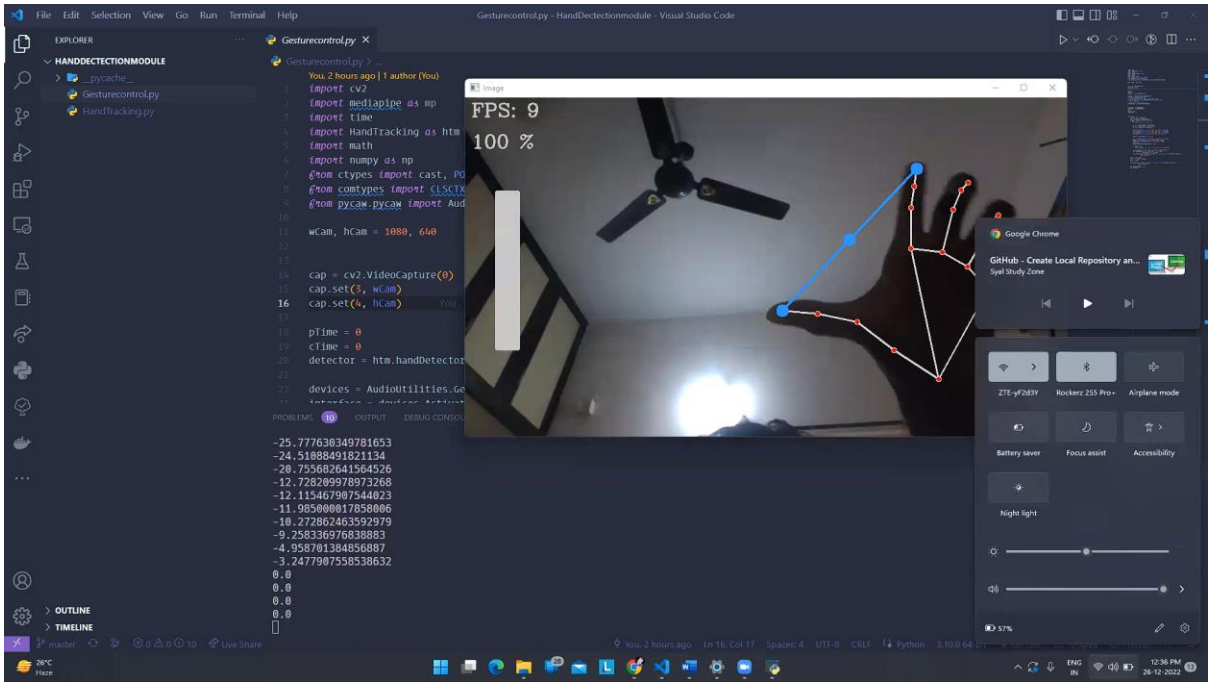
```
import cv2
import mediapipe as mp
import time
import HandTracking as htm
import math
import numpy as np
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

wCam, hCam = 1080, 640


cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)

pTime = 0
cTime = 0
detector = htm.handDetector()

devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(
    IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
# volume.GetMute()
# volume.GetMasterVolumeLevel()
volumeRange = volume.GetVolumeRange()


minVolume = volumeRange[0]
maxVolume = volumeRange[1]
vol = 0
volBar = 400
volPercent = 0


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

```
    if len(lmList) != 0:
        # print(lmList[4], lmList[8])

        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cx, cy = (x1+x2)//2, (y1+y2)//2

        cv2.circle(img, (x1, y1), 10, (255, 153, 51), cv2.FILLED)
        cv2.circle(img, (x2, y2), 10, (255, 153, 51), cv2.FILLED)
        cv2.circle(img, (cx, cy), 10, (255, 153, 51), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 153, 51), 3)

        length = math.hypot(x2-x1, y2-y1)

        vol = np.interp(length, [30, 300], [minVolume, maxVolume])
        volBar = np.interp(length, [30, 300], [400, 150])
        volPercent = np.interp(length, [30, 300], [0, 100])
        print(vol)
        volume.SetMasterVolumeLevel(vol, None)

        if length < 30:
            cv2.circle(img, (cx, cy), 10, (0, 255, 0), cv2.FILLED)

        cv2.rectangle(img, (50, 150), (85, 400), (200,200,200), 3)
        cv2.rectangle(img, (50, int(volBar)), (85, 400),
                      (200,200,200), cv2.FILLED)
        cv2.putText(img, f'{int(volPercent)} %', (10, 80),
cv2.FONT_HERSHEY_COMPLEX, 1,
                    (200,200,200), 2)

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime

    cv2.putText(img, f'FPS: {int(fps)}', (10, 30), cv2.FONT_HERSHEY_COMPLEX,
1,
                (200,200,200), 2)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Output:


File Edit Selection View Go Run Terminal Help
Gesturecontrol.py - HandDectectionmodule - Visual Studio Code
EXPLORER
HANDDECTECTIONMODULE
__pycache__
Gesturecontrol.py
HandTracking.py
Gesturecontrol.py
You, 2 hours ago | 1 author (You)
import cv2
import mediapipe as mp
import time
import HandTracking as htm
import math
import numpy as np
from ctypes import cast, POI
from comtypes import CLSCTX
from pycaw.pycaw import Aud
wCam, hCam = 1080, 640
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
pTime = 0
cTime = 0
detector = htm.handDetector
devices = AudioUtilities.Ge
PROBLEMS 10 OUTPUT DEBUG CONSOL
-59.78571741469908
-60.30137359893652
-61.25161060556939
-62.270491855006043
-62.13474351360757
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
OUTLINE
TIMELINE
master
Live Share
Ln 16, Col 17
Spaces: 4
UTF-8
CRLF
Python
3.10.0 64
Image
FPS: 8
0 %
Google Chrome
GitHub - Create Local Repository an...
Syel Study Zone
ZTE-yF2d3Y
Rockerz 255 Pro+
Airplane mode
Battery saver
Focus assist
Accessibility
Night light
57%
26°C
Haze
ENG
IN
12:36 PM
26-12-2022


File Edit Selection View Go Run Terminal Help
Gesturecontrol.py - HandDectectionmodule - Visual Studio Code
EXPLORER
HANDDECTECTIONMODULE
__pycache__
Gesturecontrol.py
HandTracking.py
Gesturecontrol.py
You, 2 hours ago | 1 author (You)
import cv2
import mediapipe as mp
import time
import HandTracking as htm
import math
import numpy as np
from ctypes import cast, POI
from comtypes import CLSCTX
from pycaw.pycaw import Aud
wCam, hCam = 1080, 640
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)
pTime = 0
cTime = 0
detector = htm.handDetector
devices = AudioUtilities.Ge
PROBLEMS 10 OUTPUT DEBUG CONSOL
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-63.5
-60.398931459353605
-52.8603122454886
-46.61629134263035
OUTLINE
TIMELINE
master
Live Share
Ln 16, Col 17
Spaces: 4
UTF-8
CRLF
Python
3.10.0 64
Image
FPS: 8
26 %
Google Chrome
GitHub - Create Local Repository an...
Syel Study Zone
ZTE-yF2d3Y
Rockerz 255 Pro+
Airplane mode
Battery saver
Focus assist
Accessibility
Night light
57%
26°C
Haze
ENG
IN
12:36 PM
26-12-2022



Conclusion:

This project will help to control sound wirelessly by running the python file.

The output of changing sound levels can be seen in the below sound bar.