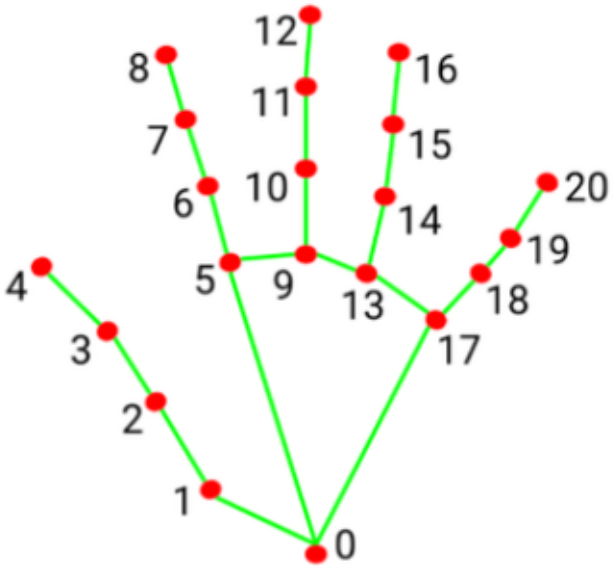
HAND GESTURE RECOGNITION



Amina Babić

10.11.2021.

PATTERN RECOGNITION AND IMAGE PROCESSING

Dr. sc. Samir Omanović

Faculty of Electrical Engineering
Sarajevo

INTRODUCTION

Hand gestures are form of communication and when those gestures are detected, they can be used for communication between deaf-mute people, robot control, home automation, human-computer interaction(HCI).

Examples:

- For people that are deaf-mute, we can create dataset of images that will be labeled with specific meaning. Some gestures can mean “Thank you”, “Yes”, “No”, “Welcome” etc. With this dataset we can train model and help with communication between people with specific characteristics.
- For robot control example would be that robot has some camera and trained model to recognize hand gestures and to repeat them after someone.
- For home automation example would be that we can with some gesture volume up or down music, turn off/on some device etc.
- For human-computer interaction example would be virtual quiz. We can turn on camera of our laptop/computer. On camera frame questions would be shown and we can select answer on question by some hand gesture.

These are just some examples of hand gesture detection use.

In this project I will use Mediapipe to implement hand gesture recognition/detection to volume up/down sound.

IMPLEMENTATION

- Using python 3.10.
- mediapipe
- Opencv-python
- Other modules will be listed later

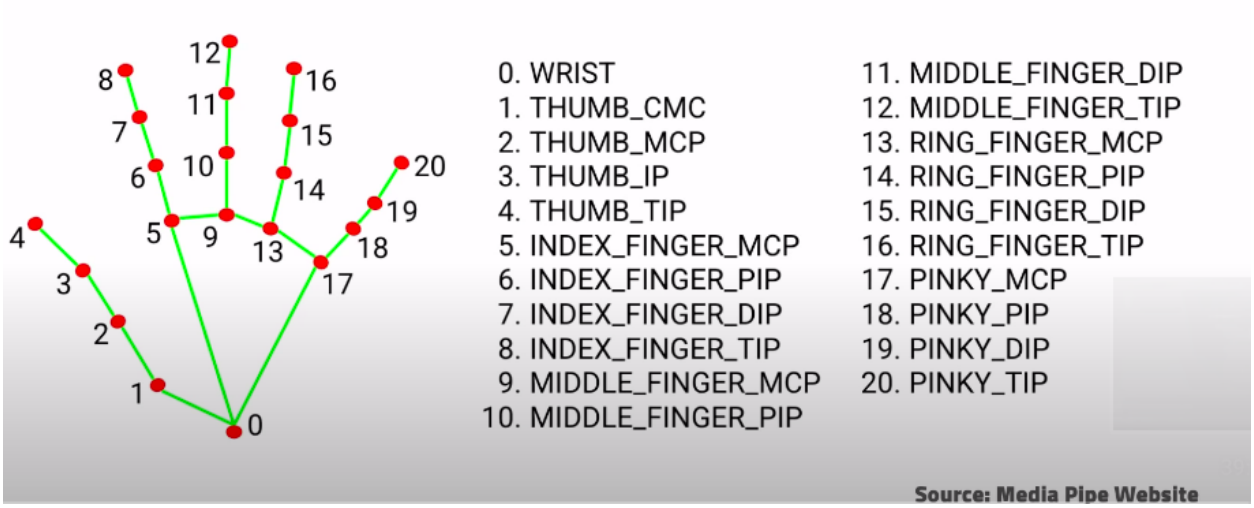
In this approach I will be using the framework Mediapipe (developed by Google). With mediapipe, they created some models that can be used for different purposes.

For example, I could create my own dataset of images using Tensorflow and OpenCV and then train some model to demonstrate this idea at the end. That would be possible but not good as Google team did. So, instead of that, I used Google model that is trained on really large dataset.

This model is already widely used for almost every type of recognition (face, body, movements, hans etc.)

Model that will be used in this project is the Hand Tracking model which is using module-hand landmarks.

Hand landmarks module finds 21 different landmarks on the image of the hand.

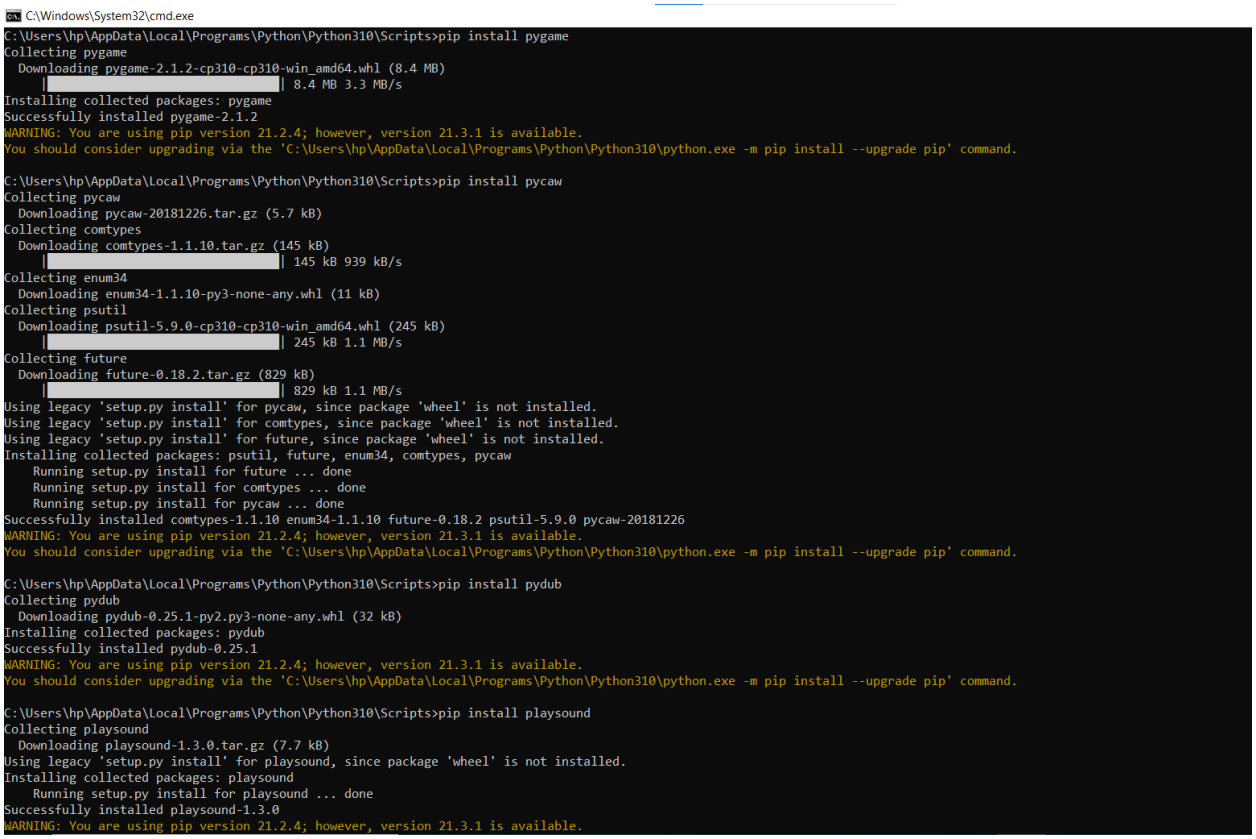


For this module, to train model, they made a dataset of 30.000 different hands.

This is the reason why this framework works well with hand tracking.

So, at the beginning of the project the OpenCV-python package and mediapipe package were installed.

Also, below is a screenshot of other modules that should be installed.



First of all, a module called HandTrackingModule was created.

This module is actually a detector. It creates model and detects the hand.

For example, this time I want to do something with a specific gesture, but next time I will have a different idea. For every new idea there will be steps that I will repeat.

Those steps are setting landmarks, drawing connections between them and finding the position of specific landmarks. To avoid repeats, a module is created with implemented functions *findPosition* and *findHand*.

Constructor

```
def __init__(self, mode=False, maxHands=1, detectionConfidence=0.5, trackConfidence=0.5):
    # parameters of model Hands
    self.mode = mode
    self.maxHands = maxHands
    self.detectionConfidence = detectionConfidence
    self.trackConfidence = trackConfidence
    # To get hand model from mediapipe
    self.mpHand = mp.solutions.hands
    # To set parameters
    self.hands = self.mpHand.Hands()
    #  To draw connections between landmarks
    #  Initializing the drawing utils for drawing the landmarks on image
    self.mpDraw = mp.solutions.drawing_utils
```

In this part is created object Hands and added landmarks and connections.

Model is actually from mediapipe and all that has to be done here is to set parameters and initialize drawing utils for drawing connections between landmarks.

METHOD findHand()

```
def findHand(self, img):
    # THIS METHOD WILL DETECT HAND
    # Object Hands can accept only rgb image
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgRGB)  # process as detect
    # If hands is not None, then for every detected hand draw landmarks and connections
    if self.results.multi_hand_landmarks:
        for singleHand in self.results.multi_hand_landmarks:
            # mediapipe provided us with function that will draw landmarks
            # otherwise there will be a lot of math for drawing dots and connections
            self.mpDraw.draw_landmarks(img, singleHand, self.mpHand.HAND_CONNECTIONS)
    return img
```

Method is explained through comments.

Since mediapipe object Hands can accept only rgb image, image from video capture is converted to imgRGB. This image is then processed to determine if there is a hand detected in that image. If there is a hand, or multiple of them, *multi_hand_landmarks* collection will not be empty.

MULTI_HAND_LANDMARKS
(https://google.github.io/mediapipe/solutions/hands.html#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 this method, hand is detected and landmarks are connected.

This method will change hand look on original image and will return that image.

Next step is to run this method. I will send image and change that image if hand is detected.

```
def main():
    cap = cv2.VideoCapture(0)

    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHand(img)
        cv2.imshow('MediaPipe Hands', cv2.flip(img, 1))
        cv2.waitKey(1)
```

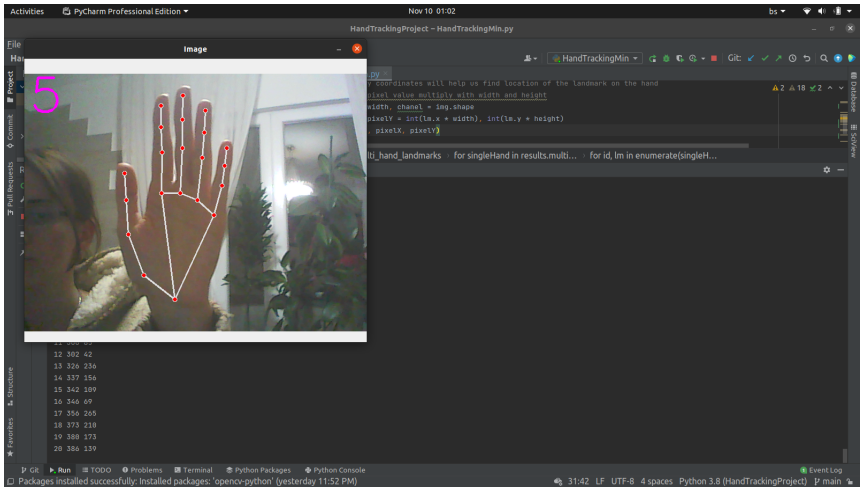
After running this part I will see landmarks and connections on my hand.

METHOD findPosition()

```python
def findPosition(self, img, handNo = 0, draw = False):
    lmList = []
    if self.results.multi_hand_landmarks:
        singleHand = self.results.multi_hand_landmarks[handNo]
        # get id and landmarks(x y coordinates)
        for id, lm in enumerate(singleHand.landmark):
            # x,y coordinates are reprecented as decimal x:0.93294 y:0.34723894
            # x and y coordinates will help us find location of the landmark on the hand
            # to get pixel value multiply with width and height
            height, width, chanel = img.shape
            pixelX, pixelY = int(lm.x * width), int(lm.y * height)
            lmList.append([id, pixelX, pixelY])
            if draw:
                cv2.circle(img, (pixelX, pixelY), 20, (255, 0, 255), cv2.FILLED)
    return lmList
```

If I print elements of lmList, this is one example of output:

```
0 271 451
1 242 418
2 232 386
3 232 361
4 236 342
5 262 370
6 253 345
7 233 352
8 220 366
9 279 368
10 261 347
11 238 358
12 220 373
13 300 370
14 281 349
15 254 357
16 236 373
17 326 375
18 317 356
19 297 358
20 281 368
```

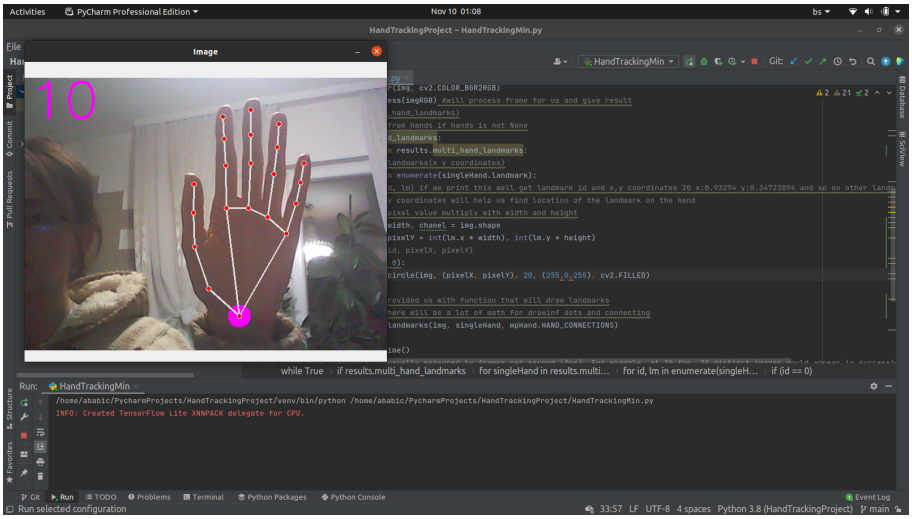


On console output is, respectively, landmark_id, positionX, positionY.

positionX and positionY are coordinates of specific landmarks and use of this will be explained later.

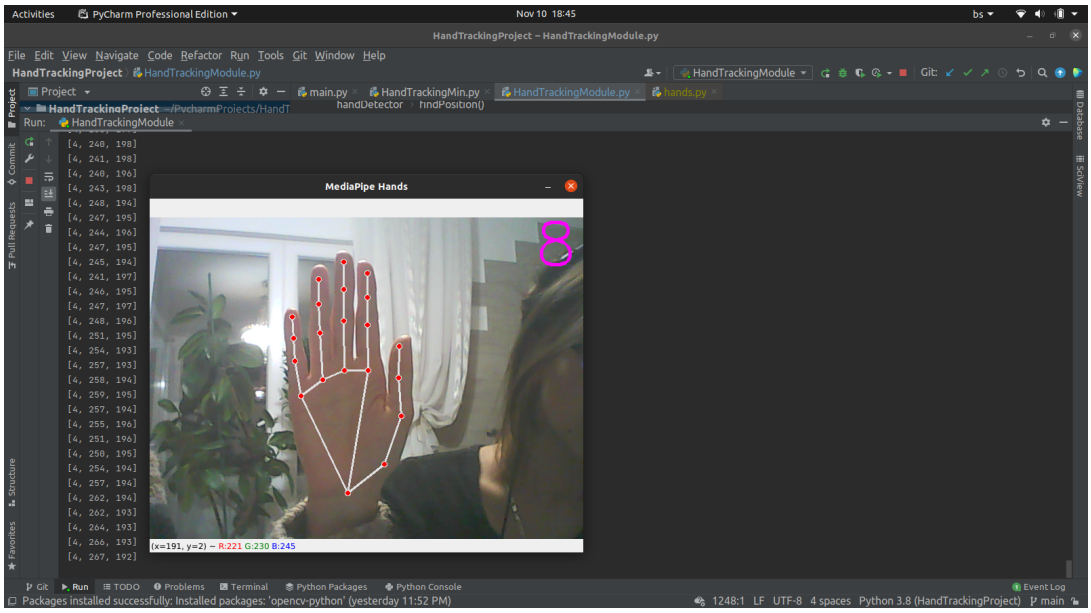
Id is important because it gives us access to the landmark.

For example if id is zero, we can access to it and draw circle



To find position of landmark 4, we will call this method and get lmList[4]

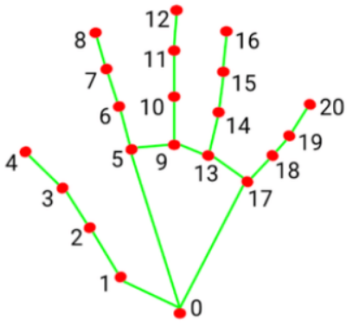
```
lmList = detector.findPosition(img)
if len(lmList) != 0:
    print(lmList[4])
```



Volume Hand Control

After creating, previous module can be used for different purposes.

The idea is to use top of two fingers and to calculate distance between them. That result will define volume.



I will use landmarks 4 and 8 for this.

First I will import previous module with *import HandTrackingModule as htm*.

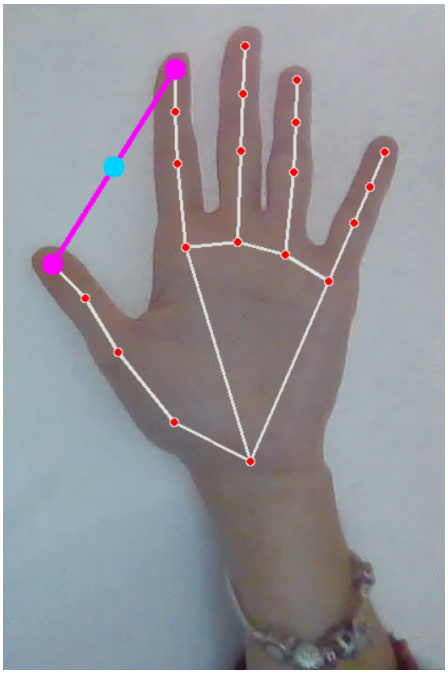
With next code snippet I will findHand and positions of landmarks.

If hand is detected, I will get positionX and positionY of landmarks 4 and 8 as I decided that previously.

For better visualization I will draw circles on those landmarks and connect them with line.

```
while True:
    success, img = cap.read()
    img = detector.findHand(img)
    lmList = detector.findPosition(img)
    if len(lmList) != 0:
        x1, y1 = lmList[4][1], lmList[4][2]
        x2, y2 = lmList[8][1], lmList[8][2]
        cv2.circle(img, (x1, y1), 10, (255, 0, 255), cv2.FILLED)
        cv2.circle(img, (x2, y2), 10, (255, 0, 255), cv2.FILLED)
        cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
        centerLineX, centerLineY = (x1 + x2) // 2, (y1 + y2) // 2
        cv2.circle(img, (centerLineX, centerLineY), 10, (255, 210, 0), cv2.FILLED)
```

After running this part of code:

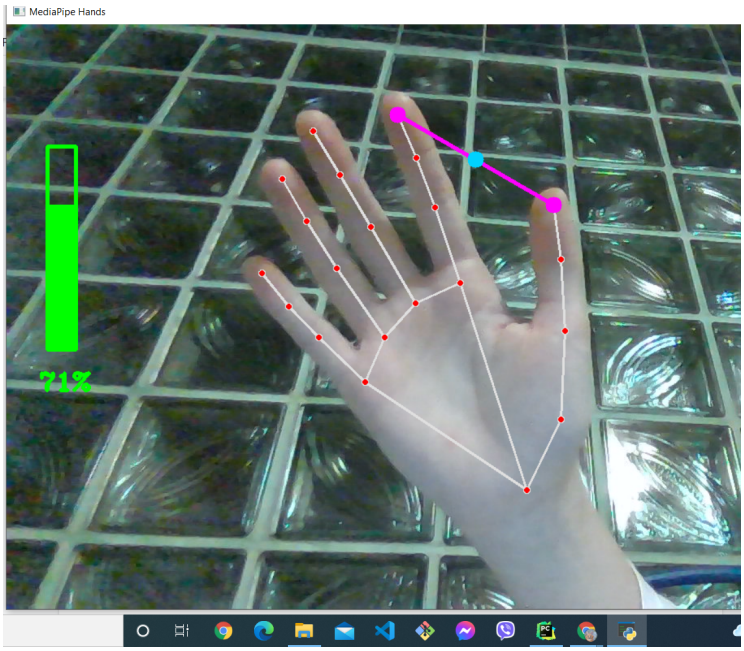


Next step is counting distance and setting volume.

In this step, interpolation should be done. If I print distance that is calculated, I can see that max distance is about 300 and minimum distance is about 30. So, that range should be converted to new one [0, 1]. This one interpolation is for setting sound. There should be also conversion from [30, 300] to [0, 100] for percentage information and from [30, 300] to [400, 150] for bar

visualisation. So, when volume is low, it goes to 400 and when is high, it goes to 150 and that is showed on side-bar.

```
    # finding distance between two landmarks
    # distance bewteen 2 coordinates is d= sqrt((x2-x1)^2 + (y2-y1)^2)
    distance = math.hypot((x2 - x1), (y2 - y1))
    # Convert distance value
    # Hand range was [30, 300]
    # [0, 1] is the range to which we want to convert it
    vol = np.interp(distance, [30, 300], [0, 1])
    volBar = np.interp(distance, [30, 300], [400, 150])
    volPer = np.interp(distance, [30, 300], [0, 100])
    sound1.set_volume(vol)
cv2.rectangle(img, (50, 150), (85, 400), (0,255,0), 3)
cv2.rectangle(img, (50, int(volBar)), (85, 400), (0, 255, 0), cv2.FILLED)
cv2.putText(img, f'{int(volPer)}%', (40,450), cv2.FONT_HERSHEY_COMPLEX,1 ,(0,255,0), 3)
```



MATERIALS

1. https://google.github.io/mediapipe/solutions/hands.html
2. HandTracking tutorial https://www.youtube.com/watch?v=NZde8Xt78Iw

CODE: https://github.com/ababic2/HandTrackingProject

DEMO: https://drive.google.com/drive/folders/1I2O3E5eSIasIlyPGeNwzJ8mbwIrFinBG

SECOND APPROACH

Here, I will just explain my second approach for this task. More precisely, I will explain through steps how this could've been done if I haven't used Mediapipe.

Using:

- python 3.10.
- Tensorflow Object Detection
- opencv

Steps:

- Collect images for deep learning using webcam and OpenCV
- Label images for sign language detection using LabelImg
- Setup Tensorflow Object Detection pipeline configuration
- Use transfer learning to train a deep learning model
- Detect sign language in real time using OpenCV