MCT-243:	Computer Programming-II
SEMSETER PROJECT [CLO-5, CLO-6]	
Instructor:	Muhammad Ahsan Naeem
Deadlines:	Will be announced Soon
Place:	Simulation Lab

Objective:

Students are given a chance to prove their learning of programming language at an advanced level. They shall be able to demonstrate whatever they have learned over the whole semester in form of this project. Projects have been designed in such a way that each encompasses not just a vast area of programming, but also includes advanced features like Computer Vision and Machine Learning. Such projects will help students to develop a sense of formulating algorithms and implementing them using suitable state-of-the-art packages/libraries such as MediaPipe developed by Google.

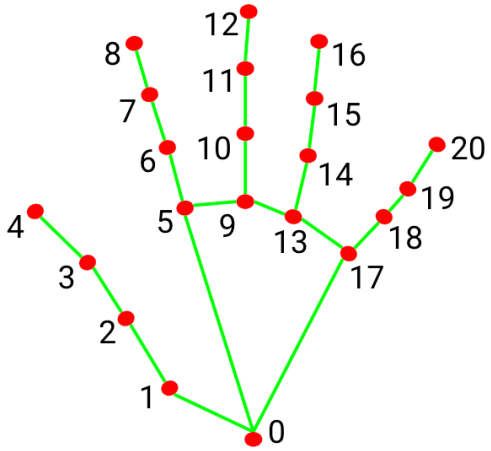
What is MediaPipe?

MediaPipe is a framework developed by Google that provides open-source cross-platform, customizable Machine Learning solutions for the live and streaming media (images and videos). The MediaPipe trained many thousands of images and provides the final trained models to be directly used in different programming languages including Python. The machine learning solutions provided by MediaPipe include Hand Tracking, Human Pose Detection & Tracking, Face Mesh, Face Detection, Holistic Tracking, and a few others. The details can be found at [1].

The solutions students will be using in this project are Hand Tracking and Human Pose Detection & Tracking. However, they are encouraged to use some other solution provided by MediaPipe and can suggest a project other than the project list provided by the instructor.

Hand Tracking:

The Hand Tracking solution of MediaPipe employs machine learning (ML) to infer **21** 3D landmarks of a hand from just a single frame. This solution 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. The details of 21 landmarks are explained here:



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

Figure 1: Landmarks detail of Hand

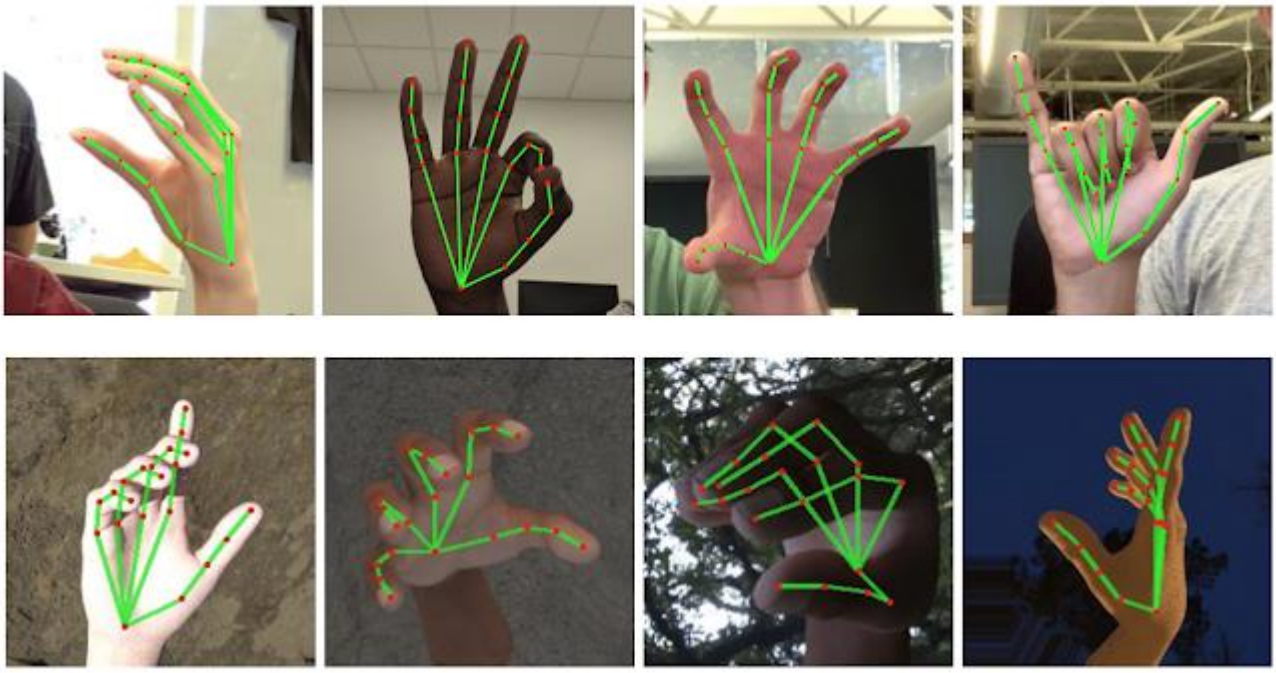
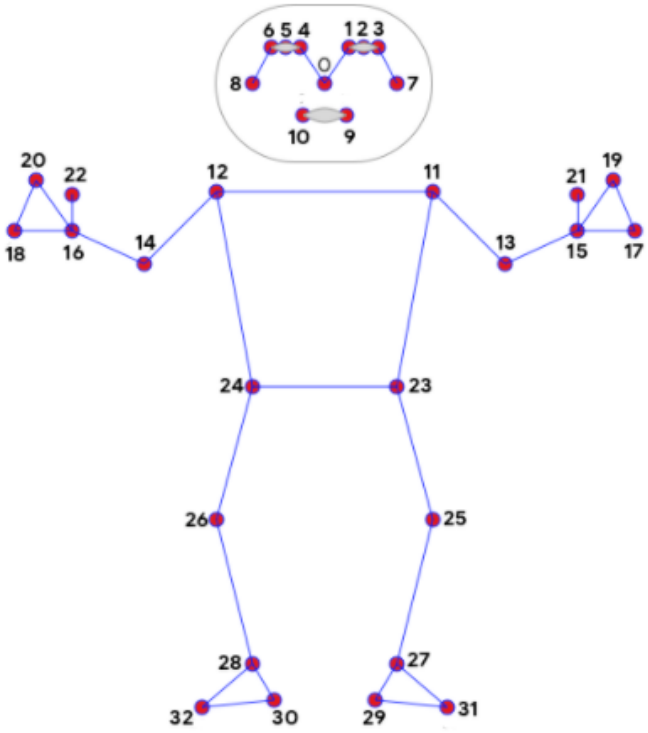


Figure 2: Sample Hand Tracking by Hand Tracking Solution of MediaPipe

Further details on Hand Tracking can be explored at [2].

Human Pose Detection & Tracking:

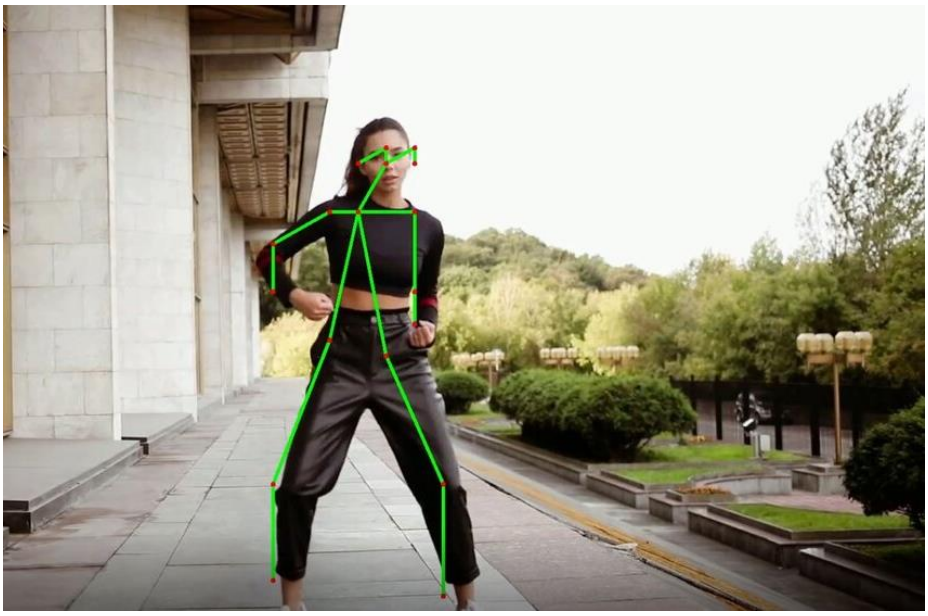
Similar to Hand Tracking, this solution tracks the full-body pose with 32 3D landmarks as shown here:



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 3: Human Pose Landmarks Detail

This solution can be used in pose detection/classification for so many human activity related domains. An example estimation is shown in the following figures:



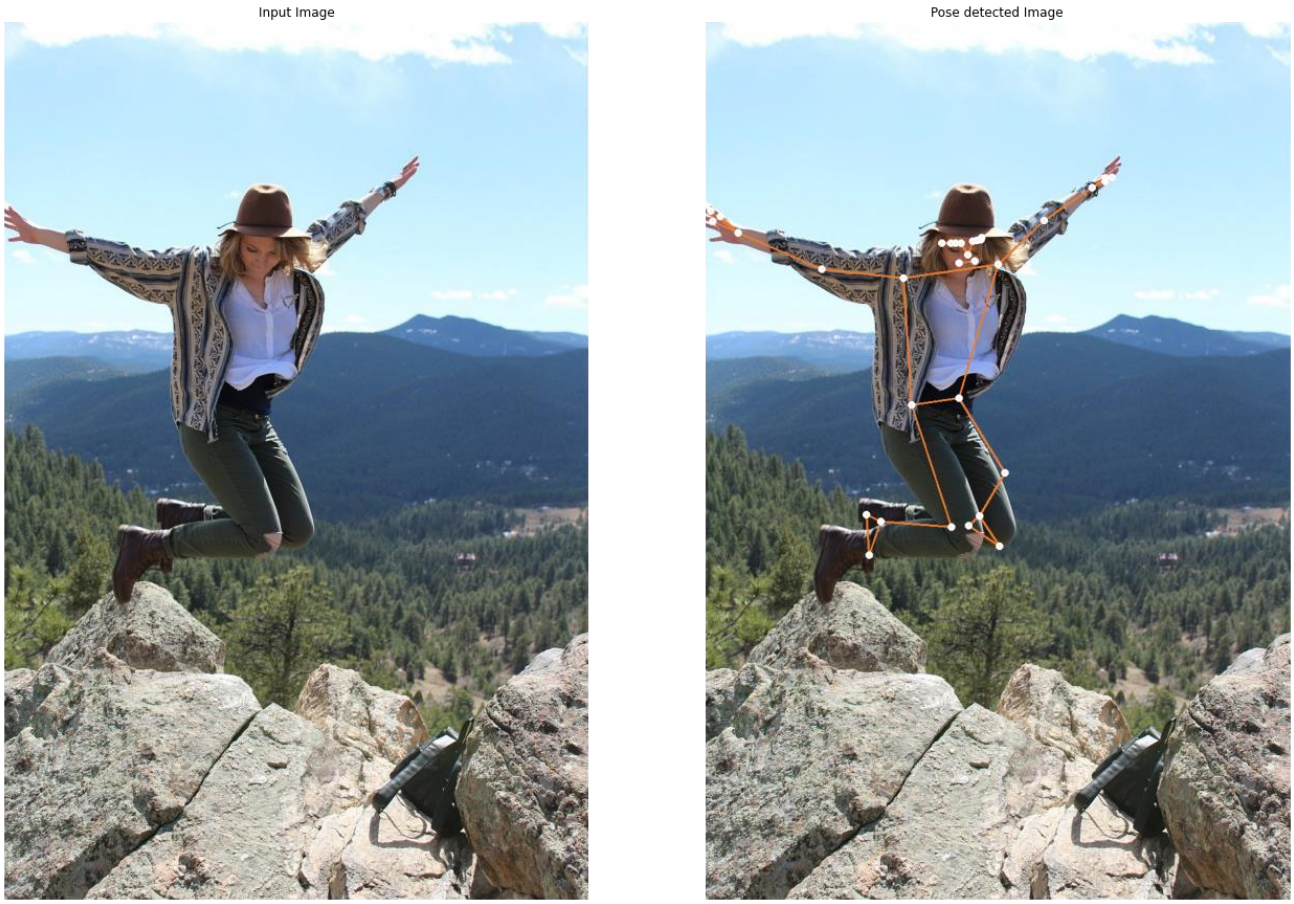


Figure 4: Examples of Human Pose Detection

The Project Detail:

There is a list of projects you can choose from and in general, the core idea is tracking and estimating the human pose or hand to generate meaningful analytics. To give a simple example, it can be the Bowling Action detector for a cricket game bowler. The program will find the elbow angles of the bowler while bowling and will determine if the bowling action is correct or not. There can be many examples from sports and physical workouts. As another example, consider the Push Up exercise and the program will determine if a person is doing the Push-Ups correctly and will also count the number of Push Up.

As mentioned earlier, there will be many projects to choose from, so it's hard to specify the general requirements for each project. But here is a description of the most likely attributes each project should cover although these can slightly vary for different projects.

- *Results for Recorded Videos as well as the Live videos*
  Your project should provide the results for the recorded videos (may be downloaded from YouTube) and should also work on Live Camera.
- *Angle Calculations*
  Except for a few projects, mostly will depend on different joint angles. So, there should be angle calculations and those angles should be displayed as overlayed content on the video.
- *Counting an activity*
  Many of the projects include counting an activity e.g., counting the number of Push Ups, Sit Ups, etc.
- *Determining the Quality of the activity*
  Each activity has some standard benchmarks. Your project will also determine how closed is the activity being performed to the standard benchmark.
- *Time Calculations*
  Most of the projects also involve the time calculation for the activity.

Machine Vision:

The project includes the use of Machine Vision. It will be a very basic use of Machine Vision and in this section, you are provided with the basics and the necessary code. You will be using **OpenCV** which is an open-source machine vision library having support in C++, Python, and Java.

The first step is installing the OpenCV library. The packages are installed using the package manager PIP. To install any library in Python, first, you need to ensure a couple of path settings and then use the PIP manager. A separate document is provided for this purpose.

The documentation of OpenCV can be found at [3]. Here you will be guided through the basic code to access the camera and get the video frames which then will be processed by MediaPipe. Below is the code to read the camera frame and display that (assuming you have installed the OpenCV package).

[Most of the codes has been taken from a YouTube video [4]. The YouTube channel [5] of the video creator is also a very good resource to learn Machine Vision.

```
import cv2
cap=cv2.VideoCapture(0)#Opens Camera for Video Capture
# The input argument 0 in above function is the index of Camera
# In case there are multiple cameras attached with the PC, this index
can be used to switch to other
while True:
```

```
    success,img=cap.read()#Grabs, decodes and returns the next video
frame
    # img is 3D array in numpy of shape (480, 640, 3)
    # 480x640 is the default resolution of Camera
    # You can view the shape as print(img.shape)
    cv2.imshow("Live",img)
    cv2.waitKey(1) #Waits for 1 msec before closing
```

You can change the resolution of the captured video if the camera supports higher resolution. Moreover, you can flip the image vertically (axis 0) or horizontally (axis 1) if needed. Here is the code:

```
import cv2
cap=cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
while True:
    success,img=cap.read()
    img=cv2.flip(img, 1)
    cv2.imshow("Live",img)
    cv2.waitKey(1)
```

Using MediaPipe Hand Tracking Solution

To use the MediaPipe Hand Tracking Solution, you need to follow these steps:

- Import **MediaPipe** package.
- Create MediaPipe Hand solution object as **mediapipe. solutions.hands**
- Apply **Hands()** method on the above-created Hand Solution object to get the **Hands** object.
- MediaPipe works on **RGB** Color format while OpenCV works on **BGR** Color format. You need to convert the frame captured using OpenCV to RGB format using the **cvtColor()** method of OpenCV.
- Use **process()** method on the **Hands** object and pass the RGB frame. The result will be landmarks of one or two hands in the frame.
- The results can be plotted using the **drawing_utils** inside the MediaPipe package.

The code is provided here:

```
import cv2
import mediapipe as mp

cap=cv2.VideoCapture(0)#Opens Camera for Video Capture
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

# mpHands=mp.solutions.hands # <-- This works fine but doesn't
provide the Intellisence Help in VSC
# Hence use the statement below for VSC
mpHands=mp.solutions.mediapipe.python.solutions.hands     #     From
different solutions available in MediaPipe, it will create the object
of Hands Solution
hands=mpHands.Hands() # The hands object from Hands Solution
```

```
mpDraw=mp.solutions.mediapipe.python.solutions.drawing_utils   #This
will be used to draw the landmarks and lines etc
while True:
    success,img=cap.read()
    img=cv2.flip(img, 1) #if needed to flip Horizontally
    imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB) # BGR image converted
to RGB
    results=hands.process(imgRGB)
    # print(results.multi_hand_landmarks)
    if results.multi_hand_landmarks:
        for hand in results.multi_hand_landmarks:
            # mpDraw.draw_landmarks(img,hand) # Will draw just the
hands
            mpDraw.draw_landmarks(img,hand,mpHands.HAND_CONNECTIONS)
#Will draw the connections as well

    cv2.imshow("Live",img)
    cv2.waitKey(1) #Waits for 1msec before closing
```

There are 21 landmarks on each hand. And we can view those from the output of the process method as coded here:

```
results=hands.process(imgRGB)
    if results.multi_hand_landmarks:
        for hand in results.multi_hand_landmarks:
            for i in range(21):
                mark=hand.landmark[i]
                print(i,mark)
            # You can avoid indexing by using enumerate function
            # for id,mark in enumerate(hand.landmark):
            #     print(id,mark)
```

If you observe the output of the above code, you will see the x, y, z values of each landmark point. On example value of landmark index 20 is shown here:

```
20 x: 0.34332752227783203
y: 0.3595533072948456
z: -0.09572944790124893
```

These values are the ratio of the image (frame) with the position of the landmark. The **z** value shows the depth value which is the least accurate, and we will not use that in most of the cases. The other two values i.e., **x** and **y** must be converted to the pixel values (1280x720 or other resolution). We can convert those to pixel values by multiplying those by the width and height of the image as shown here:

```
for i in range(21):
    mark=hand.landmark[i]
    height,width,channels=img.shape
    px,py=int(mark.x*width),int(mark.y*height)
    print(i,px,py)
```

One example output for Landmark 20 is shown here which indicates the pixel values:

```
20 341 243
```

The index in above **for** loop is the landmark index and we can specify any landmark using the index **i** of the **for** loop. For example, if you see Figure 1 you can find the landmark number for the tip of the index finger is 8. Let's just display a circle on the tip of the index finger and also display its pixel position on the frame. We will use **circle()** and **putText()** methods of the OpenCV. Complete code is provided here, you should run this and move your hand to see the pixel position of the index finger. You should try to move the tip to the corners of the frames to see the pixel value.

```
import cv2
import mediapipe as mp

cap=cv2.VideoCapture(0)#Opens Camera for Video Capture
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
mpHands=mp.solutions.mediapipe.python.solutions.hands  # From different solutions available in MediaPipe, it will create the object of Hands Solution
hands=mpHands.Hands() # The hands object from Hands Solution
mpDraw=mp.solutions.mediapipe.python.solutions.drawing_utils #This will be used to draw the landmarks and lines etc
while True:
    success,img=cap.read()
    img=cv2.flip(img, 1) #if needed to flip Horizontally
    imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB) # BGR image converted to RGB
    results=hands.process(imgRGB)
    if results.multi_hand_landmarks:
        for hand in results.multi_hand_landmarks:
            for i in range(21):
                mark=hand.landmark[i]
                height,width,channels=img.shape
                px,py=int(mark.x*width),int(mark.y*height)
                if i==8:
                    cv2.circle(img,(px,py),20,(0,255,255),cv2.FILLED)
                     cv2.putText(img,f'{px},{py}',(10,50),cv2.FONT_HERSHEY_COMPLEX,2,(0,255,255),2)
                mpDraw.draw_landmarks(img,hand,mpHands.HAND_CONNECTIONS) #Will draw the connections as well
    cv2.imshow("Live",img)
    cv2.waitKey(1) #Waits for 1msec before closing
```

If you just need the landmark of the Tip of the Index finger, you can just pick that at index 8 rather than using the **for** loop as coded here:

```
import cv2
import mediapipe as mp

cap=cv2.VideoCapture(0)#Opens Camera for Video Capture
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
```

```
mpHands=mp.solutions.mediapipe.python.solutions.hands # From different solutions available in MediaPipe, it will create the object of Hands Solution
hands=mpHands.Hands() # The hands object from Hands Solution
mpDraw=mp.solutions.mediapipe.python.solutions.drawing_utils #This will be used to draw the landmarks and lines etc
while True:
    success,img=cap.read()
    img=cv2.flip(img, 1) #if needed to flip Horizontally
    imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB) # BGR image converted to RGB
    results=hands.process(imgRGB)
    height,width,channels=img.shape
    if results.multi_hand_landmarks:
        for hand in results.multi_hand_landmarks:
            mark=hand.landmark[8]
            px,py=int(mark.x*width),int(mark.y*height)
            cv2.circle(img,(px,py),20,(0,255,255),cv2.FILLED)
             cv2.putText(img,f'{px},{py}',(10,50),cv2.FONT_HERSHEY_COMPLEX,2,(0,255,255),2)
         mpDraw.draw_landmarks(img,hand,mpHands.HAND_CONNECTIONS) #Will draw the connections as well
    cv2.imshow("Live",img)
    cv2.waitKey(1) #Waits for 1msec before closing
```

FPS Calculations:

A video is a series of frames. The frames played per second is known as **FPS**. For seamless good quality video streaming, it is recommended that minimum fps should be 24. In our program, the MediaPipe processing consumes time, and we can display the current FPS on the frame. The required calculations are done in this code using the **time** module and instead of the pixel points, FPS value is shown.

```
import cv2
import mediapipe as mp
import time

cap=cv2.VideoCapture(0)#Opens Camera for Video Capture
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
mpHands=mp.solutions.mediapipe.python.solutions.hands # From different solutions available in MediaPipe, it will create the object of Hands Solution
hands=mpHands.Hands() # The hands object from Hands Solution
mpDraw=mp.solutions.mediapipe.python.solutions.drawing_utils #This will be used to draw the landmarks and lines etc
cTime=0 # Current Time Value
pTime=0 # Previous Time Value

while True:
    success,img=cap.read()
    img=cv2.flip(img, 1) #if needed to flip Horizontally
```

```
    imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB) # BGR image converted to
RGB
    results=hands.process(imgRGB)
    height,width,channels=img.shape
    if results.multi_hand_landmarks:
        for hand in results.multi_hand_landmarks:
            mark=hand.landmark[8]
            px,py=int(mark.x*width),int(mark.y*height)
            cv2.circle(img,(px,py),20,(0,255,255),cv2.FILLED)
             cv2.putText(img,f'{px},{py}',(10,50),cv2.FONT_HERSHEY_COMPL
EX,2,(0,255,255),2)
          mpDraw.draw_landmarks(img,hand,mpHands.HAND_CONNECTIONS)  #Will
draw the connections as well
    # FPS Calculations
    cTime=time.time()
    fps=1/(cTime-pTime)
    pTime=cTime
     cv2.putText(img,f'FPS:{int(fps)}',(1000,50),cv2.FONT_HERSHEY_COMPLE
X,2,(0,255,255),2)

    cv2.imshow("Live",img)
    cv2.waitKey(1) #Waits for 1msec before closing
```

Object-Oriented based Solution:

It will be better to use the concept of Object-Oriented Programming and convert the basic tracking code to a class and then use that class for some meaningful task. Here is the conversion of the Hands Solution to OOP in a file named as **HandTrackingModule.py**.

```
"""
Hand Tracking Module
By: Computer Vision Zone
Website: https://www.computervision.zone/
"""

import cv2
import mediapipe as mp

class HandDetector:
    """
    Finds Hands using the mediapipe library. Exports the landmarks
    in pixel format. Adds extra functionalities like finding how
    many fingers are up or the distance between two fingers. Also
    provides bounding box info of the hand found.
    """

       def __init__(self, mode=False, maxHands=2, detectionCon=0.5,
minTrackCon=0.5):
        """
         :param mode: In static mode, detection is done on each image:
slower
```

```
        :param maxHands: Maximum number of hands to detect
        :param detectionCon: Minimum Detection Confidence Threshold
        :param minTrackCon: Minimum Tracking Confidence Threshold
        """
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.minTrackCon = minTrackCon

        self.mpHands = mp.solutions.hands
           self.hands  =  self.mpHands.Hands(static_image_mode=self.mode,
max_num_hands=self.maxHands,min_detection_confidence=self.detectionCon,
min_tracking_confidence=self.minTrackCon)
        self.mpDraw = mp.solutions.drawing_utils
        self.tipIds = [4, 8, 12, 16, 20]
        self.fingers = []
        self.lmList = []

    def findHands(self, img, draw=True, flipType=True):
        """
        Finds hands in a BGR image.
        :param img: Image to find the hands in.
        :param draw: Flag to draw the output on the image.
        :return: Image with or without drawings
        """
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        allHands = []
        h, w, c = img.shape
        if self.results.multi_hand_landmarks:
            for handType, handLms in zip(self.results.multi_handedness,
self.results.multi_hand_landmarks):
                myHand = {}
                ## lmList
                mylmList = []
                xList = []
                yList = []
                for id, lm in enumerate(handLms.landmark):
                    px, py, pz = int(lm.x * w), int(lm.y * h), int(lm.z
* w)
                    mylmList.append([px, py, pz])
                    xList.append(px)
                    yList.append(py)

                ## bounding Box
                xmin, xmax = min(xList), max(xList)
                ymin, ymax = min(yList), max(yList)
                boxW, boxH = xmax - xmin, ymax - ymin
                bbox = xmin, ymin, boxW, boxH
                cx, cy = bbox[0] + (bbox[2] // 2), \
                         bbox[1] + (bbox[3] // 2)
```

```
                myHand["lmList"] = mylmList
                myHand["bbox"] = bbox
                myHand["center"] = (cx, cy)

                if flipType:
                    if handType.classification[0].label == "Right":
                        myHand["type"] = "Left"
                    else:
                        myHand["type"] = "Right"
                else:
                    myHand["type"] = handType.classification[0].label
                allHands.append(myHand)

                ## draw
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
                                               self.mpHands.HAND_CONNECTIONS)
                    cv2.rectangle(img, (bbox[0] - 20, bbox[1] - 20),
                                  (bbox[0] + bbox[2] + 20, bbox[1] +
bbox[3] + 20),
                                  (255, 0, 255), 2)
                    cv2.putText(img, myHand["type"], (bbox[0] - 30,
bbox[1] - 30), cv2.FONT_HERSHEY_PLAIN,
                                2, (255, 0, 255), 2)
        if draw:
            return allHands, img
        else:
            return allHands
```

A sample program that will count the number of Index Finger Ups (Index Finger Up Counts) is here:

```
from HandTrackingModule import HandDetector
import cv2
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
detector = HandDetector(detectionCon=0.8, maxHands=2)
upCount=0 # Count of Index Fingure UPs
checkAgain=True
while True:
    success, img = cap.read()
    img=cv2.flip(img, 1)
    hands, img = detector.findHands(img,flipType=False)  # with draw
    # hands = detector.findHands(img, draw=False)  # without draw

    if hands:
        hand1 = hands[0]
        lmList1 = hand1["lmList"]  # List of 21 Landmark points
        tipy=lmList1[8][1] #8 for tip of index finger, 1 for y coordinate
        pipy=lmList1[6][1] #6 for pip of index finger, 1 for y coordinate
        if tipy<pipy and checkAgain:
```

```
            upCount+=1
            checkAgain=False
        if pipy<tipy:
            checkAgain=True
    # Display
     cv2.putText(img,f'{upCount}',(10,50),cv2.FONT_HERSHEY_COMPLEX,2,(0,
255,255),2)
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

Human Pose Solution:

Another solution by MediaPipe is Pose Solution. It detects and track 32 3D landmarks of full body as shown in Figure 3. Following the similar approach, we can create a class for Pose estimation with file name **PoseModule.py**:

```
"""
Pose Module
By: Computer Vision Zone
Website: https://www.computervision.zone/
"""
import cv2
import mediapipe as mp

class PoseDetector:
    """
    Estimates Pose points of a human body using the mediapipe library.
    """

    def __init__(self, mode=False, smooth=True,
                 detectionCon=0.5, trackCon=0.5):
        """
         :param mode: In static mode, detection is done on each image:
slower
        :param upBody: Upper boy only flag
        :param smooth: Smoothness Flag
        :param detectionCon: Minimum Detection Confidence Threshold
        :param trackCon: Minimum Tracking Confidence Threshold
        """

        self.mode = mode
        self.smooth = smooth
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpDraw = mp.solutions.drawing_utils
        self.mpPose = mp.solutions.pose
        self.pose = self.mpPose.Pose(static_image_mode=self.mode,
                                     smooth_landmarks=self.smooth,
```

```
                                        min_detection_confidence=self.dete
ctionCon,
                                    min_tracking_confidence=self.trackCon)

    def findPose(self, img, draw=True):
        """
        Find the pose landmarks in an Image of BGR color space.
        :param img: Image to find the pose in.
        :param draw: Flag to draw the output on the image.
        :return: Image with or without drawings
        """
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.pose.process(imgRGB)
        if self.results.pose_landmarks:
            if draw:
                                            self.mpDraw.draw_landmarks(img,
self.results.pose_landmarks,
                                              self.mpPose.POSE_CONNECTIONS)
        return img

    def findPosition(self, img, draw=True, bboxWithHands=False):
        self.lmList = []
        self.bboxInfo = {}
        if self.results.pose_landmarks:
           for id, lm in enumerate(self.results.pose_landmarks.landmark):
                h, w, c = img.shape
                cx, cy, cz = int(lm.x * w), int(lm.y * h), int(lm.z * w)
                self.lmList.append([id, cx, cy, cz])

            # Bounding Box
            ad = abs(self.lmList[12][1] - self.lmList[11][1]) // 2
            if bboxWithHands:
                x1 = self.lmList[16][1] - ad
                x2 = self.lmList[15][1] + ad
            else:
                x1 = self.lmList[12][1] - ad
                x2 = self.lmList[11][1] + ad

            y2 = self.lmList[29][2] + ad
            y1 = self.lmList[1][2] - ad
            bbox = (x1, y1, x2 - x1, y2 - y1)
            cx, cy = bbox[0] + (bbox[2] // 2), \
                     bbox[1] + bbox[3] // 2

            self.bboxInfo = {"bbox": bbox, "center": (cx, cy)}

            if draw:
                cv2.rectangle(img, bbox, (255, 0, 255), 3)
                cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)

        return self.lmList, self.bboxInfo
```

A sample program to estimate and track the pose is here:

```
from PoseModule import PoseDetector
import cv2

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
detector = PoseDetector()
while True:
    success, img = cap.read()
    img=cv2.flip(img, 1)
    img = detector.findPose(img)
    detector.findPosition(img, bboxWithHands=False)
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

References:

[1]. https://mediapipe.dev/index.html
[2]. https://google.github.io/mediapipe/solutions/hands
[3]. https://docs.opencv.org/4.x/index.html
[4]. https://www.youtube.com/watch?v=01sAkU_NvOY
[5]. https://www.youtube.com/c/MurtazasWorkshopRoboticsandAI