Pose Detection Using Mediapipe Solutions (DabMove Detection) Using Python

Introduction

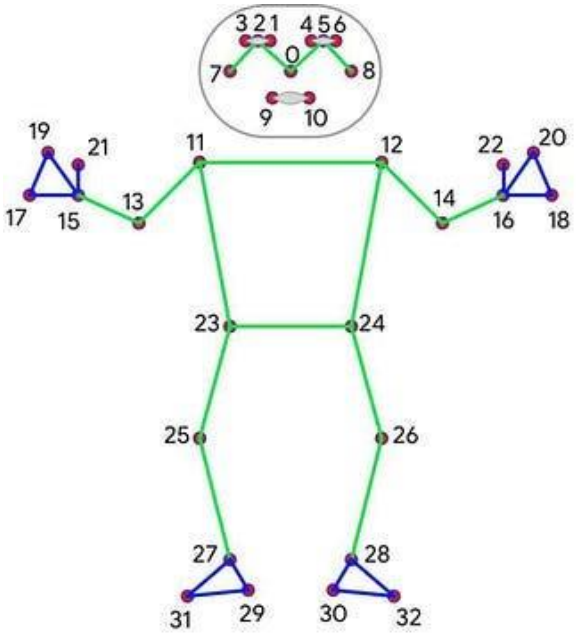
This DabMove Pose Detector is a computer vision program that uses MediaPipe and OpenCV to detect and track the human pose in real time. The application detects and tracks the location of various body parts, including the shoulders, elbows, wrists, and hips, and then analyzes their movements to detect the “dab” dance move.

The application uses MediaPipe’s Pose Detection model to detect the human pose, which provides key landmarks for each body part. These landmarks are then tracked using OpenCV’s optical flow algorithm, which estimates the motion of the landmarks between consecutive frames of the video.

Once the landmarks have been tracked, the application analyzes their movements to detect the “dab” dance move. The “dab” move involves raising one arm to shoulder height and extending the other arm to the side, with the head bowed down towards the extended arm.

The application uses a simple algorithm to detect the “dab” move based on the position of the shoulders, elbows, and wrists. If the application detects that the arm on one side of the body is extended to the side while the other arm is raised to shoulder height, and the head is bowed down towards the extended arm, then it concludes that the user is performing the “dab” move.

Overall, the DabMove Pose Detector is a fun and engaging application that demonstrates the power and flexibility of MediaPipe and OpenCV for real-time computer vision applications.



0. nose
1. right eye inner
2. right eye
3. right eye outer
4. left eye inner
5. left eye
6. left eye outer
7. right ear
8. left ear
9. mouth right
10. mouth left
11. right shoulder
12. left shoulder
13. right elbow
14. left elbow
15. right wrist
16. left wrist
17. right pinky knuckle #1
18. left pinky knuckle #1
19. right index knuclke #1
20. left index knuckle #1
21. right thumb knuckle #2
22. left thumb knuckle #2
23. right hip
24. left hip
25. right knee
26. left knee
27. right ankle
28. left ankle
29. right heel
30. left heel
31. right foot index
32. left foot index

Here is a link to the [MediaPipe Pose documentation]:

MediaPipe

MediaPipe is an open-source framework developed by Google that provides pre-built components for building pipelines for processing perceptual data such as images, videos, and audio. The framework is built on TensorFlow and provides a high-level API that allows developers to integrate machine learning models into their projects.

MediaPipe's pre-built components include object detection, face detection, hand tracking, pose estimation, and more. These components can be customized and combined to build pipelines for processing perceptual data.

MediaPipe is designed to run on a wide range of hardware platforms, including CPUs, GPUs, and specialized hardware like Google's Edge TPU. This makes it suitable for building real-time applications that can run on different devices.

Overall, MediaPipe is a flexible and powerful framework that simplifies the development of complex pipelines for processing perceptual data.

we can install it using this command in cmd or VScode

```
pip install mediapipe
```

Sources

1. github.com/google/mediapipe

OpenCV

OpenCV (Open Source Computer Vision) is an open-source computer vision and machine learning library. Originally developed by Intel, it is now maintained by the OpenCV team. OpenCV provides a range of pre-built functions and algorithms for image and video processing tasks such as image filtering, feature detection, object recognition, and more.

One of the key features of OpenCV is its cross-platform compatibility, which allows it to run on different operating systems like Windows, Linux, and macOS. OpenCV also supports a variety of programming languages like C++, Python, and Java, making it accessible to a wide range of developers.

Overall, OpenCV is a powerful and flexible library that simplifies the development of computer vision and machine learning applications. Its cross-platform compatibility and support for multiple programming languages make it an ideal choice for developers who want to build applications that can run on different platforms.

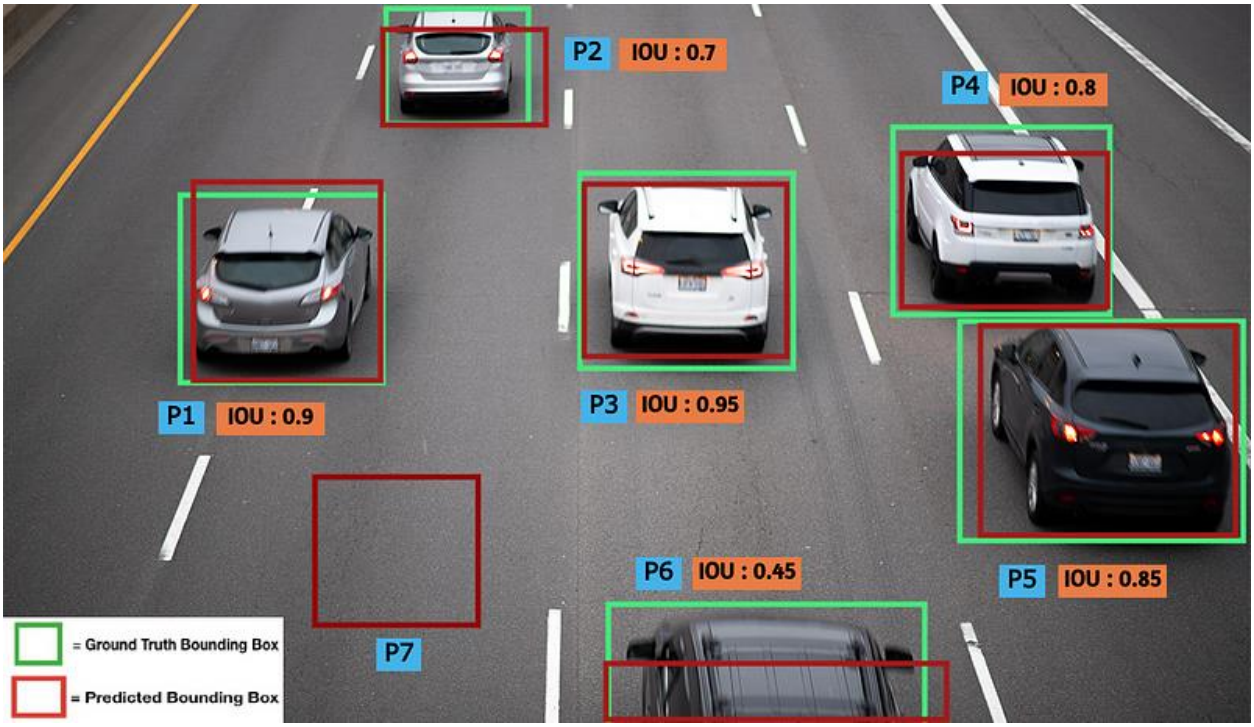
we can install it using this command in cmd or VScode

```
pip install opencv-python
```

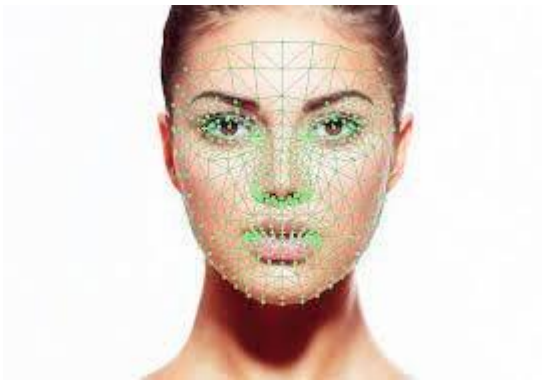
MediaPipe Solutions

MediaPipe provides a range of pre-built solutions that developers can use to build computer vision and machine learning applications quickly and easily. Here are some of the solutions offered by MediaPipe:

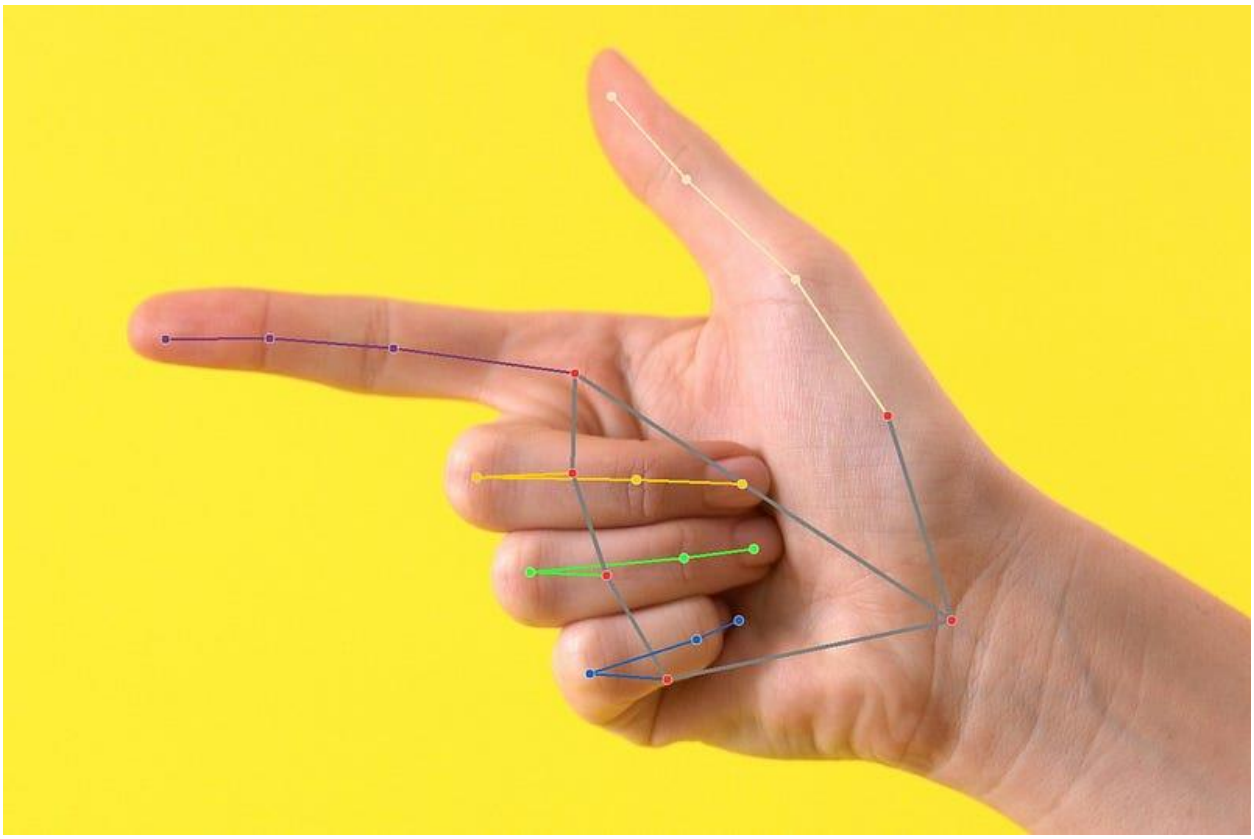
1. **Object Detection**: MediaPipe's object detection solution allows developers to identify and track objects in real-time. This solution uses a deep learning model to detect objects and track their movements over time.



2. **Face Detection**: MediaPipe's face detection solution allows developers to detect faces in images and videos. This solution uses a deep learning model to detect faces and can identify facial features like eyes, nose, and mouth.



3. **Hand Tracking**: MediaPipe's hand tracking solution allows developers to track hands in real-time. This solution uses a deep learning model to detect and track hand movements and can identify individual fingers.



4. **Pose Estimation**: MediaPipe's pose estimation solution allows developers to estimate human body poses in real-time. This solution uses a deep learning model to detect and track key points on the human body, like the shoulders, elbows, and knees.

5. **Objectron**: Objectron is a 3D object detection solution offered by MediaPipe. It allows developers to detect and track objects in 3D space using a single RGB camera. This solution is useful for AR and VR applications.

These are just a few examples of the solutions offered by MediaPipe. Each solution comes with pre-built models and pipelines that developers can use to build their applications quickly and easily. MediaPipe's solutions are designed to be flexible and customizable, allowing developers to tailor them to their specific needs.

DabMove Detection

I have recently created a pose detection program using OpenCV and MediaPipe. The program can detect a person's pose and identify the positions of their shoulders, elbows, wrists, hips, knees, and ankles. It uses the MediaPipe Holistic solution, which includes both body and hand pose estimation models. The program can display the detected keypoints on the input video stream and also draw a line connecting each pair of adjacent key points. Additionally, I have included a "Dab" pose in the program, where it will display a "Dab!" message on the screen whenever a person performs the dab pose. The program has been built using Python and provides a simple yet powerful way to detect a person's pose in real-time video.

example video dab.mp4

Methodology

1. Import libraries

```
import cv2
import math
import tkinter as tk
from tkinter import *
import mediapipe as mp
from PIL import Image,ImageTk
import threading
from tkinter import filedialog
```

1. Make a Class to make the use of program simple

```
class DabMOve_DetectionGUI:
    def
__init__(self,master,video_path,min_detection_confidence=0.5,min_tracking_con
```

```
fidence=0.5 ):
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_holistic = mp.solutions.holistic

        self.master = master
        self.Dabmove = 0
        self.count1 = False
        self.count2 = False
        self.count3 = False
        self.count4 = False
        self.c1,self.c2,self.c3,self.c4=0,0,0,0

        self.cap = cv2.VideoCapture(video_path)
        self.holistic =
self.mp_holistic.Holistic(min_detection_confidence=min_detection_confidence,
min_tracking_confidence=min_tracking_confidence)

        # title on canvas
        self.master = master
        self.master.title("DabMove Detector")
        #  creation of canvas with dimensions
        self.canvas = tk.Canvas(self.master, width=640, height=480 ,
bg="black")
        self.canvas.pack()
        #  title on left upper corner
        self.button_frame = tk.Frame(self.master)
        self.button_frame.pack(side=tk.LEFT)
        # self.button_frame.configure(bg="blue")
        #  start button
        self.start_button = tk.Button(self.button_frame, text="Start",
command=self.start_video,padx=1, width=15, height=3 , bg="green",
font=("Arial Black", 8,"bold"))
        self.start_button.pack(side=tk.LEFT)
        # stop button
        self.stop_button = tk.Button(self.button_frame, text="Stop",
command=self.stop_video,padx=1, width=15, height=3 , bg="red", font=("Arial
Black", 8,"bold"))
        self.stop_button.pack(side=tk.LEFT)
        #  upload button
        self.upload_button = tk.Button(self.button_frame, text="Upload Video
or image", command=self.upload_video,padx=3, width=16, height=3, bg="yellow"
, font=("Arial Black", 8,"bold"))
        self.upload_button.pack(side=tk.LEFT)

        #  Add the following line to create a new label to show the leg lift
count
        self.DabMove_count_label = tk.Label(self.master, text=f"DabMove
Counts: 0",fg="red", font=("Arial Black", 8,"bold"))
        self.DabMove_count_label.pack(side=tk.LEFT, anchor=tk.CENTER)
        self.angle_correction = tk.Label(self.master, text=f"Incorrect
DabMove",fg="red", font=("Arial Black", 8,"bold"))
        self.angle_correction.place(x=130, y=25, anchor='ne')
        self.DabMove_image = None
        self.video_running = False
        self.video_detection()
```

```
    # module for upload video from directory
    def upload_video(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            self.cap = cv2.VideoCapture(file_path)
    # module for start video
    def start_video(self):
        self.video_running = True
        # Start a new thread to read and display video frames continuously
        threading.Thread(target=self.video_detection).start()
    # module for stop or pause video
    def stop_video(self):
        self.video_running = False

    # angle update




    # function for pose detection
********************************************************************************
*****************************

    def detect_pose(self,results):



        # Get the coordinates of the left hip and right hip
        left_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_SHOULDER]
        right_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_SHOULDER]


        # angle1 between left parts points 11,13,15
        # left_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_SHOULDER]
        left_elbow =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_ELBOW]
        left_wrist =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_WRIST]


        if results.pose_landmarks is not None:
            angle1 = abs(angle_between_lines(left_shoulder.x,
left_shoulder.y, left_elbow.x, left_elbow.y, left_wrist.x, left_wrist.y))
        else:
            angle1=0
        print("Left Arm(Shoulder, Wrist) :",angle1)

        # angle2 between left parts points 23,11,13
        left_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_HIP]

        if results.pose_landmarks is not None:
            angle2 = abs(angle_between_lines(left_hip.x,
left_hip.y,left_shoulder.x, left_shoulder.y, left_elbow.x, left_elbow.y))
```

```
        else:
            angle2=0
        print("Left Shoulder-Hip:",angle2)

        # angle3 between left parts points 24,12,14
        right_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_HIP]
        right_elbow =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_ELBOW]

        if results.pose_landmarks is not None:
            angle3 = abs(angle_between_lines(right_hip.x, right_hip.y,
right_shoulder.x, right_shoulder.y, right_elbow.x, right_elbow.y))
        else:
            angle3=0
        print("Right Shoulder-Hip:",angle3)

        # angle4 between left parts points 24,12,14
        right_wrist =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_WRIST]

        if results.pose_landmarks is not None:
            angle4 = abs(angle_between_lines(right_shoulder.x,
right_shoulder.y, right_elbow.x, right_elbow.y, right_wrist.x,
right_wrist.y))
        else:
            angle4=0
        print("Right Shoulder-Wrist:",angle4)
        if (angle1 > 0 and angle1 <= 45):
            self.count1 = True
        else:
            self.count1 = False
        print("Count 1 :",self.count1)
        if (60< angle2 < 155):
            self.count2 = True
        else:
            self.count2 = False
        print("Count 2 :",self.count2)
        if (angle3 > 75 and angle3<145):
            self.count3 = True
        else:
            self.count3 = False
        print("Count 3 :",self.count3)
        if (angle4 > 0 and angle4  < 30):
            self.count4 = True
        else:
            self.count4 = False
        print("Count 4 :",self.count4)

        if ((self.c1==False or self.c2==False  or self.c4==False or
self.c3==False) ):    #
            if(self.count1==True and self.count2==True and self.count3==True
and self.count4==True):
                self.Dabmove = self.Dabmove + 1
                self.DabMove_count_label.config(text="DabMove Counts:
{}".format(self.Dabmove),fg="blue")
```

```
        if(self.count1==True and self.count2==True and self.count3==True and
self.count4==True):

            self.angle_correction.configure(text="DabMove is
correcct",fg="green" )
        else:

            self.angle_correction.configure(text="Incorrecct
DabMove",fg="red" )
        print("DAB detected : ",self.Dabmove)
        self.c1=self.count1
        self.c2=self.count2
        self.c3=self.count3
        self.c4=self.count4


    def video_detection(self):
        ret, frame = self.cap.read()

        if ret:
            # Convert the image to RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Make a detection
            results = self.holistic.process(image)

            # Draw the detection points on the image
            annotated_image = image.copy()
            self.mp_drawing.draw_landmarks(annotated_image,
results.pose_landmarks, self.mp_holistic.POSE_CONNECTIONS)

            self.detect_pose(results)
            Bally_Button = self.Bally_Button(results)

            # Draw a circle at the Bally_Button
            cv2.circle(annotated_image, (int(Bally_Button[0] *
annotated_image.shape[1]), int(Bally_Button[1] * annotated_image.shape[0])),
5, (255, 0, 0), -1)

            # # Show the annotated image
            # cv2.imshow('MediaPipe Holistic', annotated_image)
            # Resize the image to match the size of the canvas
            resized_image = cv2.resize(annotated_image, (640, 480))
            self.leg_lift_image =
ImageTk.PhotoImage(Image.fromarray(resized_image))
            self.canvas.create_image(0, 0, anchor=tk.NW,
image=self.leg_lift_image)

            # Exit if the user presses the 'q' key
        if self.video_running:
            self.master.after(50, self.video_detection)

        # Release the webcam and close the window
    def __del__(self):
            self.cap.release()
            cv2.destroyAllWindows()
```

```
    def Bally_Button(self,results):
        left_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_HIP]
        right_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_HIP]
        midpoint = ((left_hip.x + right_hip.x) / 2, (left_hip.y +
right_hip.y) / 2)
        distance_from_hip= (abs(midpoint[0]-left_hip.x),abs(midpoint[1] -
left_hip.y))
        bally_button=(midpoint[0]  , (midpoint[1] - distance_from_hip[0])-
distance_from_hip[0]/4)
        return bally_button
```

Methods in Class

Pose Detection

i have make a methode in program for pose detection for DabMove

```
def detect_pose(self,results):



        # Get the coordinates of the left hip and right hip
        left_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_SHOULDER]
        right_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_SHOULDER]


        # angle1 between left parts points 11,13,15
        # left_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_SHOULDER]
        left_elbow =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_ELBOW]
        left_wrist =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_WRIST]


        if results.pose_landmarks is not None:
            angle1 = abs(angle_between_lines(left_shoulder.x,
left_shoulder.y, left_elbow.x, left_elbow.y, left_wrist.x, left_wrist.y))
        else:
            angle1=0
        print("Left Arm(Shoulder, Wrist) :",angle1)

        # angle2 between left parts points 23,11,13
        left_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_HIP]

        if results.pose_landmarks is not None:
            angle2 = abs(angle_between_lines(left_hip.x,
left_hip.y,left_shoulder.x, left_shoulder.y, left_elbow.x, left_elbow.y))
        else:
            angle2=0
```

```
        print("Left Shoulder-Hip:",angle2)

        # angle3 between left parts points 24,12,14
        right_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_HIP]
        right_elbow =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_ELBOW]

        if results.pose_landmarks is not None:
            angle3 = abs(angle_between_lines(right_hip.x, right_hip.y,
right_shoulder.x, right_shoulder.y, right_elbow.x, right_elbow.y))
        else:
            angle3=0
        print("Right Shoulder-Hip:",angle3)

        # angle4 between left parts points 24,12,14
        right_wrist =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_WRIST]

        if results.pose_landmarks is not None:
            angle4 = abs(angle_between_lines(right_shoulder.x,
right_shoulder.y, right_elbow.x, right_elbow.y, right_wrist.x,
right_wrist.y))
        else:
            angle4=0
        print("Right Shoulder-Wrist:",angle4)
        if (angle1 > 0 and angle1 <= 45):
            self.count1 = True
        else:
            self.count1 = False
        print("Count 1 :",self.count1)
        if (60< angle2 < 155):
            self.count2 = True
        else:
            self.count2 = False
        print("Count 2 :",self.count2)
        if (angle3 > 75 and angle3<145):
            self.count3 = True
        else:
            self.count3 = False
        print("Count 3 :",self.count3)
        if (angle4 > 0 and angle4  < 30):
            self.count4 = True
        else:
            self.count4 = False
        print("Count 4 :",self.count4)

        if ((self.c1==False or self.c2==False  or self.c4==False or
self.c3==False) ):    #
            if(self.count1==True and self.count2==True and self.count3==True
and self.count4==True):
                self.Dabmove = self.Dabmove + 1
                self.DabMove_count_label.config(text="DabMove Counts:
{}".format(self.Dabmove),fg="blue")

        if(self.count1==True and self.count2==True and self.count3==True and
self.count4==True):
```

```
            self.angle_correction.configure(text="DabMove is
correcct",fg="green" )
        else:

            self.angle_correction.configure(text="Incorrecct
DabMove",fg="red" )
        print("DAB detected : ",self.Dabmove)
        self.c1=self.count1
        self.c2=self.count2
        self.c3=self.count3
        self.c4=self.count4
```

Methode for running video

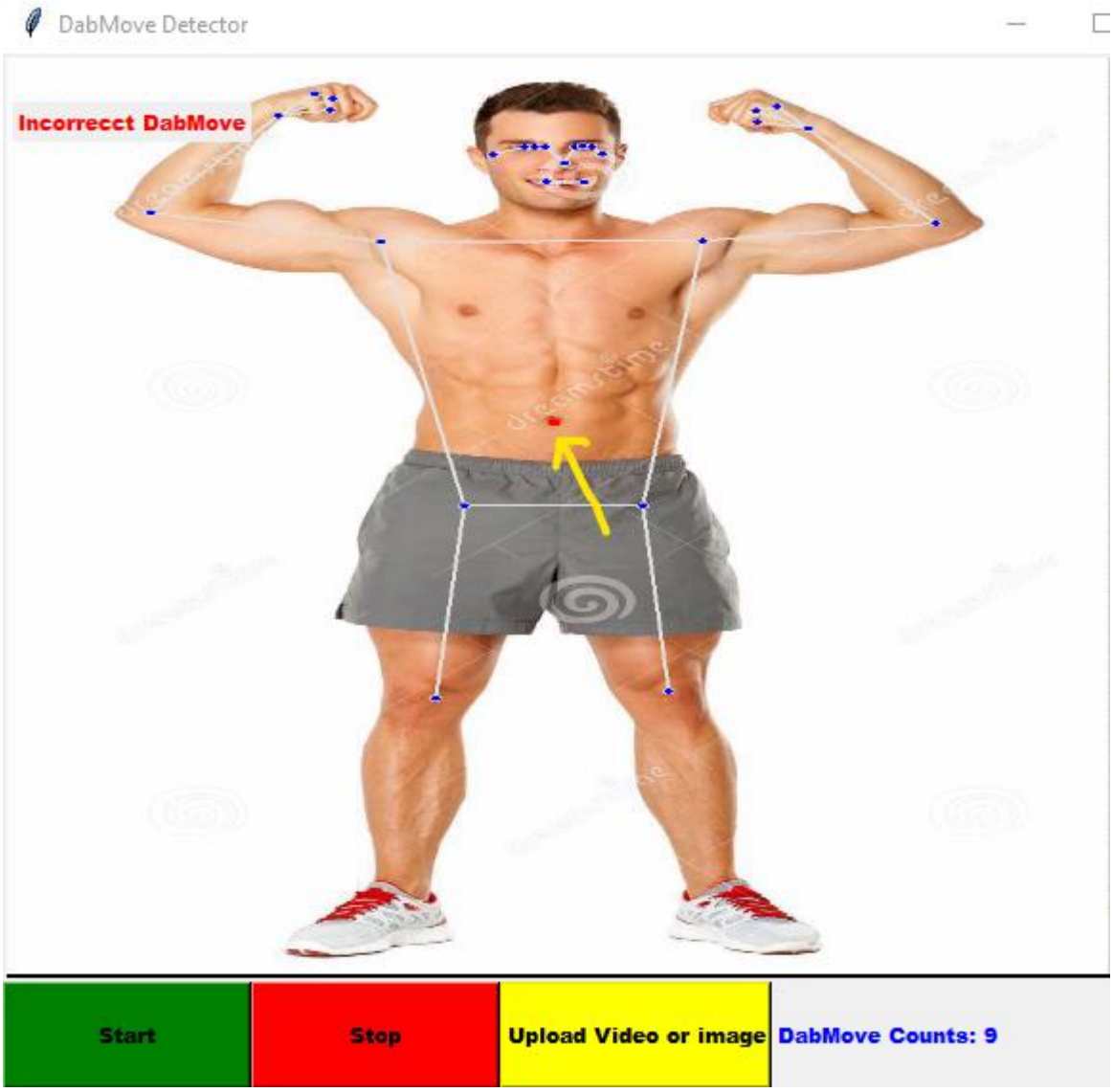
Here is the Methode that uses the pose detection Methode and run the video

```
 def video_detection(self):
        ret, frame = self.cap.read()

        if ret:
            # Convert the image to RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Make a detection
            results = self.holistic.process(image)

            # Draw the detection points on the image
            annotated_image = image.copy()
            self.mp_drawing.draw_landmarks(annotated_image,
results.pose_landmarks, self.mp_holistic.POSE_CONNECTIONS)

            self.detect_pose(results)
            Bally_Button = self.Bally_Button(results)

            # Draw a circle at the Bally_Button
            cv2.circle(annotated_image, (int(Bally_Button[0] *
annotated_image.shape[1]), int(Bally_Button[1] * annotated_image.shape[0])),
5, (255, 0, 0), -1)

            # # Show the annotated image
            # cv2.imshow('MediaPipe Holistic', annotated_image)
            # Resize the image to match the size of the canvas
            resized_image = cv2.resize(annotated_image, (640, 480))
            self.leg_lift_image =
ImageTk.PhotoImage(Image.fromarray(resized_image))
            self.canvas.create_image(0, 0, anchor=tk.NW,
image=self.leg_lift_image)

            # Exit if the user presses the 'q' key
        if self.video_running:
            self.master.after(50, self.video_detection)

        # Release the webcam and close the window
    def __del__(self):
```

```
            self.cap.release()
            cv2.destroyAllWindows()
```

Additional Point

I have added an additional point in pose detection and that is for *Bally Button*




```
 def Bally_Button(self,results):
        left_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_HIP]
        right_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_HIP]
        midpoint = ((left_hip.x + right_hip.x) / 2, (left_hip.y +
right_hip.y) / 2)
        distance_from_hip= (abs(midpoint[0]-left_hip.x),abs(midpoint[1] -
left_hip.y))
        bally_button=(midpoint[0]  , (midpoint[1] - distance_from_hip[0])-
```

```
distance_from_hip[0]/4)
        return bally_button
```

Main function To run the program

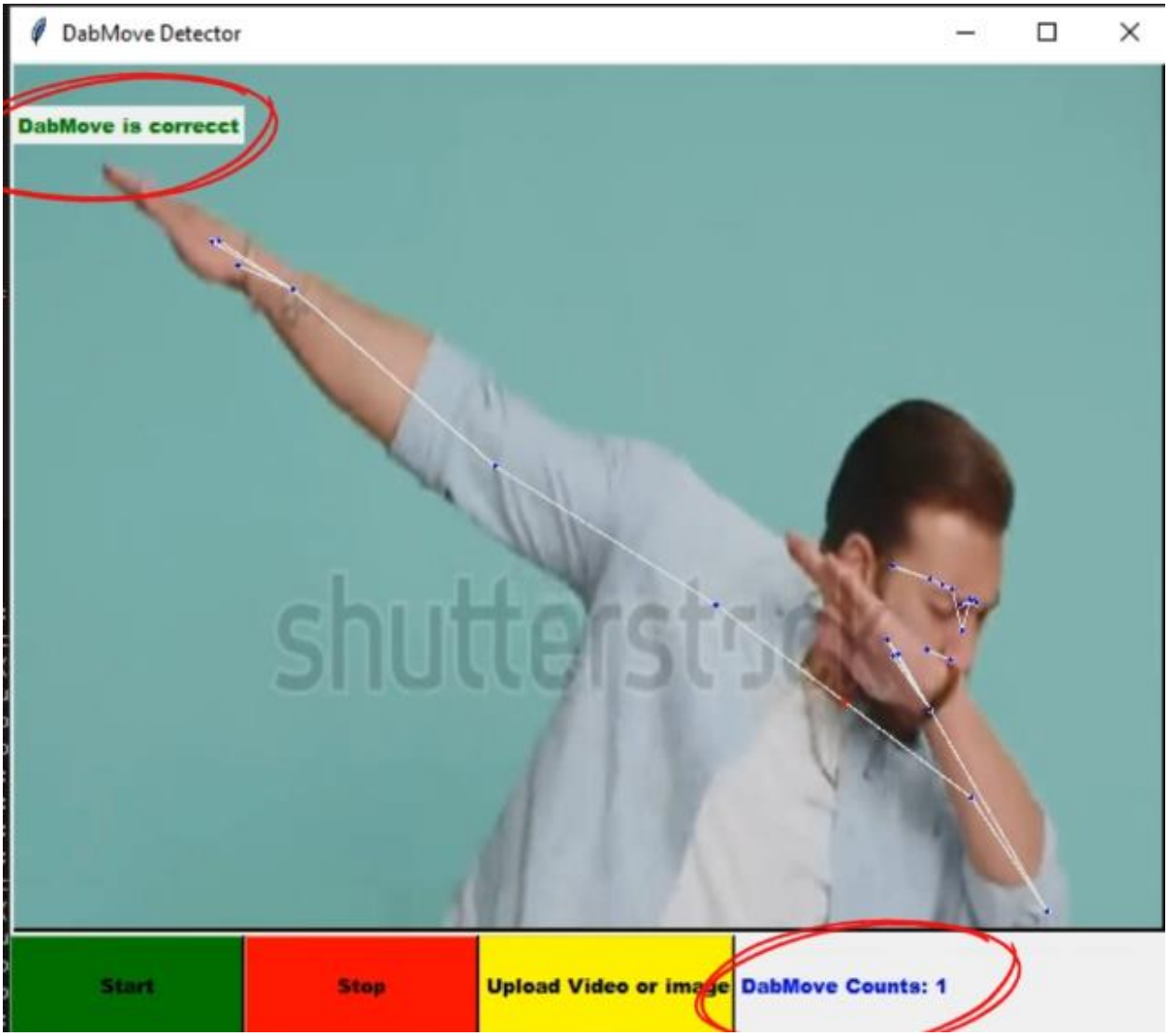
```
if __name__ == "__main__":
    root = tk.Tk()
    app = DabMOve_DetectionGUI(root, 0)
    root.mainloop()
    root.configure(background='black')
    root.mainloop()
```

GUI of this program

I have also done the GUI of this program. The things I have added in it are :

1. Start Button
2. Stop Button
3. Upload Video or Image Button
4. Status statements

All the above things you can see in this image:



Complete Code

```
import cv2
import math
import tkinter as tk
from tkinter import *
import mediapipe as mp
from PIL import Image,ImageTk
import threading
from tkinter import filedialog


def angle_between_lines(x1, y1, x2, y2, x3, y3):
    # Calculate the slopes of the two lines
    slope1 = (y2 - y1) / (x2 - x1)
```

```
    slope2 = (y3 - y2) / (x3 - x2)

    # Calculate the angle between the two lines
    angle = math.atan2(slope2 - slope1, 1 + slope1 * slope2)

    # Convert the angle to degrees and return it
    return math.degrees(angle)



class DabMOve_DetectionGUI:
    def
__init__(self,master,video_path,min_detection_confidence=0.5,min_tracking_con
fidence=0.5 ):
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_holistic = mp.solutions.holistic

        self.master = master
        self.Dabmove = 0
        self.count1 = False
        self.count2 = False
        self.count3 = False
        self.count4 = False
        self.c1,self.c2,self.c3,self.c4=0,0,0,0

        self.cap = cv2.VideoCapture(video_path)
        self.holistic =
self.mp_holistic.Holistic(min_detection_confidence=min_detection_confidence,
min_tracking_confidence=min_tracking_confidence)

        # title on canvas
        self.master = master
        self.master.title("DabMove Detector")
        #  creation of canvas with dimensions
        self.canvas = tk.Canvas(self.master, width=640, height=480 ,
bg="black")
        self.canvas.pack()
        #  title on left upper corner
        self.button_frame = tk.Frame(self.master)
        self.button_frame.pack(side=tk.LEFT)
        # self.button_frame.configure(bg="blue")
        #  start button
        self.start_button = tk.Button(self.button_frame, text="Start",
command=self.start_video,padx=1, width=15, height=3 , bg="green",
font=("Arial Black", 8,"bold"))
        self.start_button.pack(side=tk.LEFT)
        # stop button
        self.stop_button = tk.Button(self.button_frame, text="Stop",
command=self.stop_video,padx=1, width=15, height=3 , bg="red", font=("Arial
Black", 8,"bold"))
        self.stop_button.pack(side=tk.LEFT)
        #  upload button
        self.upload_button = tk.Button(self.button_frame, text="Upload Video
or image", command=self.upload_video,padx=3, width=16, height=3, bg="yellow"
, font=("Arial Black", 8,"bold"))
        self.upload_button.pack(side=tk.LEFT)
```

```
        #  Add the following line to create a new label to show the leg lift
count
        self.DabMove_count_label = tk.Label(self.master, text=f"DabMove
Counts: 0",fg="red", font=("Arial Black", 8,"bold"))
        self.DabMove_count_label.pack(side=tk.LEFT, anchor=tk.CENTER)
        self.angle_correction = tk.Label(self.master, text=f"Incorrect
DabMove",fg="red", font=("Arial Black", 8,"bold"))
        self.angle_correction.place(x=130, y=25, anchor='ne')
        self.DabMove_image = None
        self.video_running = False
        self.video_detection()


    # module for upload video from directory
    def upload_video(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            self.cap = cv2.VideoCapture(file_path)
    # module for start video
    def start_video(self):
        self.video_running = True
        # Start a new thread to read and display video frames continuously
        threading.Thread(target=self.video_detection).start()
    # module for stop or pause video
    def stop_video(self):
        self.video_running = False

    # angle update




    # function for pose detection
*****************************************************************************
*****************************

    def detect_pose(self,results):



        # Get the coordinates of the left hip and right hip
        left_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_SHOULDER]
        right_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_SHOULDER]


        # angle1 between left parts points 11,13,15
        # left_shoulder =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_SHOULDER]
        left_elbow =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_ELBOW]
        left_wrist =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_WRIST]
```

```
        if results.pose_landmarks is not None:
            angle1 = abs(angle_between_lines(left_shoulder.x,
left_shoulder.y, left_elbow.x, left_elbow.y, left_wrist.x, left_wrist.y))
        else:
            angle1=0
        print("Left Arm(Shoulder, Wrist) :",angle1)

        # angle2 between left parts points 23,11,13
        left_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_HIP]

        if results.pose_landmarks is not None:
            angle2 = abs(angle_between_lines(left_hip.x,
left_hip.y,left_shoulder.x, left_shoulder.y, left_elbow.x, left_elbow.y))
        else:
            angle2=0
        print("Left Shoulder-Hip:",angle2)

        # angle3 between left parts points 24,12,14
        right_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_HIP]
        right_elbow =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_ELBOW]

        if results.pose_landmarks is not None:
            angle3 = abs(angle_between_lines(right_hip.x, right_hip.y,
right_shoulder.x, right_shoulder.y, right_elbow.x, right_elbow.y))
        else:
            angle3=0
        print("Right Shoulder-Hip:",angle3)

        # angle4 between left parts points 24,12,14
        right_wrist =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_WRIST]

        if results.pose_landmarks is not None:
            angle4 = abs(angle_between_lines(right_shoulder.x,
right_shoulder.y, right_elbow.x, right_elbow.y, right_wrist.x,
right_wrist.y))
        else:
            angle4=0
        print("Right Shoulder-Wrist:",angle4)
        if (angle1 > 0 and angle1 <= 45):
            self.count1 = True
        else:
            self.count1 = False
        print("Count 1 :",self.count1)
        if (60< angle2 < 155):
            self.count2 = True
        else:
            self.count2 = False
        print("Count 2 :",self.count2)
        if (angle3 > 75 and angle3<145):
            self.count3 = True
        else:
            self.count3 = False
        print("Count 3 :",self.count3)
```

```
        if (angle4 > 0 and angle4  < 30):
            self.count4 = True
        else:
            self.count4 = False
        print("Count 4 :",self.count4)

        if ((self.c1==False or self.c2==False  or self.c4==False or
self.c3==False) ):    #
            if(self.count1==True and self.count2==True and self.count3==True
and self.count4==True):
                self.Dabmove = self.Dabmove + 1
                self.DabMove_count_label.config(text="DabMove Counts:
{}".format(self.Dabmove),fg="blue")

        if(self.count1==True and self.count2==True and self.count3==True and
self.count4==True):

            self.angle_correction.configure(text="DabMove is
correcct",fg="green" )
        else:

            self.angle_correction.configure(text="Incorrecct
DabMove",fg="red" )
        print("DAB detected : ",self.Dabmove)
        self.c1=self.count1
        self.c2=self.count2
        self.c3=self.count3
        self.c4=self.count4


    def video_detection(self):
        ret, frame = self.cap.read()

        if ret:
            # Convert the image to RGB
            image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            # Make a detection
            results = self.holistic.process(image)

            # Draw the detection points on the image
            annotated_image = image.copy()
            self.mp_drawing.draw_landmarks(annotated_image,
results.pose_landmarks, self.mp_holistic.POSE_CONNECTIONS)

            self.detect_pose(results)
            Bally_Button = self.Bally_Button(results)

            # Draw a circle at the Bally_Button
            cv2.circle(annotated_image, (int(Bally_Button[0] *
annotated_image.shape[1]), int(Bally_Button[1] * annotated_image.shape[0])),
5, (255, 0, 0), -1)

            # # Show the annotated image
            # cv2.imshow('MediaPipe Holistic', annotated_image)
            # Resize the image to match the size of the canvas
            resized_image = cv2.resize(annotated_image, (640, 480))
```

```
            self.leg_lift_image =
ImageTk.PhotoImage(Image.fromarray(resized_image))
            self.canvas.create_image(0, 0, anchor=tk.NW,
image=self.leg_lift_image)

            # Exit if the user presses the 'q' key
        if self.video_running:
            self.master.after(50, self.video_detection)

        # Release the webcam and close the window
    def __del__(self):
            self.cap.release()
            cv2.destroyAllWindows()

    def Bally_Button(self,results):
        left_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.LEFT_HIP]
        right_hip =
results.pose_landmarks.landmark[self.mp_holistic.PoseLandmark.RIGHT_HIP]
        midpoint = ((left_hip.x + right_hip.x) / 2, (left_hip.y +
right_hip.y) / 2)
        distance_from_hip= (abs(midpoint[0]-left_hip.x),abs(midpoint[1] -
left_hip.y))
        bally_button=(midpoint[0]  , (midpoint[1] - distance_from_hip[0])-
distance_from_hip[0]/4)
        return bally_button


if __name__ == "__main__":
    root = tk.Tk()
    app = DabMOve_DetectionGUI(root, 0)
    root.mainloop()
    root.configure(background='black')
    root.mainloop()
```

This code is also available on Github

https://github.com/MuhammadTayyab777/Posedetector_python_mediapipe_solution/upload

References:

1. Mediapipe solutions
2. Mediapipe Solution Github code
3. OpwnCV
4. ChatGPT
5. Tkinter
6. Pillow library