Adhi Software Pvt. Ltd.

Running Form Analysis

## DOCUMENTATION

Achyut G Sarojini Pavitra

**Requirements:**

*Preinstalled requirements and the terminal commands to install them*

OpenCv

pip install opencv-python

Mediapipe

pip install mediapipe

Ffmpeg

pip install ffmpeg-python

**Notes:**

**The window has started but is stuck and not analyzing the video**: I have put an input statement to analyze each frame, please comment the following (line 200) or keep pressing enter in the terminal to go to the next frame.

    input(a)  #halt

**Press q to stop**

**Code Explanation**

**Part 1. Landing Analysis:**

Importing None type and OpenCv for video analysis

from types import NoneType

import cv2

importing mediapipe for part detection, os for saving frames in folders, numpy, math for angle and slopes, ffmpeg for detecting phone video rotation(not working fully as intended)

from cv2 import VideoCapture

import mediapipe as mp

import time

import math

import numpy as np

import os

import ffmpeg

To check if the video shot in a phone is rotated upside down(Work in progress)

def check\_rotation(path\_video\_file):

    # this returns meta-data of the video file in form of a dictionary

    meta\_dict = ffmpeg.probe(path\_video\_file)

    # from the dictionary, meta\_dict['streams'][0]['tags']['rotate'] is the key

    # we are looking for

    rotateCode = None

    try:

        if int(meta\_dict['streams'][0]['tags']['rotate']) == 90:

            rotateCode = cv2.ROTATE\_90\_CLOCKWISE

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 180:

            rotateCode = cv2.ROTATE\_180

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 270:

            rotateCode = cv2.ROTATE\_90\_COUNTERCLOCKWISE

    except KeyError:

        pass

    return rotateCode

To correct the phone rotation

def correct\_rotation(frame, rotateCode):

     return cv2.rotate(frame, rotateCode)

Function to find the angle between three points

def angle3(a,b,c):

    ang = math.degrees(math.atan2(c[1]-b[1], c[0]-b[0]) - math.atan2(a[1]-b[1], a[0]-b[0]))

    return ang + 360 if ang < 0 else ang

Function to find and return the slope of a line with two given points

def slopee(x1,y1,x2,y2):

    try:

        return ((y2 - y1) / (x2 - x1))

    except:

        return

ox and oy are to be used as points for displaying the text where the landing was identified

#variables

ox=0

oy=0

count=0

ang=0

c=5 #for counting the number of frames the resulting text should appear

prev\_ground\_assumption=0 #checking previous

flag=0 #for checking foot not moving (ground)

print(cv2.\_\_version\_\_)

Input for opencv and mediapipe and checking if video is not loaded

#cv2 taking input

video\_path="testrun.mp4"

cap = cv2.VideoCapture(video\_path)

if not cap:

    print("Video not loaded")

    exit()

To check phone rotation

rotateCode = check\_rotation(video\_path)

Setting up mediapipe's point and line drawing utils and pose mode

#init mediapipe

mpDraw=mp.solutions.drawing\_utils

mpPose =mp.solutions.pose

pose=mpPose.Pose()

Getting the frame height and width from capture

frame\_width = int(cap.get(3))

frame\_height = int(cap.get(4))

size = (frame\_width, frame\_height)

print(f"frame width{frame\_width}")

Initialising facing\_left to 0 (taken to be facing right but will be changed if the person is facing left)

facing\_left=0

Reducing font size in lower quality videos or when the height and width are less

if frame\_width>400:

    font\_size=.8

else:

    font\_size=.5

result=cv2.VideoWriter('filename.avi', cv2.VideoWriter\_fourcc(\*'MJPG'),10, size)

Loop to read throught each frame of video

while(True):

    ret,img = cap.read()

Break if no frames exist

    if not ret:

        break

#    if rotateCode is not None:

#        frame = correct\_rotation(frame, rotateCode)

Converting color to RGB for mediapipe

    imgRGB=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

sending frames for mediapipe to process

    results=pose.process(imgRGB)

    #print(results.pose\_landmarks)

To stop the program from breaking if the points are not available in some frames, they are enclosed in an if statement and try

    if (results.pose\_landmarks):

        mpDraw.draw\_landmarks(img,results.pose\_landmarks,mpPose.POSE\_CONNECTIONS)

Getting the pose landmarks(points) from mediapipe and passing them into variables for use

    try:

        #LEFT------------------------------------

        #LEFT INDEX

        left\_toe\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_FOOT\_INDEX].x)\*frame\_width)

        left\_toe\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_FOOT\_INDEX].y)\*frame\_height)

        #LEFT HEEL

        left\_heel\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HEEL].x)\*frame\_width)

        left\_heel\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HEEL].y)\*frame\_height)

        #LEFT HIP

        left\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].x)\*frame\_width)

        left\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].y)\*frame\_height)

        #KNEE

        left\_knee\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_KNEE].x)\*frame\_width)

        left\_knee\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_KNEE].y)\*frame\_height)

        #RIGHT------------------------------------

        #RIGHT INDEX

        right\_toe\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_FOOT\_INDEX].x)\*frame\_width)

        right\_toe\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_FOOT\_INDEX].y)\*frame\_height)

        #RIGHT HEEL

        right\_heel\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HEEL].x)\*frame\_width)

        right\_heel\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HEEL].y)\*frame\_height)

        #RIGHT HIP

        right\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].x)\*frame\_width)

        right\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].y)\*frame\_height)

        #KNEE

        right\_knee\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_KNEE].x)\*frame\_width)

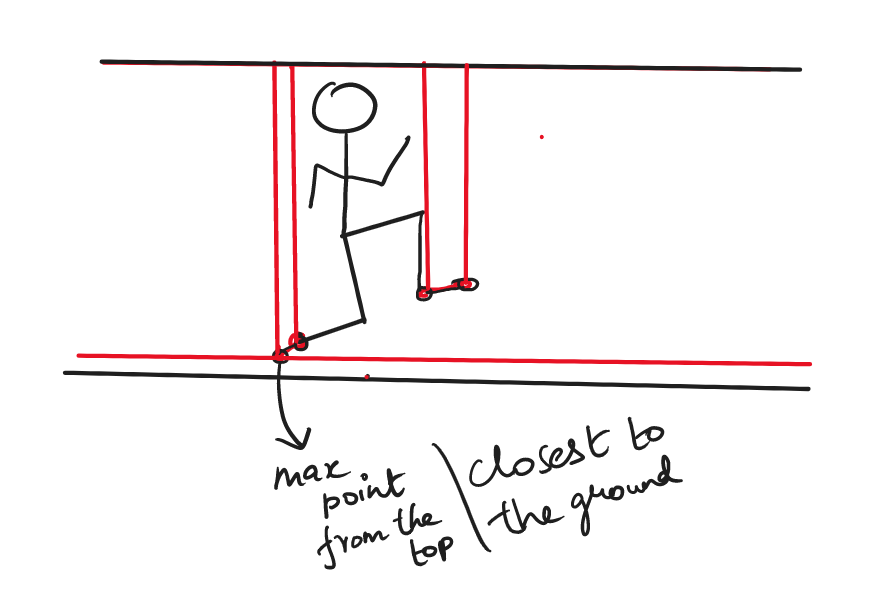
        right\_knee\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_KNEE].y)\*frame\_height)

        #ANCLE

    except:

        continue

An assumption for what could be the ground is taken as a ground assumption the point in the foot which is the max distance from the frame height is likely to hit the ground first. That is we keep track of the points( y coordinates of both heels, and toes) and if they are the closest to the ground should hit the ground first.



    ground\_assump=max(left\_toe\_y,right\_toe\_y,right\_heel\_y,left\_heel\_y)

    print(f"right toe height:{right\_toe\_y} left toe height:{left\_toe\_y}")                 #greater than statement added as test

So if the foot hits the ground the y coordinates should stay the same for the time which the foot stays on the ground. So we check if the ‘ground assumption’(described in the previous paragraph) stays in the ground for 3 frames(worked for most videos) without changing the 10's digit and also without the previous ground assumption being greater than the current frame’s ground assumption(which would mean the foot is moving up), it would be assumed as ground.

For eg:

Ground assumption=259

Previous ground assumption=256

Checking if int(259/10)=int(256/10) (considering only till 10’s digit to account for camera shake)

It is equal (25=25) and 256<259 so count would increase

If count is reached flag is set and count is reset for the next round of checking

    if(int(ground\_assump/10)==int(prev\_ground\_assumption/10) and not prev\_ground\_assumption>ground\_assump):  #checks if bottomost leg stays at the same place for 3 frames

        if(count==3): #3 frames found to be ideal               ^#upto the 10s digit as units place my vary no matter what

            flag=1

            count=0

        else:

            print("ground")

            count+=1

if the flag is not set yet and count is not 3 set the previous ground assumption

    elif not flag==1:

        prev\_ground\_assumption=ground\_assump

checking if left foot hit the ground by equating both the heel and toe points to the ground assumption value

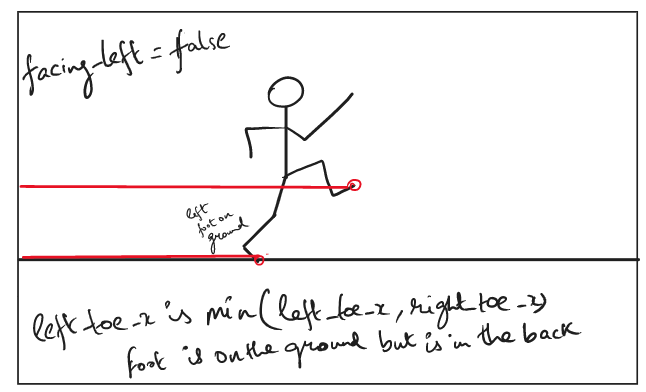
ground\_assump==left\_heel\_y or ground\_assump==left\_toe\_y

also checking if the left foot is in the front.

If the person is facing left( of the screen), the left toe should be the maximum of the left toe and right toe (left toe < right toe)

not facing\_left and left\_toe\_x==max(left\_toe\_x,right\_toe\_x

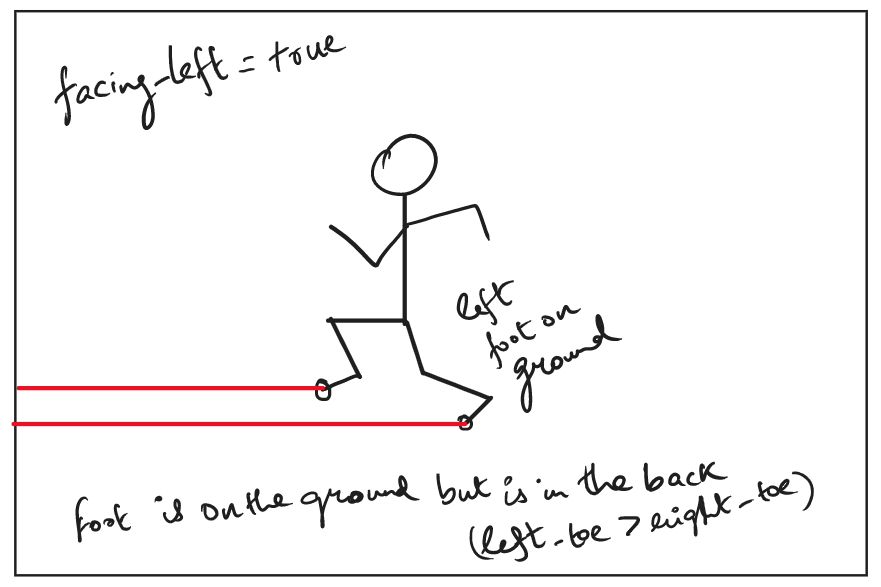
for avoiding the condition below



If the person is facing left( of the screen), the left toe should be the minimum of left toe and right toe (left toe < right toe)

facing\_left and left\_toe\_x==min(left\_toe\_x,right\_toe\_x)

for avoiding the condition below



        if (ground\_assump==left\_heel\_y or ground\_assump==left\_toe\_y) and ((not facing\_left and left\_toe\_x==max(left\_toe\_x,right\_toe\_x)) or (facing\_left and left\_toe\_x==min(left\_toe\_x,right\_toe\_x))): #to check left is on the ground or right

            #alternating foot check(no idea)

            #left=1

            #right=0

            print("ox is the left toe")

            ox=left\_toe\_x

            oy=left\_toe\_y

            slope=slopee(left\_heel\_x,left\_heel\_y,left\_toe\_x,left\_toe\_y)

            ang=angle3((left\_knee\_x,left\_knee\_y),(left\_heel\_x,left\_heel\_y),(left\_toe\_x,left\_toe\_y))

or else if the right foot is on the ground

        elif (ground\_assump==right\_heel\_y or ground\_assump==right\_toe\_y):

            #alternating foot check

            #left=0

            #right=1

            print("ox is the right toe")

            ox=right\_toe\_x # as we dont know if the left side or the right side touches the ground while displaying slope instead we send the values as ox and oy as

            oy=right\_toe\_y

            slope=slopee(right\_heel\_x,right\_heel\_y,right\_toe\_x,right\_toe\_y)

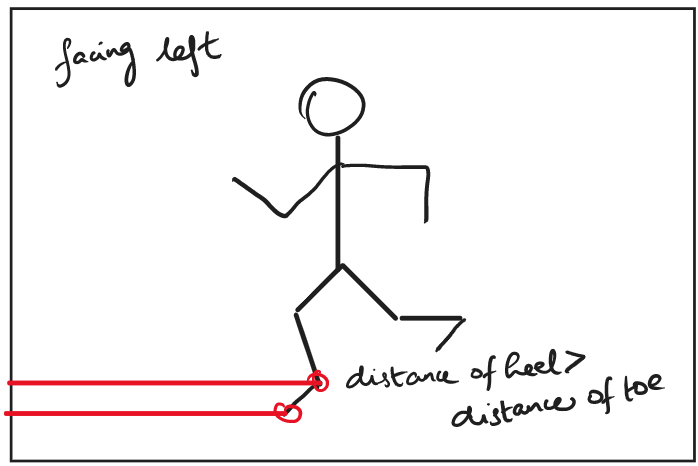
            ang=angle3((right\_knee\_x,right\_knee\_y),(right\_heel\_x,right\_heel\_y),(right\_toe\_x,right\_toe\_y))

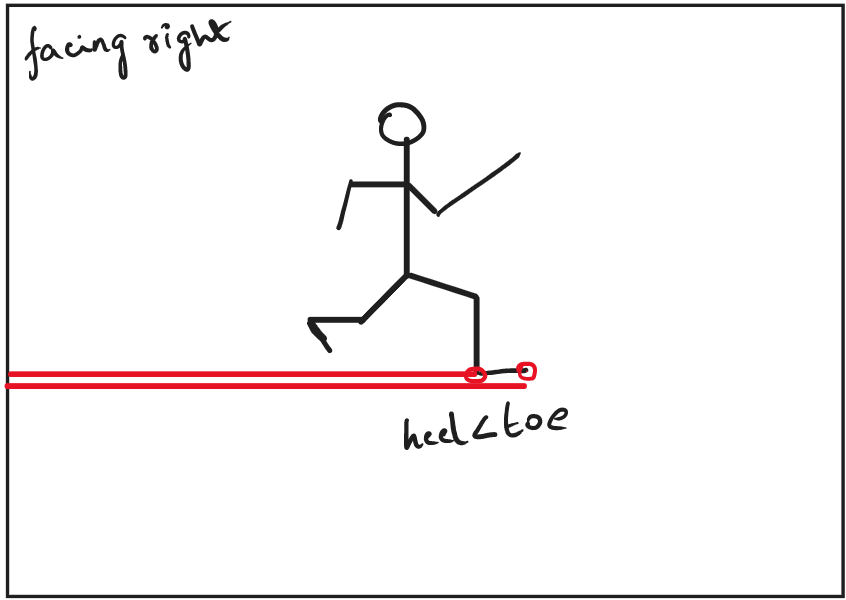
        else:

            pass

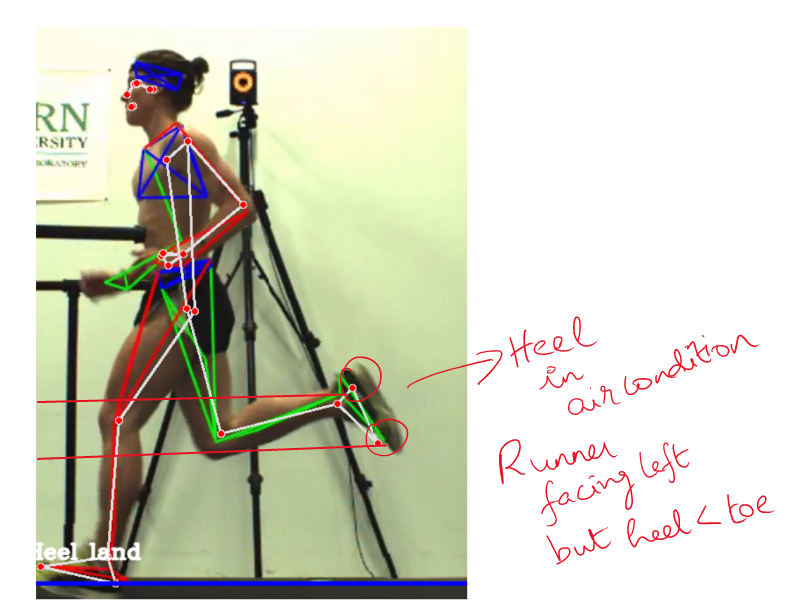
Conditions for checking if the person is facing left

A person facing left is found out by the foot direction, if the toe<heel the runner is facing left, or else if the toe>heel the runner is facing right





But a position would disrupt this condition, That is if the mentioned foot is in the air, So we have to look out if the foot is in the air and check both feet before making a prediction.



So we check if the heel is in the air by checking min of all the foot points is the heel of the feet we are taking the measure from and check and confirm the direction from the other foot by taking the heel and toe

        if((right\_heel\_x>right\_toe\_x)) and min(left\_toe\_y,right\_toe\_y,right\_heel\_y,left\_heel\_y)!=right\_heel\_y: #check left or right and ruling out heel in air condition

            print(f"angbefore{slope}")

            if(ang>250):  #impossible movement

                ang=360-ang

            try:

                slope\*=-1

            except:

                slope=15 #to show Unknown

            print(f"slafter{slope}")

            print("left")

            facing\_left=1

        elif left\_heel\_x<left\_toe\_x : #checking the other leg to rule out the heel in air condition

            facing\_left=0

Impossible angles(angles>250) are changed or if slope does not exist(foot not available in frame) slope is set to a random value for which the final if statement will recognize to show unknown

        else:

facing\_left=1

            print(f"sangbefore{slope}")

            if(ang>250):  #impossible movement

                ang=360-ang

            try:

                slope\*=-1

            except:

                slope=15 #to show Unknown

            print(f"slafter{slope}")

            print("left")

input statement and ‘a’ for stopping at each frame remove for continuous playback

    a=0

    input(a)  #halt

    print(f"angle {ang}")

    #print(f"slope {slope}")

    print(f"rheel {right\_heel\_x}")

    print(f"toe {right\_toe\_x}")

    print(ground\_assump)

Visualizing the ground by drawing a line

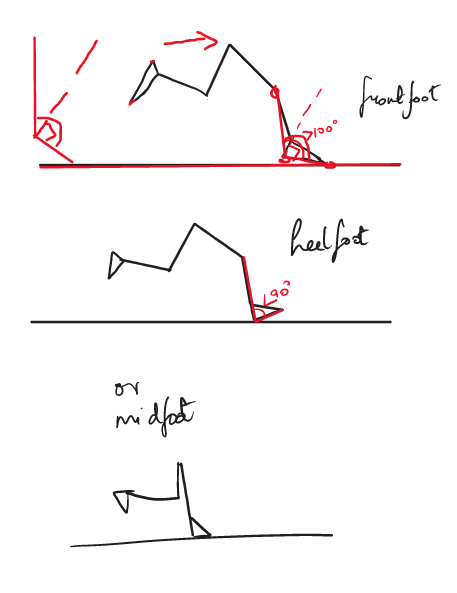
    if ground\_assump:

        cv2.line(img,(0,int(ground\_assump)),(int(frame\_width),int(ground\_assump)),(255,0,0),3)

if the flag is set and c is to check if the text is displayed for 4 frames(could display for more frames number of frames the program detected the ground)

    if flag==1 or c<4 and c>=0:

The landing type is determined by taking the angle and slope as given below



        if slope is NoneType:

            a=1 #garbage for not crashing

            landingtxt=''

        elif((not facing\_left and 0<slope) or ang>100):

            landingtxt='FrontFoot land '    #check landing type with slope

        elif(0>slope and ang<90):

            landingtxt='Heel land'

        elif(-1.25<slope<1.25):

            landingtxt='MidFoot land'

        else:

            landingtxt='Unknown'

        print(f"angle in if{ang}")

        print(f"slope in if{slope}")

        print(f"facing left={facing\_left}")

        print(f"hips{left\_hip\_x,right\_hip\_x}")

        print(f"ox ={ox}")

        print(f"going for a {landingtxt}")

Second check to verify if the leg is not behind the hip and only calculate the landing for the leg in front (changes color to blue to convey landing at the position is unsure)

Facing left and foot is

        if ((ox<(min(left\_hip\_x,right\_hip\_x))) and facing\_left): #not displaying when the leg is behind the hip

            cv2.putText(img,landingtxt,(ox,oy),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,(255,255,255),2, lineType=cv2.LINE\_AA)

w

        elif (not facing\_left and ox>(max(left\_hip\_x,right\_hip\_x))):

            cv2.putText(img,landingtxt,(ox,oy),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,(255,255,255),2, lineType=cv2.LINE\_AA)

w

        else :

            cv2.putText(img,landingtxt+'?',(ox,oy),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,(255,0,0),2, lineType=cv2.LINE\_AA)

reduce frame count (c) for frames displayed and if the count is done count and flag are reset

        c-=1

        if(c==0):

            c=5

            flag=0

display resultant image

    cv2.imshow("Image",img)

save the frames in a folder

    cv2.imwrite(os.path.join('folder',"{:d}.jpg".format(count)), img)

    result.write(img)

press q to stop running

    ch=cv2.waitKey(1)

    if ch & 0xFF == ord('q'):

        break

cv2.destroyAllWindows()

**Part 2. Hip Drop:**

Please refer similar parts from above

from types import NoneType

import cv2

from cv2 import VideoCapture

import mediapipe as mp

import math

import numpy as np

import os

import ffmpeg

def check\_rotation(path\_video\_file):

    # this returns meta-data of the video file in form of a dictionary

    meta\_dict = ffmpeg.probe(path\_video\_file)

    # from the dictionary, meta\_dict['streams'][0]['tags']['rotate'] is the key

    # we are looking for

    rotateCode = None

    try:

        if int(meta\_dict['streams'][0]['tags']['rotate']) == 90:

            rotateCode = cv2.ROTATE\_90\_CLOCKWISE

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 180:

            rotateCode = cv2.ROTATE\_180

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 270:

            rotateCode = cv2.ROTATE\_90\_COUNTERCLOCKWISE

    except KeyError:

        pass

    return rotateCode

def correct\_rotation(frame, rotateCode):

     return cv2.rotate(frame, rotateCode)

#from original code

def angle3(a,b,c):

    ang = math.degrees(math.atan2(c[1]-b[1], c[0]-b[0]) - math.atan2(a[1]-b[1], a[0]-b[0]))

    return ang + 360 if ang < 0 else ang

def calculateDistance(x1,y1,x2,y2):

     dist = math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)

     return dist

def slopee(x1,y1,x2,y2):

    try:

        return ((y2 - y1) / (x2 - x1))

    except:

        return

#variables

ox=0

oy=0

count=0

flag=0

print(cv2.\_\_version\_\_)

#cv2 taking input

video\_path="testrun.mp4"

cap = cv2.VideoCapture(video\_path)

if not cap:

    print("Video not loaded")

    exit()

rotateCode = check\_rotation(video\_path)

#init mediapipe

mpDraw=mp.solutions.drawing\_utils

mpPose =mp.solutions.pose

pose=mpPose.Pose()

frame\_width = int(cap.get(3))

frame\_height = int(cap.get(4))

size = (frame\_width, frame\_height)

print(f"frame width{frame\_width}")

facing\_left=0

if frame\_width>400:

    font\_size=.8

else:

    font\_size=.5

result=cv2.VideoWriter('filename.avi', cv2.VideoWriter\_fourcc(\*'MJPG'),10, size)

while(True):

    ret,img = cap.read()

    if not ret:

        break

#    if rotateCode is not None:

#        frame = correct\_rotation(frame, rotateCode)

    imgRGB=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

    results=pose.process(imgRGB)

    #print(results.pose\_landmarks)

    if (results.pose\_landmarks):

        mpDraw.draw\_landmarks(img,results.pose\_landmarks,mpPose.POSE\_CONNECTIONS)

    try:

        #LEFT HIP

        left\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].x)\*frame\_width)

        left\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].y)\*frame\_height)

        left\_hip\_z=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].z)\*frame\_width)

        #RIGHT------------------------------------

        #RIGHT HIP

        right\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].x)\*frame\_width)

        right\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].y)\*frame\_height)

        right\_hip\_z=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].z)\*frame\_width)

    except:

        continue

We calculate slope on both xy and xz as it might seem that hip dropped in the side view but it might not have in real life so we the z-axis.

    slopex=slopee(left\_hip\_x,left\_hip\_y,right\_hip\_x,right\_hip\_y) # calculating slope with respect to xy axis

    slopez=slopee(left\_hip\_z,left\_hip\_y,right\_hip\_z,right\_hip\_y) # calculating slope with respect to zy axis

    print("lhip z", left\_hip\_z)

    print("rhip z", right\_hip\_z)

    print("slope for x", slopex)

    print("slope for z", slopez)

    try:

As the hip in real life cannot stay straight with 0 slope. The value 0.245 for xy and 0.012 for zy as the threshold gave good results for the test videos ran through the program

        if (slopex<0.245 and slopex>-0.245) and (slopez<0.012 and slopez>-0.012):

            print("No hip drop")

            count=0

        elif ((slopex>0.245 or slopex<-0.245) and (slopez>0.012 or slopez<-0.012)) or (slopez<-0.023 or slopez>0.023): # slope in x and y exists or a large slope in the z axis

If a hip drop is detected the text is displayed for 5 frames(could display for more frames number of frames the program detected a hip drop)

            count+=1

            if(count>4):

                flag=5

                count=0

            print("hip dropped")

        else:

            count=0

        print(f"count {count}")

        cv2.line(img,(right\_hip\_x,right\_hip\_y),(left\_hip\_x,left\_hip\_y),(255,0,0),3)

Check for the lower hip and show hipdrop

        if flag>0:

            if left\_hip\_y>right\_hip\_y:

                cv2.putText(img,"Left Hip drop",(left\_hip\_x,left\_hip\_y),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,(255,255,255),2, lineType=cv2.LINE\_AA)

            else:

                cv2.putText(img,"Right Hip drop",(right\_hip\_x,right\_hip\_y),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,(255,255,255),2, lineType=cv2.LINE\_AA)

            flag-=1

    except:

        pass

halt by Input for analysing

    a=0

    input(a)

    cv2.imshow("Image",img)

    cv2.imwrite(os.path.join('folder',"{:d}.jpg".format(count)), img)

    result.write(img)

    ch=cv2.waitKey(1)

    if ch & 0xFF == ord('q'):

        break

cv2.destroyAllWindows()

**Part 3. Body Lean:**

Please refer similar parts from above

from types import NoneType

import cv2

from cv2 import VideoCapture

import mediapipe as mp

import time

import math

import numpy as np

import os

import ffmpeg

def check\_rotation(path\_video\_file):

    # this returns meta-data of the video file in form of a dictionary

    meta\_dict = ffmpeg.probe(path\_video\_file)

    # from the dictionary, meta\_dict['streams'][0]['tags']['rotate'] is the key

    # we are looking for

    rotateCode = None

    try:

        if int(meta\_dict['streams'][0]['tags']['rotate']) == 90:

            rotateCode = cv2.ROTATE\_90\_CLOCKWISE

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 180:

            rotateCode = cv2.ROTATE\_180

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 270:

            rotateCode = cv2.ROTATE\_90\_COUNTERCLOCKWISE

    except KeyError:

        pass

    return rotateCode

def correct\_rotation(frame, rotateCode):

     return cv2.rotate(frame, rotateCode)

def angle3(a,b,c):

    ang = math.degrees(math.atan2(c[1]-b[1], c[0]-b[0]) - math.atan2(a[1]-b[1], a[0]-b[0]))

    return ang + 360 if ang < 0 else ang

def slopee(x1,y1,x2,y2):

    try:

        return ((y2 - y1) / (x2 - x1))

    except:

        return

#variables

ox=0

oy=0

count=0

ang=0

flag=0

c=5 #for counting the number of frames the resulting text should appear

prev\_ground\_assumption=0 #checking previous

flag=0 #for checking foot not moving (ground)

print(cv2.\_\_version\_\_)

#cv2 taking input

video\_path="testrun.mp4"

cap = cv2.VideoCapture(video\_path)

if not cap:

    print("Video not loaded")

    exit()

rotateCode = check\_rotation(video\_path)

pTime =0

#init mediapipe

mpDraw=mp.solutions.drawing\_utils

mpPose =mp.solutions.pose

pose=mpPose.Pose()

frame\_width = int(cap.get(3))

frame\_height = int(cap.get(4))

size = (frame\_width, frame\_height)

print(f"frame width{frame\_width}")

facing\_left=0

if frame\_width>400:

    font\_size=.8

else:

    font\_size=.5

result=cv2.VideoWriter('filename.avi', cv2.VideoWriter\_fourcc(\*'MJPG'),10, size)

while(True):

    ret,img = cap.read()

    if not ret:

        break

#    if rotateCode is not None:

#        frame = correct\_rotation(frame, rotateCode)

    imgRGB=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

    results=pose.process(imgRGB)

    #print(results.pose\_landmarks)

    if (results.pose\_landmarks):

        mpDraw.draw\_landmarks(img,results.pose\_landmarks,mpPose.POSE\_CONNECTIONS)

    try:

        #LEFT------------------------------------

        left\_ear\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_EAR].x)\*frame\_width)

        left\_ear\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_EAR].y)\*frame\_height)

        #LEFT INDEX

        left\_toe\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_FOOT\_INDEX].x)\*frame\_width)

        left\_toe\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_FOOT\_INDEX].y)\*frame\_height)

        #LEFT HEEL

        left\_heel\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HEEL].x)\*frame\_width)

        left\_heel\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HEEL].y)\*frame\_height)

        #LEFT HIP

        left\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].x)\*frame\_width)

        left\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].y)\*frame\_height)

        left\_hip\_z=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].z)\*frame\_width)

        #KNEE

        left\_knee\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_KNEE].x)\*frame\_width)

        left\_knee\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_KNEE].y)\*frame\_height)

        #RIGHT------------------------------------

        right\_ear\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_EAR].x)\*frame\_width)

        right\_ear\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_EAR].y)\*frame\_height)

        #RIGHT INDEX

        right\_toe\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_FOOT\_INDEX].x)\*frame\_width)

        right\_toe\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_FOOT\_INDEX].y)\*frame\_height)

        #RIGHT HEEL

        right\_heel\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HEEL].x)\*frame\_width)

        right\_heel\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HEEL].y)\*frame\_height)

        #RIGHT HIP

        right\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].x)\*frame\_width)

        right\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].y)\*frame\_height)

        right\_hip\_z=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].z)\*frame\_width)

        #KNEE

        right\_knee\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_KNEE].x)\*frame\_width)

        right\_knee\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_KNEE].y)\*frame\_height)

        #ANCLE

    except:

        continue

Getting the middle point of the head by applying the mid point formula

    head\_middle\_x=((left\_ear\_x+right\_ear\_x)/2)

    head\_middle\_y=((left\_ear\_y+right\_ear\_y)/2)

    hip\_middle\_x=(left\_hip\_x+right\_hip\_x)/2

    hip\_middle\_y=(left\_hip\_y+right\_hip\_y)/2

    cv2.line(img,(int(hip\_middle\_x),0),(int(hip\_middle\_x),int(hip\_middle\_y)),(255,255,0),3)

    cv2.line(img,(int(head\_middle\_x),int(head\_middle\_y)),(int(hip\_middle\_x),int(hip\_middle\_y)),(255,0,0),3)

    angle=angle3((left\_knee\_x,left\_knee\_y),(left\_heel\_x,left\_heel\_y),(left\_toe\_x,left\_toe\_y))  #lean angle

    ang=angle3((int(hip\_middle\_x),0),(int(hip\_middle\_x),int(hip\_middle\_y)),(int(head\_middle\_x),int(head\_middle\_y)))  #foot angle to check foot in air condition

    print("lhip z", left\_hip\_z)

    print("rhip z", right\_hip\_z)

    print("angle", ang)

Forward and backward lean changes the runner's direction, so we change the angle to 360-(body lean angle) if the runner is facing left for the correct results.

    if((right\_heel\_x>right\_toe\_x)) and min(left\_toe\_y,right\_toe\_y,right\_heel\_y,left\_heel\_y)!=right\_heel\_y: #check left or right and ruling out heel in air condition

            if angle>250:  #impossible movement

                facing\_left=facing\_dir\_prev #if ankle in air check the previous frame for reference

            else:

                facing\_left=1

            ang=360-ang   #

            print("left")

    elif left\_heel\_x<left\_toe\_x : #checking the other leg to rule out the heel in air condition

        facing\_left=0

        print("right")

    else:

        if(angle>250):  #impossible movement

            ang=360-ang

        print("left")

    facing\_dir\_prev=facing\_left

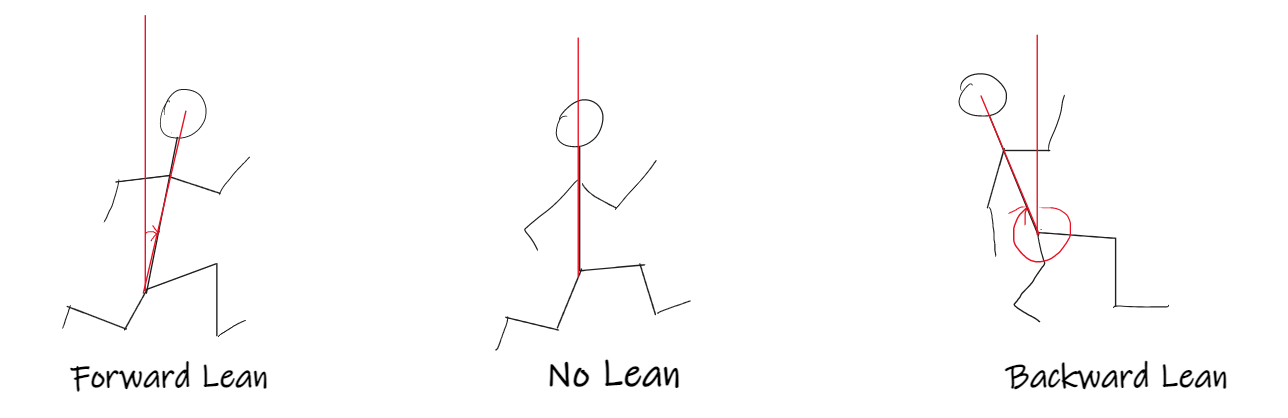
If the angle is less than 5 degrees, No lean

If the angle is greater than 5 and less than 60, Forward Lean

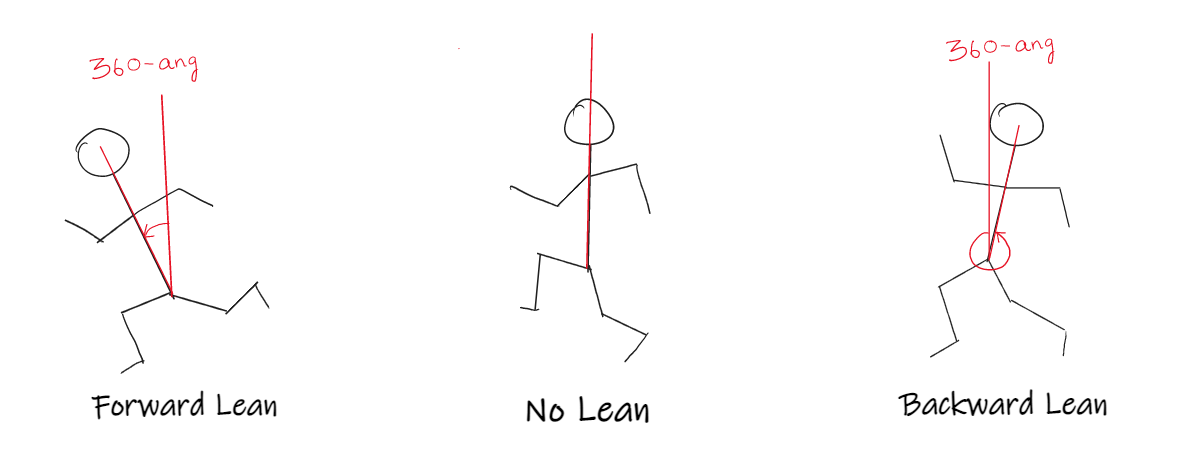
If the angle is greater than 250, It is a backward lean

Anything else is an improper lean

Facing right:



Facing left



    try:

        if(ang>5 and ang<60):

            text="Forward Lean"

            color=(0,255,0)

        elif(ang>=0 and ang<=5):

            text="No lean"

            color=(255,255,255)

        elif(ang>250):

            text="Backward lean"

            color=(0,0,255)

        else:

            text="Improper lean"

            color=(0,0,255)

        cv2.putText(img,text,(int(hip\_middle\_x),int(hip\_middle\_y)),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,color,2, lineType=cv2.LINE\_AA)

        cv2.putText(img,"Angle:"+str(round(ang,2)),(int(hip\_middle\_x),int(hip\_middle\_y)+25),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,color,2, lineType=cv2.LINE\_AA)

    except:

        pass

    a=0

    #input(a)

    cv2.imshow("Image",img)

    cv2.imwrite(os.path.join('folder',"{:d}.jpg".format(count)), img)

    result.write(img)

    ch=cv2.waitKey(1)

    if ch & 0xFF == ord('q'):

        break

cv2.destroyAllWindows()

**Part 4. Vertical bounce:**

Please refer similar parts from above

from turtle import width

from types import NoneType

import cv2

from cv2 import VideoCapture

import mediapipe as mp

import time

import math

import numpy as np

import os

import ffmpeg

def check\_rotation(path\_video\_file):

    # this returns meta-data of the video file in form of a dictionary

    meta\_dict = ffmpeg.probe(path\_video\_file)

    # from the dictionary, meta\_dict['streams'][0]['tags']['rotate'] is the key

    # we are looking for

    rotateCode = None

    try:

        if int(meta\_dict['streams'][0]['tags']['rotate']) == 90:

            rotateCode = cv2.ROTATE\_90\_CLOCKWISE

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 180:

            rotateCode = cv2.ROTATE\_180

        elif int(meta\_dict['streams'][0]['tags']['rotate']) == 270:

            rotateCode = cv2.ROTATE\_90\_COUNTERCLOCKWISE

    except KeyError:

        pass

    return rotateCode

def correct\_rotation(frame, rotateCode):

     return cv2.rotate(frame, rotateCode)

#from original code

def calculateDistance(x1,y1,x2,y2):

     dist = math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)

     print("distance", x1 , y1, x2, y2)

     return dist

#variables

ox=0

oy=0

count=0

ang=0

c=5 #for counting the number of frames the resulting text should appear

prev\_ground\_assumption=0 #checking previous

flag=0 #for checking foot not moving (ground)

print(cv2.\_\_version\_\_)

#cv2 taking input

video\_path="testrun.mp4"

cap = cv2.VideoCapture(video\_path)

if not cap:

    print("Video not loaded")

    exit()

rotateCode = check\_rotation(video\_path)

#init mediapipe

mpDraw=mp.solutions.drawing\_utils

mpPose =mp.solutions.pose

pose=mpPose.Pose()

frame\_width = int(cap.get(3))

frame\_height = int(cap.get(4))

size = (frame\_width, frame\_height)

print(f"frame width{frame\_width}")

facing\_left=0

if frame\_width>400:

    font\_size=.8

else:

    font\_size=.5

result=cv2.VideoWriter('filename.avi', cv2.VideoWriter\_fourcc(\*'MJPG'),10, size)

while(True):

    ret,img = cap.read()

    if not ret:

        break

#    if rotateCode is not None:

#        frame = correct\_rotation(frame, rotateCode)

    imgRGB=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

    results=pose.process(imgRGB)

    #print(results.pose\_landmarks)

    if (results.pose\_landmarks):

        mpDraw.draw\_landmarks(img,results.pose\_landmarks,mpPose.POSE\_CONNECTIONS)

    try:

        #LEFT------------------------------------

        #LEFT INDEX

        left\_toe\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_FOOT\_INDEX].x)\*frame\_width)

        left\_toe\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_FOOT\_INDEX].y)\*frame\_height)

        #LEFT HEEL

        left\_heel\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HEEL].x)\*frame\_width)

        left\_heel\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HEEL].y)\*frame\_height)

        #LEFT HIP

        left\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].x)\*frame\_width)

        left\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_HIP].y)\*frame\_height)

        #KNEE

        left\_knee\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_KNEE].x)\*frame\_width)

        left\_knee\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.LEFT\_KNEE].y)\*frame\_height)

        #RIGHT------------------------------------

        #RIGHT INDEX

        right\_toe\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_FOOT\_INDEX].x)\*frame\_width)

        right\_toe\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_FOOT\_INDEX].y)\*frame\_height)

        #RIGHT HEEL

        right\_heel\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HEEL].x)\*frame\_width)

        right\_heel\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HEEL].y)\*frame\_height)

        #RIGHT HIP

        right\_hip\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].x)\*frame\_width)

        right\_hip\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_HIP].y)\*frame\_height)

        #KNEE

        right\_knee\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_KNEE].x)\*frame\_width)

        right\_knee\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_KNEE].y)\*frame\_height)

        #ANKLE

        right\_ankle\_x=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_ANKLE].x)\*frame\_width)

        right\_ankle\_y=int((results.pose\_landmarks.landmark[mpPose.PoseLandmark.RIGHT\_ANKLE].y)\*frame\_height)

    except:

        continue

As we need a temporary reference point to calculate vertical bounce we need moment to capture the hip reference for before and after versions. For the purpose we take part of the landing code above and remove any parts, we do not require (Landing type slope and landing.

We don’t need facing\_left for the vertical bounce but we do require it for checking if the runner’s landing foot is behind the hip.

    hip\_middle\_x=(left\_hip\_x+right\_hip\_x)/2

    hip\_middle\_y=(left\_hip\_y+right\_hip\_y)/2

    ground\_assump=max(left\_toe\_y,right\_toe\_y,right\_heel\_y,left\_heel\_y)

    print(f"right toe height:{right\_toe\_y} left toe height:{left\_toe\_y}")                 #greater than statement added as test

    if(int(ground\_assump/10)==int(prev\_ground\_assumption/10) and not prev\_ground\_assumption>ground\_assump):  #checks if bottomost leg stays at the same place for 3 frames

        if(count==3): #3 frames found to be ideal               ^#upto the 10s digit as units place my vary no matter what

We set the flag if the foot is detected still in the y axis for 3 frames.

            flag=1

            count=0

        else:

            print("ground")

            count+=1

    elif not flag==1:

        prev\_ground\_assumption=ground\_assump

        if (ground\_assump==left\_heel\_y or ground\_assump==left\_toe\_y) and ((not facing\_left and left\_toe\_x==max(left\_toe\_x,right\_toe\_x)) or (facing\_left and left\_toe\_x==min(left\_toe\_x,right\_toe\_x))): #to check left is on the ground or right

            #alternating foot check(no idea)

            #left=1

            #right=0

            print("ox is the left toe")

            ox=left\_toe\_x

            oy=left\_toe\_y

        elif (ground\_assump==right\_heel\_y or ground\_assump==right\_toe\_y):

            #alternating foot check

            #left=0

            #right=1

            print("ox is the right toe")

            ox=right\_toe\_x # as we dont know if the left side or the right side touches the ground while displaying slope instead we send the values as ox and oy as

            oy=right\_toe\_y

        else:

            pass

        if((right\_heel\_x>right\_toe\_x)) and min(left\_toe\_y,right\_toe\_y,right\_heel\_y,left\_heel\_y)!=right\_heel\_y: #check left or right and ruling out heel in air condition

            print("left")

            facing\_left=1

        elif left\_heel\_x<left\_toe\_x : #checking the other leg to rule out the heel in air condition

            facing\_left=0

        else:

            facing\_left=1

    a=0

    input(a)  #halt

    print(f"rheel {right\_heel\_x}")

    print(f"toe {right\_toe\_x}")

    print(ground\_assump)

If condition to set a reference hip only if the foot is on the ground(flag is 1) and landing leg is behind the hip even if the running is facing both sides.

    if flag==1 and ((facing\_left and ox<=hip\_middle\_x) or (not facing\_left and ox>=hip\_middle\_x)):

To set reference hip to the hip position in the frame as foot is on the ground.

        reference\_hip\_y=(left\_hip\_y+right\_hip\_y)/2

        flag=0

Set bounce distance threshold to 10 for optimal bounce (displays in blue) anything greater than 10 is bad and will display in red.

    bounceThresh=10

    try:

Calculating distance from reference hip to the current hip for every frame that a reference hip is set

        bounce\_dist=calculateDistance(hip\_middle\_x,reference\_hip\_y,hip\_middle\_x,hip\_middle\_y)

        cv2.line(img,(int(left\_hip\_x),int(reference\_hip\_y)),(int(right\_hip\_x),int(reference\_hip\_y)),(255,255,0),3)

        if bounce\_dist<=bounceThresh:

            color=(255,0,0)

        else:

            color=(0,0,255)

        cv2.line(img,(int(hip\_middle\_x),int(reference\_hip\_y)),(int(hip\_middle\_x),int(hip\_middle\_y)),color,3)

        cv2.putText(img,str(bounce\_dist),(int(hip\_middle\_x+10),int(reference\_hip\_y)),cv2.FONT\_HERSHEY\_COMPLEX,font\_size,color,2, lineType=cv2.LINE\_AA)

    except:pass

    cv2.imshow("Image",img)

    cv2.imwrite(os.path.join('folder',"{:d}.jpg".format(count)), img)

    result.write(img)

    ch=cv2.waitKey(1)

    if ch & 0xFF == ord('q'):

        break

cv2.destroyAllWindows()

**Calling type program changes:**

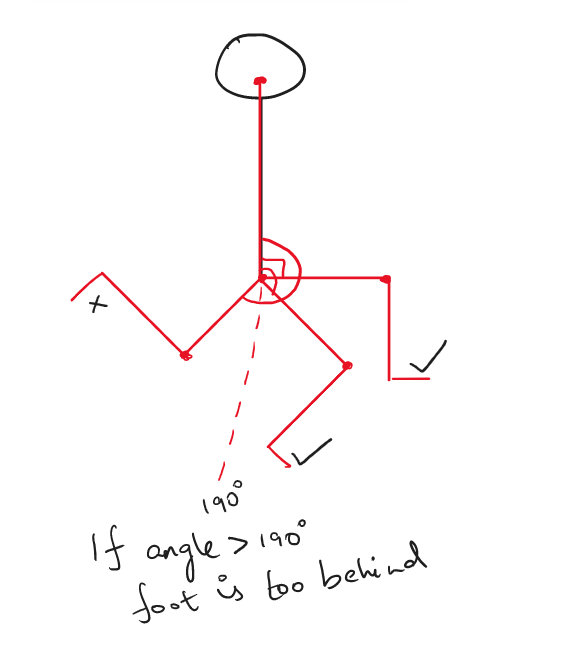
Included body part variables (left\_hip\_x,left\_heel\_y…..) into a dictionary (partdict) for easy passing and returning

The Program returns all and takes in the variables as arguments in the next frame whose values are taken or depend on the previous frames.

**Pending ideas:**

**Stride length:** Just taking 2x the largest distance for each landing

**Guessing if the foot is too behind the hip (different and better method):** angle between head middle, hip middle and landing foot’s knee (foot on ground)

****