# Human Posture Detection

Open in Colab

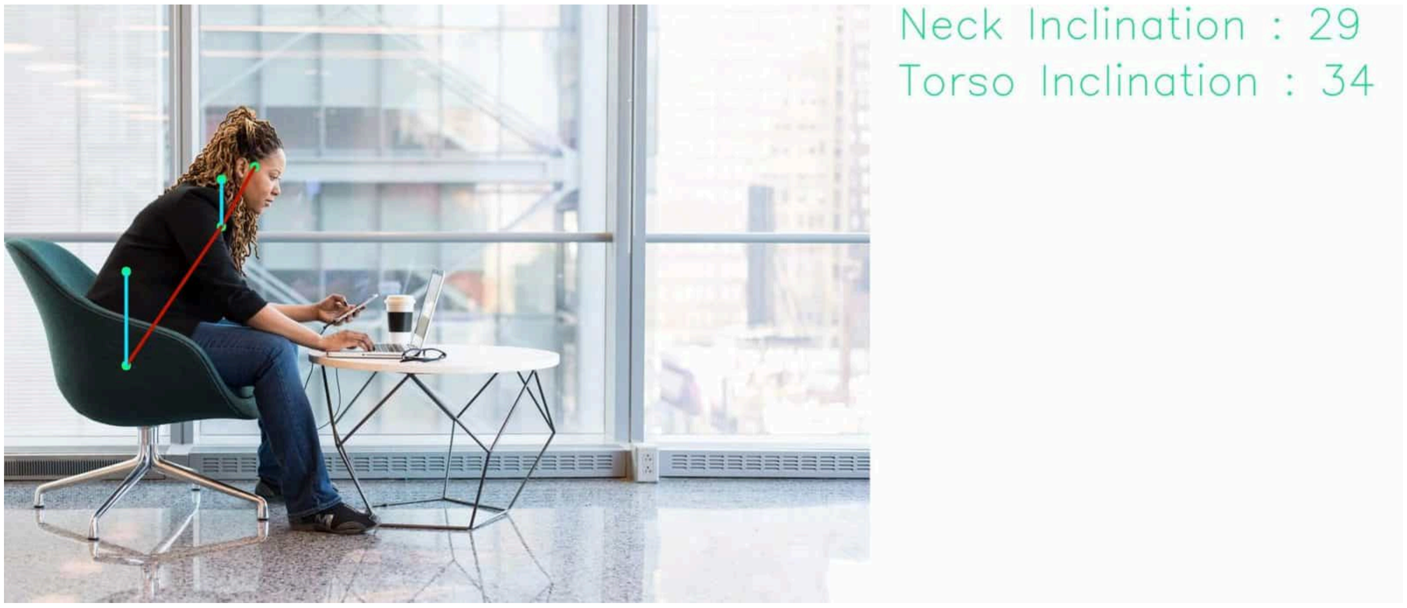
Poor posture is a modern-day epidemic. Research has shown that children as young as 10 years of age are demonstrating spinal degeneration on x-ray. Certain postures, like forward head posture, have been linked to migraines, high blood pressure, and decreased lung capacity. In this notebook, you will learn to create an application to detect posture using mediapipe. We are going to use side view samples to perform analysis and draw conclusion. Practical application would require an webcam pointed at the side view of the person.

## Workflow

- Find point of interests (landmarks) to check angle.
- Perform analysis on standard sample images.
- Find threshold range for good and bad posture.
- Apply on video/webcam input.

### Goal

To detect neck inclination and torso inclination as shown below.



```
if 'google.colab' in str(get_ipython()):
    !pip install opencv-python
    !pip install mediapipe
    !wget https://raw.githubusercontent.com/spmallick/learnopencv/blob/master/Posture-anal
else:
    pass
```

## Import Libraries

```
import cv2
import time
import math as m
import mediapipe as mp
```

```
# Initilize medipipe selfie segmentation class.
mp_pose = mp.solutions.pose
mp_holistic = mp.solutions.holistic
```

## Function to calculate offset distance

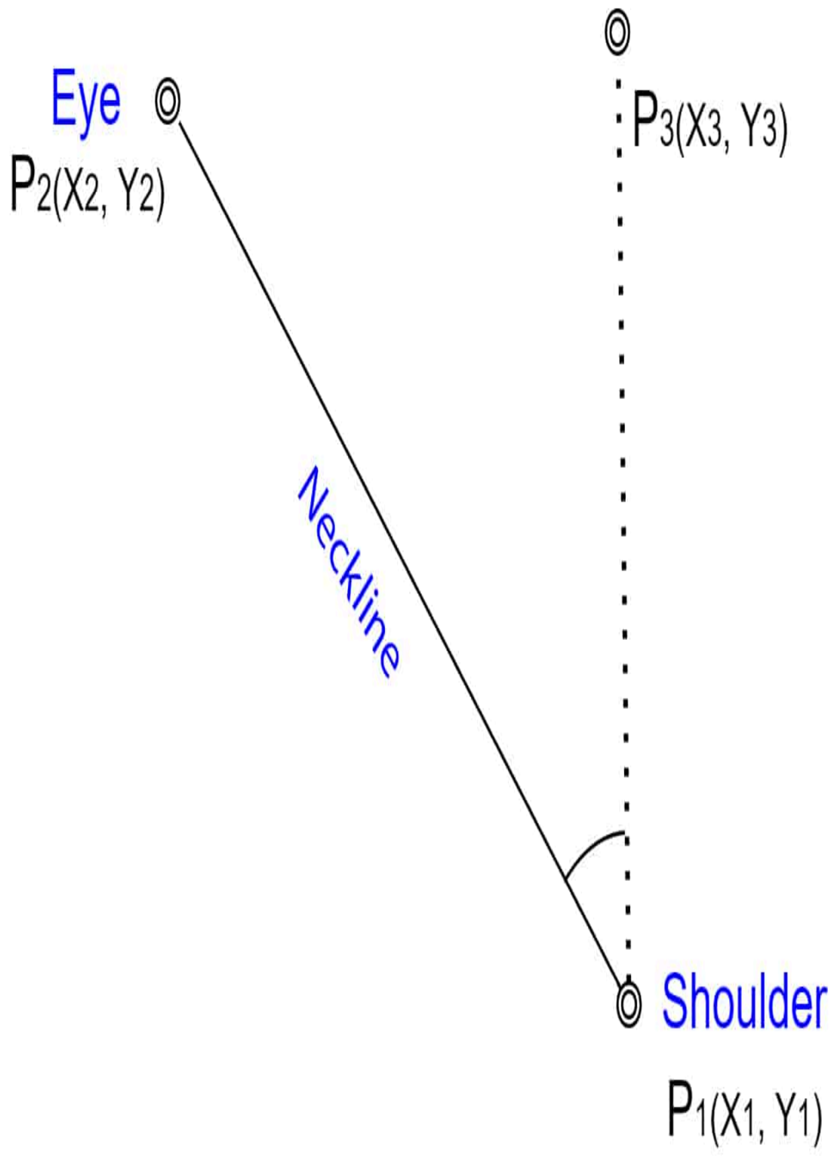
The setup requires the person to be in proper side view. **`findDistance`** function helps us determine offset distance between two poinst. It can be the hip points, the eyes or shoulder. As these points are always more or less symmetric about the central axis. With this, we are going to incorporate camera alignment assistance in the script. Distnace is calculated using the distance formula.

$$distance = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

```
def findDistance(x1, y1, x2, y2):
    dist = m.sqrt((x2-x1)**2+(y2-y1)**2)
    return dist
```

## Function to calculate angle subtended by the line of interest to y-axis

This is the primary deterministic factor for the posture. Using the angle subtended by the neck line and the torso line to y-axis. The neck line connects the shoulder and the eye, here we take shoulder as the pivotal point. Similarly the torso line connects hip and the shoulder, where hip is considered pivotal point.



Taking neck line as an example, here points are $P_1(x_1, y_1)$(shoulder), $P_2(x_2, y_2)$ (eye) and $P_3(x_3, y_3)$ (any point on vertical axis passing through $P_1$).
Hence, for $P_3$ x-coordinate is same as to that of $P_1$ and since $y_3$ is valid for all y, let's take $y_3 = 0$ for simplicity.

To find the inner angle of three points, we take vector approach. Angle between two vectors $\overrightarrow{P_{12}}$ and $\overrightarrow{P_{13}}$ is given by,

```
# Calculate angle.
def findAngle(x1, y1, x2, y2):
    theta = m.acos((y2 -y1)*(-y1) / (m.sqrt((x2 - x1)**2 + (y2 - y1)**2) * y1))
    degree = int(180/m.pi)*theta
    return degree
```

## Function to send alert

Use this function to send alerts when bad posture is detected. Feel free to get creative and customize as per your convenience. You can use telegram notifiction alert system, chek out Telegram bot automation here. Or you can take it up a notch by creating an android app.

```
def sendWarning(x):
    pass
```

## Constants and Initializations

```
# Initilize frame counters.
good_frames = 0
bad_frames = 0

# Font type.
font = cv2.FONT_HERSHEY_SIMPLEX

# Colors.
blue = (255, 127, 0)
red = (50, 50, 255)
green = (127, 255, 0)
dark_blue = (127, 20, 0)
light_green = (127, 233, 100)
yellow = (0, 255, 255)
pink = (255, 0, 255)


# Initialize mediapipe pose class.
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()

# Initialize video capture object.
# For webcam input replace file name with 0.
file_name = 'input.mp4'
cap = cv2.VideoCapture(file_name)

# Meta.
fps = int(cap.get(cv2.CAP_PROP_FPS))
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
frame_size = (width, height)
fourcc = cv2.VideoWriter_fourcc(*'mp4v')

# Initialize video writer.
video_output = cv2.VideoWriter('output.mp4', fourcc, fps, frame_size)
```

## Processing

Processing the image using mediapipe is fairly simple, after setting minimum detection confidence and minimum tracking confidence, we need to pass the RGB image to `pose.process()` method. The documentation can be found in this **link**. With the acquired result, we find the coordinates of specific landmarks. Then we find the angles suntended by the line of interset to y-axis and draw conclusion on the basis of results obtained from analysis script. The code is self explanatory.

```
print('Processing..')
while cap.isOpened():
    # Capture frames.
    success, image = cap.read()
    if not success:
        print("Null.Frames")
        break
    # Get fps.
    fps = cap.get(cv2.CAP_PROP_FPS)
    # Get height and width.
    h, w = image.shape[:2]

    # Convert the BGR image to RGB.
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Process the image.
    keypoints = pose.process(image)

    # Convert the image back to BGR.
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    # Use lm and lmPose as representative of the following methods.
    lm = keypoints.pose_landmarks
    lmPose = mp_pose.PoseLandmark

    # Acquire the landmark coordinates.
    # Once aligned properly, left or right should not be a concern.
    # Left shoulder.
    l_shldr_x = int(lm.landmark[lmPose.LEFT_SHOULDER].x * w)
    l_shldr_y = int(lm.landmark[lmPose.LEFT_SHOULDER].y * h)
    # Right shoulder
    r_shldr_x = int(lm.landmark[lmPose.RIGHT_SHOULDER].x * w)
    r_shldr_y = int(lm.landmark[lmPose.RIGHT_SHOULDER].y * h)
    # Left ear.
    l_ear_x = int(lm.landmark[lmPose.LEFT_EAR].x * w)
    l_ear_y = int(lm.landmark[lmPose.LEFT_EAR].y * h)
    # Left hip.
    l_hip_x = int(lm.landmark[lmPose.LEFT_HIP].x * w)
    l_hip_y = int(lm.landmark[lmPose.LEFT_HIP].y * h)

    # Calculate distance between left shoulder and right shoulder points.
    offset = findDistance(l_shldr_x, l_shldr_y, r_shldr_x, r_shldr_y)

    # Assist to align the camera to point at the side view of the person.
    # Offset threshold 30 is based on results obtained from analysis over 100 samples.
    if offset < 100:
        cv2.putText(image, str(int(offset)) + ' Aligned', (w - 150, 30), font, 0.9, green,
    else:
        cv2.putText(image, str(int(offset)) + ' Not Aligned', (w - 150, 30), font, 0.9, re

    # Calculate angles.
    neck_inclination = findAngle(l_shldr_x, l_shldr_y, l_ear_x, l_ear_y)
```

```
torso_inclination = findAngle(l_hip_x, l_hip_y, l_shldr_x, l_shldr_y)

# Draw landmarks.
cv2.circle(image, (l_shldr_x, l_shldr_y), 7, yellow, -1)
cv2.circle(image, (l_ear_x, l_ear_y), 7, yellow, -1)

# Let's take y - coordinate of P3 100px above x1,  for display elegance.
# Although we are taking y = 0 while calculating angle between P1,P2,P3.
cv2.circle(image, (l_shldr_x, l_shldr_y - 100), 7, yellow, -1)
cv2.circle(image, (r_shldr_x, r_shldr_y), 7, pink, -1)
cv2.circle(image, (l_hip_x, l_hip_y), 7, yellow, -1)

# Similarly, here we are taking y - coordinate 100px above x1. Note that
# you can take any value for y, not necessarily 100 or 200 pixels.
cv2.circle(image, (l_hip_x, l_hip_y - 100), 7, yellow, -1)

# Put text, Posture and angle inclination.
# Text string for display.
angle_text_string = 'Neck : ' + str(int(neck_inclination)) + '  Torso : ' + str(int(to

# Determine whether good posture or bad posture.
# The threshold angles have been set based on intuition.
if neck_inclination < 40 and torso_inclination < 10:
    bad_frames = 0
    good_frames += 1

    cv2.putText(image, angle_text_string, (10, 30), font, 0.9, light_green, 2)
    cv2.putText(image, str(int(neck_inclination)), (l_shldr_x + 10, l_shldr_y), font,
    cv2.putText(image, str(int(torso_inclination)), (l_hip_x + 10, l_hip_y), font, 0.9

    # Join landmarks.
    cv2.line(image, (l_shldr_x, l_shldr_y), (l_ear_x, l_ear_y), green, 4)
    cv2.line(image, (l_shldr_x, l_shldr_y), (l_shldr_x, l_shldr_y - 100), green, 4)
    cv2.line(image, (l_hip_x, l_hip_y), (l_shldr_x, l_shldr_y), green, 4)
    cv2.line(image, (l_hip_x, l_hip_y), (l_hip_x, l_hip_y - 100), green, 4)

else:
    good_frames = 0
    bad_frames += 1

    cv2.putText(image, angle_text_string, (10, 30), font, 0.9, red, 2)
    cv2.putText(image, str(int(neck_inclination)), (l_shldr_x + 10, l_shldr_y), font,
    cv2.putText(image, str(int(torso_inclination)), (l_hip_x + 10, l_hip_y), font, 0.9

    # Join landmarks.
    cv2.line(image, (l_shldr_x, l_shldr_y), (l_ear_x, l_ear_y), red, 4)
    cv2.line(image, (l_shldr_x, l_shldr_y), (l_shldr_x, l_shldr_y - 100), red, 4)
    cv2.line(image, (l_hip_x, l_hip_y), (l_shldr_x, l_shldr_y), red, 4)
    cv2.line(image, (l_hip_x, l_hip_y), (l_hip_x, l_hip_y - 100), red, 4)

# Calculate the time of remaining in a particular posture.
good_time = (1 / fps) * good_frames
bad_time =  (1 / fps) * bad_frames

# Pose time.
if good_time > 0:
    time_string_good = 'Good Posture Time : ' + str(round(good_time, 1)) + 's'
    cv2.putText(image, time_string_good, (10, h - 20), font, 0.9, green, 2)
else:
    time_string_bad = 'Bad Posture Time : ' + str(round(bad_time, 1)) + 's'
```

```
        cv2.putText(image, time_string_bad, (10, h - 20), font, 0.9, red, 2)

    # If you stay in bad posture for more than 3 minutes (180s) send an alert.
    if bad_time > 180:
        sendWarning()
    # Write frames.
    video_output.write(image)
print('Finished.')
cap.release()
video_output.release()
```