Iran University of Science and Technology

Machine Vision

Dr.Shahbazi

Final Project

Human Pose Detection and Classification Using MediaPipe and Depth Camera

---

Name: Siavash Mahmoudi

Semester: 01

TA Name: –

Student Num: 99741195

E-Mail: si-mahmoudi@mecheng.iust.ac.ir

Date:Jan 2022

Contents

1 Abstract

Human activity monitoring via pose estimation has a lot of real-world applications. Real-time pose estimation can be used for solving many problems in fields like fitness training, sports coaching, gaming, motion capture, assisted living, and Robotics. Our focus was on utilizing human pose estimation and classification for Robotics perception. I use MediaPipe which is Google's open-source framework that estimates the human pose and Intel RealSense D435 depth camera for getting depth frame. The focus of this software is to identify some poses to our quadruped robot. To achieve this, we reviewed different pose estimation models, aligning depth frame with RGB frame, using extracted human body joints and their depth value for posture classification. the time and depth differences help us to the determination of velocity and acceleration.

Keywords: Pose Estimation, Mediapipe, Human Activity Recognition, Depth Camera, Intel RealSense,

2 Introduction

Human activity monitoring via pose estimation has a lot of real-world applications. Pose estimation can be used for solving many problems in many fields like sports coaching (to monitor athlete's movements to a high degree of precision), assisted living (fall detection for elder, differently-abled people), industrial spaces (monitoring workers poses for avoiding some dangerous tasks like smoking) and so on.

Pose estimation predicts the body key points of a person in each frame of a video feed. Current state-of-the-art deep learning-based pose estimation models perform well however for the use cases discussed above, real-time on-device execution on mobile devices and mid-range desktop setups is very crucial along with good accuracy and speed. Our objective was to make a real-time application that performs pose estimation with high speed and accuracy while providing a smooth control.

Many models have been tested in this field including Lightweight OpenPose [1], Pifpaf [2], Tensorflow-Lite(mobile net) [3], TensorflowJS (mobile net, resnet) [4], and Blazepose(via Mediapipe API) models.

In an attempt to abstract over the video information, skeleton poses have been suggested as an explainable, person independent, privacy-preserving, and low-dimensional representation that provides the signer body pose and information on how it changes over time. Theoretically, skeletal poses contain all the relevant information required to understand signs produced in videos, except for interactions with elements in space.

The recording of accurate human skeleton poses is difficult and often intrusive, requiring signers to wear specialized and expensive motion capture hardware. Fortunately, advances in computer vision now allow the estimation of human skeleton poses directly from videos. However, these estimation-based theories can have acceptable results due to their affordabilities.

Ohri et al. in 2021 [5], demonstrated a way to build an application capable of doing realtime ondevice pose estimation and correction. they tested various deep learning based Blazepose pose estimation models and methods which can be used for pose correction.

Moryossef et al. in 2021 [6], evaluated two pose estimation systems and demonstrate their suitability (and limitations) for sign language recognition and their results shows that pose estimation tools suffer from shortcomings when body parts interact so pose estimation tools are not immediately applicable for the use in sign language recognition. Bahukhandi and Gupta in 2021 [7], use mediapipe estimation library for yoga pose classifying Logistic regression classifier achieves a maximum score of 94% among all classifiers. For classification a threshold value is used which is set at 97% below which no pose detected is given as output to the user.

Bashirov et al. in 2021 [8], combine the predictions of those estimators into temporally smooth human pose including face and hands articulations from RGBD images or videos in real time. They show that depth-based pose estimation still leads to considerable improvement in accuracy (and speed) compared to RGB-only state-of-the-art approaches.

In this report, we implement the MediPipe method for identifying certain human poses to order robots through using an intel realsense depth camera. we try to have a comparison between the estimated z value from the model and their real depth data from the depth camera.

3 Pose Estimation

3.1 Choosing Model

Pose estimation uses the pose and orientation of an entity to predict and track its location in an image/ video frame. A prediction outputs x,y coordinates and confidence values for each body keypoint detected. Based on [5] studing few factors which would be important while surveying and testing different models including accuracy, speed, memory and hardware, we We went with the BlazePose model as it aligned the most with our use case and showed great speed as well as accuracy.

BlazePose model generates 33 keypoints instead of the 17 keypoints generated by most models. Blazepose uses a two step pose estimation technique that first detects and then tracks the person. The detector detects the region of interest for pose estimation and then the tracker tracks the 33 landmarks. The pose detection uses BlazeFace to detect the human body. It also predicts the midpoint of a person's hips along with the radius of a circle circumscribing the whole person and the incline angle of the line connecting the shoulder and hip midpoints. This results in consistent tracking even for very complicated cases. The tracking model predicts the 33 keypoints with their x location, y location and their visibility [9].

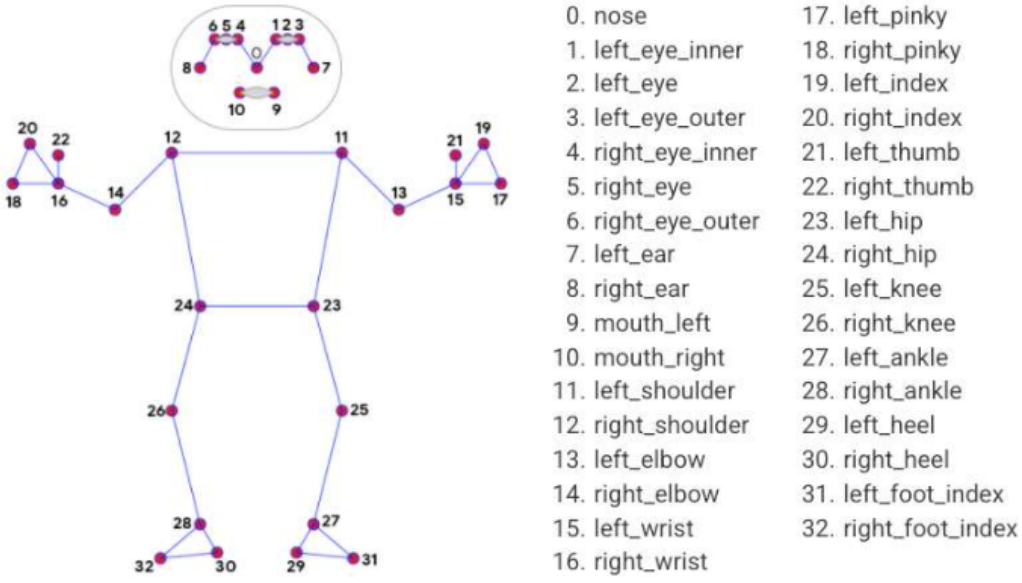


Figure 1: Pose landmarks in BlazePose model [9]

3.2 MediaPipe

MediaPipe provides a robust solution capable of predicting thirty-three 3D landmarks on a human body in real-time with high accuracy even on CPU. It utilizes a two-step machine learning pipeline, by using a detector it first localizes the person within the frame and then uses the pose landmarks detector to predict the landmarks within the region of interest.

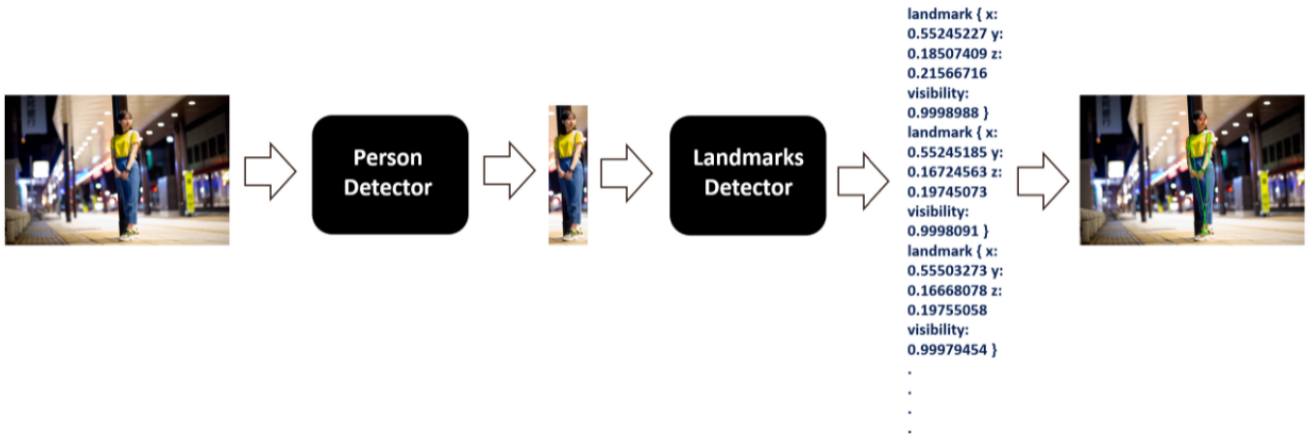


Figure 2: MediaPipe Algorithm

For the videos, the detector is used only for the very first frame and then the ROI is derived from the previous frame's pose landmarks using a tracking method. Also when the tracker loses track of the identify body pose presence in a frame, the detector is invoked again for the next frame which reduces the computation and latency.
The first thing that we need to do is initialize the pose class using the `mp.solutions.pose()` syntax and then we will call the setup function with the arguments:

- `static_image_mode` - It is a boolean value that is if set to `False`, the detector is only invoked as needed, that is in the very first frame or when the tracker loses track. If set to `True`, the person detector is invoked on every input image. So you should probably set this value to True when working with a bunch of unrelated images not videos. Its default value is `False`.

- `min_detection_confidence` - It is the minimum detection confidence with range (0.0 , 1.0) required to consider the person-detection model's prediction correct. Its default value is 0.5. This means if the detector has a prediction confidence of greater or equal to 50% then it will be considered as a positive detection.

- `min_tracking_confidence` - It is the minimum tracking confidence ([0.0, 1.0]) required to consider the landmark-tracking model's tracked pose landmarks valid. If the confidence is less than the set value then the detector is invoked again in the next frame/image, so increasing its value increases the robustness, but also increases the latency. Its default value is 0.5.

- `model_complexity` - It is the complexity of the pose landmark model. As there are three different models to choose from so the possible values are 0, 1, or 2. The higher the value, the more accurate the results are, but at the expense of higher latency. Its default value is 1.

- `smooth_landmarks` - It is a boolean value that is if set to True, pose landmarks across different frames are filtered to reduce noise. But only works when `static_image_mode` is also set to False. Its default value is True.

Then we will also initialize `mp.solutions.drawing_utils` class that will allow us to visualize the landmarks after detection, instead of using this, you can also use OpenCV to visualize the landmarks.

```
# Initializing mediapipe pose class.
mp_pose = mp.solutions.pose

# Setting up the Pose function.
pose = mp_pose.Pose(static_image_mode=True, min_detection_confidence=0.3, model_complexity=2)

# Initializing mediapipe drawing class, useful for annotation.
mp_drawing = mp.solutions.drawing_utils
```

- **Perform Pose Detection**

Now we will pass the image to the pose detection machine learning pipeline by using the function `mp.solutions.pose.Pose().process()`. But the pipeline expects the input images in RGB color format so first we will have to convert the sample image from BGR to RGB format using the function `cv2.cvtColor()` as OpenCV reads images in BGR format (instead of RGB).
After performing the pose detection, we will get a list of thirty-three landmarks representing the body joint locations of the prominent person in the image. Each landmark has:

- `x` : It is the landmark x-coordinate normalized to [0.0, 1.0] by the image width.
- `y` : It is the landmark y-coordinate normalized to [0.0, 1.0] by the image height.
- `z` : It is the landmark z-coordinate normalized to roughly the same scale as x. It represents the landmark depth with midpoint of hips being the origin, so the smaller the value of z, the closer the landmark is to the camera.
- `visibility` : It is a value with range [0.0, 1.0] representing the possibility of the landmark being visible (not occluded) in the image. This is a useful variable when deciding if you want to show a particular joint because it might be occluded or partially visible in the image.

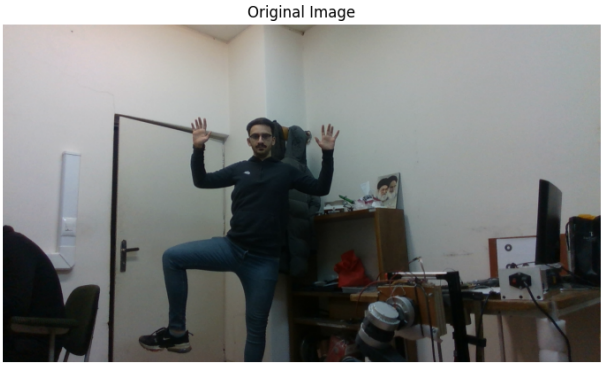
```
# Perform pose detection after converting the image into RGB format.
results = pose.process(cv2.cvtColor(sample_img, cv2.COLOR_BGR2RGB))

# Check if any landmarks are found.
if results.pose_landmarks:
```

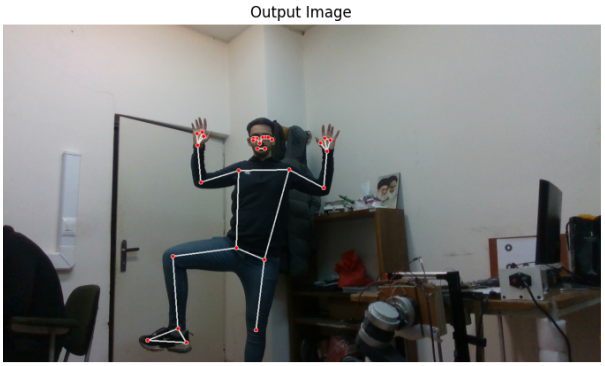
Then convert the normalized landmarks discused above into their original scale by using the width and height of the image.

```
# Retrieve the height and width of the sample image.
image_height, image_width, _ = sample_img.shape
```

Now we can draw the detected landmarks on the sample image using the function `mp.solutions.drawing_utils.draw_landmarks()` and display the resultant image using the matplotlib library.

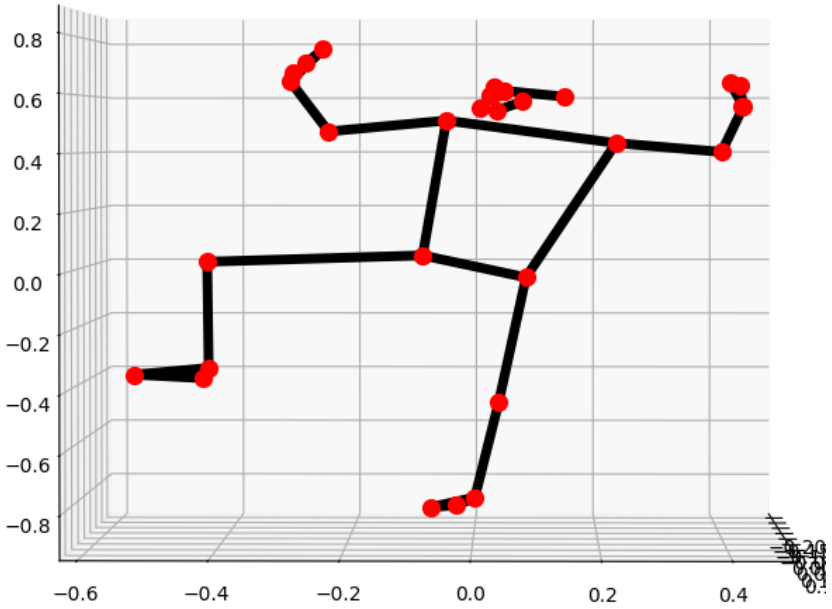
```
# Check if any landmarks are found.
if results.pose_landmarks:

    # Draw Pose landmarks on the sample image.
    mp_drawing.draw_landmarks(image=img_copy, landmark_list=results.pose_landmarks, connections=mp_pose.POSE_CONNECTIONS)

    # Specify a size of the figure.
    fig = plt.figure(figsize = [10, 10])

    # Display the output image with the landmarks drawn, also convert BGR to RGB for display.
    plt.title("Output");plt.axis('off');plt.imshow(img_copy[:,:,::-1]);plt.show()
```





Now visualize the landmarks in three-dimensions (3D) using the function `mp.solutions.drawing_utils.plot_landmarks()` . We will need the `POSE_WORLD_LANDMARKS` that is another list of pose landmarks in world coordinates that has the 3D coordinates in meters with the origin at the center between the hips of the person.

```
# Plot Pose landmarks in 3D.
mp_drawing.plot_landmarks(results.pose_world_landmarks, mp_pose.POSE_CONNECTIONS)
```



Note: As you see Although the left foot is not in the frame, models landmarks are completed in the 3D plot so Mediapipe can estimate all of landmarks and connections based on the hip landmarks. it will also verified again with using depth value instead of model estimated values.

3.3 Intel RealSense

For Capturing we use Intel RealSense D435i which is an USB-powered camera that includes wider field of view depth sensors and a RGB sensor. It is ideal for makers, educators, hardware prototyping and software development. The camera is designed for ease of setup and portability. Perfect for applications involving lots of motion – global shutter allows for this.

Figure 3: Intel RealSense d435 [10]

For Using this camera we should work with RealSense SDK, we write a class named `realsense_camera` returning color frame and depth frame.

Note: My specifice realsense_camera library is available open-source at https://github.com/SiaMahmoudi/MediaPipe-pose-estimation-using-intel-realsense-debth-camera/blob/31cd693f55c3093a227592f89e63c871e70746d7/realsense_camera.py

3.4 Using Real Depth Value

So we should align the depth frame and RGB frame together after making sure that the frame rate of both is the same. For 3D-plotting we should use MediaPipe for estimation x and y and realsense depth frame for z value. From `results.pose_landmarks.landmark.value` we can access to each landmarks x, y and z values. Hence we substitute depth value for model z value. Picture below shows the result.

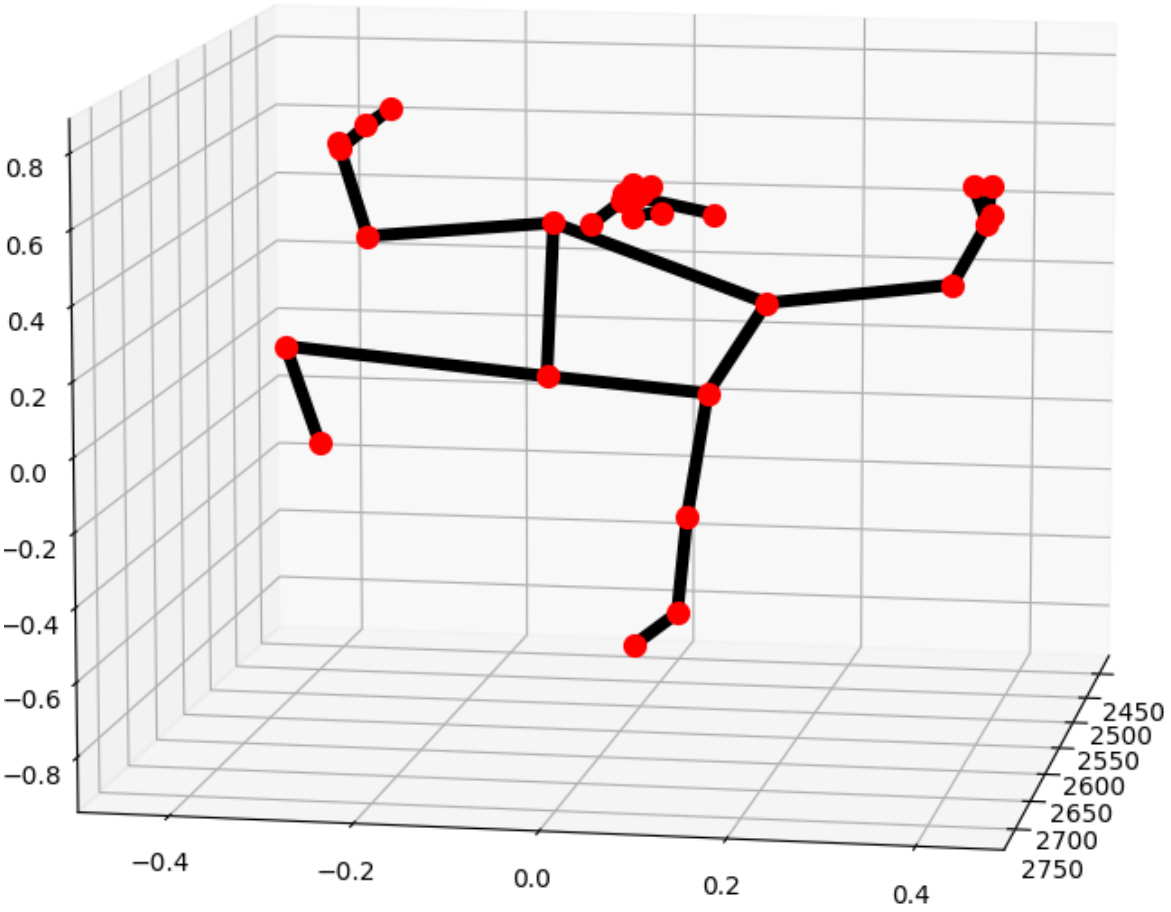


Figure 4: 3D plot of Pose detection using depth frame

As you see, both 3D plots are approximately the same except in landmarks whose location was not in the depth frame. On the z-axis, you can see the distance of the model from the camera which is around 2.5 meters.

By aligning these two frames we can also have the distance of each landmark from the Camera on the RBG picture, helping better perception as shown below.

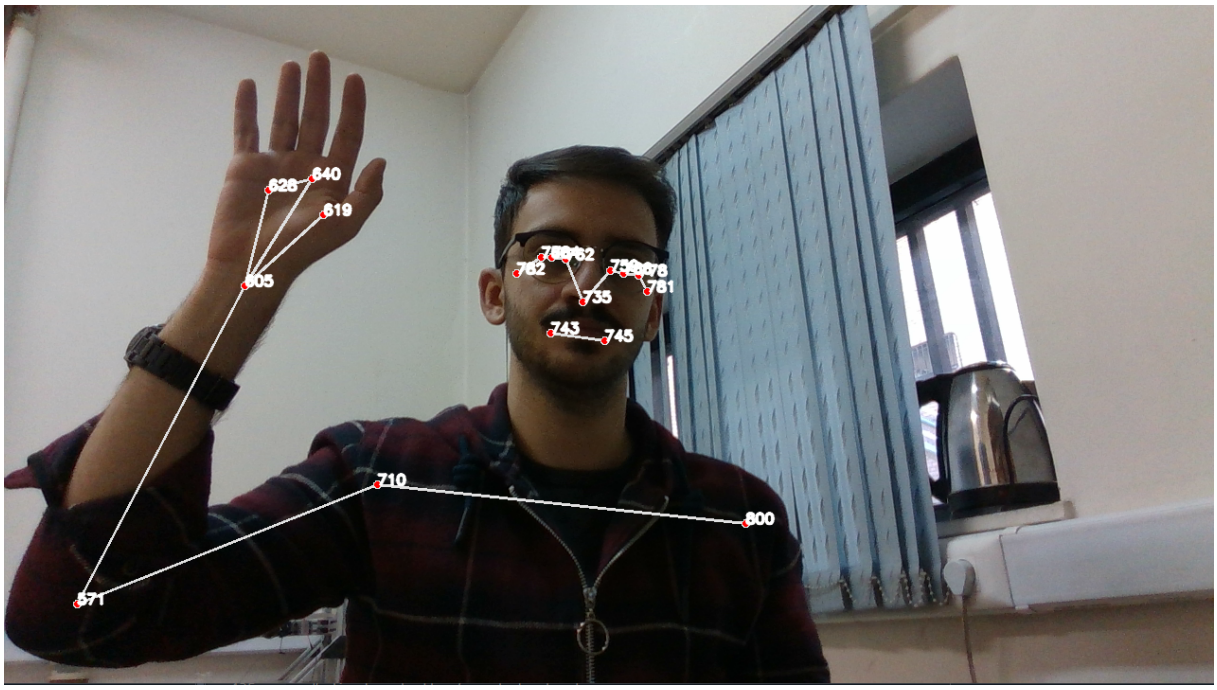


Figure 5: MediaPipe pose detevtion aligned with depth value for each landmark.

3.5 Pose Classification with Angle Heuristics and Landmarks depth

After we could perform pose detection, we level up our program by also classifying different poses using the calculated angles of various joints and their depth value. We will first detect the pose landmarks and then use them to compute angles between joints and depending upon those angles we will recognize the pose of the prominent person in an image.

But this approach does have a drawback that limits its use to a controlled environment, the calculated angles vary with the angle between the person and the camera. So the person needs to be facing the camera straight to get the best results.

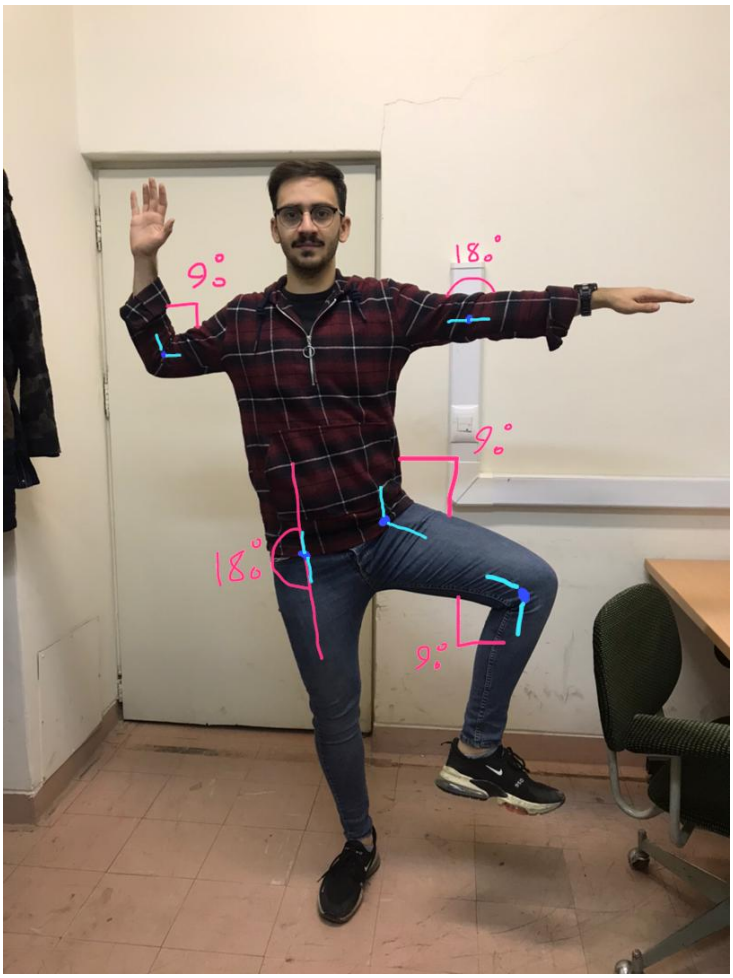


Figure 6: Different joints Angles of human body.

• **a Function to Calculate Angle between Landmarks**

we have created a function named `calculateAngle(landmark1, landmark2, landmark3)` that will be capable of calculating angles between three landmarks. The first point (landmark) is considered as the starting point of the first line, the second point (landmark) is considered as the ending point of the first line and the starting point of the second line, and the third point (landmark) is considered as the ending point of the second line.

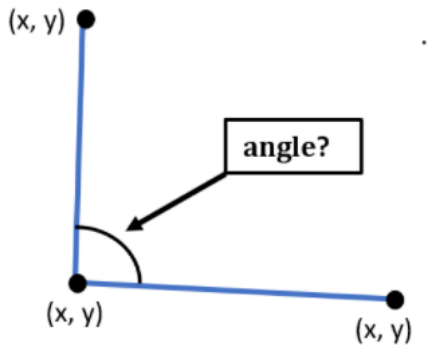


Figure 7: Find angle between two connection

• **a Function to Perform Pose Classification**

we have created a function that will be capable of classifying different poses using the calculated angles and depth of various joints named `classifyPose(landmarks, output_image, joints, depth_frame,display=False)` .

- **landmarks** : A list of detected landmarks of the person whose pose needs to be classified.
- **output_image** : A image of the person with the detected pose landmarks drawn.
- **joints** :The list of x, y and z of the landkmarks.
- **depth_frame** : depth_frame extracted from depth camera.
- **display** : A boolean value that is if set to true the function displays the resultant image with the pose label written on it and returns nothing.

Returns:

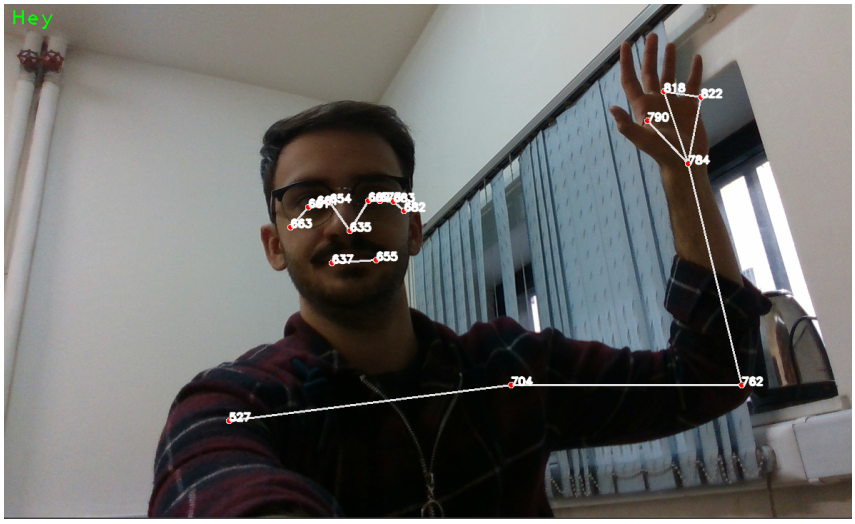
- **output_image** : a image with the detected pose landmarks drawn and pose label written.
- **label** : The classified pose label of the person in the output_image.
- **Time_Hey** : Set the time of the last frame of Hey pose.
- **Time_GO** : Set the time of the First frame of GO pose.

• **Poses**

As we want to create a dynamic movement for ordering robots, three pose has been considered:

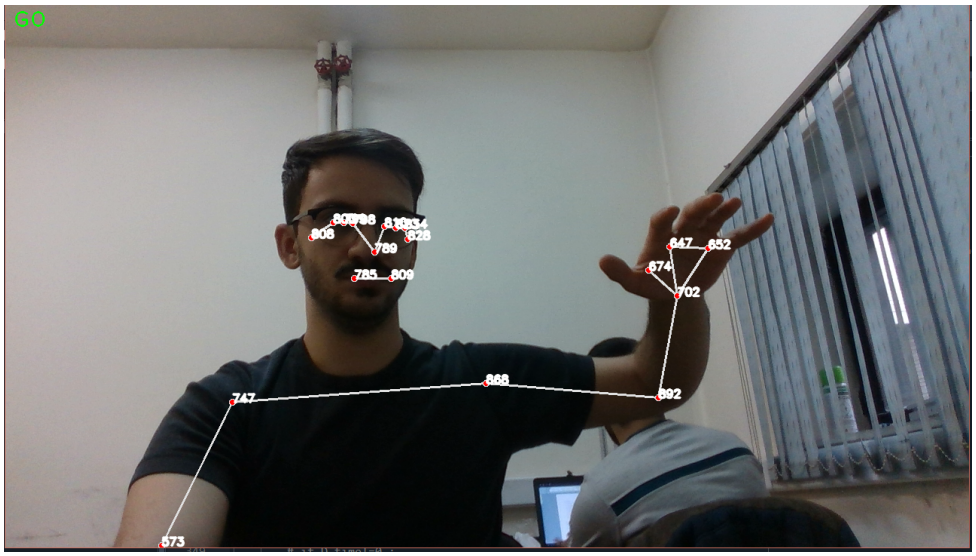
- **Hey Pose**

For this pose, the shoulder angle and elbow angle should be around 180 and 90 degree, respectively. Additionally, wrist and elbow joints must have approximately a same depth (vertically pose) as bellow figure.



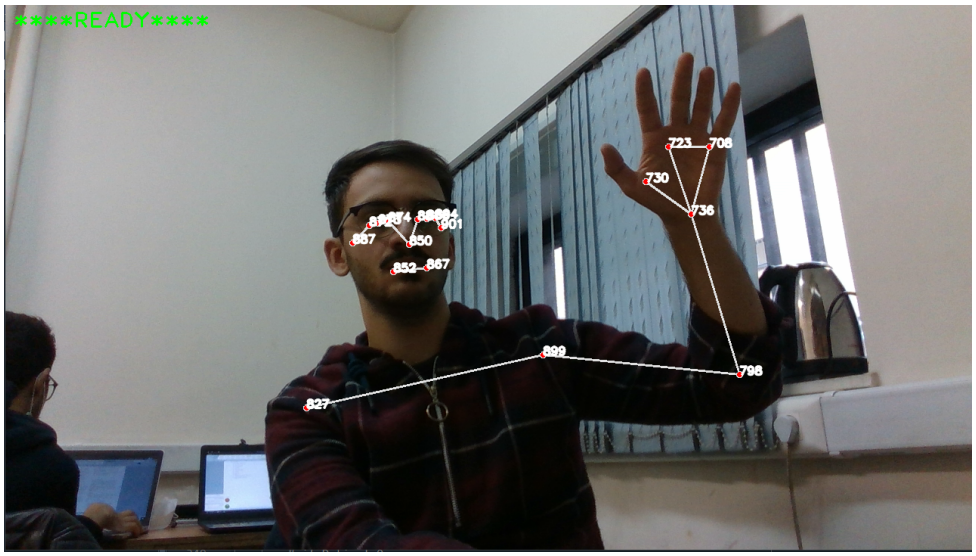
- **GO Pose**

For this pose, the shoulder angle and elbow angle should be around 180 and 90 degrees, respectively like the Hey pose. nevertheless, wrist and elbow joints do not have a same depth. naturally, the distance of elbow and wrist is around 300 mm so if the difference between wrist and elbow would be around 300 mm, the pose is changed to GO pose.



- **Ready Pose**

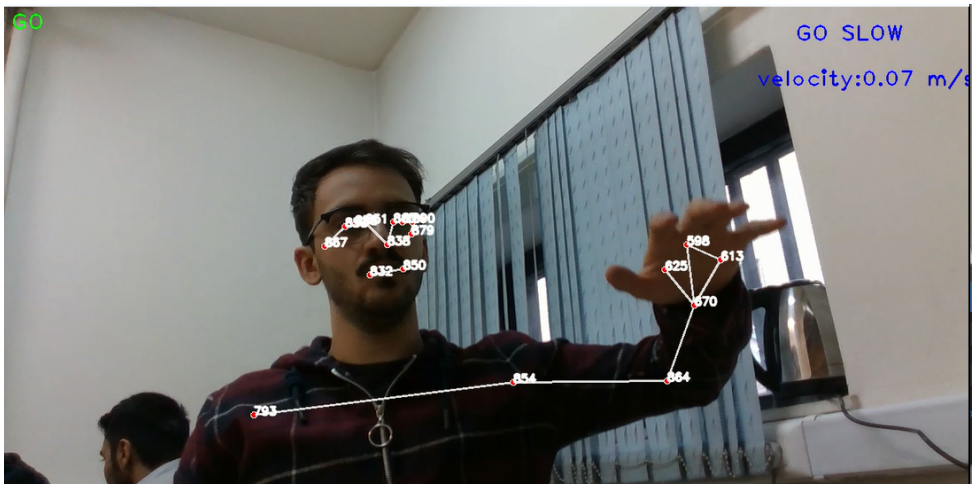
Lastly, a pose between Hey and Go poses would be ready. in this pose we just calculate time for the determination of the velocity and the acceleration of movements.

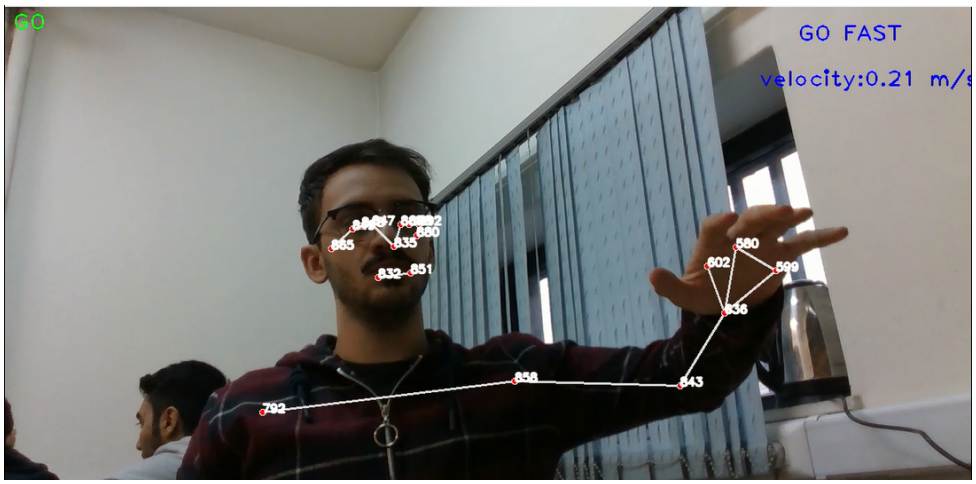


In the ready pose, we do not consider the depth, as this pose is somehow a transition between the other two main poses.

3.6 Velocity and Acceleration of Movements

With a combination of two or three poses, we can create movements. Therefore some concepts like velocity stand out helping classify our orders based on movements velocity. Currently, we have a movement including three poses (Hey, Ready, Go). So thanks to the velocity of the wrist from the Hey pose and Go pose or time of the Ready pose, we divide the movement into the GO FAST and GO SLOW .





Note: Final version of the project code is available open-source at https://github.com/SiaMahmoudi/MediaPipe-pose-estimation-using-intel-realsense-debth-camera/MediaPipe_Pose_detection_using_%20depth_camera.py

4 Summary and Future Work

In this project, we use MediaPipe pose estimator, an end-to-end pose tracking solution that achieves real-time performance and depth value of Intel RealSense d435 depth camera for human body pose estimation and classification. we combine depth value and the MediaPipe model resulting creation of some movements based on velocity and time. also, a comparison between model estimated z value and depth value that Importantly, we show depth-based pose estimation still leads to considerable improvement in accuracy (and speed) compared to RGB-only state-of-the-art approaches.
We currently working on a depth estimation of human body pose using MediaPipe and without a depth camera. Depth estimation can be addressed using deep neural networks trained in a fully supervised manner with the RGB image(s) as input and the estimated depth as output. As no dense depth information can be collected in the real world, a synthetic dataset from the depth camera has been utilized for training, which provided RGB images, depth maps, and semantic segmentation from stereo cameras. The architecture of the deep convolutional neural network was encoder-decoder with skip connections.
This feature has lots of applications in robotics, unmanned aerial vehicles, Self-Driving cars, and so on.

5 Refrences

[1] Osokin, Daniil. "Real-time 2d real-time pose estimation on CPU: Lightweight OpenPose." arXiv preprint arXiv:1811.12004 (2018).

[2] Kreiss, Sven, Lorenzo Bertoni, and Alexandre Alahi. "Pifpaf: Composite fields for human pose estimation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[3] Tensorflow Lite Pose Estimation. https://www.tensorflow.org/lite/models/pose_estimation/overview

[4] Tensorflow-JS. https://github.com/tensorflow/tfjs models/tree/master/posenet

[5] Ohri, Ashish, Shashank Agrawal, and Garima S. Chaudhary. "On-device Realtime Pose Estimation & Correction."

[6] Moryossef, Amit, et al. "Evaluating the Immediate Applicability of Pose Estimation for Sign Language Recognition." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

[7] Bahukhandi, Utkarsh, and Shikha Gupta. "YOGA POSE DETECTION AND CLASSIFICATION USING MACHINE LEARNING TECHNIQUES."

[8] Bashirov, Renat, et al. "Real-time RGBD-based Extended Body Pose Estimation." Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. 2021.

[9] Mediapipe Pose Documentation. https://google.github.io/mediapipe/solutions/pose.htm

[10] https://www.intelrealsense.com/depth-camera-d435i/ accessed: 1/26/2022