International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1691-1696 www.ijaem.net ISSN: 2395-5252

On-device Realtime Pose Estimation & Correction

Ashish Ohri, Shashank Agrawal, Garima S. Chaudhary

Submitted: 01-07-2021 Revised: 10-07-2021 Accepted: 13-07-2021

ABSTRACT
Human activity monitoring via pose estimation and correction 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 and assisted living. Our focus was on utilizing human pose estimation for fitness training. We created an AI trainer that estimates the human pose and corrects it while performing a workout routine. The focus of this software is to create an immersive home workout experience with an AI-assisted system. To achieve this, we reviewed different pose estimation models and pose correction algorithms. Our main aim was to create an application that runs in realtime on almost every device and gives real time pose correction commands.
Keywords: pose estimation, pose correction, angle correction, DTW, human activity recognition

I. INTRODUCTION

Human activity monitoring via pose estimation and correction has a lot of real world applications. Bad posture is common while sitting or walking. Incorrect technique can lead to injuries while working out. Pose estimation can be used for solving these problems and in many other fields like sports coaching (to monitor athlete's movements to a high degree of precision) and assisted living (fall detection for elder, differently-abled people). These use cases involve two parts: pose estimation and pose correction. Pose estimation can be done for single-person or multi-person. This project is focussed on single-person pose estimation.

Pose estimation predicts body keypoints of a person in each frame of a video feed. On the other hand, pose correction involves calculating the angles based on these keypoints and then doing pose correction for incorrect angles. Current state-of-the-art deep learning based pose estimation models perform really 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 on-device application which performs pose estimation and correction with high speed and accuracy while providing a smooth user experience.

Hence, we tested a few pose estimation models on factors like speed, accuracy, memory consumption, realtime on-device execution on two different setups. The models tested include Lightweight OpenPose [2], Pifpaf [1], Tensorflow-Lite(mobilenet) [7], TensorflowJS [8] (mobilenet, resnet) and Blazepose(via Mediapipe API) models.

We further dove into methods for pose correction techniques that are performed on top of the blazepose model. We used techniques for keypoint angle calculation and correction using Dynamic Time Warping (DTW) [10]. We also developed algorithms for managing/reducing the amount of pose correction prompts generated for a better user experience.

The paper is structured in the following manner: section 2 gives an overview of pose estimation, section 3 discusses techniques for pose correction and we give our concluding remarks in section 4.

II. POSE ESTIMATION

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. We tested a number of pose estimation models keeping in mind our requirements of accuracy, speed, low on-device memory consumption etc.

2.1. Testing Methodology

Since our goal was to find pose-estimation methods which are fast, accurate and work in real-time on the client device browser, we tested all these different implementations on our own general-purpose laptops, with the kind of specification which any other user's system would have.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1691-1696 www.ijaem.net ISSN: 2395-5252

Going with this approach also had another advantage that all methods are evaluated on the same setup. This is very important as the results (FPS metrics) which each paper mentions are typically on different specification setups depending on the use case which the authors intended for their implementation. For eg: Pifpaf is better suited for urban mobility tasks such as self-driving cars.

Influencing Factors : Our problem statement was to do real-time pose estimation and correction, so we came up with a few factors which would be important while surveying and testing different models.

2.1.1. Accuracy

Accuracy of pose keypoints depends on 2 factors: accurate x,y coordinate values and high confidence values of each keypoint. Along with fast estimation of pose keypoints it's important to have high accuracy too. This is because pose corrections are based on the body keypoints generated from pose estimation and to make sure that the user is getting accurate and useful correction instructions, the estimated keypoints should have high accuracy.

It's important to take into consideration the speed-accuracy tradeoff of different models depending on the type of activity being monitored.

2.1.2. Speed

Depending on the type of physical activity being analyzed, there might arise a need for quick real-time feedback on which the pose correction stage is dependent. As pose correction relies on pose estimation hence, it's important for the pose estimation model to give real-time body keypoint prediction so that the pose correction process can execute in real time too. Whereas, in some cases, quick real-time feedback might not be necessary (eg: Sitting Posture Correction) and slower model predictions wouldn't cause a problem. In such cases, we can use a model with higher accuracy and lower speed (speed-accuracy tradeoff).

2.1.3. On-device execution

There are numerous reasons for prioritizing on-device execution. Once the model is loaded onto the client browser, on-device execution would enable client-side processing. This means that all the processing is happening locally and no video data is being sent to any remote server which ensures privacy of the user. This is important because pose estimation involves monitoring the user's movements via webcam feed and the user might be uncomfortable if their video feed is leaving the device. Also, on-device execution means that after the model is initially loaded, there won't be dependence on the network speed.

2.1.4. Memory Consumption

Since all execution is happening on-device, hence, it's crucial that the pose estimation and correction processes don't hog too much RAM. Therefore, the size of the model being loaded on to the client device shouldn't be too large. Also, pose estimation and correction should occur in real-time (frame by frame) because storing data for all frames and making predictions after the activity has been performed would take up a lot of memory. Real-time prediction is also needed since for exercise pose correction, fall detection the feedback should be instant.

2.1.5. Hardware

Now since we've made on-device execution a priority that means that the hardware setup specifications should be enough to handle real-time execution without any lag in performance. Also, our main aim is to find methods for human activity monitoring which a decent section of the population can easily run without any additional hardware requirement. Also, a lot of pose-estimation models perform really well on CUDA-enabled hardware/ NVIDIA GPUs but many users may not have an Nvidia GPU as part of their regular setups and we don't want hardware to be a bottleneck. Models should be able to perform well solely on CPU too. So, we tested the models on 2 distinct setups.

2.2. Model Selection

We tested six pose estimation models on the two setups (as detailed in Table 1). The pose estimation models tested were: Lightweight OpenPose (Python) [2], Tflite (mobilenet v1) [7], Pifpaf [1], Tfjs (mobilenet v1) [8], Tfjs (Resnet 50) [8] and BlazePose [13]

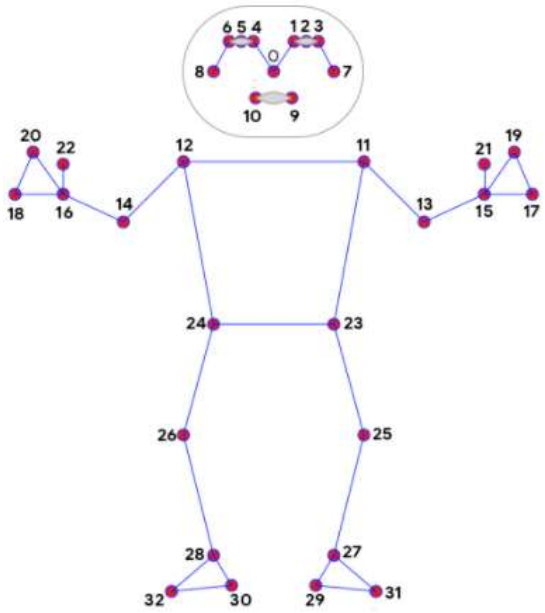
International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1691-1696 www.ijaem.net ISSN: 2395-5252

	Setup-1	Setup-2
CPU	Core i5 7th Gen	Core i7 9th Gen
Graphics	AMD R7 M445 4GB	NVIDIA Geforce GTX 1660 Ti, 6GB
RAM	DDR4 8 GB	DDR4 16 GB

Table 1: System Specifications of the setups used for model testing. NOTE: Setup-2 has dedicated graphics

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 (link). The tracking model predicts the 33 keypoints with their x location, y location and their visibility. It also detects the two alignment keypoints mentioned above.



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

Fig 1: Pose landmarks in BlazePose model [14]

III. POSE CORRECTION

The second part of our problem statement is to identify efficient methods for pose correction. Just like the pose estimation part depends on various factors like speed, accuracy, memory consumption etc., in the same way different factors have to be kept in mind while choosing pose correction methods. The mediapipe model is pretty fast when it comes to estimating the pose. However, when dealing with correcting the pose, there might arise for which we added a few algorithms to create a smooth user experience.

3.1 Type of Activity

Depending on the type of activity being monitored different methods for pose correction can be used. We came up with our own taxonomy/categories of activities based on factors such as frequency of movement in a frame and urgency (how quick the user needs correction feedback).

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1691-1696 www.ijaem.net ISSN: 2395-5252

3.1.1. Low-intensity Activity

Activities which involve minimal movement in a frame and don't have a sense of urgency associated with them. For eg: Sitting posture correction. These include activities that are not complex and either have few pose keypoints in observance or do not have continuously changing posture.

3.1.2. High-intensity Activity

Activities which involve a lot of quick movements or where there's a sense of urgency (to correct pose) can be categorized as high-intensity activities. For eg: workout pose monitoring [3], sports coaching, monitoring of old and differently-abled people.

In our case of workout correction, there are a lot of quick movements so it's important to capture data points from each frame to ensure that the pose analysis is correct. This is because incorrect workout techniques can lead to injuries so the user needs to be corrected immediately.

3.2. Methods

Pose correction involves correcting the user's pose. This is done by calculating the angles between desired keypoints. If for example we need to check whether the user is slouching while performing an exercise, we can calculate the angle between shoulder and hip line and hip and knee line. This can be done using either predefined corrections or matching techniques such as Euclidean matching or DTW [12]. For our pose correction methods we used eight keypoint angles in total (left and right inclusive). The angles included are shoulder elbow wrist angle, elbow shoulder hip angle, shoulder hip knee angle and hip knee ankle angle. These angles are calculated for both left and right side of the body hence making a total of eight angles.

3.2.1 Predefined Corrections

In this type of corrections, we use predefined angle values that need to be corrected. If we need to check whether the user shows a fault in an angle in observance, then we simply check the difference between the angle of the user's state and the correct angle that is predefined. This technique is mostly feasible in low intensity activities. Since this does not align with our software purpose, we used DTW for corrections instead of predefined corrections.

3.2.2 DTW

DTW or Dynamic time warping is a technique that checks for similarity between two temporal sequences [12]. DTW is used for time series classification [10] and also in gesture recognition [5]. A similar technique like DTW is Euclidean matching. However, DTW is considered better especially in this use case. The reason for this is that dynamic time warping compares a single index in an array with many others (one to many) whereas Euclidean matching compares a single index with another single index (one to one) [12].

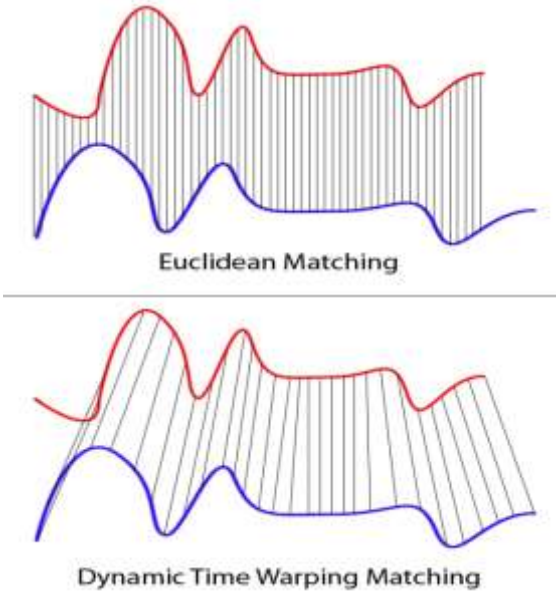


Fig 2: Difference between DTW and Euclidean [12].

International Journal of Advances in Engineering and Management (IJAEM)
Volume 3, Issue 7 July 2021, pp: 1691-1696 www.ijaem.net ISSN: 2395-5252

DTW is preferable to be used over predefined corrections in complex environments where defining angles is difficult or not feasible. This happens in environments where there are too many factors into play (high intensity activity) like playing a sport or working out [3]. In our case, we usually cannot directly focus on a single angle to check for correctness. Hence we select a correct angles array* and compare the users array for correctness by applying DTW on the two arrays.

*Correct angles array refers to array storing angles between keypoints obtained. Correct angles array fits this use case however is not a necessity and one may use any other property such as keypoint slopes while checking for correctness.

3.2.3 Time Threshold

If we have a speed of 100 FPS on a device and we raise errors for each frame, it would create a chaotic user experience. Even minor errors while performing a routine or errors that are due to the model inaccuracy would be introduced. To avoid such a situation we created a time threshold i.e. a quantitative value of time for which the error of the current frame must persist. Even though it reduces the FPS, it improves the performance significantly. The value of time threshold may be used as a variable, varying with the speed of the device. This value can be defined by taking into factor the speed of the model on the test device so that we do not create a too large or too small threshold. A threshold too large would miss most of the errors whereas a threshold too small would raise more frame errors.

A time array is used for each keypoint angle so as to raise an error only when the program is completely sure of its existence. If an error occurs in a specific keypoint angle during the routine and that error persists for a time greater than the time threshold then only would the voice command be used to correct the user. If the error is corrected before that, then the time array for that keypoint angle would be changed to null/initial state.

3.2.4 Joint Threshold

For each exercise, we also have predefined joints in consideration for each workout. For exercise such as bicep curl, we considered shoulder elbow wrist angle and elbow shoulder hip angle. Also, for the joints in consideration for each exercise, we also have a predefined threshold of error for each joint angle. If the user makes a mistake exceeding the threshold angle of the keypoint, then only will the user be informed of the error. This further enhances the user experience and removes unnecessary errors.

IV. CONCLUSION

In this paper, we demonstrate a way to build an application capable of doing realtime on-device pose estimation and correction without sacrificing a good user experience. We tested various state-of-the-art deep learning based pose estimation models and methods which can be used for pose correction. We demonstrated a way to manage and reduce the quantity of pose correction prompts for a better user experience.

We observed that there are a few models which give pose predictions with high accuracy and speed however, the choice of the model depends on the use case and factors such as speed-accuracy trade off, type of hardware, memory consumption etc.

We hope this project helps in the development of more real-world applications in the domain of human activity recognition and monitoring which prioritize on-device, real time execution.

REFERENCES

[1] 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.

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

[3] Chen, Steven, and Richard R. Yang. "Pose Trainer: correcting exercise posture using pose estimation." arXiv preprint arXiv:2006.11718 (2020).

[4] Dang, Qi, et al. "Deep learning based 2d human pose estimation: A survey." Tsinghua Science and Technology 24.6 (2019): 663-676.

[5] Schneider, Pascal, et al. "Gesture recognition in RGB videos using human body keypoints and dynamic time warping." Robot World Cup. Springer, Cham, 2019.

[6] Samkit Jain, Physio Pose: A virtual physiotherapy assistant. https://medium.com/@_samkitjain/physio-pose-a-virtual-physiotherapy-assistant-7d1c17db3159

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

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

[9] Ivan Kunyankin, Pose estimation and matching with TensorFlow lite PoseNet model https://medium.com/roonyx/pose-estimation-and-matching-with-tensorflow-lite-posenet-model-ea2e9249abbd

[10] Mark Regan, Time series Classification: KNN & DTW. https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping/blob/master/K_Nearest_Neighbor_Dynamic_Time_Warping.ipynb

[11] Sudharshan Chandra Babu, A 2019 guide to Human Pose Estimation with Deep Learning https://nanonets.com/blog/human-pose-estimation-2d-guide/

[12] Jeremy Zhang, Dynamic Time Warping: Explanation and Code Implementation. https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd

[13] Bazarevsky, Valentin, et al. "BlazePose: On-device Real-time Body Pose tracking." arXiv preprint arXiv:2006.10204 (2020).

[14] Mediapipe Pose Documentation. https://google.github.io/mediapipe/solutions/pose.html