Computer Vision Course Project: A Virtual Training Assistant

Rafik Hachana
Team Lead
r.hachana@innopolis.university

Fadi Younes
f.younes@innopolis.university

Hasan Khadra
h.khadra@innopolis.university

Kamil Sabbagh
k.sabbagh@innopolis.university

***Abstract*—We present a Virtual Training Assistant using Computer Vision and Human Pose Estimation. Our final solution counts the number of exercise repetitions that were performed on an exercise, can generalize to any exercise or repetitive movement pattern, and also provides the user with feedback about the speed of their exercise execution by using a reference video that should also be supplied by the user. We have also implemented an exercise classifier that does exercise recognition using the keypoints of the human pose, however its performance was not satisfactory due to the lack of training data.**

***Index Terms*—Computer Vision, Human Pose, Sport Exercise**

I. Introduction

In this report, we present our Virtual Training Assistant project. The main idea of this project is to use Computer Vision in order to analyze sport exercise videos, and play the role of a human trainer by giving the user feedback on their execution of the exercise. This is another case of employing automation in order to make repetitive tasks more efficient.

There exists multiple solutions to this task, from which we got some inspiration, some of them use hard-coded validations for specific, while others present a generic solution that can apply to any repetitive sport exercise.

We have defined the scope of the project to the task of counting the number of repetitions that the user executed on a given exercise, giving the user feedback on the speed of execution of the exercise by comparison to a user-provided reference video. And finally, we have also experimented with an exercise classifier that uses the evolution of the human pose in the video in order to identify the exercise. However, the latter was a tough challenge due to the severe lack of publicly available free datasets, therefore the results of the classifier were significantly subpar. More details will be presented in the subsequent sections.

We have included all of our implementation alongside a demo Jupyter notebook on our GitHub repo.

II. Related Work

The task of assessing sport performance or exercise performance has been tackled many times previously in both the academic literature, and by multiple companies, businesses, and online educators who put their ideas on the Internet. However, the approach that they take for the problem are different.

What is common between all of the solutions is their usage of a human pose estimation algorithm, which usually relies on a deep neural network that performs the pose estimation, as is the case in Ferreira's work [1]. Ferreira used machine learning further down the pipeline of exercise recognition, repetition counting and exercise validation, using synthetic video data. However, this option is not available to all cases, as it requires both data and resources for training. Most of the other approaches use hard-coded piplines instead. For instance, Moran et. al [2] used an LSTM network to classify the exercise using the sequence of keypoints that were generated, then use exercise-specific hard-coded repetition counters. On the other hand, Alatiah et al. [3] used a single frame classifier for the exercise classification: The keypoints of each frame are fed to a neural network, and this is done for all frames of the video. Then the final prediction is the prediction that was predicted the most. The same experiment also proposes a generic solution for repetition counting, using the minima and maxima of the keypoints time evolution, which is an idea that we adopted for our own solution.

For the human pose estimation part of the pipeline, the method is similar between different cases, the only difference is the used model architecture, or the used pretraine model. The most popular pretrained model for human pose estimation is the BlazePose model by Google, which is part of the MediaPipe library [4]. Many online blogs propose implementations to integrate it into an exercise evaluator, such as [5]. On the other hand, YOLOv7 is also a viable alternative [6]. However, MediaPipe is more popular as it is faster in inference, and does not need to use detection on every frame if it can track the keypoints movement [7].

III. Methodology

As mentioned in the introduction, the scope of our project encompasses 3 main features: The repetition counter, the exercise classifier, and the exercise evaluator. All of these features are based on the result of a human pose estimation model. Therefore, in this section we will tackle the features one by one, starting first with the human pose estimation which is the foundation of our project.

Introduction to Computer Vision course, Innopolis University, Fall semester 2022

A. Human Pose Estimation

Human pose estimation can be defined as the conversion of an image with humans in it to a set of keypoints that describe the position of the bodies of these humans. It is therefore a technique that will give us specific points of interest in the image that we can later use for analysis. In our case, we have looked into human pose estimation algorithms that are based on pre-trained deep learning models. These algorithms work by feeding the image into a neural network and then getting the positions of a predefined set of keypoints. The set of keypoints are predefined and they depend on the model used. The model output is a vector of detected keypoints. Depending on the model and method used, the keypoints can be 2-dimensional vectors, or 3-dimensional vectors in case of 3D pose estimation. In some cases, the pose estimation algorithm can detect multiple people in one image, while others are not capable of doing this.

In our case, we have decided to explore 2 solutions for human pose estimation in videos: The BlazePose model offered by Google through their MediaPipe Python library, and the YOLOv7 model that was trained on the COCO human pose dataset. A comparison of the keypoints used by both of them is presented in figure 1.

1) Human Pose Estimation using BlazePose: The first option we considered for pose estimation is the BlazePose model by Google. This model is optimized for human pose estimation in videos. It has 2 main components: A pose detector and a pose tracker. This 2-component architecture allows it to be faster when inferring the human pose evolution in a video: It starts first by detecting the pose. Once the pose is detected with a certain confidence value, it uses a pose tracker for the next frames instead of detecting the pose again for each new frame. In case the tracker loses the pose because of occlusion or an abrupt change in the video, the model will revert back to trying to detect the pose, this is what makes it the fastest model for quick pose inference from videos, also given the fact that it runs on the CPU, so no GPU is needed for inference.

The model expresses the pose in terms of 33 keypoints, illustrated in figure 1. Each keypoint (also called Landmark in the terminology of the mediapipe library) contains 4 values: a 3D vector expressing the estimated 3D pose of the keypoint, using the hip center of the person as the origin point, and a 4th scalar value expressing the visibility of the keypoint in the video, which can be used as a detection confidence value. The only disadvantage of this model is that it only works for a single person, so in case there multiple people in the video, it will only detect one, and hopefully it will keep that person in detection if using a suitable value for tracking confidence.

We have integrated the model in our solution using the MediaPipe Python library, through its simple API. The library offers 3 versions of the model: BlazePose GHUM Heavy, BlazePose GHUM Full, and BlazePose GHUM Lite, in descending order of complexity. Using a more complex variant yields better results but requires more resources. In our case, we have using the BlazePose GHUM Heavy variant, since we care more about the accuracy of the results than about the speed, as we are not doing real-time tracking.

2) Human Pose Estimation using YOLOv7: Another alternative that we have considered is the YOLOv7 Pose model, which was trained on the COCO dataset of human pose. This model uses a different set of keypoints compared to BlazePose, with only 17 keypoints illustrated in figure 1. This not a disadvantage for our case, since our current solution does not need to use the extra keypoints offered by BlazePose. YOLOv7 also outputs 3D keypoints. And it can run on either the CPU or the GPU. However, it does not contain a pose tracker, so in case of video processing, it will reuse its pose detection on every single frame of the video. YOLOv7 can also detect the poses of multiple people in the image easily. But since we are interested in analyzing exercise videos of a single person, we cannot really benefit from this advantage. The big problem with this model is its huge resource consumption, which made it unpractical to use for the final solution. More on this in the Evaluation section.

In order to integrate this model into our solution, we cloned the YOLOv7 GitHub repository and downloaded model weights corresponding to our task. We need to implement all the model loading and inference ourselves, since there is no available library API solution for this.

3) A model-agnostic solution: One of our initial design decisions was to make a solution that can work with any pose estimation algorithm. This can work for doing an analysis of human pose in sport exercises, since we usually only care about the keypoints that represent the main joints of the body (e.g. knees, shoulders, ...) and we are not really interested in extra keypoints that describe the fact for instance. Therefore, the rest of our implementation only uses 12 keypoints: 2 shoulders, 2 elbows, 2 wrists, 2 hips, 2 knees, and 2 ankles. In this way, the rest of the implementation is compatible with both of the models that we considered for experimentation, and can compatible with other pose estimation solutions without changing other parts of the code.

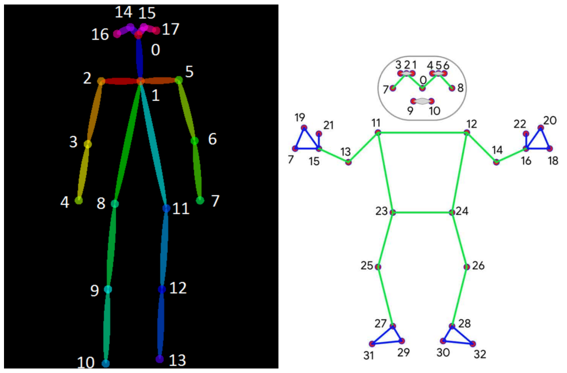

Fig. 1. Comparison between the human pose keypoints detected by the YOLOv7 model (left) and by the BlazePose model (right)

B. Repetitions Counter

Once we acquire the 12 keypoints of the human pose. We use multiple signal processing techniques to assess the repeating patterns in the pose over time, and therefore count how many exercise repetitions the person did. The advantage of our solution is that its exercise-agnostic, therefore it can work on any exercise, given that the exercise is a repeating movement pattern that involves at least some of the 12 keypoints that we used for the pose estimation. This is different than other solutions that are hardcoded for specific exercises, which perform better for the specific exercise, but will need a lot of extra implementation to support new exercises, and will need to be tuned manually to those exercises.

Our solution predicts the number of repetitions done in one exercise using the following pipeline:

1) **Keypoints to angles**: For each frame, we calculate 8 angles using the given keypoints: the 2 elbow angles, the 2 shoulder angles, the 2 hip angles, and the 2 knee angles. The angles are calculated using the formula described in equation 1. We use a clipped dot product of the keypoints vectors here. We have the option to either use 2D or 3D vectors here.

$$\alpha = \arccos(\min(\max(x.y, -1), 1)) \tag{1}$$

2) **Angles timeseries**: Once we calculate the angles for each frame, we aggregate the angles along the time dimension to get 8 different timeseries, which describe the evolution of the angle of the given joint during the video.
3) **Noise reduction**: We have noticed that the angles timeseries can be noisy, especially when using the full 3D keypoints to calculate the angles, or when trying to use a lower complexity model. To reduce noise, we have applied a Moving Average Filter to each of the angles timeseries. The filter's window covers one second, therefore the length of the window in terms of samples is calculated using the frame rate of the video.
4) **Extracting Maxima and Minima**: Once we get the smoothed signal, we extract the local extrema from it and use them as reference points for further processing. The local extrema also include the start and end frames of the timeseries.
5) **Counting the repetitions**: Using the extrema, we predict the number of repetitions. Using the absolute maximum and absolute minimum, we define the acceptable range of motion for the joint for each repetition to be 60% of the maximum vertical span of the timeseries. Therefore, we count one repetition for the user whenever they cross this range twice: once in each direction. This will protect us from accidentally counting half repetitions into the final count, and will further make the algorithm resistant against noise. After getting the rep count, we also calculate the average repetition length, and use it to discriminate repetitions that were too fast (at least 25% faster than the mean repetition length).
6) **Consolidating different repetition counts**: At this point, we have 8 different repetition counts, each one using one of the previous 8 angles timeseries. The last step is to define the final predicted repetition count. Our implementation gives a score to each prediction using the maximum angle range of its corresponding joint. This stems from the fact that the joint that moved the most are more descriptive of the exercise execution, while joints that moved less might have moved accidentally, and their movement is not a part of the exercise itself. Therefore, the score for each repetition count is the sum of angle ranges for the joints that yielded this repetition value. The final prediction is the count with the maximum score. The repetition counter also returns the smoothed data, the extrema it found, the maximum angle range for each joint, and also the average repetition duration for the final repetition count, which can be used further to asses the execution of the exercise.

C. Exercise Evaluator

After counting the repetitions, we can use the extra returned metrics, such as the angle ranges and the repetition duration, to make a simple exercise evaluator. The main idea of our exercise evaluator is that it will use a reference video for the exercise (for example the video of a professional trainer executing the exercise), and then it analyze it (extracting reps, ranges, and rep duration) to make a comparison between the reference video and the video given by the user. This method will also make our exercise evaluator general-purpose, as there are no parts of it that are hard-coded for a certain specific exercise.

The exercise evaluator implemented in our final solution is a simple Proof-Of-Concept implementation that only compares the speed of the exercise execution. We also aimed at using the angle ranges for the comparison, but it seems that even with the 3D pose estimation, even symmetric movements can have asymmetric angles, so the angle values are not accurate enough to do a comparison with the reference video.

D. The exercise classifier

We also wanted to further extend our solution to exercise identification. The final goal here was to identify the exercise executed by the user, and then picking the correct reference video for it from a predefined library of reference videos. We have considered 2 variants of the exercise classifier: A classifier that takes a single frame, and a classifier that takes a sequence of 50 frames. As we will discuss further in the evaluation section, the resultant classifiers were far from good enough to be included in the final solution, because of the lack of data.

1) Acquiring the data: The biggest challenge for training an exercise classifier model is finding a public and available exercise video dataset online. There are no big enough datasets. The best dataset we have found is a synthetic video dataset from Infinity AI. Since it is synthetically generated, it can be customized with different locations, angles, light conditions, etc. However, it is protected by a paywall and is not freely available to the public. The best alternative we have found is a user-submitted workout video dataset on Kaggle (here).

This dataset contains short video clips of a single repetition of a single exercise, it contains 8 exercise types. The dataset has 249 videos in total. We have used it to train our exercise classifiers, using the same exercise labels as the ones in the dataset.

2) The frame classifier: Our first model idea was the frame classifier, which is trained on single frames from an exercise video. In the inference stage, we run this model on all the frames of the video, and then take the most frequent prediction as the correct one for the entire video.

The model is simply constructed of 2 fully-connected neural layers, with a hyporbolic tangent activation function between them. The input of the model is the 3D pose of the frame using our chosen 12 keypoints.

In order to construct the dataset for this model, we have run our pose estimation algorithm (using BlazePose) on the videos of the Kaggle workout dataset. Then we have saved the keypoints of each frame of each video as a row in the dataset, along with the frame position, the video ID (to be used later for evaluating inference), and the exercise label. This resulted in a dataset of 23684 rows.

3) The frame sequence classifier: Our second model idea is an LSTM network that will take in multiple frames from a video and use them for classification. The model is constructed out of 2 LSTM layers, and 2 dense layers with a ReLU activation. The dense layers take the last hidden state of the 2nd LSTM layer as input.

The dataset for this model is contructed similarly to the frame classifier. We have predicted the pose keypoints for each video of the dataset, then we have calculated the joint angles that are of interest to us. Then we aggregated the angle evolution of all joints into 50 frames sequences. Some videos were dropped in this process and we ended up with only 216 sequences for training.

IV. Experiments and Evaluation

After implementing our solution, we need to further make decisions about the models and the parameters of the pose estimation. The results presented here guided us into optimizing our implementation.

A. Experimenting with the human pose estimators

Even though we have implemented our solution to be able to use both the BlazePose and YOLOv7 pose estimators. We were not able to use the YOLOv7 estimator in our final solution as it needed to many resources. On a local machine with 16GB RAM, the estimator was killed after analyzing 2 frames of the video because of high memory consumption. When running it on Google Colab, we could make it to 5 frames, but then the runtime crashes because of high memory usage.

We have tried to look for a solution to this problem. The problem is in a function implemented by the YOLO team that calculates non-maximum suppression in order to find the human pose keypoints. It seems that this function has a memory leak which keeps ramping up the memory usage the more predictions we make. We tried manually using the Python garbage collector here to optimize the memory, but the problem is still there. Therefore, we are only using BlazePose for the final solution with the RepCounter and Evaluator. However, we have included the code for the YOLOv7 pose estimation in our demo Jupyter notebook in case the user wants to try it, it will at least generate a video with a few frames containing the results of the pose estimation, even if the runtime crashes.

Furthermore, we have experimented with different parameters of the MediaPipe library when using the BlazePose estimator. Using the default parameters, we could not use the 3D keypoints for calculating joint angles, as the data along the 3rd dimension was too noisy. So we only used 2D keypoints in that case. We have later adjusted the model complexity from the default value to the highest value (therefore using the BlazePose GHUM heavy model instead of BlazePose GHUM full) and increased the minimum confidence threshold to use tracking instead of detecting for the human pose, which made angles with 3D keypoints less noisy. The final solution uses the 3D keypoints with the angles.

B. Tuning the Repetition Counter parameters

Our repetition counter contains multiple manual parameters that need tuning. We have tuned them for the best results on our set of evaluation sample videos. However, they might need to be tuned more for other use cases. We have tuned the window of the moving average filter, the required angle thresholds to register a repetition, the minimum repetition durations, and the minimum overall angle range of the given joint in order to consider it in the calculation for the repetitions estimation. The final values for the parameters are presented in the table I.

TABLE I
Final tuned parameters for the Repetition counter

Parameter name	**Value**
MA Filter window length	1 second
Rep upper threshold	(AbsoluteMax $-$ 0.25 $\times$ AbsoluteRange)
Rep lower threshold	(AbsoluteMin $-$ 0.25 $\times$ AbsoluteRange)
Minimum Rep Duration	(0.75 $\times$ MeanDuration)
Minimum Joint Angle Range	0.65 radians

C. Evaluating the exercise classifier

After constructing the data and training pipelines for both of the models. We have evaluated both of them: The single frame classifier was evaluated using the mode of predictions method that we explained in the methodology section, while for the sequence classifier, we have used the same evaluation as the one in the training pipeline, on the same data, as the data was not enough in the first place. The results are presented in table II. We can see the single frame classifier performed really poorly in both training and evaluation. The frame sequence classifier performed well during training. But it looks like it was overfitting: It could not generalize to our separate sample of testing videos for the overall solution. All the predictions it gave were wrong, therefore it seems like it was just overfitting to the training data. Which is a problem we cannot mitigate since we don't have enough data. Our final decision was to drop the exercise classifier from the final solution.

TABLE II
EXERCISE CLASSIFIERS EVALUATION RESULTS

Model	Training Acc	Eval Acc
SingleFrameClassifier	0.378	0.321
FrameSequenceClassifier	0.689	0.689

D. Evaluating the whole solution

The final solution was evaluated manually on a set of sample exercise videos that we have acquired from a stock footage website. After tuning the parameters of the repetition counter, we got correct results on most of the videos, except one case where the repetitions were too fast, and one case where the program counted one extra repetitions in the Deadlift exercise, because it also counted when the person did the movement without lifting the barbell.

For the other cases, we got the correct predictions for the number of repetitions. An example of the used joint angles timeseries used to count the reps is illustrated in figure 2.

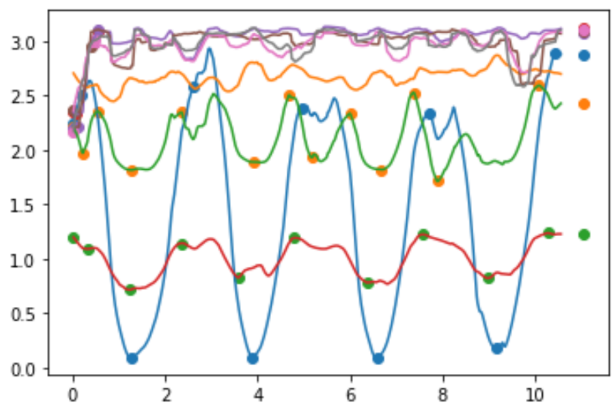

Fig. 2. The processed joint angles evolution that is used by the repetition counter (after applying the MA filter), with the extrema points

V. Analysis and Observations

After experimenting with the different aspects of our solution and picking the best working set of features. We have made several observations and conclusions about the limitations of our solution, and how it can be improved:

A. The manual tuning problem

Since we are tuning the repetition counter parameters manually, it can stop working for extreme cases (reps too fast). This manual tuning could be fixed in the future by detecting this case, or by using deep learning to assist the process if enough data is available.

B. Classifier failure

We have experimented with the exercise but it did not work due to the lack of data. One potential solution here is to use synthetic data.

C. The 3rd dimension issue

Even when using the highest complexity of the BlazePose model, the 3rd dimension is still noisy and not very reliable. If it was detecting depth correctly, it would give the same angles for joints on symmetric exercises. This limits our current ability to compare angle ranges to the reference video for feedback, simply because the absolute values of the angles are not comparable. Because of this issue, we are currently only using the shape of the timeseries (period, number of extrema) instead of the exact values.

D. The performance issue

Even with the fast BlazePose model, the pose estimation takes at least twice the duration of the video. This could be improved by trading off some quality by dropping some frames completely. But if we want to run the pose estimation on all frames, we definitely cannot extend our solution to a real-time solution.

E. The limitations of only tracking the human body

This is also another observation that we noticed, and that extends to all the virtual training solutions that we researched: Only tracking the human body might not be accurate in some cases, especially in weightlifting exercises. For instance, if a person performs deadlifts (the exercise of picking a barbell from the ground), then does the movement pattern one more time without actually lifting the bar, that will be counted as one repetition. This problem can be solved by extending the current pose estimation solutions to also track the movement of the barbell, or whatever sport equipment is being moved around in the exercise. This is definitely a big challenge, as there are no annotated datasets, and there are not enough video datasets online in the first place.

VI. Conclusion

In this report, we have described our Virtual Training Assistant project as well as the process of developing it. We went over the existing solutions, our 3 implemented components that are based on human pose estimation: repetition counting, exercise evaluation, and exercise classification. According to the results, the only reliable parts that can be offered to the end-user are the repetition counter, and the exercise evaluator, since we did not have enough data to train the classifier. Therefore, we can safely say that we have achieved a minimum viable product for our project.

This work can be extended further by involving more deep learning approaches to avoid manual tuning of parameters, by optimizing the performance more to adapt to real-time requirements in some cases, and by extending the pose estimation to also account for the used sport equipment, which requires more data to train on.

References

[1] B. Ferreira, P. M. Ferreira, G. Pinheiro, N. Figueiredo, F. Carvalho, P. Menezes, and J. Batista, "Deep learning approaches for workout repetition counting and validation," *Pattern Recognition Letters*, vol. 151, pp. 259–266, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786552100324X

[2] A. Moran, B. Gebka, J. Goldshteyn, A. Beyer, N. Johnson, and A. Neuwirth, "Muscle Vision: Real Time Keypoint Based Pose Classification of Physical Exercises," p. 9.

[3] T. Alatiah and C. Chen, "Recognizing Exercises and Counting Repetitions in Real Time," May 2020, arXiv:2005.03194 [cs]. [Online]. Available: http://arxiv.org/abs/2005.03194

[4] "Pose." [Online]. Available: https://google.github.io/mediapipe/solutions/pose.html

[5] A. Fortes, "Deep Learning based Human Pose Estimation using OpenCV and MediaPipe," Jun. 2021. [Online]. Available: https://medium.com/nerd-for-tech/deep-learning-based-human-pose-estimation-using-opencv-and-mediapipe-d0be7a834076

[6] "Pose Estimation/Keypoint Detection with YOLOv7 in Python," Sep. 2022. [Online]. Available: https://stackabuse.com/pose-estimation-and-keypoint-detection-with-yolov7-in-python/

[7] "YOLOv7 Pose vs MediaPipe in Human Pose Estimation." [Online]. Available: https://learnopencv.com/yolov7-pose-vs-mediapipe-in-human-pose-estimation/