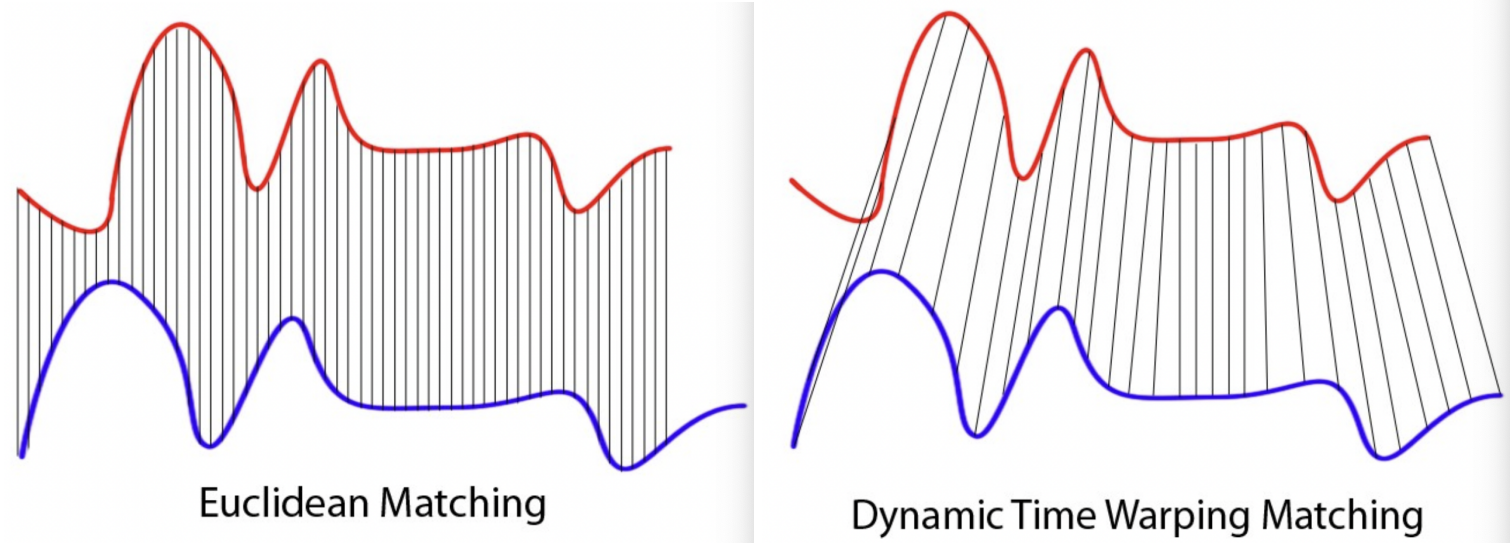
Algorithms to Compare Fitness Videos using DTW

What is DTW?

DTW or Dynamic Time Warping is used to compare the similarity to calculate the distance between two arrays or time series of different length.

Suppose we want to compare 2 arrays of equal lengths. One obvious way is to match them up in a one-to-one fashion, component-wise and sum up the total distance of each component. But there is no way to use this methodology to compare two arrays of unequal length. This is where DTW comes in. Just as the name indicates, it wraps the series so that they can match up, by ignoring the one-to-one mapping and employing many-to-one and one-to-many strategy, so that the total distance can be minimized between the 2 series.

Consider the following example:



Clearly these two series follow the same pattern, but the blue curve is longer than the red. If we apply the one-to-one match, shown on the **left**, the mapping is not perfectly synced up and the tail of the blue curve is being left out.

DTW overcomes the issue by developing a one-to-many match so that the troughs and peaks with the same pattern are perfectly matched, and there is no left out for both curves as shown in the **right** figure.

To read more about DTW algorithms follow this link.

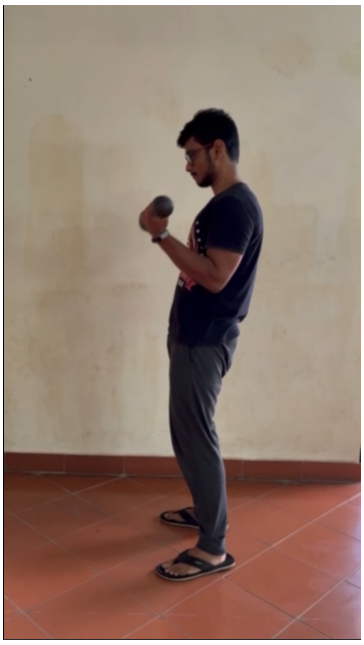
How is DTW useful for our project?

Our goal is to compare two videos, i.e. trainer and trainee to come up with a score, to represent how similar the trainee video is to the expert, i.e. the trainer. Since videos are just collections of frames, as a function of time, we can think of them as time-series datas. So, multi-dimensional DTW seems like a very natural solution to this.

Implementation

Data

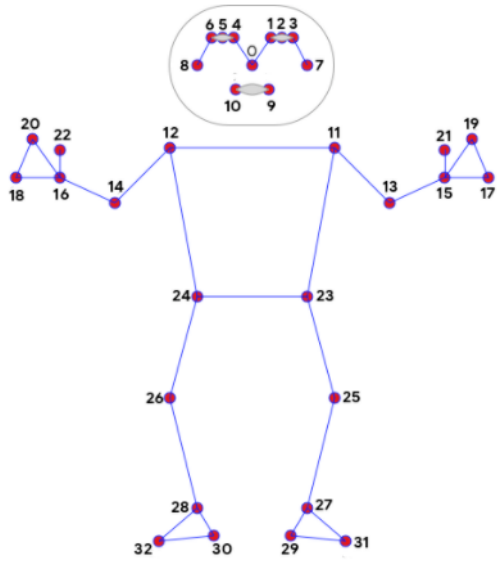
- Used 3 different videos of subjects doing bicep curls for preliminary comparison:
  - Video 1: Trainer Video
  - Video 2: Trainee Video with correct form
  - Video 3: Trainee Video with wrong form



- We use these videos as a baseline to test out the preliminary results of DTW. Post visual inspection, it is evident that Video 2 is more similar to Video 1 than Video 3, which is what the scores will end up showing.

Pre-Processing

- We use Mediapipe Blazepose to extract 33-key points coordinates in the x,y and z axis, which is normalized w.r.t. the size of the frame.



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

- Suppose we compare 2 videos, each containing n1 and n2 frames respectively.
- We extract the coordinates of 33 key-points in each frame into (n1, 33, 3) dimensional array for the first video and (n2, 33, 3) dimensional array for the second video. Now, each frame contains a (33, 3) dimensional array and we use DTW to compare these 2 time-series data.
- We do further preprocessing based on the algorithm we are trying to use, which we will be describing next. Here is how the key-points were detected in our test videos:

Algorithms

For DTW, we use the dtw_ndim.distance from the dtaidistance library.

We have tried out 2 algorithms to test our claims:

- **Algorithm 1.1**: We first reshape the (n, 33, 3) dimensional array to (33, n, 3) dimensional array. Then we loop over each of the 33 key-points individually. Let's say we are focussing on key-point **0**, which corresponds to the **nose**. We consider the (n1, 3) and (n2, 3) dimensional arrays corresponding to this key-point in the first and second video respectively and use DTW to obtain a score corresponding to each key-point. We obtain 33 such scores and consider the average score out of all.
- **Algorithm 1.2**: This is a modification of **Algorithm 1.1**. We use our domain expertise about the exercise, i.e. bicep curls and argue that only the key-points corresponding to the upper body matters. Hence we limit our observations to only the key-points in the upper body and only consider (n, k , 3) dimensional array, where **'k'** is the no. of key-points considered ($k \leq 33$). Everything else is exactly the same as before, and we end up considering the average score of these 'k' key-points.
- **Algorithm 2.1**: Instead of reshaping the array as in **Algorithm 1.1, 1.2**, we now consider the whole (n, (33,3)) array of each video for comparison. So, each frame contains information of all the key-points in a (33, 3) dimensional array. We use DTW on this and consider the score.
- **Algorithm 2.2**: This is again a modification of **Algorithm 2.1** and **Algorithm 1.2**, where we limit ourselves to the relevant key-points (in this case the upper-body) and directly compare the (n, (k, 3)) dimensional arrays of each of the videos to obtain the score. So, each frame contains information of only the relevant 'k' key-points (of the upper-body) in the (k, 33) dimensional array. We use DTW on this to consider the score.

Results

- These are the scores we obtained after comparing the videos using all the algorithms:

	Video_1 vs Video_1	Video_1 vs Video_2	Video_1 vs Video_3
Algorithm 1.1	100.0	89.4	76.7
Algorithm 1.2	100.0	88.5	76.4
Algorithm 2.1	100.0	85.6	72.1
Algorithm 2.2	100.0	83.7	70.6
Average Scores	100.0	86.8	73.95

Observations

- First comparison acts as a sanity check, comparing the same video with itself gives us a score of 100 every time.
- As expected from the beginning, Video 2 is more similar to Video 1 than Video 3, which is also reflected by the score.

DTW: Normalizing the subject before comparison

Why do we need normalization?

These are some of the problems that we might face while comparing exercise videos:

- We might compare the videos of subjects of different sizes (tall vs short, bulked vs lean).
- Another problem might be caused by the difference in zooms on the subjects in the videos. A person might appear big in a frame, which has zoomed in onto the subject, whereas the same person will appear small when zoomed out.
- Another problem might be the position of the subject with respect to the frame. For instance, we might have a subject in the top left corner of the frame, and another person in the bottom right corner of the frame.

Even though they might perform the same action, using the raw coordinates to compare using DTW will result in inaccurate scores. We would ideally want our model to be robust to these differences and output a score purely based on the actions performed by the subjects. This is where normalization comes into play. We normalize the 3D-coordinates obtained via Mediapipe Blazepose by squeezing them into a unit cube (with relevant zooming and translation of the coordinate axes) and then comparing them into DTW. This makes the comparison fair.

Example:

Notice how the subject is performing the same exercise in both frames. The only difference being that the first frame has zoomed in on the subject, whereas that is not the case in the second. But Mediapipe Blazepose will output a set of coordinates which are very different from each other, which when used to compare using DTW will give us a very low score. But won't be a representative comparison between these 2 videos. This is where normalization will help us even the playing field.

Implementation

Data

- We use the same 3 videos as before for comparison.

Pre-Processing (Cube Normalization)

- As in the previous Algorithms, we again use Mediapipe blazepose to extract 33-key points coordinates in the x,y and z axis, which is normalized w.r.t. the size of the frame. But this time we go one step further and implement the "***Cube Normalization***".
- **Cube Normalization**:
  - For each frame, first we extract the (x_{min}, y_{min}, z_{min}) and (x_{max}, y_{max}, z_{max}) which are the **min** and **max** coordinates of each of the components, amongst all the 33 key-point coordinates in a particular frame.
  - We use these min-max coordinates to transform the coordinates of the 33 key-points using the following equation:
    - $$(x, y, z) = (\frac{x - x_{\min}}{x_{\max} - x_{\min}}, \frac{y - y_{\min}}{y_{\max} - y_{\min}}, \frac{z - z_{\min}}{z_{\max} - z_{\min}})$$
  - After this transformation, we only focus on the key-points of the subject, which makes the background obsolete. This also manages to squeeze all the coordinates into a unit cube, with the origin translated to the (x_{min}, y_{min}, z_{min}).

Example:

These are visualizations of how cube normalization works. First we encapsulate the subject via a cuboid obtained by extracting the extreme coordinates. Then we squeeze this cuboid via a unit cube, which in turn makes the comparison even fairer.

Algorithms

Again for DTW, we use the dtw_ndim.distance from the dtaidistance library.

We again try this out with 2 algorithms, following the same format as before (for detailed description read about **Algorithm 1.1, 1.2, 2.1, 2.2**). The only difference being, we perform the cube normalization first:

- **Algorithm 3.1**: Cube normalization and **Algorithm 1.1**.
- **Algorithm 3.2**: Cube normalization and **Algorithm 1.2**.
- **Algorithm 4.1**: Cube normalization and **Algorithm 2.1**.
- **Algorithm 4.2**: Cube normalization and **Algorithm 2.2**.

Results

- These are the scores we obtained after comparing the videos using all the algorithms:

	Video_1 vs Video_1	Video_1 vs Video_2	Video_1 vs Video_3
Algorithm 3.1	100.0	81.8	71.0
Algorithm 3.2	100.0	82.1	71.7
Algorithm 4.1	100.0	80.7	67.3
Algorithm 4.2	100.0	77.5	61.6
Average Scores	100.0	80.53	67.9

Observations

- As expected from the beginning, Video 2 is more similar to Video 1 than Video 3, which is also reflected by the score.
- In this algorithm the scoring seems to have become stricter, as the average scores seem to have dropped in both the comparisons.

DTW: Adding Sensitivity hyperparameter to control scoring

Why do we need the sensitivity hyperparameter?

We need sensitivity hyperparameters to control how strict or lenient we would want our model to be. In our implementation:

- Higher Sensitivity => Lenient Model => Higher Average Score
- Lower Sensitivity => Strict Model => Lower Average Score
- Default Sensitivity = 1

Implementation

Data

- We use the same 3 videos as before for comparison.

Pre-Processing (Adding Sensitivity hyperparameter)

- We keep everything the same as seen in Cube Normalization.
- We only tweak the `compare_vid` function (this takes in the arrays of coordinates extracted from the video and uses DTW to output a score) to add an extra parameter called `sensitivity` (with default value = 1), which is divided in the numerator along with the norm of the array of coordinates from the video extracted using the `extract_arr` function.
- This enables us to control the strictness of the model.

Algorithm

- **Algorithm 5**: These remain exactly the same as Cube Normalization aka **Algorithm 3.1.**
- **Algorithm 6**: These remain exactly the same as Cube Normalization aka **Algorithm 4.1.**

Result

- These are the scores we obtained after comparing the videos using all the algorithms at different sensitivity values, i.e. at **0.5**, **1**, **1.5**, **2** and **5**:
-

Algorithm 5: Sensitivity	Video_1 vs Video_2	Video_1 vs Video_3
0.5	63.7	42.0
1 (Default)	81.8	71.0
1.5	87.9	80.6
2	90.9	85.5
5	96.3	94.2

-

Algorithm 6: Sensitivity	Video_1 vs Video_2	Video_1 vs Video_3
0.5	61.3	34.6
1 (Default)	80.7	67.3
1.5	87.1	78.2
2	90.3	83.6
5	96.1	93.4

Observations

- As expected and observed before, Video 2 is more similar to Video 1 than Video 3.
- As inferred from the scores in the tables above, the sensitivity of **1.5** seems to suit us the most (because of the relatively large difference in score, with the right amount of strictness).

DTW: Angle Invariance with Euler's Rotation Matrix

What is Angle Invariance?

In the context of Fitness Activity Recognition, Angle Invariance would mean that the algorithm works irrespective of the camera angles of the videos that we are comparing. For example, suppose we are comparing two videos with people doing bicep curls. In the first video, we can see the left half of the subject and in the second, we can see the front view of the subject. But we would ideally like to compare these two videos and score post comparison.

Why do we need Angle Invariance?

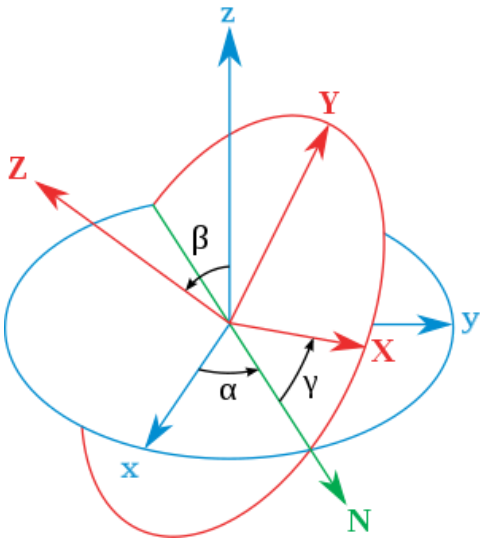
We need Angle Invariance to make the model more robust, to enable comparisons between trainer and the trainee, even when the angle of the trainee doing the exercise is different to the trainer video. Simply applying DTW without preprocessing would result in an unfair comparison and hence output a meaningless score.

What is Euler's Rotation Matrix?

In linear algebra, a **rotation matrix** is a transformation matrix that is used to perform a rotation in Euclidean space.

We use the notion of basic rotation (also called elemental rotation) is a rotation about one of the axes of a coordinate system. The following three basic rotation matrices rotate vectors by an angle θ about the x-, y-, or z-axis, in three dimensions, using the right-hand rule—which codifies their alternating signs. The same matrices can also represent a clockwise rotation of the axes. Here are the 3 rotation matrices to rotate the coordinates w.r.t. the corresponding axes:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Implementation

Data

- We use the same 3 videos as before along with 3 new videos for comparison.
- Here is the description of the new videos:
  - Video F: Bicep Curls Front View
  - Video L: Bicep Curls Left View
  - Video R: Bicep Curls Right View



- Even though the camera angles view of the subject is different in each of these videos, we would ideally want the model to output a high score, when compared with each other (because they are performing the same exercise).

Pre-Processing (Using Euler's Rotation Matrix)

- We use the following (mentioned below) rotation matrix to rotate the set of coordinates of the trainee video about the y-axis and compare them with the trainer video to obtain a score using the algorithms above:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

- For any fixed angle θ, we follow the following pipeline:
  - For angles $\alpha \in \{\theta, 2\theta, \dots k\theta\}$, where $k = [\frac{2\pi}{\theta}]$.
  - We find the rotation matrix corresponding to α, i.e. $R_y(\alpha)$.
  - We then extract the coordinates using Mediapipe Blazepose, and rotate the coordinates of each key-point in each of the frame by an angle of α, in counter-clockwise direction, using the rotation matrix $R_y(\alpha)$, by following the equation: $(x, y, z)^T \mapsto R_y(\alpha)(x, y, z)^T$.
  - After doing this for every frame in the video, we compare them as previously done using DTW and store the score obtained corresponding to this rotation.
  - After doing this for each α, we output the best score out of the lot.

Algorithm

- **Algorithm 7:** The base algorithm remains similar to **Algorithm 5** as described above.

Limitations

- Since this requires comparing multiple (rotated versions) of the trainee video to the trainer video, it takes more time to compute the scores.
- Smaller the angle θ, larger the no. of comparisons required, larger the time required
- Due to the confidence limitations of certain coordinates (with lower visibility in the frame), Mediapipe Blazepose approximates the coordinates, which after rotation may not represent the relative positions of coordinates in real world scenarios, which leads to unfair comparisons at times.

Result

- We first observe the results by comparing these videos using the algorithms that do not incorporate any form of Angle Invariance:

	Front vs Left	Left vs Right	Right vs Front
Algorithm 5	47.6	38.3	52.1
Algorithm 6	43.1	22.8	43.5

- Now, we check the comparison results at different values of θ, i.e. for $\theta \in \{ \frac{\pi}{2}, \frac{\pi}{4}, \frac{\pi}{6}, \frac{\pi}{8} \}$:

Angles (degrees)	Front vs Left	Left vs Right	Right vs Front
90	54.0	60.4	52.2
45	60.3	60.4	60.4
30	58.9	60.4	59.7
22.5	60.3	60.4	40.4
Max Score	60.3	60.4	60.4

Observations

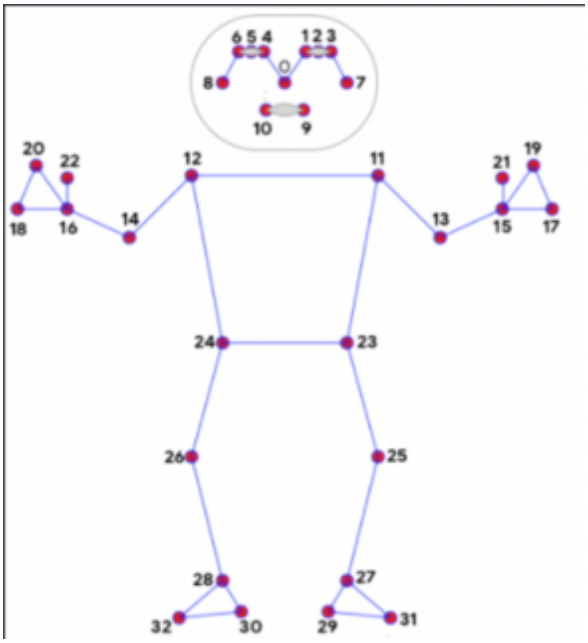
- We get a clear improvement in scores as obtained from the naive comparison (without rotating the coordinates) than after rotation as observed from the table.

DTW: Angle Invariance by comparing Joint Angles

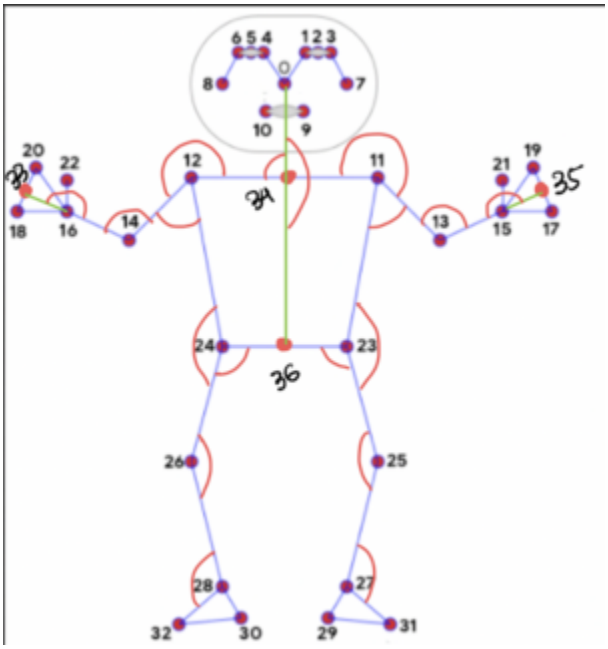
What is Angle Invariance by comparing Joint Angles?

We now move on to our best set of algorithms, which inherently takes care of the problem of angle variance. In this algorithm, we identify and consider various relevant joint angles, which are obtained by using the key-point coordinates extracted by Mediapipe Blazepose during the exercises on a per frame basis. We then compare these sets of angles obtained from the first video with the set of angles obtained from the next video.

These are the key-points coordinates (labeled from 0 till 32) that we obtain per frame by using Mediapipe Blazepose:



On top of these 33 key-points we add 4 new key-points (labeled 33, 34, 35, 36), to consider a few important joint angles which are relevant to compare exercises. We consider 18 joint angles in total, which have also been marked in the following illustration:



Implementation

Data

- Again, we use the videos used to check the algorithms associated with Angle Invariance with Euler's Rotation Matrix.
- We use Video F, Video L and Video R, which are videos of subjects doing bicep curls captured from different angles, i.e. front, left and right.

Pre-Processing (Extracting Joint Angles)

- We first define a helper function named `calculate_angle` to calculate the angles between any three 3D coordinate points. Say the points are A, B, C. We then calculate the angle between the vectors AB and CB by using the cosine rule.
- We then store an array named `keypoint_index` which stores all the 18 three-tuples, which stores the indices of the key-points required to identify a particular joint angle (which will be identified by the 3 coordinates corresponding to the indices in the tuple).
- Now, using the helper function `calculate_angle` and the array `keypoint_index` we can extract the 18 joint angles on a per frame basis and store them in a single array of size (n,18), n being the no. of frames.
- We then use DTW to compare these arrays obtained in the respective videos.

Algorithm

- **Algorithm 8.1:**
    - This is implementation without the cube normalization. We extract the coordinates of the key-points via mediapipe blazepose. We also add 4 new key-points for our use at the following locations:
        - Left Hand
        - Right Hand
        - Neck
        - Middle Pelvis
    - We then use the 37 key-points to extract the 18 joint angles per frame as described above and finally construct a (n,18) (n is the no. of frames in the video) and use this to compare the videos using DTW.
- **Algorithm 8.2:** This follows exactly the same pipeline as above, except after implementing cube normalization on a per-frame basis as described in the algorithms above.

Result

- We first observe the results by comparing these videos using the algorithms that do not incorporate any form of Angle Invariance:

	Front vs Left	Left vs Right	Right vs Front
Algorithm 5	47.6	38.3	52.1
Algorithm 6	43.1	22.8	43.5

- Now, we check the comparison results using **Algorithm 8.1** and **8.2**:

	Front vs Left	Left vs Right	Right vs Front
Algorithm 8.1	62.8	48.3	60.4
Algorithm 8.2	76.8	74.2	75.7

Observations

- We get a clear improvement in scores as obtained from the naive comparison (without implementing angle invariance implementations) than using the newer algorithms.
- Clearly **Algorithm 8.2** seems to be the best of the lot (which has also been confirmed by tests on other datasets).

DTW: Tests on Non-Vanilla Datasets

Test Methods

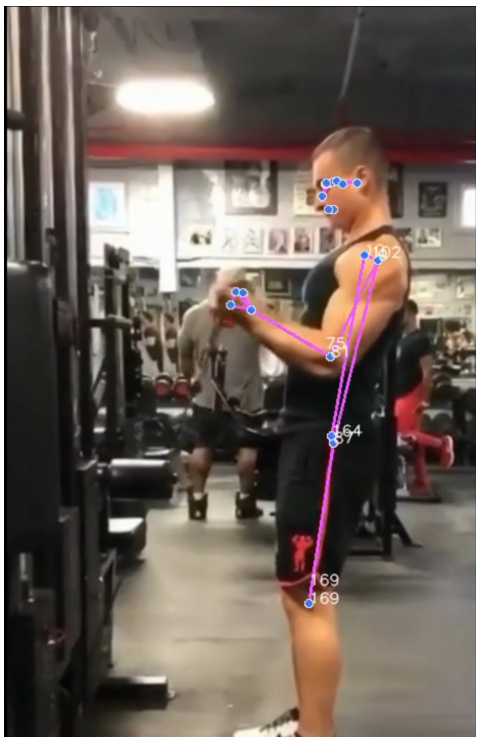
We tried testing the algorithms on various non-vanilla datasets of people doing various exercises. Amongst those, **Algorithm 8.2** seemed to perform the best overall. This is possibly due to the robustness of the algorithm obtained via Cube Normalization (which helps tackle variation of location subject w.r.t. the frame and the size of the subject) and Angle Invariance obtained by considering the 18 relevant joint angles per frame (which helps tackle the variation because of camera angles).

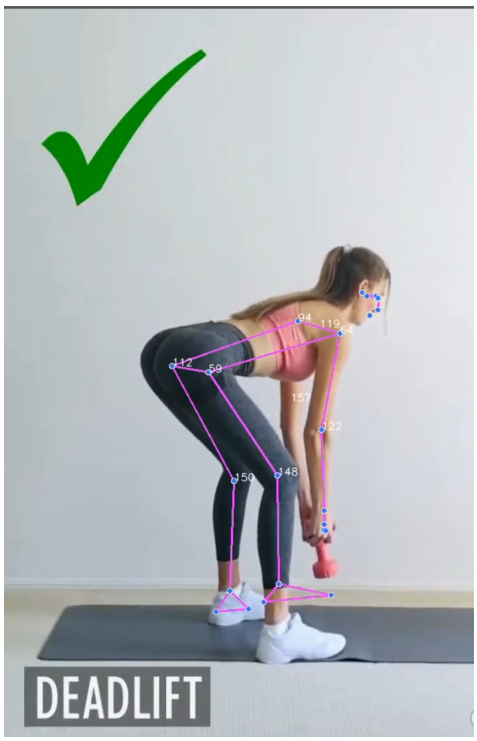
We consider the following exercises for testing:

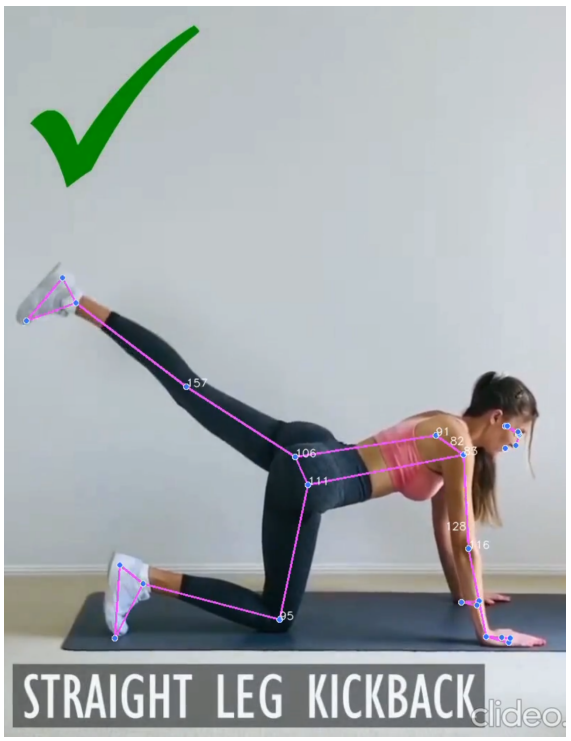
- Cable Bar Curls
- Deadlift
- Straight Leg Kickbacks
- Reverse Fly
- Plank
- Lunges
- Front Arm Raises
- Donkey Kick
- Bulgarian Split Squats
- Boat Pose
- Dumbbell Curls

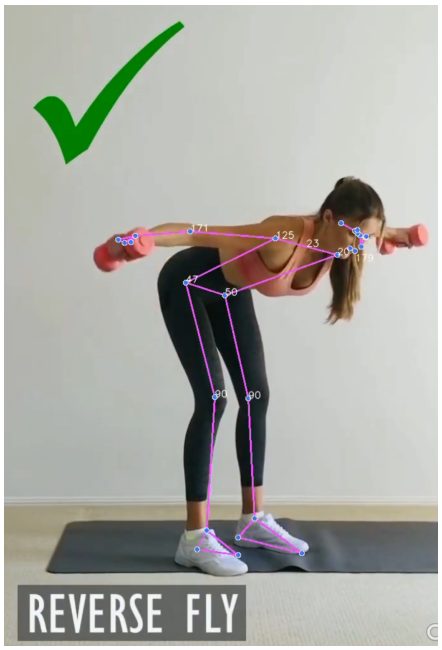
Snapshots of some of the test videos:

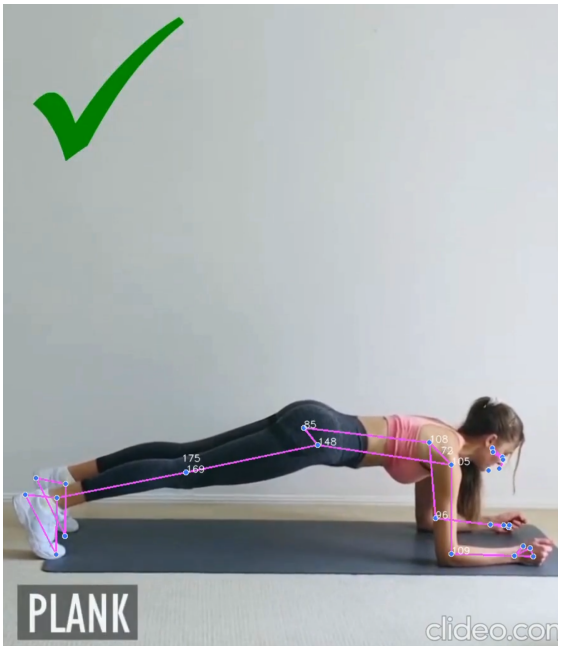
The videos have marked all the key-points that Mediapipe Blazepose was able to detect with a confidence parameter greater than the threshold of 30% set by us.
We have also added the joint angles corresponding to the key-points in the videos.

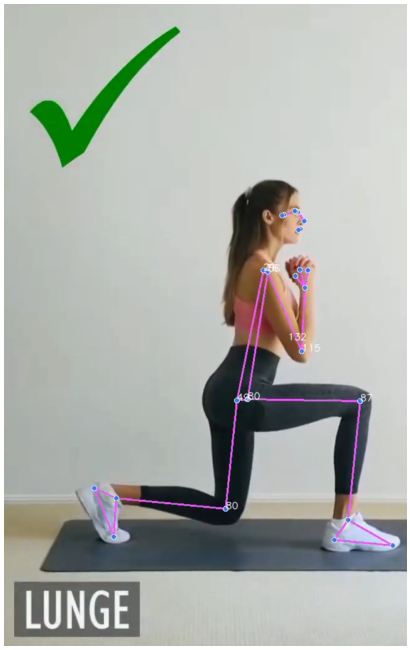


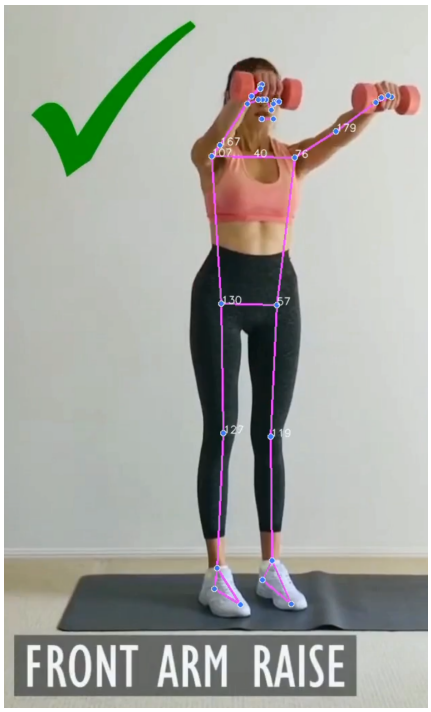


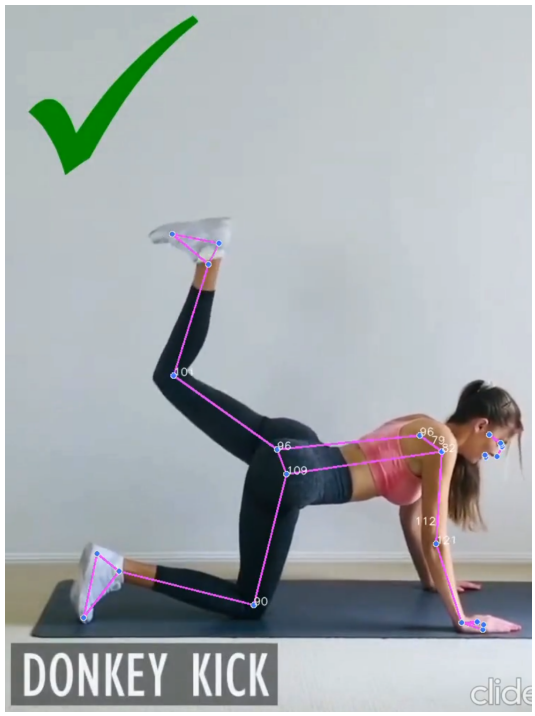


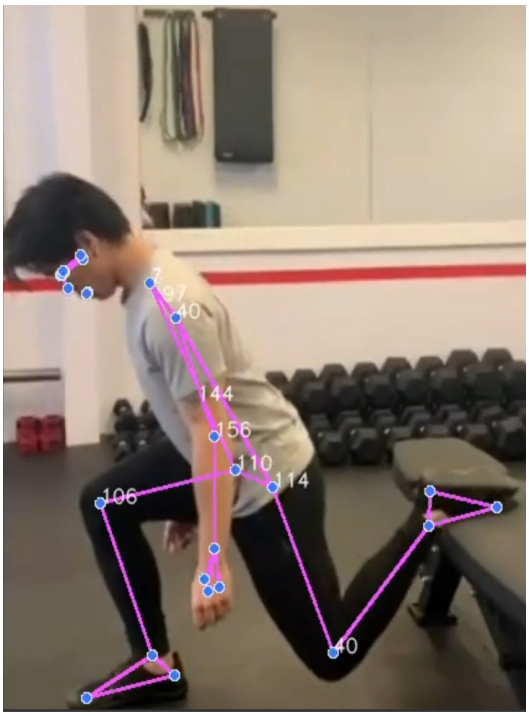

REVERSE FLY

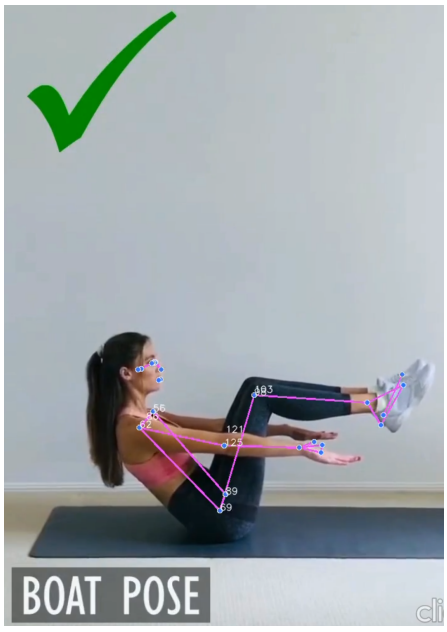

PLANK

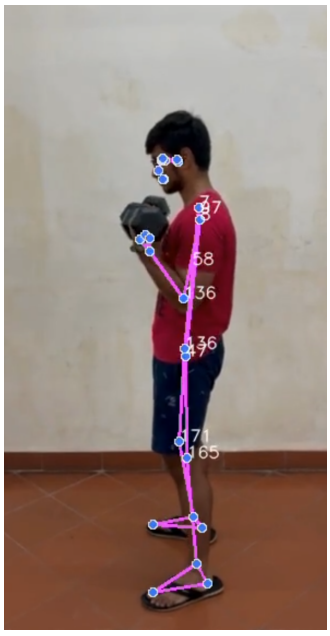

LUNGE


FRONT ARM RAISE


DONKEY KICK




BOAT POSE



For each exercise we consider 2 sets of videos:

- **Right Videos**: These are collections of videos where the particular exercise is being performed correctly with the correct form.
- **Wrong Videos**: These are collections of videos where the particular exercise is being performed incorrectly with incorrect form.

Now, we do two types of comparison using **Algorithm 8.2**:

- We first consider all the videos from the collection of **Right Videos** and compare them against each other and output the scores of individual comparisons and finally the **average score** (we are comparing the right videos to verify that exercises done by the experts should be more similar to each other).
- Then we consider all the videos from the collection of **Wrong Videos** and compare them against all the videos from the collection of **Right Videos** and output the scores of of individual comparisons as well as the **average score** (we are basically comparing the amateur videos, the ones with incorrect forms to the videos of the experts).

Results

Below is the table containing the average scores obtained using the above methods on the exercises mentioned above using **Algorithm 8.2**:

EXERCISES	Right vs Right Videos	Wrong vs Right Videos
Cable Bar Curls	89	57
Deadlift	92	70
Straight Leg Kickbacks	91	80
Reverse Fly	94	85
Plank	89	76
Lunges	86	80
Front Arm Raises	93	73
Donkey Kick	86	82
Bulgarian Split Squats	91	83
Boat Pose	94	80
Dumbbell Curls	84	78
AVERAGE SCORES	89.9	76.7

It is quite evident from above that the algorithm seems to work well while distinguishing between videos where the exercise is being performed with correct form against the ones with wrong movements. This has the potential to be a tool that can be used to compare trainee exercise videos with the already available trainer/expert videos and obtain feedback based on that.

Advantages

- The algorithm is quite fast, (C version of DTW for python) which makes it suitable for large video comparisons.
- Algorithm is robust to modifications; we can tweak the algorithm to use other key-point coordinate extraction techniques instead of Mediapipe Blazepose. We can also introduce some more key-points (than the already existing 37) and hence more joint angles (than the already existing 18) and make improvements in the comparison.
- Makes the comparison fair by taking the size and height of the person out of the equation, scores solely based on the actions of the subjects.
- Tackles the problem of camera angle invariance, results in better comparison.
- Can increase or decrease the scoring sensitivity of the model to obtain a stricter/lenient comparison.
- Model is general purpose and exercise invariant.
- As a non-ML model, it does not require any data for training (Blazepose being pre-trained).

Limitations

- Cannot distinguish between subtle bad forms in videos against the videos with good form (relatively better at distinguishing exaggerated inaccuracies at form).
- Model is dependent on Mediapipe Blazepose for key-point detection. If Mediapipe cannot confidently predict the coordinates of a key-point in certain frames of a video, the result might be affected.