Yoga Train Everywhere: Video Coaching and Comprehensive Support

Nicole Lin
Columbia University
nsl2126@columbia.edu

I. Introduction:

The popularity of fitness and wellness programs has been steadily increasing in recent years, with the global fitness industry being valued at $94 billion in 2018 and expected to grow to $106 billion by 2023, according to the IHRSA [1]. With this growth comes a need for tools and resources that can help individuals safely and effectively achieve their fitness goals without the potential for injury.

Computer vision-aided fitness coaching uses advanced algorithms and machine learning to monitor and analyze a person's movements and provide personalized guidance and feedback. This technology can provide real-time feedback on posture, form, and technique, helping individuals ensure that they are performing exercises correctly and safely. Additionally, computer vision-aided fitness coaching can provide motivation and accountability through personalized goal setting and progress tracking. With the increasing availability of this technology, individuals have access to personalized fitness guidance and support, even if they don't have access to a personal trainer or subscriptions.

II. Objective:

I introduce Yoga Train Everywhere, an innovative deep pose estimation solution to help individuals improve their yoga practice and overall physical fitness through video and image coaching. I bring the experience to everyone with access to a laptop, hence the name “Yoga Train Everywhere.”

To use Yoga Train Everywhere, the user inputs a YouTube link to an instructor video as well as start and stop timestamps of the video segment they’d like to follow. The system then tracks the similarity between the instructor and user's yoga poses by analyzing the instructor video and live video feed from the user. This tracking allows the system to identify any differences between the user and instructor, which will cause the system to alert the user visually that a specific body part is at the wrong angle. A similarity score between poses is also displayed in real-time. These real-time feedbacks can be useful in helping the user correct their form and technique, reducing the risk of injury and ensuring that they are performing exercises correctly and safely.

At the end of the yoga session, Yoga Train provides a final image report of their incorrect poses (if there were any) and an overall similarity score that determines how accurate the session was. This feedback allows the user to track their progress over time and identify areas for improvement.

An Aside on Originality: Since MediaPipe Pose is a public resource, many people have used it for pose estimation projects to detect body landmarks and draw skeletons on an image or video [2]. Some applications have also used MediaPipe Pose for the purpose of fitness: counting the number of repetitions of a specific exercise [3]. However, those applications do not involve comparing the user to an instructor video simultaneously (using multiprocessing) and are always a standalone user image or video. Those projects also do not give real-time feedback on form and technique and an output file at the end of the program, but simply detect poses, calculate one angle, and count the number of repetitions.

III. Limitations and Assumptions:

The limits that have been placed on the investigation include the absence of yoga equipment such as physio balls, rollers, etc. since I will be performing a live demonstration in Professor Kender's office or lab that does not have yoga equipment. This means that all exercises demonstrated in real-time will be equipment-less. While this may limit the range of exercises that can be demonstrated, it also means that the system developed will be accessible to a wider range of individuals who may not have access to yoga equipment.

Another limitation of the project is that the real-time camera (webcam) must be capturing video at a similar angle as the input instructor video with all body joints in camera view to ensure a correct similarity score. (Or the user must orient themselves the same way as the instructor.) This is because differences in video capturing between the instructor and user will result in vastly different determined poses/movements, which will decrease the accuracy of the system. Therefore, the system requires ensuring that the camera angle is consistent so that the system is able to accurately track and analyze the user's movements.

Lastly, it is important that the user is able to follow along with the trainer video with approximately the same timing in order to determine the correct synchronization score. In order to ensure accurate synchronization scores, the user should aim to perform the instructor's yoga pose within 2 seconds after the instructor has changed his/her pose. This limitation may require the user to have a basic level of fitness and familiarity with the exercises being demonstrated. By following these timing guidelines, the system can provide more accurate feedback and help the user to improve their yoga practice.

IV. Capture, Approach, Design and Algorithms, and Output:

Capture:

**Domain engineering of these videos explained in *Design and Algorithms: Steps 1.1 and 2.1*

1. User videos: I use the camera on a MacBook Pro for live webcam video capture of users doing yoga exercises who are following along with the instructor video (with limitations stated above).
2. (Pseudo) Video database: Any instructor video from YouTube doing equipment-less, isometric yoga exercises such as T-pose, warrior, reverse warrior, half forward bend, etc.

Approach:

I use the following summarized pipeline for Yoga Train Everywhere (Figure 1):

1. Video Input: Ask for the user to input the YouTube video link and start and stop timestamps of the portion of the video he/she wants to follow along with. Then, convert the YouTube (instructor) video to mp4 format and trim the video at the given timestamps.
2. Pose Calculation: At run time, take the instructor video and user's webcam feed and calculate poses and joint coordinates using Google's MediaPipe Pose simultaneously using python's multiprocessing package [4].
3. Angle Calculation: From the x and y coordinates of the landmarks, get the angles for the landmarks for both the user and instructor.

4. Similarity Calculation: Compare the landmark angles of the user and instructor in real-time to determine specific landmark similarity and pose similarity which are both labeled on the user video and eventually output to an Excel file (explained in the next subsections).

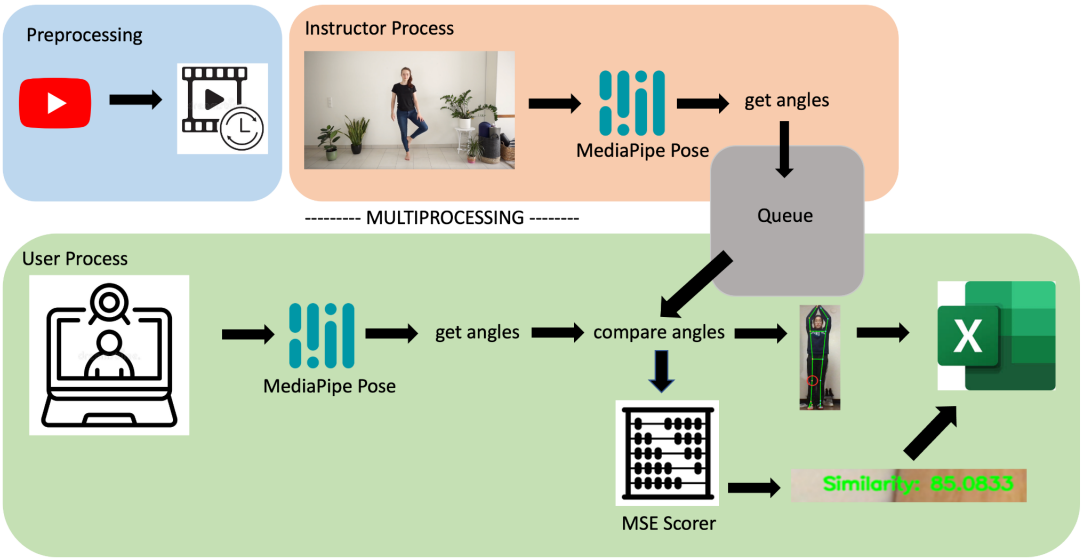


Figure 1: Overview of Yoga Train Everywhere Pipeline

Design and Algorithms:

0.1 Package Imports

The following packages were imported to the program: cv2 (OpenCV), numpy, subprocess, multiprocessing, tkinter, time, pyautogui, xlsxwriter, datetime, and mediapipe. Their uses will be explained in the next sections.

1.1 Instructor Video Formatting and Cutting

The application asks the user to input their desired YouTube video link as well as start and stop timestamps in the terminal in the format `python yoga_train.py '<YouTube video link>' <START timestamp> <STOP timestamp>`. The timestamps have the format HH:MM:SS. The yt-dlp command line program downloads the video into a .mp4 file that can be read by OpenCV. Then, the command-line tool ffmpeg trims the video according to the user's input start and stop timestamps. The subprocess module allows me to run the command-line arguments using python. Afterwards, I found the number of seconds between the two input timestamps by using datetime.strptime() and subtracting the two times.

2.1 Reading Instructor and User Video

Given the system's requirement to simultaneously process two input video streams – the instructor's video and the live user's video – I utilize the multiprocessing module [4] for concurrent execution of two processes. The first process, p1, runs the function "instructor" and the second process, p2, runs the function "user." I then create a multiprocessing queue and a multiprocessing value. The queue and value objects are used to communicate between the two processes. The instructor process puts the array of instructor angles into the queue, and the user process reads the array of instructor angles from the queue to compare to that of the user. The count value is used to keep track of the number of frames processed by

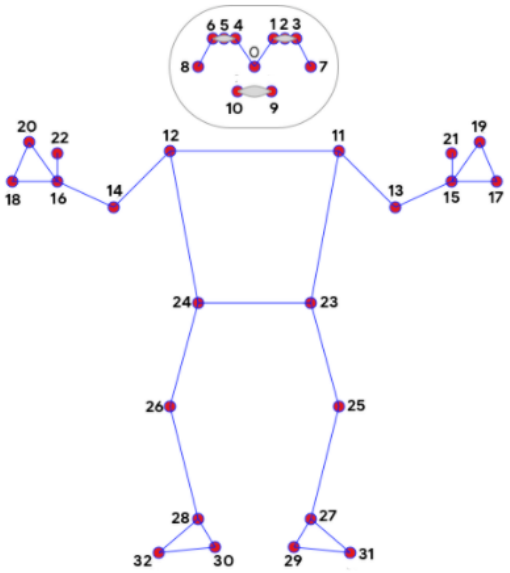
the instructor process that is read by the user process. Then, the processes are started using the start() method and the join() method is called on both processes at the end.

OpenCV is employed in both the “instructor” and “user” functions to read the frames of both videos. Both videos are resized to half of the laptop’s screen width (found using tkinter) using cv2.resize so that the videos are side-by-side to each other at the top of the screen, where users can view themselves as well as the instructor. The instructor video and user's live webcam footage are also flipped for a mirror-effect using cv2.flip.

2.2 Pose Calculation

To extract the body poses of both the instructor and user, my approach involves utilizing the MediaPipe Pose package [5]. I decided on MediaPipe Pose because it is optimized for real-time performance, making it suitable for Yoga Train Everywhere that requires low-latency processing on a laptop. It is also flexible and designed to work with a wide range of input sources including webcams and pre-recorded videos. MediaPipe Pose also extrapolates landmark coordinates even when they are out of frame, allowing for consistent landmark angle calculations.

Before running MediaPipe Pose on the instructor and user videos, I incorporated OpenCV's BackgroundSubtractorMOG2 [6] for background subtraction to possibly further enhance MediaPipe’s performance. However, the foreground mask resulted in an isolated white human figure against a black background made it even more difficult for MediaPipe Pose to accurately determine landmarks as the shadows wiped out many human features, resulting in a “blob-like” structure. Therefore, I kept the original video and allowed MediaPipe Pose to automatically perform background segmentation.



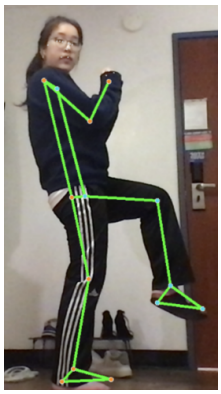
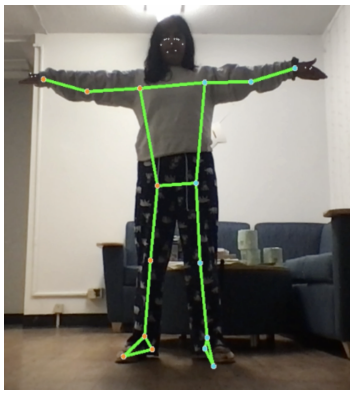
0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
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

Figure 2: The location of 33 pose landmarks predicted by MediaPipe Pose

MediaPipe Pose simultaneously determines 33 3D pose landmarks for both the instructor and user (Figure 2), represented as 3D coordinates stored in a JSON format. However, only 16 out of the 33 landmarks are relevant to this application to visualize the poses of both the instructor and user: left_shoulder,

right_shoulder, left_elbow, right_elbow, left_wrist, right_wrist, left_hip, right_hip, left_knee, right_knee, left_ankle, right_ankle, left_heel, right_heel, left_foot_index, and right_foot_index.

I annotate both videos using the MediaPipe drawing class [7] to draw dots on each of the 16 landmarks and connect each landmark with a line to form a skeleton (Figures 3 and 4). The remaining landmarks and their connections are not drawn. By overlaying the skeletons of both the user and instructor on their respective videos, the user can easily compare his/her movements with those of the instructor, helping him/her to improve their technique and similarity.

Figures 3 and 4: Examples of MediaPipe Pose for pose tracking on instructor video and user webcam

3.1 Finding Instructor and User Angles

The next step in my system is to compare the similarity between the poses of the instructor and user.

Directly comparing the (x, y, z) coordinates of the instructor and user's pose landmarks found in Section *2.2 Pose Calculation* by calculating the Euclidean distance between them is one way to measure similarity. However, this approach does not account for differences in body types, limb lengths, and camera distance, and would not produce an accurate score.

Therefore, my solution was to use the (x, y) coordinates of the instructor and user's pose landmarks to calculate the angles of the user's and instructor's landmarks (i.e., degree of knee bend) and then compare the angles between them [8]. This method is more reliable since angles are invariant to body size and camera distance. However, it is important to note that the angle of a landmark changes as the user's orientation changes. Thus, to address this issue, I have imposed a limitation that the user must be oriented towards the camera in the same way that the instructor is oriented (refer to the Limitations section).

The angles for 8 landmarks – left shoulder, right shoulder, left elbow, right elbow, left hip, right hip, left knee, and right knee (labeled on Figure 5) – are calculated using the Numpy package and the formula below [9]:

```
angle = np.arctan2(P3.y - P1.y, P3.x - P1.x) - np.arctan2(P2.y - P1.y, P2.x - P1.x)
```

where P1, P2, P3 represent three points needed to form an angle. Using the arc tangent formula, we could find the angle between P3 and P1 with respect to the origin and similar for P2 and P1. Then subtract the two angles to find the final angle in radians for the three points.

Pertaining to my code, the system grabs (x, y) coordinates from each of the 8 landmarks and puts them in separate Numpy arrays. Then, the angle function above is called to find the angle of a certain landmark. For example, left_elbow_angle was found using the coordinates from left_shoulder, left_elbow, and left_wrist with the left_elbow in the middle, denoting that is the angle we are trying to find.

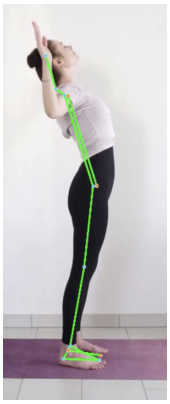
Lastly, since the instructor and user angles have to be compared in real time, I utilized a multiprocessing.queue() that sent the instructor angles from the instructor process to the user process. In the code, the instructor process puts the array of instructor landmark angles into the queue using the put() method and the user process retrieves them using the get() method. The queue ensures that the messages are passed in the order they were added, so that the user process can compare the correct set of angles with those from the instructor.

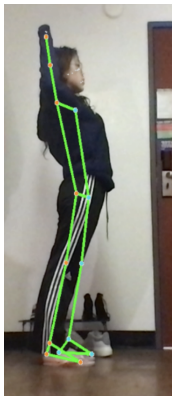
Figure 5: Angles for the 8 landmarks labeled on instructor

3.2 Landmark Coaching and Support

The comparison between the instructor angle array and user angle array is performed using semicircular subtraction which calculates the absolute semicircular difference between angles of two arrays. Since angles are circular measurements and two angles that differ by 180° are actually pointing in opposite directions, I decided to use semicircular difference to find the shortest distance between them instead of simply subtraction. If the largest semicircular difference between the angles of the instructor and user landmarks exceeds a threshold of 30°, the specific landmark is circled in red using cv2.circle(). This threshold value was determined based on both intuition and experimentation to be a visible and easily interpretable change in angle.

However, if the landmark with the largest difference happens to be the right or left shoulder, a different threshold range of 50° to 160° is used. This range was established through multiple tests of different poses. MediaPipe Pose tends to struggle in capturing the shoulder landmarks and their respective angles when the user or instructor is positioned at a 90° angle from the camera. In Figures 6 and 7, we observe that although the user and instructor are in similar poses, their right and left shoulder angle calculations differ significantly. While the instructor's body coordinates somewhat overlap each other, the user's body coordinates clearly form a rectangle.





Figures 6 and 7: Annotated skeletons of instructor and user facing to the right

Then, every 5 seconds, the application checks for circled landmarks and the amount of movement by the instructor. The 5-second timing allows users to adjust their landmark positions as necessary. If the instructor is moving frequently, likely changing poses, differences between the user's and instructor's angles will not be penalized and a screenshot will not be taken. To determine whether the instructor has changed poses, I compare the current and previous five seconds' arrays of instructor landmark angles. If the average difference between them is less than 50, it indicates that the instructor did not change poses. This threshold of 50 was determined through multiple trials of instructor videos with varying amounts of movement from minor movements during a pose to completely changing poses from a pose to another pose.

If there is a circled landmark and the instructor is moving minimally, the system takes screenshots of both the instructor and user videos using the 'pyautogui' package. These images are resized and then added to the 'yoga_train_output' Excel workbook using the 'xlsxwriter' package. The Excel workbook (Figure 8) serves as a useful tool for users to refer back to when analyzing their incorrect poses and what needs to be fixed.

Figure 8: Snapshot from an example Excel workbook

3.3 Pose Scoring

While the *3.2 Landmark Coaching and Support* step calculates the angle differences between specific landmarks of the user and instructor, the overall similarity between the user and instructor's poses is also determined to score the user's performance in real-time (Figures 9 and 10). In my proposal, I initially

suggested using cosine similarity [10] using the vectors formed from the 8 landmark angles (i.e., <90°, 87°, 180°, 179°, 178°, 182°, 60°, 65°>). However, this approach did not consider the magnitude of the landmark angles. For example, if the user's angle vector is <20°, 20°, 20°, 20°, 20°, 20°, 20°, 20°> and the instructor's angle vector is <80°, 80°, 80°, 80°, 80°, 80°, 80°, 80°>, the cosine similarity between two vectors would be 1, but the user should not be getting a high similarity score. To address this issue, mean square error (MSE) was used instead as it is more interpretable as larger MSE values correspond to larger angle differences between user and instructor.

Figure 9: A high similarity score when the user is performing a very similar pose to that of the instructor

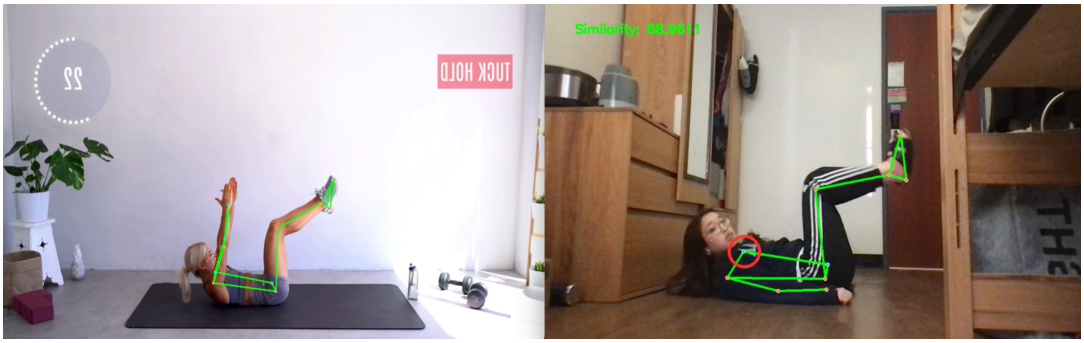


Figure 10: An average similarity score when the user is performing a pose that is somewhat similar to that of the instructor

To penalize large-angle differences by giving them a lower score, rather than a linear penalty, the hyperbolic tangent (tanh) function was employed. The function maps the range of MSE from $[0, \infty)$ to $[0,1]$. Since MSE increases as the user pose become more different than the instructor poses, $1 - \tanh(0.0002 * \text{MSE})$ was determined and the constant 0.0002 was used to make the thresholding easier. Figure 11 illustrates the scoring function vs. angle difference which shows that as the angle difference increases, the value of the function decreases drastically. The larger drop in values as angle difference increases makes it easier to find similarity scores.

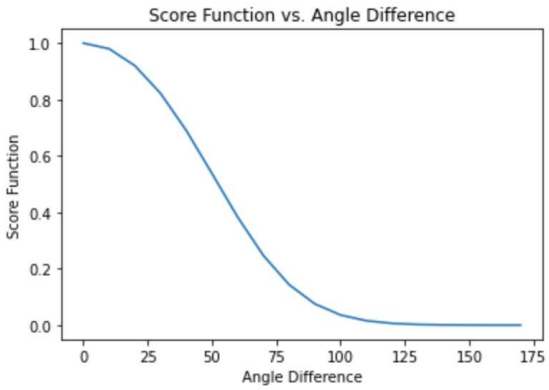


Figure 11: 1 − tanh(0.0002 ∗ MSE) vs. Angle Difference.

Additionally, the system will store the current, real-time similarity scores in a Numpy array every second to calculate the average similarity score using np.mean() upon termination of the instructor video. The average similarity score is written to the Excel workbook with the images to complete the overall report.

4.1 Closing Instructor and User Video

The application closes the instructor video window and user video once the instructor video ends using cv2.destroyAllWindows(). To time when the user webcam should end, I use the multiprocess.value() object that stores the current frame number of the instructor video and compare it to the value obtained by frames per second * number of seconds of instructor video (total number of instructor video frames).

Outputs:

The system has a total of six visual outputs:

1. Annotated instructor video with skeleton overlay
2. Annotated user video with skeleton overlay
3. Displayed real-time pose similarity score (modified MSE) as a percentage
4. Red circle around specific landmark on user video when landmark angle is different than the landmark angle of the instructor
5. Overall similarity score as a percentage in the excel workbook
6. Excel workbook of side-by-side instructor and user pictures (with circles) whenever the user's specific landmark exceeds an angle difference of 30° so that users have a reference to what poses and specific landmarks they performed incorrectly at the end of the session.

V. Evaluation:

I evaluated Yoga Train Everywhere's performance through quantitative and qualitative measures. Quantitative measures include pose tracking and similarity scores. Specifically, the application should be able to track the user's pose accurately in real-time with minimal delay and error. In terms of similarity scores, the system should provide feedback and scores that accurately reflect the user's ability to match their poses with the instructor's poses, not taking into account the differences in body types or limb length ratios.

1. Quantitative measures:
    a. Accurate pose tracking:
        i. For instructor (input video): I ran my system with three instructors performing ten different exercises for a total of 30 trials to see if the skeleton overlay on the instructor accurately reflects the instructor's pose. The accuracy score is calculated by the number of accurate skeleton overlays/30, and the score should be above 90%.
        ii. For users (live webcam video): I ran my system with three different users of different body types performing 10 exercises in real-time for a total of 30 trials to see if the skeleton overlay on the user accurately reflects the user's pose. The accuracy score is calculated by the number of accurate skeleton overlays/30, and the score should be above 90%.
    b. Accurate Landmark Comparisons:
        i. I assessed acceptable performance of the landmark comparisons (red circle marking on the user video) by running my system with 3 different poses at different camera angles for the 8 landmark angles. For each of the 24 trials, the landmark angle was varied past the threshold value (50° for left shoulder and right shoulder angle, and 30° for the rest of the landmark angles) while the other landmark angles were kept as similar to the landmark angles of the instructor. I counted the number of times a red circle was displayed on the user video over the specific landmark correctly divided by the number of trials to get an accuracy score. An acceptable performance would be an accuracy of greater than 80%.
    c. Accurate similarity scores: average similarity score across multiple trials (of singular exercises).
        i. I assessed acceptable performance by running my system with five different users of different body types performing five different exercises with "bad," "ok," and "perfect" yoga poses for a total of 75 trials. Specifically, one user was given five different instructor videos of singular yoga exercises (i.e., video 1: warrior, video 2: T-pose, video 3: downward dog, video 4: tree, video 5: cat/dog). Each of these exercises were run three times during which in the first run, the user performs "bad" fitness forms, the second run the user performs "ok" fitness forms, and the third run the user performs "perfect" fitness forms. When the user performs the "bad" pose for a specific exercise (i.e., lying flat when supposed to do the T-pose), I expect an average similarity score of below 50%. The "ok" pose should have an average similarity score between 50% and 85% depending how much the user deviates from a "perfect" form, while the "perfect" pose should have an average similarity score above 85%. I expect the similarity scores to be relatively consistent across all five users (if not, see Failures section below for potential issues and why they might have occurred).
2. Qualitative measures include feedback from users on the overall experience and ease of use of the system.

Failures in performance could provide valuable insights into areas for improvement. By analyzing why the system failed to accurately track poses or provide accurate similarity scores, I could identify specific

areas of the system that need improvement, specifically, the pose tracking algorithms (MediaPipe Pose) or the similarity scoring system. While I cannot directly correct failures in the pose tracking system (MediaPipe Pose), I can indirectly help the algorithm better do pose tracking by performing better preprocessing on the input video and live video. If there are failures in the landmark comparisons, there is likely an issue regarding determined angle thresholds. If there are failures in the similarity scoring system, there is likely an issue with how I am calculating the differences between the angles of the location markers given by MediaPipe Pose (MSE scorer), and will likely have to change my similarity metric. Therefore, by understanding why failures occur, I can make informed decisions on how to improve the system's accuracy and overall user experience.

VI. Results:

Quantitative Evaluation Results: Accurate Pose Tracking

Out of the 30 trials conducted on instructor videos, 28 of them resulted in accurate skeleton overlays that aligned with the instructor's pose, resulting in an accuracy of 93.33%. However, there were 2 trials where the skeleton overlays did not align with the instructor's pose, as shown in Figure 12. In this specific case, the right foot and the right foot index landmark dots were not drawn, and the connection lines between the right knee & right foot, right foot & right foot index, right foot & right heel, and right heel & right foot index were, therefore, also not drawn.

Figure 12: An incorrect instructor pose trial

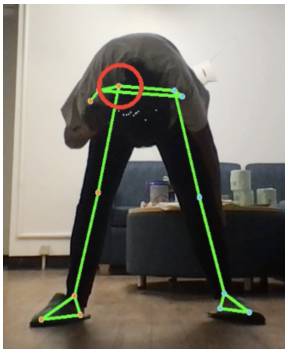


Figure 13: An incorrect user pose trial

Out of the 30 trials conducted on live user webcam feed, 24 of them resulted in accurate skeleton overlays that aligned with the user's pose, resulting in an accuracy of 80%. The 6 trials where the skeleton overlays did not align with the user's pose were due to similar issues as the instructor's pose tracking: missing landmarks and their respective connections. In Figure 13, we see that the entirety of the user's right arm is not overlaid with a green line since the backlighting caused MediaPipe to determine the arm as part of the leg. Because the accuracy of the live camera feed is less than 90%, this indicates that like any system, MediaPipe has its limitations and may not always provide accurate results, especially in certain conditions such as bad lighting (Figure 10), camera angle, convoluted poses (Figure 10), clothing, occlusions, and abrupt movements. These inherent limitations of MediaPipe Pose itself can explain the inconsistencies in calculating landmark similarity scores and pose similarity scores. The application could possibly be improved by using other state-of-the-art pose estimation models such as OpenPose [11], BlazePose [12], AlphaPose [13], and MoveNet [14] other than MediaPipe Pose.

Quantitative Evaluation Results: Accurate Landmark Comparisons

Out of the 24 trials conducted, 21 of them produced accurate red circle marking on the user video when landmark angle was varied past the threshold value (50° for left shoulder and right shoulder angle, and 30° for the rest of the landmark angles). The resultant accuracy was 87.5%, indicating good performance.

The three trials where there was not a circle displayed on the user video, even when the angle of the user's landmark surpassed the threshold of either 30° or 50°, was unsurprisingly left shoulder, right shoulder, and right hip when the user is facing 90° away from the camera. Like mentioned before in section 3.2 (Figures 6 and 7), MediaPipe Pose is not always accurate at determining poses of the side profile of a person. Therefore, the differences between the side profiles of the instructor and user are especially evident for the left shoulder and right shoulder, and somewhat evident for left hip and right hip. This will result in the system not being able to accurately perform landmark comparisons.

Quantitative Evaluation Results: Accurate Similarity Scores

Out of the 75 trials conducted, 68 of them produced results consistent with the "perfect," "ok," and "bad" thresholds provided in the evaluation section; therefore, the accuracy for similarity scores is 90.67%. Overall, the user receives high similarity scores for "perfect" poses, medium similarity scores for "ok" poses, and low similarity scores for "bad" poses. None of the "perfect" poses performed during the 75 trials achieved an 85% similarity score or below.

Table I displays the user with the most inconsistent results, including two inconsistencies with a "bad" pose and an inconsistency with an "ok" pose. Exercise 3 (warrior II) resulted in a high similarity score (87.5578%) for an "ok" pose, greater than the given threshold for "ok" poses set at 50% to 85%. It also resulted in a similarity score of 66.0247% for a "bad" pose which is greater than the given threshold for "bad" poses set at 0% to 50%.

Table I: Table of User 2's Similarity Scores for 5 Different Exercises

	Exercise 1	Exercise 2	Exercise 3	Exercise 4	Exercise 5
Perfect	87.5597	99.6369	98.4886	97.9780	99.1944
Ok	49.1624	64.6386	**90.5582**	73.8199	77.7129
Bad	9.9731	46.7318	**52.8689**	17.3347	**57.6686**

** The inconsistent results are bolded

From the trials, I also observed a trend where it was easier to achieve a higher similarity score for simpler poses, such as warrior II pose, making it more difficult to receive a low similarity score for a "bad" pose (Figure 14). This is possibly explained because the warrior II pose has many similar angles to other poses which causes my MSE scorer to determine two poses as similar.

Figure 14: Top = “Ok” pose that resulted in a similarity score of 90.5582. Bottom = “Bad” pose that resulted in a similarity score of 52.8689

It is also worth noting that my application’s minute leniency in determining pose ratings as “perfect,” “ok,” and “bad” could impact the user's progress and improvement. As such, it may be necessary to adjust the threshold of the MSE scorer to be higher or implement additional measures to ensure more consistent and accurate results.

Qualitative Evaluation Results:
I received positive feedback on the informative scoring system (similarity score) and the red circle that highlights the most different landmarks, making it easy for users to correct their body positions. Users appreciated the side-by-side display of the instructor and user videos for comparison and the output file of wrong yoga poses for future reference.

However, some users found it challenging to view both the user and instructor videos simultaneously. This can be improved by providing a larger screen or hooking up the laptop to a monitor. Therefore, in future iterations, I will explore ways to address this issue and make the application more user-friendly.

VII. Reflection and Future Work:

Through this project, I have successfully developed a scoring algorithm and demonstrated the possibility of real-time yoga training on a laptop. From this project, I learned several key skills and concepts including a solid understanding of pose estimation techniques regarding body landmarks, keypoint detection, and multi-view geometry (identifying the positions and orientations of various body parts). I

also learned how to use multiprocessing and gained experience in process synchronization, inter-process communication, and parallelism. Additionally, I developed skills in processing real-time video streams, including frame rates and how to optimize video processing performance for videos with large volumes of data.

In addition to the feedback from users received from the results section, there are some basic improvements that can be implemented to enhance the user experience:

1. Finding an alternative method to annotate YouTube videos with MediaPipe Pose without requiring the user to download the entire video, which is both time-consuming and space-consuming.
2. Creating a user interface using Flask to make Yoga Train Everywhere more user-friendly for those who are not familiar with the terminal and how to run this project from the command line.
3. Finding an alternative to screenshotting and inserting images to an Excel workbook because the user video freezes for 0.7 seconds every time an image is screenshotted and inserted into Excel.

Furthermore, I acknowledge that yoga is not just about achieving the correct poses; it is also about how one transitions between them, commonly referred to as "yoga flow." By incorporating optical flow to measure the similarity of movements (pose synchronization, not similarity) between the user and instructor, we can capture the "flow" of yoga. Currently, my algorithm only considers the spatial aspect of yoga (poses), and incorporating optical flow will track the temporal aspect of yoga (flow/moves) as well.

VIII. References:

[1] "2022 IHRSA Global Report Recognizes Fitness Industry Resilience." *IHRSA*,https://www.ihrsa.org/improve-your-club/2022-ihrsa-global-report-recognizes-fitness-industry-resilience/.

[2] I. Grishchenko, V. Bazarevsky, MediaPipe holistic—simultaneous face, hand and pose prediction, on device. Google AI Blog, Google, 10 Dec 2020. https://ai.googleblog.com/2020/12/mediapipe-holistic-simultaneous-face.html

[3] Renotte, Nicholas, director. *AI Pose Estimation with Python and MediaPipe | Plus AI Gym Tracker Project*. *YouTube*, YouTube, 14 Apr. 2021, https://www.youtube.com/watch?v=06TE_U21FK4&ab_channel=NicholasRenotte. Accessed 1 May 2023.

[4] "Multiprocessing - Process-Based Parallelism." *Python Documentation*,https://docs.python.org/3/library/ multiprocessing.html.

[5] "Pose." *Mediapipe*, https://google.github.io/mediapipe/solutions/pose.html.

[6] "How to Use Background Subtraction Methods." *OpenCV*, https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html.

[7] Gulati, Aman Preet. "Pose Detection in Image Using Mediapipe Library." *Analytics Vidhya*, 1 Mar. 2022, https://www.analyticsvidhya.com/blog/2022/03/pose-detection-in-image-using-mediapipe-library/.

[8] "Pose Classification Options | ML Kit | Google Developers." *Google*, Google, https://developers.google.com/ml-kit/vision/pose-detection/classifying-poses.

[9] jaxjax 3, et al. "Python Code to Calculate Angle between Three Point Using Their 3D Coordinates." *Stack Overflow*, 1 Dec. 1962, https://stackoverflow.com/questions/35176451/python-code-to-calculate-angle -between-three-point-using-their-3d-coordinates.

[10] Raj, Krishna. “Human Pose Comparison and Action Scoring Using Deep Learning,OpenCV& Python.” *Medium*, 20 Mar. 2023, https://medium.com/analytics-vidhya/human-pose-comparison-and-action-scoring-using-deep-learning-opencv-python-c2bdf0ddecba.
[11] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019
[12] Google. 3d pose detection with Mediapipe Blazepose ghum and Tensorflow.js, 2021.
[13] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. Rmpe: Regional multi-person pose estimation. In ICCV, 2017.
[14] Ronny Votel and Na Li. Next-generation pose detection with MoveNet and Tensorflow.js, May 2021.

Other References:

[15] Kukil. “Body Posture Detection & Analysis System Using MediaPipe.” *LearnOpenCV*, 7 Mar. 2023, https://learnopencv.com/building-a-body-posture-analysis-system-using-mediapipe/.
[16] H. J. Lee and Z. Chen. Determination of 3D human body postures from a single view. Computer Vision, Graphics and Image Processing, 30:148–168, 1985.
[17] Martinez, Julieta, et al. “A Simple Yet Effective Baseline for 3D Human Pose Estimation.” 2017 IEEE International Conference on Computer Vision (ICCV), 4 Aug. 2017, doi:10.1109/iccv.2017.288.

~/yoga/yoga_train.py

```
import cv2
import numpy as np
import subprocess
import sys
import multiprocessing
from mediapipe.python.solutions.pose import PoseLandmark
from mediapipe.python.solutions.drawing_utils import DrawingSpec
import tkinter as tk
import time
import pyautogui
import xlsxwriter
from datetime import datetime
import mediapipe as mp
# drawing the pose landmarks and connections on the video frames
mp_drawing = mp.solutions.drawing_utils
# customize the style of the landmarks and connections, such as the thickness and color
mp_drawing_styles = mp.solutions.drawing_styles
# detecting and estimating the human pose from the video frames using machine learning mod
mp_pose = mp.solutions.pose


def get_youtube():
    # Remove the original video and download a new one
    cmd_str1 = 'rm -rf youtube.mp4'
    cmd_str2 = 'rm -rf video.mp4'
    # download the youtube video with the given ID i.e. https://www.youtube.com/watch?v=$Y
    # example: https://www.youtube.com/watch?v=KYbFGbLPlb0
    cmd_str3 = 'yt-dlp -f \'bestvideo[ext=mp4]\' --output "youtube.%(ext)s" \'' +  sys.arg
    # timestamp format is: HH:MM:SS
    cmd_str4 = 'ffmpeg -hide_banner -loglevel panic -ss ' + sys.argv[2] + ' -to ' + sys.ar
    # run each terminal command
    subprocess.run(cmd_str1, shell=True)
    subprocess.run(cmd_str2, shell=True)
    subprocess.run(cmd_str3, shell=True)
    subprocess.run(cmd_str4, shell=True)

    time1 = datetime.strptime(sys.argv[2], "%H:%M:%S")
    time2 = datetime.strptime(sys.argv[3], "%H:%M:%S")

    # find number of seconds elapsed between two timestamps in the format HH:MM:SS
    time_difference = time2 - time1
    seconds = time_difference.total_seconds()

    return seconds

# This function gets the angle of the middle point given three points
# parameters -> pt1,pt2,pt2: [x_coordinate, y_coordinate]
# formula: https://stackoverflow.com/questions/1211212/how-to-calculate-an-angle-from-thre
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End
```

```
    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b[0])
    angle = np.abs(radians*180.0/np.pi)

    if angle > 180:
        angle = 360-angle
    return angle

# This function performs a semicircular subtraction operation on two arrays of angles a an
# greater than 90 degrees to the equivalent difference within a semicircle.
def semicircular_subtraction(a,b):
    diff = a-b
    return abs(((diff + 90) % 180) - 90)

################################################################################################
# source: https://stackoverflow.com/questions/75365431/mediapipe-display-body-landmarks-on
# I am allowed to use this code because it is on a public website

# This code gets Mediapipe to only draw body specific landmarks (i.e. exclude facial landma
# It modifies the DrawingSpec and POSE_CONNECTIONS to "hide" a subset of landmarks

# get the default style and connections from the MediaPipe drawing package
custom_style = mp_drawing_styles.get_default_pose_landmarks_style()
custom_connections = list(mp_pose.POSE_CONNECTIONS)

# list of landmarks to exclude from the drawing
excluded_landmarks = [
    PoseLandmark.LEFT_EYE,
    PoseLandmark.RIGHT_EYE,
    PoseLandmark.LEFT_EYE_INNER,
    PoseLandmark.RIGHT_EYE_INNER,
    PoseLandmark.LEFT_EAR,
    PoseLandmark.RIGHT_EAR,
    PoseLandmark.LEFT_EYE_OUTER,
    PoseLandmark.RIGHT_EYE_OUTER,
    PoseLandmark.NOSE,
    PoseLandmark.MOUTH_LEFT,
    PoseLandmark.MOUTH_RIGHT,
    PoseLandmark.RIGHT_PINKY,
    PoseLandmark.LEFT_PINKY,
    PoseLandmark.RIGHT_THUMB,
    PoseLandmark.LEFT_THUMB,
    PoseLandmark.RIGHT_INDEX,
    PoseLandmark.LEFT_INDEX
]

for landmark in excluded_landmarks:
    # change the way the excluded landmarks are drawn
    # draw very small, almost invisible circles
    custom_style[landmark] = DrawingSpec(color=(220,209,191), thickness=None, circle_radius
    # we remove all connections which contain these landmarks
    custom_connections = [connection_tuple for connection_tuple in custom_connections if la

################################################################################################

def instructor(window_width, window_height, queue, count):
```

```
    cap = cv2.VideoCapture('video.mp4')
    # set the frame height to be half the dimensions of the screen size
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, window_width)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, window_height)

    # error checking to see if there is an error in opening the instructor video
    if cap.isOpened() == False:
        print("Error opening video stream or file")
        raise TypeError

    time.sleep(2)
    frame_number = 0

    with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
        while cap.isOpened():
            success, image = cap.read()
            if not success:
                break

            count.value = frame_number

            # image is now read-only and cannot be modified in place for performance optim
and memory-intensive
            image.flags.writeable = False
            # converts the color space of the image from BGR to RGB since MediaPipe Pose m
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            # detect poses in the image and returns a dictionary of landmarks and their po
            results = pose.process(image)

            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            # Draw the pose annotation on the image and pass in the modified connections l
            mp_drawing.draw_landmarks(image,
                results.pose_landmarks,
                connections = custom_connections,
                connection_drawing_spec = DrawingSpec(color=(0,255,0), thickness=4, circle
                landmark_drawing_spec = custom_style
            )

            # Resize the frame to half its size
            frame = cv2.resize(image, (int(window_width), int(window_height)))

            if results.pose_landmarks is not None:
                # detect pose landmarks in the current frame
                landmarks = results.pose_landmarks.landmark

                # grab (x,y) coordinates for different landmarks
                left_shoulder = [landmarks[11].x,landmarks[11].y]
                left_elbow = [landmarks[13].x,landmarks[13].y]
                left_wrist = [landmarks[15].x,landmarks[15].y]
                left_hip = [landmarks[23].x,landmarks[23].y]
                left_knee = [landmarks[25].x,landmarks[25].y]
                left_ankle = [landmarks[27].x,landmarks[27].y]

                right_shoulder = [landmarks[12].x,landmarks[12].y]
                right_elbow = [landmarks[14].x,landmarks[14].y]
                right_wrist = [landmarks[16].x,landmarks[16].y]
```

```
                right_hip = [landmarks[24].x,landmarks[24].y]
                right_knee = [landmarks[26].x,landmarks[26].y]
                right_ankle = [landmarks[28].x,landmarks[28].y]

                # calculate angles for the 8 landmarks
                left_elbow_angle = calculate_angle(left_shoulder, left_elbow, left_wrist)
                left_shoulder_angle = calculate_angle(left_elbow, left_shoulder, left_hip)
                left_hip_angle = calculate_angle(left_shoulder, left_hip, left_knee)
                left_knee_angle = calculate_angle(left_hip, left_knee, left_ankle)

                right_elbow_angle = calculate_angle(right_shoulder, right_elbow, right_wris
                right_shoulder_angle = calculate_angle(right_elbow, right_shoulder, right_h
                right_knee_angle = calculate_angle(right_hip, right_knee, right_ankle)
                right_hip_angle = calculate_angle(right_shoulder, right_hip, right_knee)

                # put all calculated angles into array to send over to user process
                angle_arr_instructor = np.array([left_shoulder_angle, right_shoulder_angle,
left_knee_angle, right_knee_angle])

                # put instructor angle array into queue which sends message over to user pr
                queue.put(angle_arr_instructor)

            # display video on the screen
            frame_flip = cv2.flip(frame,1)
            cv2.imshow('Instructor Video', frame_flip)
            if cv2.waitKey(5) & 0xFF == 27:
                break

            frame_number += 1

    # Release the Mediapipe Pose
    cap.release()
    # close all the opened windows
    cv2.destroyAllWindows()

def user(queue, window_width, window_height, count, sec):
    cap = cv2.VideoCapture(0)

    # initialize excel workbook and worksheet to write images and average similarity to
    workbook = xlsxwriter.Workbook('yoga_train_outputs.xlsx')
    ws = workbook.add_worksheet()

    # initialize array that contains similarity scores for each second of the video
    similarity_total_arr = []
    # landmarks in the order of the instructor and user angle arrays
    landmark_dict = ["left shoulder", "right shoulder", "left elbow", "right elbow", "left

    with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
        frame_number = 0
        img_num = 0
        old_instructor_angles = [0,0,0,0,0,0,0,0]
        first_time = True

        # find frames per second of instructor video and number of total frames
        fps = cv2.VideoCapture('video.mp4').get(cv2.CAP_PROP_FPS)
        frames = int(fps * sec)
```

```
        while cap.isOpened():
            # check if the instructor process ended (video ends) by determining if the inst
            if (count.value == frames - 5) or (count.value == frames - 4) or (count.value =
(count.value == frames) or (count.value == frames + 1) or (count.value == frames + 2) or (
+ 5):
                # Release the Mediapipe Pose
                cap.release()
                # close all the opened windows
                cv2.destroyAllWindows()
                break

            frame_number += 1
            success, image = cap.read()
            if not success:
                print("Ignoring empty camera frame.")
                # If loading a video, use 'break' instead of 'continue'.
                continue

            # image is now read-only and cannot be modified in place for performance optim
and memory-intensive
            image.flags.writeable = False
            # converts the color space of the image from BGR to RGB since MediaPipe Pose m
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            # detect poses in the image and returns a dictionary of landmarks and their po
            results = pose.process(image)

            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            # Draw the pose annotation on the image and pass in the modified connections li
            mp_drawing.draw_landmarks(image,
                results.pose_landmarks,
                connections = custom_connections,
                connection_drawing_spec = DrawingSpec(color=(0,255,0), thickness=4, circle_
                landmark_drawing_spec = custom_style
            )

            # Resize the frame to half of the screen size and and flip the frame so that i
            frame = cv2.resize(image, (window_width, window_height))
            frame = cv2.flip(frame, 1)

            # get message (instructor angles array) from the queue that the instructor pro
            queue.get()
            message = queue.get()

            # if the instructor angles array has valid values for all 8 angles
            if len(message) == 8:
                if results.pose_landmarks is not None:
                    # detect pose landmarks in the current frame
                    landmarks = results.pose_landmarks.landmark

                    # grab (x,y) coordinates for different landmarks
                    left_shoulder = [landmarks[11].x,landmarks[11].y]
                    left_elbow = [landmarks[13].x,landmarks[13].y]
                    left_wrist = [landmarks[15].x,landmarks[15].y]
                    left_hip = [landmarks[23].x,landmarks[23].y]
                    left_knee = [landmarks[25].x,landmarks[25].y]
```

```
                    left_ankle = [landmarks[27].x,landmarks[27].y]

                    right_shoulder = [landmarks[12].x,landmarks[12].y]
                    right_elbow = [landmarks[14].x,landmarks[14].y]
                    right_wrist = [landmarks[16].x,landmarks[16].y]
                    right_hip = [landmarks[24].x,landmarks[24].y]
                    right_knee = [landmarks[26].x,landmarks[26].y]
                    right_ankle = [landmarks[28].x,landmarks[28].y]

                    # calculate angles for the 8 landmarks
                    left_elbow_angle = calculate_angle(left_shoulder, left_elbow, left_wris
                    left_shoulder_angle = calculate_angle(left_elbow, left_shoulder, left_h
                    left_hip_angle = calculate_angle(left_shoulder, left_hip, left_knee)
                    left_knee_angle = calculate_angle(left_hip, left_knee, left_ankle)

                    right_elbow_angle = calculate_angle(right_shoulder, right_elbow, right_
                    right_shoulder_angle = calculate_angle(right_elbow, right_shoulder, rig
                    right_knee_angle = calculate_angle(right_hip, right_knee, right_ankle)
                    right_hip_angle = calculate_angle(right_shoulder, right_hip, right_knee

                    # put coordinates of all landmarks into an array
                    coordinate_arr_user = [left_shoulder, right_shoulder, left_elbow, right
                    # put all calculated user angles into array to comapre with instructor
                    angle_arr_user = np.array([left_shoulder_angle, right_shoulder_angle,
left_knee_angle, right_knee_angle])

                    # Get the semicircular difference between instuctor angles and user ang
                    diff = semicircular_subtraction(message,angle_arr_user)
                    # find the index of the largest semicircular subtraction. This will be
                    idx = np.argmax(diff)

                    # thresholds chosen because sometimes mediapipe doesn't capture the pos
                    if (50 <= diff[idx] <= 160 and (landmark_dict[idx] == "left shoulder" o
(landmark_dict[idx] == "left shoulder" or landmark_dict[idx] == "right shoulder")):
                        # if body index is actually in frame
                        # Before circling the specific landmark, I check to see if the coo
                        if int(coordinate_arr_user[idx][0] * window_width) < window_width a
                            # circle wrong body part with red circle
                            cv2.circle(frame, (abs(frame.shape[1] - int(coordinate_arr_use
25, (40, 50, 255), 4)
                            # every 5 seconds, check if there are any incorrect body parts
                            if frame_number % 50 == 0:
                                # compare with new_instructor_message = message
                                # if different by a certain percentage, then dont grab scr
                                instructor_diff = np.sum(np.abs(message - old_instructor_a
                                if instructor_diff < 50 or first_time:
                                    first_time = False
                                    old_instructor_angles = message
                                    print("say cheese!")
                                    # grab screenshot of frame and paste into pdf document
                                    screenshot = pyautogui.screenshot(region=(0, 110, 3550
                                    screenshot.save('./yt_screenshot' + str(img_num) + '.p
                                    ws.set_row(img_num, 300)
                                    # insert screenshot into excel
                                    ws.insert_image("A"+str(img_num + 1), './yt_screenshot
                                    img_num += 1
```

```
                    # find mean squared error of between the instructor angle array and use
                    mse = np.sum((message - angle_arr_user) ** 2)/8
                    # MSE scring function to find similarity score
                    score = (1- np.tanh(0.0002*mse)) * 100
                    # display similarity score
                    cv2.putText(frame, "Similarity: " + str(round(score,4)), (50, 50), cv2

                    # if timer is at one second, save mse scoring function between two arra
                    if (frame_number % 10 == 0):
                        similarity_total_arr.append(score)

            cv2.imshow('Personal Video', frame)
            cv2.moveWindow('Personal Video', window_width, -75)
            if cv2.waitKey(5) & 0xFF == 27:
                # Close the workbook
                workbook.close()
                break

    # find the mean of the similarity scores generated per second and write to the excel fi
    average_similarity = np.mean(similarity_total_arr)
    ws.write("A"+str(img_num + 1), "Average similarity: " + str(round(average_similarity,4

    # Close the workbook
    workbook.close()
    # print("Your Yoga Train report is generated.")

if __name__ =="__main__":

    seconds = get_youtube()

    # Create a window and set its size to half the screen
    root = tk.Tk()
    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()
    window_width = int(screen_width / 2)
    window_height = int(screen_height / 2)

    # initialize workbook object
    workbook = xlsxwriter.Workbook('yoga_train_outputs.xlsx')
    ws = workbook.add_worksheet()

    # initialize multiprocessing queue and value. Value is frame number of instructor video
    queue = multiprocessing.Queue()
    count = multiprocessing.Value('i', 0)

    # creating processes
    p1 = multiprocessing.Process(target=instructor, args=(window_width, window_height, queu
    p2 = multiprocessing.Process(target=user, args=(queue, window_width, window_height, cou

    # starting process 1 and 2
    p2.start()
    p1.start()

    # wait until processes 2 and 1 (instructor video) are finished
    p1.join()
    p2.join()
```

```
    p1.terminate()

    # once both processes finish
    print("Congrats! Yoga Session Completed")
```