Work It

Authors: Madeline Mianzo, Sarah Tan, Zixuan Zou: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—An entertaining, motivating, at-home exercise game, using real-time, full-body motion tracking to obtain user positioning data as they complete exercises. The game will cater to a variety of fitness levels, ranging from beginner to advanced. These levels will vary the amount of repetitions and the length of workouts to account for different users' abilities. Each user will be evaluated on their performance based on their form accuracy and the number of repetitions they completed. The system involves a display, a camera that takes images, a TX2 Xavier board to process the images taken by the camera, and a Exercise Library of exercise GIFs that will be combined to generate the workouts.**

***Index Terms*—Computer Vision, OpenCV, Pose Detection, PyGame, Real Time Analysis, SQLite, Tensorflow Openpose, Workout**

1 INTRODUCTION

The continued spread of COVID-19 has required people to adjust their lifestyles to minimize in person interactions, including placing restrictions on gyms and fitness clubs. Many people have transitioned to working out at home, often via YouTube videos or workout apps, but these workouts commonly lack variety and lead to a progress plateau over time.

Our new fitness game—*Work It*—aims to customize workouts specific to the user in order to provide the best exercise plan for their abilities. The user will earn points based on the quality of their workout completion and be given a workout score to track their progress over time. Through varying workouts to avoid repetition as well as motivating for improvement, our game will decrease the chances of a progress plateau and make at-home workouts more exciting.

The game will have three different exercise types: arms, legs, and core. Since our game is focused on the user, it will initially evaluate the user's fitness level for each of those categories with a standard set of nine-three per category-exercises to provide the best possible full body workout. As they complete more workouts, the user will be re-evaluated and *Work It* will modify the user's workout to match their progress.

2 DESIGN REQUIREMENTS

2.1 Joint tracking/Key Point Detection

One of our main requirements is joint detection and tracking. Since it is essential to our project, we require the key point detection to be accurate enough. To meet this requirement, we tested the algorithm with images of different workout poses and compared the key point locations detected to the actual key point locations. Based on the results, we can know which poses are harder to be detected and will pre-process the frames from these poses to ensure that enough valid key points are detected.

2.2 Pose Comparison

For pose comparison, we want to ensure that the algorithm has 100% accuracy if the positions of the key points are detected accurately in the previous step. Since the key points of the body in TensorFlow OpenPose are represented as 2D coordinates in a Cartesian plane, the process of computing the angles between the limbs based on the coordinates is very straight forward. For front-view poses, we will simply compare the angles between the limbs; for side-view poses, we will only compare a specific set of angles because some limbs might be hidden behind the body. We will test the algorithm with both front-view and side-view images of different poses to ensure the accuracy of the comparison results.

2.3 Transfer Between Hardware

With our use of the Jetson TX2 for workout generation, score computation, image processing, and pose comparison, as well as the USB Webcam A1 for capturing video, it is crucial that we require efficient transfer of information between our hardware components. Additionally, we had initially intended on using AWS to store the workout library, but since we ended up storing it on the TX2 instead, we no longer need to factor in the transfer of information from AWS to the GPU. Instead, we need to test that our camera is reaching our desired frames per second (fps) when receiving video input of the user and sending these images to the GPU for processing. It will also be crucial that the video feed is displayed in time with the live input on the Jetson TX2.

2.4 Runtime

We require that the wait time for the score generation after the workout ends to be less than 1 minute. We will have several 15-second breaks between exercises, which will be used to mask the time it takes for image processing, posture analysis, and score computation. Assuming that our easiest workout generated will have at least 5 exercises, then there will always be at least 1 full minute in total break time. By testing TensorFlow OpenPose with some images, we know that the average runtime of pose estimation for one image is approximately 1.1 seconds on a laptop. Since we are only comparing certain frames from the user video, 1 minute of extra computation time throughout the workout should be enough.

If our latency is more than 1 minute, and the breaks between exercises aren't enough to mask our algorithms runtimes, we will also include stretching and a cool-down at the end of each workout. This segment of the workout will not be scored, and it's only functions will be to mitigate muscle cramping and stiffness [3] (for the user) and provide us with more computation time.

2.5 User Interface

Our game's user interface should be simple to navigate and inviting to use. The main purposes of the interface are to allow users to create accounts (to save their fitness progress), show users demonstration videos to mimic throughout a workout, and display a score based on how well they performed their workout.

The user is able to interact with the interface using a keyboard and mouse. To create an account for the game, users provide a unique username by typing it using the keyboard. Logging in requires the user to type in that same unique username. Pressing specific keys on the keyboard will allow the user to control previewing workouts, starting workouts, ending workouts, accessing their previous scores, and returning to the login screen.

The functionality of our user interface was tested by pressing keys on a keyboard and ensuring that the correct keys trigger the correct behaviors. The triggered behavior had a 100% execution rate when the correct keys are pressed.

3 ARCHITECTURE OVERVIEW

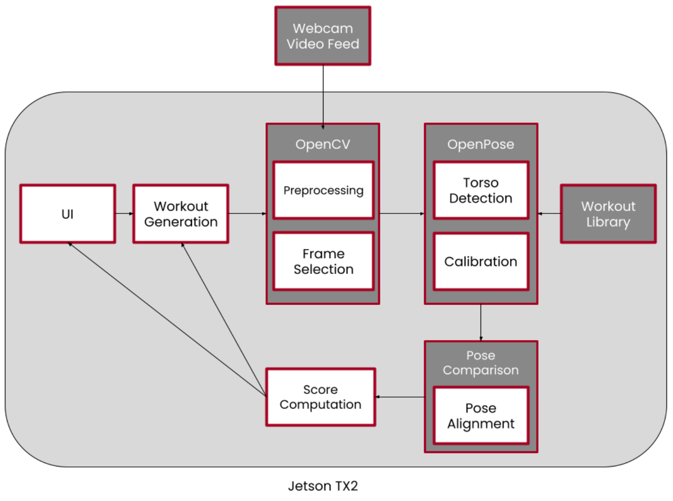


Figure 1: Block Diagram.

3.1 Python

We are using Python as our programming language. We are using OpenCV for image pre-processing, and PyGame for user interface, which are both compatible with Python. In addition, TensorFlow OpenPose is also developed in Python. Thus, it is reasonable for us to use Python for our project.

3.2 OpenCV

We will use OpenCV for image pre-processing in our project. After we have selected the certain key frames from the user's video captured by webcam, these key frames will be pre-processed by OpenCV before being sent to TensorFlow OpenPose for pose estimation. We will resize, rotate, or increase the color contrast for some frame to ensure the pose detection accuracy.

3.3 Tensorflow OpenPose

After we have selected and pre-processed the key frames of the user doing workouts, TensorFlow OpenPose will process these key frames for pose estimation. It will detect 18 key points on the user's pose and represent these key points as 2D coordinates in a Cartesian plane.

3.4 Pose Comparison

The key points detected by TensorFlow OpenPose will be used for pose comparison. We will use the extracted 2D coordinates to compute the angles between limbs. We can compute the difference between the angles of user's pose and the angles of the standard pose. For front-view poses, we can simply compare the angles; for side-view poses, we will only compare a specific set of angles because we might not be able to detect all key points since some limbs are hidden behind the body. Based on the comparison results, we will be able to determine how accurate the user's poses are.

3.5 Workout Library

We originally planned to use AWS to store our Exercise Library, but the Jetson TX2 board had enough memory to store the library, so we didn't use AWS. The Exercise Library contains GIFs of 72 different exercises. The GIFs are separated into 3 categories depending on the part of the body they target: arms, legs, and core. The GIFs are of 1 repetition of each exercise so they can be looped and used as demonstration videos for users to mimic during their workouts. The forms from the clips are also used as a standard to compare users' forms to.

This library only contains clips of female demonstrators. This shouldn't impact scoring or pose comparison much because we are using the angles between key points to determine form correctness for all exercises. The only impact it may have is on the user satisfaction of male users.

3.6 PyGame

Our user interface was created using the Python library, PyGame, with OpenCV. The interface starts on a general home screen. The user can press ”enter” to continue to a log in screen where they can enter a unique user id. Once entered, the user is taken to their user log in screen. Here, they can choose to either play the game or view their progress. The progress tracker is a graph of the user’s previous workout scores.

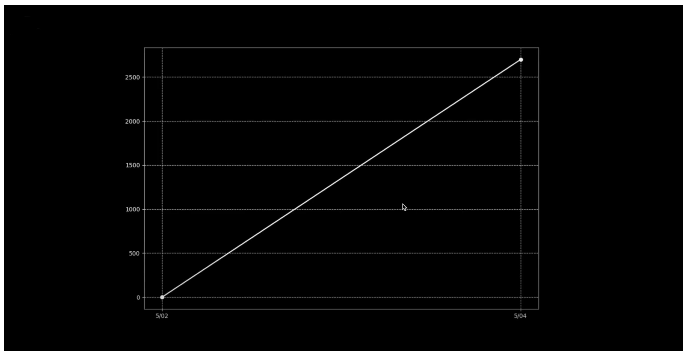


Figure 2: progress graph

Once viewed, the user can press ”delete” to return to their home screen. If the user chooses to play the game, they will be prompted to do an evaluation workout. The first screen in this sequence is an instructions page. This is to tell the user how to preview the exercises of the evaluation workout: they may use the left and right arrow keys to cycle through the exercises until they are ready to begin the evaluation. For each exercise, the user is shown the name of the current exercise and the number of repetitions required, the next exercise name and the number of repetitions required, a demonstration video of the current exercise, and a live camera feed:

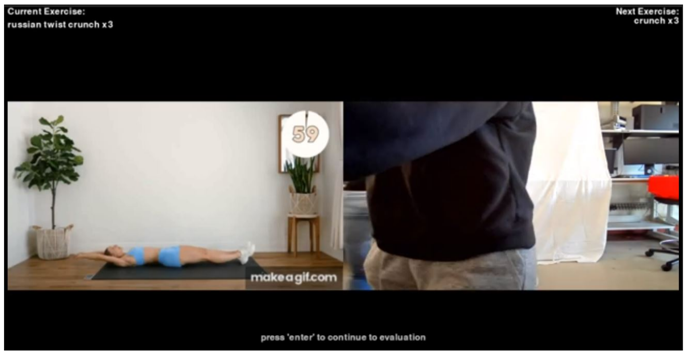


Figure 3: preview screen

The purpose of the previews is for the user to familiarize themselves with the exercises and learn how to perform them correctly. The demonstration video is formed by looping a GIF that’s pulled from the Exercise Library. During the preview stage, each GIF is looped infinitely, until the user proceeds to the next or previous exercise, or starts the actual evaluation workout.

The user can press enter to start the evaluation workout. They will see a disclaimer explaining how the workout is formatted: the exercises will be in the same order as shown in the preview. However, between each exercise, the user will have a 15 second rest period. Additionally, the user will have 15 seconds to position themselves at the start the workout. Throughout the workout, the screen will have the same format as during the preview: it’ll show the names and repetitions of the current and next exercises, a demo video, and a live camera feed. During a rest period, the demo video will be of the next exercise because the current one is the rest period. This will allow the user to prepare for it by positioning themselves to start the exercise at the end of the rest period. The rest period will also show a countdown of the time remaining.

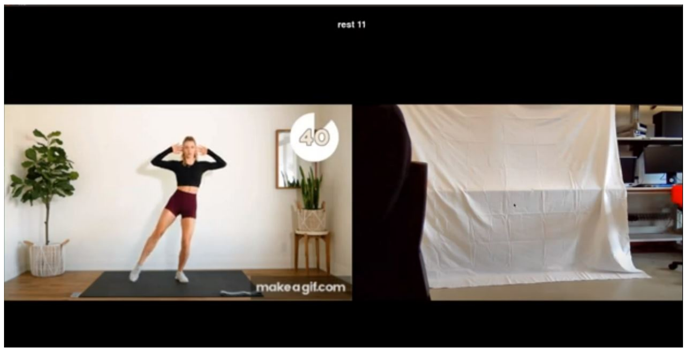


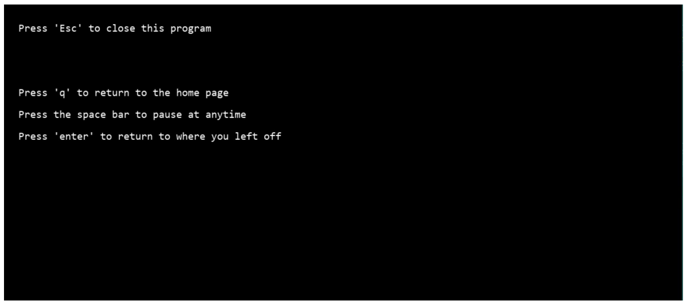
Figure 4: rest screen

During an exercise period, the demo video will be of the current exercise, looped for the number of repetitions specified on the screen.

Upon completion of the evaluation workout, the user will receive a score. This score is based on how closely they were able to match the demonstration video in timing and form. This score is automatically saved to the user’s profile so it can be displayed on the user’s progress graph the next time they check it. Based on this score, the program will generate a personalized workout for the user. Once this workout is generated, the user can start their workout.

This workout sequence works the same way as the evaluation workout. First, the user will preview the exercises in the workout. Then, they’ll see a reminder of the rest periods between their exercises of the actual workout, a 15 second readying period, and then the workout sequence. Upon completion of the workout, the user will receive a score, which is also automatically saved to the user’s profile. After this score is shown, the user can choose to return to the home page or quit the game completely.

Throughout the game, users can receive help with navigating the screens by pressing ”h”. This brings them to the screen shown in figure 5:



Press 'Esc' to close this program

Press 'q' to return to the home page
Press the space bar to pause at anytime
Press 'enter' to return to where you left off

Figure 5: help screen

The only portions of the game that don't support the help and pause functions are during the workouts (evaluation and personalized workout). This is to prevent the user from taking extra breaks. However, they still have the choice to abandon their workouts by either returning to the home screen ("q") or exiting the game completely ("esc").

3.7 SQLite

SQLite was used to store information about the user's workout scores so that they would be able to view progress graphs. The information and the date they completed the workout are stored in a table, and new scores are added to the user's specific information to update progress over time. Once the user has reached five workout completions, aka they have five available scores to them, earlier scores will be disposed of as new workouts are performed. The user can at most view their five most recent scores.

To see their personal progress chart, the user must enter their name into a text box. This name is used to look up their stored information, and must be typed the same way each time to receive their correct information.

4 DESIGN TRADE STUDIES

4.1 Jetson TX2

The Jetson TX2 is a fast, power efficient GPU that will benefit the image processing requirement of our project. The TX2 is preferable to the Jetson Nano, another board that we considered, because it offers 2.5 times the performance using only 7.5 Watts. The TX2 family is ideal for real-time processing applications where bandwidth and latency can be issues, which is very suitable for our product's requirements.[2] Additionally, OpenPose has hardware compatibility with the NVIDIA CUDA cores, which is the technology used in the TX2.

The other two Jetson models that we considered for use on our project were the Jetson Xavier NX and the Jetson AGX Xavier. However, due to our other project requirements we decided against those two models. In a comparison looking at processed frames per second for pose estimation using OpenPose, the Jetson TX2 has the capability to process 34 fps, while the Xavier NX processes 239 fps and the AGX Xavier can process 439 fps.[4] Though the 239 and 439 are impressive, they are much more powerful than anything we would need for this project. We plan to analyze around 10 fps for our pose comparison, for which the Jetson TX2 would meet our requirements. Additionally, our camera, like many standard USB webcams, captures video at 30 fps, so the capabilities of the NX or the AGX are far beyond what is necessary for the scale of our project.

Additionally, looking at our project budget, we were able to get the TX2 developer kit from capstone inventory, so we spent $0 on our board. The AGX developer kit, in comparison, would have been $699, exceeding our budget and leaving no room for any other purchases. The NX developer kit, on the other hand, would be within our budget at $399, but we decided that the increased performance was not worth the money that we would need to spend, especially when those performance increases aren't necessary for the success of our game.

We used a GPU for the pose analysis to reduce the amount of computation required by the CPU. Since we required real-time processing, it is most efficient for our GPU to do the image processing work instead of trying to run our application on only a laptop. We wanted low latency in our performance to provide user satisfaction.

4.2 Webcam

We chose our webcam requirements based on a trade-off between image processing speed and how much of a user's movements we capture (number of frames). Real-time image processing is as important as capturing key forms of an exercise in our game. To determine the ideal frame rate that would satisfy both of the aforementioned requirements, we compared the forms captured for running and walking at different frame rates[1]. These were chosen as the representatives of slow (ex: slow mountain climbers, push ups) and fast exercises (ex: jump squat).

At 30 fps, the typical video frame rate, tiny changes were captured for both exercises. These captured changes weren't large enough to justify the number of frames we would have to analyze. At 10 fps, larger changes were captured, but the changes were of key forms of each of the running and walking exercises. And, at 1 fps, a single repetition of either exercise couldn't be captured.

From these results, we concluded that around 10 fps would be enough to capture the major parts of most exercises. This lower frame rate will translate to fewer images to process, which will help us reach our real-time image processing goal. By this metric, we wanted a camera that exceeds 10 fps, preferably recording in the 20-50 fps range. The IFROO 1080p USB Webcam A1 model offers 30 fps, which suits our needs. It also offers an 82 degree non-fisheye lens, which will capture a large enough field of view or the user.

Initially we had looked at a variety of CSI camera options, but the USB webcam will provide us with more than enough fps, while costing at least $100 less than the CSI cameras, so it is more suitable for our game.

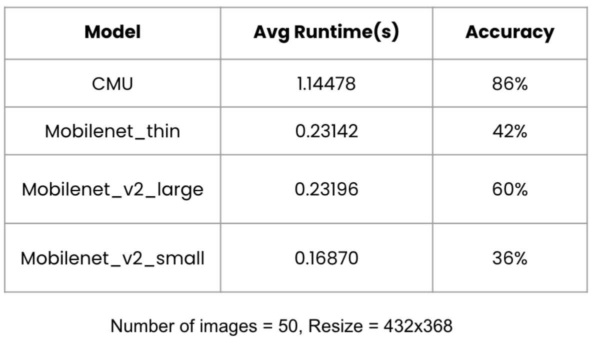
4.3 Tensorflow OpenPose

At the beginning, we planned to use OpenPose for pose estimation. It is a real-time system developed by CMU Perceptual Computing Lab to jointly detect human body key points for images and videos. It supports hand and foot key point detection as well. However, after installing OpenPose and testing it on a laptop, we found that the frame rate was lower than 0.1 fps when a video was processed, and the runtime of pose estimation for one image was approximately 9 seconds. Since we want low latency for our project, we switched from OpenPose to TensorFlow OpenPose.

TensorFlow OpenPose is a library built based on the original OpenPose. It includes the same model for pose estimation and takes out some features, such as key points detection for hand and foot. TensorFlow OpenPose is not as detailed as OpenPose but is faster. The average runtime for TensorFlow OpenPose to process one image is about 1.1 seconds on a laptop. Since we will not being using key points on hand or foot for our pose comparison, and we want low latency, TensorFlow OpenPose is a better choice for us.

4.4 CMU Model

TensorFlow OpenPose library includes four different models for pose estimation: CMU, Mobilenet_thin, Mobilenet_v2_large, and Mobilenet_v2_small. We ran a set of 50 images with each model to test the runtime and accuracy. We manually evaluated the accuracy of key point positions detected by TensorFlow OpenPose because the sample size is not too large. We considered the detection result for a image to be accurate if all visible joints of the body have been correctly detected.



Model	Avg Runtime(s)	Accuracy
CMU	1.14478	86%
Mobilenet_thin	0.23142	42%
Mobilenet_v2_large	0.23196	60%
Mobilenet_v2_small	0.16870	36%

Number of images = 50, Resize = 432x368

Figure 6: Test with different models.

We found that the CMU model works pretty well overall. It sometimes makes mistakes when the background is not clean, the person is lying down, or some limbs are hidden behind the body. The Mobilenet models are much faster compared to the CMU model, but for some images they are only able to detect parts of body, or are missing out the key points completely.

Based on the results, it appears that the CMU model is obviously the most accurate one; since we want high accuracy for our project, we choose to use the CMU model though it takes longer than the other models.

4.5 PyGame

Our user interface is important to our game because it significantly influences a user's experience. An inviting, easy to use interface will increase a user's satisfaction.

Since we chose Python as the language for our project, we considered using Tkinter as well as PyGame for the creation of our user interface. We eliminated Tkinter in favor of PyGame because Tkinter is simpler and doesn't have as many builtin features as PyGame. PyGame is also meant to simplify the game-making process, which better matches our goal of creating an exercise game.

5 SYSTEM DESCRIPTION

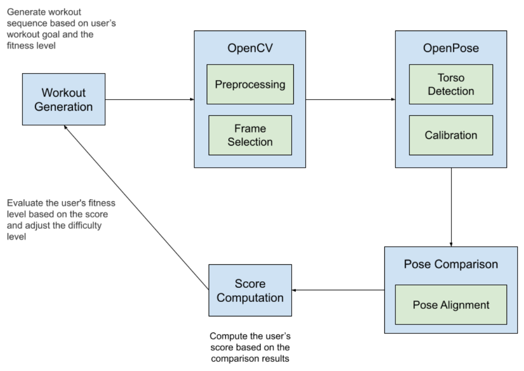

Generate workout sequence based on user's workout goal and the fitness level
Workout Generation
OpenCV
Preprocessing
Frame Selection
OpenPose
Torso Detection
Calibration
Evaluate the user's fitness level based on the score and adjust the difficulty level
Score Computation
Pose Comparison
Pose Alignment
Compute the user's score based on the comparison results

Figure 7: Software block diagram.

5.1 Workout Metrics Library

For each exercise in our workout library, we first select 2 to 4 key frames from it. These key frames should be the most important ones for the exercise. For example, for side lunges, we select two frames, one with the person doing the lunge on the left side, the other one with the person doing the lunge on the right side. The poses in all the other frames are not important for other pose analysis, since the person is either standing, or moving from one side to the other. We only need information of the two selected frames to determine how accurate the users' poses are. Thus, We only compare users' poses with the ones in these specific frames.

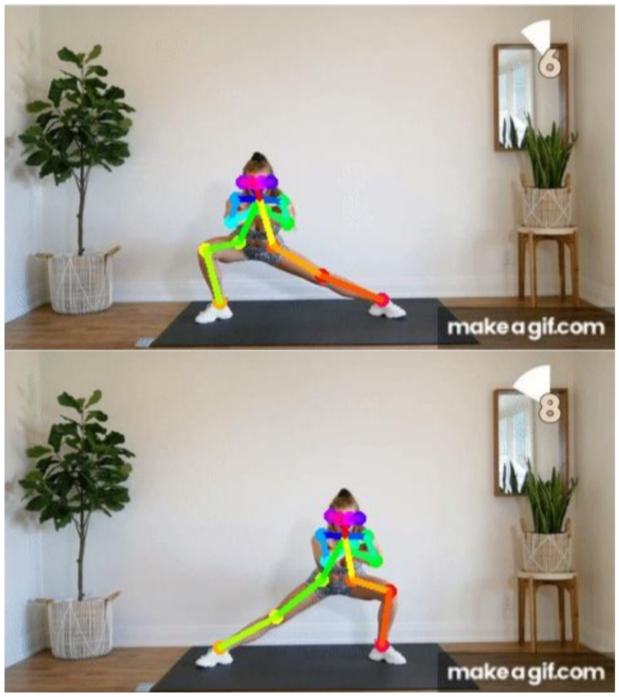


Figure 8: Key frames of side lunges.

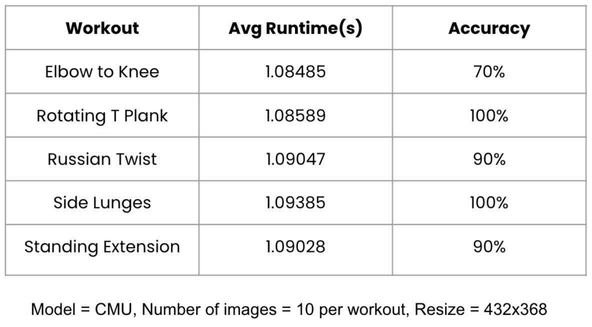
We run TensorFlow OpenPose with these selected frames, and save the timestamp, coordinate, and angle information for each exercise. This metrics library is used when we select frames from the user video input and compute the angle differences.

5.2 Image Pre-processing

We use OpenCV for image pre-processing before feeding the frames into TensorFlow OpenPose for pose estimation. We first get the timestamps of the frames we want to select by searching the workout name in the metrics library. Then, we will select 10 frames around that timestamp and feed them into OpenPose for pose detection. After we get the angle information for each frame from OpenPose, we will use the set of angles closest to the standard angles to compute the user's score. We do this to allow some speed difference between the user and the demonstration video. The user can do the exercises at a slight faster or slower speed but still be considered accurate.

We also need to pre-process the frames to ensure the accuracy of pose estimation. We tested TensorFlow OpenPose with images from 5 different workout exercises, 10 images per workout, to understand which poses TensorFlow OpenPose have difficulties detecting.

From the test results, as shown on figure 9 below, it appears that poses with twisted body and poses lying down are harder for TensorFlow OpenPose to detect correctly. In that case, we will try to increase the detection accuracy by pre-processing the frames. For instance, we will rotate the frames with the person lying down, such as the poses from the Elbow to Knee exercise, since it would be easier for TensorFlow OpenPose to perform estimation on poses with the human head at the top of the image.



Workout	Avg Runtime(s)	Accuracy
Elbow to Knee	1.08485	70%
Rotating T Plank	1.08589	100%
Russian Twist	1.09047	90%
Side Lunges	1.09385	100%
Standing Extension	1.09028	90%

Model = CMU, Number of images = 10 per workout, Resize = 432x368

Figure 9: Test with different workouts.

5.3 Pose Alignment

After TensorFlow OpenPose performs pose estimation on the user's poses, we need to align the poses before we proceed to pose comparison. We determine the differences between user's pose and standard pose by computing the angles between limbs. Since angles would not be affected by resizing or rotating, and we do require the users to place the camera at a specific angle, we do not need do pose alignment in space.

However, we do need to perform pose alignment in time. We will penalize the users for doing the exercises significantly faster or slower than the standard exercise clip shown to them on the screen, but we want to allow some tiny speed differences. If the user does the exercises slightly faster or slower than the standard one, for example, the difference is within 1 second, the score will not be affected. Thus, pose alignment in time is needed. Since we are selecting 10 frames around the key frames, by running TensorFlow OpenPose on each of them, and comparing the angles of each with the standard angles, we will find the frame with the best match pose and use it for score computation. Thus, users are allowed to have some minor speed difference with the demonstration video.

5.4 Pose Comparison

TensorFlow OpenPose will output 18 key points after the torso detection. We use the positions of these key points to perform pose comparison. The key points positions are represented as 2D coordinates in a Cartesian plane, so we can compute the angles between the limbs by simply computing the angles between the vectors defined by these coordinates.

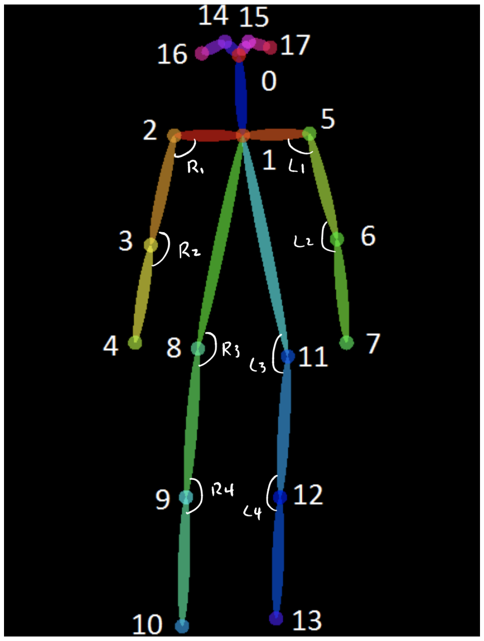


Figure 10: OpenPose angles.

As shown on figure 10 above, we want to compute and compare a total of 8 angles: R1, L1, R2... and L4. For instance, if we want to get angle R1, we will first compute the vectors from key point 2 to 3 and 2 to 0, and compute the angle between these two vectors.

The difference between the user pose and standard workout pose can be determined by computing the difference between the angles. For front-view poses, we compare all the valid angles. For side-view poses, since some limbs might be hidden behind the body, we only compare a specific set of angles for certain workout poses. We will later use the angle differences for score computation.

5.5 Workout Generation

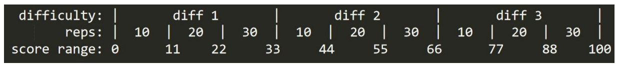
First, the user must complete an evaluation workout. This workout is static and consists of 3 pre-selected exercises from each exercise category (arms, legs, core). Each of the 3 exercises of each category is of a different difficulty level (1, 2, 3). The exercises are pulled from the Exercise Library stored on the TX2 board.

The Exercise Library is composed of GIFs of a single repetitions of exercises. Each exercise is labeled in the format of

[exercise category][difficulty rating]_[id]

The exercise category is one of {"arm", "leg", "core"}, the difficulty rating is one of {1, 2, 3}, and the id is to differentiate exercises of the same difficulty level and exercise category. The difficulty ratings were subjectively chosen by my group mates and I, with 1 representing the easiest exercises and 3 representing the hardest exercises.

Upon completion of the evaluation workout, Work It computes a score for each exercise category for the user. So, the user will get separate scores for arms, legs, and core (the sum is the score shown to the user). Comparing the categorical score the user received to the total possible categorical score they could've received generates a percentage out of 100. The difficulty range of the exercises and the number of repetitions chosen for each category is based on its percentage:



```
 difficulty: |        diff 1        |        diff 2        |        diff 3        |
       reps: |  10  |  20  |  30  |  10  |  20  |  30  |  10  |  20  |  30  |
score range: 0      11     22     33     44     55     66     77     88    100
```

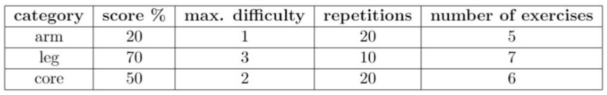
Figure 11: difficulty ranges (top) and number of repetitions (middle) based on score percentage (bottom)

If the percentage is between 0 and 33, the maximum exercise difficulty will be set to 1. If the percentage is between 34 and 66, the maximum exercise difficulty will be set to 2. If the percentage is between 67 and 100, the maximum exercise difficulty will be set to 3.

If the percentage is between [0, 11], [33, 44], or [66, 77], the number of repetitions assigned to each exercise will be set to 10. If the percentage is between [12, 22], [45, 55], or [78, 88], the number of repetitions assigned to each exercise will be set to 20. If the percentage is between [23, 33], [56, 66], or [89, 100], the number of repetitions assigned to each exercise will be set to 30.

After the difficulty ranges and number of repetitions are determined for each category, exercises that meet those requirements are randomly chosen. Since the GIFs stored in the Exercise Library are named based on category, difficulty, and id, extracting valid candidates is simple. Only exercises starting with "[category][value from determined difficulty range]_" are considered. The number of repetitions for all exercises in the category is set to the determined number of repetitions. The ids are determined by randomly choosing ids from the valid candidates. The total number of ids generated per category is 4 more than the maximum determined difficulty. This is to ensure that at least 5 exercises, and no more than 7 exercises, are included in the workout. We want to push the user to improve their fitness level, but we don't want them to be discouraged by an extremely difficult workout.

As an example, suppose a user's scores for the arm, leg, and core categories be 20%, 70%, and 50% of the total scores they could've received from the evaluation workout. Then, for each category the requirements would be set as the values listed below, in figure 12:



category	score %	max. difficulty	repetitions	number of exercises
arm	20	1	20	5
leg	70	3	10	7
core	50	2	20	6

Figure 12: category and score percentage $\longrightarrow$ maximum difficulty, number of repetitions, and number of exercises

So, our algorithm could choose 5 arm exercises of difficulty level 1, 7 leg exercises of any difficulty level, and 6 core exercises of difficulty levels 1 or 2.

5.6 Score Computation

Score computation is based on pose analysis and the number of repetitions for the exercise. We use an "all or nothing" approach to scoring each repetition of each exercise. The user must perform each repetition accurately, with a total angle difference below a certain threshold to gain any points. This threshold was manually determined for each exercise in our Exercise Library. The number of points the user can gain for each repetition is based on the difficulty of the exercise: exercise_difficulty * 100.

Repetition scores are summed together to get the total scores for each category. These categorical totals are summed to get the overall total workout score. This total workout score is what is shown to the user at the end of their workout.

5.7 User Accounts

SQLite is used for robust storage of user data. User's are identified by the name that they enter at the start of the game. This name is then used to look up the user's scores that are stored from previous workouts.

By saving a workout score along with the date completed, we efficiently track progress overtime of the user's *Work It* usage and display it in a chart that is easy for the user to interpret. At the completion of a workout, the score is entered into the user's row of the table. If they have less than five workouts saved, then the new score will just be added and the game will proceed from there. If instead they have more than five, their earliest score will be removed and all of the scores shifted, so that only five scores are kept at a time. This serves the purpose of keeping the graph easy to interpret.

6 TEST & VALIDATION

6.1 Hardware Performance

To keep our game engaging, we decided that keeping the time required for image processing down was of utmost importance. Due to our inclusion of 15 second rest breaks throughout the workout, we decided that as long as the time needed for image processing stayed beneath 1 minute, the rest breaks would account for that time and the user wouldn't be dissatisfied with our game's performance. Thus, we set our requirement for hardware performance at less than 1 minute.

In our testing of 6 repetitions on a sample size of 20 exercises, our pose comparison algorithm returned the angle computations immediately, followed by a 2 second delay with score computation. This put our Hardware Performance at a much better value than we expected, with computation taking less than 3 seconds total. This value exceeded our expectations for this category.

6.2 OpenPose Detection

For our pose detection testing, we looked at which frame size for the images will deliver the most accuracy. We expected an accuracy rate of 90% for the frame size that we would use for our game.

Using a sample size of 50 images, we ran OpenPose on frames from our exercise GIFs from our workout library to determine how well the key points on the model were being tracked. We found that a lower resolution for the images results in a shorter runtime but lower accuracy. The highest resolution sometimes led to more errors, either from our algorithm picking up other 'people' in the background, or not being able to detect the whole body. The 432x368 frame size was calculated to have an accuracy of 90%, meeting our requirement. These results are displayed in the table below.

Frame Size	Avg Accuracy
160x96	72%
432x368	90%
656x432	88%

Figure 13: Frame Size Accuracy Values

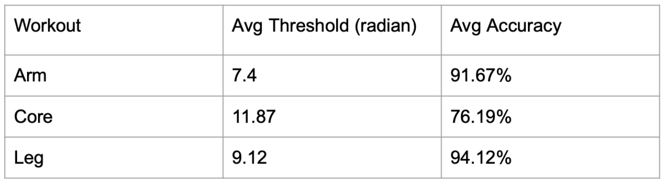
6.3 Pose Alignment

To test pose alignment, we first did some exercises slightly faster or slower than the demonstration video. We still got nearly perfect scores, which means that some small speed differences are allowed. Users can be slightly faster or slower than the demonstration video shown on the screen, and their workouts will still be considered accurate and complete.

We then did the exercises significantly faster or slower than the demonstration video, and the resulting scores were around 30% to 50% of the max score, which is what we want. Since we were too slow or too fast, we missed the 10-frame detection interval for each key frame, and were penalized for not following the demonstration video.

6.4 Pose Comparison

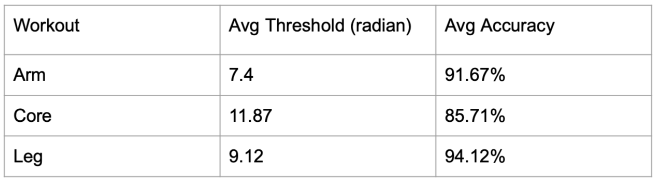
We developed the concept of a threshold for our angle comparison in our pose comparison evaluation. These threshold tests were done with a sample size of about 40 exercises. In our testing, we found that there would always be some angle differences between the user and demo, even if the user does their best in matching the demo video. To determine an appropriate threshold for each exercise, we tested the exercise by running through it 4-6 times. From there we computed a threshold (in radians) for which the user will receive full points for the repetition if their angle difference falls below the value. Each exercise has a different calculated threshold associated with it. After establishing an appropriate threshold value, we ran accuracy tests with a sample size of about 40 exercises. We tested exercises from the 3 categories and found that the detection accuracy for core exercises is lower than the others. The table below displays the accuracy values for each category of exercises.



Workout	Avg Threshold (radian)	Avg Accuracy
Arm	7.4	91.67%
Core	11.87	76.19%
Leg	9.12	94.12%

Figure 14: Accuracy Values by Exercise Type

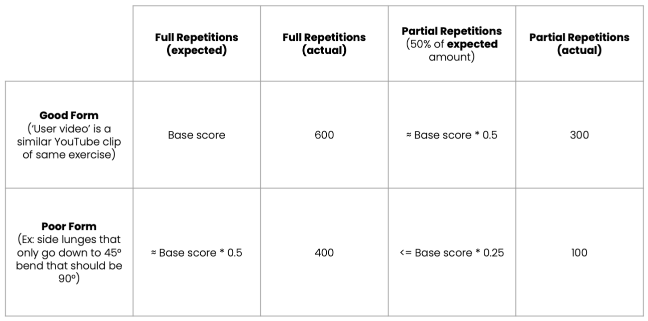
The core exercises often have the user lying down with limbs hidden behind the body which makes it more difficult for us to analyze the user and demo video's poses. In some frames some limbs were not detected and in others the user was not detected at all. Since our goal for accuracy of pose comparison was 90%, we decided to remove a few of the core exercises that were causing these issues from our workout library. We decided to value accuracy over a larger core library, and since these particular exercises were not crucial to the overall success of our game, we decided removing them was worth it. The updated accuracy values are displayed in the table below. These accuracy values meet our expected 90% standard.



Workout	Avg Threshold (radian)	Avg Accuracy
Arm	7.4	91.67%
Core	11.87	85.71%
Leg	9.12	94.12%

Figure 15: Accuracy - Modified Core Library

6.5 Score Computation



	Full Repetitions (expected)	**Full Repetitions (actual)**	**Partial Repetitions** (50% of **expected** amount)	**Partial Repetitions (actual)**
Good Form ('User video' is a similar YouTube clip of same exercise)	Base score	600	≈ Base score * 0.5	300
Poor Form (Ex: side lunges that only go down to 45º bend that should be 90º)	≈ Base score * 0.5	400	<= Base score * 0.25	100

Figure 16: Score Computation Testing

To verify that our score computation changed with user performance, we performed 5 rounds of 6 reps of side lunges. Since the scoring algorithm is the same for all exercises, we decided that testing one exercise would be enough to verify that the scores change correctly with user performance.

For the first round, we performed the lunges as perfectly as we could. This generated full points (600) for the exercise, since the difficulty for this exercise is 1 and we completed all reps with few errors.

In the second round, we completed only half the repetitions. However, for the ones we performed, we imitated the demo as closely as we could. The resulting score was 300, or 50% of the highest possible score. This was expected because we only did half the reps.

For the poor form and full repetition category, we performed 4 repetitions with form that happened to be decent (angle difference was still below the threshold for the exercise), so 400 points were awarded for the exercise.

And for poor form and partial repetitions, only one rep was beneath the threshold so only 100 points were awarded.

Our final round was not performing any reps, and the resulting score was 0.

7 PROJECT MANAGEMENT

7.1 Schedule

Our schedule was broken into three phases in order to distinguish generally between setup, working on individual components, and integration and testing. Throughout the semester, we updated our gantt chart to allow more time for setup of all of the components from phase 1, as well as allowing more time for adding gifs to our workout library.

7.2 Team Member Responsibilities

Zixuan lead image processing and analysis using Tensorflow OpenPose. She created the pose comparison, angle difference computation, and frame selection algorithms.

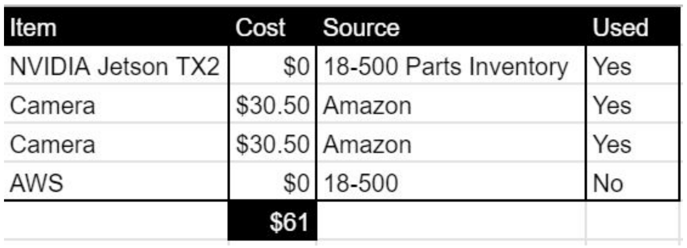
Maddie lead the integration of the software components and the TX2 board. She was also in charge of creating user accounts.

Sarah lead the creation of the user interface, workout generation algorithm, and scoring algorithm.

Maddie and Sarah contributed clips to the Exercise Library. This included manually ranking the difficulty of the exercises and creating the gifs.

We all tested and debugged our algorithms for pose comparison, workout generation, and score generation. We all also performed user testing to ensure that our game functioned correctly.

7.3 Budget



Item	Cost	Source	Used
NVIDIA Jetson TX2	$0	18-500 Parts Inventory	Yes
Camera	$30.50	Amazon	Yes
Camera	$30.50	Amazon	Yes
AWS	$0	18-500	No
	$61		

Figure 17: Bill of Materials

We borrowed our Jetson TX2 board from the Capstone course and bought 2 webcams from Amazon.

We originally planned on using only 1 webcam. However, after integrating our pose comparison code with our TX2 board, we realized the algorithms functioned differently on the board than on a laptop. We figured out that the different frame rates of the laptop camera and our webcam generated different behavior, so we ordered a second webcam to allow for local testing/debugging on a laptop. Local testing was important because Zixuan was working remotely, so she wasn't able to debug the code on the board.

We also originally planned on using AWS to store our Exercise Library, so we received AWS credits from the Capstone course. However, we found that the library was able to fit on the TX2 board, so we didn't end up using the AWS credits.

7.4 Risk Management

One of the risks we encountered at the beginning is high latency. The runtime of OpenPose was longer than we wanted. Since we want to provide real-time feedback to the user by showing the OpenPose result on the screen, we switched to TensorFlow OpenPose, which cannot detect the fingers but is faster than the original OpenPose. We also choose not to run TensorFlow OpenPose on all frames from the user input; instead, we only run pose detection on the 10 frames around the key frames and find the best-match pose among them, because we only need these frames for pose analysis.

Another risk is that TensorFlow OpenPose may not be able to detect all key points on the user. After some testing, we learned that both clothing and background can affect the estimation accuracy. We require the users to wear tighter clothes and have a clean background. TensorFlow OpenPose also tends to make mistakes when the person is lying down. We pre-process the frames, such as rotating and resizing, for certain poses to increase the accuracy.

We want the score to reflect the user's performance accurately enough, so we tested it with the good form vs. the poor form, doing full repetitions vs. doing only partial repetitions of the exercise (as mentioned in the score computation section). The score should reflect the corresponding level of completion and accuracy. By doing this test, we were able to know whether the scoring algorithm works and make adjustments to it.

8 ETHICAL ISSUES

Some possible ethical issues arise from our use of a camera and our collection of fitness scores.

A camera is used to provide a live feed of a user in a room. It is only meant to be on when the user is using the program. However, if a malicious attacker were to gain access to the program remotely, they could also gain access to the camera feed. This could result in images or videos taken in situations that a user wouldn't want them to be taken in.

Another possible issue is from the data we collect about users. If our program is hacked, the stored fitness data about users could be leaked. This information could be used against the users by corporations such as insurance companies. If a user has a history of low workout scores, insurance companies could deny health insurance to them, citing the low scores as proof of a higher likelihood of pre-existing health conditions. If the insurance companies are able to save our angle difference data, then they could also gain insight into the form of the user. This could possibly be used to determine susceptibility to certain physical ailments, such as back or joint issues. If the user habitually performs exercises using incorrect form, the angle difference information could allow companies to predict pain in body parts affected by the incorrect form.

9 RELATED WORK

Two similar Capstone projects are *Falcon: Pro Gym Assistant*, from Fall 2020, and *Virtual Yoga Coach*, from Spring 2019. *Falcon: Pro Gym Assistant* customizes workouts for it's users and provides periodic feedback on their posture. So, it's similar to *Work It* because it also generates customized workouts for their users. However, it differs in how it process it's images (use an FPGA) and in when it provides feedback to it's users (periodically throughout the workout).

Virtual Yoga Coach provides form feedback for users' on a limited number of yoga poses. Although the exercises it supports are different from the ones we support, it's still similar to *Work It* because it also requires image processing and analysis, and uses the same software (Tensorflow OpenPose). Because it uses the same software libraries, and it's goals were similar to ours, we were able to guide some of our design choices by looking at what the creators of this project did. This is how we decided to use angles in our pose detection algorithm.

10 SUMMARY

Our system was able to meet most of our design specifications. Our hardware performance was much faster than we originally anticipated (<5 seconds vs 1 min). Our pose alignment worked with 90% accuracy, pose comparison averaged out to 90% accuracy, and score computation changed according to user form accuracy.

The only design specification we didn't meet was OpenPose detection requirement of 90%. We ended with a 86% rate of detection, but this is because of the exercise forms we chose. Some of the forms have hidden limbs, or require twisting. These make parts of the body not visible to the camera, and thus not able to be detected by OpenPose.

If we had more time, we would've pruned our Exercise Library a bit more. We started removing exercises that were not ideal for OpenPose detection, and replaced a few of them with more OpenPose-friendly ones. This is why we only have 72 exercises and not more. If we optimized our Exercise Library, our OpenPose detection requirement would've been met.

10.1 Lessons Learned

We learned about the importance of communication when working remotely. Even though we met and discussed our work during our lab meetings, we learned that we needed more frequent communication through extra zoom meetings or slack.

We also learned to allocate more time than we thought we would need for everything. Especially for integration. Even if our code worked locally on our own laptops, the dependencies on the board were not the same, so the behavior of the code was sometimes different. Also, the board can only support certain dependencies, so testing changes in code on the board often helps keep dependency errors minimal.

References

[1] IPVM. "Frame Rate Guide for Video Surveillance". In: (Jan. 2021). URL: `"https://ipvm.com/reports/frame-rate-surveillance-guide"`.

[2] NVIDIA. "Jetson TX2". In: (). URL: `"https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/"`.

[3] American Heart Association editorial staff. "Warm Up, Cool Down". In: (Sept. 2014). URL: `"https://www.heart.org/en/healthy-living/fitness/fitness-basics/warm-up-cool-down"`.

[4] Elaine Wu. "Compare NVIDIA Jetson Xavier NX with Jetson TX2 Developer Kits". In: (2020). URL: `"https://www.seeedstudio.com/blog/2020/05/19/compare-nvidia-jetson-xavier-nx-with-jetson-tx2-developer-kits/"`.

Appendix A

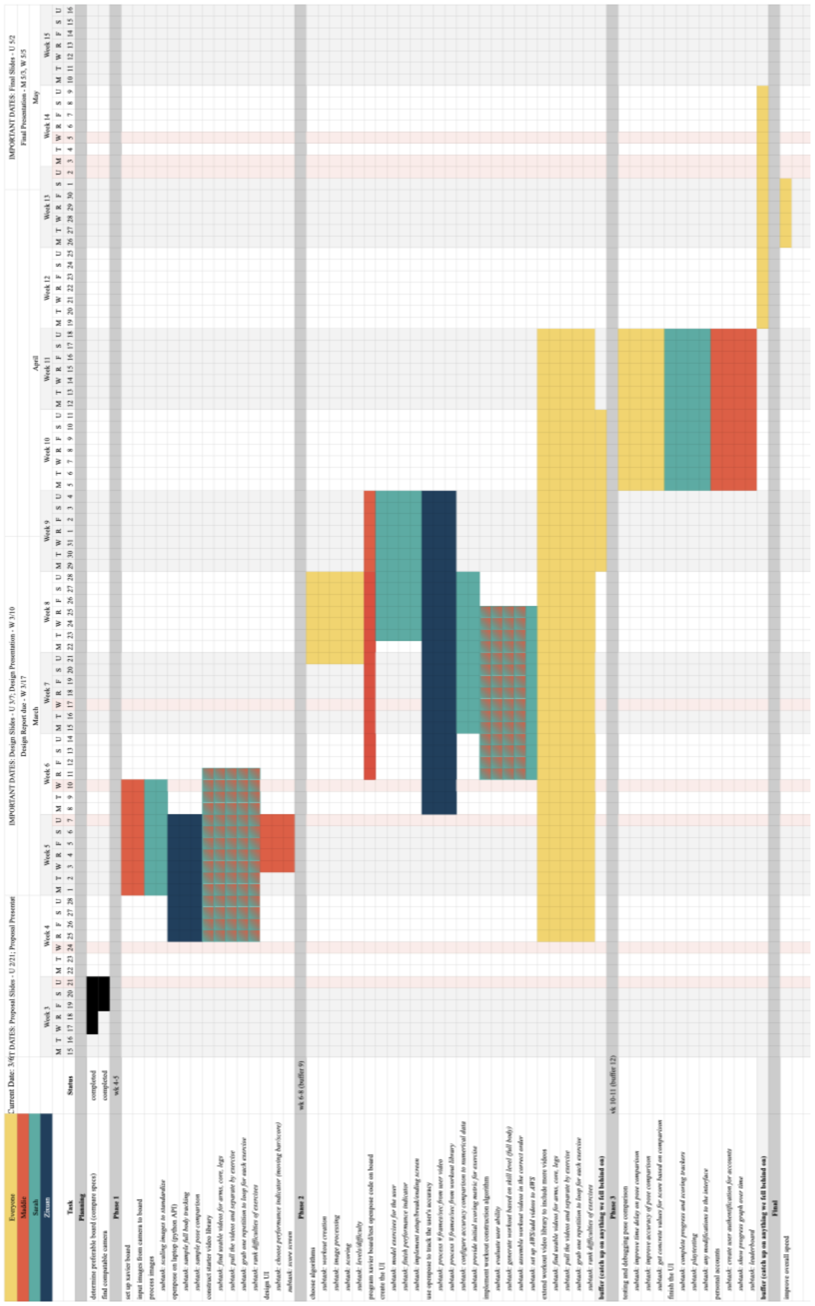


Figure 18: Gantt Chart