Work It

Authors: Maddie 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. 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 compare with friends or track their progress over time. Through progress motivation and tracking, as well as varying workouts to avoid repetition, our game will decrease the chances of a progress plateau and make home workouts more exciting.

The game will have three different workout 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 exercises to provide the best possible full body workout. Over time, the user will be re-evaluated and Work It will modify the user's workouts 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 AWS on a laptop for the workout library, the Jetson TX2 for image processing and pose comparison, and the USB Webcam A1 for capturing video, it is crucial that we require efficient transfer of information between our hardware components. We will 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.

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 for each exercise, and display a score based on how well they performed their workout.

The user should be able to interact with the interface solely through a keyboard. To create an account for the game, users should be able to uniquely name their accounts by typing it using the keyboard. Logging in will similarly only require the user to type in their unique username. Pressing specific keys on the keyboard will allow the user to control starting workouts, ending workouts, accessing their previous scores, and returning to the login screen.

The functionality of our user interface can be tested by pressing keys on a keyboard and ensuring that the correct keys trigger the correct behaviors. The intuitiveness of the interface can be tested by obtaining feedback from volunteers (our housemates and friends) about the ease of use of our interface.

3 ARCHITECTURE OVERVIEW

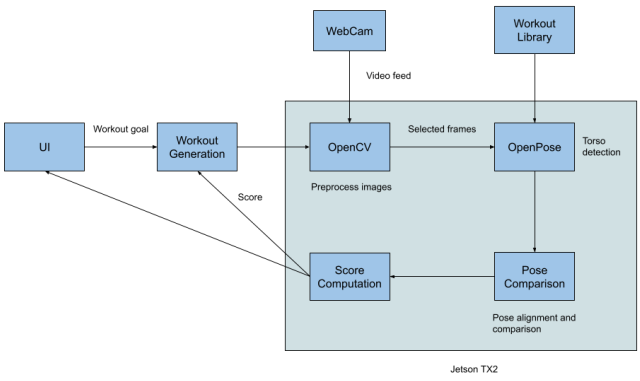

Figure 1: Block Diagram.

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. We will use OpenCV to select certain key frames from the user's video captured by webcam for pose comparison. These key frames will be pre-processed by OpenCV before being sent to TensorFlow OpenPose for pose estimation. The frames from certain workout poses need to be rotated as well 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

AWS will be used to store our exercise library. This library will contain clips of various exercises. The clips will be looped so they can be used as the demonstration videos for users to mimic during their workouts. The forms from the clips will also be used as a standard to compare users' forms to.

This library will only contain clips of female demonstrators. This aspect will only potentially affect the user experience of male users. However, there shouldn't be any impact on scoring or pose comparison because we are using the angles between key points to determine form correctness for all exercises. As a stretch goal, we want to add clips of male demonstrators for the exercises in our workout library.

3.6 PyGame

Our user interface will be created using the Python library, PyGame. The demonstrations of exercises will be composed of exercise clips taken from the Standard Exercise Library. These clips will be looped for the number of repetitions chosen by our workout generation algorithm.

The interface will look similar to figures 2 and 3 (below). They are modeled off of the layout of YouTube video workouts.

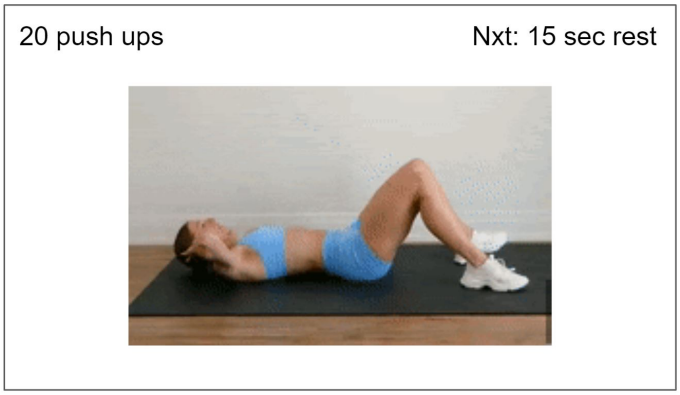

Figure 2: UI exercise demonstration page.

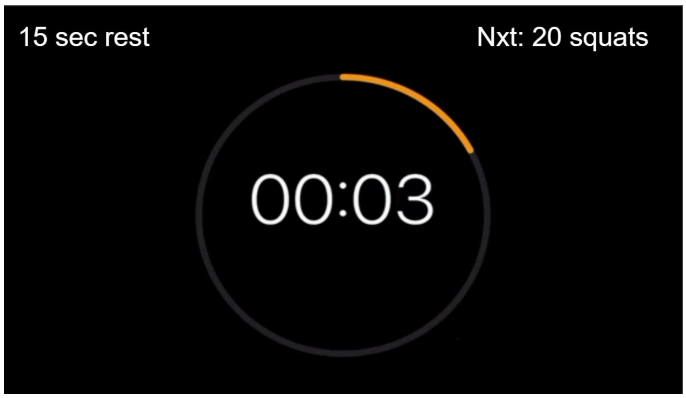

Figure 3: UI water break page.

3.7 SQLite

SQLite will be used to store previous workout scores for each user. This will allow users to see their progress over time.

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 want to use a GPU for the pose analysis to reduce the amount of computation required by the CPU. Since we require real-time processing, it would be most efficient for our GPU to do the image processing work instead of trying to run our application on only a laptop. We want 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 it to be extremely slow. 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 accurate as OpenPose but is much faster. The average runtime for TensorFlow OpenPose to process one image is about 1.1 seconds. Since we will not being using key points on hand or foot for our pose comparison, and we want low latency, TensorFlow OpenPose is clearly 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 4: 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 will 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

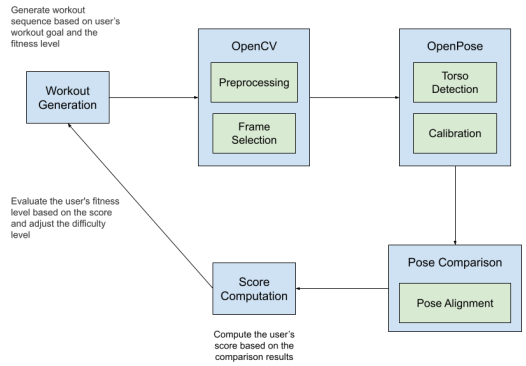

Figure 5: Software block diagram.

5.1 Image Pre-processing

We will use OpenCV for image pre-processing before feeding the frames into TensorFlow OpenPose for pose estimation. We will first select the key frames which we want to analyze from the video captured by webcam. Our plan is to select specific frames based on timestamps in OpenCV, and the timestamp for each exercise varies depending on the standard workout videos we are using from our workout library. We will select frames within 1 second around that timestamp to allow some speed differences.

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 im-

ages per workout, to understand which poses TensorFlow OpenPose have difficulties detecting.

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 6: Test with different workouts.

From the test results, as shown on figure 6, 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.

5.2 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. We are looking in to the Dynamic Time Warping method, an algorithm that measures similarity between two temporal sequences. We will look for the best alignments between two time series, and compare the frames of the poses that align with the standard poses.

5.3 Pose Comparison

TensorFlow OpenPose will output 18 key points after the torso detection. We will 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.

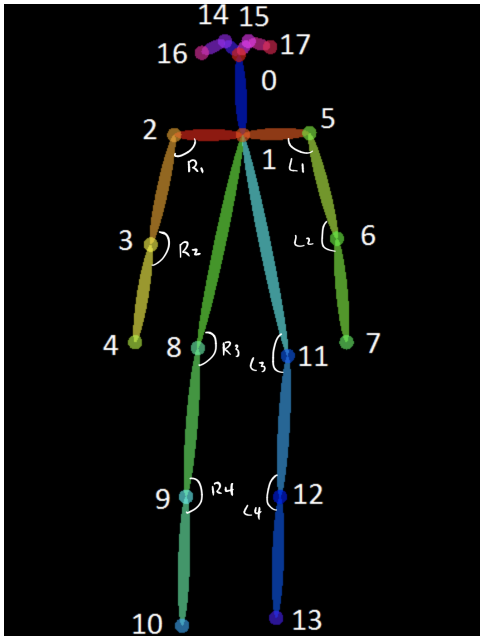

Figure 7: OpenPose angles.

As shown on figure 7 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 will compare all the valid angles. For side-view poses, since some limbs might be hidden behind the body, we will only compare a specific set of angles for certain workout poses. We will later use the angle differences for score computation.

5.4 Workout Generation

We will use AWS to store a library composed of single repetition clips of all of our potential exercises. Each exercise will be labeled by their name, and will be accordingly categorized by the type of exercise (arms, legs, core) and it's difficulty level.

Workout generation will work hand in hand with the pose comparison that we will be performing. Upon beginning the game, the user will perform a basic set of exercises (ex: for the core evaluation, users could be asked to perform 40 crunches, 20 russian twist crunches, 20 leg lowers, and 20 reverse crunches). Using pose comparison, Work It will compute a score for each of the arms, legs, and core requirements. This score will be scaled to determine the amount of repetitions that the user should complete in that category to build up their full body workout. For example, a score of 60/100 for core may have a scale factor of 3 and recommend a workout that includes 180 repetitions of core

exercises for which the user would strive for precision with the demo video.

After the required total repetitions is determined, we will randomize the selected exercises from within the library options. Repetitions of each selected exercise should range from 20-40, depending on difficulty. In our library we plan to assign each exercise a difficulty level of either easy, medium, or hard (1, 2, or 3), where, for instance, an easy exercise would be assigned closer to 40 repetitions while a hard exercise would be assigned 20 repetitions.

After our algorithm has specified which exercises are chosen at a specified number of repetitions, the selected exercise library videos will be looped for that amount of repetitions to be displayed on the screen for the user to follow as they complete that exercise.

5.5 Score Computation

Score computation will be based on pose analysis and the number of repetitions for the exercise. For a breakdown of a specific exercise, we can give each individual repetition a score ranging from 0-1, where a zero would be absolutely no accuracy between the user video and the demo video for each compared frame, and a one would require all angles between key points to match with very minimal error. Due to variance in body types, even with normalization and alignment of the user video, a very well performed exercise should score between .9 and 1 for each repetition.

Repetition scores will be summed together for the total exercise score, and that exercise score will be added to the overall total. Because there will be differences in lengths of workouts (variance of repetition amounts based on difficulty levels) we anticipate needing to scale these total scores by some scale factor in order to always provide a score out of a set total (ex. a full body workout score will always be given to the user out of 1000).

We prefer to give the user an integer score instead of a percentage for their performance since we are sticking to the idea of the game format. Similar to the video game Just Dance, an integer score will increase the ease at which the user can determine their performance and improvement.

5.6 User Accounts

SQLite will be used for robust storage of user data. We will be able to assign a user an ID and use this ID to look up all previous information regarding their past workouts.

By saving a workout score along with the date completed, we can efficiently track progress overtime of Work It usage and display it in a chart that is easy for the user to interpret. We will also use a combination of user IDs for the construction of a leader board. The top scores along with the date completed and the user account will be stored in the SQLite database for easy lookup when a user wants to view the current leader board. By using the SQLite database, we will be able to substitute in current high scores when they pass scores on the current leader board.

6 PROJECT MANAGEMENT

6.1 Schedule

Currently, we are transitioning between our setup phase and our first execution phase. We have chosen a list of videos from which to assemble our workout library, as well as done initial OpenPose testing using a laptop on still frames from a few of the workout videos. We have also begun setup of our TX2, though we still need to finish OpenPose installation of the physical board.

We are moving into execution, where we will now be assembling the library in AWS, as well as implementing our workout construction algorithm from those videos. From there, we will be analyzing the pose of two different videos of the same exercise to implement pose comparison and test the way we are evaluating the accuracy between poses. These steps will take place over the next few weeks.

6.2 Team Member Responsibilities

Even though we split our project tasks up among ourselves, we will all work together on the tasks. The assignment of responsibilities is more for leadership of the task.

Zixuan has been, and will continue, leading image processing and analysis using Tensorflow OpenPose. Maddie will lead programming and testing OpenPose on the Xavier board. She will also be in charge of creating user accounts and score tracking over time. Sarah will lead the creation of the user interface, the scoring algorithm, and the exercise library in AWS. Sarah and Maddie will work together to create the workout construction algorithm that will generate workouts based on users' fitness levels.

All of us will contribute clips to the exercise library. This includes manually ranking the difficulty of the exercises. We all will also test and debug our algorithms for pose comparison and score generation.

6.3 Budget

Item	Cost	Source
Jetson TX2 Xavier	$0	18-500 Parts Inventory
Camera	$30.50	Amazon
AWS	$0	18-500

6.4 Risk Management

One of the risks we might encounter is high latency. To reduce computation and runtime, we will only select and compare key frames from the user's workout video and use the 15-second breaks to process the images.

Another potential risk is that TensorFlow OpenPose may not be able to detect all key points on the user. After some testing, we learned that clothing and background can both affect the estimation accuracy. We will 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 will do some pre-processing,

such as rotating and resizing, for certain poses to increase the accuracy.

If the score generate is not able to reflect the performance accurately enough, we will test it with the good form vs. the poor form, doing full repetitions vs. doing only partial repetitions of the exercise. The score should reflect the corresponding level of completion and accuracy. Therefore, we can figure out which part of our algorithm needs to be fixed.

7 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.

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

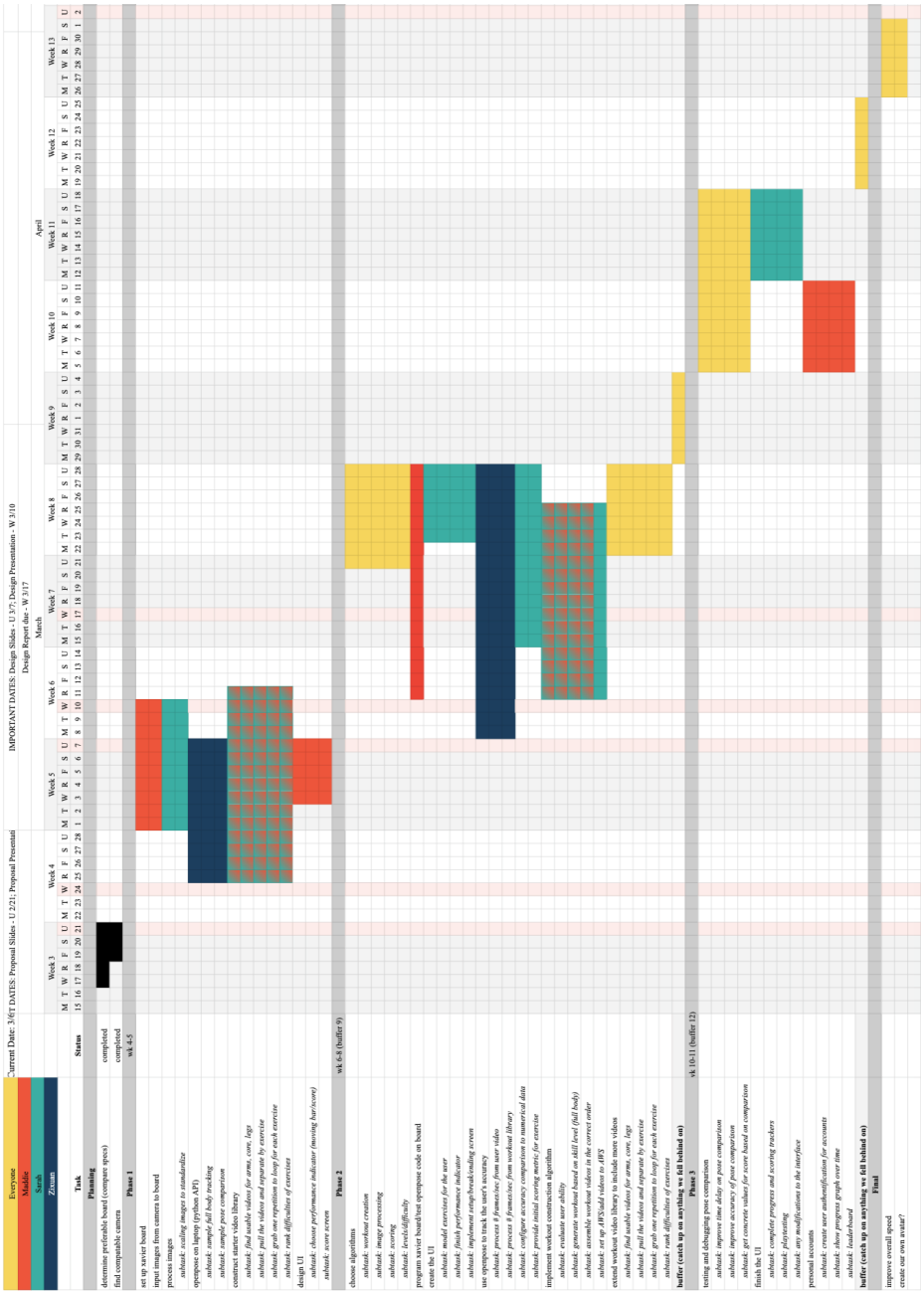

Figure 8: Gantt Chart