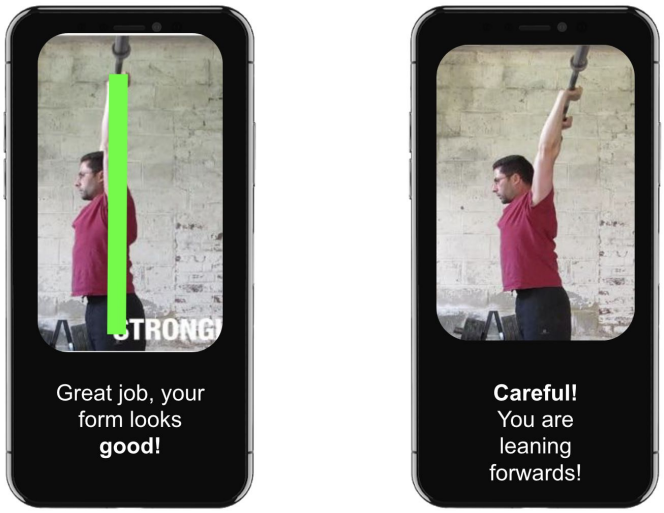
Kinisi

The Apple Watch Form Correction Coach

Author: Adrian Markelov, Kyle Jannak-Huang, Matthew Spettel. Electrical and Computer Engineering, Carnegie Mellon University



***Abstract*—An app for the iPhone and Apple Watch that is capable of analyzing the user's weightlifting form. It uses live data from the watch's inertial measurement unit to extract features of good and bad lifting form, and classifies the form using a convolutional neural network. The app will give visual and instructional feedback to users, with the goal of helping improve their form and preventing injury while training.**

***Index Terms*—Form Detection/Correction, Apple Watch, Inertial Measurement Unit, Data Analysis, IOS App, Flask Server, Pytorch, MSNN, Time Series Classification, Convolutional Neural Network, Dynamic Time Warping**

I. Introduction

The need for a quick and inexpensive system for evaluating weight training form has grown from a recent explosion in fitness popularity. Proper form during training can maximize results while minimizing the risk of injury while weightlifting. The supply of personal trainers has not kept up with demand. Thus, the cost of hiring a personal trainer in the U.S. can range from $60/hr to $160/hr. The goal of Kinisi is to make personal training accessible to all, and to reduce the likelihood of injury while weightlifting.

Kinisi is an app for the iPhone and Apple Watch that uses inertial measurement data from the watch to recognize risky issues in the user's lifting form, and provide visual and instructional feedback on the user's phone. Feedback is provided promptly after the user finishes each set, so he/she may correct their lifting form on the next set. Risky errors in the user's lifting form will be detected with high accuracy.

II. Design Requirements

A. *Classification Accuracy*

Our app will need to be as accurate as possible in classifying the user's lifting form in order to minimize the likelihood of injury to the user. For the purposes of validating our approach, we will begin by classifying only bicep curls. The classifier will have a bucket for good form and at least four buckets for known characteristics of bad form, such as "swinging arms" and "splayed elbows". It will be trained and tested on data collected and labeled by us; at least 20% of the data for each bucket will be testing data, and there will also be testing data from a variety of users. We will then measure the accuracy of our classifier based on how it classifies the testing data.

Quantitatively, we want to minimize the number of false positives and false negatives reported by our classifier. False positives (the case in which the user has good form but is labeled as one of the many types of bad form) must be kept below 33% on exercise sets. Bicep curls are typically done in three sets, so if only one of the sets (or less) are reported as having bad form, the user will not lose confidence in our system. False negatives must be kept below 80% on reps, and below 1% on sets. The shortest sets of any exercise are typically 5 reps, so if the user has bad form during such a set, we want to be able to classify at least one of the reps as bad form in order to flag it and give feedback to the user. If an entire set of bad form is overlooked, however, it may cause the user to continue with bad form and eventually cause injury.

B. *Timely Feedback*

Our app must provide feedback to the user quickly, so that they can view the feedback on their lifting form and correct it as soon as possible. Quantitatively, this means that the feedback must be available on the user's phone app while they are resting between sets, with enough time before the next set to process the feedback. Typically, rest times between sets of bicep curls are no shorter than one minute. Since our feedback will be succinct (GIFs of form from different angles and a sentence or two), a generous estimate is to allow the user about 20 seconds to process the feedback. Thus, the classification of the form, along with the visual and instructional feedback, must be viewable on the phone app within 40 seconds after the completion of a set.

III. Architecture and/or Principle of Operation

The complete architecture of the product can be divided into three super general categories: the Apple Watch, the Iphone and the backend AWS server. The Apple Watch forwards data to the iPhone via BTLE, then the iPhone makes a RESTful API call to the AWS server with the collected data asking it for an informative response on the weight lifting form. Amongst these three the most important by far to zoom in on is the backend AWS server that will analyze the Apple Watch data and generate the required feedback for the user. The backend server will be managed by Flask and will have three super important analytical processing parts to it: exercise rep demarcation, rep data preprocessing, and rep classification. The Flask server will parse the RESTful json and forward the received imu data to this processing unit. Once the server knows the form class it will return pre-saved data from a SQL database that described the issue or lack of issue in the exercise form. Please see figure 3 on the last page of this report for a visualization of the full system architecture.

Generals Systems Architecture

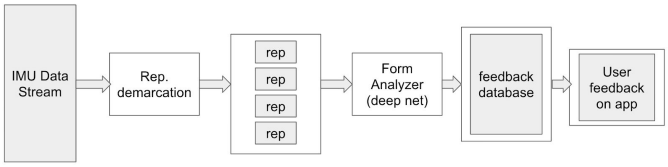
A. *Rep Demarcation*

Rep demarcation is the first sub-system that the raw IMU data will be passed through in the backend server. The purpose of this sub-system is to automatically separate the raw time-series data into repetitions of exercise. The raw data is 9-dimensional time series data: acceleration, orientation, and rotation velocity in three dimensions each. It is collected starting when the user presses the "start" button on the app, and ends when the user presses "stop". Thus, the signals can contain extraneous artifacts, such as picking up weights at the beginning of the set, stretches, or even a short break between reps. The rep demarcator will isolate just the reps by detecting repetition in the IMU signals, using a combination of peak detection and sliding dynamic time slicing. This architecture will be discussed in detail in section V. Please see figure 4 on the last page of this report; it shows the current design for the rep demarcator, although there will likely be significant additions to the design as we do more research.

B. *Form Detection*

Form detection is the second sub-system that will take the raw IMU data that has been parsed by the rep demarcation software and classify each rep's form from a finite set of buckets including good and various types of bad form. It will be done by a two step process of filtering (for pre-processing) followed by a deep net for classification. Since the data received by the rep demarcation software will be irregular (varying in time samples) and have important features at both a global and local level, the preprocessing step will normalize the data then decompose it into channels that will allow the network to look at the data from different time and frequency perspectives. Compression might also be done along the way and explained later. Finally, a deep convolutional network will extract features from the individual channels. These features will then be concatenated together and more convolution layers will be executed on them with a final fully connected layer that will classify the new feature space. This architecture will be discussed in detail in section V.

Data Processing Architecture



IV. Design Trade Studies

A. Rep Demarcation

The academic name for rep demarcation is "finding time series motifs in time series data". Once we discovered this terminology, we were able to search for a huge collection of academic papers discussing how to best solve this problem - some of these papers even discussed detecting reps within a set of exercise. Given the time constraints of the class we had to make some difficult design decisions about which method to explore and test. We estimated that we only had time to try one method in any depth, given how complex a lot of the approaches are. After reading through the eight most popular papers, we started by eliminating all of the papers we did not understand at all after the first read through - this got us down to four approaches: two of which were ML/SVM based and two were more traditional signal processing. We ruled out the two ML/SVM approaches because they were relatively much more complex (and would require even more training data collection/labeling/cleaning, which we are already doing a lot for our form detection phase (discussed below). That left us with two approaches: one used a combination of blurring, thresholding, and simple autocorrelation in the frequency domain - essentially it was looking for spikes in a given frequency. This approach was very simple, but we knew that it would fall short when reps sped up and/or slowed down during the course of a set. Since this fatiguing (and slowing down) happens during almost every set, we did not want to rely on a constant period in our signal throughout an entire set to demarcate reps. This led us to the last paper, which suggested using an approach called dynamic time warping to make the last correlation step more resistant to changes, or warping, in the signal's period over the course of the set sample. Even this relatively simple approach took two weeks to implement to a testing level of robustness (and thankfully we were able to use a pre-built python package to run the actual math behind the dynamic time warping), so we are confident that we made the right design tradeoffs in optimizing for simplicity and period invariance.

B. Form Detection

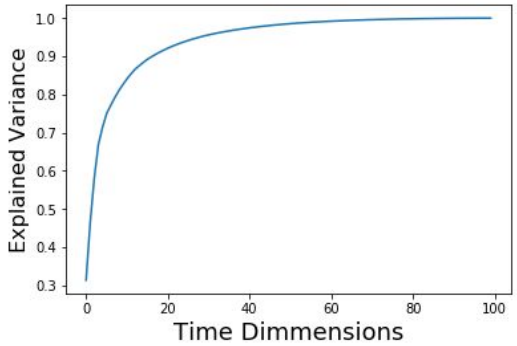
There are many potential solutions to classifying the form of a user's weight lifting. All solutions though must be in the basis of multidimensional time series classification (MDTSC). One of the most common solutions to MDTSC is dynamic type warping (DTW). As of now we will not be doing DTW for the following reasons: sensitivity to noise, separation of feature extraction from classification which limits accuracy (Cui and Chen) and neglects that most time series have different important features at different time scales.

These problems can be addressed with deep learning solutions such as RNN's and CNN's. We have decided to go with CNN's. The decision to go with CNNs over RNNs is a hard one. In concept it is likely that both methods will work very well. As a matter of fact it superficially seems like RNN's will work better as they are most commonly used for time series data. The assumption in our case though is a little different from normal time series data. In our data, for a single data point (a single rep of a workout) the most important characteristic of the value of any dimensional value is the locational magnitude of where the watch is. This means that vectorized, a single vector index in one rep should correspond to the same euclidean location in workout space as another rep in the same index. This can be done trivially by scaling the data between the start and stop of a set to the same vector space where CNN's can trivially be used. However, when we use RNN's a very significant emphasis is put onto the difference in time between samples as adjacent data points are now in the bases of time difference vs location difference. The form of a rep is completely independent from the speed of the rep, good form can be both fast, slow or medium. Thus, it will be very difficult for a RNN to see outside of this because data in the same form buckets will look drastically different to each other when in this domain thus making it harder to classify.

Amongst the thousands of types of CNN's we will be using the framework described in *Cui and Chen* Multi-Scale Convolutional Neural Network for Time Series Classification. This framework addresses all of the problems listed in the other two models and is can still be fast with GPU optimization.

On the account of potentially compressing the data set to help prevent overfitting during training, we have decided to go with PCA. There are many ways to compress data but PCA works very well because it has a framework for determining the amount of encoded information at a given compression rate. In our tests so far we have been able to compress the data by 6.25 times (84%) and retain 90% of the explained variance. There is a potentially huge drawback to compressing the data if it is not needed. The compressed data set has much more high frequency that may destroy the information in the shape of the original data that correlates with time. Thus, we fill first test without compression.

Cumulative Explained Variance of Time Series/ Dimmension




```
In [17]:  1  np.cumsum(pca.explained_variance_ratio_)[16]
Out[17]: 0.8987758023556448
```

(c)

C. *Joint Estimation of the User's Arm*

One of the original design plans was to use accelerometer data from the Apple Watch's inertial measurement unit (IMU) to estimate the position of the human arm. This would enable us to generate an animation of what the user's motion looked like, and show it to the user alongside an animation of the correct form. The visual feedback, provided in conjunction with instructional feedback, would allow the user to correct mistakes in their form. Ultimately, for reasons described below, we decided to replace this functionality with example GIFs of a human doing the exercise correctly.

The joint estimation problem requires specifying joint angles in the shoulder, elbow, and wrist. Typically, this would require solving a 7 degree-of-freedom inverse kinematics problem, using positional data from the IMU (6 degrees of freedom in 3-dimensional space). The IMU does not directly provide positional data; it must be obtained by double-integrating the accelerometer data. Unfortunately, accelerometer data typically drifts significantly. Accurately estimating the position would require removing this drift, as it has an exponential factor on the estimated position after double integration. The industry standard way of doing this typically involves using Kalman filtering with carefully tuned constants and covariances. In order to avoid the time-sink of implementing this, we tried using regression to find the drift as a function over time, and subtracting it out from the acceleration data. Figure 1 below shows the result of attempting to remove drift this way and integrating once. Figure 2 shows the result after the second integration. The velocity graphs look reasonable, however the positional graphs are dominated by a lower frequency signal that stems from the residual between the actual signal and the regression line.

After experimenting with this and looking at different options for Kalman filtering, we decided that the results we would get from joint estimation would not be accurate or consistent enough to be useful to the user. Even if the drift were successfully removed, the positional data could still be jittery. Furthermore, the inverse kinematics problem using a wristwatch is grossly underdetermined, and massive simplifications needed to be made to even approximate a solution. Ultimately, the user would rather just watch their own form in a mirror than look at an animation of our joint estimates, therefore we removed this sub-system from the overall design.

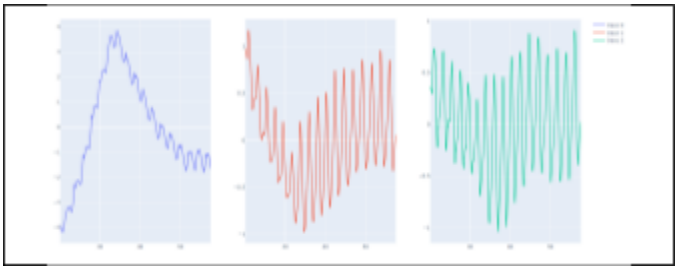


Fig 1. Velocity in x, y, and z directions after linear regression fit removed.

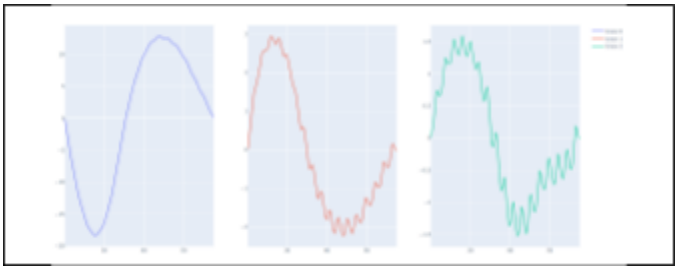


Fig 2. Position in x, y, and z directions after linear regression fit removed.

V. System Description

A. *iOS/Watch + BTLE Communication*

The iOS system will rely heavily on Apple-provided communication functions. Specifically, the sendMessage and sendUserInfo functions will be used to transmit data back and forth between the watch and phone. When the watch screen is off, the default state of a watch app is background suspended, meaning that none of the application level code is run, except for background network requests. This means that to run our IMU-polling code constantly when the user is obviously not going to be looking at their wrist (because they will be working out) we need to put the watch into a special mode. Fortunately for us, one of these special modes in a "workout mode" that enables background sensor reading, including IMU and heart rate monitor - this special mode allows our application-level code to run in the background roughly 50 times per second. The tricky part becomes, how do we get the watch into this mode reliably when it does not allow our code to run in the background by default - in other words, how do we reliably send a message to the watch when it is not running our code? Fortunately Apple provides a summonExtensionApplication function that allows us to summon our watch app out of a background mode to the foreground for a split second. In this split second, we can begin a workout session and then have reliable communication. Once a workout session is started, we will send packages consisting of 9 dimensions of IMU data and a timestamp (10 dimensions x N samples, where N is 150-300, and all values are saved as 64-bit doubles) directly to our backend server using an open source iOS HTTP request library called Alamofire. By default, Alamofire uses the iPhone's network (WiFi or 3g/4g) as a proxy to complete the request; that is, the watch will send the data to the phone via BTLE automatically and then the phone will make the request

with its more powerful antennae and processor (as well as battery life). If the phone is currently not paired or available, the watch will then attempt to make the request over its own networks (WiFi or LTE if a newer watch). The watch will receive regular HTTP responses to these HTTP requests and use the encoded JSON information to decide whether or not to end the workout mode and what information to display to the user. The server can send URL's to GIFs that we have found and saved ahead of time to display certain examples of good/bad form to the user, as well as any text tips/notifications for the user.

B. Cloud Server

The cloud server subsystem is relatively simple. The server is hosted on an AWS EC2 instance which provides a completely isolated linux machine for us to SSH into and run whatever we want on. In this case, we have chosen to use Flask as our serving logic (to keep all of the server-side code in Python), and Gunicorn as our production WSGI server to be able to configure how many worker threads we will have and what their behavior will be. We have also set up NGINX as a reverse proxy in the event that we will want to run multiple servers or want to quickly switch between different server configurations during testing. Using a reverse proxy allows us to switch between different ports/servers on the EC2 internally, while the iOS code (and what port it is talking to) remains completely unchanged. This web server ultimately hosts and runs the code discussed in sections C and D, where the actual work of the project is done.

C. Rep Demarcation

The rep demarcator will reside in our backend server. It will take as input the raw IMU data as a .csv file, and output the same data but marked for where reps occur. The raw data has nine dimensions that vary over time. Linear acceleration, orientation, and rotational velocity of the Apple Watch are each given in three dimensions. The orientation is described by the "gravity vector", a normalized vector that points in the direction that gravity is pulling. These nine signals will be heavily Gaussian blurred to remove high frequency noise. Using simple peak detection from SciPy, we can split the signal where there is likely to be reps. However, these peaks could be a variety of other motions, such as the user bending over to pick up their weights.

In order to reject these signals, we will use an algorithm called sliding dynamic time warping. This algorithm takes as input a kernel signal, such as a sample rep, and a longer signal, such as the entire data stream for the set. It outputs a continuous correspondence function that is high-valued where the kernel signal is similar to the longer signal, and low-valued where it is not. The "sliding" is similar to convolution, and the "dynamic time warping" refers to the algorithm's method of generating correspondence values for a range of time-warped kernel signals over the original signal. This will allow us to detect repetition in the IMU data without the correspondence function being entirely dependent on the duration of each rep.

Our current design will use the middle rep, extracted from the initial peak detection, as the kernel signal. This is due to the tendency of weightlifters to slow down as their set goes on. We choose the middle rep because it is most likely to be the average duration. This also reduces the likelihood that we accidentally choose a kernel that is not a rep of the exercise. As we make progress on the project, we may choose to generate correlation functions for multiple sample reps, choosing to use the one that generates the highest correlation function.

Once we generate a continuous correlation function for each of the nine dimensions of IMU data, we can sum them to one overall correlation function. Using all nine correlation functions will make the rep demarcator as robust as possible. For example, bicep curls typically have very clear repetition of signal in two directions of linear acceleration and orientation, but will be very sensitive to noise along the axis perpendicular to the elbow. If we have the chance to extend the project to multiple exercises, the dimensions that are most important in generating correlation functions may change. With this in mind, we may change the design to weight different dimensions more than others, depending on the results of our experimentation.

Using the total correlation function, we can finally perform peak detection once more to separate where the reps are most likely to be in the original signal. We can then output this information to a .csv file, for use with the form classifier. For a graphic of how the rep demarcator works, see figure 4 on the last page of this report.

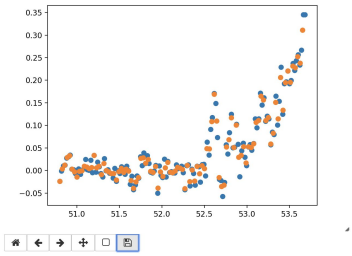
D. Form Classification

Form detection will be done by a joint process of preprocessing and deep learning, for justification on why these methods over other common methods please see Design Trade Studies. The form detection process will start out with the assumption that the received data has been correctly demarcated by the start and stop of each rep by time markings of each of the 9 dimensional data points.

These markings though do not guarantee a fixed sample size between the reps because reps can be done at different velocities. To address this as well as many other subtle time based features on the data we will pre-process the reps before we use any classification methods.

Pre-processing will include interpolating the data, decomposing the data into multi-scale and multi-frequency channels, and potentially compressing the data. The first most important step is interpolating so all of the data is in the same basis. Since the average rep of a bicep curl is 2 sec and the imu sensor samples at 50Hz, we will normalize the data by super/sub sampling to 100 samples/rep.

Aligning Reps with Interpolation



Here the original blue data has less than 100 samples and is super-sampled to 100 in orange

In the 100 sample basis we will generate three mapping of the data: an identity mapping, a multi-frequency channel mapping and a multi-scale channel mapping. The first map has only one channel and it's an exact copy of the original 9D signal. The Second will create a finite number of frequency channels via different sized smoothing kernels, and third map will create a finite number of scaling channels via subsampling. Finally, we may implement a process of compressing the data if we cannot provide the deep net with enough data to prevent overfitting. This compression would be done with PCA. PCA will not officially be part of the design unless it's needed. For reasonings on why we should/n't use PCA please see Design Trade Studies. However if we do use it, it would be used to take the first 100x9DOF time-series and compress them into 16x9DOF time-series or it could be used post signal map channeling right before the deep net (please note these are our design options not justifications).

Frequency Channel Transform

$$T^{\ell} = \frac{x_i + x_{i+1} + \ldots + x_{i+\ell-1}}{\ell}$$

By using a running average at different length l we can analyze the data with varying ranges of its frequencies when starting from 0Hz.

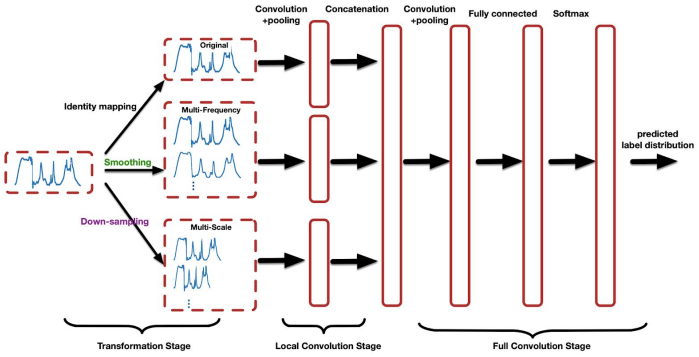
Scale Channel Transform

$$T^{k} = \{t_{1+k*i}\},\ i = 0, 1, \ldots, \lfloor \frac{n-1}{k} \rfloor.$$

By subsampling the time series at different scales we can use small convolution kernels to extract both global and local features from the time series reducing kernel parameters.

Deep learning for classification will be used via the Multi-Scale Convolutional Neural Network for Time Series Classification framework described by *Cui and Chen*. This method begins with the three tier mapping described in the previous preprocessing section. On the basis of the three maps, this method will first apply three separate convolutional networks (with max pooling) to each map. Extracting features from the three basis allows us to simulate a multi-dilated network without having to actually learning extra parameters of larger convolution kernels. This reduction in parameters will help us a lot with overfitting down the line. After these three seperate networks have extracted local and global features from the maps, these features will be concatenated together and put through another convolutional layer with max pooling. Finally, these final features will be put through a fully connected neural network module topped with a softmax for function to generate probabilities of each class.

MSNN Model



VI. Project Management

A. *Schedule*

Our schedule is divided into three main phases. This first phase is very short and involves brainstorming a lot of potential solutions to the form analysis problem. Clearly, we are already past this phase and have settled on a wearable IMU / Apple Watch strategy. The second phase's objective is to construct an MVP, or minimally viable product. We have defined an MVP as a system where a user can do a single set of bicep curls (with no user interface - just a start/stop button) and have our backend console spit out a probability distribution across at least three different "buckets" of form. For example, these buckets could be "good", "swinging", "bent-back". We have a rough requirement for the MVP to be able to have less than 33% false positives on sets and less than 80% false negatives on reps. Phase three is focused on adding polish and robustness to the entire system - iOS UI for starting and stopping exercises with GIFs demonstrating proper form will be built out. By fine tuning our rep demarcation, data preprocessing, and form classification algorithms, we will increase our accuracy to meet the requirements defined in section II above. By the end of phase three, the project should be ready for a live demo where a user does a set of bicep curls, and within the required time window, is presented with clear and instructive feedback on their iPhone. The schedule is pictured in Figure 5 on the last page of this report.

B. *Team Member Responsibilities*

Our team is strategically made up of machine learning, iOS, computer systems, and signal processing experts. Spettel has the most experience with quickly building iOS frontend and phone/watch communication, and as a result is responsible for building both the iOS and watchOS applications. Jannak-Huang is skilled in signal processing and kinematics, making him a good fit for the rep demarcation stage of our

processing. And lastly, Markelov is the stand out machine learning expert of the team and consequently is responsible for building the data preprocessing and form classification modules. Spettel and Markelov together have a secondary responsibility of setting up and managing the AWS EC2 instance and the associated Flask server, Gunicorn wsgi server, and Nginx reverse proxy. All three team members have the additional responsibility of collecting labeled training data, with this being a secondary responsibility for Jannak-Huang.

C. Budget

Given the almost entirely software nature of this product, we have not used any of, and do not plan to use any of the provided budget. Between the three of us, we had two Apple Watches and two iPhones from the start, and deemed that a third pair will not be necessary. We are using the free tiers of all cloud services and avoiding expensive server costs by training models on our own high-performance GPU machines - used for gaming in a different life. All of our software tools including XCode, Flask, PyTorch, Gunicorn, Scipy, and Alamofire are free to use.

D. Risk Management

1) Design and Schedule: As discussed above in section IV, we used simplicity and robustness as our main criteria in picking between different approaches. From our previous product development experience, we know that everything takes roughly five times longer than you initially would expect. Essentially, we designed as if we thought we had a fifth of the time in the back of our heads (and using that mindset to keep our schedule realistic), and we expect that it will still come out pretty tight at the end of the semester.

2) Resources: As discussed in section C above, we do not have any budget constraints for this project. However, we have very strenuous personnel constraints. We only have one iOS expert and one machine learning expert in a semester that is very busy for all of us. Furthermore, Spettel works full-time on his startup in addition to his course work. The main risk associated with this extreme lack of (undivided) man hours in simply not finishing the project by the end of the semester. To mitigate this risk, we simplified the scope of the project as much as possible. Although we knew the "product" would be more attractive as a system that could identify dozens of exercises, we decided to focus on only a single simple exercise: the dumbbell bicep curl.

VII. Related Work

A. Computer Vision Based Systems:

1) **Perch**: This Boston-based startup using a depth-enabled camera (similar to the Intel Depth-Sense) to track an athlete's movement. The system is mounted in a fixed position at the top of a squat rack. This limits the amount of potentially tracked exercises to squats, deadlifts, and olympic lifts - this selection makes up a large portion of athlete workouts, but is only about 10% of the exercises that casual gym-goers use. Furthermore, the system is focused on measuring the velocity and "explosiveness" of the athlete, and does not offer direct coaching on form - it is assumed that athletes are capable of using correct form and/or they are under the direct supervision of very experienced coaching staff.

2) **GymCam**: This CMU research project out of the HCI school using a system of wall-mounted cameras to identify what exercises gym goers are performing and how many reps they have performed. They do not perform any kind of form analysis.

B. Wrist-Based Wearable Systems:

1) **Atlas Wearables**: This Texas-based startup built their own consumer product wearable that automatically tracks a user's activity in the gym. They perform some rudimentary form analysis by providing a percentage score for each set that you do. It does not, however, tell you what you did wrong, and in experimentation, our "good" form was scoring anywhere from 60% to 99% - the same range that our "bad" form was scoring in - the percentage metric simply doesn't seem to mean anything.

2) **Gymatic**: This California-based startup runs completely on the apple watch platform. Their software is able to detect exercises and count reps. Similar to Atlas Wearables (discussed above), the provide a percentage score for each set, meant to tell the user how good his/her form was. In testing, this percentage has little correlation to whether the user is actually using safe form or not.

C. Bar-Based Systems:

1) **Beast Sensor**: Likely the most simple of the technology listed - this IMU-in-a-box sits directly on an olympic bar and is held in place magnetically. Similar to Perch (discussed above), due to the constraint of the bar, this product only works with a very small subset of exercises, and only provides feedback based on velocity and explosiveness, not on form directly.

VIII. Summary

While it is too early in our development process to tell how well we will meet our requirements, we have done significant testing of our rep demarcation. Currently, we can guess the exact number of reps 70% of the time and be within one rep 90% of the time on our dedicated testing dataset.

A. Future work

We will definitely continue to work on this project after the conclusion of the semester. The startup that inspired this idea, DeltaTrainer, will likely transfer the rights to the IP created in this class to itself and integrate form analysis into the next generation of it's personal training app.

References

[1] GymCam: http://smashlab.io/pdfs/gymcam.pdf

[2] RecoFit (Microsoft Research): https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/Morris_Workout_CHI_2014.pdf
[3] Atlas Wearables: https://atlaswearables.com/
[4] Gymatic (Created by Vimo Labs): http://www.vimo.co/
[5] Probabilistic Discovery of Time Series Motifs: https://dl.acm.org/citation.cfm?id=956808
[6] Exact Discovery of Time Series Motifs: https://epubs.siam.org/doi/abs/10.1137/1.9781611972795.41
[7] Using Dynamic Time Warping to find patterns in Time Series: https://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf
[8] Multi-Scale Convolutional Neural Networks for Time Series Classification: https://arxiv.org/pdf/1603.06995.pdf
[9] How to Use Convolutional Neural Networks for Time Series Classification: https://towardsdatascience.com/how-to-use-convolutional-neural-networks-for-time-series-classification-56b1b0a07a57

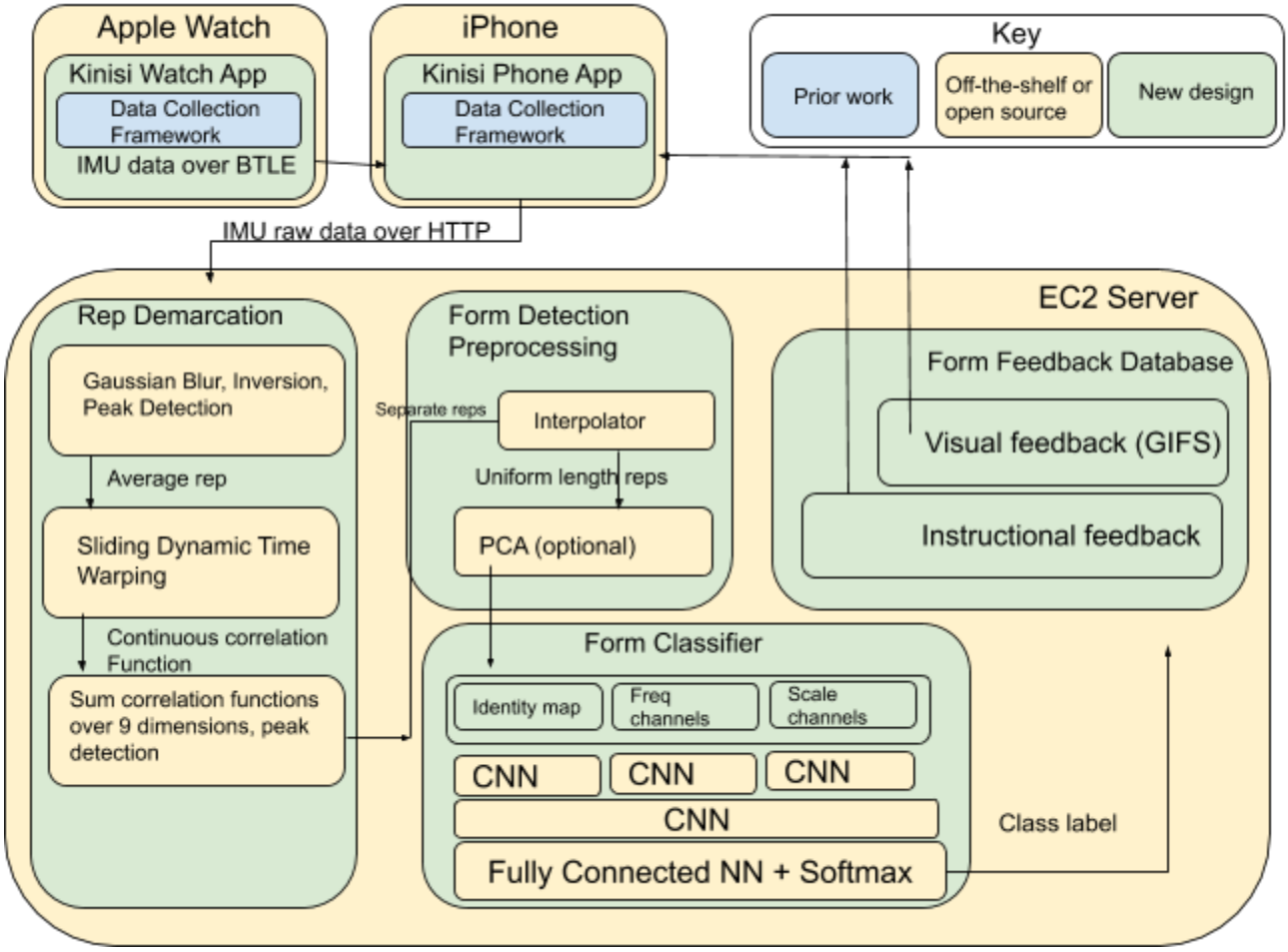


Fig 3. Overall Kinisi system architecture.

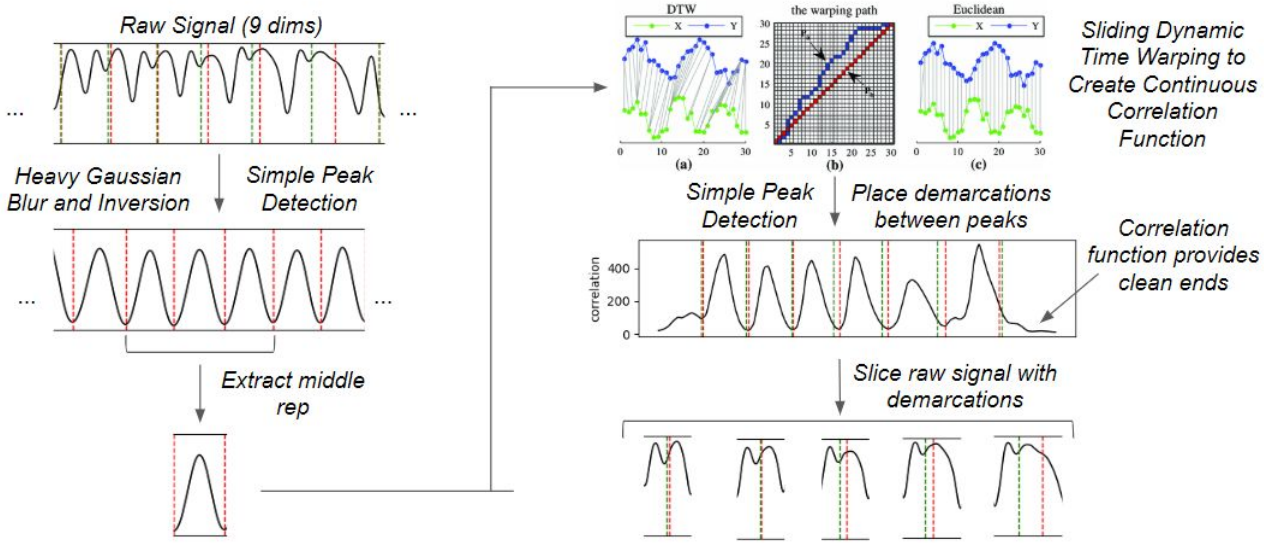


Fig 4. Rep demarcator design.

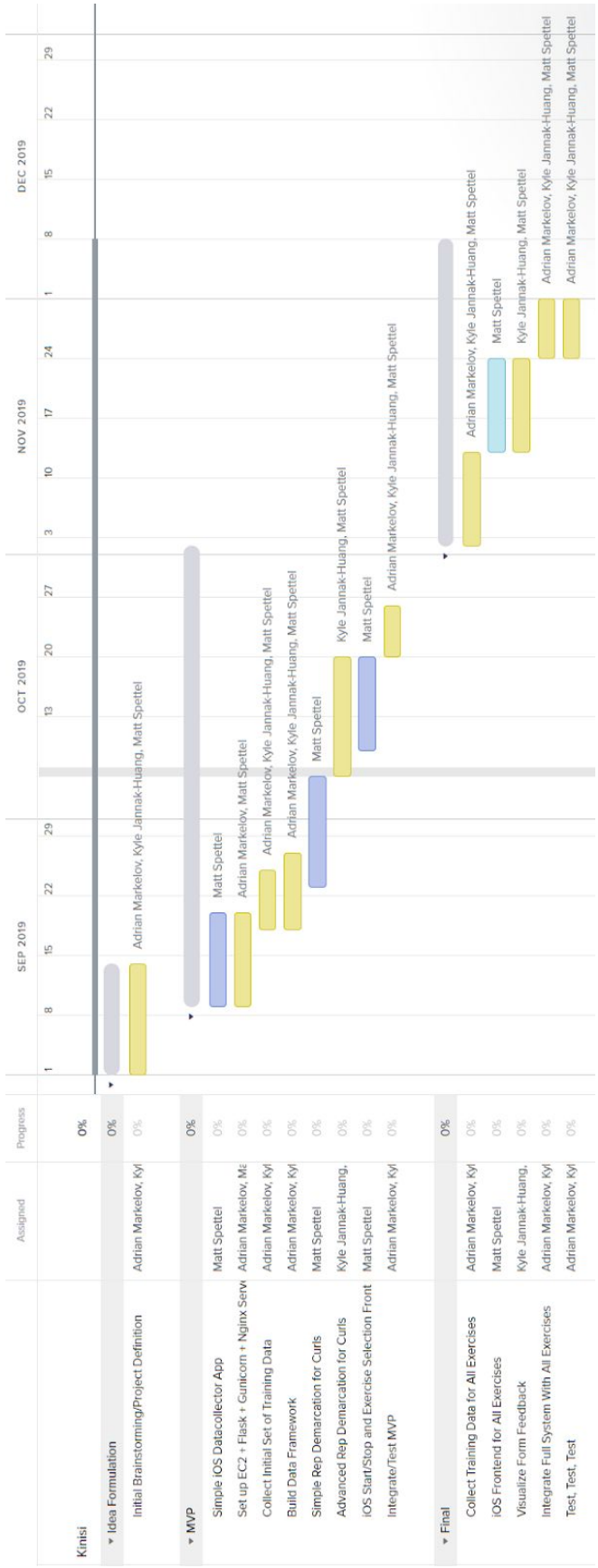


Fig. 5 Gantt chart