Developing a Mobile Application and Cloud Computing Framework for Gesture Recognition Using Electromyographic and Inertial Measurement Unit Data

**Ricardo Colin1, Victor Delaplaine1, Paul Leung1 , Danny Ceron1**

**Xiarong Zhang2 , Alexander David2**

**1Cañada College**

**2 School of Engineering, San Francisco State University**

# Abstract

Humans skeletal muscles naturally produce electrical activity by the contraction of muscles. Electromyography (EMG) is the technique of recording and evaluating these electrical activities. EMG Pattern Recognition (PR) is the process of taking these signals and processing them with the intention of classifying human gestures and can be used to control prosthetics, virtual/augmented reality, robotics, and more. EMG PR systems must operate quickly in real time. They must also be accurate, durable and robust in order to be used in practice. EMG PR is data driven, the amount and quality of data determines the success of the process. Proper data storage and processing techniques could allow us to compile data from many users to create one PR model that can be applied to any human. A mobile Android application, paired with the MYO Armband EMG and IMU sensor (Thalmic Labs), was developed as a user friendly and mobile EMG PR solution. The application is aided by Amazon Web Services to store and process large amounts of data. The concept of users is introduced to the application through user registration and login pages in order to store and evaluate user-specific data. This paper discusses ongoing development of the application and experiments conducted evaluating the feasibility of creating a model composed of data from all users.

# I. Introduction

Much of today's technology is based around the idea of a person interacting with a computer. The concept of a computer interaction with a user is known as Human Interface Machine (HMI). HMI’s are used to collect data from a user and then produce an output. For example in rehabilitation centers users with no limbs are subjectected to prosthetic limbs. These artificial controlled limbs can be activated by using different movement intent by using a wearable sensing device. These wearable devices have electrodes (sensors), accelerometers, gyroscope, and magnetometer. Gesture Recognition can be produced by other methods, such as video detection. Video detection has faults in its accuracy for gesture recognition, the results are blurred by different lighting environments.

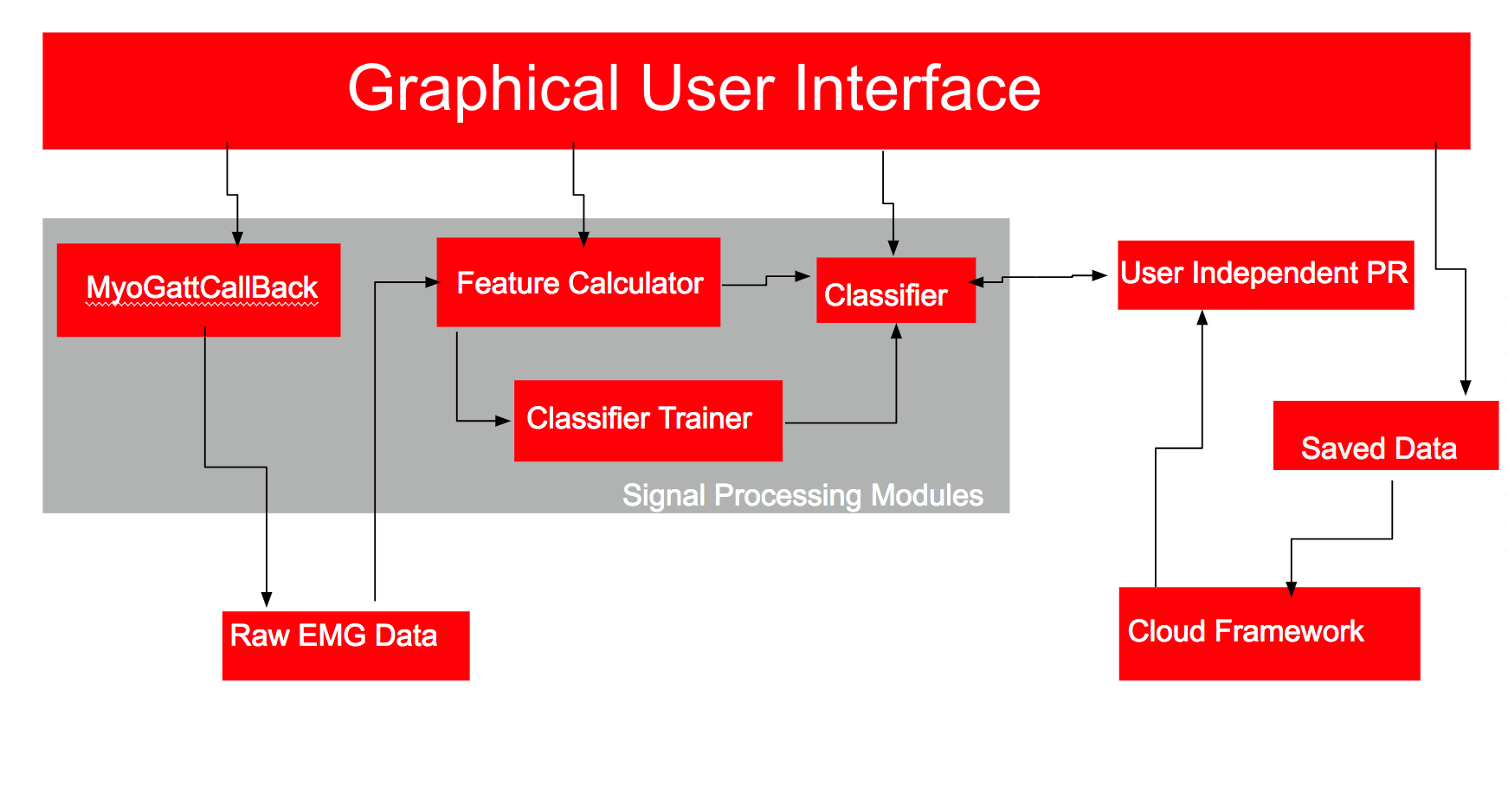
Wearable device sensors takes advantage of electrical activity of our muscles known as electromyographic(EMG) signals[1]. This method can be used to control prosthetic limbs and is used to help limbless patients. In order to get viable results the wearable device must meet a set of conditions[2]. Firstly the system must be accurate. This means that the gesture that is read needs to be the same gesture that is mechanically being done by the system. The second condition that the system must meet is to be user-friendly so that the app is natural to use. Finally the system must be able to work in real time under a delay of 200ms.

Using the constraints above we set off to work on an android application that is capable of processing EMG data from a wearable sensing device. The raw EMG muscle signals that are to be extracted to the via the armband must be sent to our application. This applications must be able to process that data and output a gesture by using the myo armband by Thalmic labs.



*Figure 1. Myo Armband developed by Thalmic Labs.*

Figure 1 shows a low costing and portable device that we have chosen to meet the requirement that we set earlier. The armband has 8 EMG sensor that are suppose to stream raw EMG data at 200Hz and the 9-axis inertial measurement unit (IMU) that is suppose to stream data at 50Hz. The IMU contained in the armband above has a three axis gyroscope, accelerometer and magnetometer.



*Figure 2. Structure of the android application.*

In Figure 2 above shows a design of our android application and its frameworks. This application was developed using the programming language java and was developed using Android studios. For the classification and model validation algorithms our application uses some prewritten functions from the Statistical Machine Intelligence and Learning Engine ( SMILE) library. Our application works with Amazon Web Services which allows users to login and retrieve there data as well as use pre trained data for gesture recognition.

# **II. Background**

### **EMG Data**

The application created through visual studios receives raw EMG data from the myo armband through the use of bluetooth low energy (BLE). Sensor data is affected by several sources of errors, some errors can be caused by external sources, hardware inaccuracies and imprecision, and it can also be affected by the environment it is in. This data must be subjected to data processing; in order to do this, the data is separated into discrete windows for analysis.

**Feature Calculator**

*Data Windowing*

There are two data windowing schemes: adjacent and overlapped sliding.[2]

|  |  |
| --- | --- |
| Adjacent Windows.png    *Figure 3. Adjacent window scheme.[1]* | Overlapped Windows.png  *Figure 4. Overlapped sliding window scheme.[1]* |

For adjacent windows, the window size and the window increments have the same length as shown in Figure 3. In contrast, the overlapped sliding window is a flow control method that has a smaller window increment compared to the window length, which can be seen in Figure 4. The advantages of using this method allow the application to obtain a denser data set, meaning the receiving device allows input of many sets of data and perform analysis before waiting for an acknowledgment or the current window to finish. Ultimately, using overlapping windows takes advantage of the computing power of the device [1].

*Feature Extraction*

The Application algorithms calculate the following time-domain features: mean absolute value (MAV), waveform length (WAV), number of slope sign changes (Turns), and number of zero crossings (Zeros). Two additional features called scaled mean absolute value (SMAV) and adjacency uniqueness (AC) are also calculated.[2]

Mean absolute value is a commonly used feature that detects muscle contraction levels [3].

for i = 1,...,i (1)

Waveform length refers to the total length of the waveform in a segment of time [3].

(2)

Slope sign changes indicates the amount of times there has been a change from positive to negative or negative to positive in the slope for three consecutive segments in a given threshold epsilon () [3].

and (3)

Zero crossings calculates the amount of times the waveform has crossed zero in a given threshold epsilon () [3].

and (4)

Scaled mean absolute value indicates gesture intensity:

(5)

Adjacency uniqueness is defined with equation 6:

(6)

**C. Classifier Trainer, Classification Algorithms, and Cross Validation**

The features collected from the feature calculator are arranged into a data vector. These vectors are utilized during the training of the classification algorithm and the prediction of gestures after it has been trained.

*Classification Trainer*

Training and classification requires, the feature vectors and knowledge of the corresponding gesture, or class, that is associated with that feature vector..[2] From the data acquired models are created to help differentiate and isolate classes based on their similarities and differences from the feature vectors. For instance, a resting gesture that does not activate much electrical activity in the forearm would produce a very low SMAV whereas a fist gesture would generate a much larger SMAV. When training the classifiers, the model would separate these two gestures based on SMAV.

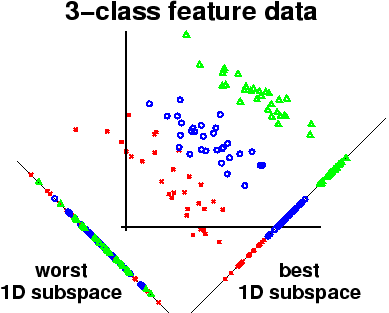
These models are then used to predict gestures, by comparing incoming feature vectors that need to be predicted with the model built by the classification trainer.[2]

*Classification Algorithms*

Classification algorithms take in feature vectors, but unlike when training do not take in corresponding class labels, or gestures. The algorithms predict the classes by comparing the feature vectors to the models generated by the classification trainers. In our project, we used four different classification algorithms, Linear Discriminant Analysis (LDA), Logistic Regression, Decision Trees, and K-Nearest Neighbor (KNN).[2]

*1. Linear Discriminant Analysis*

LDA reduces the dimensional feature vector to learn the data more efficiently. In our project, we might received many data that is large to perform any reasonable computation in reasonable time. LDA reduces the dimensionality of the data in the most optimal way for classification. This groups similar classes together and separates different classes. The figure below represents how LDA works,



*Figure 5. 3 dimensional input feature vector is reduced to 1 dimensional feature vector,*

*2. Logistic Regression*

Multinomial logistic regression builds upon binomial logistic regression, where the number of classes is limited to just two (for instance 0 or 1) and the number of features limited to one. In binomial logistic regression, we determine a logistic function which relates to the probability of either being one of the classes with a linear model. This probability can be summed up by the algorithm below:

(7)

Whereand are regression coefficients determined in the training phase that help fit a linear model to the logistic model and is the value of the feature, and Pr(0) is the probability that the incoming feature is in the class 0. This probability can be extracted through the following algorithm:

(8)

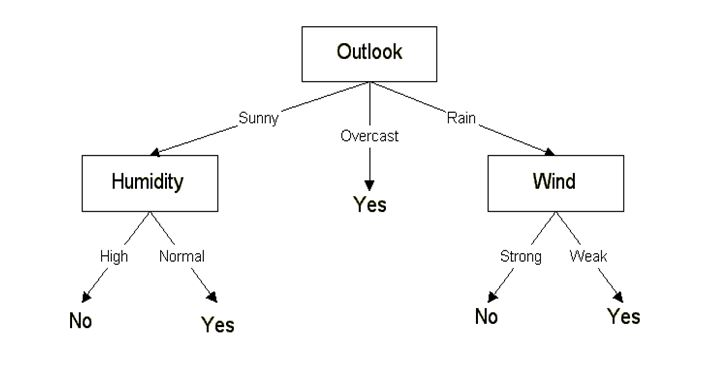
Note, the linear model represented in (7) can be represented as a dot product between vectors where the first element in the feature vector is 1. This binary classification and can be used for multiple classes. Logistic regression can find K possible classes by running K-1 independent binomial logistic models which can be represented through the following algorithm:

(9)

where Y is an integer that represents one of the classes. Multinomial logistic regression calculates the probability for each class from 1 to K. The highest probable class is then the final prediction the classifier outputs. [5]

*3. Decision Trees*

Decision tree learning go from observations about an item (represented in the branches) to a conclusion about the item’s target value. A decision tree is a diagram that uses a tree-like graph or model of decisions and their possible consequences. It is one way to display an algorithm that only contains conditional control statements [3]. It breaks up the data (or the concept) based on the characteristics of the data. It stops dividing into parts as long as the child nodes are fully classified.



*Figure 6. Representation of a decision tree. Rectangular boxes represent the attributes while the one with out boxes represent the actual splits to create the tree.*

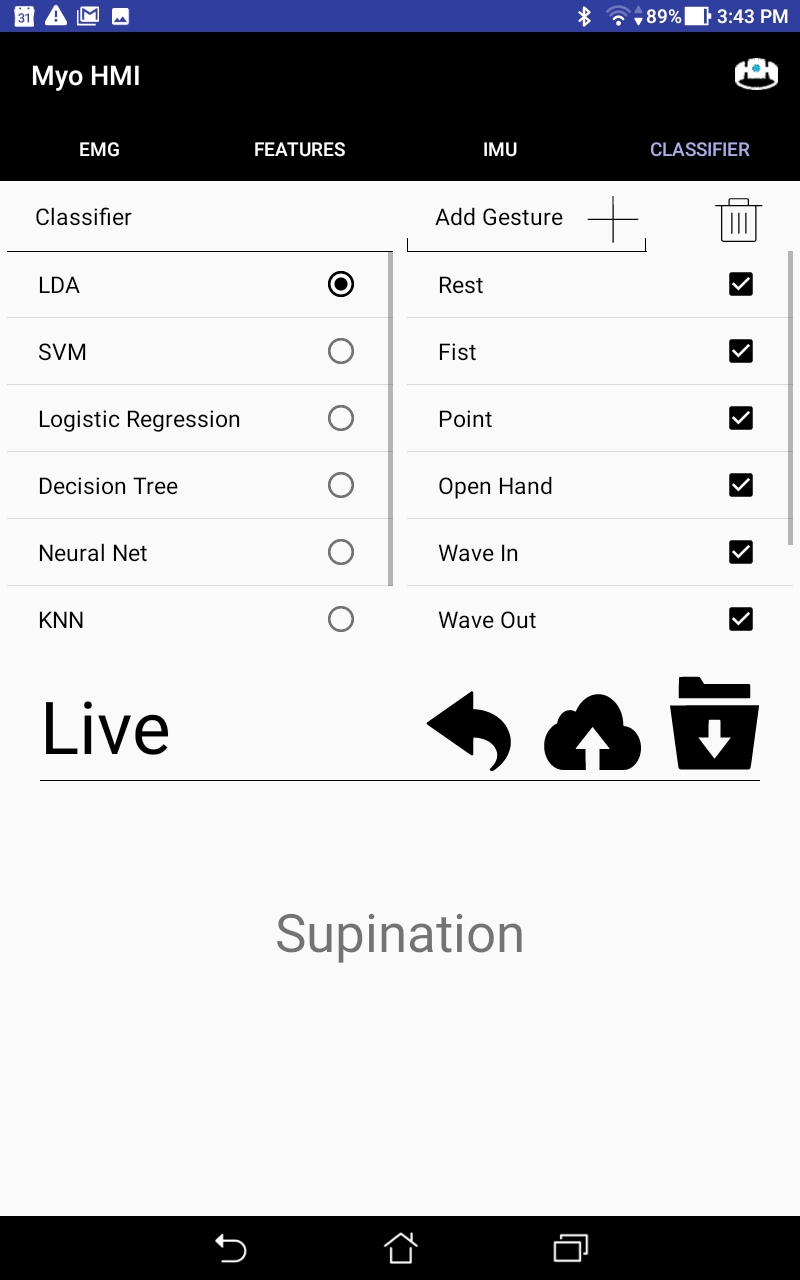
An example of a decision tree can be explained by the above diagram. Let’s say if we want to make a decision whether going for a vacation or not. The decision nodes here are questions such as “how’s the weather?”, “does it windy?”, “does it humid”, and the leaves which are the outcomes such as “high”, “normal”, “strong”, and “weak” to help us to determine the path. In this case, it is a binary classification example.

*4. K-Nearest Neighbor*

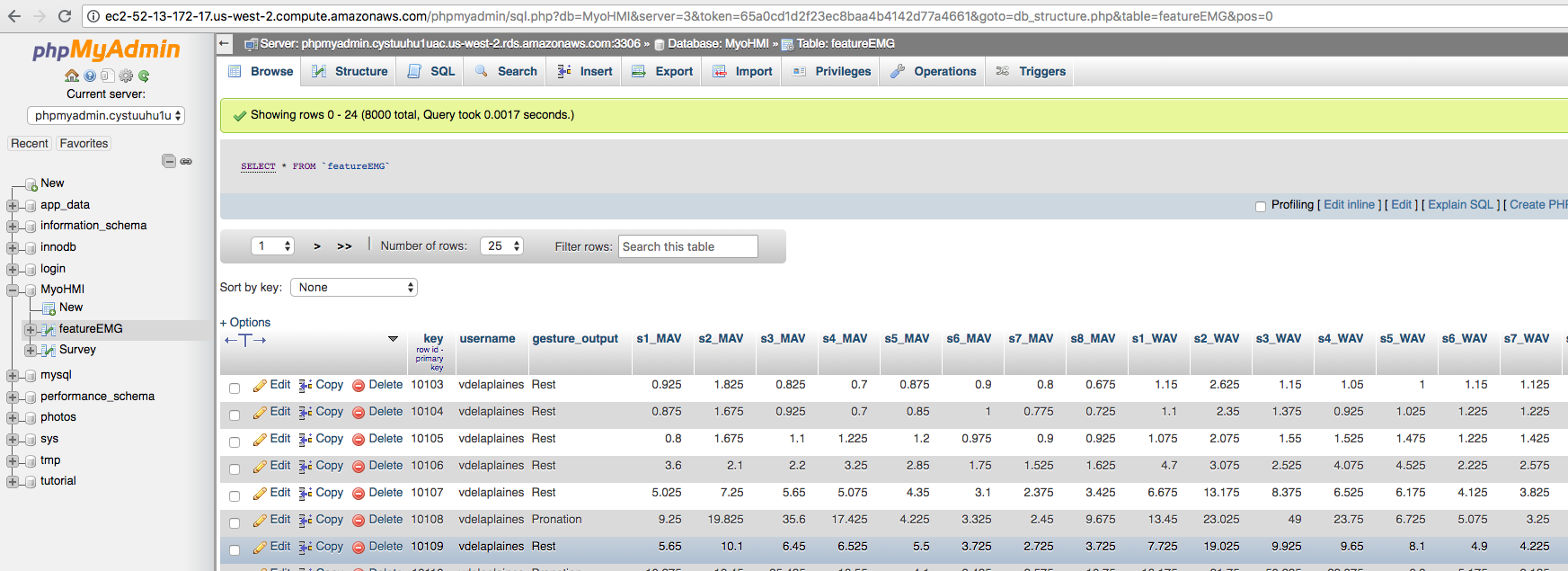
KNN is a non parametric, learning algorithm that separates data points in different classes, which can later be used for predicting the classification of new sample points. It uses the majority vote of its neighbors as its prediction. The object being predicted is assigned to the class most common amongst its k(positive interger) nearest neighbors. The graph below resembles a 2-class, 2-feature example of KNN[2]

Previews design and architecture of this app was based on three module hierarchies: *Signal processing modules*, *File interaction Modules,* and *Graphical User Interface (GUI).*  Signal processing module, contains the data collection module *MyoGattCallback.* Which allows the connection of the application to devices via BLE, updates the stream, and receives two streams of data of 8 bytes. vectors of 8 bytes. The feature calculation module - FeatureCalculator. Since the armband can only receives 8 bytes at a time, The raw EMG data has to be segmented into two vectors. This modules segments the data into overlapped sliding windows of length 40 with window increments of 8. In each window, the features: MAV, WAV, turns, zeros, SMAV, and AU are extracted from each of the 8 channels of the myo armband and then concatenated into one data (feature) vector. This gives flexibility to the user as they are able to select which features they want to use. [source]. And *Classification Trainer and Classification Module - Classifiers and SMILE library,* this module utilizes the SMILE (Statistical Machine Intelligence and Learning Engine) library written by Haifeng Li[2]. it contains many classifiers that can be trained with 100 data vectors per gesture. This classifiers can only take in arrays of doubles or integers. After the data is collected, a classification model is created. The remaining data from the classifiers is sent to the classification module for prediction, in this module the incoming data is not labeled.

*Data Storage Modules*

*This modules allows user to upload trained data vectors to the cloud in a mysql database or have them stored in a text file on the android device. After the training period there is an option on in the application that allows you to export your data to the cloud or phone.* 

*Figure 7. shows a screenshot of the myoHMI application after the training session had ended. The cloud button in Figure 7 allows you to upload your data to the mysql base so it's easy to access.*



*Figure 9. Users Featured EMG data stored in a mysql database.*

In Figure 9 is a picture of users data that we had collected, and then uploaded to the cloud. The visual you see is phpmyadmin, it allows the admin of the database to use a graphical interface to manage the data.

# III. Design and Implementation

**A. MyoHMI Modules**

*Graphical User Interface (GUI)*

The GUI implementation of the app is done by using Java and eXtensible Markup Language (XML). This is an appealing and user friendly GUI which contains four main tabs: EMG, Features, IMU, and Classification.

|  |  |
| --- | --- |
| Screenshot_20170807-104146.png  *Figure 10. Previous - Bluetooth scanner screen in the MyoHMI application, displays all bluetooth devices detected.* | Screenshot_20170803-130112.jpg  *Figure 11. Previous - Classification tab in the MyoHMI application, allows training and prediction of gestures and data storage.* |

Figure 10, shows the previous bluetooth scanner screen layout which display the detected bluetooth devices. Figure 11, shows the previous classification tab. The past application only had three tabs “EMG, Features, and Classification”. Below is the newest Android Application GUI with a new login screen, IMU tab, and many upgrades to functions and layout esthetics.

|  |  |
| --- | --- |
| *Figure 12. Login Screen in the MyoHMI*  *application, displays the option to sign in.* | *Figure 13. Registration Screen in the MyoHMI application, displays the fields to register a new account.* |

The application starts with a splash screen and a smooth sound displaying the SF STATE logo and the name of the application “Myo HMI”. It then redirects the user to the login screen as shown figure 12. It lets the user input their username and password if they have previously created an account; if not they can always register by clicking the “Register Here” option. The benefit of registering is that it will enable users to upload and retrieve their trained data to or from the cloud, this will prevent the users from going through the training process every time they want to launch the app. Also the data uploaded by the user can be used for Pattern Recognition (PR). If the user would opt not to register, they can still continue by pressing the “Click here to continue” option, but they would not use the new feature implemented. If the user decide to register, he or she would be redirected to the register screen shown in figure 13. There are 4 fields asking for their name, username, password, and age. After filling these fields and selecting “Register”, they will be send back to the login screen to successfully access the application with their new account.

|  |  |
| --- | --- |
| *Figure 14. Bluetooth scanner screen in the MyoHMI*  *application, displays all bluetooth devices detected.* | *Figure 15. EMG tab in the MyoHMI application, displays the name of the armband connected.* |

In Figure 15, a button with the armband logo on the top right corner of the app gives users the options to connect or disconnect to the armband. After clicking connect, it will direct them to Figure 14, where it would automatically start scanning for bluetooth devices and displaying them.

Once the user selects the desired device, it will take them back to the EMG tab. As shown in Figure 15, the EMG tab includes a text message on the center that displays the status of the myo armband, connected or disconnected. It also has a circular image with clickable colored buttons representing each sensor on the device. The graph’s color displaying the raw data output will change according to the selected sensor. The bottom right icon is used to start and/or stop streaming raw data. The icon on the bottom left corner will vibrate the armband to demonstrate its connectivity.

|  |  |
| --- | --- |
| *Figure 16. EMG tab in the MyoHMI application, displays a graph with raw EMG data coming from the connected armband.* | *Figure 17. Features tab in the MyoHMI app,*  *displays selected features in radar graph in real time.* |

In Figure 16, the raw EMG data coming from the top gray color sensor is displayed in a real time graph . This EMG graph is display after connecting the armband and pressing the EMG button located on bottom right corner. In figure 17, the featured tab contains 6 different features from which the user is able to select before training the gestures. In addition, there is a radar graph that displays the values of the features corresponding to each channel.

|  |  |
| --- | --- |
| *Figure 18. IMU tab in the MyoHMI application, displays multiple axis selections, an artificial horizon, and a graph that displays the chosen axis.* | *Figure 19. Classification tab in the MyoHMI application, allows training, prediction of gestures, and data storage/retrieving options.* |

In Figure 18, the IMU tab has a total of 10 options to choose from, four options for orientation and 3 options for magnetometer and Gyroscope. For orientation there are w, x, y, and z axis. As for magnetometer and Gyroscope the options are x, y, and z axis. After selecting an option the graph at the bottom will display the corresponding values in real time. The implementation of the artificial horizon was to visualise and see the dynamic movement of the armband. This is helpful for users to properly position the armband on their arm, and also for future implementation of the Virtual Reality (VR). The classification tab shown in Figure 19, shows the user a list of predetermined gestures and classifiers from which they can choose any combination to train. Moreover, the user has the flexibility to delete or add a custom gesture to the list. After the user has trained the desired classifier according to the particular parameters they have chosen, the live box will output the predicted gesture. Additionally, the user can save the training data onto the device’s local storage or upload it to the cloud storage service with a unique id.

**Data Storage**

Anticipating an increase in data storage and use for this application, we investigated intuitive ways to store and access our data. We previously added all of our feature and emg data in text files and stored each text file. This system lacked organization and accessing the data cleanly required programs to parse the files back into data formats. We since moved to storing in a MySQL database provided by Amazon Relational Database Service (RDS) and hosted on our EC2 instance. MySQL is an open source relational database system that allows any administrator to easily add or access data via text queries. For example, if we want to retrieve all feature data where “user1” was training a “Fist” we could execute the query:

*SELECT \* FROM `featureEMG` WHERE username = `user1` AND gesture\_output = `Fist`*

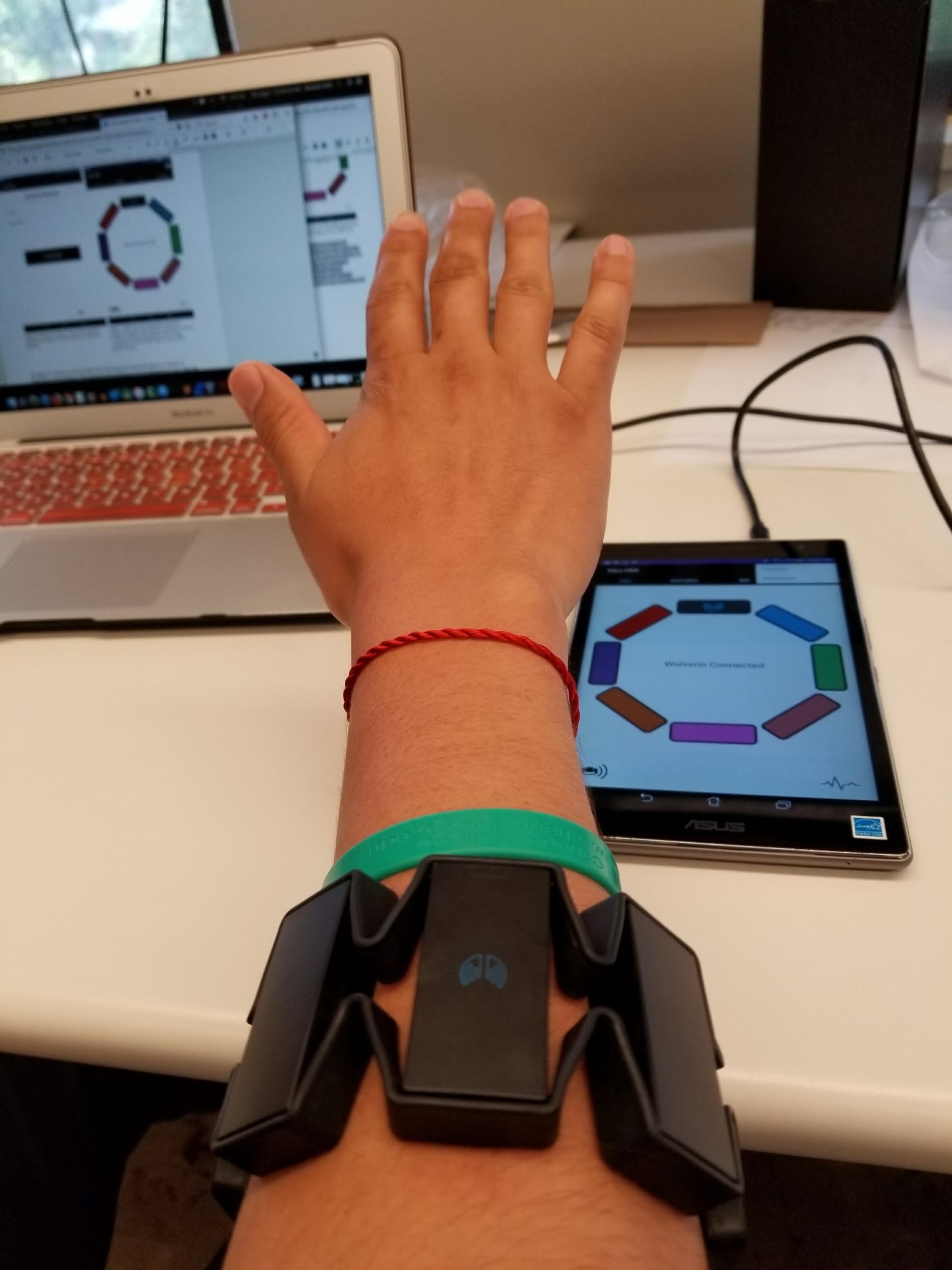
Easily accessing data like this is crucial if we want to build features that process large volumes for various users/gestures.

# V. Results and Analysis

**A. Experiment: User Independent Pattern Recognition**

With a complete SQL database at our disposal, experimenting with data from all of our users becomes seamless. We wanted to see if we could create one pattern recognition model, composed of data from all of our users, that can fit anyone.Our hypothesis states that if we compile large amounts of well controlled data for several hand gestures and train a machine learning algorithm, any human will be able to take this model directly and use it to predict gestures in real time without going through the training process themselves. This process, referred to as User Independent Pattern Recognition (UIPR), has the potential to eliminate the laborious training portion of our gesture recognition platform.

Developing a UIPR system poses a few challenges. The data must be controlled as it enters our database. Before this UIPR endeavor, when a user wanted to use our application they just put the armband on their forearm without thinking and started the training process. Now we must ensure that the orientation of the armband is consistent. The Myo Armband consists of 8 sensors placed around the forearm. If the “Wave In” gesture focuses all muscle activity in the anterior of the forearm, then this must be reflected similarly across the same sensors for all users in order for UIPR to work. The solution is simple, we must control the orientation of the armband for all users, both during data collection and when users intend to use the model in practice. We decided to line up the Myo Armband LED symbol with the middle finger, and keep the charging port facing out. The IMU tilt graphic provides an aide to let users know that the armband is on properly.



*Figure 20. Armband in a consistent position for all UIPR data collection and use.*

**B. Experimental Protocol**

Now we must collect data. In time, we wish to collect data from as many users as possible to see how a very large data volume can affect this process. Making our application open source and available to everyone will help us achieve this goal. For now, we want to collect data from a controlled group and see the effect of data size even on a small scale. We collected data from 10 users, keeping the armband in the position described above and conducted the training process for the following 8 gestures: Rest, Fist, Point, Open Hand, Wave In, Wave Out, Supination, and Pronation. After the training process, the users touch the “Cloud Upload” button which sends all of the collected data to the database to be stored.

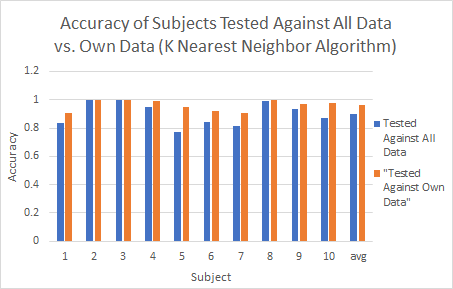
**C. Data Analysis**

For the sake of this experiment, we wrote an additional java program to analyze the data collected. This program can be stored and ran on an AWS server to constantly give us up to date metrics about the performance of our UIPR system. We calculate 6 features per sensor for each sample window, 48 features total. These 48 features are collected 100 times per gesture. With 10 subjects performing 8 gestures each. This means we need to store and process 307,200 features, stored as a double data type. We want to use all of this data to train the UIPR model which takes a great deal of time and computing resources. The training process on a Lenovo Laptop can take up to 10 minutes, depending on the pattern recognition algorithm. We wanted to know how each user performed against the model comprised of all other subjects. To do this, we trained data for all subjects but one (9 subjects total), and tested the left out subject against this model. Specifically, we give features from the one subject to the model, if the model predicts the gesture that the subjects data is labelled with, that sample is predicted correctly. More specifically, we give all feature data labelled with “Fist” to the UIPR model, if the model predicts “Fist” every time, the accuracy of the model is 1.0 for the “Fist” gesture.

We performed a second analysis to see how each subject would perform against a model trained from all 10 subjects, as well as how each subject performed against his/her own data.

**D. Experimental Results**

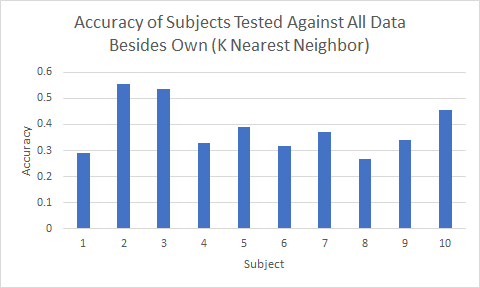
The results of this experiment can be seen below.



*Figure 21. Subjects tested against subject 1 vs. subjects tested against all data.*

The blue bars in **Figure 21.** represent each subject tested against all data while the orange bars represent each user tested against just their own data. The discrepancy between the two is understandable since orange is just data tested against itself it should be high, while blue includes all data making it apply less specifically to the subject. This result tells us that if we collect user data it should be isolated to that user and will have higher accuracies if not combined with data from other users.

The orange bar for subject 1, where he/she was tested against his/her own data, is a bit higher that the UIPR model because this is purely testing data against itself. The rest of the orange bars are lower, our lowest at 0.04, confirming our theory that testing any user against data from only one other user yields low accuracies. When we compile all of the data together and test each user against it the accuracies increase. Some users are near 1.0 and the lowest just under 0.8. Differences in accuracies between users can be attributed to a number of things. It could be that our 0.8 user has outlying forearm size. Other factors could be attentiveness during the training phase, if they are not paying attention they may make the wrong gesture when collecting data. Also, the user might have been careless when aligning the armband in the correct position.



*Figure 22. Each subject tested against all 9 other subject’s data.*

This is the final analysis of the UIPR experiment and tests the feasibility of the concept. If we want to test a model that can be applied to any human, we must exclude each subject’s data from the other 9 as if they are this human. The accuracies are low, too low for this to be used in practice. It seems that discrepancies in forearm size make a user independent model impractical.

**E. Experimental Conclusion**

We hope that as our user base increases and users start contributing data, we can see more solutions to making UIPR possible. For now, the user login allows each user to go through the training process only once, since they can download their saved training data. Making our platform easy to use and practical, and continuing development of the ICELab website, will help draw users to our application. As our data grows, the computations become more demanding and will require more powerful servers which is practical now more than ever. Training this User Independent Pattern Recognition system in the cloud and transferring the model to a local device allows us to migrate all of the intense computations away from our mobile devices. It also has the potential to eliminate the laborious training phase since the model is ready to go at the will of the users.

**F. Future Work**

You may be wondering, “what if the user doesn’t put the armband on correctly, or doesn’t make the proper gestures, and still uploads data”? The answer is bad. That data will contribute towards the corruption of our UIPR system, We can implement some filtering methods by filtering out data sets that return low accuracies using the same program that was used for analysis in this experiment. We also must perform experiments to see which pattern recognition algorithms are best suited for this application.

# **VI. Conclusion**

This project is a continuation of the 2017 Aspires “MyoHMI” project. New additions to the project include the IMU tab, a project website, incorporating data storage in a MySQL database, user logins, and storing user data for training in the cloud. The IMU tab allows users to visualize gestures and motion of the armband. The project website provides instructions on how to run and use the application. Making the application usable by anyone with an android device and armband promotes open source use which hopefully can someday reach those who need this application most. Also, with more users uploading data we can gain more insights to the possibility of user independent pattern recognition from larger data analysis.

A MySQL database was introduced to this project which stores data by row and can be easily accessed by the SQL query language. Having a proper database, opposed to storing data in files, allows us to scale our data analysis efforts. We also introduced the concept of users to the application. A user login allows users to create accounts. The users and their data can be stored in the database allowing users to access their own data and avoid training the application multiple times.

Having the database allowed us to conduct an experiment testing the concept of User Independent Pattern Recognition where one model could be applied to any random user. A model compiled from 9 users and tested against data from one user yields low accuracies, too low to be used in practice. As our user base and data grow we can perform more in depth tests on this concept.

**V. Future Work**

*Implementation with 3rd party Applications and devices*

Currently the app is able to recognize gestures through pattern recognition and and calculate the motion of the armband, which is a step forward for the use of virtual reality and remotely controlled prosthetics.

*Cloud Storage and Cloud Computing*

Right now we are able to store and access user’s independent data in the cloud. The next step would be incorporate the cloud computing to speed up the prediction at real time when using large amounts of data.

*Independent Pattern Recognition*

*Currently we are working on improving the accuracy of the USPR method. From our results we haven't got the best accuracy because of our low number of subjects. When we collect more data we hope to increase our accuracy of the the USPR method.*

# VI. Acknowledgment

This project is supported by the US Department of Education through the Minority Science and Engineering Improvement Program (MSEIP, Award No. P120A150014); and through the Hispanic-Serving Institution Science, Technology, Engineering, and Mathematics (HSI STEM) Program, Award No. P031C110159.

We would like to thank Dr. Amelito Enriquez from Canada College for the opportunity to participate in this internship and for guiding us through the whole program. Also, we would like to express ur appreciation to Dr. Xiaorong Zhang from San Francisco State University, our faculty advisor, and our San Francisco State graduate mentor, Alexander David, for all his guidance and advice throughout the whole internship.

**References**

1. Zhang, X., Chen, X., Li, Y., Wang, Kongqiao., and Yang, J. “A Framework for Hand Gesture Recognition Based on Accelerometer and EMG Sensors.” *IEEE Transactions on Systems, Man, and Cybernetics.* 41.6 (2011).
2. Chang-Kam K., Abad K.,Colin R., Malloy C., Tolentino C., Zhang X., David A. “Developing a Mobile Application and Cloud Computing Framework for Gesture Recognition Using Electromyographic and Inertial Measurement Unit Data.”(2017)
3. https://en.wikipedia.org/wiki/Decision\_tree