Swift-HAR: Human Activity Recognition on iOS (iPhone 6s)

Nond Hasbamrer
Department of Electrical Engineering
Columbia University
nh2518@columbia.edu

Abstract — **Human activity recognition (HAR) is the use of sensors and algorithms to classify human action. Applications of HAR in healthcare involve remote monitoring of patient physical activity, detecting falls in elderly patients, and helping classify movement in patients with motion disorders. HAR can also be used in wearable electronics that track user exercise and activity. In this paper, I demonstrate an HAR application written in Swift 3.1 for the iPhone 6s running iOS 10.3.1. Data logs gathered from the CoreMotion API feeds into a Feed Forward Neural Network created using the Swift-AI framework. Both training and inference from the neural network is executed locally in an iPhone 6s. The neural network can recognize multiple repetitive and continuous activities by changing the number of output nodes to match the number of unique activities the user logs. The trained neural network successfully classifies the tested user activities as standing still, walking, or doing bicep curls.**

Keywords- human activity recognition; swift; iPhone; iOS; user behavior analysis; activity classification; neural network;

I. INTRODUCTION

Human activity recognition (HAR) involves the use of sensor data to classify human action. Traditionally, HAR involves attaching discrete sensors to an individual's body with wires coming from each sensor to a data acquisition board. This type of set up is cumbersome, but possible in a laboratory setting. Outside of the lab, battery powered sensors with data storage can be used to log data for analysis on a separate computer later.

With the advancement low-powered microprocessors and inclusion of accelerometers and gyrometers in modern smartphones, these smartphones have become a possible platform for HAR applications.

II. RELATED WORKS

Recent work in HAR have focused on adding HAR capabilities to common devices that users may already carry such as smartphones. [1], [2], [6], [9], and [11] created an android interface for measuring accelerometer and gyroscope data from a Samsung S2 smartphone. The data from the smartphone is then split into a training set used to train a machine learning classifier and a testing set to evaluate the system. Atlas Wearables created device worn on the wrist that is able to recognize numerous exercises such as bicep curls, box jumps, squats, etc. [4]. However, the device cannot train the classifier. The training is done on a separate server and the trained model is imported into the device. Current work on iOS devices involving extracting accelerometer and gyroscope data from an iPhone using a native app, but then doing the classifier training and classification on an external computer [8]. The goal of this project is to combine data logging, data tagging, data processing, classifier training, and activity classification into a single iOS app.

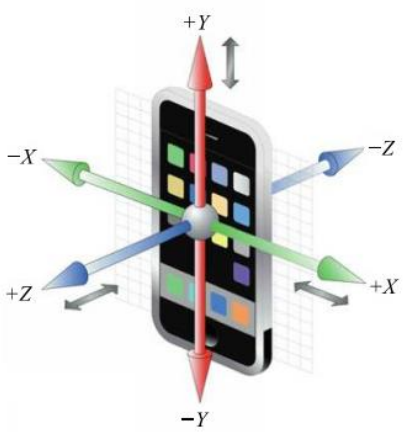

Figure 1: Accelerometer axes on an iPhone 6s [11].

III. MILESTONE PROGRESS SUMMARY

For milestone 1, I researched the different ways user activity could be tracked on an iOS device. Once I decided to pursue HAR using iPhone sensor data, I researched the current work on smartphone HAR along with the frameworks I might need to use to do HAR in iOS. One of these frameworks is the Swift-AI repository by Collin Hundley [7].

For milestone 2, I spent the majority of the time learning how to work with Xcode 8.0 and Swift. I created a single page app that shows live acceleration and rotational data sensed by an iPhone 6s.

In milestone 3 focused on data logging, analysis, and classification. I added features for data logging, exporting, and bootstrapping. I set up and trained the neural network used for binary activity classification (standing still and walking). I also updated the single page UI to accommodate all these features. This version of Swift-HAR was tested by logging tagged data for walking and standing, training the classifier, and then letting the classifier infer 20 untagged actions (10 walking trials and 10 standing still trials). The classifier correctly identified 20 out of 20 activities.

Milestone 4 (the final course project), includes updates to the UI, data logger, and classifier to accommodate activities such as standing, walking, and bicep curls. The updated UI can be seen in Figure 2. The neural network can now adjust the number of output nodes depending on how many different activities were logged. The neural network and the data logger now utilizes data for all three accelerometer axis (previous versions only looked at the z axis). Finally, the app now runs on Swift 3.1 instead of Swift 3.0.

IV. Software Package Overview

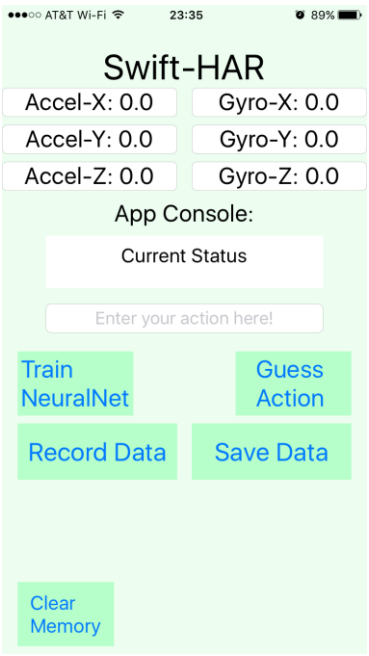

Figure 2: Milestone 4 UI. Current UI which includes all the buttons and components described in section V. The "Record Data" button initiates the data logging routine and tags the data with an action label from the "Enter your action here!" text field. The "Train NeuralNet" button trains the neural network based on the available training data in memory. The "Guess Action" button starts logging data and then gives an inference for what Swift-HAR thinks that action was. "Clear Memory" erases all data stored in memory.

An overview of the system can be seen in Figure 3. The Swift-HAR app uses Swift 3.1 and runs on iOS 10.3.1 or later. The app was written using Xcode 8.3.2. The app accesses accelerometer and gyroscope data using the Swift CoreMotion API. The classifier is a feed-forward neural network (FFNN) created using the NeuralNet class of Swift-AI [7]. Currently, Swift-HAR classifies user action based only on the tri-axis accelerometer data (see Figure 1 for iPhone axes information). At this time, Swift-HAR only supports repetitive and continuous activities such as walking, running, standing still, bicep curls, jump roping, etc. Swift-HAR does not support non-periodic activities such playing tennis, playing basketball, fencing, golfing, etc.

To record training data, the user enters the tag for the activity in the text field that says "Enter your action here!" and presses the "Record Data" button. The user then has 2 seconds to start doing the activity. An audible "beep" signals that Swift-HAR has started recording data. A second "beep" signals that Swift-HAR has finished recording data for that activity (~15 seconds after the first beep). The recorded acceleration data for the three axes is then exported into a csv file saved in the app's document folder as well as kept in memory.

Once the user is done recording different activities, pressing "Train NeuralNet" will take all of the recorded data stored in memory and use it to train a classifier. Each activity's data set will be bootstrapped into a training and testing set along with the user's activity tag. The data then feeds into a 3 layer feed-forward neural network with 300 input nodes (100 nodes for each axis), 240 hidden layer nodes, and a number of output nodes equal to the number of unique activities the user has logged.

After training, the NeuralNet is ready to classify user activities. The user can ask Swift-HAR to guess an activity by pressing the "Guess Action" button. Swift-HAR will then log data for ~15 seconds, and provide a guess of what activity the neural network thinks the user just performed.

V. Software Package Details

Code for the UI is contained in "Swift-HAR/Swift-HAR/Base.lproj/Main.storyboard". The UI was built for an iPhone 6s in portrait orientation.

The main entry point to the code is in "Swift-HAR/Swift-HAR/ViewController.swift". This ViewController class contains all the variables and information used by the app including connections to UI elements. The marked variable outlets contain all the text that appear in the UI. The variable "currentAppStatus" acts like a console in the UI that displays what the app is doing. For example, messages like "Data logging started," "Data export complete," and the NeuralNet predictions gets pushed to the UI using this variable. The text field "actionText" links to the text field that users can input an action tag. The other text fields display live readings from the accelerometer and gyrometer.

```
// MARK: Var outlets

@IBOutlet weak var accelXText: UITextField!
@IBOutlet weak var accelYText: UITextField!
@IBOutlet weak var accelZText: UITextField!
@IBOutlet weak var gyroXText: UITextField!
@IBOutlet weak var gyroYText: UITextField!
@IBOutlet weak var gyroZText: UITextField!
@IBOutlet weak var currentAppStatus: UITextView!
@IBOutlet weak var actionText: UITextField!
```

Upon starting the app, the first function that loads is the viewDidLoad() function. This function establishes initial variables such as the initial texts of all the UI elements. This functional creates the preliminary structure and configuration of the untrained neural network and then initializes an instance of the class.

The function also pre-allocates space in memory for all the data arrays by creating arrays of the correct size filled with null values. Then the function checks whether an accelerometer and gyrometer is available and then sets up the data query frequency. Currently the app checks the sensors at a rate of 100 Hz.

After the initialization with viewDidLoad(), the behavior of the app depends entirely on user input. Pressing "Record Data" will call the "recDataButtonPressed" function which logs the action tag from the actionText field. After a 2 second delay, the function plays an audible "beep" and initiates accelerometer data logging for that action. After 15 seconds since the button was pressed, the function plays another beep, exports all the recorded data to a csv file, checks to see if data for the current action already exists in memory. If data for the current action exists, that data gets overwritten by the new data, if not, the recorded data gets appended to the overall dataMatrix. The same goes for the action tag.

The button "Train NeuralNet" calls the function "trainNNBPressed." This function is a wrapper for calling two other functions, "nnInit" and "trainNNMisc". The function "nnInit" modifies the neural network by changing the number of output nodes to match the number of actions in the actionsLog. The "trainNNMisc" function takes the large array, dataMatrix, containing all previously logged action data and calls the "triAxisBootstrapper" function in order to create training and testing data for the neural network. Then, the training and testing data gets pushed into the neural network's training routine which uses an error threshold of 0.05 and has a maximum training time of 30 epochs. The maximum number of epochs is set to 30 to allow the training routine to exit in case the routine gets stuck at a local minimum.

The boot strap function "triAxisBootWrapper" processes the data and creates multiple points for training and testing by taking one second long samples from the original 10 second data starting at varying points in time. Since the activities that Swift-HAR recognizes are assumed to follow a repetitive and continuous cycle, data samples created by bootstrapping is assumed to be representative of the overall population. More details of what the raw data looks like and

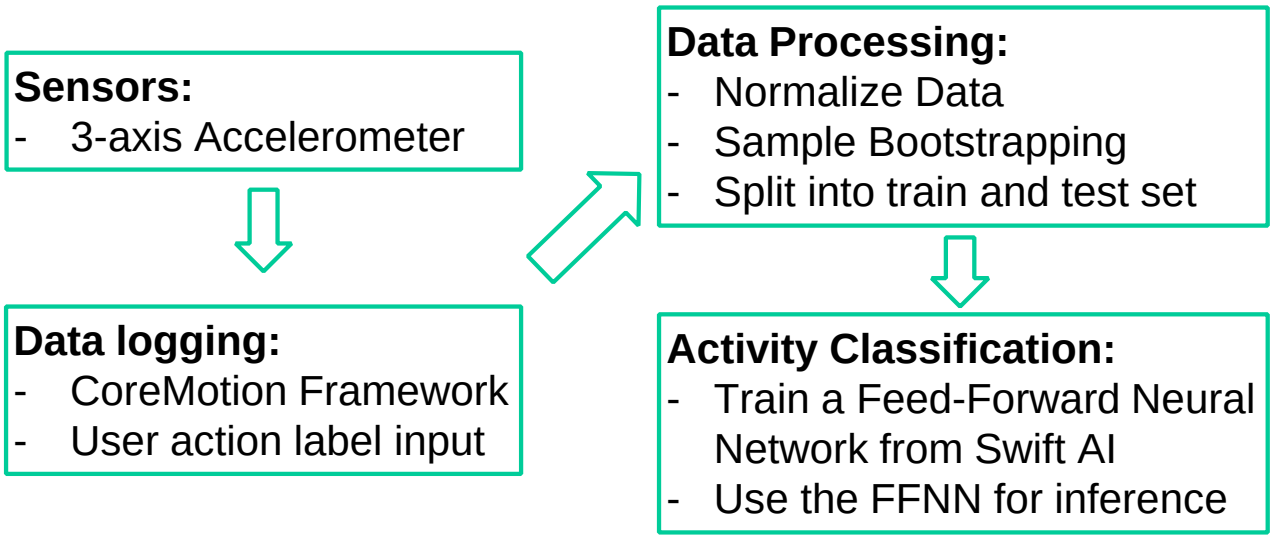

Figure 3. High-level System Overview. The iPhone 6s contains a 3 axis accelerometer and a 3 axis gyroscope. Currently Swift-HAR only uses data from the 3 axis accelerometer. Swift's CoreMotion API provides access to data from these sensors. A data logger records samples of motion data along with an action tag given by the user then passes the data into a processing and bootstrapping function. The bootstrap function combines the recorded and tagged data into a training data set to train a Feed-Forward Neural Network (FFNN) and a test set to test the neural network's classification. After the network is trained, the FFNN can classify untagged motion data according to the available set of tags.

how the bootstrapper applies additional process can be found in section VI.

The "Guess Action" button calls the function "guessActBPressed" which records data for 10 seconds and then calls the infer method of the NeuralNet class to predict what action the user did in the past 10 seconds. The possible guesses are the values are the set of actions in the actionsLog. This button only works after the NeuralNet has been trained.

The output of the NeuralNet is an array called "inference" with n number of 32-bit floating point numbers where n is equal to the number of unique actions in actionsLog. For example, if three actions were recorded, the actions might be encoded as follows:

$$Stand = [1, 0, 0]$$

$$Walk = [0, 1, 0]$$

$$bicepCurls = [0, 0, 1]$$

The values in the array of the real output from the NeuralNet can take values ranging from 0 to 1. The app finds the index of the maximum value in the inference array and matches that to the index of actions in the actionsLog. The prediction then gets pushed through the "currentAppStatus" outlet and displayed in the App Status field.

VI. Experiment Results

Figure 4: Forearm band. For all trials and tests, the iPhone 6s was secured to the user's forearm using a VUP+ 180° Rotatable Sports Wristband. This allowed the iPhone to move with user while allowing the user to still interact with the controls in the GUI.

For all trials and tests, the phone was secured to the user using a forearm band as seen in Figure 4. The raw data recorded for standing still, walking, and doing bicep curls can be seen in Figure 5. The logged acceleration gets adjusted so that the minima of the sample is approximately 0.0g. This helps the NeuralNet focus on the variability in the signal rather than the vertical shift. A sample of the process data can be seen in Figure 6.

VII. Conclusion and Future Work

Swift-HAR includes features for logging 3-axis accelerometer data from an iPhone 6s running iOS 10.3.1, exporting the data to a CSV file, training a feed-forward neural network classifier, and classifying what action the user is doing. Currently, Swift-HAR supports only actions which are repetitive and continuous (such as walking, running, standing still, bicep curls, sit-ups, etc). The data logging, classifier training, and inference are all processed locally on the iPhone 6s. The data logger collects accelerometer data at 100 Hz. The classifier is a 3-layer feed-forward neural network with 300 input nodes, 240 hidden layer nodes, and variable number output nodes dependent on the number of unique actions previously recorded and tagged by the user. Using the current tuning parameters for the classifier, the NeuralNet successfully classifies whether a user is standing, walking, or doing bicep curls.

Even though the classifier can predict whether a user is standing, walking, or doing bicep curls, known issues do exist. Certain training data sets yield inaccurate predictions. Furthermore, adding more and more unique actions to the classifier seems to reduce the accuracy of the predictions. The classifier also has difficulty telling the difference between actions that have very similar acceleration magnitude and frequency. Adding additional accelerometer data processing steps such as a Fast-Fourier transform along with data from the gyrometer could help increase the accuracy of the classifier. Remedies to these known issues could be the subject of future work in the area of iOS based human activity recognition. Future work could also include code refactoring of the Swift-HAR which moves most of the functions from the view controller to a separate class. This will help streamline future feature additions.

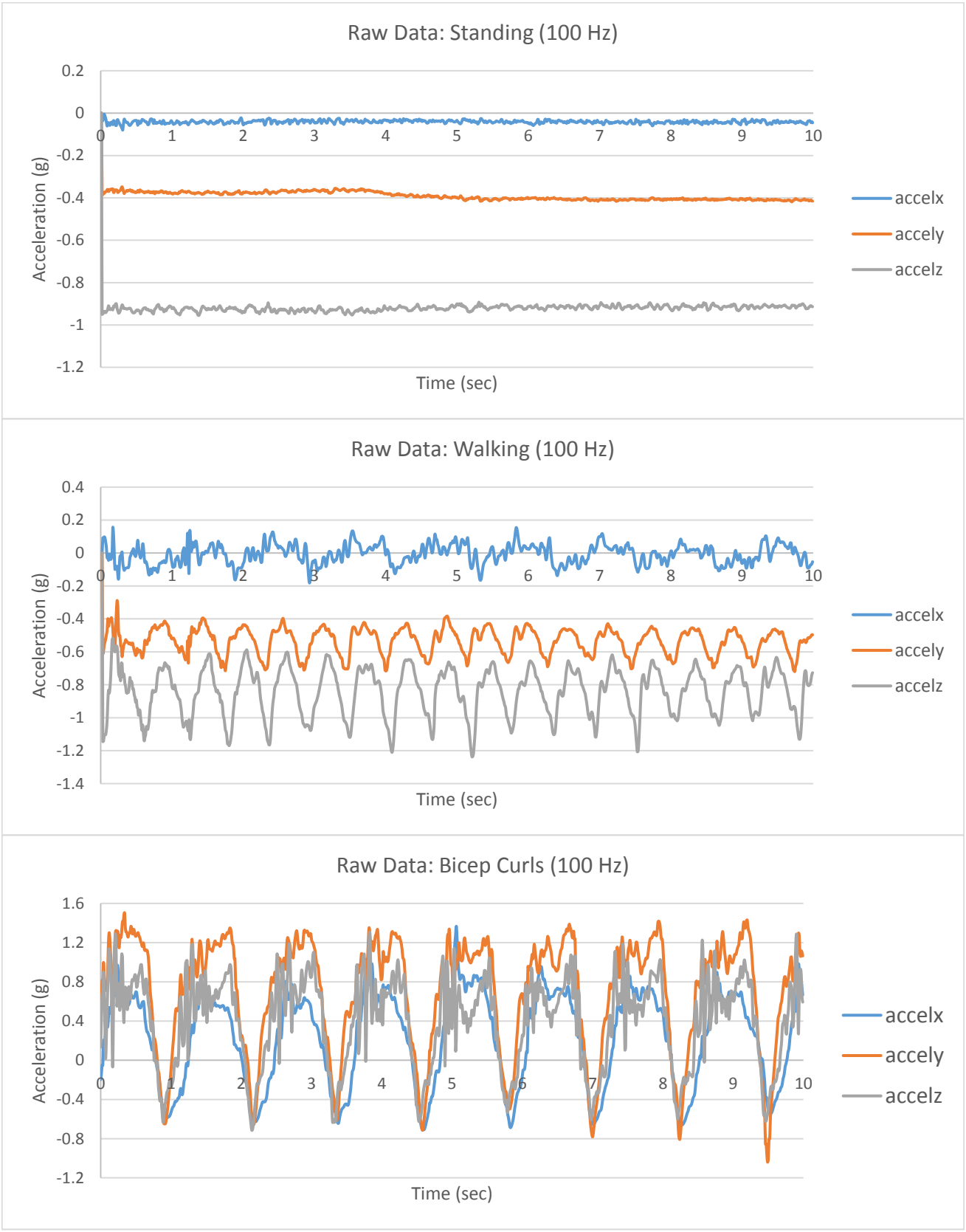

Figure 5: Raw data from the accelerometer of a user standing still (top), walking (middle), and doing bicep curls (bottom) with the phone secured to the forearm using a forearm band.

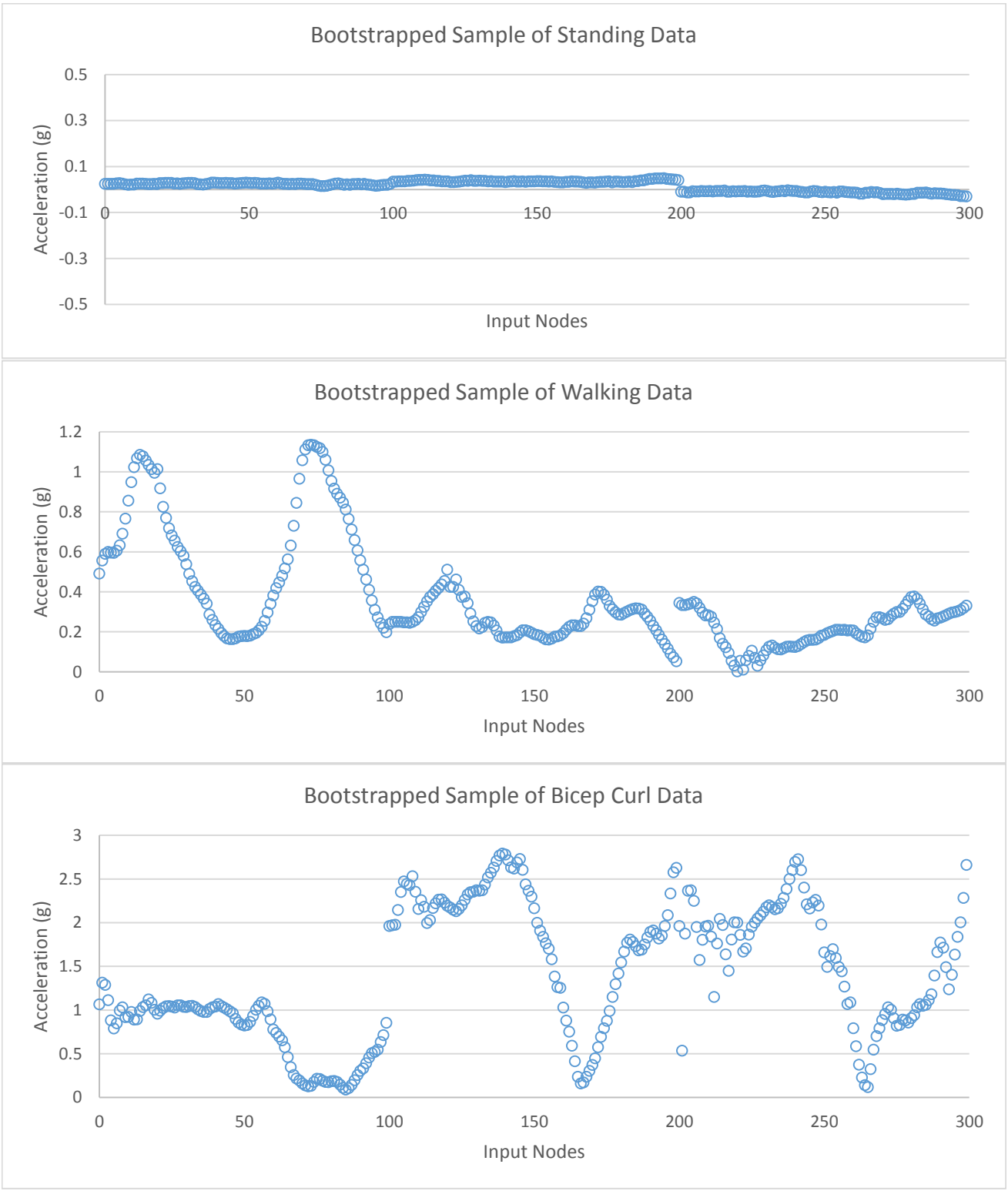

Figure 6: Processed data of a user standing still (top), walking (middle), and doing bicep curls (bottom). The bootstrapper creates a 1 second sample from the 10 seconds of recorded data. The data is also adjusted, so that the mean is approximately 0.0 g. Nodes 0 to 99 encapsulates the 1 second sample for the x axis. Nodes 100 to 199 encapsulates the 1 second sample for the y axis. Nodes 200 to 299 encapsulates the 1 second sample for the z axis.

APPENDIX :

The complete software package is available at: https://github.com/Aneapiy/Swift-HAR

REFERENCES :

[1] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A Public Domain Dataset for Human Activity Recognition using Smartphones," in *ESANN*, 2013.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Energy Efficient Smartphone-Based Activity Recognition using Fixed-Point Arithmetic," *J. UCS*, vol. 19, no. 9, pp. 1295-1314, 2013.

[3] Apple. (2017, February 23, 2017). API Reference. Available: https://developer.apple.com/reference/

[4] A. Wearables. (March, 30). *Wristband 2*. Available: https://www.atlaswearables.com/wristband2/

[5] K. Coble, "AI Toolbox," ed, 2017.

[6] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," presented at the Proceedings of the 4th international conference on Ambient Assisted Living and Home Care, Vitoria-Gasteiz, Spain, 2012.

[7] C. Hundley. (2017). *Swift-AI/NeuralNet*. Available: https://github.com/Swift-AI/NeuralNet

[8] B. Remseth and J. Jongboom, "Live analyzing movement through machine learning," in *Engineering @TelenorDigital* vol. 2017, ed, 2015.

[9] J. L. Reyes-Ortiz, D. Anguita, L. Oneto, and X. Parra, "Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set," ed. UCI Machine Learning Repository, 2015.

[10] R. Smith and J. Ho. (2017, February 23, 2017). *The Apple iPhone 6s and iPhone 6s Plus Review*. Available:

[11] X. Su, H. Tong, and P. Ji, "Activity recognition with smartphone sensors," *Tsinghua Science and Technology*, vol. 19, no. 3, pp. 235-249, 2014.