STMicroelectronics SensorTile Reference Design:  
Hook Shot Trainer

Ziyue Yang & Zhitong Qian

Table of Contents

[**Introduction**](#_30j0zll)3

[**System Introduction**](#_4v53p95cx7ep)3

[**Motivation of the System**](#_1018e1kju6li)3

[**System Architecture**](#_o3ozmv2d453x)3

[**Sensors of the System**](#_lcx8dvv4o2d2)3

[**Characteristics of the System**](#_d7u833yalzeq)5

[**Preparation**](#_x3pruelpqf9k)5

[**System Architecture**](#_1t3h5sf)5

[**Code System**](#_isx2ls2cin9u)5

[**Dual Data Acquisition**](#_6t4ge3srwzun)6

[**System Sensor Data Acquisition and Experimental Methods**](#_17dp8vu)7

[**Hardware and Software Setup**](#_7odahprp9ovg)7

[**Experimental Methods for Training and Testing**](#_jho20cqwno64)7

[**System Sensor Data**](#_lnxbz9)10

[**This describes your sensor data sources and includes example data figures**](#_35nkun2)10

[**Signal Feature Extraction**](#_1ksv4uv)11

[**This describes feature extraction methods.**](#_44sinio)11

[**Neural Network Implementation**](#_2jxsxqh)16

[**Use of FANN Library**](#_dugaw7dtiniz)16

[**Demonstration**](#_3j2qqm3)16

[**Video Demo**](#_1y810tw)16

[**The video demo is in the following link:**](#_ovhxrbpv0bdq)16

[**Source Code**](#_1ci93xb)21

[**Source Code Setup**](#_osmebfyi9o64)21

[**Source Code Description**](#_2bn6wsx)22

[**Design Guidance**](#_1pxezwc)28

[**Development Process**](#_3eqcp85j09fx)28

[**Challenges**](#_z6jfks9cuw7)28

[**Advice for New Users**](#_g1mh8qj3ez61)28

[**Reference**](#_fb8i4ton2sry)29

# Introduction

## System Introduction

This reference design introduces a Machine Learning system using the BeagleBone platform and SensorTile wireless sensor system. The goal is to build a hook shot trainer that can train a average person to perform a correct basketball hook shot. This reference design can provide you with a great introduction and experience to the IoT Machine Learning systems and its future potential implementations.

For this reference design, we will first familiarize you with the hardware and software configuration including the installation of the necessary data acquisition software and basic hardware setup. We will then develop the neural network system in the following order: 1) Hardware and software setup. 2) Data acquisition via BLE communication from SensorTile and parse the data. 3) Train the system for different hook shot gestures. 4) Test the performance of the system.

## Motivation of the System

Nowadays, many sports start to rely on technology to help training athletes. In the realm of basketball, such system can help player to increase the on-shot rate. Though there are many similar systems training on shooting, hook shot is always ignored. Therefore, we want to design a system that can help user to learn how to perform a standard hook shot.

## System Architecture

The system is mainly composed of three parts, the two SensorTile, the BeagleBone, and a personal computer. The two SensorTiles are used as sensors collecting the data of user’s motion, including acceleration data of x, y, z axis and gyroscope data of x, y, z axis. The collected data is transmitted through Bluetooth to the BeagleBone and the program inside BeagleBone will use trained neural network to do classification on user’s data to give suggestions, which will be printed on the screen of the personal computer.

## Sensors of the System

The sensors are the SensorTiles. One will be put on the back of the right hand and another will be put on the forearm of the right arm. The following figures shows how to wear the two SensorTiles.



*Figure 1. The two sensors*



*Figure 2. Wearing two sensors*

## \*Make sure to place the SensorTile in the position where the exposed sensor is facing downwards. If you want to use your own orientation methodology, always check if the orientation is right before conducting any data acquisition, training or testing process. Consistency is the key factor here.

## Characteristics of the System

The system collects user’s motion data from two SensorTile simultaneously and write the data into two csv files. Then the program on BeagleBone will extract the data from the csv files and use the trained neural network to determine the three major characteristics of a standard hook shot: the speed of raising arm, the flip of the wrist, and whether the arm is raised straightly. After testing phase, the suggestions regarding the three characteristics will be displayed on the screen.

## Preparation

**Prerequisite Tutorials:**

Users should ensure they finish BeagleBone tutorial 1 – 8, SensorTile tutorial 1 – 8, synchronized data acquisition, and IOT machine learning reference design before proceeding.

**List of Required Equipment and Materials**

• 1x BeagleBone Green Wireless

• 2x SensorTile

• 2x USB 2.0 A-Male to Micro B Cable (micro USB cable)

• 2x USB 2.0 A-Male to Mini Cable (mini USB cable)

• 2x Arm strap

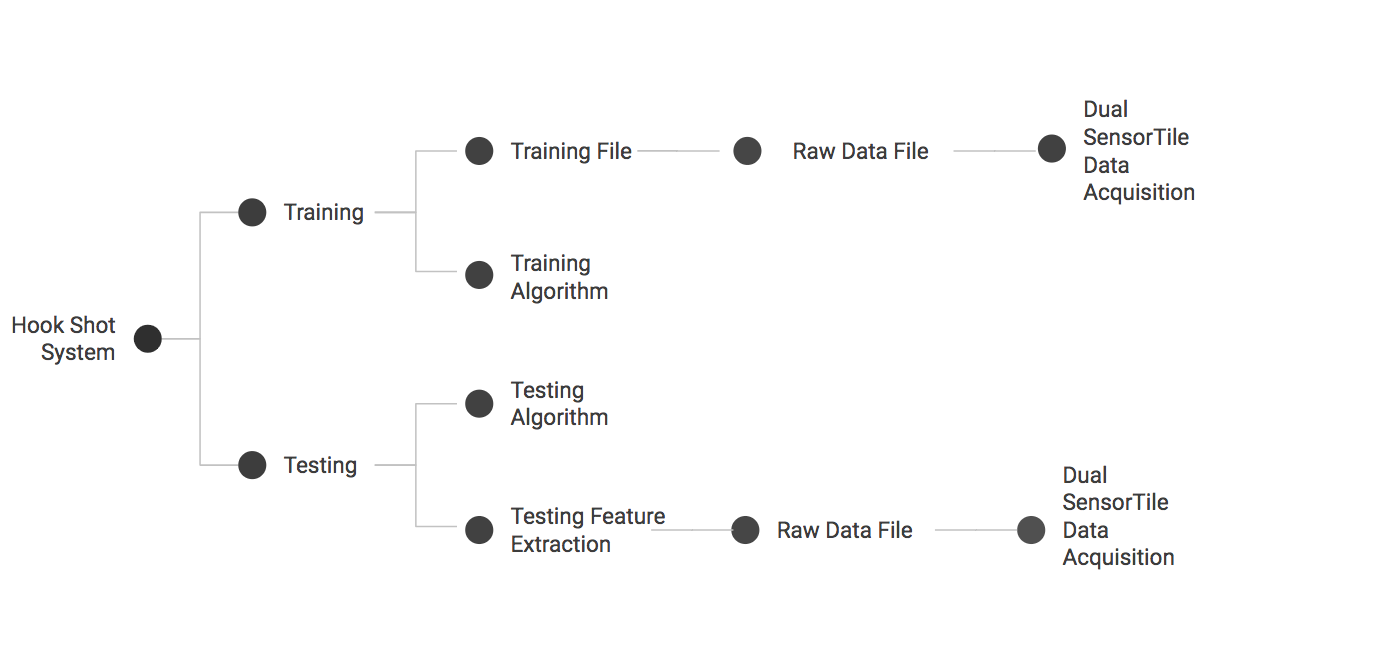
• A personal computer

• Access to network with internet connection

# System Architecture

## Code System

The code system mainly consists of two phases, the training phase and the testing phase. The architecture is shown in the following figure.



*Figure 3. Code System Architecture*

The training phase first extract user motion data from the two SensorTiles and write it into raw csv data files. Then we normalize and analyze the raw data file and write to the training file. There are 3 training files. The first training file corresponds to the speed of raising arm, the second correspond to the motion of wrist flipping, and the third is about whether the user move their arm straightly. Inside the training file, it contains the raw data, particularly the min and max value of certain acceleration or gyroscope data. It will combine with the training algorithm. The training algorithm is a neural network implemented by the FANN library.

The testing phase is like the training phase. It will first use Dual SensorTile Data Acquisition program to extract the raw data and write it to the csv files. Then our program will use functions to extract the features, min and max value of acceleration and gyroscope data, and pass it to the testing algorithm. The testing algorithm will use the neural network trained the training phase and classify the motion of users. Finally, our program will give users suggestions on the motion based on the output of the neural network.

## Dual Data Acquisition

The Dual Data Acquisition System is provided by EE180D instructor William Kaiser and EE180D TA Zhang Xu. It enables our system to extract data simultaneously from the two SensorTile, which can give use more data from the motion of hook shot.

On the basis of the code provided by Zhang Xu and Prof. Kaiser, we modified the source code and add time data into it, which can help us analyze the feature.

The SensorTiles will collect data from the motion transmitted to the BeagleBone and the raw data will be written to two separate csv files. The first csv file corresponds to the motion data of user’s wrist. The second csv file corresponds to the motion data of user’s forearm. There are 7 columns in each csv file, columns correspond to time, x-acceleration, y-acceleration, z-acceleration, x-gyroscope, y-gyroscope, z-gyroscope.

# System Sensor Data Acquisition and Experimental Methods

## Hardware and Software Setup

**SensorTile**

1. Flash the firmware FP-SNS-ALLMEMS1 to your SensorTiles according to SensorTile

tutorial 8 - **STMicroelectronics SensorTile Tutorial: Introduction to Motion Data**

**Acquisition via BLE Communication**.

2. Solder each SensorTile on the cradle board, connect it to the battery and put it into the plastic box.

3. Power your SensorTiles.

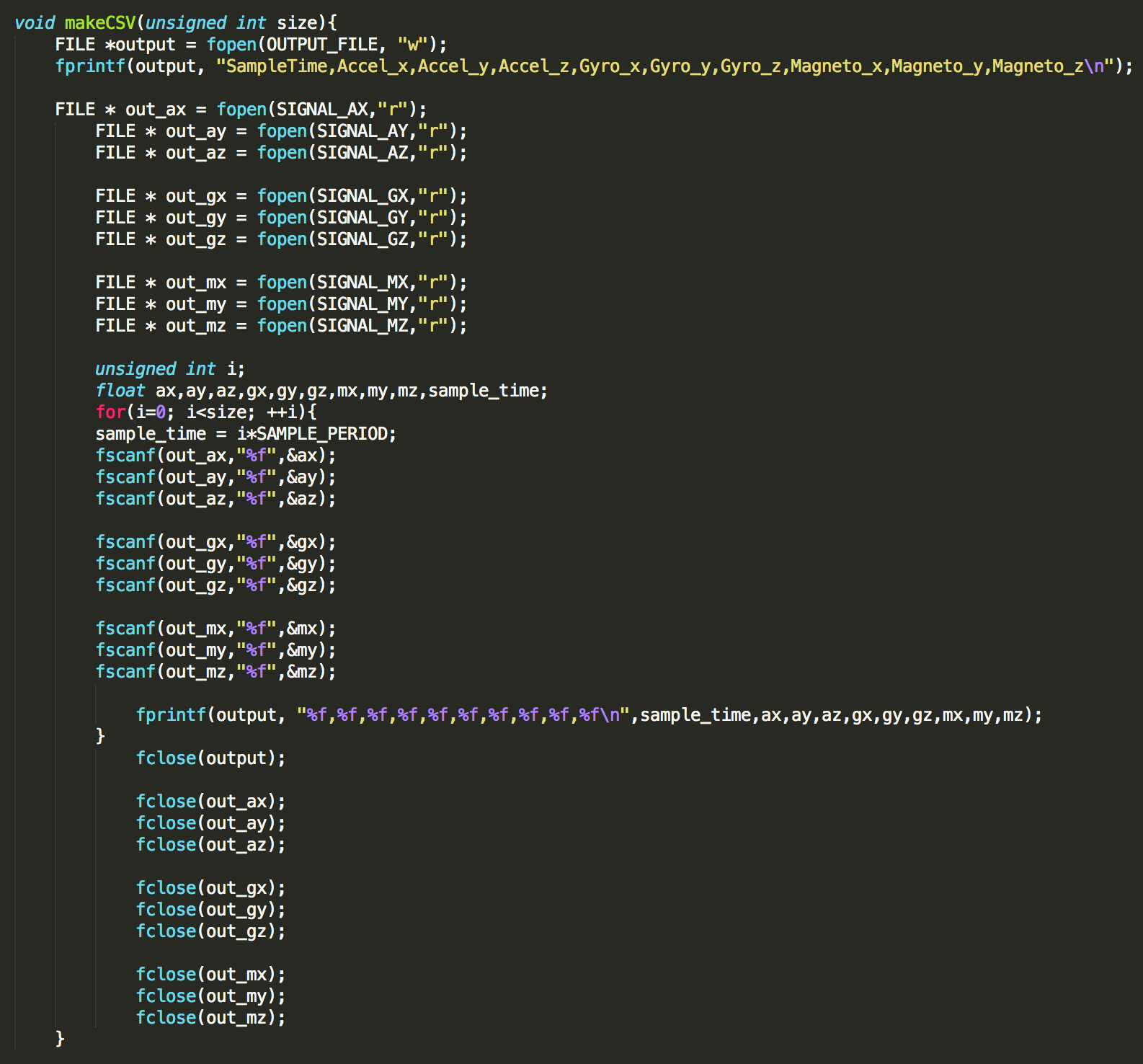
**BeagleBone**

1. Connect your BeagleBone to a Wi-Fi network with Internet access.

## Experimental Methods for Training and Testing

**Training method:**

To train the neural network, first a coach with standard gesture of hook shot will perform hook shot for 10 times. Then he will intend to make some mistakes. The first mistake is raising the arm too fast. The second mistake is raising the arm too slow. The third mistake is not flip the wrist. The fourth mistake is not lifting arm straightly. For each mistake, the coach will do it times. After this process, we have collected 10 sets of data for correct motion, and 10 sets of data for each incorrect motion. We will then use those data for training network. Following steps should be followed.

1. Put the two plastic boxes with soldered SensorTile kit into the arm straps.
2. Fix the position of the two plastic boxes by inserting some filler such as paper
3. Open your BeagleBone, navigate to the directory where you installed the synchronized data acquisition tutorial.
4. Edit the two files motion\_data\_sensortile\_1.sh and motion\_data\_sensortile\_1.sh, change the device ID in the two files to your own SensorTile device id.
5. Open the Acquire\_LowPass\_Contunuous\_1.c, edit the function **void makeCSV** like the following.

*Figure 4. Code to Be Modified*

Here, we edit the original c file and add an additional column of output to provide information of the timestamp of each data, which can help us to analyze the motion feature later.

1. Apply step 6 to the file Acquire\_LowPass\_Contunuous\_2.c to make same changes.
2. After we have modified the source code in the two c files, in order to make it work, we need to recompile the executables. Please use the following command in your BeagleBone.

**gcc -o Acquire\_LowPass\_Continuous\_1 Acquire\_LowPass\_Continuous\_1.c -lm -lc -lliquid -lrt**

**gcc -o Acquire\_LowPass\_Continuous\_2 Acquire\_LowPass\_Continuous\_2.c -lm -lc -lliquid -lrt**

1. Make sure your two SensorTile has power, you can check it using the mobile app **ST BlueMS**.
2. Now wear the two arm straps on your right arm tightly, one on the wrist and another on the forearm.
3. On your BeagleBone make sure both SensorTile is paired but not connected with your BeagleBone.
4. Take the motion detection using the following command

**sensor\_sample.sh -t <TIME IN SECONDS>**

TIME IN SECONDS represents the time in seconds you want to wait before your motion. You should apply a standard hook shot just like the video in the following link:

[**https://www.youtube.com/watch?v=7we0Dob8UPY**](https://www.youtube.com/watch?v=7we0Dob8UPY)

After about 10 seconds of data collection, you will have a full motion dataset in the two files **motion\_data\_output\_1.csv** and **motion\_data\_output\_2.csv.** It contains the information of timestamp, acceleration information in x, y, z directions, gyro information in x, y, z directions.

1. Repeat the step 9-11 another 9 times and properly save the collected datasets.
2. Repeat the step 9-12 for other 4 kind of wrong motion, raising arm too fast with a correct movement of the wrist, raising arm too slow with a correct movement of the wrist, move your arm in a circular track，move your arm without moving the wrist.
3. Now, you have 50 data sets, you can collect more for that, and we will start to analyze the feature of it in the following part.

**Testing method:**

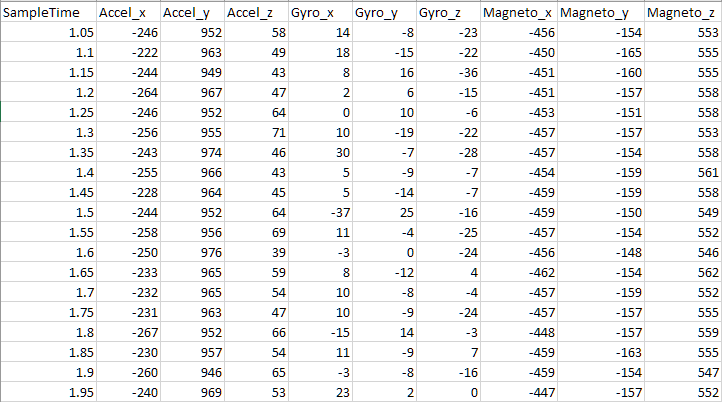
To perform testing, the user will wear the two SensorTiles and perform a shot. The system will collect motion data, write to csv files, and our program will perform classification on the extracted data.

# System Sensor Data

## Sensor Data Description

After we successfully acquired the ability to extract data from both SensorTiles simultaneously, we can start the data collecting process. It’s important for us to understand the raw data acquired from the SensorTiles to select the best feature and see the motion pattern.

After we implemented the data acquisition process, we will get csv files named **motion\_data\_output\_1.csv** and **motion\_data\_output\_2.csv**. We can open these csv files using the text editor inside the BeagleBone interface, or we can copy the files to our own desktop and use software like Microsoft Excel. The raw data will look like the following:



*Figure 5. raw data csv file*

The data is quite straightforward to look at. First, we see the first column represents the sample time of our system. Columns 2 through 4 represent the accelerometer data of the 3 different axes of the SensorTile. Columns 5 through 7 represent the gyroscope data of the 3 different axes of the SensorTile. Columns 8 through 10 represent the magnetometer data of the 3 different axes of the SensorTile.

One thing to notice here is that always make sure your SensorTile’s orientation is correct and consistent for your data acquisition process, otherwise the data can be inconsistent and confusing for you to analyze.

# Signal Feature Extraction

## Feature Extraction Methods

To better understand the feature selection process for hook shot trainer, you must understand the motion of a correct hook shot. A hook shot can be decomposed into two different motions. The first motion is to lift your arm straight up carrying the basketball and the second motion is to flip your wrist and shoot the ball into the basket. The reason we decomposed hook shot into two parts is that the decomposition enabled us to better select the proper feature for these two totally different motions so that we can more accurately categorize different hook shot motion and give the correct instruction and feedback.

Before we start the feature extraction process, we need to categorize the hook shot motion and determine how many different patterns we want to recognize with our system. And for our system, we chose 3 different features to fulfill our motion categorization purpose.

Our Features are:

* The y-axis acceleration data of the forearm sensor.
* The x-axis gyro data of the wrist sensor.
* The z-axis gyro data of the forearm sensor.

The first y-axis acceleration sensor data can help us determine the speed of our arm lifting, which has 3 results: too fast, too slow and correct. The second x-axis gyro data can help us determine if we correctly flip our wrist or not, which has 2 results: yes and no. The third z-axis gyro data can help us determine if we life our arm correctly, which has 2 results: yes (straight up) and no (circular motion). And with all the combinations of these 3 features, the system can recognize a total of 12 different hook shot motion pattern. For the purpose of simplicity, we chose 4 motion patterns to demonstrate the feature extraction process.

**Feature 1: Y-axis acceleration data of the forearm sensor**

* + - 1. Standard hook shot

*Figure 6. raw acceleration data of forearm sensor*

* + - 1. Too fast

*Figure 7. raw acceleration data of forearm sensor*

* + - 1. Too slow

*Figure 8. raw acceleration data of forearm sensor*

For the first feature, we need to focus on the y-axis acceleration data, which is marked in the color red on these 3 figures. We can see that for fast, slow and standard, the first peak value of the y acceleration is different. So, this feature can help us easily identify the speed of our arm lifting and by setting a suitable threshold value in our algorithm, we can classify between these 3 outcomes.

**Feature 2: X-axis gyroscope data of the wrist sensor**

With correct wrist flip

*Figure 9. raw gyroscope data of wrist sensor*

Without correct wrist flip

*Figure 10. raw gyroscope data of wrist sensor*

For the second feature, we need to focus on the x-axis gyroscope data of the wrist sensor. As we can see from the figures above, when we perform a correct wrist flip, we can see a negative peak much larger than when we don’t perform a correct wrist flip, which enable our system to classify between these two different hook shot gesture.

**Feature 3: Z-axis gyroscope data of the forearm sensor**

1. Arm lifted straight up

*Figure 11. raw gyroscope data of forearm sensor*

1. Arm with wrong circular motion

*Figure 12. raw gyroscope data of forearm sensor*

For the third feature, we need to focus on the z-axis gyroscope data of the forearm sensor. We can see from the figures above that when we lift our arm straight up, the z-axis gyro data will remain small. However, if we don’t do the correct gesture and make a circle with our arm, a big peak value will appear for the z-axis gyroscope data. Hence, we can easily classify between these two different motions using this feature.

# Neural Network Implementation

## Use of FANN Library

The FANN Library is used to build training and testing algorithm. The training function in the source code will take the three training data sets, and output .net file, which will be used later for testing.

In the testing phase, the function will use the .net file generated in the previous part and calculate the chance of each class. Then, it will output the class with the highest probability.

# Demonstration

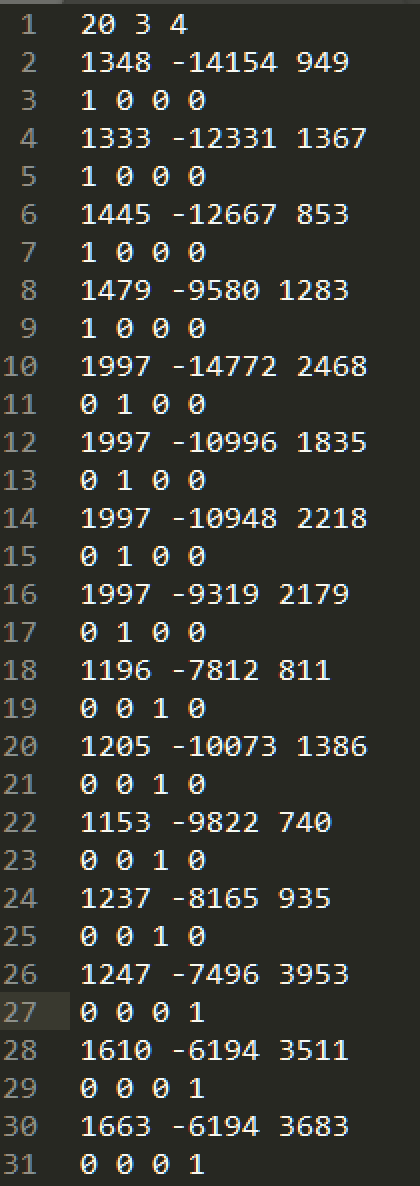
## Video Demo

## The video demo is in the following link:

<https://drive.google.com/file/d/1JbuduhmsWVLXojY-U_p8ngamZ6AzSu6_/view?usp=sharing>

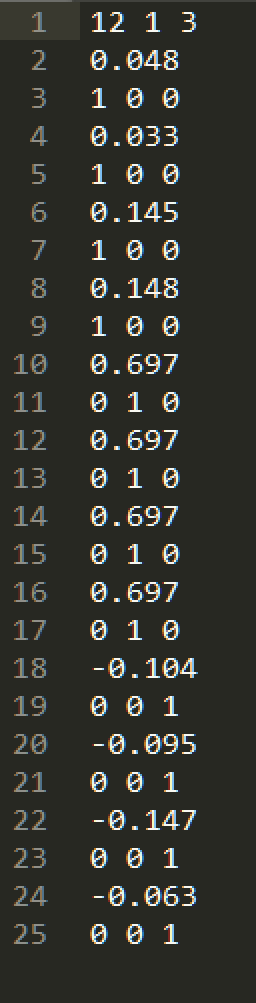
## Code Implementation

1. Sample training data set



*Figure 13. Training data set*

1. Normalized training data set

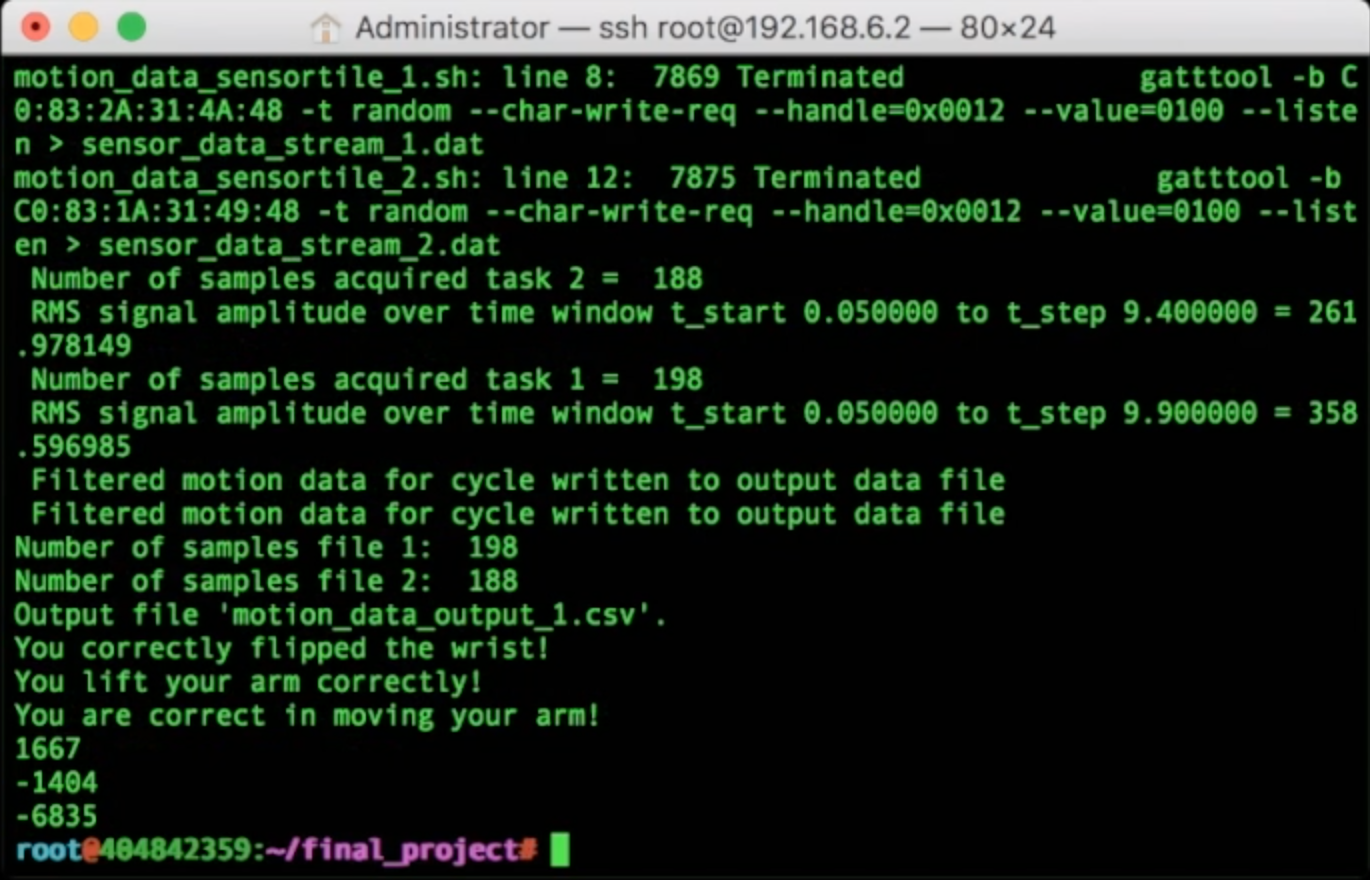


*Figure 14. Normalized training data set*

Notice that in the training data set section, the normalization methods can vary greatly depending on the data you acquired from your own implementation. Generally speaking, you want to normalize the data to -1, 0, 1 interval so that there’s a big enough separation to let the neural network know the difference, which will make the training results much more promising.

1. BeagleBone System UI

We designed our system that when you implement the “./hook” command to begin the training process, it will initialize a 5 second delay, after which you can start to perform your hook shot. After the motion is completed, you can wait for the following code to show up and analyze the result:



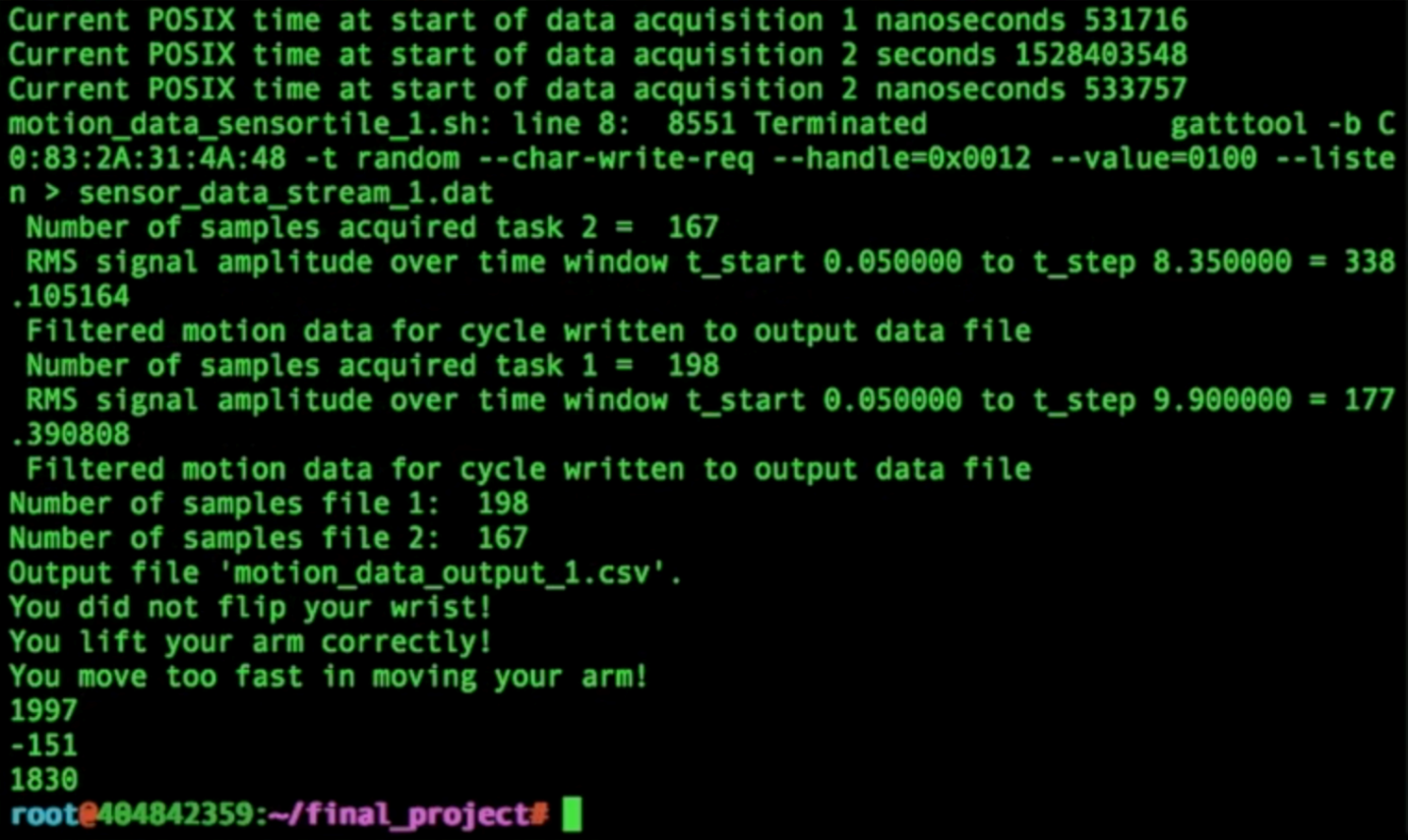
*Figure 15. System output of standard hook shot*

The highlighted area in red is the result of your performed hook shot. These three lines of outcomes correspond to the three features we selected. Each line of outcome can provide the user with instructions needed to know what to improve in the next shot.

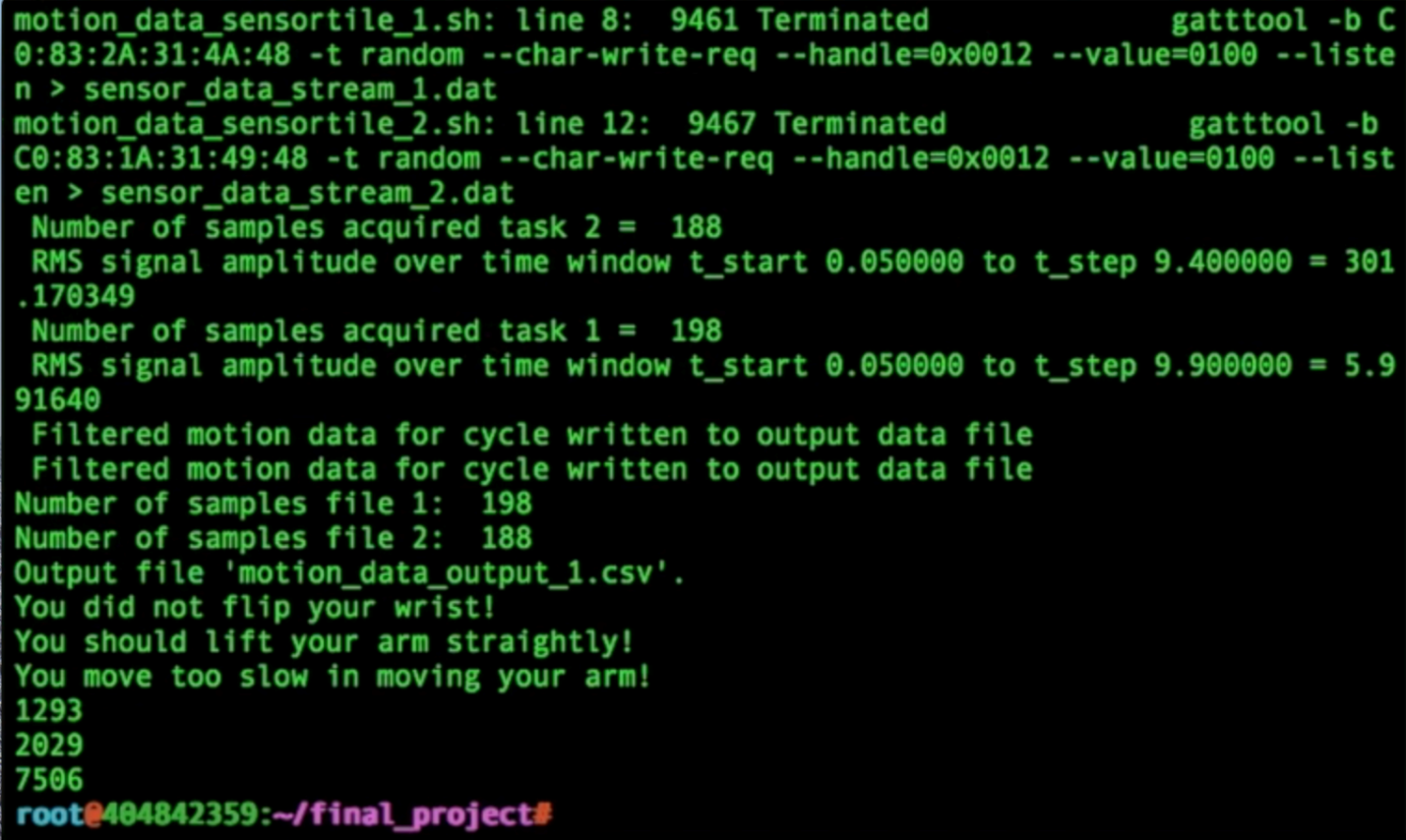
The highlighted area in blue is three lines of output we added for the convenience of you to see the actual raw data output of the three features so that you can know if the system is making the correct classification or not and make adjustment accordingly.

The feedback output lines can be flexible. You can change the output to whatever instructions you feel suitable for you own project.

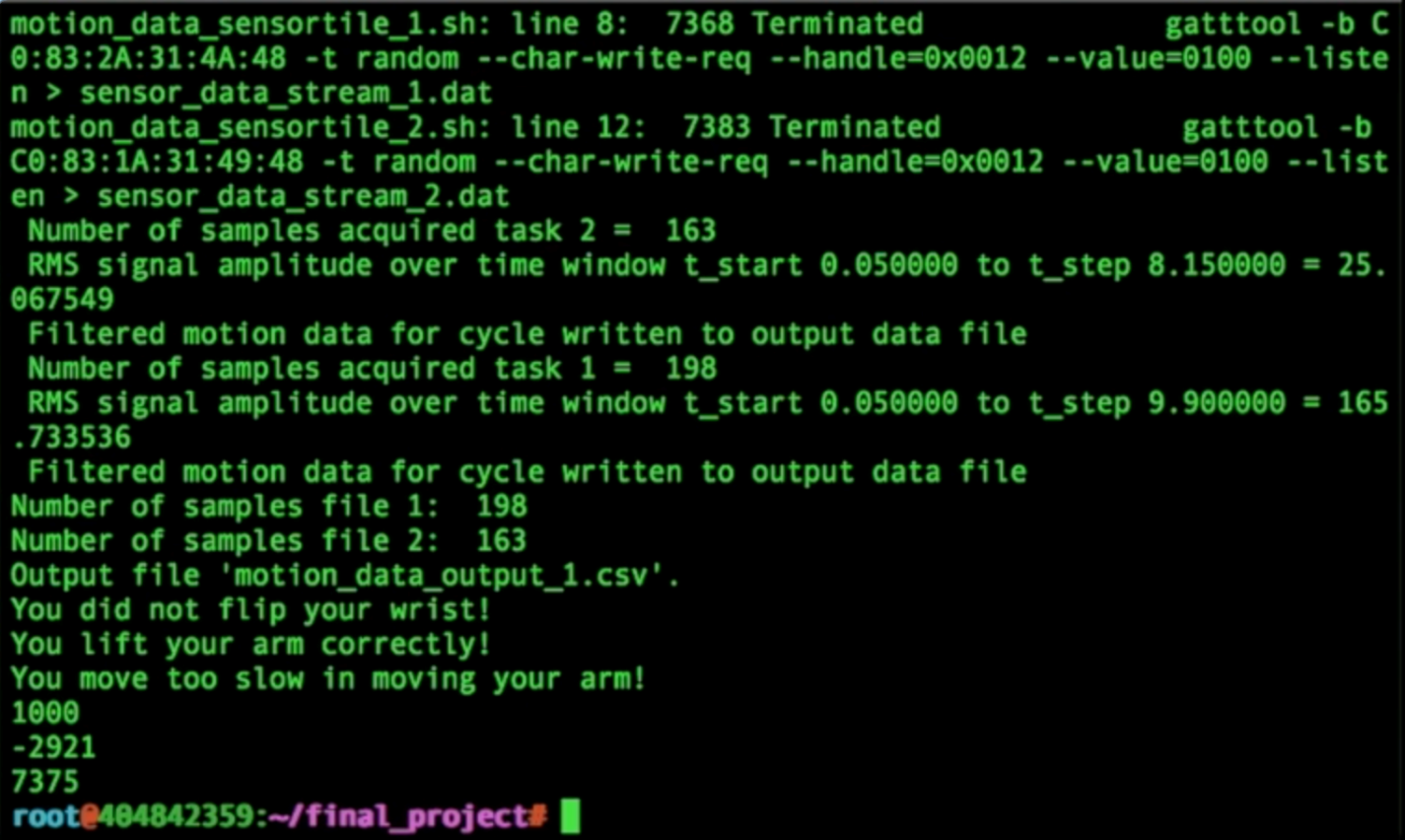
Following figures are results of three other wrong hook shot attempts for reference purpose:



*Figure 16. System output of too fast hook shot*



*Figure 17. System output of too slow hook shot*



*Figure 18. System output of wrong hook shot motion*

1. Confusion Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Standard | Too Fast | Too Slow | Circular (Wrong) |
| Outcome: Standard | 9 | 0 | 1 | 0 |
| Outcome: Too Fast | 0 | 10 | 0 | 0 |
| Outcome: Too Slow | 1 | 0 | 9 | 0 |
| Outcome: Circular (Wrong) | 0 | 0 | 0 | 10 |

*Figure 19. Confusion Matrix*

We can see from the confusion matrix above the correctness of our hook shot trainer is high and the system is proven to be robust and reliable. The only thing that needs to be tuned is the slight difference between slow and standard speed.

# Source Code

## Source Code Setup

1. Download the source code from the following link

<https://drive.google.com/file/d/1oklracycy4tu-tZLeKCt9_ypIJmPE6kd/view?usp=sharing>

1. Decompress the hook\_shot.zip file
2. Following files should be inside the folder

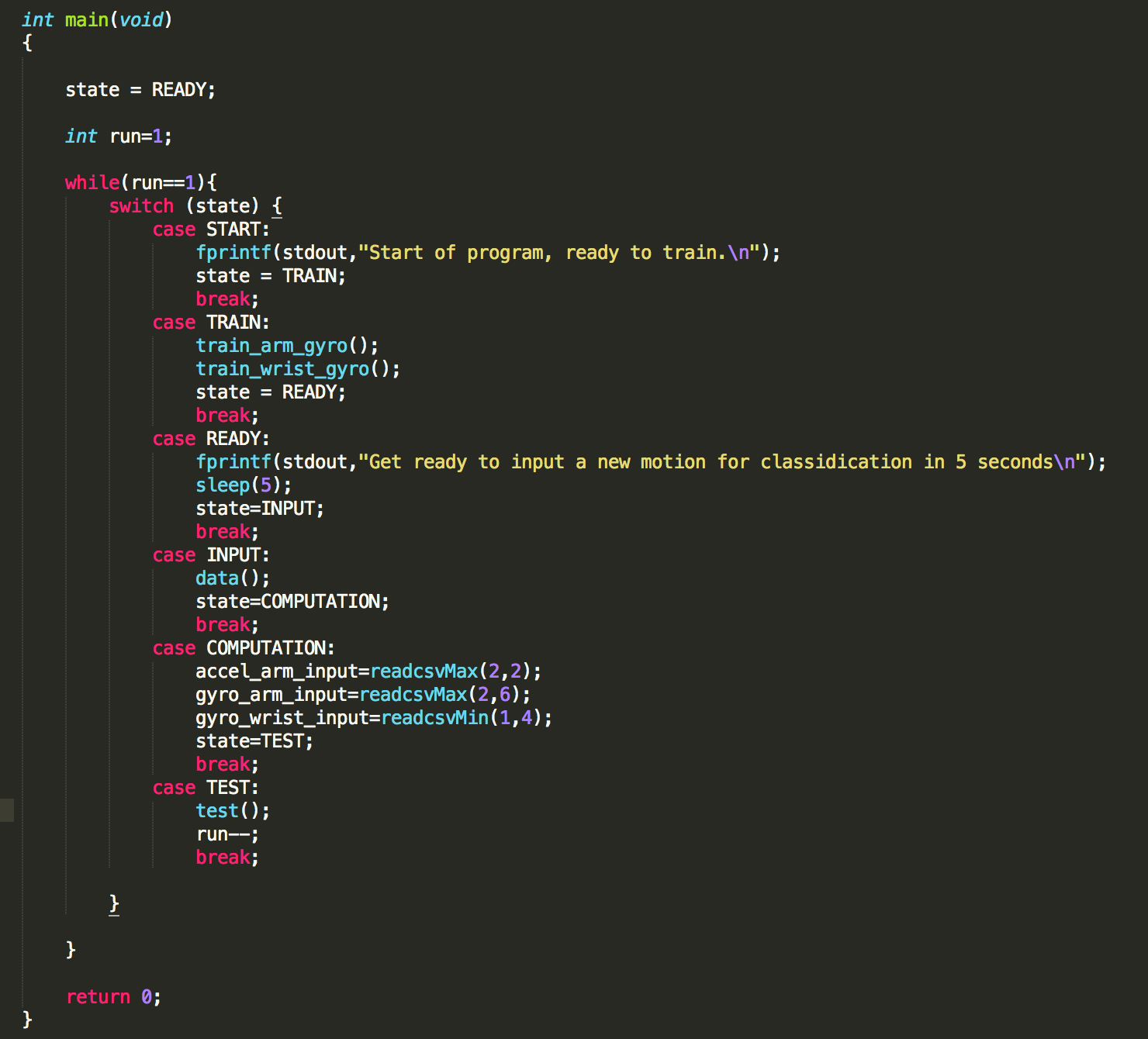


*Figure 20. Required code files*

4. The executables are already compiled in the folder

## Source Code Description

Here is the main function of the program:



*Figure 21. Main function*

It is a state machine, with states of START,TRAIN,READY, INPUT, COMPUTATION, and TEST. It will repeatedly to acquire new motion input, compute the data, and classify it with suggestions of motion.

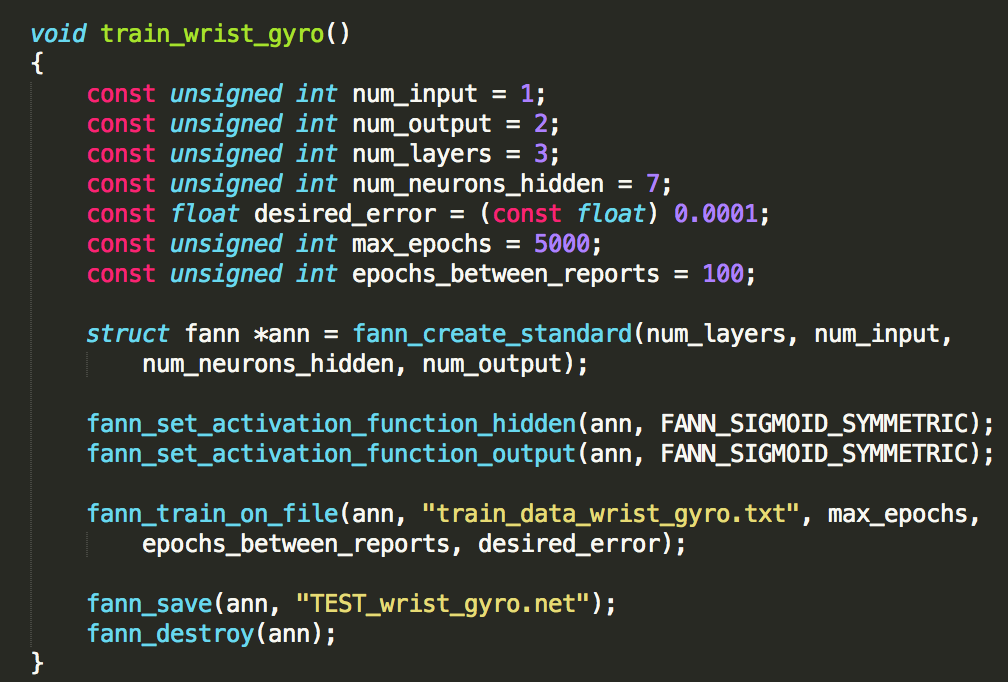
Here are the functions to help extracting max and min value from the two csv files.



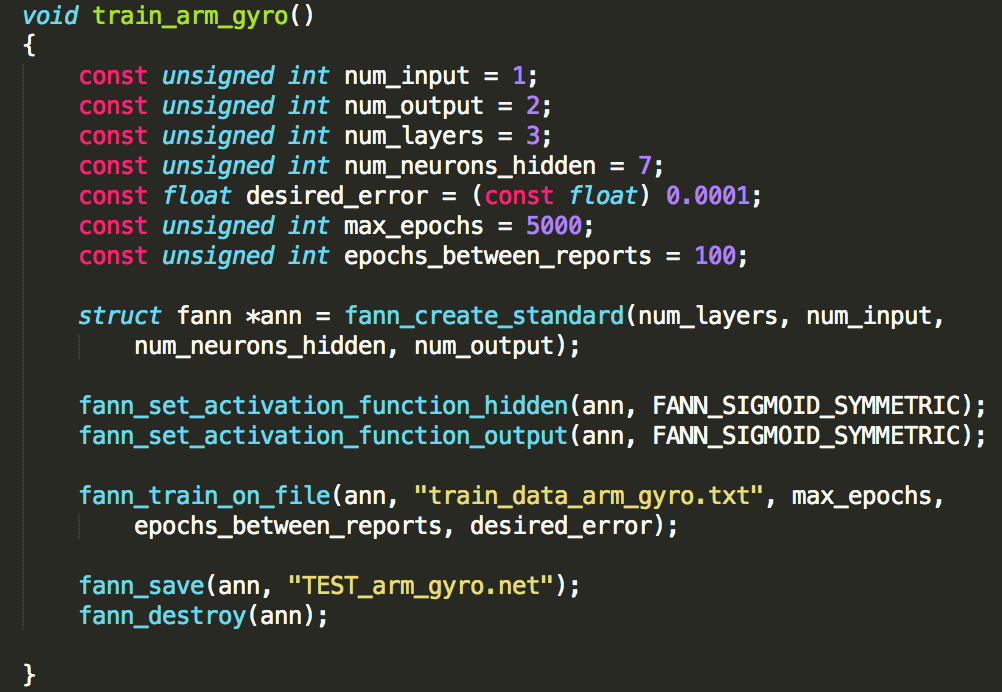
*Figure 22. readcsvMax and readcsvMin function*

It takes to argument of input, file is used to specify which csv to read, and position is used to specify which column we want to read.

Here are the two training functions that uses fann library to train two neural networks based on the train files.



*Figure 23. train wrist gyro function*

****

*Figure 24. train arm gyro function*

Both functions take 1 input and classifies to 2 types of output. The number of layers, number of neurons, desired error, max\_epochs can also be modified.

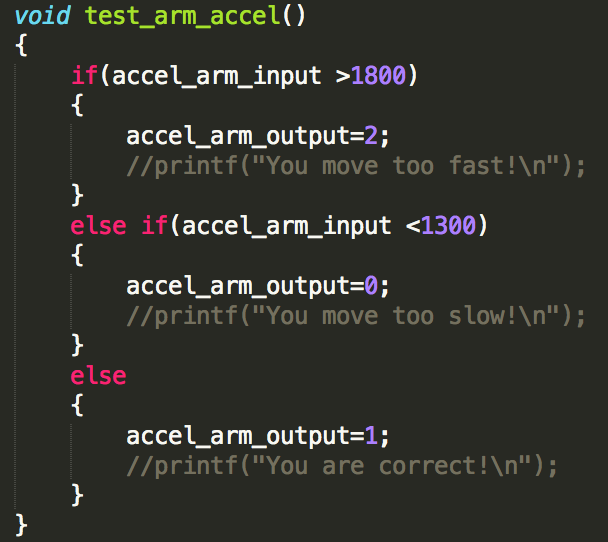
Here are the 3 test functions of the 3 features, arm\_gyro and wrist\_gyro is classified based on the the two functions of training mentioned previously. The arm\_accel is classified using the threshold value we determined based on observing the original collected data.



*Figure 25. test wrist gyro function*

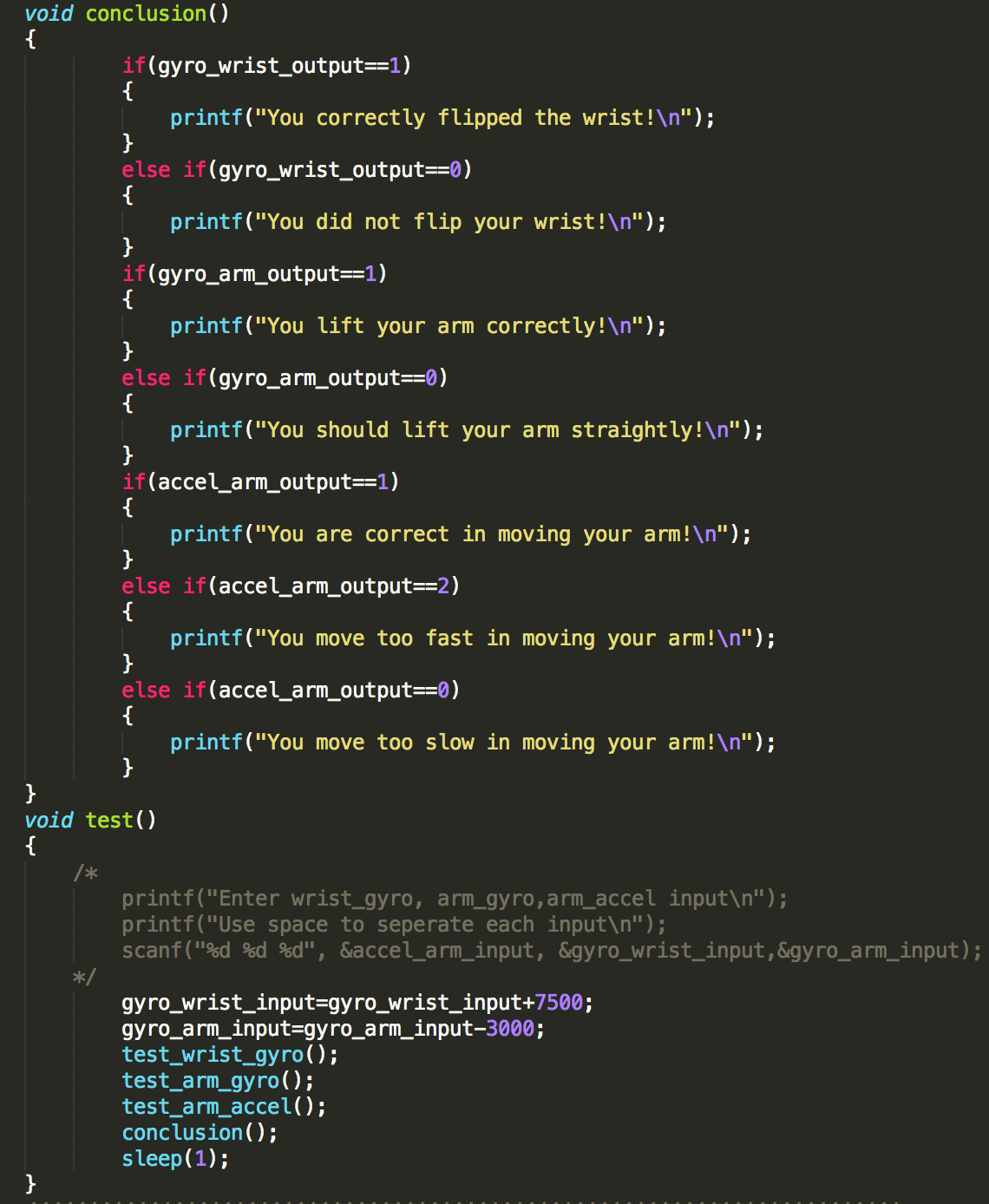


*Figure 26. test arm gyro function*



*Figure 27. test arm accelerometer function*

This is the two function we use to produce a list of suggestions for the user. The function **test** is calling the resting function mentioned previously to do classification and it will then call the **conclusion** function to print out the suggestions on the screen.



*Figure 28. Conclusion and test function*

## 

# Design Guidance

## Development Process

The initial goal for this design is to classify hook shot motions with 1 correct motion and 4 different incorrect motion. As we study through the data and analyze it with our understanding with the hook shot motion. We decided to make our system compatible with all kinds of combination of micro motions. We have three kinds of arm raising speed, fast, moderate, and slow. We can determine whether the user flipped the wrist or not. We can determine whether the user move their arm straightly. Combined all together, we have a system that can classify in a total of 12 classes. With the use of file manipulation and FANN library, our system become able to classify user motion and provide three suggestions corresponding to each aspect of a hook shot.

## Challenges

There are three challenges in the design.

1. How to decompose a standard hook shot and decompose it to various micro motions?

After carefully going through many hook shots tutorial, we determined that a success and correct hook shot mainly composed of two parts, the raise of arm and the flip of the wrist. For the arm, the speed and the trajectory are important, and these are two features. The wrist is another feature. Therefore, we decomposed the hook shot into 2 motions and 3 features.

1. How to identify and analyze the micro motions?

We carefully examined the raw data and compare it with the actual motion to identify features that can helps us to do classification.

1. How to give suggestions on these micro motions?

We write a function to give suggestions based on the neural network classification.

## Advice for New Users

There are also modifications can be made by new users to make this system have better performance.

To make the system have better performance, two new SensorTile on users’ feet will be included in the system. Because in a standard hook shot, the user need to jump. Due to the constraints that there are only two SensorTile available. We decide to focus on motion of arm and hand.

To make the system more personalized, users height, wright will be incorporated into the algorithm. Because different users have different height, weight, length of arm, to make it suitable for the user, we recommend including those parameters in the algorithm.

Last but not the least, the users can extend this system into other basketball motions or even other kinds of sports.

# Reference

* BeagleBone tutorials:

http://iot.ee.ucla.edu/UCLA-BeagleBone/index.php/Home

* SensorTile tutorials:  
  http://iot.ee.ucla.edu/UCLA-STMicro/index.php/Home
* FANN system code provided by EE180D
* Dual Sensor Acquisition Code by EE180D