STMicroelectronics SensorTile Reference Design:  
Climb On

Craig Young

Loic Maxwell

Table of Contents

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

[1.1.](#_1fob9te) Description 3

[1.2.](#_3znysh7) Motivation 3

[1.3.](#_2et92p0) system architecture 3

[1.4.](#_tyjcwt) sensing 3

[1.5.](#_3dy6vkm) system characteristics 3

[**2.**](#_1t3h5sf) **System Architecture 4**

[2.1.](#_4d34og8) code system and operation 4

[2.2.](#_2s8eyo1) data acquisition system 7

[**3.**](#_17dp8vu) **System Sensor Data Acquisition and Experimental Methods 4**

[3.1.](#_3rdcrjn) Training Data acquisition 7

[3.2.](#_26in1rg) testing Data acquisition 10

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

[4.1.](#_35nkun2) Sensor Data and Plots 10

[**5.**](#_1ksv4uv) **Signal Feature Extraction 13**

[5.1.](#_44sinio) Feature Extraction Techniques: Arrival Stability and Smoothness 13

[**6.**](#_2jxsxqh) **Neural Network Implementation 14**

[6.1.](#_z337ya) FANN neural network architecture 14

[**7.**](#_3j2qqm3) **Demonstration 14**

[7.1.](#_1y810tw) system demonstration information 14

[7.2.](#_4i7ojhp) Collected data 14

[7.3.](#_2xcytpi) Demonstration video 15

[**8.**](#_1ci93xb) **Source Code 15**

[8.1.](#_3whwml4) Other source code modules 15

[8.2.](#_3whwml4) Google Drive link 18

[**9.**](#_1pxezwc) **Design Guidance 18**

[9.1.](#_49x2ik5) design process and experience 18

[9.2.](#_2p2csry) challenges. 19

[9.3.](#_147n2zr) advice 19

# Introduction

## Description

Climb On is an exciting venture into movement analytics with a focus on rock climbing. It is a beginner rock climber’s tool to improve climbing technique, using wearable sensors and tracking motion while climbing. By following this reference design, one can cover topics such as machine learning, waveform feature extraction, and signal processing.

## Motivation

This project was created by two electrical engineers who rock climb as a hobby. They wanted to combine their two worlds and expand the rock climbing technology field. The vision was for this tool to be used by beginner rock climbers, but with the possibility of becoming sophisticated enough to be used as a professional tool.

## System Architecture Description

The system architecture of Climb On is composed of multiple sensor devices as well as a computing device. The sensors are ST Micro SensorTiles (STs) housed in the included plastic casing and installed within a wearable wristband. The computing device is a TI BeagleBone Green Wireless, further referenced as “BGB.” In the reference design, the BGB is connected to the host computer through a wired USB connection and accessed through a local SSH. The BGB collects data from the STs through a wireless Bluetooth connection (to be described later).

## Sensing

Sensing is done by the ST Micro SensorTiles (STs). The STs collect three dimensional accelerometer, gyroscope, and magnetometer data and write to local text files on the BGB during the Acquire\_LowPass\_Continous.c routine. These are stored onboard the BeagleBone and later used for training and testing of the system’s Neural Network.

## System Characteristics

The Acquire\_LowPass\_Continuous.c routine collects data from the SensorTiles at 20 Hz and applies a low pass filter to the data. The 3D accelerometer data is used to extract features from. These features are used to train the Neural Network and later to test it.

# System Architecture

## Code System and Operation

The following will describe the way to best use this system. The main purpose of this section is to guide an outside user through the main source modules necessary to run the system, and the ones to modify to match their own data acquisition devices. The FANN and gnuplot libraries should have been installed already. Below is a rough flow diagram of the way this system runs:

**acquire multiple data sets → extract features → train neural network → test neural network one set at a time**

The neural network is given established output data from the training group, and test the network by having it predict the output on a test data file. For each climbing move, the smoothness and arrival stability metrics are extracted from the data, which serve to help train the neural network classifier so that future sets described by each metric can be correctly categorized. The files listed below are the main ones necessary for the use of the system. The full description of all source files will be included in section 8.

1. Before attempting any operation of this system, the user must make sure the contents of the following google drive in section 8.2 are in the working directory. Also “data” folder must be created in that same directory (must be spelled that way), which will serve as the storage area for all collected data files:

> mkdir data

1. **Makefile** - Before doing anything, the user will type “make all” in the command line, which will compile all source modules and set up the system environment. If the executables ever need to be removed, the “make clean” command can be initiated. If any .c files are modified the “make all” command can be invoked to recompile whatever is necessary

> make all

1. **acquire\_data.c** - This is the main acquisition program. This program is called to acquire however many data sets that are desired, for processing later. The user will specify a number of iterations to run as an argument. Instructions within the program will guide the user through the acquisition process
2. **motion\_data\_sensortile\_1.sh -** Before taking data the address of the first sensortile (left hand) needs to be updated as it is described in the file. This ensures the BGB knows what to connect towhen attempting to collect data.
3. **motion\_data\_sensortile\_2.sh -** The same must be performed for the second sensortile(right hand)
4. **check\_file.sh**- This serves as a data organization tool, and is called by *acquire data.c*. When the user wants to start taking data, the bounds of this file organization loop can be changed, which determines how the trials will be stored in the data folder (currently 100-200). This is important, so that the different data acquisition groups can be separated. For the current setup, the data sets are stored in a numerically increasing manner from 100 to 200:

the first set will be under climbing\_data\_<hand #>\_100

the second will be under climbing\_data\_<hand #>\_101, etc.

1. **process\_data.c -** This may be called on any set of data, given a set number and a hand (“1” signifying left and “2” signifying right). This will create plots that can be observed by typing in:

<BBG IP address>:8080/graphics/waveform\_plot\_1.png and

<BBG IP address>:8080/graphics/waveform\_plot\_2.png

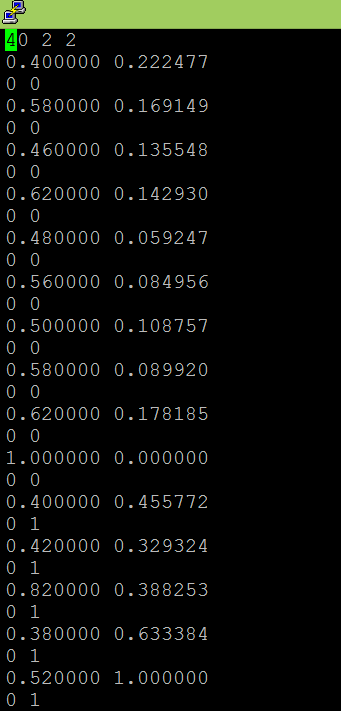
This will also write the arrival stability and smoothness values to the *shaking\_data.dat* and *timing\_data.dat* files, respectively. This becomes important later when creating the training file later. It will also write these same values to a test file called *climbing\_test.dat*, which is an input to the program that tests the neural network.

1. **training.c -** This program will collect all the data from a specified set of data, print them to a training file, and also print the normalization values to a separate file. When you the user has determined which sets they wish to use to train their data, they will copy them over to a concurrent set of data within the “data” directory (sets 800-814 for a group of 15 sets, for example). Then they run the training program on this set of data, with two arguments that determine the beginning/end of the set group. The third argument specifies the hand executing the motion from the move being analyzed. This actually calls a program called *mass\_process.c*, which repeatedly calls the *process\_data.c* program, and eventually writes all metric values to *climbing\_training\_file.dat*.

**\***IMPORTANT**\*** - The training file is not done yet however! Some compatibility issues still need to be worked through. Using a text editor, at the top of the file, the user must include the following line:

<# of sets> 2 2

The first value determines how much data there is (15 according to our previous example), and the next two specify the number of input and output values, both of which do not change. Underneath each set of metrics, the user prints the two output values that specify the quality of the smoothness/stability of the move (smoothness is printed first). A 0 represents a good result, while a 1 represents a bad result. For example, if a move was smooth but the arrival was not stable, a “0 1” should be printed underneath that set of data. The resulting file should look something like this:



*Figure 1: For training data, we took 40 sets to train our neural network as can be seen by the header at the beginning of the file. The first 10 values represent sets with good smoothness and good arrival stability, while the rest represent good smoothness and bad arrival stability.*

This is the training part of our system, where the network is fed examples for which their classifications as good/bad stability and good/bad smoothness are known so that it can learn and create decision boundaries. The FANN neural network requires the training file to be in a very specific format for it to function correctly, which is why these previous additions are made.

1. **climbing\_train -** Running this program will train the neural network based on the input training file from the *training.c* program. This takes no arguments, and will read in data from the “climbing\_training\_file.dat” that was previously modified.
2. **climb\_on -** This the main test program that was developed to determine how well the neural network is trained. It runs the *acquire\_data.c* program once, then the *process\_data.c* to extract both features and write them to a test file, and finally the *climbing\_test.c* program to test this set and output a rating based on the results.

**Example System Use:**

> make all → compile files

> ./acquire\_data 15 → collect 15 sets of data

> ./train 100 114 1 → train on samples 100 - 114 from the data/ directory for hand 1

#Correctly label all data in the training file as it was described before

> ./climbing\_train → train neural network

> ./climb\_on 1 → test your network for hand 1! see how it does for a single move

## Data Acquisition System

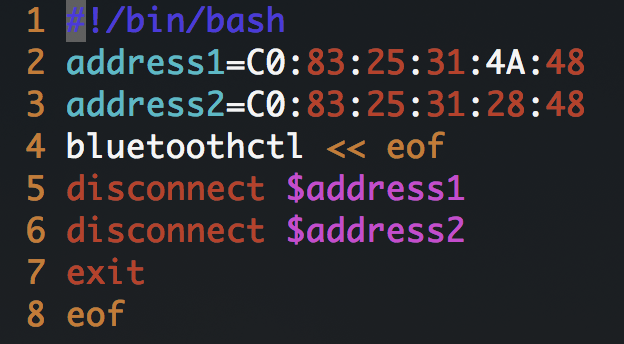
The general idea behind our acquire\_data program was that it called the Dual\_SensorTile\_Acquire.sh routine, which in turn calls the Acquire\_LowPass\_Continuous functions which set up a 10 second data acquisition period for each sensortile. These individually call their motion\_data\_sensortile.sh functions to initiate the bluetooth connection with the sensortile and pull their accelerometer data, which are then stored in the motion\_data\_output.csv files. The check\_file.sh script will copy these files over to the data directory, for processing later. The next section will go more in depth with regards to the technical aspects of each of these programs and the way we used the system for our training and test data sets.

# System Sensor Data Acquisition and Experimental Methods

## Training Data Acquisition

In order to train the neural network for classification, the system requires a training file with a specific format. Enough data must be collected in order to allow for a robustly trained neural net. The more data that is collected, the better the classifications will be.

To acquire data, we use the acquire\_data.c routine. When we examine this file, we see that the program has two possible states: OBSERVE1 and EXIT. The state starts in OBSERVE1, during which the program prompts the user to press enter when ready. After a pause, there is a system call to *disconnect.sh,* which is a shell script that ensures both STs are disconnected. In order to make sure this happens correctly, enter the MAC Address of SensorTile1 as address1 and that of SensorTile2 as address2 (see figure below).

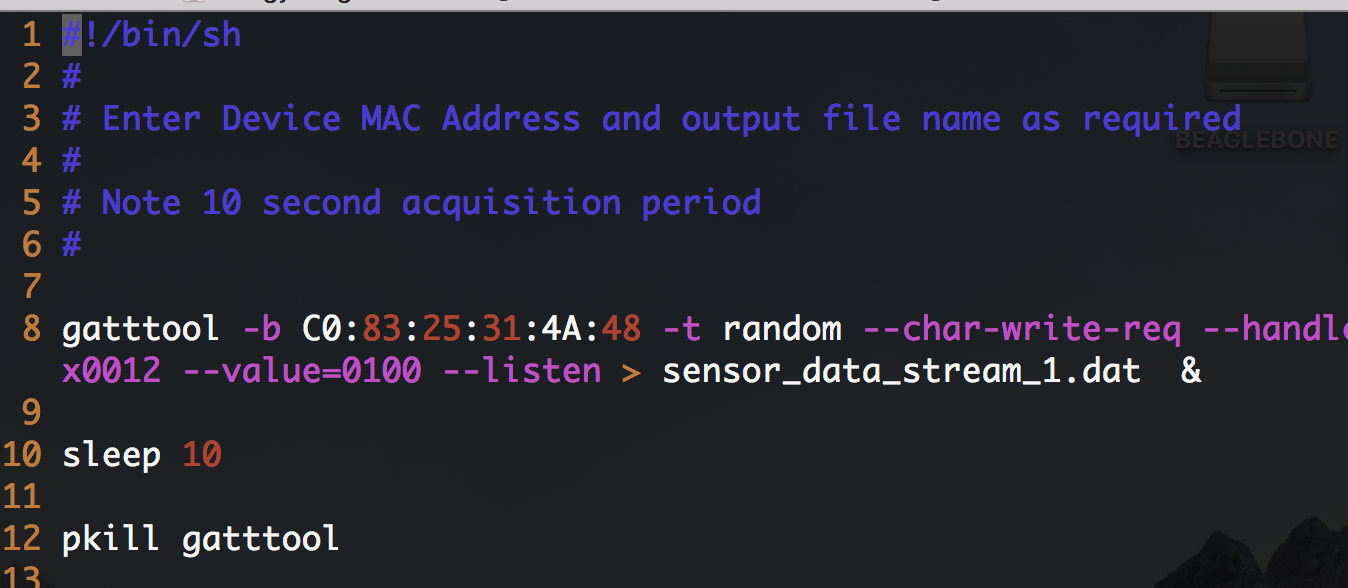


*Figure 2: Enter the MAC Address of ST1 and ST2 in disconnect.sh*

After ensuring the STs are disconnected from BlueTooth, the routine executes a system call to run *Dual\_SensorTile\_Acquire.sh* with the flag “-t 3”. This executes the shell script to acquire data from both STs simultaneously, by calling the *Acquire\_LowPass\_Continuous\_1* and ...*Continuous\_2* routines, with a trigger time of 3 seconds (data acquisition occurs 3 seconds after system call). the ‘&’ symbol after the execution causes the acquisition routines to run in parallel, and they are synced by the time delay.

The *Acquire\_LowPass\_Continuous\_1* and *\_2* routines parse the stream of Bluetooth data from the STs and write the data into a temporary data file, *motion\_data\_output\_1.csv*  and *motion\_data\_output\_2.csv.*

For this to work, the proper ST1 MAC Address must be entered into the gatttool command in *motion\_data\_sensortile\_1.sh*, and likewise for ST2 in *motion\_data\_sensortile\_2.sh*.



*Figure 3: replace the MAC Address in the gatttool command with that of ST1 and ST2 in each of their respective scripts, in motion\_data\_sensortile\_1.sh and \_2.sh.*

The program acquire\_data.c then renames these files with the format *climbing\_data\_X\_Y.csv*, where the X is either 1 or 2 for ST1 or ST2, and the Y is incrementing numbers to label the data sets, through the shell script *check\_files.sh*. This checks the highest data set and renames the motion\_data\_output file to fit the format *climbing\_data\_X\_(Y+1).csv*.

Once all the code is compiled with gcc and working, the following command can be used to collect X amount of data sets in a row:

*./acquire\_data X*

These will be stored in the following format as *climbing\_data\_X\_Y.csv* in the *data* directory, and can be processed later and used to create the training file.

Collecting data must be done in a specific manner. IMPORTANT: all data must be collected for the same move on the same climb in order for the system to work. The climber should situate themselves, standing in a position to make the climbing move in question. The operator should execute the acquire\_data executable and press enter when ready, at which point the operator should notify the climber to “climb on,” or place their feet on the footholds (off the ground). After 3 seconds, the move should be done by reaching with the hand containing ST1. See Video 1 for an idea of what the data collection process looks like.

Enough data should be collected to provide accurate classifications for each category. In the example, 40 sets of data were collected for training - fewer or greater sets could be collected depending on the degree of accuracy desired. When enough data has been collected, training can begin. *training.c* should be used to perform data processing on multiple files at once, using the format:

./training X Y H

where X is the number of the first data set to be tested, Y is the number of the last data set to be tested, and H is the hand number. This is done with a system call to:

./process\_data i H where i iterates from X to Y, incrementing by 1. H is still the hand #

process\_data.c executes a system call to *find\_waveform\_peaks.sh* on the data for ST1. Then, the system call for *rename\_files.sh* allows the same *find\_waveform\_peaks.sh* shell script to be used for the ST2 data. These create GNUPlot files, and store png files of the plots of the 3D accelerometer data. These plots can be viewed by opening a browser in the computer of the local ssh and entering in the url “beaglebone.local:8080/graphics”.

*process\_data.c* performs feature extraction using functions provided in *process\_timing.h* and process\_shaking.h, to be described later.

Each of the .h files appends extracted feature data into *timing\_data.dat shaking\_data.dat*. These are later used to create the training file.

Then, the script *training.c* can be used to create the training file by reading the *timing\_data* and *shaking\_data*. Finally, the script *climbing\_train.c* will read the *climbing\_training\_file.dat* and train the neural network.

## Test Data Acquisition

To test the neural network classification, compile and run climb\_on.c. The program will tell the operator to press enter when the climber is ready. The climber should then perform the climbing move, and the program will output the classification. See Video 2 at 2:40.

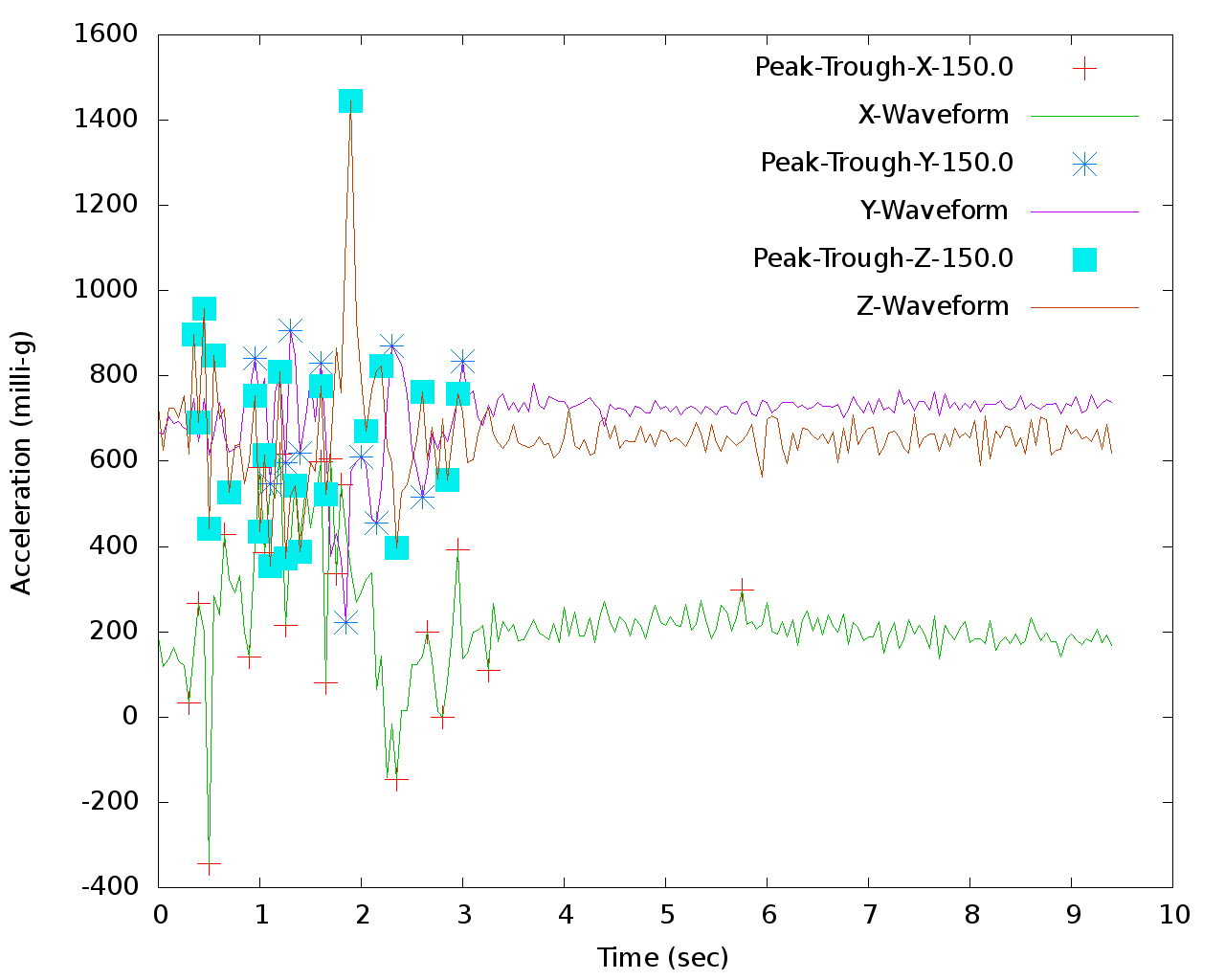
# System Sensor Data

## Sensor Data and Plots

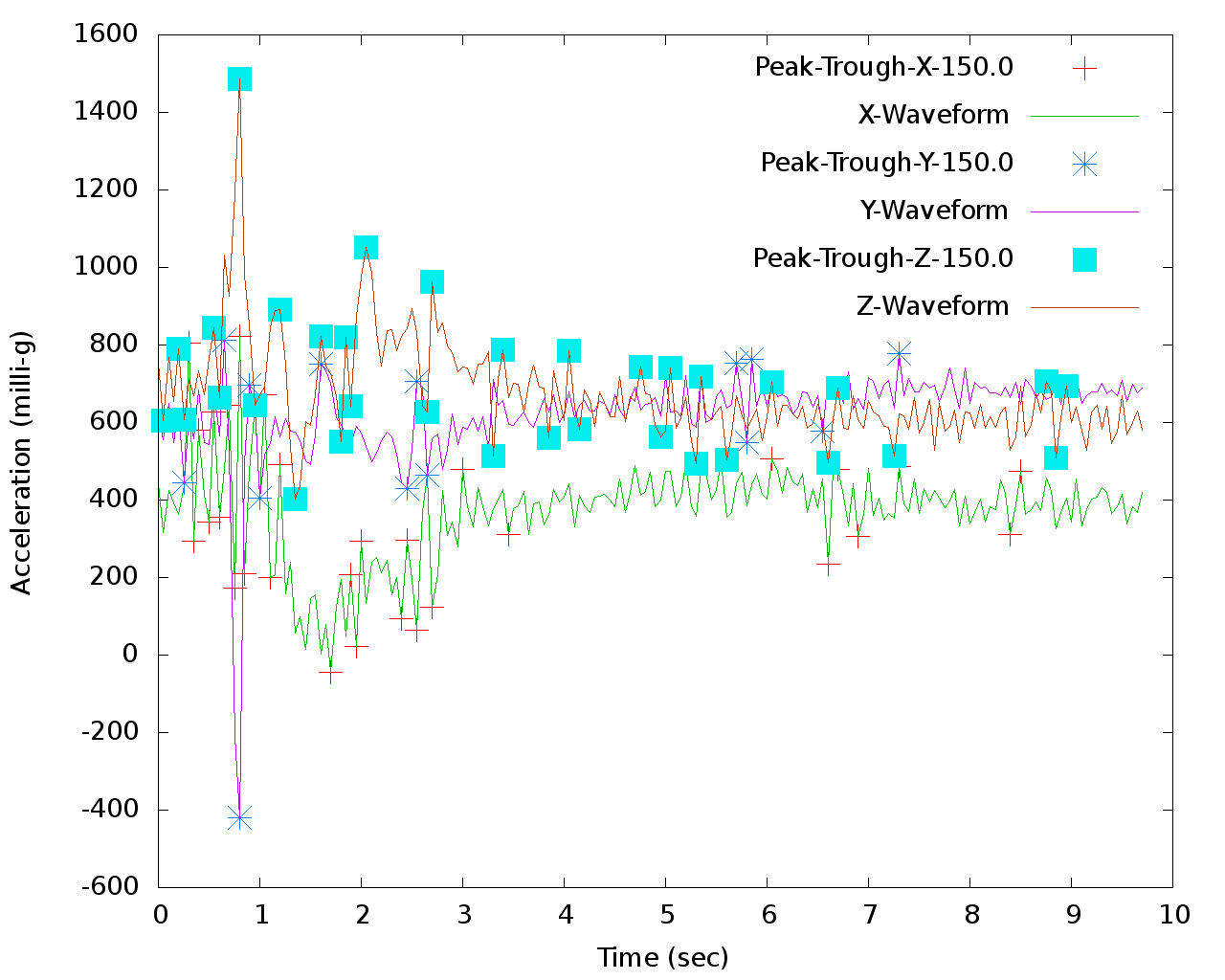
The Climb On system uses two ST Micro SensorTiles for all data collection while executing a climbing move. After the connection is established between the BGB and both devices, the following command is invoked on each by the “motion\_data\_sensortile” scripts:

> gatttool -b <ST MAC ADDRESS> -t random --char-write-req --handle=0x0012 --value=0100 --listen

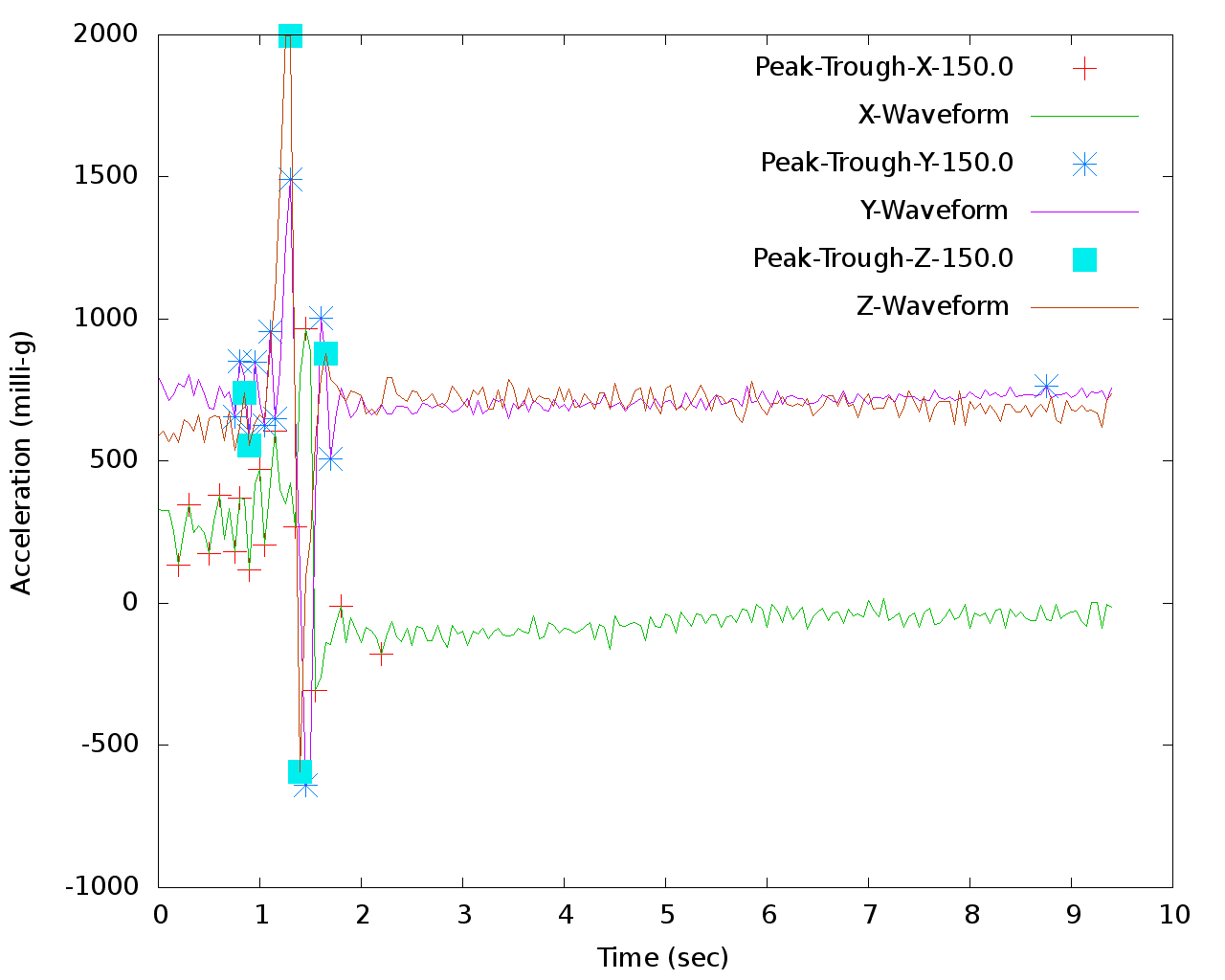
This triggers a “listen” on the ST accelerometer, magnetometer, and gyroscope data, which gets transferred and stored in a CSV file. The *process\_data.c* program uses a separate peak-finding script to plot these CSV files so that they could be observed for feature extraction purposes. Because the hand angles varied so much, even for the same move, only the accelerometer data was used. Shown below are plots of the data for all four different classifications, resulting from each combination of the good/bad outputs from both features.



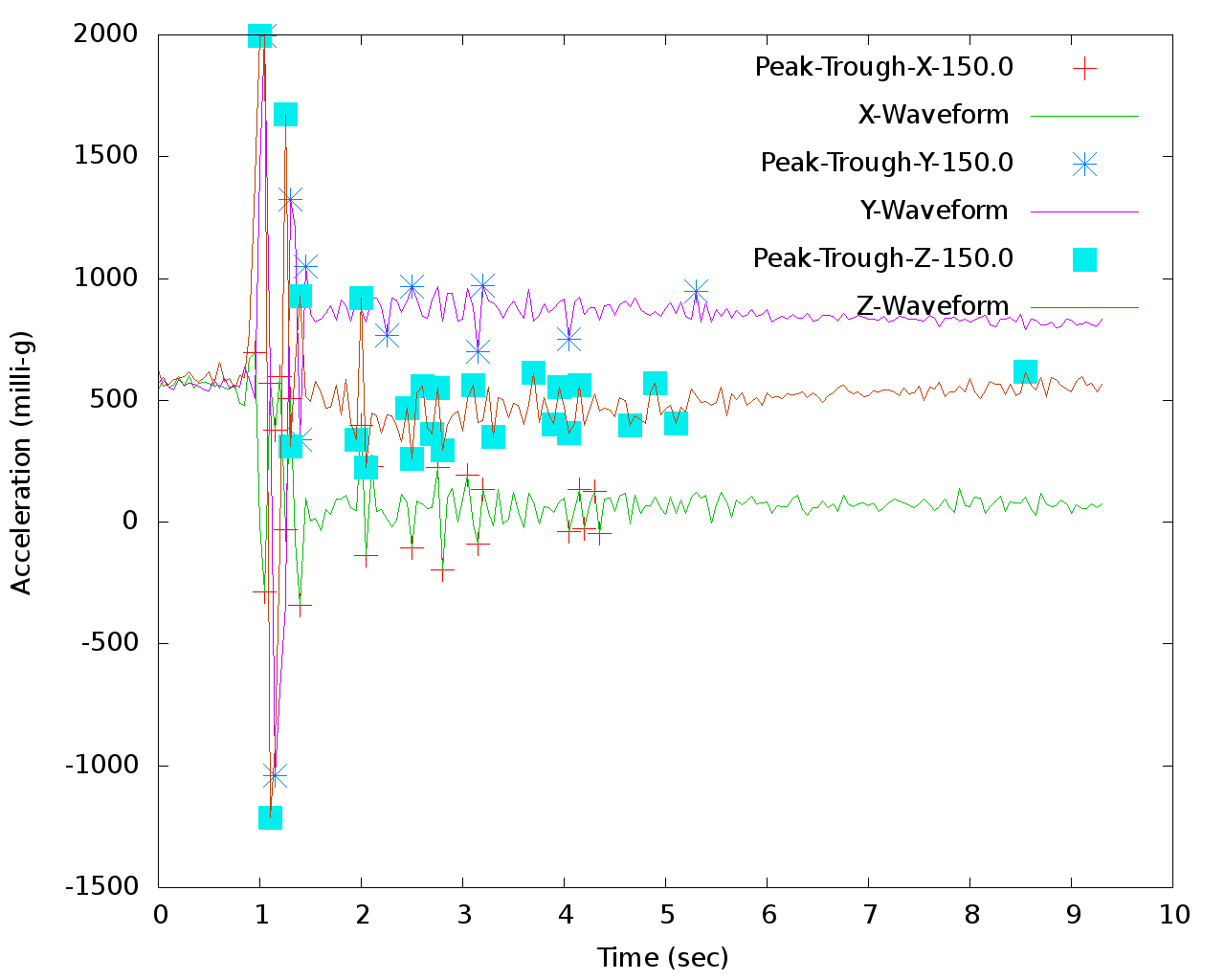
*Figure 4: A smooth move with good arrival stability. The move was not executed too quickly, and the noise following the move was very minimal.*



*Figure 5: A move with good smoothness but bad arrival stability. Though the timing of the move was good, there was a lot of residual noise following the end of the move due to shaking*



*Figure 6: A move with bad smoothness but good arrival stability. This move was executed too quickly, but the climber had a stable grip upon arrival.*



*Figure 7: This represents the worst of both metrics, where the move was not smooth and the arrival was not stable.*

# Signal Feature Extraction

## Feature Extraction Techniques

**Arrival Stability -** The measurement for this metric was extracted from the shakiness that occurred following the end of the move. We used our technique for determining the start and end of the move to figure out a time stamp at which to begin the extraction of the stability metric. From there, we determined the average acceleration values from the X, Y, and Z waveforms over a period of 3 seconds following the end of the move. Following this, we determined the deviation of each data point from their respective average acceleration values and summed them out to achieve a value representing the post move stability. The data for a particular axis will oscillate about a mean value following the move, so a larger general deviation from this value over those next few seconds will indicate instability on the arrival. When processing multiple files at once to create the training file with the “training” program, the maximum and minimum stability values will be used to normalize the data. This helps prevent this metric from overshadowing the smoothness metric values during the neural network training.

**Smoothness** - This feature was extracted by determining the total “move time”, defined as the time from when the hand leaves the start hold to when the hand grabs the end hold. This was accomplished by examining the 3D acceleration waveforms. The data points for the times, as well as X, Y, and Z accelerations were extracted from the climbing\_data file in process\_timing.h and stored in arrays. The X, Y, and Z accelerations were then offset so that they would start a set threshold value, in milli-g’s, apart from each other.

This threshold value was set to be 200 milli-g’s a the front to determine the start time. For example, the Y set of data would be shifted down by the first data point so that the stream started from 0. Then, the X set of data would be shifted so that the stream started from 200, and the same with Z to start at 400. The data would then be looped through to check if any of the X, Y, and Z waveforms crossed. This crossing time was when the climber’s hand accelerated enough to start the move.

A similar method was used to find the end time of the move. Due to varied amounts of shakiness and noise at the end of the waveform, the threshold was determined by integrating the absolute value of acceleration for X, Y, and Z for the last 2 seconds of the climb to determine how much noise there was at the end of the data, with three options for threshold values. For example, if the waveform was very shaky at the end of the file, the threshold value would be set to 450 from the back end. If the waveform was a medium amount, the threshold was 400, and if there was a small amount of shaking, the threshold was 300.

Finally, the start time was subtracted from the end time to come up with the total move time. In training, the move time is normalized with the shortest move time 0 and the longest move time 1. This move time determined the smoothness metric and was used to train the neural network was one of the inputs.

# Neural Network Implementation

## Fann Architecture

In order to determine the best possible sensor layout for our neural network, we had to do some experimentation with the number of neurons and hidden layers in order to minimize our training and test error for the data we had. We determined that with 3 neurons and 3 hidden layers these values were minimized. An important thing to note however is the fact that this may not be the best for a different group of data that exhibits similar behavior across all sets. If our work were to be reproduced then this neural network tradeoff analysis would also need to be redone, though our current setup would be a good starting point.

# Demonstration

## System Demonstration Information

This section contains sample data for demonstration of the system’s functionality. The demonstration videos show examples of the data collection process, as well as the testing process.

Section 3 details how to train and test the system. The following section contains sample data from a test.

## Collected Data

This section contains data collected during initial testing of the system. The data is displayed in the following confusion matrix, showing the resulting classifications.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Predicted** | | | | |
| **Actual** |  | Smooth/  Stable | Smooth/Unstable | Not Smooth/ Stable | Not Smooth/ Unstable |
| Smooth/Stable | 5 | 1 | 0 | 0 |
| Smooth/Unstable | 0 | 4 | 0 | 0 |
| Not Smooth/Stable | 0 | 0 | 5 | 0 |
| Not Smooth/Unstable | 0 | 0 | 0 | 5 |

*Table 1: Confusion matrix showing the results from the testing of the program*

Some sample data for testing can be found in the Google Drive “Testing Data” section. They fit under the following categories:

Smooth/Stable: climbing\_data\_1\_1.csv

Smooth/Unstable: climbing\_data\_1\_2.csv

Not Smooth/Stable: climbing\_data\_1\_3.csv

Not Smooth/Unstable: climbing\_data\_1\_4.csv

## Demonstration Videos

Video 1 shows samples of collecting the data. Video 2 at 2:40 shows samples of testing the data.

# Source Code

## Other Source Code Modules

Some of the system source files were previously discussed. Their locations within this reference design are listed following:

**check\_file.sh** - Section 2

**training.c -** Section 2

**climb\_on.c -** Section 2

**acquire\_data.c -** Section 3

**process\_data.c -** Section 3

**Acquire\_LowPass\_Continuous 1 and 2 -** Section 3

**Dual\_SensorTile\_Acquire.sh -** Section 3

**find\_waveform\_peaks.sh -** Section 3

**mass\_process.c -** Section 3

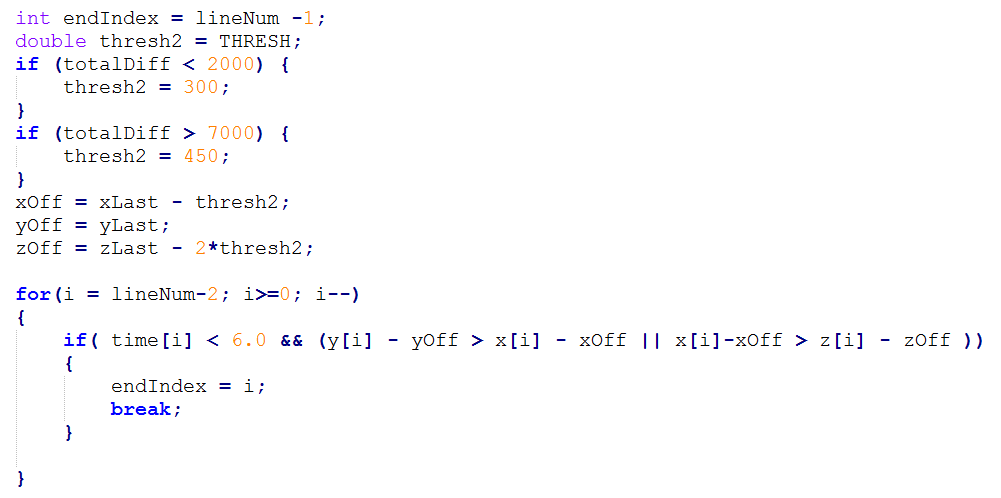
**rename\_files.sh -** Section 3

**disconnect.sh -** Section 3

**waveform\_sensortile\_data 1 and 2 -** Section 4

The rest of the source code files are listed below in more detail:

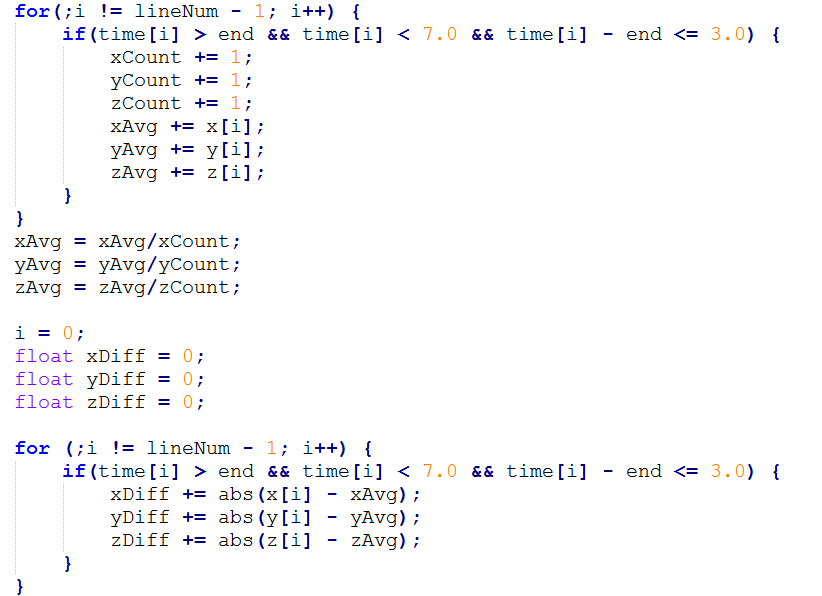
**process\_timing.h -**

****

*Figure 8: This represents the feature extraction for the endpoint of the period of time that represents the move. Previously, a stability calculation was carried out on the end of the move to determine what threshold to use when normalizing the axis data and seeing where they intersect. The higher the shaking at the end, the larger the threshold needs to be to correctly capture the move*.

The start time is subtracted from the end time to get the move time, which is written to a staging file *timing\_data.dat* for the *training.c* file to use and the test file *climbing\_test.data* so that the *climb\_on* end user program can use the data.

**process\_shaking.h -**



*Figure 9: Code snippet that shows the shakiness metric extraction. The averages are computed, followed by the difference from the average for those same points used to compute the average. The data tends to oscillate about a single value (for a single axis), so this computes a number representing the deviation from this average across 3 seconds.*

This function then appends this value to the *shaking\_data.dat* and the *climbing\_test.dat* files, which may be used by the *training.c* or *climb\_on.c* programs later if desired.

**climbing\_train.c -** This program contains the specifics regarding the layout of the neural network, further explained in Section 6. It is also called when the neural network needs to be trained. By default it reads in training data from *climbing\_training\_file.dat*, which is created by the *training.c* program.

**climbing\_test.c -** This program contains the code that actually tests the neural network that was previously trained using *climbing\_train.c*. By default it reads in the *climbing\_test.dat* file created by the *process\_data.c program* and interprets the data just like the training program did with the *climbing\_training\_file.dat* file. Currently, the *climb\_on.c* program calls this program after acquiring a single set of the data and processing it. However, if an outside user wishes to test multiple sets of the data at once, they may actually follow the same procedure as the creation of the training file, found under the *training.c* instructions from Section 2. The training program can be called, and the resulting training file can be renamed to *climbing\_test.dat.* Then the *climbing\_test.c* program can be called directly to classify the desired data!

**waveform\_gnuplot\_script -** This is called by *find\_waveform\_peaks.sh* to use the gnuplot library and created plots from the input waveform data.

**waveform\_peak\_find.c -** This is called by *find\_waveform\_peaks.sh* to actually parse the motion data file and determine the number of peaks in each given a threshold. The peaks are represented by the colored dots on the plots. This data analysis tool was only used to observe the waveform plots, so the peaks do not actually mean anything.

**Makefile -** The file includes the targets representing the programs to be compiled, and their dependencies. Since multiple C program files depend on each other, the Makefile is a useful tool to make sure all executables are up to date.

## Google Drive Link

Below is the link for the google drive that contains all necessary modules for the operation of this system. It also contains the test sets and the videos.

<https://drive.google.com/open?id=10bFj6wcHtQM6gM8PPOxQ8bqT40zdQl9A>

# Design Guidance

## Design Process and Experience

This sections details the design process and experience for the original developers of the project.

The original goal for Climb On was to be able to classify the quality of moves for an entire climb of several moves. The idea was to separate the waveform data into each move, and classify them separately.

The development began by preparing the simultaneous data acquisition program for both STs. At the same time, research was done in order to determine what constituted quality of a climbing move, and what metrics could be used. The metrics needed to be independent and crucial to determine the quality of the climb. These metrics were chosen to be “smoothness” - reaching without jerky motions, and “arrival stability” - not shaking or moving the hand upon grabbing a new hold. These metrics would be rated as either “good” or “poor” for a move, resulting in 4 possible classifications.

Data was collected and examined in order to determine features that could be extracted in order to differentiate the classifications. The features were chosen to be the time of the move, and the integrated absolute value of acceleration in the 2 seconds following arrival at the new hold.

Feature extraction took the bulk of the time spent on this project. The smoothness metric was derived from the time of the move, and the arrival stability metric was derived from the integrated absolute value of the acceleration following arrival.

Multiple different possible methods were attempted for feature extraction. The chosen method was as described in Section 5.

Once feature extraction was finalized, many sets of data were collected and used for training. The neural network was trained, and finally tested. The results from the testing are in Section 7.

## Challenges

The most difficult challenge was successfully developing the feature extraction methods that could be used to accurately train and be classified by the neural network. In particular, determining the time of the move was the most difficult feature extraction problem to solve. This is because every move, regardless of classification, was very similar. This meant that the feature extraction had to be able to successfully determine between many closely related movements. Certain movements were very difficult, such as the unstable/shaky ones. These often made the time classification difficult.

The other main challenge was to determine the metrics that could be used to classify the climbing quality. They needed to be independent, and able to determine whether the move was good or poor quality. Research, as well as experimentation at the rock wall, was used to come up with these metrics.

## Advice

Climb On is a very exciting venture into the field of movement analytics within climbing. Similar technology in other sports can give athletes a competitive edge, and is worth millions of dollars. The concept behind Climb On (a beginner’s tool) could be made sophisticated enough to be used by professional rock climbers.

Should readers wish to continue developing the technology used in Climb On, they should carefully work on feature extraction and be able to generalize feature extraction methods to work on many different type of moves. They should constantly test and collect new data from different sources in order to ensure that the methods are robust.

Steps to improve Climb On would be to develop more metrics, and more ratings within each metric. Adding a third metric, as well as a rating of “so-so” between “good” and “poor” would provide climbers with more fidelity - 9 different classifications.

Additionally, given the current state of Climb On, it would not be so difficult to enable classification of several moves within a complete climb. The time of data collection would need to be increased to capture the entire climb. Then, the modular technique for classifying a single move could be used on the data after separating the waveforms into individual moves.