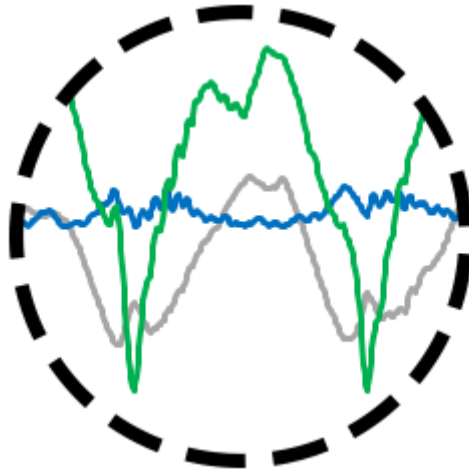


ANDROID MOTION SENSOR CAPABILITY



FOR DETECTING, TRACKING AND ANALYSING RESISTANCE-TRAINING EXERCISES

By Gerry Brosnan

Bachelor of Science (Hons) in Computing

Institute of Technology Tralee

2015

Primary Supervisor: Paul Collins

Secondary Supervisor: Billy Stack

Abstract

Technology plays a big role in modern sports and fitness. Professional athletes and teams use technology to improve their performance. Mobile applications have also become very popular among non-professionals and enthusiasts. A wide range of applications provide functionality to track and monitor cardiovascular training sessions such as running and walking. However, there are few applications designed specifically for resistance-training where repetitions and sets need to be counted and analysed to help the user improve their performance. The few that do exist, depend on additional and expensive wearable devices.

The aim of this project was to test the capability of the motion sensors in an Android device, for detecting motion during the performance of an exercise. The hardware and software fusion for the micro sensors were researched and tested. A server was set up to collect the sensor data for analysis, interpretation and feedback for the user.

Three exercises were tested which were the bicep curl, the lateral raise and the triceps kick-back. Data produced from the sensors were smooth and noise free and each exercise type produced, had distinct patterns of data. For the pre-processing of this raw data, peak detection techniques were used to extract reps from the data, and then each rep was resized and normalized so its data could be used efficiently with a neural network.

A neural network was used for detecting the exercise patterns. A supervised network was setup using an external Java library. A backpropagation rule was implemented using a three layer architecture; an input layer, a hidden layer and an output layer. The network was highly successful in detecting exercises using the pre-processed data. The network converged and was successfully trained using just ten reps from each exercise as training data.

For user feedback, a velocity-based training method was implemented where the last rep of a set was timed and then compared to further exercises carried out. With this method, the application developed the ability to recommend adjustments to the amount of weight lifted for future exercise sets.

The successful results shows that an application can be used in an Android device for detecting, analysing and providing feedback for resistance-training exercises, without the need for additional wearable sensors.

Table of Contents

Abstract.....	1
Chapter 1: Introduction	5
1.1 Overview	5
1.2 Problem Statement.....	6
1.3 Aims and Objectives.....	6
Chapter 2: Research.....	7
2.1 Resistance Training	7
2.1.1 Principles.....	7
2.1.2 Principle of Progressive Overload	7
2.1.3 1RM.....	8
2.1.4 Velocity-based training	9
2.1.5 Velocity Measuring Devices	10
2.1.6 Workout Planning	10
2.1.7 Strength Training Conclusion	10
2.2 Android Sensors	12
2.2.1 Galaxy S3 Sensor List.....	12
2.2.2 Accelerometer.....	12
2.2.3 Gyroscope	13
2.2.4 Barometer	13
2.2.5 Sensor fusion and filtering	14
2.2.6 Galaxy S3 Sensor Hardware	15
2.2.7 ST iNemoEngine_PAAP	17
2.2.8 Android Sensor API	18
2.2.9 The Physics of Movement.....	20
2.3 Artificial Neural Networks.....	21
2.3.1 Overview	21
2.3.2 History.....	21
2.3.3 Neural Networks vs Von Neumann.....	22
2.3.4 Perceptron	23
2.3.5 Paradigms and Learning Rules	25
2.3.6 Implementations.....	26
2.3.7 Data Pre-Processing	27
2.3.8 Peak Detection.....	27
Chapter 3: Methodology.....	29

3.1 Exercises.....	29
3.2 Android	29
3.3 The Server	30
3.4 Development Tools	30
3.5 Persisting Data	30
3.5 Data Pre-processing	31
3.6 Neural Network Implementation.....	31
Chapter 4: Design.....	33
4.1 Vision Document.....	33
4.1.1 Scope.....	33
4.1.2 Risk Analysis - Overview.....	34
4.1.3. Risk Analysis - Full List.....	35
4.1.2 High Level Feature List	36
4.1.3 Prioritization of Features	37
4.2 Functional Specification	38
4.2.1 Use-Case Diagram	38
4.2.2 User Stories	39
4.2.3 Prototype	46
4.3 System Design	47
4.3.1 Architecture	47
4.3.2 Sprints	48
Chapter 5: Implementation	50
5.1 Sprint 1: Start-up/Prototype	50
5.2 Sprint 2: Pre-Processing of Data	56
5.3 Sprint 3: Artificial Neural Network.....	71
5.4 Sprint 4: Android.....	74
5.5 Sprint 5: Analysis (Front-End)	76
5.6 Sprint 6: Creating CSV files.....	79
5.7 Sprint 7: Front-End and Testing Neural Network.....	81
5.8 Sprint 8: Decision Making / Feedback.....	87
Chapter 6: Findings, Analysis and Conclusion.....	91
6.1 Introduction	91
6.2 Sensor Capabilities	91
6.3 Pre-Processing of Data	91
6.4 Artificial Neural Network	92
6.5 Decision Making.....	92

6.6 Wearing the Samsung Galaxy S3.....	93
6.7 Further Recommendations	93
6.8 Conclusion.....	94
References	95

Chapter 1: Introduction

1.1 Overview

In a culture where people have become more aware of their physical well-being, we see all around us how technology can aid in physical fitness. Fitness apps are used to track fitness data such as speed, foot-steps, time, calorie counts and heart rate. It's common to see people with smart phones strapped onto their arms while doing cardio training, in both outdoors and indoor fitness sessions. Not only can smart phones track fitness data, they can also analyse this data to measure progression and performance which can be used to provide intelligent feedback to the user.

One area of fitness where smart phones are not so common is in resistance training. A quick browse of the Google Play store may show many resistance training apps that can log data for reps and sets. However, none of these apps can dynamically track and analyse live data similar to the apps design for cardio workouts. During a resistance training workout, sets and reps need to be counted while other factors such as speed and breaks can be influential in a person's workout session. Currently no phone applications provide features for tracking such data with the exception of a few that require other expensive wearable devices.

This study will assess the important factors of tracking a resistance training workout. The sensor hardware capabilities of an Android smartphone will be determined for the functionality that is required in reading resistance training exercises. Live data will be analysed and interpreted for feedback to the user to help them track their progress. This data analysis and interpretation will be the main area of focus for this project and a neural network will be used for data pattern recognition.

This chapter gives an overview and aims for this project including the main project research question.

The Research chapter looks into three main areas. Research is provided for gym training techniques and practises, while the ability and application of Android-based sensors are investigated in the second area. The final area looks into the concept for neural networks and how one can be implementation into this project.

The Methodology chapter provides overview of what techniques and technologies will be used for the implementation, while the Design chapter specifies the requirements which includes prioritization, diagrams and user-stories.

The Implementation chapter details all the work carried out in each sprint while screenshots of code and front-end functionality are also provided for a detailed understanding.

Finally, the Conclusion, Findings and Analysis chapter summarizes the work done in the project and gives an overview of the analysis and results taken from the tests carried out. Recommendations are also included for future development of the project.

1.2 Problem Statement

The research question is as follows:

Can an Android smart phone be independently used, to detect, track and analyse a user's strength-training activities, and also provide live and accurate feedback?

1.3 Aims and Objectives

Aim: To create an application for an Android smart phone to track and analyse a resistance training work out and provide instant feedback for the user.

Objectives:

- Use the phone sensors to track movements and time
- Send data to a server for analysis
- Receive results and suggestions back from the server application

Sending sufficiently accurate data to the server application and interpreting this data for user feedback could have many benefits for the user. The important factors of gym plans and their progression must be researched for the writing up of a user requirements list. The capabilities and accuracy of smart phone sensors will have an influence on how exercise data can be analysed for the user. Extensive research into how this data can be analysed and interpreted will be carried out. This research will aim to come up with solutions to provide reliable feedback for the user.

Chapter 2: Research

2.1 Resistance Training

2.1.1 Principles

Resistance Training involves doing any type of exercise that can make the muscles contract against a chosen external resistance (Weil, 2014). It has many health benefits, especially in today's world where many people very rarely get to work their muscles for what they were designed for.

In a report by the NSCA, there are three basic principles discussed which are overload, variation and specificity (Stone, 2000):

- Overload is for stimulating properly a specified physical adaptation. It involves exercising beyond normal levels of intensity. Intensity is measured by the amount of weight lifted where as volume is measured by the amount of repetitions and sets performed in each exercise. The best estimation for the amount completed is Volume load (load x volume).
- Variation refers to the speed of movement and volume and exercise selection. Specific changes to this area can result in superior enhancement in performance abilities.
- Specificity is described as the most important principle. It involves the mechanics of exercise movements including body regions, direction, muscle regimes, timing and velocity. The factor that is seen as the most important one is the rate of force produced. This involves the timing and velocity of movements.

This information points out to a key part of the project proposal where sensors could help in the analysis and feedback of timing and velocity movement.

2.1.2 Principle of Progressive Overload

Principle of progressive overload is well known in the area of Strength Training and it is viewed as “universally accepted as the model that creates the greatest gains in strength” (Weil, 2014). The physiologist explains that to follow this principle, one has to lift weights which are heavy enough to cause fatigue in the muscle towards the end of a set. Once you are able to reach the end of a set, the weight can then be increased to keep the challenge progressive. More weight means fewer reps but increasingly strong muscles means one can accomplish more of these reps.

2.1.3 1RM

reps performed	squat coefficient	bench coefficient	deadlift coefficient
1	1.000	1.000	1.000
2	1.0475	1.035	1.065
3	1.13	1.08	1.13
4	1.1575	1.115	1.147
5	1.2	1.15	1.164
6	1.242	1.18	1.181
7	1.284	1.22	1.198
8	1.326	1.255	1.22
9	1.368	1.29	1.232
10	1.41	1.325	1.24

Figure 2.1-1 Rep co- efficiencies (Butt, 2001)

As with the Principle of Progressive Overload, one can accomplish more repetitions with a lighter weight compared to doing it with a heavier load. A weigh trainer article (Butt, 2001) explains if one can perform a number of repetitions with a certain weight, a calculation can be made to determine how much weight the person can lift in just one rep. This weight is known as the one rep max (1RM). If a person's 1RM is known, a personal trainer can then use this to determine the weight load for the amount of desired reps for any given workout plan. This article provides a table, **Figure 2.1-1**, from the National Strength and Conditioning Association that provides co-efficiencies for every rep performed in three well known exercises.

With these efficiencies, three areas of useful information is provided for the trainer. This includes determining muscle efficiency, judging progress and determining weak links in a person's strength. These figures could be used to provide essential feedback to a user of a gym application.

$$1 \text{ rep max} = \frac{\text{weight lifted}}{1.0278 - (0.0278 * \# \text{ of reps})}$$

Figure 2.1-2 The Brzycki Forumla (Butt, 2001)

A formula mentioned in the article for determining the 1RM is the Brzycki Formula, **Figure 2.1-2**. The Scientific Electronic Library Online (SciELO) provides an article that investigates the accuracy of this formula (Materko, 2007). According to this article, the tests performed gave positive results. Acceptable reliability was proven and it concluded that this formula can be used as a tool for predicting a person's 1RM weight load.

2.1.4 Velocity-based training

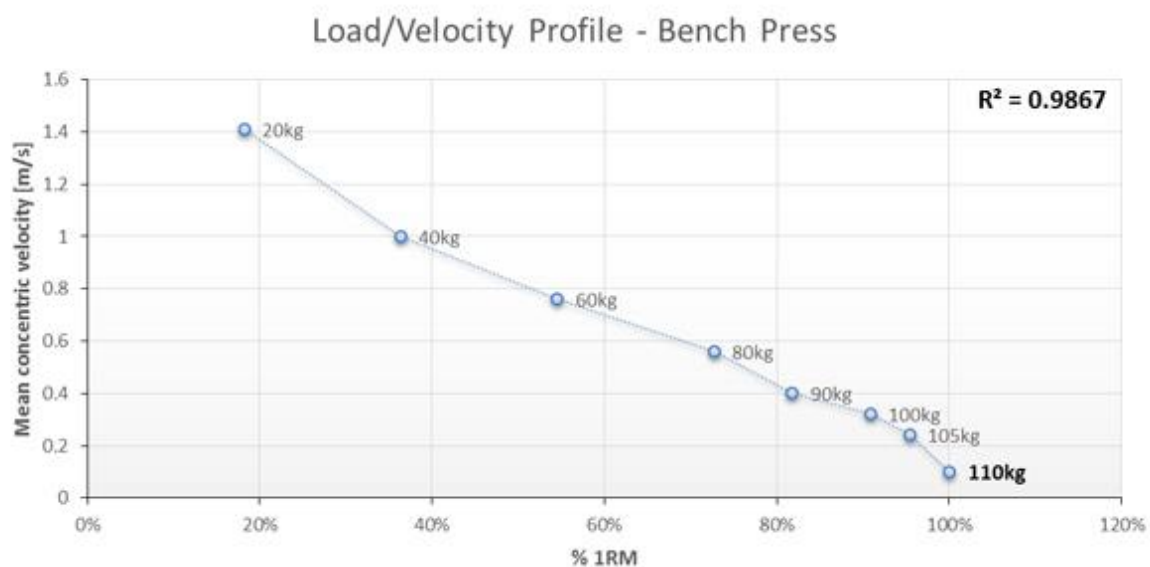


Figure 2.1-3 The Load/Velocity Relationship (Jovanović, 2014)

An article in Elifefts (Jovanović, 2014) explains the Load/Velocity Relationship. The load on each rep has an impact on how fast the user can complete it. The lighter the load, the faster the rep, while the heavier the load, the speed of the rep gets lower. It also explains how this relationship can be graphed as a simple linear as shown in **figure 2.1-3**.

It points out as an example, that 80% of a user's 1RM would always have a similar velocity regardless of changes to the user's 1RM over time (if the user's 1RM were to improve over a number of weeks). This means that by knowing the speed of a particular percentage of a 1RM, the user would always know they have reached a certain exertion level if that particular velocity could be detected for them.

Another interesting point is the Velocity/Exertion relationship. The velocity of the final rep (failure) is similar if not exact to the velocity of the 1RM. This is true regardless of load or the amount of repetitions. When the user keeps going until exertion is reached, the same velocity should always occur.

The relationships described here represents a concept that is used in velocity based strength training. It's a concept that can predict, monitor and regulate a person's strength training program.

2.1.5 Velocity Measuring Devices

Velocity and using velocity measuring devices is discussed in an NSCA article (Mann, 2014). It points out that tracking real-time velocity and power can provide information on how an athlete moves weight at a certain velocity and improve a specific goal. The use of velocity can determine the correct load for the day that is not based on previous sessions like traditional methods might do. It's claimed that a velocity measuring device can provide live feedback which can determine the correct loads to lift for the user.

2.1.6 Workout Planning

There are many types of workout plans that can be used weekly and an article in Muscle & Fitness describes four classic templates including the conjugate system, the linear periodization, undulating periodization and the 5 x 5. (Tuthill, 2014). All these templates involve doing different exercises depending on the day of the week. Sets and weights change weekly in accordance with the progressive overload as previously discussed.

With this information, it is obvious that a software application could provide benefits in keeping track of workout data and providing feedback to the user, especially on the principle of progressive overload where weight progression can change after each workout.

2.1.7 Strength Training Conclusion

A brief investigation reveals how an application can play a big role in analysing, adjusting and providing feedback for a strength training plan. There are many scenarios depending on the training plan goal and progress, where a lot of different data will need to be tracked and adjusted.

Velocity can play a huge role in strength training and if the phone's sensors can monitor and compare the different velocities involved in a user's program, than the user can benefit greatly without ever the need for buying any extra hardware.

2.2 Android Sensors

This section will look into Android's sensor capabilities. In particular the sensors for Samsung's Galaxy S3 as this is the device that will be used in this project.

2.2.1 Galaxy S3 Sensor List

According to Samsung's specification list (Samsung, 2014), the Galaxy S3 has six sensors;

- Accelerometer
- RGB light
- Digital compass
- Proximity
- Gyroscope
- Barometer.

2.2.2 Accelerometer

The accelerometer is explained as a device that can measure acceleration or vibration of a moving object (Omega, 2014). The explanation describes how the vibration or motion change causes force which in turn causes the squeeze in mass within the device. In some accelerometers this can cause electric charge which is proportional to the acceleration.

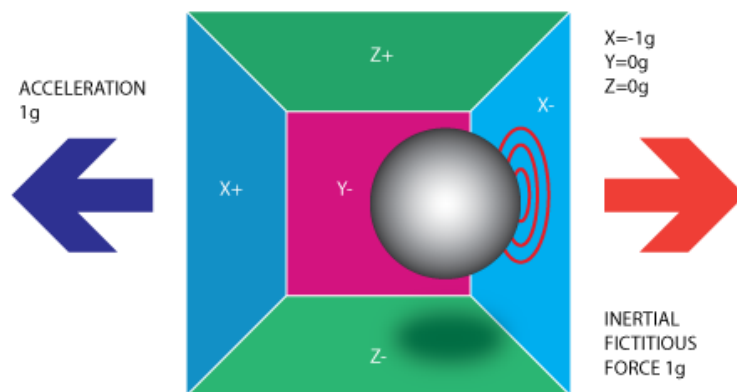


Figure 2.2-1 Acceleration and force (Anon., 2014)

An Instructables tutorial (Anon., 2014) provides a simple illustration, **Figure 2.2-1**, where the acceleration of the box is in the direction of the left (X+). This forces the ball towards the right, the opposite direction, where the ball hits the x- wall. The result is a gravity force of -1g.

There are many areas where accelerometers are used (Electronics, 2014), such as vibration of vehicles, machines, in agriculture to detect behaviour of animals and they can also be found in health monitoring applications to report data. The accelerometer has become so widely used that Wired UK published an article titled “Accelerometers are the most under-appreciated technology innovation”. The article (Tatton-Brown, 2012), discusses the impact these sensors have made on technologies we use and how it has made certain gaming features possible such as tilting on control devices.

2.2.3 Gyroscope

A Canadian science museum defines a Gyroscope as “any object mounted so that it turns very quickly around an axis of symmetry” (Museum, 2014).

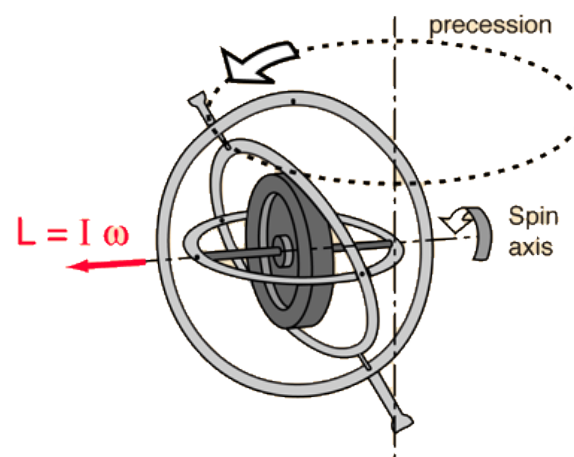


Figure 2.2-2 A conventional gyroscope (HyperPhysics, 2014)

Gyroscopes respond to rotation and they can measure rotation in inertial space. The example in **figure 2.2-2** illustrates a conventional gyroscope where as in recent years solid state gyroscopes have become available. These newer devices measure vibration and the direction the vibration is applied from (Altheris, 2014).

Gyroscopes can be used in many situations (Turner, 2014) including robotics, racing cars, science demonstration and computing pointing devices where movements and directions of movements are measured for various types of scenarios.

2.2.4 Barometer

Barometers measure atmospheric pressure and are used in phones for weather forecasting and detecting altitude (Levi, 2011). It also is a passive sensor that is not heavy on battery use.

2.2.5 Sensor fusion and filtering

An EE article explains Inertial Measurement Units (IMUs) that combine a three-axis accelerometer and a three-axis gyroscope into one unit (Johnson, 2014). It focuses on an IMU unit by Bosch that uses different combinations of sensors such as an accelerometer-magnetometer for navigation and the accelerometer-gyroscope combination for gaming and recognising gesture. This unit also includes a barometer. The IMU device has features motion tracking, tremor cancellation, distortion detection and stability enhancement for indoor tracking.

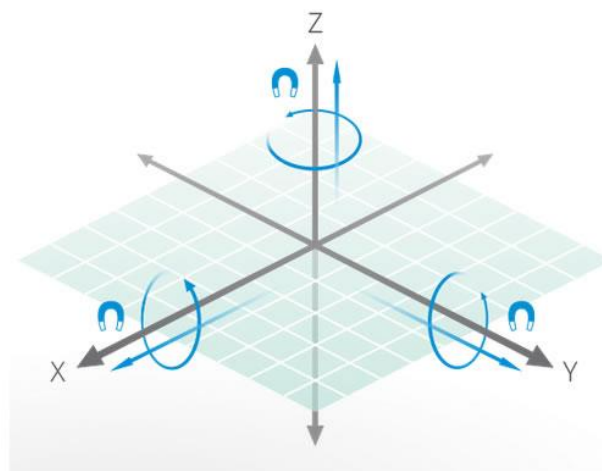


Figure 2.2-3 Acceleration (Sailing, 2014)

An illustration of a nine-axis sensor, **figure 2.2-3**, shows the different movement that can be detected combining an accelerometer, a gyroscope and a magnetic sensor. Note the three axis of X, Y and Z. Movements can move along these axis, as well as movement of rotation. The magnetic sensor on each axis completes the nine axis. By taking the magnetic sensor out, it becomes a six-component force sensor described by Showa technology company (Showa, 2007). These six kinds of forces can detect movement in vehicle tests, man-shaped robot controlling and medical applications where human behaviours can be analysed when the human body is in standing upright.

In an introductory to sensor fusion (Elmenreich, 2014), it is explained how sensor fusion combines data from multiple sensors so it results in a more reliable end result. Motivations in the paper for using sensor fusion include imprecision and uncertainty of the sensors. The expected advantages of using the fused data include reliability, extended spatial coverage and reduced ambiguity although the paper claims slight scepticism exists on the ability to perfect an end result. Fusion algorithms are discussed for smoothing, filtering and predicting data as well algorithms for situations when sensors may become temporarily blocked and/or when data need to be reduced. It names the Kalman Filter and the Bayesian reasoning as the tools most commonly used in sensor fusion.

The Kalman Filter is claimed to be “one of the most important and common data fusion algorithms in use today” (Faragher, 2012). The British scientist points out that the Kalman Filter was in the Apollo computer for the Neil Armstrong moon landing and it is widely used to today in every satellite navigation device and smart phone as well as many computer games for the improvement of navigation and movement detection purposes.

2.2.6 Galaxy S3 Sensor Hardware

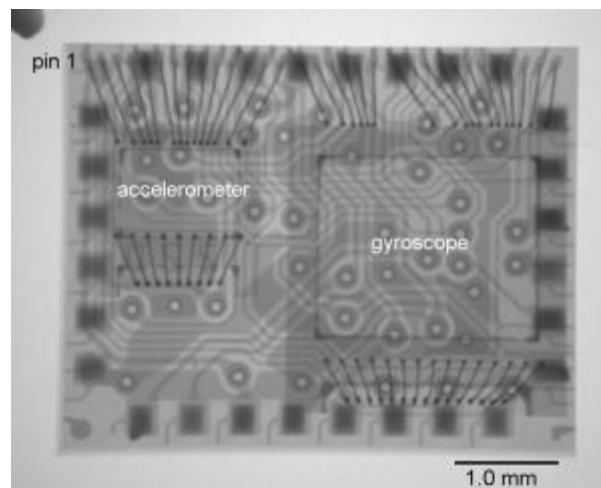


Figure 2.2-4 LSM330DLC device by ST Microelectronics (Dixon-Warren, 2014)

In the Galaxy S3, the accelerometer and gyroscope are combined as a six axis Inertial Sensor made by ST Microelectronics as discovered in a “teardown analyse” by Chipworks (Dixon-Warren, 2014). This analyses describes the LSM330DLC device, **Figure 2.2-4**, as a three-axis accelerometer and a three-axis gyroscope combined functionality. The ST specification (Microelectronics, 2012) lists features which include a low power mode with three independent acceleration channels and three angular rate channels. The device also includes high-pass and low-pass filter configuration.

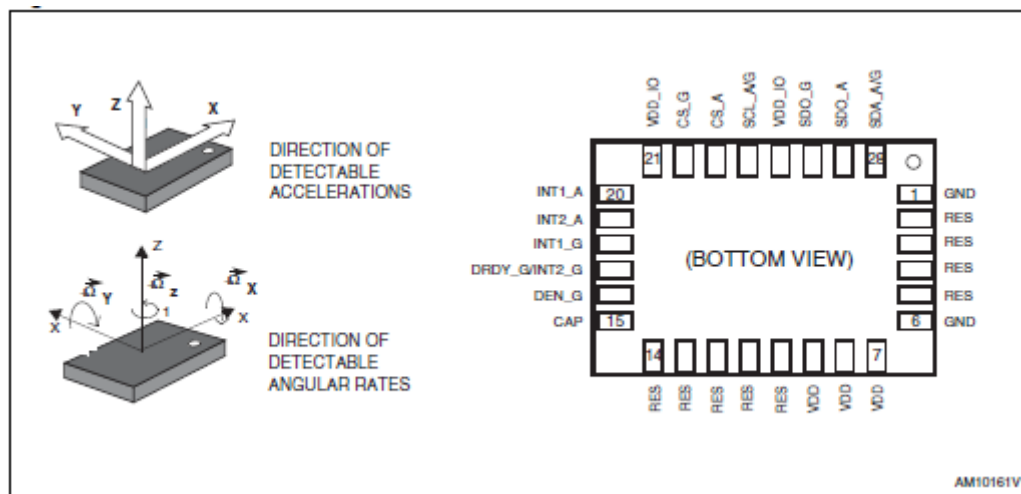


Figure 2.2-5 LSM330DLC Direction of Detection (Microelectronics, 2012)

The specification illustrates the direction of detection caused by the two sensors in the unit, **Figure 2.2-5**.

The Electronic Compass in the Galaxy S3 is the AKM8975 Electronic Compass sensor in the Galaxy S3 (Chipworks, 2012). The AKM provides a datasheet that describes the sensor as a 3-axis magnetometer device for use in GPS-equipped cell phones or to realize pedestrian navigation (Kasei, 2004).

The proximity sensor explained by Android (Android, 2014), determines how close an object is to the front of device. There are many examples in the Android Play store where the sensor can sense hand movements being passed in front of it. The application can perform any action necessary when it detects these movements. This sensor is included in the list of sensors for the Galaxy S3.

2.2.7 ST iNemoEngine_PAAP

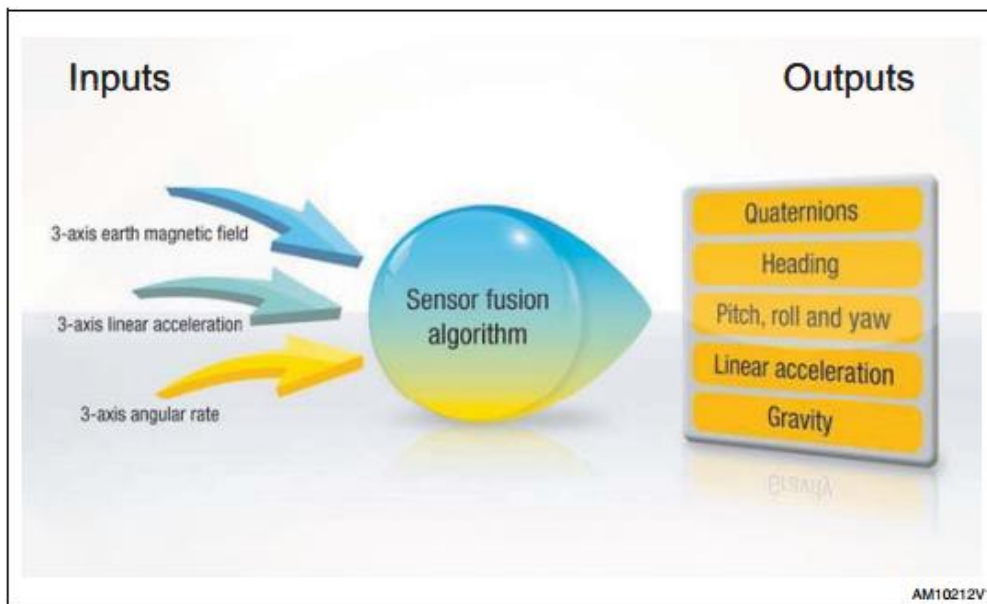


Figure 2.2-6 Sensor Fusion input and output (ST, 2014)

The same company, ST Microelectronics, that provide the Accelerometer and Gyroscope device on the Galaxy S3, is also responsible for the iNemoEngine_PAAP. It's product page (ST, 2014), provides documents on this software library. The engine is described as advanced software that fuses data from the accelerometer, gyroscope and the magnetometer. This provides a 9-axis sensor fusion as illustrated in, **Figure 2.2-6**, which also illustrates the inputs and outputs involved. Some key features listed include motion tracking accuracy, immunity to magnetic interference, accurate direction and compatibility with Android 2.3.3+. This is all possible by using an advanced algorithm based on the Kalman filter. The engine allows correction of magnetic distortion from the magnetometer, dynamic distortion from the accelerometer and drift issues from the gyroscope. Trading between performance and power saving can be configured in the library that is easily integrated into smart consumer devices.

On researching reviews on this product, all sources have had very positive opinions. For example, an article on Digi-Key, (Mathas, 2012), reflects on how sensor fusion has gone from military to consumer use and it targets the iNemoEngine as an example of this. It confirms much of what is in the iNemoEngine documentation and describes it as a firmware software.

The EE Times (Buckley, 2011), reports on the iNEMOEngine after it was unveiled in 2011. It reports on much of the features discussed already. However, it must be said that its report is based on information from ST Microelectronics as opposed to an independent review. It quotes the company's General Manager as saying "The iNEMOEngine delivers a quantum leap in the performance level required by next-generation smart mobile devices to enable a myriad of new exciting applications" (Buckley, 2011).

2.2.8 Android Sensor API

The Android Sensor API is well documented (Android, 2014). There is also a comprehensive list on the hardware with the methods and parameters that implement them under the Sensor class (Android, 2014). To use the sensors, a class needs to implement the SensorEventListener interface. The SensorManager manages every sensor within the class while the Sensor class is used for every Sensor object created.

A table of all the Sensor types supported by Android are listed in the API guide. Some of these are hardware based, while others are both hardware and software based. These software based Sensors, or Virtual Sensors, take data from one or more hardware based Sensors.

The following list takes Sensors from the Android list that may be of use to this project. All of these are motion based sensors or they at least facilitate in determining location and motion with other sensors.

TYPE_ACCELEROMETER: This is a hardware-based sensor that measures acceleration force on three axes. It is commonly used for motion detection and returns values of m/s^2 for x, y and z.

TYPE_GRAVITY: This is a virtual sensor that measures force of gravity on three axes. It is commonly used for motion detection and returns values of m/s^2 for x, y and z.

TYPE_GYROSCOPE: This is a hardware-based sensors that measures rate or rotations on three axes. It is commonly used for rotation detection and returns values of rad/s for x, y and z.

TYPE_LINEAR_ACCELERATION: This is a virtual sensor that measures acceleration force on three axes but excludes the force of gravity. It is commonly used for monitoring acceleration on a single axis and returns values in m/s^2 for x, y and z.

TYPE_MAGNETIC_FIELD: This is a hardware-based sensor that measures the ambient geomagnetic field for all three axes. It is commonly used for creating a compass and returns values for x, y and z.

All the above sensors are confirmed compatible for Android version 2.3+ (API Level 9+).

One could assume that for the Galaxy S3, the virtual sensors in the Android API implements the iNemoEngine software library discussed in the previous section. However, Google may have their own implementation for these software-based sensors. Further investigation will be needed to determine the implementation method of these virtual sensors to determine if any type of data filtering will be needed on the returned data. The API documentation provides advice on an implementation to test a device for which sensors, both hardware and software based, are using which particular hardware device and software firmware/library. This type of implementation may confirm if the iNemoEngine is used in the Galaxy S3.

Despite the fact that extra filtering may be needed or not, the documentation points out one issue that will need to be dealt with. This issue is offset. All values are based on the original point/position of the phone when the sensor started to detect data. According to the documentation, the offset will need to be taken from all data return to determine the true value.

2.2.9 The Physics of Movement

An understanding of the physics related to motion may be needed to calculate position, distance and displacement related acceleration and gravity values.

An article explains Newton's Law of Motion (Lucas, 2014). Newton's three laws tell us why everything moves the way it does. The first law states "a body at rest will remain at rest, and a body in motion will remain in motion unless it is acted upon by an external force" (Lucas, 2014). To break this down more simply, an object will not move on its own unless some force will make it move. In fact, it will resist change and this is known as inertia. The second law explains the reaction of when objects are forced to move. The mass of an object multiplied by its acceleration will equate to the force against it. This law looks into how objects can be forced into constant acceleration, how force can change their constant speed and also their direction. The third law is more simple and states that action will cause an equally opposite action. Examples used to explain these include pulling a rope and a rocket launching from the Earth.

Some motion terms are explained on a physics website (Boundless, 2014). It explains that vectors represent physical quantities that contain magnitude and direction. Acceleration is explained as time rate of change of velocity and for this value to be helpful, the magnitude and direction must be known also. Displacement is also explained where distance is defined between the positions of two objects. It can also define distance of an object's position to a certain reference point. This point could be the object's original position. Magnitude and direction are also used in defining velocity. The change of velocity must be described using speed and direction.

Knowledge of the physics of movement may be important throughout this project in understanding the behaviour of the sensors during exercise movements. It may be especially important when deciding how raw data is analysed and how movement is detected. When the above research, displacement and velocity may be calculated during the execution of exercises if this is required.

2.3 Artificial Neural Networks

2.3.1 Overview

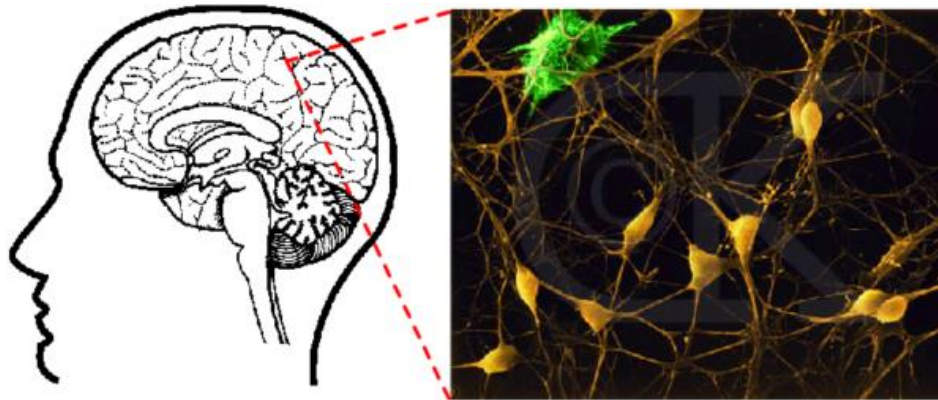


Figure 2.3-1 A Biological Inspiration (Stanford, 2000)

The idea of an Artificial Neural Network (ANN) in computing is a biological inspiration based on how the human/animal brain processes and memorizes information (Stanford, 2000). The diagram taken from Stanford, **Figure 2.3-1**, is used to illustrate the neural network in the brain. The yellow areas represent neurons which are connected by the lines known as the input and output channels. With about 10 billion neurons and each one connected to thousands of its neighbours, the brain can perform very complex tasks. Neurons exchange electrochemical inputs from each other and they only fire these signals if a cell body exceeds a certain level. Transmissions are formed as part of a large number of simple process units, which fire a binary signal if the weighted sum of its inputs reaches a certain level. Though ANNs are based on this idea, they are very simplistic in comparison but they have shown to be good at problems such as image recognition and making predictions based on past knowledge.

2.3.2 History

ANNs dates back to the 1940s but it was the emergence of computers in the 1950s that researches came together to create better networks (Bros, 2014). Physiologists, psychologists and computer engineers were among the types of researchers that contributed. It was this time that the Perception neural network was formed which was based on the processing within the eye. It was from work on improving this network that the back-propagation network was created which became popular in the 1980s.

2.3.3 Neural Networks vs Von Neumann

```
if (IsTired()) {  
    GoToSleep();  
} else {  
    StayAwake();  
}
```

Figure 2.3-2 Top-down programming (Stanford, 2000)

The comparison between how the traditional von-Neumann computers and neural networks work are discussed by Neural Network Solutions (Bros, 2014) and Stanford (Stanford, 2000). The von-Neumann style is reliable in computing sequential data in defined steps such as an algorithm. The computer needs to be given in advance the details of the sequence of instructions and algorithms. Not only must the algorithm be known in advance, the data needs to be precise also. The variability of problems and data in the real world is too complex for this type of computing. Take the simple top-down approach of a simple If-Else statement, **Figure 2.3-2**. What if the person is in an important meeting and can't sleep? Or it is late at night but there is an assignment due in the morning? Or maybe the assignment is not that important? The number of If-Else conditions needed to satisfy these questions makes things very complex for a decision the human brain would see as simple. From this example, it is hard to see how top-down programming could ever meet the complex demands of every day decision making the human brain can deal with. In contrast, the bottom-up approach is about doing and learning by example and mistakes. The human brain is a mix of both the top-down and bottom-up approach. Some behaviours are hard-wired into the brain while other behaviours are learnt over time. Arthur Samuel's chess machine is used as an example of how a machine cannot start off with a blank state. Certain knowledge of chess had to be given initially. Other comparisons include parallel processing compared to the limit of threads in von-Neumann computers. These computers also function in logic and rules while ANNs can function using images and concepts. Self-programming and speed (multiple processors) are other factors where ANNs can have advantages over the traditional computing methods.

2.3.4 Perceptron

A perceptron is an artificial neuron that takes several binary inputs and produces a single binary output (Nielson, 2014). Nielson gives an example to explain the perceptron and how it uses weight.

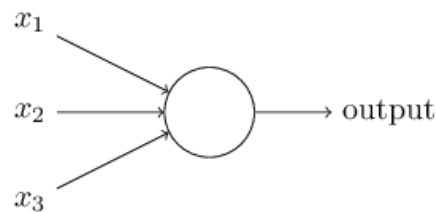


Figure 2.3-3 Several binary inputs, one binary output (Nielson, 2014)

A perceptron takes in several binary inputs and then computes a binary output using a weight with each input **Figure 2.3-3**. The perceptron also has a threshold value. If the weighted sum is greater than the threshold, the output is 1 or if it's below the threshold the output is 0. The algebraic term is as follows where x is an input and w is the weight:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Figure 2.3-4 Algebraic term of a perceptron (Nielson, 2014)

For a real-world example, a decision to go to a festival is used. Inputs represent factors that influences a decision to go or not. These factors include the weather, ease of public transport and availability of a friend for company. A person may be reluctantly go without the friend but the bad weather could be a complete turn off. This is where the weights come in. Here, the weather input would get a large weight while the friend input would get a small weight. The sum of the weights are measured against the threshold. This threshold can be adjust and the lower the threshold, the more willing the person is to go regardless the factors. As noted in the previous section, the model of human decision-making is far more complex. However this example shows how decisions can be made by weighing up different sources of evidence.

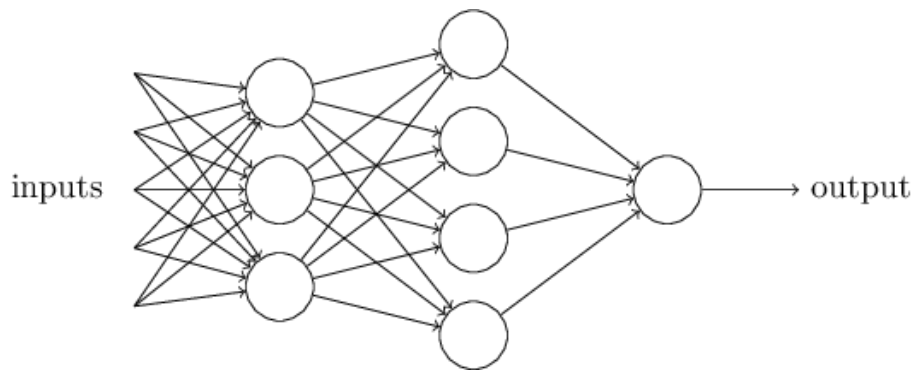


Figure 2.3-5 A complex network of perceptrons (Nielson, 2014)

In **Figure 2.3-5**, a network has different layers of perceptrons. The first layer could weigh up simple decisions and send the outputs to the next layer. The next layer could then make decisions on a more complex and abstract level compared to the first layer. In this way, having several layers of perceptrons means that more complex and sophisticated decisions are possible in a network.

Nielson gives other examples of how a perceptron network can be used such as using a bias instead of a threshold. Using weights and a threshold with a double digit binary number, a neural can act as NAND logic gate which gives opportunity for all types of functional computation in a network. However, the author points out that a neural network isn't just another example of a circuit of NAND gates. Introducing a learning algorithm can independently tune the weights and biases within the network. These learning algorithms opens up ways of making decisions that are radically different to normal logic gates.

With these examples given by Nielson, it is clear that neural networks can make sophisticated decisions. Where it would be difficult, and even impossible, to foresee how certain problems and situations develop for a top-down programming solution, a neural network on the other hand can learn based on how a problem develops and then adjust weights and biases accordingly.

2.3.5 Paradigms and Learning Rules

In a comprehensive introduction to Neural Networks, the different paradigms and learning rules involved are explained (Kriesel, 2005). Three paradigms of Neural Networks include unsupervised learning, reinforced learning and supervised learning. With unsupervised learning, only the input data is given to the network. It is then the task of the network to detect and find patterns that may exist in the data. With the reinforced learning paradigm, a value is returned to the network after there is a completion of a calculation. The value will give the network an indication whether it is learning correctly or incorrectly. With supervised learning, the network both receives input data along with the correct output. In this type of network, after an iteration, the error between the actual output and correct output is known. With the known error vector, the weights in the neurons can be adjusted accordingly at each iteration so eventually the network will be trained. Back propagation is also explained as the training of the perceptrons by changing their weights using a gradient descent procedure.

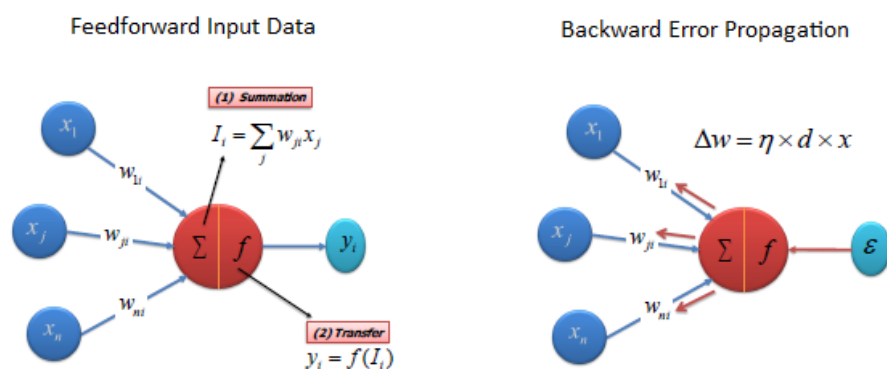


Figure 2.3-6 Feedforward and Backward Error Propagation (Sayad, 2014)

Neural Networks can be classified into two categories; feed-forward and feed-back (Sayad, 2014). Where a feed-forward is linear and all the data goes one direction, feedback has data moving both directions as illustrated in **Figure 2.3-6**. Loops are present in feed-back networks which means they are said to be non-linear.

Backpropagation learning rules can be used with a technique called Momentum (Istook, 2015). This is used to avoid oscillation problems. Adding momentum updates weights that are proportional to the error in the particular node. This technique can improve and maintain generalization performance while speeding up convergence.

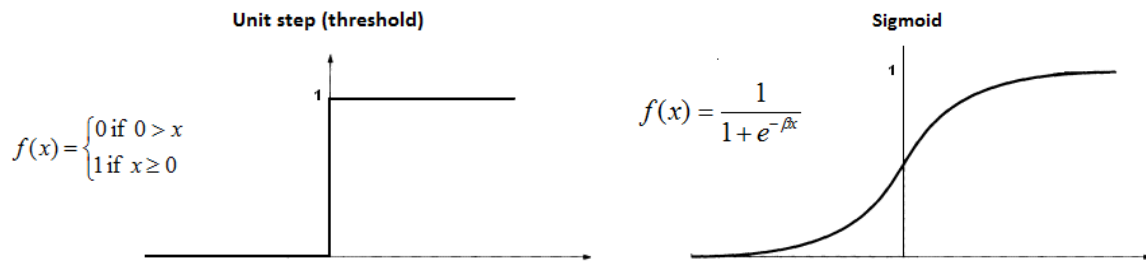


Figure 2.3-7 The Unit-Step Function and Sigmoid Function (Sayad, 2014)

Earlier the act of the perceptron transferring inputs to a single output was discussed. This is known as the transfer function (Sayad, 2014). Four common transfer functions (or activations) are presented by Dr Sayad and two of these are shown in **Figure 5.4-7**. The four functions are the Unit Step, Sigmoid, Piecewise Linear and Gaussian. The Unit Step function's output is simply based on the result of the input being greater or less than a threshold value. The other three have more complex ways of deciding the output. The Sigmoid is said to be the most common function (Gershenson, 2014). The Sigmoid function allows for smooth transition between high and low outputs which result in being either close to one or zero. For large positive numbers it is close to one, while for large negative numbers, it is close to zero.

2.3.6 Implementations

Many libraries for Neural Networks were found during research. Weka is a collection of machine learning algorithms written in Java (Weka, 2014). Tools are available for pre-processing, classification, regression and other machine learning tasks. These algorithms can be applied to datasets or called from methods from within a user's own Java application.

Neuroph (Neuroph, 2014) is specifically for neural networks. It supports perceptrons, backpropagation rules, supervised and unsupervised learning, an IDE based on Netbeans amongst many other features. It is also a project of the Apache Foundation.

Heaton Research (Heaton, 2014), provides Encog, another Java library for Neural Networks. It is also available for .Net, C and JavaScript. The website also contains samples of Neural Networks written in Java and also using Encog.

There are also Neural Network packages for the R language (MW, 2013). With these packages, feed forward and multi-layer perceptron models can be implemented as well as back propagation. Visualization of the networks are also available where neurons can be plotted.

2.3.7 Data Pre-Processing

The performance of a neural network is very much dependent on the quality of input data (Mendelsohn, 2014). Normalization and transformation are two methods that are widely used in pre-processing input data. Transformation involves creating a single input of data for a network from several inputs of raw data. Some methods for this include calculating the difference between the inputs or taking the ratio or average values. The risk of losing important data may exist in this process but this is the trade-off for smoothing out raw data to be readable for the neural network.

$$X_{i, 0 \text{ to } 1} = \frac{X_i - X_{\text{Min}}}{X_{\text{Max}} - X_{\text{Min}}}$$

Figure 2.3-8 Normalization Equation (Etzkorn, 2011)

Normalizing data involves scaling the transformed data into a certain range (Mendelsohn, 2014). This range could be from 0 to 1 or -1 to 1. These are the ranges used for the input values of the neurons. The formula in **Figure 2.3-8**, can be used for his normalization process, however, the maximum and minimum values along with the size range must be known before this can be done. Other normalizing methods include finding the central tendency and variance which can help in removing unnecessary outliers in the data.

2.3.8 Peak Detection

Finding peaks within a dataset can involve discovering all maxima values and then filtering out unneeded maxima if desired (Kaurov, 2014). Methods for this process involve comparing points in the data to neighbouring points within a certain range. The detection process also described as comparing a local maximum value to the smooth data (Whuber, 2012). The amount of neighbouring data, or the window size will determine how many peaks are found. A large window size will mean that fewer, but more extreme peaks are

found while smaller window sizes will return a greater number of less extreme peaks. When the window size is determined, the final step is finding the maximum values within this area. Other factors to consider in peak detection is the use of standard deviation and filtering data over a certain height range. The detection of peaks may return an unsatisfactory number of peaks, so testing different techniques, window sizes and other parameters will be required to find the ideal outcome.

Chapter 3: Methodology

3.1 Exercises

For the purpose of this project, five exercises will be used to test the ability of the Android sensors with a Neural Network.

- Lateral Raise
- Bicep Curl
- Triceps Kick-Back

These exercises differ from each other in terms of motion and direction. This should allow for testing all the different ranges across the different axes. More may be added to test if similar movements can be distinguished from each other. The research provided information on the areas of strength training that are important for analysing progress. Important data that will be considered in the implementation include speed of reps, frequency of reps and break times. Data stamps could be used to aid in calculating this information. However, more complex feedback will include detecting which exercise is being executed without the user needing to tell the device.

3.2 Android

The Samsung Galaxy S3 will be used for this project. This device has Android Kit Kat version 4.3.1 which uses Android SDK API 18. Research has shown that this device can provide filtered data by fusing different sensors. Since this work is done in the firmware at a very low level using algorithms based on the Kalman filter, no additional filtering is necessary on the data retrieved. The TYPE_GRAVITY virtual sensor will be used which takes data from the accelerometer and gyroscope. Data will be sent to the server over a network using a Wi-Fi connection. Data will be in the format of a JSON object which can hold all the necessary data associated with the exercises executed including time, exercise name and user name. The device will also be receiving data from the server for the feedback results for each exercise.

3.3 The Server

The server application will be responsible for the main area of the project. The programming language of choice is Java 8 which will run on a Tomcat server. The framework Spring 4 will be used to cut down on boilerplate code in functionality including connecting to the databases. Spring Web will also be useful for providing REST calls to and from the Android application. A Mongo database will conveniently store the sensor data as JSON objects while an Oracle Database can be used to store information of the users and their login information. The server will be run locally in the development environment initially but cloud services such as Amazon EC2 or Azure can be used to allow for remote connections for the Android device.

3.4 Development Tools

The project will be built in a Java environment with Eclipse being the IDE of choice for both the Android and server application. Android Studio was considered for developing the Android application but it was still in beta mode during the research phase. Apache Maven will be the build tool choice which will bring a conventional folder structure and dependency management to the web application. This will allow for easier transition between different environments, especially moving the application from the development environment to a Linux Cloud environment when completed. Both projects will be hosted on Github to allow for source control management and the Eclipse Egit plugin will be used for pull and push commits to the SCM server from within the IDE.

3.5 Persisting Data

All the sensor data generated from the Android device will be stored in a Mongo Database. A dataset will build up during the lifetime of the project as the Android sensors will be continuously tested. This dataset can then be used for testing pre-processing techniques and also for using the neural network.

On test trials using the Android device, samples were sent from the phone to the Mongo database. For a 10-rep exercise, almost 2000 samples were collected. This amounted to about 250KB. This indicates that for a three-set exercise, data could accumulate up to 1MB but should not exceed it, unless very high reps are carried out. Tests could also be carried out to check if motion can be detected with smaller amount of data samples.

3.5 Data Pre-processing

All data input for the Neural Network must be of equal size. This poses a challenge as a set of reps could vary greatly in size depending on the amount of reps executed. Therefore reps will be extracted from the set and then sent to the neural network individually. This way, the network can learn the pattern of one single rep, as opposed to learning a set containing an arbitrary number of reps. Transforming and normalizing one rep at a time should be less complex which may allow for better input of data for the neural networks.

The trial also showed that reps clearly created extreme maxima within the data. By discovering these maxima within the data, the start and end points of each rep can be found and therefore, this should allow for the extraction of data for each rep.

Transforming the data from three axes will be achieved by calculating the averaging value of each point. This single input series of each rep will then need to be resized so it holds the correct amount of inputs according to the network size. Normalization will also be needed to bring all values within the range of 0 to 1.

3.6 Neural Network Implementation

Two of the Neural Network frameworks that were tried out from the research phase were Weka and the Neuroph Framework. Between the two of them, Neuroph was the preferred choice. While Weka had features for machine learning, only part of the application was dedicated to neural networks.

This was the opposite for Neuroph which is a dedicated framework for creating custom neural networks. It was a lot more intuitive where different datasets and networks can be easily tried out in its GUI before implementing them in into the project.

Another advantage to using Neuroph is that it comes with a large and well documented API. The API documents dozens of classes that can create networks, learning-rules, perceptrons datasets and functions among many other features. This will provide an excellent opportunity to create a custom made network to suit the type of data produced from the sensors.

Configuration of the neural network will initially follow the most common practises for classification of data. According to research, the most common practises for supervised neural networks that involve classification include the backpropagation rule that also

includes momentum for better performance. The perceptrons will use the Sigmoid Function and one hidden layer in the middle will be used. A complete rep lasts about two seconds and each sample is recorded every 10ms which should result in an average of about 200 samples per rep. Therefore, the input layer will use 200 neurons and the output layer will initially have two neurons. Two output neurons will allow a binary count of up to four which will be enough for testing three exercises. Neuroph recommends one hidden layer that uses half the amount of neurons as the input layer, which in this case will be 100. During testing, the network architecture and configuration can be changed accordingly until the most accurate results have been accomplished

Chapter 4: Design

4.1 Vision Document

4.1.1 Scope

The aim of this project is to design and implement an Android application that will provide tracking and analysis for resistance training exercises. It will provide feedback to the user during and after each exercise. Feedback will include an analysis of the user's exercises, a comparison to their previous workout and suggestions on how to progress in future workouts.

The user can indicate the start and finish of an exercise by pressing appropriate buttons on the screen. The phone will use its sensors to track each exercise and send the data to a server. The Android application will also present the feedback it receives from the server to inform the user of their progress and suggested future progression.

The server application will have the responsibility of processing the gym plan and exercise data. It will use a neural network to learn new exercise data and compare them to the current exercises being carried out by the user. It will need to determine exercise behaviours of the user such as which exercise was carried out and how many repetitions were completed. It will also analyse other data such as the time length of each rep so it can provide feedback based on velocity-based training principles.

It is aimed, through the analysis and interpretation of the server application, that the Android can receive reliable feedback for presentation to the user.

4.1.2 Risk Analysis - Overview

As this study looks into implementing technologies in new ways based on research on relevant technologies and of other related implementations, certain risks will exist throughout its lifecycle. One of the main risks will be attempting to read the data from the phone sensors and interpreting it into an accurate result. The sensors will need a certain level of accuracy for this to be successful. Using a neural network to analyse, learn and compare data will need to be successful enough to allow for relevant feedback to the user. If the network cannot learn effectively, the feedback may return inaccurate data which could influence the user to make incorrect decisions in the progress of their gym plan. Other risks include the ability of sending large data over a wireless connection, comfort level of strapping a phone to the user's arm and the ability of the phone to communicate effectively to the user while training.

4.1.3. Risk Analysis - Full List

The following table illustrates the list of risks involved in this study.

Low: 1-3, Medium: 4-7, High: 8-10

C: Cost, S: Schedule, P: Performance

No.	Risk Source	Probability			Impact			Impact Areas			Risk Response
		L	M	H	L	M	H	C	S	P	
01	Application not to user requirements	3				5			X	X	Review Design
02	Knowledge of Android		7			5			X	X	Android research
03	Android Sensors Accuracy			9			9			X	Data filtering
04	Android processing performance		7				8			X	Transfer more tasks to server application & reduce data rate
05	Knowledge of Java application on server	2			2				X	X	More research
06	Size & complexity of server application	3				4			X	X	Implement a framework
07	Knowledge of neural networks			8			8		X	X	More research
08	Neural Network performance		6				8		X	X	Algorithm review
09	Software bugs		6			6			X	X	More testing
10	Time constraint		7			7			X		Reduce exercises
11	Database performance		7			7				X	Research NoSQL databases
12	User intuitive	4				5				X	Research GUI Designs
13	Communication method of feedback	4				6			X	X	Research Android applications

4.1.2 High Level Feature List

001	The application shall be for an Android device
002	A user must login into their account to use the device
003	The user can use and create new gym routines
004	The user can train the application to learn new exercises
005	The application shall have a session mode for when the user is in the gym
006	The application shall read live data while the user performs an exercise
007	The user shall receive feedback from the application on their performance which will include data for time of reps and suggestions on how to adjust their routine according to their performance
008	The user shall have the option to share their progress on Facebook
009	The application shall provide links to web resources that inform the user on areas that need improvement
010	A diagram showing a the line of movement to compare proper and actual form of exercise shall be viewable to the user
011	A voice option shall be used to communicate to the user so they don't have to read the screen
012	The application shall create custom music playlists for the user's session
013	The application shall calculate calorie and weight estimations according to the user's progress
014	The application shall have a feature to allow the user to send exercise data to the server for analysis
15	The server application shall have a front-end to assist in the analysis of data during development. This can also show user's progress

4.1.3 Prioritization of Features

MoSCoW Method Prioritization

MUST HAVE	
001	The application shall be for an Android device
003	The user can use and create new gym routines
006	The application shall read live data while the user performs an exercise
007	The user shall receive feedback from the application on their performance which will include data for time of reps and suggestions on how to adjust their routine according to their performance
15	The server application shall have a front-end to assist in the analysis of data during development. This can also show user's progress

SHOULD HAVE	
002	A user must login into their account to use the device
004	The user can train the application to learn new exercises
005	The application shall have a session mode for when the user is in the gym

COULD HAVE	
011	A voice option shall be used to communicate to the user so they don't have to read the screen
010	A diagram showing a the line of movement to compare proper and actual form of exercise shall be viewable to the user
010	A diagram showing a the line of movement to compare proper and actual form of exercise shall be viewable to the user
013	The application shall calculate calorie and weight estimations according to the user's progress

WON'T HAVE	
009	The application shall provide links to web resources that inform the user on areas that need improvement
012	The application shall create custom music playlists for the user's session

4.2 Functional Specification

4.2.1 Use-Case Diagram

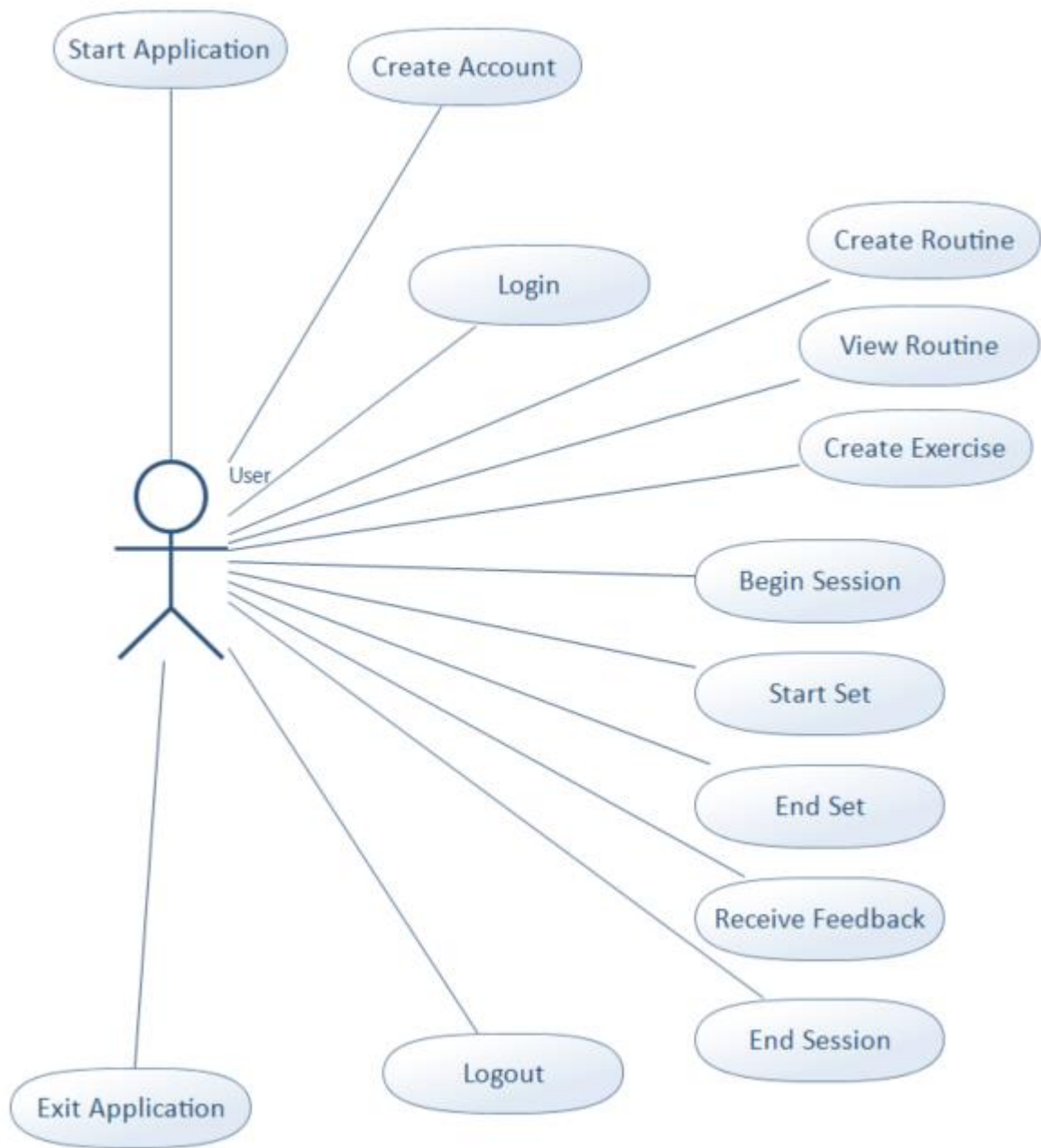


Figure 4.2-1 Use-Case Diagram

4.2.2 User Stories

Use Case Name	Start Application
Reference	001-001
Use Case Actor	User
Description	The application starts and presents to the user a login screen
Goal	As a user I want to start the application so I can use it for my gym routine
Acceptance	A successful launch and user is presented with a login screen
Flow of Events	The user press's the application icon. The application opens with a login screen
Assumptions	The application is being run on an Android smart phone

Use Case Name	Create Account
Reference	002-001
Use Case Actor	User
Description	The user provides a username and password to create an account
Goal	As a user I want to create a new account so my information can be saved for further retrieval
Acceptance	A new account is created for the user and the user is logged in
Flow of Events	The user enters a username and password and submits The application will notify the user if the username has been already taken or if the submitted details are invalid. If valid the a new account is created and the user is logged in.
Assumptions	The user is using a secure network

Use Case Name	Login
Reference	002-002
Use Case Actor	User
Description	The user enters a username and password to login to their account
Goal	As a user I want to login to my account so the application knows who I am
Acceptance	The application recognizes if the user is valid or not
Flow of Events	The user enters a username and password The application opens with a login screen if the user is valid The application asks the user to re-attempt if the user is invalid
Assumptions	The user is using a secure network

Use Case Name	Create new Gym Routine
Reference	003-001
Use Case Actor	User
Description	The user creates a gym routine to be carried out over a number of weeks.
Goal	As a user I want to create a new gym routine so I can track my progress for my new goal
Acceptance	A new gym routine, according to the user's selected options and is saved for future retrieval
Flow of Events	The user selects a start date. An option is then available to select the number of days to train each week. The user can then select what exercises to do each day. Finally, the user selects the amount of weeks and gives the routine a name and submits the information.
Assumptions	The user has some knowledge of gym planning.

Use Case Name	View Gym Routine
Reference	003-002
Use Case Actor	User
Description	The user can look at a selected gym to view progress details and edit any data if desired.
Goal	As a user I want to view and edit my gym routine to see and maintain my progress goal
Acceptance	The user sees a clear view of all the data in the routine and easily changes data that is required
Flow of Events	The user selects a particular routine from a list. An overview of the routine can be viewed and edited. Weeks and days can be selected for the selected routine and all the exercise data can be view for any selected day. Data can be edited and changes saved. When a day is selected, the user can start a session for that day.
Assumptions	The user has some knowledge of gym planning.

Use Case Name	Create new Exercise
Reference	004-001
Use Case Actor	User
Description	The user can record exercise movements for the application to recognise
Goal	As a user I want to record an exercise so the application can recognize for future use
Acceptance	The new exercise is recognised by the application
Flow of Events	The user executes the new exercise in perfect form. The application records the data and sends to the cloud for training.
Assumptions	The user carries out the exercise enough times for the application to learn

Use Case Name	Begin Session
Reference	005-001
Use Case Actor	User
Description	A user begins a session and the application goes into session mode.
Goal	The actor's goal is to get the application into session mode at the commencement of a gym session
Acceptance	As a user I want to begin a session mode so my phone is prepared
Flow of Events	The user selects a begin option from the view plan area. The screen goes dark and waits for the user's indication for the start of the next exercise set.
Assumptions	A full gym plan has been created and all exercises are known and trained into the application.

Use Case Name	Start Set
Reference	006-001
Use Case Actor	User
Description	The user indicates to the application that a set for an exercises is about to begin.
Goal	As a user I want to start a set so the phone can collect my data as I exercise
Acceptance	The application waits for 5 seconds and beeps to the user to begin exercises. Sensors are then turned on and data is collected.
Flow of Events	The user taps the phone and grabs weights. After 5 seconds of the tap, the phone beeps and the user begins exercise. The phone collects data while the user exercises.
Assumptions	The phone has the correct sensor hardware

Use Case Name	End Set
Reference	006-002
Use Case Actor	User
Description	The user indicates to the application that a set has completed
Goal	As a user I want to end a set so the phone can stop reading data and send it to the cloud
Acceptance	The application beeps in recognition and data is sent for analysis
Flow of Events	The user completes a set and taps the phone to indicate completion. The phone beeps in response. Data is sent to the server for analysis
Assumptions	The phone has the correct sensor hardware and data has successfully been tracked.

Use Case Name	Receive Feedback
Reference	007-001
Use Case Actor	User
Description	The user receives feedback from a completed set
Goal	As a user I want to receive and review the application feedback so I can be advised as to how I am progressing
Acceptance	The user can view correct data of the set including the name, amount of reps and form. Gym routine is adjusted correctly.
Flow of Events	After completion of an exercises, data is sent to the cloud for analysis. When feedback is received, the phone beeps, the screen comes on and the user can view all feedback. The user can confirm details are correct and accept new adjustments to the routine.
Assumptions	The internet is capable of sending and receive data almost instantaneous.

Use Case Name	End Session
Reference	005-002
Use Case Actor	User
Description	The user indicates to the application that the session is complete.
Goal	As a user I want to end a session so my phone can return to normal mode
Acceptance	The application exits session mode.
Flow of Events	The user selects to end session. The application ends session mode and the screen runs as normal.
Assumptions	

Use Case Name	Log out
Reference	002-003
Use Case Actor	User
Description	A user selects to log out from their account
Goal	As a user I want to logout of my account for security reasons or so another user can login to their account
Acceptance	The user is logged out and a login screen appears
Flow of Events	The use selects the log out button and the application logs out the user. A log screen is presented to the user to log in.
Assumptions	The user is logged in

Use Case Name	Exit application
Reference	001-002
Use Case Actor	User
Description	The user selects to exit the application
Goal	As a user I want to exit the application so it is not using phone resources when not in use
Acceptance	The application is terminated and doesn't use any of the phone's resources.
Flow of Events	The user selects the exit option. The application requires a confirmation. After confirmation the application terminates. The user can cancel selection where exit is cancelled
Assumptions	The user is logged out.

4.2.3 Prototype

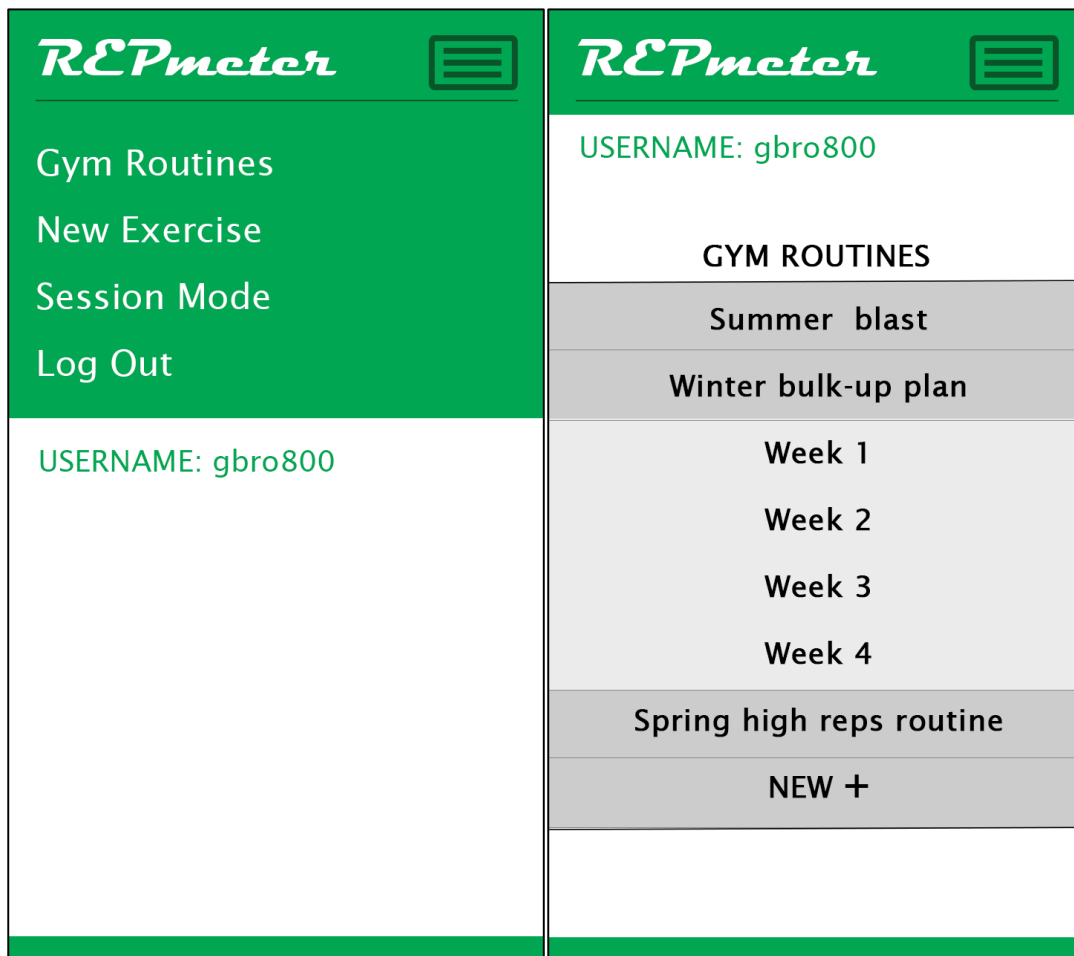


Figure 4.2-2 Menu Prototype

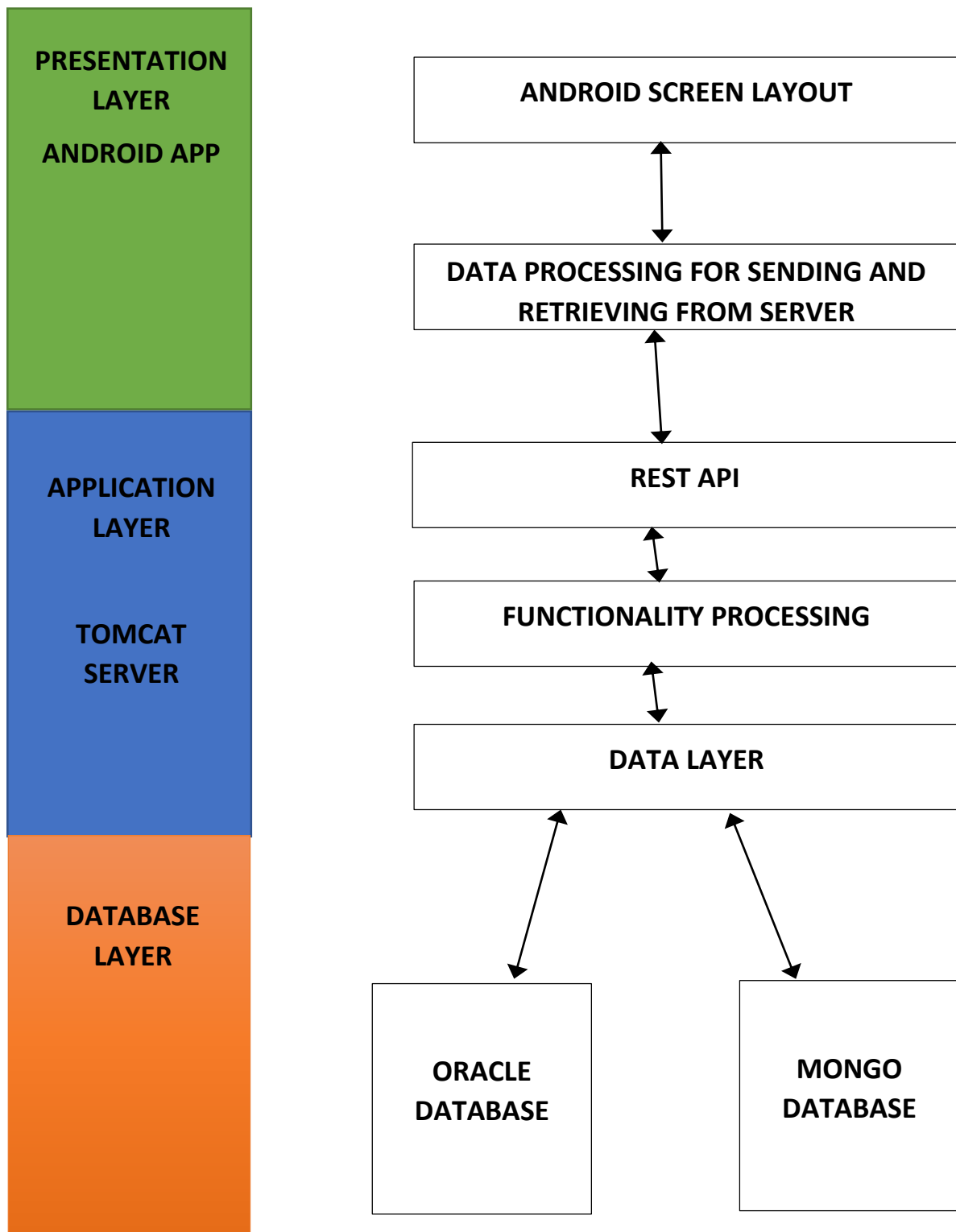
Figure 4.2-3 Gym Routines Prototype

Above are prototype screenshots of the application. On the left is a demonstration of how the menu slides down. The diagram on the right shows how the Gym Routine section may look like. The user can press a routine and the weeks drop down for selection. On the bottom of the list is an option to create a new routine. An alternative to the week drop down list is to create the list in a new screen where the user could navigate through a selected routine.

To prototype the data, the Android device was used to read sensor data which was sent to the server. This data was then saved as a JSON object in a Mongo database. Some exercises recorded over 2000 data samples over 10 repetitions. This is explained in detail in Sprint 1 of the Implementation chapter.

4.3 System Design

4.3.1 Architecture



4.3.2 Sprints

The project will have 8 sprints. Sprint 1 and 3 will last one week each, Sprint 2 will last four weeks while all other sprints will have a period of two weeks.

Sprint 1	Project Setup - Prototype
Commence	Monday, 08-December-2014
Complete	Friday, 12-December-2014
Tasks	<ul style="list-style-type: none">-Set up software tools and environment-Implement the end-to-end functionality: Send sensor data from the phone to the server and save to database.

Sprint 2	Pre-Processing of Data
Commence	Monday, 19-January-2015
Complete	Friday, 13-February-2015
Tasks	<ul style="list-style-type: none">-Transform the three axis data-Extract reps from data by detected the appropriate peaks-Adjust size of rep data to suit the input layer of the Neural Network-Normalise data

Sprint 3	Artificial Neural Network
Commence	Monday, 16-February-2015
Complete	Friday, 20-February-2015
Tasks	<ul style="list-style-type: none">-Set up the network using Neuroph Library-Test the network using different exercises and settings

Sprint 4	Android
Commence	Monday, 23-February-2015
Complete	Friday, 06-March-2015
Tasks	<ul style="list-style-type: none">-Create navigation screen-Create Dataset screen for sending exercise data to server-Create screen for demonstrating exercise detection

Sprint 5	Analysis (Front-end)
Commence	Monday, 09-March-2015
Complete	Friday, 20-March-2015
Tasks	<ul style="list-style-type: none"> -Create menu -Create Analysis page to select datasets -Show charts and allow user to change configuration of pre-processing -Create ANN page to set up neural network and CSV Files -Create Feedback page to show the feedback data

Sprint 6	Creating CSV Files
Commence	Monday, 16-March-2015
Complete	Friday, 27- March-2015
Tasks	<ul style="list-style-type: none"> -Create front-end facility for adding exercises for custom CSV files -Create menus that show lists of available datasets -Create CSV on the server with the selected data

Sprint 7	Front-End and Testing Neural Network
Commence	Monday, 30-Mar-2015
Complete	Friday, 10-April-2015
Tasks	<ul style="list-style-type: none"> -Create front-end facility for testing datasets with the ANN -Test and optimize network

Sprint 8	Decision Making / Feedback
Commence	Monday, 13-April-2015
Complete	Friday, 24-April-2015
Tasks	<ul style="list-style-type: none"> -Create Android Activity for feedback including inputs and buttons -Create webpage on the server front-end to show results

Chapter 5: Implementation

5.1 Sprint 1: Start-up/Prototype

The first sprint involved setting up the environment and getting an end-to-end implementation on sending data. More specifically, sensor data from the Android device was sent to the server and saved on a database. All Android and Server programming was done in Eclipse IDE.

First a design is created using labels and buttons.

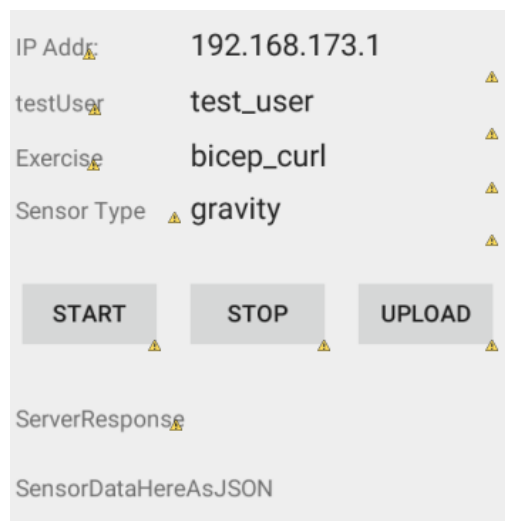


Figure 5.1-1 First Android Screenshot

A custom `SensorSample` class was created to hold the data of each sample recorded. This object is instantiated and holds the data added on every sensor change which could be as rapid as every 10 milliseconds.

```
public class SensorSample {
    private long timestamp;
    private double x;
    private double y;
    private double z;

    public SensorSample(long timestamp, double x, double y, double z) {
        this.timestamp = timestamp;
        this.x = x;
        this.y = y;
        this.z = z;
    }

    // GETTERS AND SETTERS
}
```

Figure 5.1-2 Sensor Sample Class

A custom Exercise class was created to hold data for an exercise with a List of all the Sensor samples. The following snippet shows the class header and its attributes.

```
import java.util.Date;
import java.util.List;

public class Exercise {

    private int id;
    private String username;
    private String exerciseName;
    private Date date;
    private List<SensorSample> sensorSampleList;

    public Exercise(int id, String username, String exerciseName,
                    Date date, List<SensorSample> sensorSampleList){
        setId(id);
        setUsername(username);
        setExerciseName(exerciseName);
        setDate(date);
        setSensorSampleList(sensorSampleList);
    }

    // GETTERS AND SETTERS

}
```

Figure 5.1-3 Exercise Class

Just one Android Activity class was used for this sprint. This class extends Activity and implements SensorEventListener and OnClickListener. A Sensor Manager class is instantiated to manager the sensor service. A Sensor class was instantiated with TYPE_GRAVITY sensor which is a virtual fused sensor.

```
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
sensorData = new ArrayList<SensorSample>();
```

Figure 5.1-4 Instantiating objects to read sensor data

The following snippet is the onSensorChanged method which needs to be implemented as part of the SensorEventListener interface. This executes for every new sensor change. Data values and a timestamp are assigned to variables and used to create the sample object which is then added to the List of samples. The started condition was to check if the user had pressed the start/stop button on the device.

```

@Override
public void onSensorChanged(SensorEvent event) {

    if (started) {
        double x = event.values[0];
        double y = event.values[1];
        double z = event.values[2];
        long timestamp = System.currentTimeMillis();
        SensorSample sample = new SensorSample(timestamp, x, y, z);
        sensorData.add(sample);
    }
}

```

Figure 5.1-5 Method data that reads in sensor data

On sensor stop, the exercise object is created with information from the text fields and the sensorData List. Now the Exercise object is created, it is converted to a JSON String using the Gson library for transfer to the server.

```

exercise = new Exercise(0, username, exerciseType, getNewDate(), sensorData);
Gson gson = new GsonBuilder().registerTypeAdapter(
    Exercise.class, new ExerciseJsonSerializer()).create();
exerciseAsString = gson.toJson(exercise);

```

Figure 5.1-6 Marshalling data to a JSON object

The most complex part of the code in this sprint involved sending the JSON data to the server. The Apache HttpClient was used to send the data over the network.

```
public void uploadDataToServer() {  
  
    String ipAddress = txtEnterIP.getText().toString();  
    String uri = "http://" + ipAddress + ":8080/ConnectionTest/hello/uploadtest";  
    HttpClient httpclient = new DefaultHttpClient();  
    HttpPost httppost = new HttpPost(uri);  
    String responseString = "";  
    try {  
        InputStream jsonStream = new ByteArrayInputStream(exerciseAsJsonString.getBytes());  
        InputStreamEntity reqEntity = new InputStreamEntity(jsonStream, -1);  
        reqEntity.setContentType("binary/octet-stream");  
        reqEntity.setChunked(true); // Send in multiple parts if needed  
        httppost.setEntity(reqEntity);  
        HttpResponse response = httpclient.execute(httppost);  
        StringBuilder sb = new StringBuilder();  
        BufferedReader reader = new BufferedReader(new InputStreamReader(response.getEntity()  
            ().getContent()), 65728);  
        String line = null;  
        while ((line = reader.readLine()) != null) {  
            sb.append(line);  
        }  
        responseString = sb.toString();  
        txtServerResponse.setText(responseString);  
    } catch (Exception e) {  
        txtServerResponse.setText(responseString);  
    }  
}
```

Figure 5.1-7 Streaming the data to the server using the Apache HTTP Client

On the server, a java application with Spring framework processes all the data. Here is the method in the REST layer that receives the data from the Android device. Here the JSON object in byte[] is streamed into a String. Using Gson, the data from the JSON string is transferred into an Exercise object. Now the data can be inserted into the database with the exerciseRepository object and the data is saved as an Excel file.

```

@RequestMapping(value = "/uploadtest", method = RequestMethod.POST)
public String handleFileUpload(HttpEntity<byte[]> requestEntity) {
    System.out.println("handle file");

    try {
        byte[] jsonStringBytes = requestEntity.getBody();
        String jsonString = new String(jsonStringBytes);
        Gson gson = new Gson();
        Exercise exercise = gson.fromJson(jsonString, Exercise.class);

        exerciseRepository.insert(exercise);
        excelCreator.createExcelFile(exercise);

        return "Server successfully saved Exercise";
    }
    catch (Exception ex) {
        return "Server error: " + ex.toString();
    }
}

```

Figure 5.1-8 The server REST method

The Exercise Repository is a Spring class customized to save the Exercise class to the Mongo database. This also creates the `_id` which increments starting at 1.

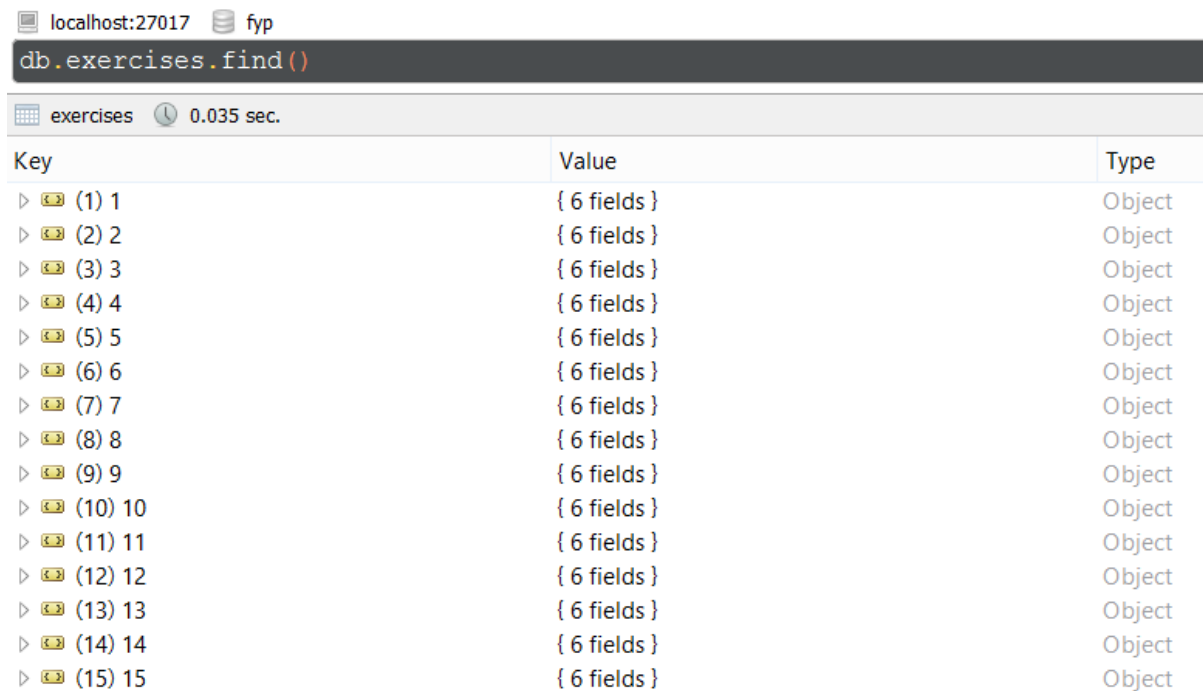
```

public void insert(Exercise exercise) {
    exercise.setId(getCollectionSize() + 1);
    mongoTemplate.insert(exercise, "exercises");
}

```

Figure 5.1-9 The exercise data is stored in Mongo

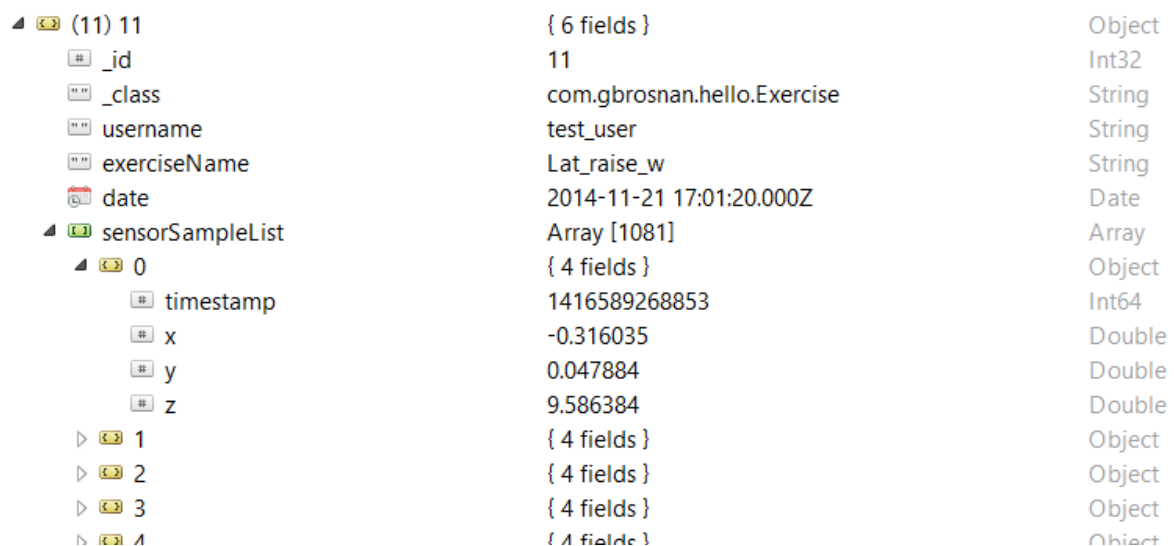
Robomongo, a tool for viewing data in a Mongo database, was used to view the raw data generated by the Android.



Key	Value	Type
▷ (1) 1	{ 6 fields }	Object
▷ (2) 2	{ 6 fields }	Object
▷ (3) 3	{ 6 fields }	Object
▷ (4) 4	{ 6 fields }	Object
▷ (5) 5	{ 6 fields }	Object
▷ (6) 6	{ 6 fields }	Object
▷ (7) 7	{ 6 fields }	Object
▷ (8) 8	{ 6 fields }	Object
▷ (9) 9	{ 6 fields }	Object
▷ (10) 10	{ 6 fields }	Object
▷ (11) 11	{ 6 fields }	Object
▷ (12) 12	{ 6 fields }	Object
▷ (13) 13	{ 6 fields }	Object
▷ (14) 14	{ 6 fields }	Object
▷ (15) 15	{ 6 fields }	Object

Figure 5.1-10 Robomongo displays the Mongo data

The final stage in demonstrating end to end data transfer shows the Android data saved into the database. This is _id 11, a lateral raise by test_user with the data and time shown. With this Exercise, there are 1081 samples of data in the array.



▲ (11) 11	{ 6 fields }	Object
_id	11	Int32
_class	com.gbrosnan.hello.Exercise	String
username	test_user	String
exerciseName	Lat_raise_w	String
date	2014-11-21 17:01:20.000Z	Date
sensorSampleList	Array [1081]	Array
0	{ 4 fields }	Object
timestamp	1416589268853	Int64
x	-0.316035	Double
y	0.047884	Double
z	9.586384	Double
1	{ 4 fields }	Object
2	{ 4 fields }	Object
3	{ 4 fields }	Object
4	{ 4 fields }	Object

Figure 5.1-11 Summary of the JSON Document

5.2 Sprint 2: Pre-Processing of Data

This Sprint involved pre-processing the raw sensor data that was taken from the Android device. This pre-processing was necessary so the Artificial Neural Network could read in the data successfully. For the data to be ready for the network, all values would need to be between the range of 1 and 0. Also, inputs would have to be of a set amount which means each dataset from the different exercises would always have to be the same size.

Since an exercise set could involve any number of reps, datasets would greatly vary in size. For this reason, it was decided to extract each rep from a set and send each rep to the neural network to be learned individually. This didn't solve the size problem completely since each rep could vary in size also, depending on the exercise, the user and the speed of execution. However, it would be easier to adjust each rep to a standard size compared to a whole set which could involve any random number of reps from three or four and right up to twenty and even more.

The first step of the pre-processing stage involved transforming the raw data from the three axes into one array. A simple solution of getting the average value from the three axes values was chosen.

```
private List<Double> setAverages(List<SensorSample> rawSensorSamples) {  
    List<Double> averages = new ArrayList<Double>();  
    double x, y, z, average;  
    for(SensorSample sample : rawSensorSamples) {  
        x = sample.getX();  
        y = sample.getY();  
        z = sample.getZ();  
        average = (x + y + z) / 3;  
        averages.add(average);  
    }  
    return averages;  
}
```

Figure 5.2-1 Method to find the average of each sample

In **Figure 5.2-1**, the method for getting the average values is shown. It takes in a List of the SensorSamples and then loops through it to and calculates the average. Each average is added to a List of Double, which is returned when complete.

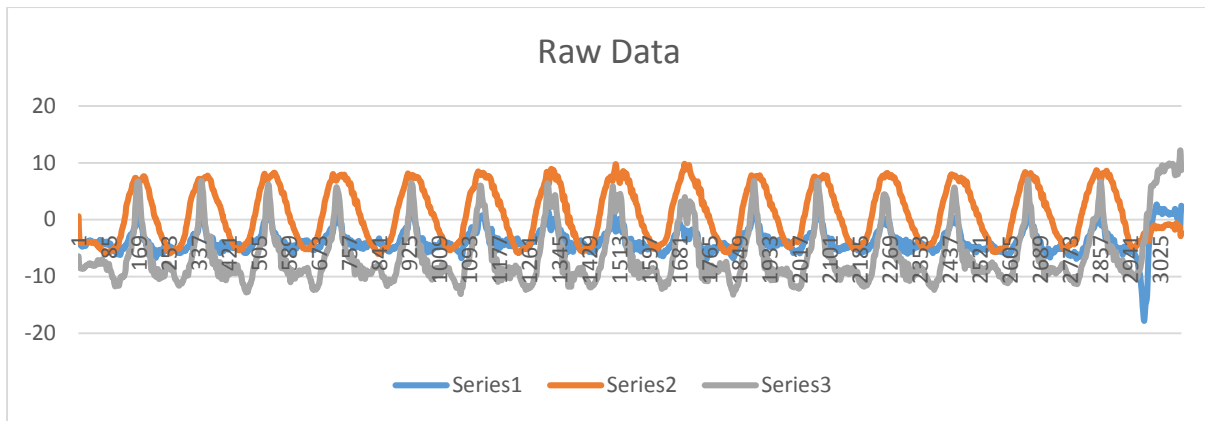


Figure 5.2-2 Raw data of X, Y & Z

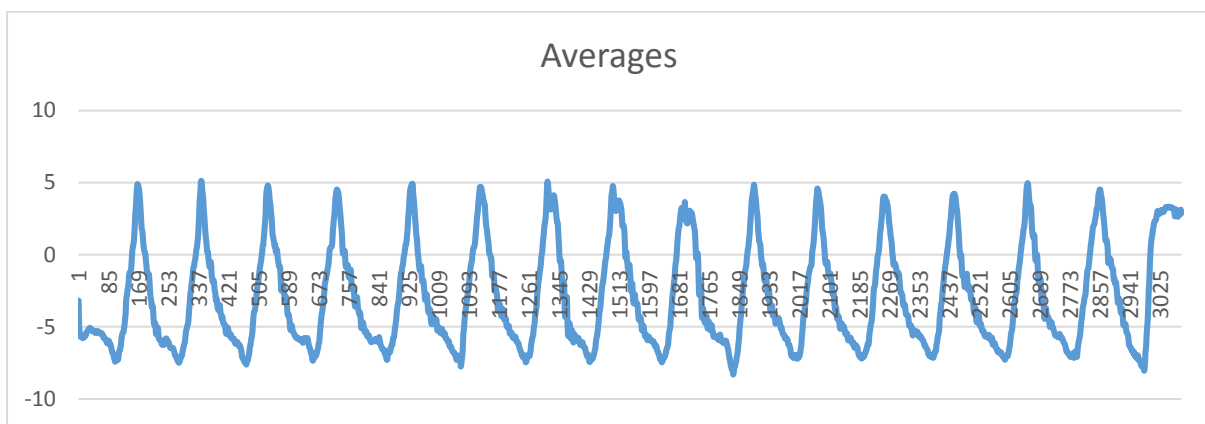


Figure 5.2-3 Single List of Averages

The results of this Sprint were saved to an Excel file and the above Excel charts show how the raw data of 15 bicep curls was taken in by the `setAverages()` method and a new List of Doubles was created. Note how the raw data values ranged from about -10 to 10 but when averaged, this was reduced from about -7 to 5. The most obvious explanation to this would be the lesser extreme values of Series 1 (blue) influenced the higher values of the other two axes to reduce when averaged. Though the extreme values had been averaged, the new set of data still showed 15 clear and similar patterns which could lead to a successful learning outcome for the neural network.

The next step was to find a way to extract each Rep from the data. The chart of Averages shows that each rep consists of one peak (maximum) and lower points at the start and end (minima). First each maximum value would need to be detected.

```

private static boolean isMaxima(int currentPoint, int startPoint,
                                int endPoint, List<Integer> maximas) {

    boolean maxima = true;
    for(int neighbour = startPoint; neighbour < endPoint ; neighbour++) {
        if(values.get(currentPoint) < values.get(neighbour) || maximas.contains(neighbour)) {
            maxima = false;
            break;
        }
    }
    return maxima;
}

```

Figure 5.2-4 Maxima

To detect the maxima, each point in the Average List had to be checked and compared to its neighbouring points. If it had the highest value within a certain range of points, it was deemed a maximum point. After some trial and error, a range of 100 sample points always gave a successful peak detection. The isMaxima() method in **Figure 5.2-4** is called for each point in the Average List. It first assumes the current point is maximum. With a range of 100, it will loop the 50 previous points and the 50 proceeding points from the current one. If it finds that any of its 100 neighbouring points has a higher value, then the current point cannot be a maximum point. Another condition is that if a maximum has already been detected within the range, the current one cannot be a maximum either. This condition was included for a situation where a peak had two neighbouring high points of the same value.

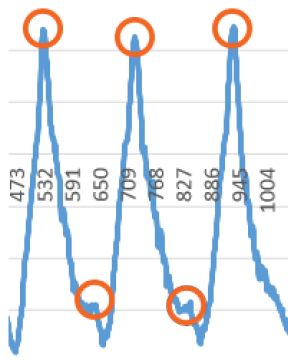


Figure 5.2-5 Range too narrow

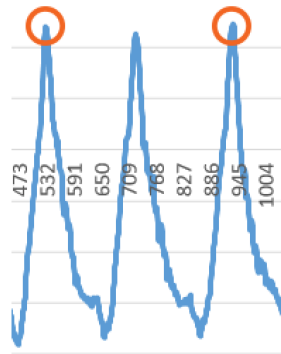


Figure 5.2-6 Range too wide

Figures **Figure 5.2-5** and **Figure 5.2-6** shows what could occur when the range is not set correctly. If the range is too narrow, small insignificant peaks are detected while if the range is too wide, some genuine peaks are skipped.

```

private static int getStartPoint(int point, int halfScope) {
    int start = point - halfScope;
    if(start < 0) {
        start = 0;
    }
    return start;
}

private static int getEndPoint(int point, int halfScope, int listSize) {
    int end = point + halfScope;
    if(end > listSize) {
        end = listSize;
    }
    return end;
}

```

Figure 5.2-7 Finding the start and end points

Other issues included getting out-of-bounds errors towards the start and end of the Average list. In the beginning, if it tries to loop through 50 of the previous points from the current point, it could go into negative values if the current point is below 50. Likewise, towards the end, if it tries to loop through 50 of the points proceeding the current point, it would go out of bounds if there were less than 50 points remaining in the Average list. The methods in **Figure 5.2-7** solved these issues.

```

public static List < Integer > discoverMaximas(List < Double > valuesProvided, int start, int range) {
    values = valuesProvided;
    List < Integer > maximas = new ArrayList < Integer > ();
    int startPoint = 0, endPoint = 0, valueListSize = values.size();

    for (int currentPoint = start; currentPoint < valueListSize; currentPoint++) {
        startPoint = getStartPoint(currentPoint, range / 2);
        endPoint = getEndPoint(currentPoint, range / 2, valueListSize);
        if (isMaxima(currentPoint, startPoint, endPoint, maximas)) {
            maximas.add(currentPoint);
        }
    }
    return maximas;
}

```

Figure 5.2-8 Finding the start and end points

The discoverMaxima() method **Figure 5.2-8** was used to loop through each point in the Average list. It takes in the list of doubles (Average list), a starting point which could be 0, and a range. The range was set to a variable as the best range wasn't known until after trial and error. This method calls on the get Start/End point methods and then uses these values when calling the isMaxima() method. All maximum points are added to a List and returned.

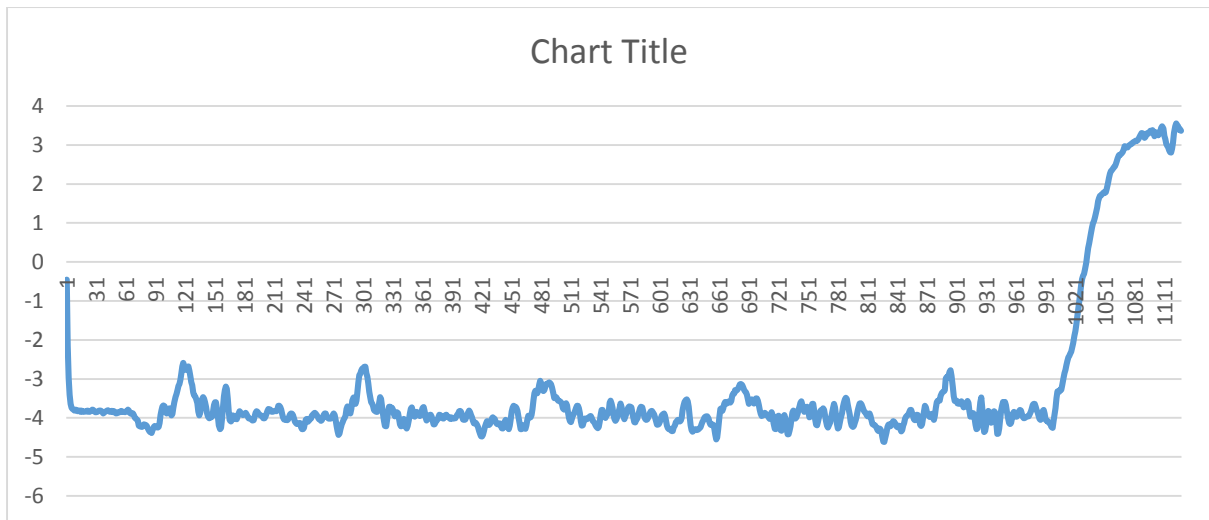


Figure 5.2-9 Different size peaks

```
public static List<Integer> filterOutFlatPeaks(List<Double> valuesProvided, List<Integer> maximas,
                                              int range, double percentageDrop) {

    List<Integer> filteredList = new ArrayList<Integer>();
    double heightRange = getHeightRangeWithinMiddleThird(valuesProvided);
    double maximaValue, startPointValue, endPointValue, dropRangeValue;
    int startPoint, endPoint;

    for(int maxima : maximas) {
        startPoint = getStartPoint(maxima, range);
        endPoint = getEndPoint(maxima, range, valuesProvided.size() - 1);
        startPointValue = valuesProvided.get(startPoint);
        endPointValue = valuesProvided.get(endPoint);
        maximaValue = valuesProvided.get(maxima);
        dropRangeValue = maximaValue - (heightRange * percentageDrop);
        if(dropRangeValue > startPointValue && dropRangeValue > endPointValue) {
            filteredList.add(maxima);
        }
    }
    return filteredList;
}
```

Figure 5.2-10 Filtering out flatter peaks code

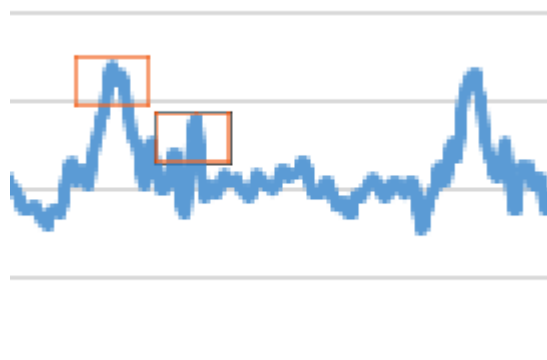


Figure 5.2-11 Filtering out flatter peaks illustration

Certain exercises such as the shoulder shrug in **Figure 5.2-9** show that many peaks of different sizes can form in the data. Many unwanted peaks could be detected in this scenario so a method `filterOutFlatPeaks`, **Figure 5.2-10**, was used to filter out these. A genuine maximum would rise clearly above all the others so by moving a certain distance to the right and left, and then down a certain distance, the location should still be above the data line. The second peak in **Figure 5.2-11** fails this test since there is other data within this range, and so is filtered out of the maxima list. Instead of actually filtering out each maxima, a new list is created where each genuine maxima is added and the list is returned by the method.

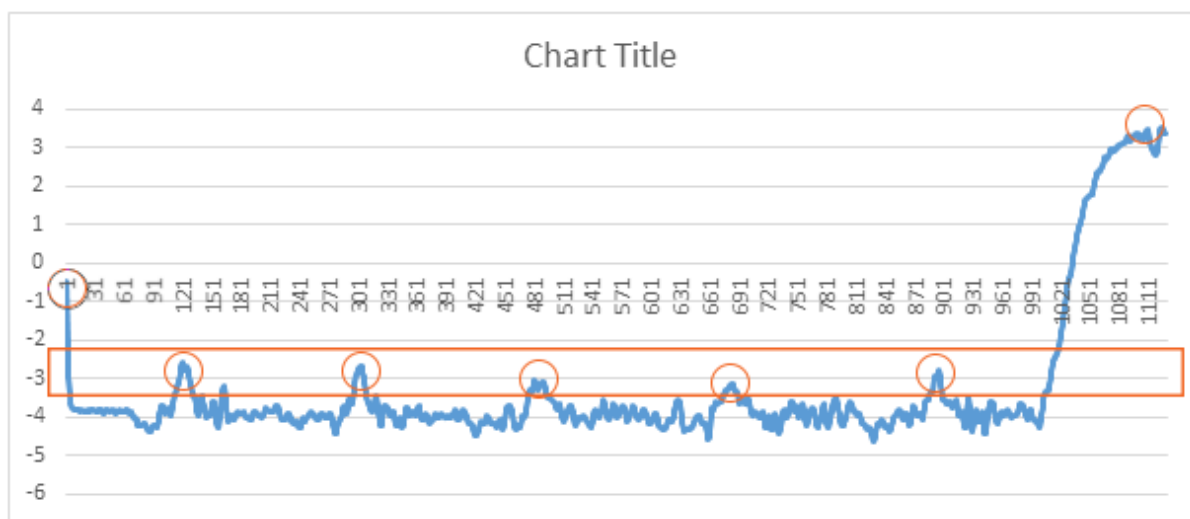


Figure 5.2-12 Filtering out by a certain height range

The chart in **Figure 5.2-12** illustrates how a dataset could have a peak detected at the very start and end of the chart. This can be caused by random movement for a few seconds before and after the execution of the exercise. Since this is random movement, extreme value ranges are recorded in the sensor data and peaks may be detected within these areas. To solve this problem, peaks outside 30% of the height range were filtered out. Using a percentage of the height was a problem since extreme heights were recorded at the start and end. Instead the height within the interquartile range was used where the highest and lowest values were found. 30% gave the best results, although it depended on the exercise also.

```

private static double getHeightRangeWithinMiddleThird(List<Double> valuesProvided) {
    double highest = -10000000;
    double lowest = 10000000;
    int[] midThirdRange = getMidRange(valuesProvided.size());
    for(int point = midThirdRange[0] ; point < midThirdRange[1] ; point ++){
        if(valuesProvided.get(point) > highest){
            highest = valuesProvided.get(point);
        }
        if(valuesProvided.get(point) < lowest){
            lowest = valuesProvided.get(point);
        }
    }
    return highest - lowest;
}

private static int[] getMidRange(int listSize) {
    int start = listSize / 4;
    int finish = (listSize / 4) * 3;
    return new int[] {start, finish};
}

```

Figure 5.2-12 The Interquartile Range (originally used for the middle third)

The screen shot in **Figure 5.2-12** shows how the height within the interquartile range was calculated. Originally the middle third was used as the name of the method suggests, however this was eventually changed to the interquartile range. This was easily changed in the `getMidRange()` method, dividing by four got a quarter (originally three to get a third). By getting the mid-range points, the loop can now use these values to loop within the interquartile range.

```

private static int discoverLowestPoint(List<Double> values, int start, int end) {
    int lowestPoint = 0;
    double lowestValue = 1000000;
    for(int point = start ; point < end ; point ++){
        if(values.get(point) < lowestValue){
            lowestValue = values.get(point);
            lowestPoint = point;
        }
    }
    return lowestPoint;
}

```

Figure 5.2-13 Discovering the lowest points (Minima)

All the static methods shown so far were saved into PeakDetect.java, a custom made utility class. These methods were created so a handler could call on them to find each of the maxima for extracting the reps. The method in **Figure 5.2-13** was taken from the next utility class, RepCreator.java. Once each maxima has been discovered, the lowest point within these points must be found to find the start and finish point of each rep. This method took in the Average list and the index numbers which represent a maximum point each. The for-loop goes through all the points within these two indexes and finds the lowest point.

```
private static Rep createRep(List<Double> values, int[] maximas, int id) {
    List<Double> repValues = new ArrayList<Double>();
    int startPoint = discoverLowestPoint(values, maximas[0], maximas[1]);
    int endPoint = discoverLowestPoint(values, maximas[1], maximas[2]);

    for(int point = startPoint ; point <= endPoint ; point ++ ) {
        repValues.add(values.get(point));
    }
    return new Rep(id, startPoint, values.get(startPoint), endPoint, values.get(endPoint), repValues);
}
```

Figure 5.2-14 Creating a Rep

The creating Rep method in **Figure 5.2-14** is the method that uses the discoverLowestPoint() to find the start and end of each Rep. It then creates an List of all the values within these two lowest points.

```
public static List<Rep> createRepList(List<Double> values, List<Integer> maximas) {
    List<Rep> reps = new ArrayList<Rep>();
    int id = 0;

    for(int maxima = 0 ; maxima < maximas.size() ; maxima ++ ) {
        if(maxima == 0) {
            reps.add(createRep(values, new int[] {0, maximas.get(maxima), maximas.get(maxima + 1)}, ++id ));
        }
        else if(maxima == maximas.size() - 1) {
            reps.add(createRep(values, new int[] {maximas.get(maxima - 1), maximas.get(maxima), values.size() - 1}, ++id ));
        }
        else {
            reps.add(createRep(values, new int[] {maximas.get(maxima - 1), maximas.get(maxima), maximas.get(maxima + 1)}, ++id ));
        }
    }
    return reps;
}
```

Figure 5.2-15 Creating a List of Rep

Finally a method is used to create a list of Rep for each exercise. It calls on the methods previously discussed. The If-conditions are in place to handle the first and last reps as they measure the lowest points from the very start and end respectively.

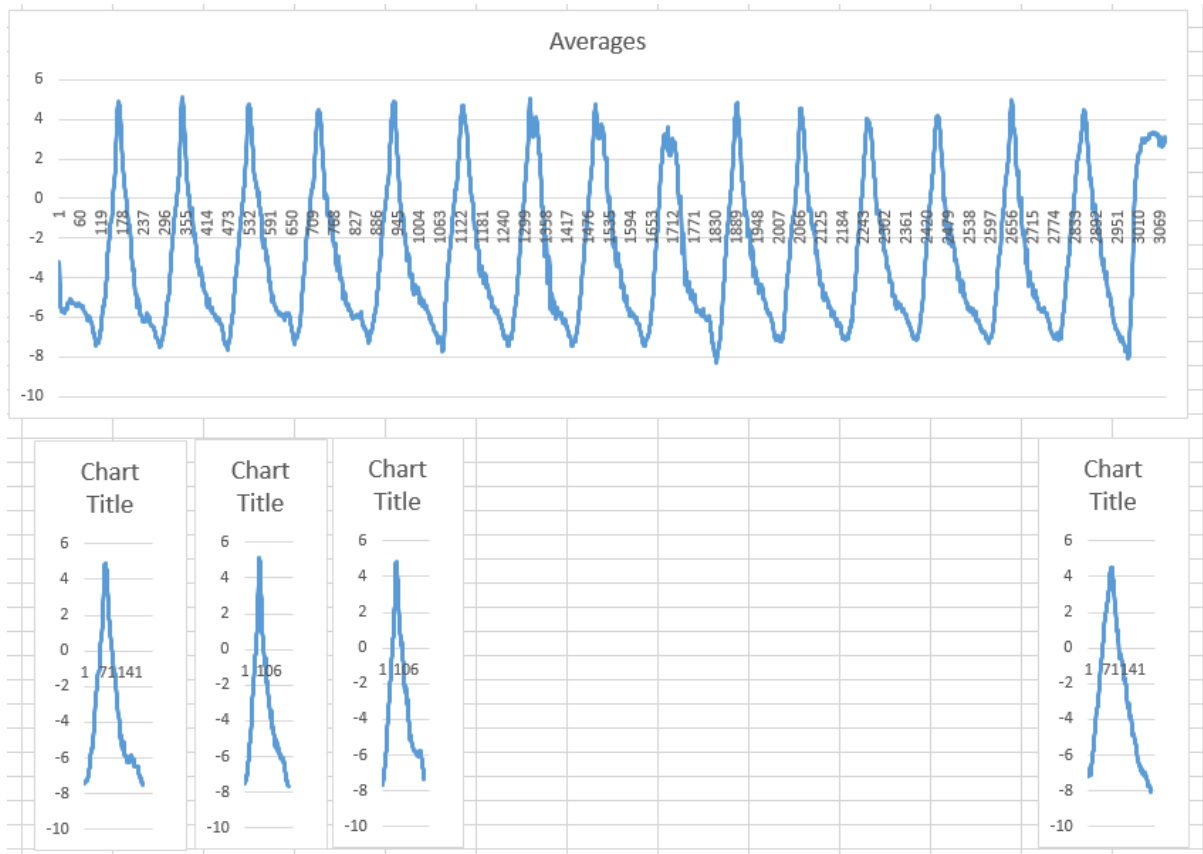


Figure 5.2-16 Rep Extraction Charts

To illustrate that the reps were extracted successfully, diagrams were inserted into Excel using the values from some of the Reps extracted. This was done with the 15 Bicep Curl exercise.

$$X_{i, 0 \text{ to } 1} = \frac{X_i - X_{\text{Min}}}{X_{\text{Max}} - X_{\text{Min}}}$$

Figure 5.2-17 Normalization Equation (Etzkorn, 2011)

The next step in the Pre-Processing stage was to normalize the data so all values would be between 1 and 0. This was done using a normalization equation for each of the values in the Average list. The max and min values first had to be found before the equation was applied.

```

private static double[] getHighAndLowValues(List<Double> values) {
    double highest = -10000000;
    double lowest = 10000000;

    for(double value : values) {
        if(value > highest){
            highest = value;
        }
        if(value < lowest){
            lowest = value;
        }
    }
    return new double[] {highest, lowest};
}

```

Figure 5.2-17 Finding the Max and Min values

The method in **Figure 5.2-17** shows how the max and min values were found. An enhanced for-loop goes through the values of an extracted Rep and determines the max and min values.

```

public static List<Rep> normalizeReps(List<Rep> reps) {
    List<Rep> normalizedReps = new ArrayList<Rep>();
    List<Double> samples, normalizedSamples;
    Rep normalizedRep;
    double high, low;

    for(Rep rep : reps) {
        samples = rep.getSamples();
        normalizedSamples = new ArrayList<Double>();
        double[] highAndLowValues = getHighAndLowValues(samples);
        high = highAndLowValues[0];
        low = highAndLowValues[1];
        for(double sample : samples) {
            double normalized = (sample - low) / (high - low);
            normalizedSamples.add(normalized);
        }
        normalizedRep = new Rep(rep.getId(), rep.getStartPointIndex(), rep.getStartPointValue(),
                                rep.getEndPointIndex(), rep.getEndPointValue(), normalizedSamples);
        normalizedReps.add(normalizedRep);
    }
    return normalizedReps;
}

```

Figure 5.2-18 Creating a List of Normalized Reps

A method was used to create a List of normalized Reps. It first took in the List of originally created Reps and returned a normalized version of it. Once the high and low values are taken for the particular rep, the inner enhanced for-loop normalizes each value using the

equation previously mentioned. A new Rep is created once the normalization is complete and added to the list.

The final step in the Pre-Processing stage was to adjust the Rep size to exactly 200 values each. For example, if a Rep had only 150 samples, 50 samples were added evenly across the list. In this example, a value would be added after every 3 samples ($150 / 50$). The value added would be the average of the one before and after it. Where a Rep would be 250 samples, 50 samples would be removed from the list. In this instance, every fifth sample would be removed ($250 / 5$).

```
public static List<Double> adjustUpwardsTo(int size, List<Double> originalList) {  
    List<Double> adjustedList = new ArrayList<Double>();  
    int originalSize = originalList.size();  
    int adjustment = size - originalSize;  
    int frequency = originalSize/adjustment;  
    if(frequency == 0) {  
        return originalList;  
    }  
    int frequencyNext = 0;  
    double value = 0;  
  
    while(originalList.size() < size) {  
        frequencyNext += frequency;  
        if(frequencyNext >= originalList.size()) {  
            frequencyNext = originalList.size() - 1;  
        }  
        value = (originalList.get(frequencyNext - 1) + originalList.get(frequencyNext)) / 2;  
        originalList.add(frequencyNext, value);  
    }  
    return originalList;  
}
```

Figure 5.2-19 Adjusting a list up to 200

The method in **Figure 5.2-19** shows how a list smaller than 200 samples would be adjusted up to 200. The original list being the normalized rep of original size. The frequency represents how often a value is added (i.e. every 3 samples for a size of 150 as explained above). The *value* variable is the average of the samples directly before and after it. The while loop keeps looping while the original size has not yet reached the required size of 200.

```

public static List<Double> adjustDownTo(int size, List<Double> originalList) {
    int originalSize = originalList.size();
    int adjustment = originalSize - size;
    int frequency = originalSize / adjustment;
    int frequencyNext = originalList.size() - 1;

    while(originalList.size() > (size + 1)) {
        frequencyNext -= frequency;
        if(frequencyNext > 0) {
            originalList.remove(frequencyNext);
        }
    }
    if(originalList.size() > size) {
        originalList.remove(size - 1);
    }
    return originalList;
}

```

Figure 5.2-20 Adjusting a list down to 200

The method to adjust down was a little less complex as values were not added. It removes values while the original list is over 200. For calculating the frequencyNext, -= was used instead of a += because otherwise the frequencyNext variable would get larger as the list became smaller, causing an out of bounds error. If-Conditions were included to prevent other errors such as trying to remove a negative index value from the list. Some trial and error was needed involving changing the directions of the loop and the amount of times of iteration. Sometimes the loop would stop at 201 which the final If-condition would remove.

```

public static List<Double> adjustTo(int size, List<Double> originalList) {
    List<Double> adjustedList;
    if(originalList.size() > size) {
        adjustedList = adjustDownTo(size, originalList);
    }
    else if(originalList.size() < size) {
        adjustedList = adjustUpwardsTo(size, originalList);
    }
    else {
        return originalList;
    }
    return adjustedList;
}

```

Figure 5.2-20 Adjust List to given size

The method in **Figure 5.2-20** decides if the List needs to be increased or decreased.

```

public ProcessedExercise preprocessRawExerciseForANN(ExerciseRaw exerciseRaw) {

    List<Double> averages = setAverages(exerciseRaw.getSensorSampleList());
    List<Integer> maximas = PeakDetect.discoverMaximas(averages, 25, 100);

    maximas = PeakDetect.filterOutFlatPeaks(averages, maximas, 100, 0.20);
    maximas = PeakDetect.filterOuterRangePeaks(averages, maximas, 0.30);

    List<Rep> reps = RepCreator.createRepList(averages, maximas);
    List<Rep> normalizedReps = RepCreator.normalizeReps(reps);

    for(Rep rep : normalizedReps) {
        List<Double> adjustedSize = AdjustArraySize.adjustTo(200, rep.getSamples());
        rep.setSamples(adjustedSize);
    }

    ProcessedExercise processedExercise = new ProcessedExercise(exerciseRaw.getUsername(),
        exerciseRaw.getExerciseName(), exerciseRaw.getDate(), exerciseRaw.getSensorSampleList(),
        averages, maximas, reps, normalizedReps, "not_sent_to_ann");

    return processedExercise;
}

```

Figure 5.2-21 The Pre-Processing Handler using all the static methods

To method in **Figure 5.2-21** summarizes everything in this sprint into one single method. The method takes in a Raw Data object and calls all the methods discussed in this sprint. First it gets the average values, then discovers the maxima, then filters these maxima, creates the reps, normalizes the reps, and then adjustd the size of the reps to 200. A List is created for the Reps and then for the normalized Reps which is all added to a new ProcessedExercise object. Original data is also taken from the Raw object and transferred to the Processed one.

```

@Document
public class ExerciseRaw {

    @Id
    private int id;
    private String type;
    private String username;
    private String exerciseName;
    private double weight;
    private int repCount;
    private Date date;
    private List<SensorSample> sensorSampleList;

    public class ProcessedExercise {

        private int id;
        private String username;
        private String exerciseName;
        private Date date;
        private double weight;
        private List<SensorSample> rawSensorSamples;
        private List<Double> averages;
        private List<Integer> maximas;
        private List<Rep> extractedReps;
        private List<Rep> normalisedReps;
        private String exerciseDetected;
    }
}

```

Figure 5.2-22 Snippets from the classes shows the extra attributes in ProcessedExercise

3081	1.42E+12	-1.61848	-1.27372	12.07635	3.061386
3082	1.42E+12	-0.91937	-2.15478	12.22958	3.051809
3083	1.42E+12	0.871489	-2.87304	11.3964	3.131616
3084	1.42E+12	1.666364	-2.49955	9.318233	2.82835
3085	1.42E+12	2.442086	-2.2697	8.724471	2.965618
3086					
3087					
3088					

Figure 5.2-23 Raw data and average data

To step through the results in Excel, **Figure 5.2-23** shows the last five lines in the original raw data. The 16 reps come to a total of 3085 samples. The first column is the time stamp, the following three are X, Y and Z while the final one is the average value of the axis values.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
175	-7.25922	-6.62077	-6.10362	-5.85781	-6.21854	-7.1028	-6.93361	-6.04935	-6.0302	-6.99745	-7.05811	-6.15789	-6.28558	-6.71653	-7.07726
176	-7.32945	-6.78038	-6.26004	-5.89612	-6.36219	-7.0613	-7.11876	-5.9855	-6.13873	-6.98149	-7.0613	-6.23131	-6.41327	-6.76123	-7.13153
177	-7.37733	-6.96234	-6.33665	-5.84504	-6.38454	-6.96234	-7.24007	-5.9887	-6.14193	-6.99426	-6.98788	-6.3175	-6.46754	-6.84103	-7.20814
178	-7.4316	-7.1028	-6.43561	-5.93762	-6.39411	-7.01341	-7.35499	-5.9855	-6.10043	-7.0613	-6.95276	-6.29196	-6.52819	-6.87296	-7.17941
179	-7.48587	-7.1826	-6.50584	-5.99189	-6.37177	-7.09322	-7.4316	-6.01743	-6.06212	-7.13153	-7.06449	-6.29515	-6.58565	-6.89211	-6.9783
180	-7.49864	-7.19537	-6.58246	-5.96316	-6.41008	-7.09961	-7.43479	-6.06531	-5.9472	-7.2241	-7.1858	-6.32708	-6.68461	-6.9368	-7.04853
181		-7.2656	-6.71015	-5.95039	-6.49627	-7.19218		-6.17066	-5.97274		-7.18899	-6.42285	-6.63353	-6.96234	-7.15387
182		-7.33903	-6.81549	-6.01743	-6.58246	-7.33583		-6.30473	-6.08127			-6.51223	-6.61119	-7.04215	-7.27518
183		-7.35818	-6.76442	-5.97593	-6.62077	-7.46672		-6.36219	-6.18342			-6.53457	-6.691	-7.06768	-7.34222
184		-7.44756	-6.79634	-5.88016	-6.6878			-6.38773	-6.24408			-6.55692	-6.76761	-6.99426	-7.30391
185		-7.44437	-7.05491	-5.8642	-6.73888			-6.33665	-6.37177			-6.63034	-6.80273	-6.99107	-7.36456
186		-7.40606	-7.26241	-5.77481	-6.75803			-6.29515	-6.5633			-6.7293	-6.80911	-7.09003	-7.46672
187		-7.44118	-7.3518	-5.70778	-6.76761			-6.33027	-6.6878			-6.76761	-6.76123	-7.08364	-7.54014
188		-7.58483		-5.85781	-6.77719			-6.47711	-6.81869			-6.91126	-6.76123	-7.09003	-7.61995
189		-7.62633		-6.04935	-6.81549			-6.62715	-6.96872			-6.96553	-6.75803	-7.09003	-7.70614
190				-6.23131	-6.90169			-6.65907	-7.11238			-6.98149	-6.76761	-7.03895	-7.73487
191				-6.28239	-6.91126			-6.70057	-7.27837			-7.01661	-6.78357	-6.88253	-7.7636
192				-6.33665	-6.92403			-6.79315	-7.29114			-7.01661	-6.85061	-6.8123	-7.71252
193				-6.41008	-7.00384			-6.90488	-7.42841			-7.07088	-6.93042	-6.99426	-7.67741
194				-6.43881	-7.15068			-6.92722	-7.61675			-7.02938	-7.00384	-7.15387	-7.72848
195				-6.53138	-7.28795			-6.9783	-7.77637			-7.01661	-7.05491		-7.85937
196				-6.55692	-7.19857			-7.09003	-7.90406			-7.0613	-7.09641		-7.94237
197				-6.54734	-7.14749			-7.19857	-7.89129			-6.98788	-7.11238		-8.05409
198				-6.55054	-7.10599			-7.27199	-7.91683			-6.94318	-7.1028		
199				-6.58565	-7.15068			-7.31987	-7.96152			-7.03895	-7.12514		
200				-6.62715	-7.2241			-7.35818	-8.11794			-7.14111	-7.16984		
201				-6.68461	-7.11557			-7.41564	-8.2999			-7.14749	-7.24964		
202				-6.8123	-6.99426			-7.46352					-7.28476		
203				-6.88892	-7.09961										
204				-6.8953	-7.17303										
205				-6.91126	-7.25922										
206				-7.09641	-7.53375										
207				-7.25922	-7.74444										
208				-7.28157											
209				-7.28476											
210															
211															
212															

Figure 5.2-24 Extracted Reps of different size

The Extracted Reps worksheet shows from line 175 downwards. This data shows Reps were in the range of 180 to 209 values. Since a sample is created at about one every 10ms, each Rep took between about 1.8 seconds to 2.1 seconds. Since the columns go up to O, it shows that 15 Reps were correctly extracted. All values visible are of negative values since each rep is coming to an end (de-acceleration).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
178	0.083912	0.106266	0.13585	0.125303	0.083459	0.130769	0.112045	0.118339	0.184028	0.144488	0.073844	0.079989	0.060698	0.04716	0.07335
179	0.082883	0.101002	0.137391	0.138478	0.079425	0.127949	0.104405	0.11442	0.187233	0.139874	0.069245	0.079704	0.052106	0.047681	0.067259
180	0.08314	0.099749	0.138161	0.12584	0.077912	0.118205	0.096766	0.105799	0.196848	0.128946	0.068163	0.076857	0.056541	0.046378	0.069543
181	0.082754	0.103008	0.135336	0.109707	0.077156	0.111538	0.092055	0.094828	0.194712	0.12154	0.067622	0.068318	0.058481	0.042731	0.085533
182	0.082368	0.102256	0.136877	0.09438	0.076399	0.103846	0.087344	0.090125	0.18563	0.114133	0.064918	0.060347	0.051552	0.036217	0.079949
183	0.082625	0.093484	0.142912	0.090078	0.073374	0.099231	0.085052	0.088036	0.177083	0.10442	0.074114	0.058355	0.0449	0.033611	0.071574
184	0.079794	0.086717	0.148947	0.085507	0.066566	0.091538	0.08276	0.092215	0.172009	0.096892	0.080335	0.056362	0.041851	0.032048	0.061929
185	0.070785	0.078947	0.149204	0.079322	0.065809	0.086923	0.080978	0.095611	0.161325	0.098349	0.074926	0.049815	0.041297	0.0284	0.056599
186	0.064865	0.066416	0.140216	0.076902	0.064801	0.078718	0.081487	0.092738	0.145299	0.097377	0.071409	0.040991	0.045455	0.026316	0.059645
187	0.059459	0.05213	0.132768	0.069105	0.058497	0.073333	0.072065	0.080721	0.134882	0.096892	0.070598	0.037575	0.045455	0.019802	0.054822
188	0.052252	0.046617	0.122496	0.066953	0.046899	0.06641	0.061115	0.068443	0.123932	0.089121	0.067081	0.024765	0.045732	0.017718	0.046701
189	0.041699	0.041103	0.109913	0.06776	0.036056	0.065897	0.055258	0.065831	0.111378	0.087421	0.055721	0.019926	0.0449	0.02371	0.040863
190	0.039125	0.034837	0.103749	0.067491	0.043116	0.064615	0.051184	0.062435	0.099359	0.087421	0.045442	0.018503	0.043514	0.023971	0.034518
191	0.038867	0.033835	0.095788	0.064533	0.047151	0.058205	0.042271	0.054859	0.08547	0.092763	0.036246	0.015371	0.037694	0.015894	0.027665
192	0.034492	0.028321	0.090139	0.061038	0.050429	0.051538	0.035905	0.045716	0.084402	0.095435	0.026508	0.015371	0.030765	0.016415	0.025381
193	0.028314	0.022556	0.083975	0.056198	0.046899	0.054872	0.039216	0.043887	0.072917	0.101506	0.017852	0.010532	0.02439	0.016154	0.023096
194	0.024196	0.021053	0.073703	0.045442	0.041099	0.062821	0.044563	0.039707	0.057158	0.101748	0.014065	0.014233	0.019956	0.015894	0.027157
195	0.019305	0.014035	0.065229	0.038989	0.049672	0.058718	0.042526	0.030564	0.043803	0.099077	0.013795	0.015371	0.016353	0.015894	0.029949
196	0.013642	0.014286	0.069337	0.038451	0.059254	0.052308	0.027757	0.021682	0.03312	0.100291	0.020016	0.011386	0.014967	0.020063	0.027919
197	0.009781	0.017293	0.066769	0.037107	0.050933	0.051795	0.01808	0.015674	0.034188	0.09932	0.022992	0.017933	0.015798	0.03283	0.025888
198	0.005405	0.014536	0.045968	0.021511	0.045134	0.044359	0.008913	0.011755	0.032051	0.09422	0.013524	0.021919	0.013858	0.038562	0.015482
199	0.00103	0.003258	0.029276	0.007798	0.038326	0.032821	0.002801	0.008621	0.028312	0.088878	0.003246	0.013379	0.009978	0.02371	0.008883
200	0	0	0.022085	0.005647	0	0.022308	0.002546	0	0	0.081836	0.002975	0.003701	0	0.010683	0
201															
202															

Figure 5.2-25 Normalized and adjusted reps

The screenshot in **Figure 5.2-25** looks at the end of the Normalized Reps worksheet. Now all data values are between 1 and 0 and every rep has exactly 200 values each.

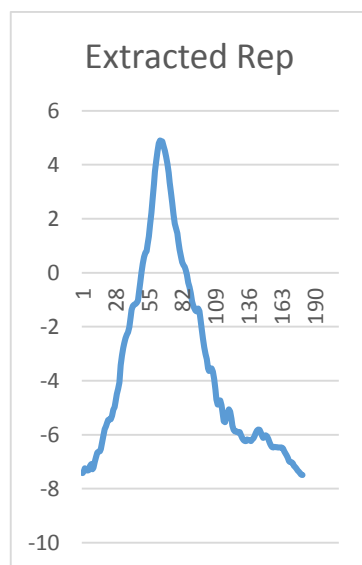


Figure 5.2-26 Narrow Range

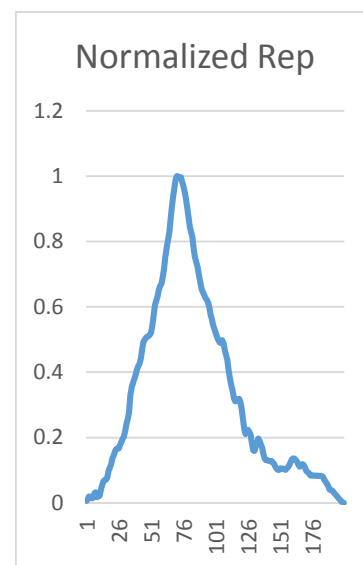


Figure 5.2-27 Wide Range

Figure 5.2-26 and **Figure 5.2-267** shows the difference between an original Rep and a Normalized rep. The original values went from about -7 to 5 with 180 values while the Normalized one is between 0 and 1 with 200 values. These diagrams also importantly show that the normalization had very little effect in the pattern change of the rep, despite the adjusting of the array size. Both patterns are almost identical which means the normalized rep data will enter the Neural Network representing the correct patterns.

5.3 Sprint 3: Artificial Neural Network

This sprint involved testing the performance of the neural network using the Neuroph Java library. However, full configuration of the network was not completed in this particular sprint. Just enough of testing was done so the project could move forward with the confidence that Neuroph was a suitable choice for machine learning. To test the library, some examples were done in Neuroph Studio but this took a lot of time since this GUI application froze many times and many restarts were needed. Importing the JAR file into an Eclipse project had the opposite effect, where everything ran smoothly without any issues.

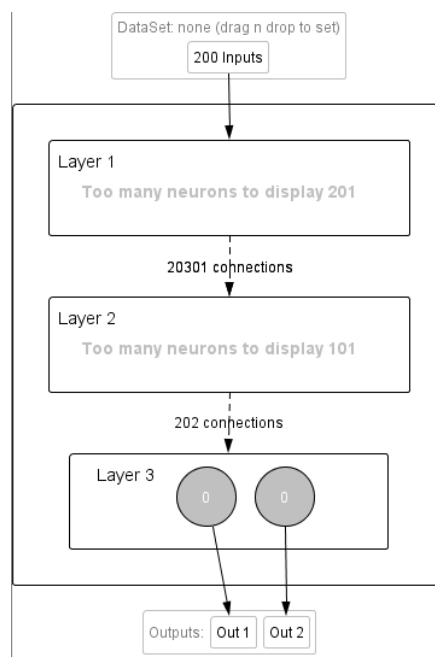


Figure 5.3-1 Diagram of a Neural Network trial in Neuroph Studio with 200 inputs

To become more familiar with Neuroph and the classes it provided, some examples from the Neuroph website were studied and practised. A classification example of animal species was imported into Eclipse as classification training would also be needed for recognizing the exercise patterns. Using this example, code was changed to suit the exercise data for a first trial in Eclipse.

```
MultilayerPerceptron neuralNet = new MultilayerPerceptron(TransferFunctionType.SIGMOID, 200, 50, 2);  
MomentumBackpropagation learningRule = (MomentumBackpropagation) neuralNet.getLearningRule();  
learningRule.setLearningRate(0.5);  
learningRule.setMomentum(0.7);  
learningRule.setMaxIterations(1000);
```

Figure 5.3-2 Creating a neural network object and setting parameters

Neuroph library provided a full range of classes and methods needed to implement the network. A TrainingImport utility classes provided a method that took in a filename for the data set, along with integer values to set the number of inputs and outputs. These integer values were needed so the file could be read properly. A CSV was created manually for now, by just pasting in the normalized rows of the CSV files from the previous sprints. Two Rep examples were taken from three different exercises which totalled 60 rows with 200 values each. At the end of each row, a two digit binary value was appended to give a binary representation of each exercise. This gave each row a total of 202 values.

As shown in **Figure 5.3-2**, the number of perceptrons were also added to the MultiLayerPerceptron class, including the number in the hidden layer. A transfer function type of Sigmoid and a learning rule of Back Propagation with Momentum was used. A learning rate and momentum rate was given and experimented with and a max iteration was also included to prevent the network from iterating for too long if the training wasn't going well.

As discovered during research, there is no set rule in how to find the best configuration for a neural network. One first has to estimate a starting point and then adjust and test the settings according to the test results. Neuroph's examples were a good source in finding a starting point, especially the classification examples. Some adjustments were made to the code, and different exercises were tired out.

```
for (DataSetRow trainingElement : dset.getRows()) {  
    nnet.setInput(trainingElement.getInput());  
    nnet.calculate();  
    double[] networkOutput = nnet.getOutput();  
    float first = (float)Math.round(networkOutput[0]);  
    float second = (float)Math.round(networkOutput[1]);  
  
    System.out.println("Output: " + first + " , " + second);  
}
```

Figure 5.3-3 Retrieving and outputting test results

The loop in **Figure 5.3-3**, gets the output from every row in the test dataset and calculates the result. The two output values which are in an array of double, are then taken and printed to the console for analysis. A screenshot of the sample of the output is shown in the following screenshot.

```
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 0.0
Output: 0.0 , 0.0
Output: 0.0 , 0.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
Output: 0.0 , 1.0
```

Figure 5.3-4 Output binary values of neural network

With different network settings tested, the resulting outputs varied. After some trials, it became clear that different exercises gave different outputs. This was essentially the desired effect for a network to be able to classify the different movements. With each type of exercise resulting in a specific binary value, exercises could now be distinguished from each other. However, it was difficult to find a particular setting that would correctly detect many exercises. Testing a bicep curl against a lat-raise almost always brought back a high accurate result, although bringing a third and fourth exercise in to the network resulted in a drop in accuracy. Sometimes most values were incorrect but with some tweaking accuracy increased improved again.

Overall, the results were very satisfactory as this sprint was aimed at just briefly testing the Neuroph Framework. Many other changes could be made to the network including a change in input and output values, learning rules and the amount of hidden layers. To allow for extensive testing, a large dataset would be required and to create a large dataset.

The next stage in the project was to develop a more advanced Android application compared to the one used in Sprint 1. Also to develop the back-end to take in many exercises and store them in the database. Also a front-end application would assist in the analysis of the datasets which would be more efficient than the current practise of browsing through excel files.

For now, the project could progress with the confidence that Neuroph could classify the processed data from the Android sensors. With more analysis and testing with a larger dataset, the network could be further improved to a more desired standard.

5.4 Sprint 4: Android

In Sprint 1, the Android functionality was implemented which included getting raw data from the sensors. For the final application, three screens were included to demonstrate Exercise detection and feedback analysis. A logo and colour was added and a number of inputs, labels and buttons were included so the user could add specific data.

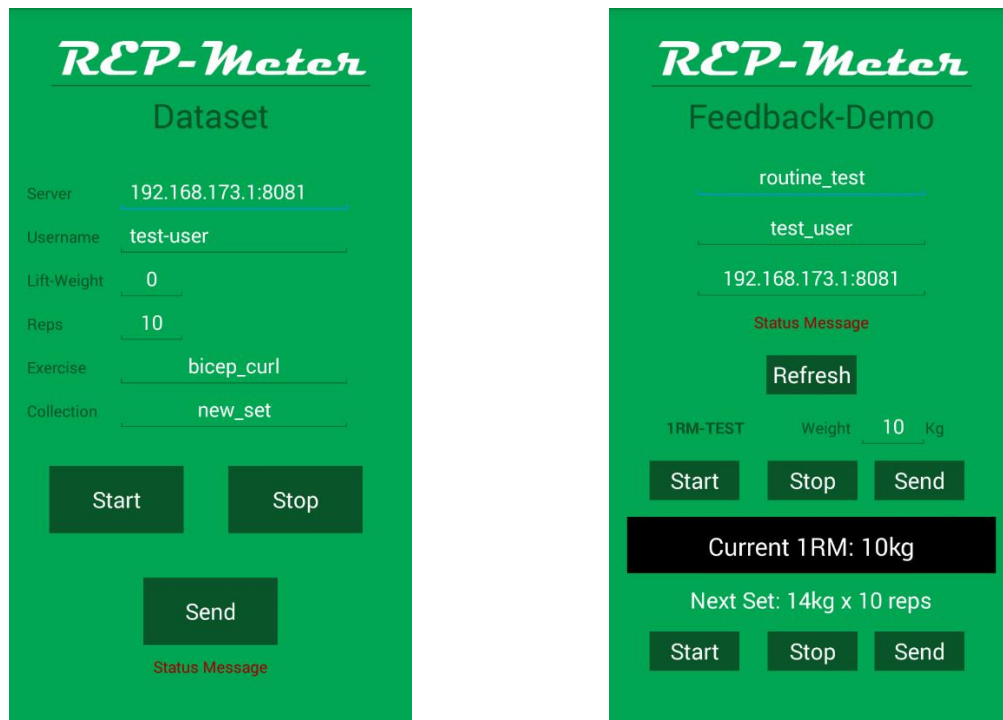


Figure 5.4-1 Android Screen shots

The Android screenshots in **Figure 5.4-1** shows a screen activity to add an exercise to a dataset in Mongo via the Java server application. The other activity is used to do a 1RM test and get feedback on what weights should be done on the next exercise.

```
<TextView
    android:id="@+id/lblDataset_username"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/lblDataset_server"
    android:layout_marginTop="20dp"
    android:textColor="#095529"
    android:text="Username" />
```

Figure 5.4.2 XML layout

As shown in **Figure 5.4.2**, most of the layout was done with XML. Eclipse had a graphic interface to help in the screen design but better accuracy of aligning the components was achieved using XML code.

```
case R.id.btnDemoExDetect:
    intent = new Intent(this, DetectActivity.class);
    startActivity(intent);
    break;
```

Figure 5.4.3 Android Intent

Screen (Activity) navigation was achieved using the Intent class to call the requested Activity according to which button was pressed.

All other functionality in the Android application was similar to what was implemented in Sprint 1, where raw data was taken from the sensors and sent to the server. The response was set as a String and this String was set to Text labels in the Android Activity for the user to see.

5.5 Sprint 5: Analysis (Front-End)

To demonstrate the raw data and the pre-processing, a front-end website was included in the Java Spring project. This also became a useful tool to further understand and analyse how the data was being processed. Instead of having to browse directories to open an Excel file, Exercises could be added from the Android Dataset Activity, then sent to the server, saved to the database and then the results could be viewed on the webpage.

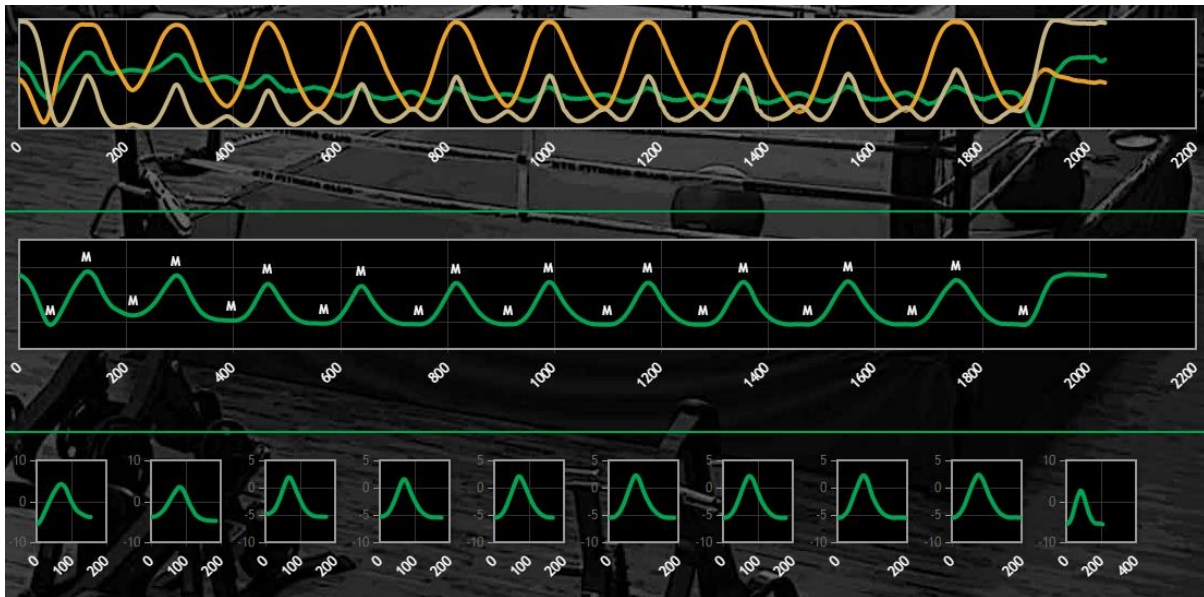


Figure 5.5-1 JQ Plot - Raw data, averaged data and extracted reps

As shown in **Figure 5.5-1**, raw data and the resulting processing could be viewed with just a few clicks immediately after executing an exercise on the Android device.

```
<script type="text/javascript" src="resources/js/jqueryFile.js"></script>
<script type="text/javascript" src="resources/js/jquery.jqplot.min.js"></script>
<script type="text/javascript" src="resources/js/jqplot.barRenderer.min.js"></script>
<script type="text/javascript" src="resources/js/jqplot.canvasTextRenderer.min.js"></script>
<script type="text/javascript" src="resources/js/jqplot.canvasAxisTickRenderer.min.js"></script>
<script type="text/javascript" src="resources/js/jqplot.categoryAxisRenderer.min.js"></script>
<script type="text/javascript" src="resources/js/jqplot.pointLabels.min.js"></script>
<script type="text/javascript" src="resources/js/analysis.js"></script>
<link rel="stylesheet" type="text/css" href="resources/css/jquery.jqplot.css"/>
<link rel="stylesheet" type="text/css" href="resources/css/analysis.css"/>
```

Figure 5.5-2 Loading Scripts and Stylesheets

JQ Plot, an open-source JQuery library was used to generate the graphs. As illustrated in **Figure 5.5-2**, JavaScript files including JQ Plot library files were added to the HTML page in addition to CSS stylesheets.

```

function getCollectionList() {
    $.ajax({
        url: 'rest/data/collectionnames',
        type: 'GET',
        success: function(collectionList) {
            displayCollectionList(collectionList);
        },
        error: function(xhr, status, error) {
            var err = eval("(" + xhr.responseText + ")");
            alert(err.Message);
        }
    });
}

```

Figure 5.5-3 AJAX call using a GET method

Figure 5.5-3 demonstrates one of many JQuery AJAX calls that was used to communicate with the back-end REST methods. Here a list of collection names are requested which will be retrieved from the Mongo FYP database. Once the asynchronous function gets the response, it lists the names by calling on the displayCollectionList function.

```

for(var item in setsList) {
    htmlString +=
        '<div class="collect_set_list_item">' +
        '<div class="collect_set_list_item_id">' + setsList[item].id + '</div>' +
        '<div class="collect_set_list_item_name">' + setsList[item].exerciseName + '</div>' +
        '<div class="collect_set_list_item_reps">' + setsList[item].repCount + '</div>' +
        '</div>';
}
$('#collect_set_list').html(htmlString);

```

Figure 5.5.4 Rendering HTML with JavaScript/JQuery

A typical example of how data from the back-end was rendered into HTML is shown in **Figure 5.5-4**. Using JavaScript's enhanced For-Loop, the htmlString appends a Div for every item found in the list. Data is included from the 'setsList' with [item] representing the loop and list index. Once the loop has completed and the htmlString has appended all the data, the JQuery html() function sets the string to the appropriate DOM object. In this case it's the "#collect_set_list" object.

```

$('#control_panel_header img').click(function(e){
    e.stopPropagation();
    $('#control_panel').animate({
        right: '-500'
    });
});

```

Figure 5.5-5 JQuery Click function and animation

JQuery was used to add much of the functionality in the webpage including clickable items, opening and closing menus and adding and removing data. **Figure 5.5-5** shows JQuery's click function which was used for all the buttons and clickable items. In this example, the page's main control animates left by 500 pixels. This allows the control panel, which is hidden away to the right of the screen, to slide in left for the user to view.

```
function plotGraphWithLabelPoints(averages, labelPoints) {
    $('#graph_area_average_chart').html('');

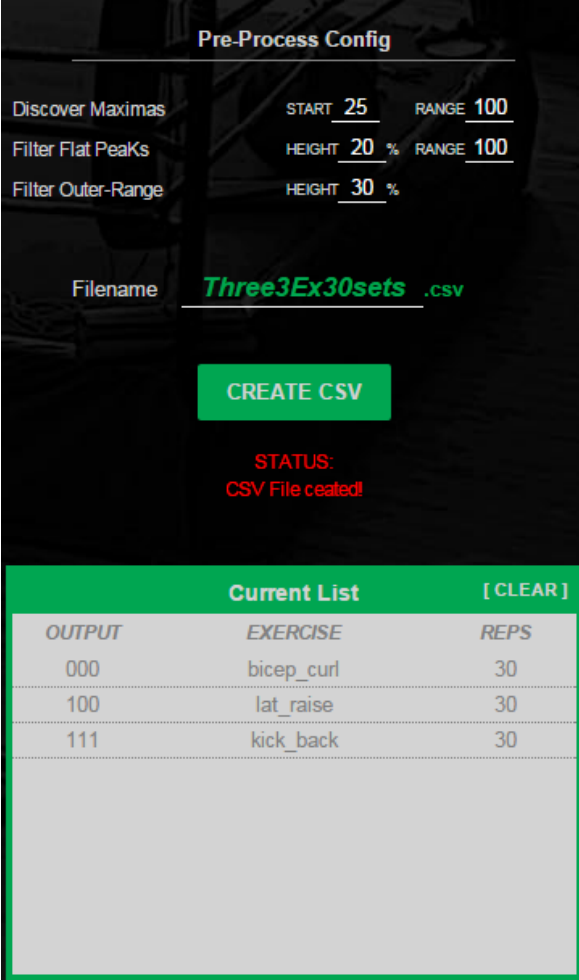
    $.jqplot('graph_area_average_chart', [averages], {
        title: false,
        animate: false,
        series: [{
            smooth:true,
            lineWidth: 4,
            color: '#00a651',
            showMarker:false,
            pointLabels: {
                show: true,
                labels: labelPoints
            },
        }],
        axes: {
            xaxis: {
                tickRenderer: $.jqplot.CanvasAxisTickRenderer,
                tickOptions: {
                    angle:-45,
                    fontSize:'10pt',
                    textColor: '#ffffff'
                },
                pad:0,
            },
            yaxis: {
                pad:1.5,
                showTicks:false
            }
        },
        grid: {
            gridLineColor:'#333333',
            background:'#000'
        }
    });
}
```

Figure 5.5-6 Plotting a graph with label points

The main feature in this Analysis webpage was viewing data using JQ Plot charts. Once the AJAX calls got the data from the back-end, functions could be called by passing in the data that the JQ Plot could render as a chart. In this example the averages array is added in with an array of integers representing maxima and minima positioning for label points. Various configuration options are available including tick options for showing the values, background grids, label points and axis colours.

5.6 Sprint 6: Creating CSV files

Though CSV files were created earlier in the project, this sprint allowed for CSV files to be created from the front-end. The functionality also allowed for more flexibility where data from different exercises could be added to a custom csv file. This was done by calling on the back-end methods, which used the datasets stored in Mongo.



Pre-Process Config

Discover Maximas START 25 RANGE 100

Filter Flat Peaks HEIGHT 20 % RANGE 100

Filter Outer-Range HEIGHT 30 %

Filename Three3Ex30sets.csv

CREATE CSV

STATUS:
CSV File ceated!

Current List			[CLEAR]
OUTPUT	EXERCISE	REPS	
000	bicep_curl	30	
100	lat_raise	30	
111	kick_back	30	

Figure 5.6-1 Saving a CSV file

As shown in **Figure 5.6-1**, the ANN webpage allows for certain configuration options for creating a CSV file. Other parts of the page include menus displaying exercises from the database where the user can add to a list. As the above screenshot shows, the list displays the binary output value and name for each type of exercise added. The rep count for each exercise in this example is 30 and this represents a total of each set that has been added. For example, the user added three sets of bicep curls that had 10 reps each and the list shows the total number of reps for this type of exercise that will be added to the CSV file. Once the user has finished configuration and presses the Create CSV button, the data is

added to a JavaScript object and is posted to the server using an AJAX POST request. Once the exercise and configuration data has been received by server, the pre-processing of the exercises is carried out and this results in a List of ProcessedExercise. For every exercise in this List, the normalized data is appended to a String.

```
for(ProcessedExercise processedExercise : preProcessedData) {
    normalizedReps = processedExercise.getNormalisedReps();
    amountOfReps = normalizedReps.size();
    rowOfValues = "";
    for(Rep rep : normalizedReps) {
        rowOfValues = "";
        for(double value : rep.getSamples()) {
            rowOfValues += value + ",";
        }
        output = getOutputValue(processedExercise.getExerciseName());
        for(int digit = 0 ; digit < output.length() ; digit ++ ) {
            digitAsString = Character.toString(output.charAt(digit));
            rowOfValues += digitAsString;
            if(digit < (output.length()-1)) {
                rowOfValues += ",";
            }
        }
        rowOfValues += "\n";
        writer.append(rowOfValues);
    }
}
```

Figure 5.6-2 Appending the Processed Data to a CSV file

The for-loop in **Figure 5.6-2** shows how processed data was appended to a String. While the outer loop iterates through all the exercises, the loop inside this, loops through every Rep that is contained in each exercise. Every Rep should contain the same amount of double values. For every double value in the Rep, the value along with a comma is added to the line. Once the output value is retrieved (i.e. 100 for a bicep_curl), a for-loop is used to loop through each output digit so these can also be appended to the end of the double values. Using Java's FileWriter, each row is appended to a file with extension ".csv".

The example in **Figure 5.6-1** had 30 reps of three different exercises. This is a total of 90 reps. Therefore, this example created a CSV file that contained 90 lines of data with the output value appended to the end of each (as required to use with Neuroph). If a rep is normalized to 200 samples each, and the binary output is a three-digit value, each row contains 203 values separated by a comma.

5.7 Sprint 7: Front-End and Testing the Neural Network

Once CSV files were created on the server, the Neural Network could be trained according to the exercises listed on the selected CSV file. Two dropdown menus were created to allow for selection of a CSV file for training the network, and another for testing the network.

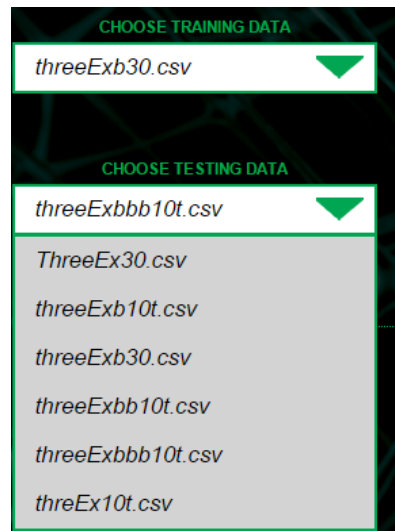


Figure 5.7-1 Choosing training and testing data

Drop down menus were created using JQuery's slide-down function while click events and menu items were included in the same manner as previous sprints. Configuration settings for the network was also included in the web page with options to set numbers of perceptrons in each layer. The learning rate and momentum options were also included. This was useful for testing the network in different scenarios. An option to set the maximum number of iterations during the learning stage was also set. Setting a maximum of iterations forced the network to stop training at a certain point which helped in certain scenarios where the learning rate took too long if the data was not suitable.

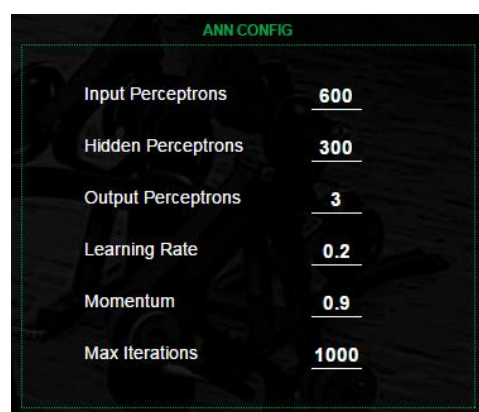


Figure 5.7-2 Options for Neural Network Configuration

Once the data and network options are selected and the user clicks the button, the back-end application trains the Network as explained in Sprint 3.

```
$.ajax({
  url: 'rest/ann/train',
  type: 'POST',
  data: JSON.stringify(annConfigData),
  contentType: 'application/json; charset=utf-8',
  dataType: 'json',
  success: function(response) {
    $('#ann_status_message').text('Network is trained');
    $('#ann_status_message').css('color', '#00a651');
    $('#train_button').show();
    $('#results_button').show();
    renderResults(response);
  },
  error: function(xhr, status, error) {
    var err = eval("(" + xhr.responseText + ")");
    alert(err.Message);
  }
});
```

Figure 5.7-3 AJAX Post function for training the network

The AJAX function in **Figure 5.7-3** uses a REST call to train the network. The configuration data is posted in an object. When the network is trained, the status message and colour is changed and the results of the tests are returned and rendered. As show in the following screenshot, the results are colour-coded by the success result of the detected in the results with a percentage of the success also available.

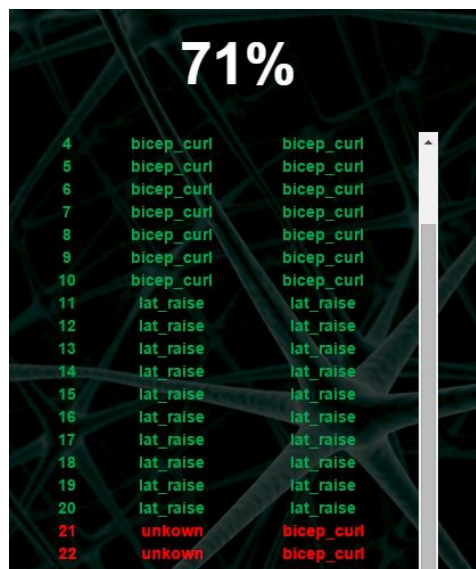


Figure 5.7-4 The list of tests results after the network has been trained

Colours were set using if-conditions while rendering the HTML similarly to previous sprints. Successful matches were counted in the loop and used to calculate the percentage.

```
var percentage = 100 / results.length * matches;
```

This webpage became a very useful for testing different configurations in the Neural Network. It was after this was completed that much of the additional testing and changing of the network settings were done.

After another review of the examples on the Neuroph website, a network was created within the project using the pre-processed data of difference exercises. Similar to the very first tests, a three layer network was created with 200 perceptrons at the input layer and just two in the output layer. The two outputs were to output binary values for the exercise detected (e.g. bicep curl = 00, lat-raise = 01, kick-back = 10, etc). A back propagation rule was used with momentum of 0.9 and a learning rate of 0.2. For the hidden layer in the middle, 100 perceptrons were used, which is half of the output size. These figures were based on recommended settings in the Neuroph documentations. However these are just starting figures and the values were changed many times in various ways to test the different outcomes.

The tests didn't seem to go as well compared to those in Sprint 3 with some tests recognizing the bicep curl better, while other settings distinguished between the other exercises better. Again, it was challenging to find a common setting that would allow a result to suit all scenarios.

To find a better solution, two major changes were made. The first change made was in how the data was pre-processed. On reviewing the averaged data, the reps from the different exercises were similar.

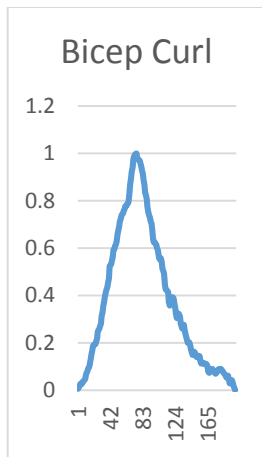


Figure 5.3-1 Averaged Bicep Curl

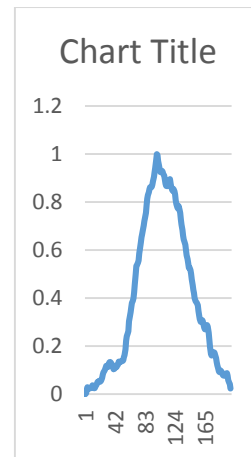


Figure 5.3-2 Averaged Lat-Raise

Figure 5.3-1 and **Figure 5.3-1** shows two different exercises after being averaged and normalized but the difference between them is not a lot considering their original three axis values were very different. In fact, it seemed that averaging the data took away some of the key characteristics that each exercise had.

To allow for a better distinction of values, it was decided to include all three axis values. First the average data was used to detect the peaks as explained in the Pre-Processing sprint. However, this time when the start/finish points were found, all three axis values were extracted and appended to each other to form one shape. These shapes turned out unusual but much more unique, even after normalizing their values.

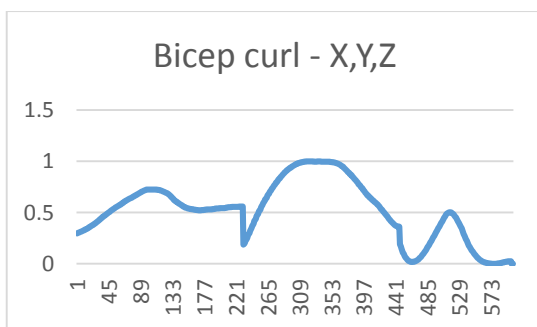


Figure 5.3-3 3-Axis Bicep Curl

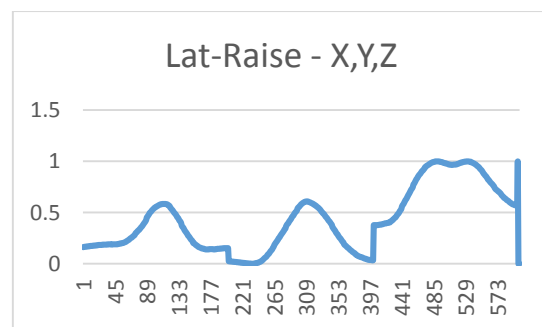


Figure 5.3-4 3-Axis Lat-Raise

Figure 5.3-3 and **Figure 5.3-4** show that the patterns from the two exercises are now very different. To the human eye, the graphs don't make a lot of sense, but for a machine the patterns are much more distinct from each other. The same position points were extracted but instead of extracting an average of the three into 200 values, all three axis values were extracted at the same locations points, then appended onto each other, resulting in a 600 value result.

Now that the Pre-Processed was changed, the next major change was adjusting the configuration of the Neural Network. The network now had to have 600 inputs perceptrons to take in the exercise values. For the hidden layer, 300 perceptrons were used and similar values were taken for the learning rate and momentum. The biggest change was made to the output layer. A third perceptron was added and the binary values for each exercise were set as far apart from each other as possible. A bicep curl was given a 000 (0) value and a triceps kick-back was given a 111 (7) value. The lat-raise was given a 100 (4) which is about middle of the other two values.

With the exercises a lot more distinct from each other and the output values being further apart, the results were dramatically improved. With only 30 samples of each, training time was under 10 seconds and tests were 100% accurate as all three exercises were distinguished from each other correctly. Rep counts were also highly accurate which were calculated during the pre-processing stage.

```
private static Rep create3AxisRep(List<SensorSample> rawData, int start, int end) {
    List<Double> points = new ArrayList<Double>();

    for(int point = start ; point < end ; point ++){
        points.add(rawData.get(point).getX());
    }
    for(int point = start ; point < end ; point ++){
        points.add(rawData.get(point).getY());
    }
    for(int point = start ; point < end ; point ++){
        points.add(rawData.get(point).getZ());
    }
    return new Rep(0, start, 0, end, 0, points);
}
```

Figure 5.3-5 Create 3 Axis Rep Data

To create the Rep data of 3 axis, a new method was used that took in the raw data with the start and end points which were calculated from the Maxima and Minima methods. Three For-Loops were used one after another for each axis, adding to the new List which was returned as show in in **Figure 5.2-5**.

```
neuralNet = new MultiLayerPerceptron(TransferFunctionType.SIGMOID, inputCount, hiddenCount, outputCount);
trainingSet = getDataSet(trainingFileName, inputCount, outputCount);
testingSet= getDataSet(testingFileName, inputCount, outputCount);
MomentumBackpropagation learningRule = getLearningRule(learningRate, momentum, maxIterations);
```

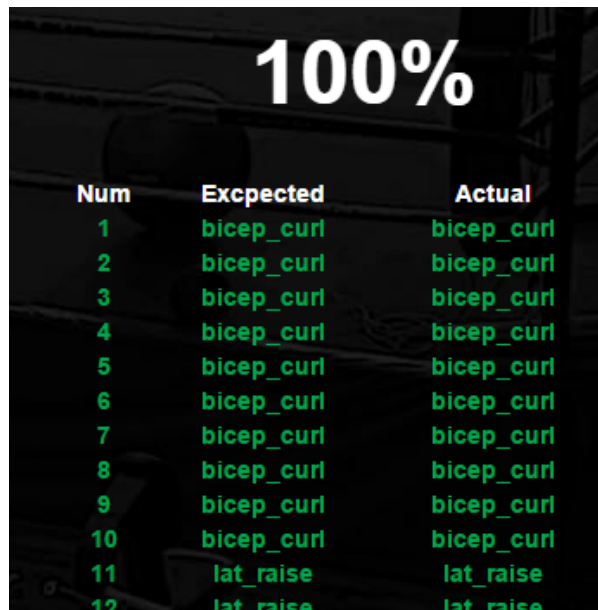
Figure 5.3-6 Instantiating the Neural Network

In **Figure 5.2-6**, the objects needed to create the Neural Network is shown. Instead of hardcoding the parameters, variables were used instead to allow for flexibility when testing out which settings would work best. The dataset took in a CSV file and these files were also created using a custom class.

```
public String recognizeExercise(DataSet dataset) {  
    List<String> exercisesRecognized = new ArrayList<String>();  
    NeuralNetwork testNet = neuralNet;  
    int row = 0;  
    for(DataSetRow dataSetRow : dataset.getRows()) {  
        testNet.setInput(dataSetRow.getInput());  
        testNet.calculate();  
        double[] networkOutput = testNet.getOutput();  
  
        int actualFirst = (int) Math.round(networkOutput[0]);  
        int actualSecond = (int) Math.round(networkOutput[1]);  
        int actualThird = (int) Math.round(networkOutput[2]);  
        exercisesRecognized.add(getExercise(actualFirst, actualSecond, actualThird));  
    }  
    return getMode(exercisesRecognized);  
}
```

Figure 5.3-7 Recognizing the Exercise

In **Figure 5.2-7** the method for recognizing an exercises is shown. A Dataset will contain a number of DataSetRows depending on how many reps were executed. The getExercise method uses the binary number results to match to get an Exercise match. Once a set of reps are detected, the getMode method counts each detected rep and picks out the most likely. This means in a situation where 10 bicep curls are executed and one or two are wrongly detected, there are still eight correct ones which represents the mode. This means an accuracy of just 60% would be needed for correct detection of a set. However, with the recent pre-processing and network changes, almost all rep detections were being detected accurately.



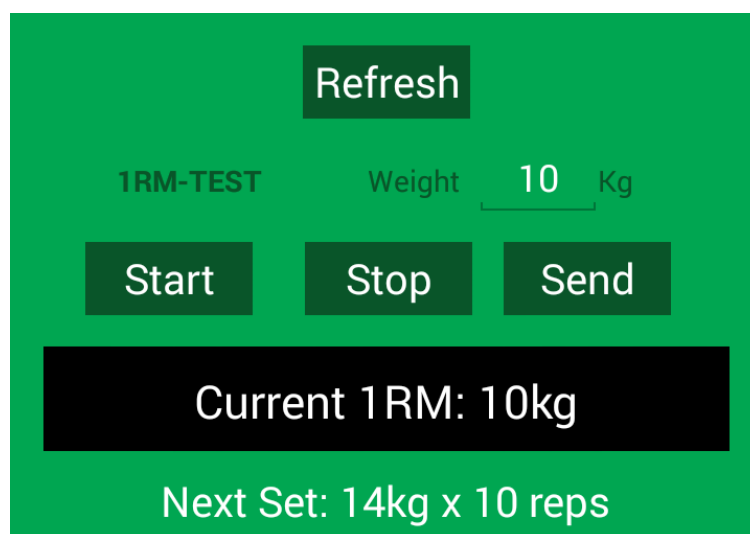
100%

Num	Expected	Actual
1	bicep_curl	bicep_curl
2	bicep_curl	bicep_curl
3	bicep_curl	bicep_curl
4	bicep_curl	bicep_curl
5	bicep_curl	bicep_curl
6	bicep_curl	bicep_curl
7	bicep_curl	bicep_curl
8	bicep_curl	bicep_curl
9	bicep_curl	bicep_curl
10	bicep_curl	bicep_curl
11	lat_raise	lat_raise
12	lat_raise	lat_raise

Figure 5.3-7 Screenshot of the 100% accuracy with just 30 reps of each Exercise

5.8 Sprint 8: Decision Making / Feedback

This sprint involved implementing the decision making so the application could advise the user of the weight to lift according to their progress. This decision making was based on the Brzycki Formula which was discussed during the research phase. With this formula the user's one-rep-max (1RM) could be calculated using the weight amount lifted and the number of reps executed until failure.



Refresh

1RM-TEST Weight Kg

Start Stop Send

Current 1RM: 10kg

Next Set: 14kg x 10 reps

Figure 5.8-1 Screenshot of the Android decision-making activity

The user first have to enter the weight they are lifting into the Android device and then execute as many reps as possible. Each rep-time can be calculated, including the last one where the user struggles to lift the last one. The data is sent to the server where the exercise is detected, reps are counted and the times and weight are used to calculate the user's 1RM.

Many type of exercises require a certain amount of repetitions. For example, gaining strength would require low reps while hypertrophy would require a higher amount. The decision making based on the Bryzcki Formula would work for any number of reps within a set. For demonstration purposes, a 10-rep set was the presumed requirement for the user's training. When the 1RM was calculated, the weight needed for a 10-rep set would be suggested to the user. If the user successfully executed 10 reps with the suggested new weight, and within the failure time, the next suggested weight would increase by a further 10%. However, if the weight was too demanding, which would force the user to execute slower than the failure rate at a certain point within the set, the 1RM would be reduced according to the weight and number of reps executed.

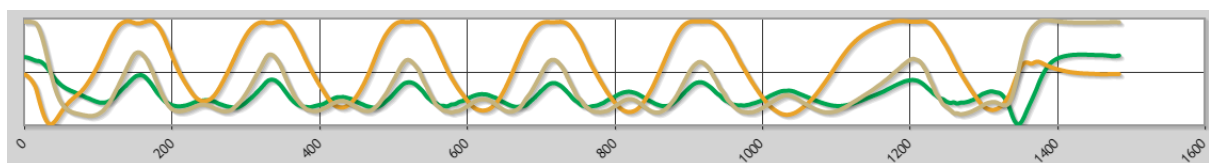


Figure 5.8-2 A graph showing six reps

Three sets were executed to test this decision-making method. A weight of 25kg was entered into the phone. Six reps of bicep curls were executed, with the last one much slower to replicate a struggle with the weight. Six reps of biceps curls were successfully detected, and a time of 2890ms was calculated for the last rep. Using the formula, a 1RM was calculated and rounded as 29kg and a suggested weight of 22kg for the next set of 10 reps. After execution of a set of 10 reps, the 1RM was increased to 30kg and the next suggested weight was given at 25kg.

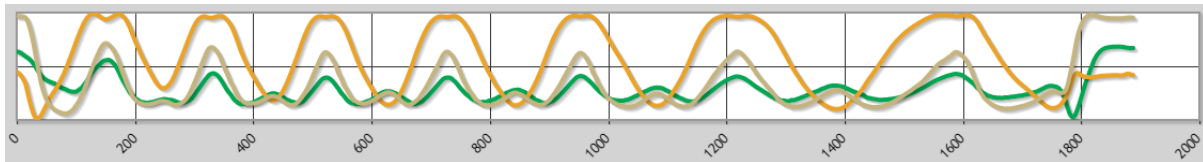


Figure 5.8-3 A graph showing the user fails to keep with the 1RM speed

A third set was executed using the current suggested weight which is now 25kg. This time on the sixth rep, the user struggled and even more so on the seventh where the user gives up. The seventh rep took almost four seconds which is much slower the original failure speed of 2890ms. This would suggest the user struggled too much to execute the final one. The Brzycki Formula is re-calculated using the weight lifted with the number of successful of reps which is now lower than 10. The resulting suggested weight to lift is back to 22kg and the 1RM has decreased also to 29kg.

With the application having the ability to calculate the times of execution, comparing them to the original failure rate and calculating accordingly, suggestions can now be given to the user on what weight to lift based on their performance. Without any intervention from the user, suggested weights can be calculated based on a formula where the user's 1RM is also calculated after each set. Instead of guessing and chancing new weights, the user can be more confident of the right weight and weekly progress which is based on the 1RM formula.

```
public int get1RM(int numOfReps, double weightLifted) {
    double oneRM = weightLifted / (1.0278 - (0.0278 * numOfReps));
    return (int) Math.round(oneRM);
}
```

Figure 5.8-4 Implementation of the Brzycki Formula

The method in **Figure 5.8-4** shows the implementation of the Brzycki Formula. It takes in the number of reps and the weight lifted and finally returns the rounded value of the 1RM weight. The reason for rounding the value is that most weights in a gym will be of an integer kilogram value so numbers with long decimals would not be of much value to users.

```

public Routine createNewRoutine(ProcessedExercise processedExercise, String collection) {
    int time = getTimeLengthOfLastRep(processedExercise);
    int oneRM = get1RM(processedExercise.getExtractedReps().size(), processedExercise.getWeight());
    int nextWeight = (int) Math.round(oneRM * 0.75);
    return new Routine(processedExercise.getUsername(), time, oneRM, nextWeight, processedExercise);
}

```

Figure 5.8-5 Creating a Routine

The decision making data was marshalled into a Routine object for transferring between the Android device, the server and database. This object stored the time of the failed rep, the 1RM weight and the next suggested weight. In **Figure 5.8-5**, the method for creating a new Routine is shown. The time of the last (to failure) rep got using a method that subtracts the timestamp of the penultimate rep from the timestamp of the last rep. After the 1RM is calculated, the next suggested weight multiplied by 75%. During research it was discovered that 75% of the 1RM was the suggested weight for lifting 10 reps.

Another method for updated the Routine after each rep was similar, except if a Rep was detected being too slow, or if the user failed to execute 10 reps, than the new suggested weight would be based on the user's lower performance. Otherwise, the weight was increased by 10%.

```

private int findFirstSlowerRep(List<Integer> repTimes, int oneRMtime) {
    for(int rep = 0 ; rep < repTimes.size() ; rep ++ ) {
        if(repTimes.get(rep) > oneRMtime) {
            return rep;
        }
    }
    return -1;
}

```

Figure 5.8-6 Detecting a slower rep

The method for updating a Routine shown in **Figure 5.8-6** checks if the user struggled too much with a weight. For example, if on the sixth rep, the user executed the rep too slow, that would mean that the user only executed five reps within the proper speed. The next suggested weight would be based on current weight using five reps. To check if a rep is too slow, the loop above iterates until a rep that took more time than the 1RM time is found. Then it can return the index of the rep before that which would have been the last properly executed rep. If no slower reps are found -1 is returned

Chapter 6: Findings, Analysis and Conclusion

6.1 Introduction

This project set out to create an Android application for use in strength conditioning exercises. All the requirements of the 'Must Have' list from the MoSCoW method were successfully completed. Requirements of lesser priority were not implemented so that all time and effort could focus on the features that would answer the research question;

Can an Android smart phone be independently used, to detect, track and analyse a user's strength-training activities, and also provide live and accurate feedback?

6.2 Sensor Capabilities

The answer on the project question depended on the capability of the sensors in the Android device. Though many issues arose during research which mainly focused on accuracy and noise, the sensors successfully produced smooth data, which produced very different patterns for each exercise tested. The Android's TYPE_GRAVITY logical sensor on the Samsung Galaxy S3 was used, which implements the device's accelerometer, gyroscope and barometer. The sensor's firmware algorithms based on the Kalman filter ensured that sensor noise was not an issue. It has been three years since the release of this particular Android device and as this can be seen as a long time in mobile technology, one can expect even better performance in today's devices and also into the future. This clearly shows that extra wearable devices should not be required by the user for resistance-training exercises.

6.3 Pre-Processing of Data

Peak detection techniques were used to extract reps from each exercise set. Though these techniques were successful on the exercises tested, different exercises may require different or more complex peak detection techniques. The tested exercises produced distinct mountains and valleys in the graph data where maxima and minima were successfully detected. If different exercises were to produce a more complex data pattern, detecting the correct maxima and minima may prove more challenging. However this would not be seen as a major barrier in the extraction of reps since there are many other peak detection techniques that could be tried. Re-sizing the rep data arrays to suit the neural network input layer did not affect the data pattern or shape. Normalization of the data to the range from 0-1 was also straight-forward and similar to the re-sizing where little or no change was seen on the pattern of the data after the pre-processing activity.

Transformation of data initially involved taking the data from all three axis and averaging them into one data set. However this took away some key characteristics of the data patterns of each exercise type and so the average patterns were too similar for the neural network to make a distinction. The method of including all three axis data, appended on to each other to form one large set of 600 samples allowed for very unique patterns.

6.4 Artificial Neural Network

The neural network proved very successful where movements were easily detected with just small amounts of training data. As little as 10 reps were used and then tested against another 30 reps which gave a 100% success rate. Improvements in the pre-processing stage was the deciding factor in the network performance as initial tests were not so successful. Another major improvement was the addition of extra neurons on the output layer. Extra neurons meant that the output binary values were larger and the exercise values could be spread further apart. For examples, a bicep curl could be 000 which is 0, and a different exercise could be 111 which is 7. The likely consequence of having just two outputs neurons where exercise binary values were just one value away from each other was that more demand was on the network to calculate the bias weights more precisely. For future developments, where a larger amount of exercises would be tested, a large number of output neurons would be recommended to obtain the current accuracy levels.

The recommended and most commonly used learning rules were used in the network. The Backpropagation rule with a learning rate of 0.2 and a momentum of 0.9 were initially used. The number of neurons in the hidden layer amounted to half of that in the input layer. Making small changes to these settings initially produced different results depending on the exercise used. However, once the pre-processing was improved and the network started to produce highly accurate results, further changes in the network settings had little impact.

6.5 Decision Making

To provide instant feedback to the user, the server application had to make decisions based on the raw data it received. The peak detection techniques along with the rep extraction proved very useful for this stage. This allowed for reps to be counted and with a timestamp included in each data sample, the timing of each rep could also be determined. With the start, end and timing of each rep known, it was possible to apply decision making for velocity-based training exercises.

Using a one-rep-max formula, the application successfully calculated a user's one-rep-max and provided suggestions to what weight to lift for a 10 rep set. According to the user's timing of rep execution, the application successfully made recommendations on what weight to lift for the user's next set.

For future development, it would be interesting to test if the sensors could provide accurate positional information. If the exact positions were known, then the distance between the start and end of a rep could be calculated which would allow for calculation of the speed (m/s) of each rep also. With the speed of the reps known, even more analysis could be carried out for the user, as opposed to just using time alone.

6.6 Wearing the Samsung Galaxy S3

It is common to see people in the gym with phones strapped onto their arms. There was no hindrance or discomfort experienced in wearing the device during the execution of the tested exercises. However for people who may find this uncomfortable, a smaller phone would be recommended. The ultimate experience would be to test the exercises using an Android smartwatch. The performance of a smart watch may not be an issue since the client device just needs to get the raw data and send it to the server. A smart watch such as the Galaxy Gear S, which provides the all needed sensors with a dual core processor, including 3G and Wi-Fi connectivity, would make a smartphone app a viable option for future development of this project.

6.7 Further Recommendations

The Android device read the sensor data every 10ms which amounted to an average of about 200 samples per rep. The high number of samples may have been a huge benefit in creating such unique patterns for each exercise. However, reading in fewer samples may still allow for accurate detection and increase efficiency in performance for both the client and server. If a fraction of the data was only needed, a whole gym session may be recorded in less than 1MB. Instead of starting and stopping the sensors at the start/end of each set, the user could start the app and leave the sensors reading continuously for the whole gym session without the worry of creating too much data. Sensor and data efficiency is an area where improvements could be made for this application.

6.8 Conclusion

Working on this project provided a lot of very interesting and complex programming challenges. Despite the challenges and complexity, the research question was answered with very positive results. It was interesting to see how the Android sensors provided such smooth and noise-free data. The pre-processing of the data was the most challenging area of the project and the majority of the time was put into this area. To successfully pre-process the data, much time was spent in statistical analysis. The high efficiency and accuracy of the neural network required small amounts of training data which was unexpected as it was foreseen that large amounts would be needed.

The results of this project bring confidence that the development of this application can be taken much further. With so many exercises and training methods practised in the gym, along with continuous advancements in mobile technology, there are numerous ways in which the technologies implemented in this project can help and advance the user's progress.

Summary of Recommendations for Future Development

- Test the current peak detection algorithm against various other exercises
- Consider an unsupervised neural network rule for extracting reps from a set
- Allow for a high number of neurons in the output layer of the neural network
- Test application using less sensor samples, to allow for more efficiency
- Test sensors for accuracy in position and distance so speed can be detected
- Test functionality on an Android smart watch such as the Samsung Galaxy Gear S

References

- Altheris, 2014. *What is a gyroscope?*. [Online]
Available at: <http://www.altheris.com/products/what-is-a-gyroscope.htm>
[Accessed 8 October 2014].
- Android, 2014. *Position Sensors*. [Online]
Available at: http://developer.android.com/guide/topics/sensors/sensors_position.html
[Accessed 16 October 2014].
- Android, 2014. *Sensor*. [Online]
Available at: <http://developer.android.com/reference/android/hardware/Sensor.html>
[Accessed 20 October 2014].
- Android, 2014. *Sensors Overview*. [Online]
Available at: http://developer.android.com/guide/topics/sensors/sensors_overview.html
[Accessed 20 October 2014].
- Anon., 2014. *Accelerometer & Gyro Tutorial*. [Online]
Available at: <http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/step1/The-Accelerometer/>
[Accessed 8 October 2014].
- beckmw, 2013. *Visualizing neural networks from the nnet package*. [Online]
Available at: <http://www.r-bloggers.com/visualizing-neural-networks-from-the-nnet-package/>
[Accessed 23 November 2014].
- Boundless, 2014. *Position, Displacement, Velocity, and Acceleration as Vectors*. [Online]
Available at: <https://www.boundless.com/physics/textbooks/boundless-physics-textbook/two-dimensional-kinematics-3/vectors-41/position-displacement-velocity-and-acceleration-as-vectors-225-5215/>
[Accessed 25 October 2014].
- Bros, M., 2014. *Introduction to Neural Networks*. [Online]
Available at: <http://www.neuralnetworksolutions.com/nn/intro2.php>
[Accessed 1 November 2014].
- Buckley, P., 2011. *Next-generation iNEMO Engine offers new level of motion-sensing applications on smart consumer devices*. [Online]
Available at: http://www.analog-eetimes.com/en/next-generation-inemo-engine-offers-new-level-of-motion-sensing-applications-on-smart-consumer-devices.html?cmp_id=7&news_id=222901769&page=0
[Accessed 18 October 2014].
- Butt, C., 2001. *The WeighTrainer*. [Online]
Available at: <http://www.weightrainer.net/training/coefficients.html>
[Accessed 26 September 2014].
- Chipworks, 2012. *Inside the Samsung Galaxy SIII*. [Online]
Available at: <http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-samsung-galaxy-siii/>
[Accessed 16 October 2014].

Dixon-Warren, S., 2014. *STMicroelectronics' Integrated Inertial Sensor MEMS Device Technology*. [Online]

Available at: <http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/stmicroelectronics-integrated-inertial-sensor-mems-device-technology/> [Accessed 16 October 2014].

Electronics, F., 2014. *What is an Accelerometer*. [Online]

Available at: <http://www.futureelectronics.com/en/sensors/accelerometers.aspx> [Accessed 8 October 2014].

Elmenreich, W., 2014. *An Introduction to Sensor Fusion*. [Online]

Available at: https://www.academia.edu/649942/An_introduction_to_sensor_fusion [Accessed 16 October 2014].

Etzkorn, B., 2011. *Data Normalization and Standardization*. [Online]

Available at: <http://www.benetz Korn.com/2011/11/data-normalization-and-standardization/>

Faragher, R., 2012. *Understanding the Basis of the Kalman Filter*. [Online]

Available at:

<http://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf>

[Accessed 16 October 2014].

Fritsch, S., 2012. *Package 'neuralnet'*. [Online]

Available at: <http://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf> [Accessed 23 November 2014].

Gershenson, C., 2014. *Artificial Neural Networks for Beginners*. [Online]

Available at: [https://datajobs.com/data-science-repo/Neural-Net-\[Carlos-Gershenson\].pdf](https://datajobs.com/data-science-repo/Neural-Net-[Carlos-Gershenson].pdf) [Accessed 19 November 2014].

Heaton, J., 2014. *Heaton Research*. [Online]

Available at: <http://www.heatonresearch.com/download> [Accessed 23 November 2014].

HyperPhysics, 2014. *Gryoscope*. [Online]

Available at: <http://hyperphysics.phy-astr.gsu.edu/hbase/gyr.html> [Accessed 8 October 2014].

Istook, E., 2015. *IMPROVED BACKPROPAGATION LEARNING IN NEURAL NETWORKS WITH*. [Online]

Available at:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.7105&rep=rep1&type=pdf> [Accessed 20 November 2015].

Johnson, C., 2014. *Inertial measurement unit integrates accelerometer, gyro*. [Online]

Available at: http://www.eetimes.com/document.asp?doc_id=1261819 [Accessed 10 October 2014].

Jovanović, M., 2014. *Set and Rep Schemes in Strength Training*. [Online]

Available at: <http://articles.elitefts.com/training-articles/set-and-rep-schemes-in-strength-training-part-1/> [Accessed 28 September 2014].

- Kasei, A., 2004. *AK8975 3-axis electronic compass IC*. [Online]
Available at: <http://www.akm.com/akm/en/product/datasheet1/?partno=AK8975>
[Accessed 16 October 2014].
- Kaurov, V., 2014. *Simple, fast compiled peak detection based on moving average*. [Online]
Available at: <http://community.wolfram.com/groups/-/m/t/96823>
[Accessed 25 November 2014].
- Kriesel, D., 2005. *A Brief Introduction to Neural Networks*. [Online]
Available at: <http://www.dkriesel.com/media/science/neuronaleetze-en-zeta2-2col-dkrieselcom.pdf>
[Accessed 15 November 2014].
- Levi, J., 2011. *What Can You Do With a Barometer on a Smartphone?*. [Online]
Available at: <http://pocketnow.com/android/what-can-you-do-with-a-barometer-on-a-smartphone>
[Accessed 8 October 2014].
- Lucas, J., 2014. *Newton's Laws of Motion*. [Online]
Available at: <http://www.livescience.com/46558-laws-of-motion.html>
[Accessed 25 October 2014].
- Mann, B., 2014. *Hot Topic Power and Bar Velocity Measuring Devices*. [Online]
Available at: <https://www.nsca.com/Education/Articles/Hot-Topic-Power-and-Bar-Velocity-Measuring-Devices/>
[Accessed 1 October 2014].
- Materko, W., 2007. *Scielo.br*. [Online]
Available at: http://www.scielo.br/pdf/rbme/v13n1/en_07.pdf
[Accessed 26 September 2014].
- Mathas, C., 2012. *Senser Fusion: The Basics*. [Online]
Available at: <http://www.digikey.com/en/articles/techzone/2012/apr/sensor-fusion-the-basics>
[Accessed 17 October 2014].
- Mendelsohn, L., 2014. *Preprocessing Data For Neural Networks*. [Online]
Available at: <http://www.tradertech.com/neural-networks/preprocessing-data>
[Accessed 24 November 2014].
- Microelectronics, S., 2012. *LSM330DLC iNEMO inertial module: 3D accelerometer and 3D gyroscope*. [Online]
Available at: <http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00037200.pdf>
[Accessed 16 October 2014].
- Museum, C. S. a. T., 2014. *Background Information for Science Surprises*. [Online]
Available at: http://www.sciencetech.technomuses.ca/english/schoolzone/Info_Science.cfm#5
[Accessed 8 October 2014].
- MW, B., 2013. *Visualizing neural networks from the nnet package*. [Online]
Available at: <http://www.r-bloggers.com/visualizing-neural-networks-from-the-nnet-package/>
[Accessed 23 November 2014].

- Neuroph, 2014. *NueroPh v2.9*. [Online]
Available at: <http://neuroph.sourceforge.net/>
[Accessed 23 November 2014].
- Nielson, M. A., 2014. *Nueral Networks*. s.l.:Determination Press.
- Omega, 2014. *What are Accelerometers*. [Online]
Available at: <http://www.omega.com/prodinfo/accelerometers.html>
[Accessed 8 October 2015].
- Sailing, M. P., 2014. *Velocitek Shift*. [Online]
Available at: <http://www.mauriprosailing.com/us/tech/Velocitek/2-Velocitek-Shift.html>
- Samsung, 2014. *Samsung Galaxy S3*. [Online]
Available at: <http://www.samsung.com/global/galaxys3/specifications.html>
[Accessed 8 October 2014].
- Sayad, D. S., 2014. *Artificial Neural Network*. [Online]
Available at: http://www.saedsayad.com/artificial_neural_network.htm
[Accessed 20 November 2014].
- Showa, 2007. *Applicaitions of 6-Component force sensor*. [Online]
Available at: http://www.showa-sokki.co.jp/english/technical_note_e/6_bnyoku_cell_ap_e.html
[Accessed 16 October 2014].
- Specifier, E., 2013. *Micro Motion for Medical Devices*. [Online]
Available at: <http://www.electronicsspecifier.com/sensors/es-design-magazine-analog-devices-adi-micro-motion-for-medical-devices>
- ST, 2014. *iNEMOEngine_PAAPPAAP engine PRO - Android Platform*. [Online]
Available at: <http://www.st.com/web/catalog/tools/FM147/CL1818/SC1528/PF252504>
[Accessed 17 October 2014].
- Stanford, 2000. *Biological Neurons*. [Online]
Available at: <http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Biology/index.html>
[Accessed 1 November 2014].
- Stone, M. H., 2000. *Training Principles: Evaluation of Modes*. [Online]
Available at: <http://www.arturmonteiro.com.br/wp-content/uploads/2014/04/Artigo-1.9.pdf>
[Accessed 25 September 2014].
- Tatton-Brown, M., 2012. *Accelerometers are the most under-appreciated technological innovation*. [Online]
Available at: <http://www.wired.co.uk/news/archive/2012-09/06/accelerometers-are-awesome>
[Accessed 8 October 2014].
- Turner, G., 2014. *Uses for gyroscopes*. [Online]
Available at: <http://www.gyroscopes.org/uses.asp>
[Accessed 8 October 2014].
- Tuthill, M., 2014. *Classic Muscle Strength Programs Reviewed*. [Online]
Available at: <http://www.muscleandfitness.com/workouts/workout-tips/classic-muscle-strength->

programs-reviewed

[Accessed 2 October 2014].

Weil, R., 2014. *Resistance Training*. [Online]

Available at: http://www.emedicinehealth.com/strength_training/article_em.htm

[Accessed 27 September 2014].

Weka, 2014. *Weka 3: Data Mining Software in Java*. [Online]

Available at: <http://www.cs.waikato.ac.nz/ml/weka/>

[Accessed 23 November 2014].

Whuber, 2012. *How do I find peaks in a dataset?*. [Online]

Available at: <http://stats.stackexchange.com/questions/36309/how-do-i-find-peaks-in-a-dataset>

[Accessed 25 November 2014].