



NUI Galway
OÉ Gaillimh

4BP – 2018/19

**B.E. Electronic and Computing
Engineering**

Football Heading Training Platform

Ronan Murphy - 15397831

April 2019

Supervisor: Mr. Liam Kilmartin

Co-Assessor: Dr. John Breslin

ABSTRACT

In today's world of highly competitive sport the issue of concussion has become a major topic. Heretofore little was known about the short-term or long-term effects of concussion on the brain and on players health and wellbeing. The early onset of brain conditions such as Parkinson's, Alzheimer's and Dementia of some well know sport heroes has raised the profile of the issue. The famous world champion boxer, Muhammad Ali, being one such example. This is leading to the requirement to be able to measure and assess the effects of high and repetitive impacts in sport. Games such as Boxing, Rugby, American football and Ice Hockey have taken serious precautionary measures to reduce the risk of players having clinical health issues later in life. The Head Injury Assessment (HIA) and Return To Play (RTP) after concussion in Rugby being a good example.

In recent years, there also have been concerns in regard to the sub-concussive effects, these are impacts that are below the concussion threshold but not severe enough to see immediate symptoms. Many studies have indicated that repetitive head impacts, from heading the ball in soccer, can potentially have long-term neurological effects.

The objective of this project is to develop a wearable sensory system, to detect the impact forces the player receives when heading a football and to save this data for analysis. This platform is enhanced by creating an active tool which feeds back information collected and analyses it. The header will be rated on a scale and the discrepancies between good and bad headers will be obtained from patterns on the graphs. The player or physician can use the data obtained, together with other clinical data, to see if the heading event or the accumulation of multiple heading events could possibly have some damaging effect to the player's brain.

ACKNOWLEDGMENTS

Firstly, I would like to sincerely thank my mentor Mr. Liam Kilmartin for his great advice and support throughout this project. Also, my co-assessor Dr. John Breslin and the lab technicians, Mr. Myles Meehan and Mr. Martin Burke, all of whom assisted me without hesitation and helped me to complete my project. Finally, I would like to thank my family and friends for their optimism, consultation and encouragement during this project.

I declare that this thesis is my original work except where stated:

Date...../...../..... Signature.....

TABLE OF CONTENTS

ABSTRACT	ii
ACKNOWLEDGMENTS	iii
TABLE OF CONTENTS	iv
GLOSSARY	vi
1 INTRODUCTION.....	1
2 BACKGROUND RESEARCH	2
3 COMPONENTS	3
3.1 Sensors	3
3.2 Microcontroller	4
3.3 Bluetooth VS Bluetooth Low-Energy (BLE)	6
3.4 Timing	6
4 PROJECT STRUCTURE	8
4.1 Circuit	8
4.2 FSR Operation	9
4.3 Accelerometer Operation	10
4.4 Data to Microcontroller	11
4.5 Bluetooth Connection	12
4.6 Bluetooth Transmission	14
5 ANDROID APPLICATION.....	16
5.1 Android vs IOS.....	16
5.2 Android Studios	16
5.3 Heading Ball Application	21
5.4 Scanning Activity	22
5.5 BLE Adapter Service	25
5.6 Handling Data	29
5.7 Graphing Activity	32
6 MACHINE LEARNING	35
6.1 The Theory	35
6.2 Implementation	36
6.3 Preparing data.....	38
6.4 Designing Algorithm	40
6.5 Results and Output	43

7 SECURING CIRCUITRY	45
7.1 Overview	45
7.2 FSR – Force-Sensing Resistors	45
7.3 Strip Board Assembly	46
7.4 Microprocessor Placement	47
8 TESTING	49
8.1 Accelerometer calibration test	49
8.2 FSR Test	49
8.3 Initial System Testing	49
9 STUDY AND RESULTS	51
9.1 Accelerometer Orientation	51
9.2 The Heading Event	52
9.3 Conclusions from Graph	55
9.4 Overall System Results	61
10 CONCLUSION	62
BROADER SOCIETAL IMPACT	63
REFERENCES	64
APPENDICES	66
A.1 Parts List	66
A.2 Code Cheat Sheet	67
HEALTH AND SAFETY	68
RISK ASSESSMENT FOR SOLDERING IN	68
SOP FOR SOLDERING IN LABORATORY	69
RISK ASSESSMENT FOR ATTACHING FOOTBALL	71
SOP FOR ATTACHING FOOTBALL	72

GLOSSARY

FSR	Force-Sensing Resistor
ADC	Analogue Digital Convertor
DAC	Digital Analogue Convertor
BLE	Bluetooth Low Energy
I/O	Input/ Output
GND	Ground
A0	Analogue pin zero
UART	Universal Asynchronous Receiver/Transmitter
GAP	General Access Profile
GATT	Generic Attribute Profile
UUID	Universal Unique Identifier
RTC	Real-Time Clock
RAM	Random-Access Memory
ATT	Attribute Protocol
CPU	Central Processing Unit
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
HCI	Host Control Interface
FIFO	First-in First-out
RX	Receiving Pin
TX	Transmission Pin
OHA	Open Handset Alliance
SDK	Software Development Kit
APK	Android Package
VM	Virtual Machine
OS	Operating System
API	Application Program Interface
GDPR	General Data Protection Regulations
GUI	Graphical User Interface
Dip	Density-Independent Pixels
CSV	Comma-Separated Values
PCB	Printed-Circuit Board
RNN	Recurrent-Neural Network
LSTM	Long Short Term Memory

MAIN TEXT

1 INTRODUCTION

The sport of soccer is the only sport in which players purposely use their head to gain advantage in a game. As soccer is the world's most popular and played sport with over 265 million people worldwide [1] of all ages playing, it is important to analyse these sub-concussive events, to test whether there is a possible danger that repetitively heading the ball could be detrimental to one's health. Since heading is such a key part of the game the negative effects of this, if any, could have serious consequences worldwide. The muscles involved in the movement to head the ball include the neck and head muscles contracting together in preparation for the contact. The force of a header can vary vastly due to the speed at which the ball is going and how fast the player contracted their muscles. The position the ball lands on the head can also be checked to determine if certain places are more harmful than others.

The main aim for the project was to detect the impact and axial forces the neck, head and ball during a header and to transmit this information to be recorded and analysed. This would give the player an indication of the negative effects on the brain, based on comparisons with other related injuries and their results.

To measure the force of impact, a device called a force-sensing resistor (FSR) can be used. These sensors decrease in resistance as more force is applied, hence it can be stated that the increase in voltage is proportional to the increase in force. To implement this, three square FSR's will be placed around the forehead and the temples. A head mask is needed to cover the head and hold the devices in place, as it is easily removable and offers a cushion layer, so the sensors wouldn't hurt the player when testing. Three accelerometers will be used to measure the acceleration forces in the x, y and z planes, one on the ball, head and neck. Once an event is recorded and the data saved, it can be graphed to see the movement of the axial forces. Multiple headers can be tested, recorded and compared in this way.

All these devices will be integrated into one system and the data will be sent back to a computer database for analysis. This provides very useful information to the player and coach, and helps to develop good heading methods and prevent brain injury, in the real world. For example, if a player has numerous high impact headers in a game, that exceed a limit that is potential damaging, then they can be forced to stop playing the game for a safe period. It will be especially useful for children learning the game as they aren't fully developed and a brain trauma such as this could be more harmful. This will increase the safety for players which is vital for the continuation of the sport. In future, this type of system could be integrated into wearable headgear for all impact sports, similar to the current sports performance tracking vests worn to gather data about the players movement, performance and fatigue levels.

2 BACKGROUND RESEARCH

To gain a better understanding about concussion, research was carried out online, to see how widespread it is in the game of soccer and to understand fully how it happens. Concussion is defined as “a blow to the head that results in the disturbance of cerebral functions” [2]. Some parts of the skull are weaker, such as the pterion which is the thinnest part of the skull and is the ‘H-shaped’ junction located between temporal, parietal, frontal and sphenoid bones [3]. Areas such as this are made of softer tissue and can handle less impact force than others before it has a damaging effect on the brain.

Further study was carried out to find the difference in concussive head injuries and more minor injuries that don’t cause unconsciousness but could potentially result in brain damage. It was revealed that they are defined as sub-concussive and that there had been many recent studies in relation to this topic. Heading the ball in soccer is one of the most common sub-concussive events, but further research was needed to determine whether multiple events like these could be detrimental to one’s health. A study was found where the researchers were trying to determine whether heading the ball has any negative effects to the brain function and structure [4]. From this research it made me inquisitive on the topic and as a former player of the game, at amateur level, I became interested in discovering if these sub-concussive events could have long term health effects. Children are more susceptible to head injury as they are not full developed. Therefore, regularly heading the ball at trainings and at matches could potentially have an even greater effect than in adults. The need for a means of measuring the impact forces and movements involved in performing a header became apparent. Comparisons could then be made and analysed, so that good skills could be developed, thus avoiding any unforeseen harmful long-term impact on the health of players.

Further research was necessary to develop the best design and to choose the equipment needed. To measure forces a component called a force-sensing resistor (FSR) was considered. Details and specification about this device were gained through helpful sources such as FSR –Interlink [5]. This describes in detail what exactly an FSR does, how it measures forces, how it’s made and circuits that it can be used with. It is important to have a good understanding of how components work in order to choose the most suitable for the design. The other device found for measuring forces at an affordable and available level was an accelerometer. These devices measure axial forces in 3 planes [6]. Through the use of these two devices all the measurements of force could be taken that were needed to provide results. The next step was to decide on specific components and to design the electronic circuitry.

Finally, a data analysis phase was needed to gather and compare results which would include the graphing and rating of each event. The data would have to be saved onto a local device and sent wirelessly, so as not to affect the range of motion of each event. After researching all options, a mobile phone application (App) seemed the most suitable for this task, because of its ease of use and adaptability.

3 COMPONENTS

3.1 Sensors

There are many considerations when developing a project such as this and it is essential to start with a good design and to choose the best components available. The first major decision was to decide on which sensory devices were needed to measure the forces. These forces are high impact and movement related, so devices to measure these must be robust, durable and flexible. FSRs were chosen to detect the force and the position of impact to the head. There will be three of these across the forehead and each will be connected to the analogue inputs in the microprocessor. They will also be able to record where on the head the ball lands, as it will be divided into three subsections, one for each FSR. These will be attached to a wearable cap (head mask), to keep them in place on the forehead.

The other sensors needed are accelerometers, used to measure the movement of the head, neck and ball throughout the heading event. The first will be fixed on top of head, the second will be on the back of the player's neck and both attached to the head mask. The final accelerometer will be attached to the offside of the ball and will be able to send data back via Bluetooth, using a second microprocessor. The devices on the head mask will feedback information to the microprocessor through analogue inputs. The specification of the microprocessor must allow for a minimum of six analogue inputs for the head circuit.

For testing, the football will be suspended by a cord, to make a pendulum, this to keep the circumstances the same for each heading event and to avoid inconsistencies. The second accelerometer and microcontroller will be attached at the back of the football.



Figure 3.1.1 FSR 402

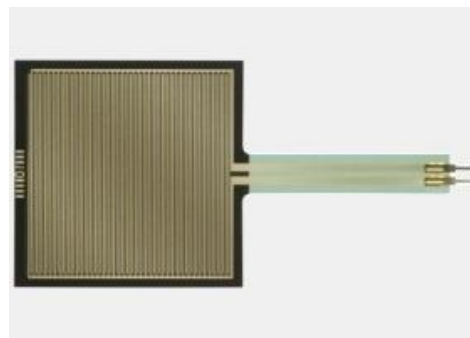


Figure 3.1.2 FSR 406

For FSRs there is a choice of two types FSR 402 figure 3.1.1 and FSR 406 figure 3.1.2 [5]. They both have very similar technologies, 14.9mm diameter and 39.4mm square active areas respectively. The main aim here is to cover as much of forehead as possible and to minimise the number of analogue

pins required. Thus, the FSR 406 was chosen as only three analogue pins are needed, as opposed to five with the 402, and it covers a bigger area. This results in three separate analogue pins being needed which will have to be taken into account when deciding on which microcontroller is needed.



Figure 3.1.3 ITEAD ADXL 335



Figure 3.1.4 SparkFun SEN-11977

There is a choice in Accelerometers of two main types ITEAD ADXL335 figure 3.1.3 [7] and the gyroscope SparkFun SEN-11977 figure 3.1.4 [8]. The advantage of using the gyroscope is that it has digital output instead of analogue leading to less inputs needed for the microprocessor. The main difference between these two devices is that an accelerometer uses vibration to determine non-gravitational acceleration but a gyroscope uses the earth's gravity to help determine orientation. They will work similarly for this experiment as the acceleration due to gravitational pull won't affect the information gathered, but testing is needed to determine this. Although having less analogue inputs is easier for choosing a microcontroller, having analogue signals as opposed to digital ones will be favourable as the microcontroller will have an in-built accelerometer which works with analogue readings. If both of these operate differently it would be harder to compare the two thus the ITEAD ADXL 335 is the chosen accelerometer.

3.2 Microcontroller



Figure 3.2.1 Arduino Genuino 101



Figure 3.2.2 Arduino Mega

A microcontroller is the major decision that has to be made. For this project the physical size of the microprocessor wasn't of vital importance as the plan is to put it in a casing and use a Velcro strap, around the shoulder/neck, to keep it in place. It will need to connect wirelessly to the software so Bluetooth is the easiest and most affordable way to do so. It will have to have enough memory to store a recording before it is transmitted back to database. Finally it will have to have enough analogue and

digital pins for the devices its recording. The two main available microcontroller types I looked into were Raspberry Pi and Arduino. I have used Arduino before and also there is a wider choice available in the laboratory so I decided to choose Arduino. The next step was to choose which type of Arduino microcontroller I need. The choice was narrowed down to two microprocessors between the Arduino Genuino 101 figure 3.2.1 [9], and the Arduino Mega figure 3.2.2 [10], as they both have the memory and pins to suffice for this project. They have 6 and 12 analogue inputs respectively and 14 and 54 input/output (I/O) digital pins respectively. The Mega is 40mm longer in physical size and 50kb extra memory, which isn't significantly more space for this to be chosen. There is built-in Bluetooth in the Genuino 101 which is a bonus as it wouldn't need extra pins setting up Bluetooth as a separate component. Also, the Genuino has a built-in accelerometer which is needed to measure forces in the neck thus making it the obvious choice. The main issue with this was the non-availability of the devices, as Arduino had stopped making them. There was only one available in the lab and the other had to be bought through an outside source.

The second microcontroller at the back of the ball, needed three analogue pins or an in-built accelerometer and to be able to support Bluetooth, to feed data back to the app. There were many choices available but the objective was to keep it as simple as possible and to avoid added complication. The Bluno Nano device was considered, seen below figure 3.23 [11], has eight analogue pins, incorporated Bluetooth and supported BLE also. It is an enhancement on the Arduino Uno with BLE incorporated. At first this was the chosen device but after testing with the extra circuit it was very hard to keep the circuit intact as the ball had so much movement. It would be easier to completely remove the need for a second circuit and just use the second Arduino Genuino 101 as it could transmit the recorded data through BLE without any circuit due to the built-in accelerometer.

This was the final component to be chosen for the project so the next step was to choose the type of Bluetooth to use and whether an RTC was necessary or not.

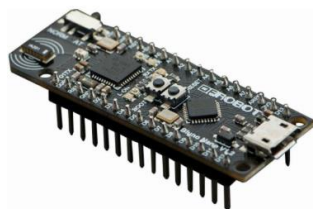


Figure 3.2.3 Bluno Nano Microcontroller

3.3 Bluetooth VS Bluetooth Low-Energy (BLE)

The choice for transmission is between standard Bluetooth and Bluetooth Low-Energy (BLE). The major difference is that the BLE transmits at very low energy, thus less power is needed for each transmission and the microcontroller won't run out of battery charge during large transmission [12]. The BLE device stays in sleep mode until a connection is made as opposed to Bluetooth which stays constantly on. The connection time for BLE is much less than Bluetooth which leads to further power savings also giving it another advantage. Bluetooth is more suitable when large amounts of data are being transferred compared to BLE which has a small limit of 20 bytes per packet of transmission [13].

Both of these methods have their advantages but for the purpose of this project BLE is more suited as the data being transmitted is small and the microcontrollers will be powered with a battery, therefore power consumption is important. The Arduino Genuino 101 had BLE built-in and therefore this was the chosen transmission method. Both of these transmission methods can connect to multiple devices where a central node like a mobile application can connect to two or more peripheral devices such as the two microcontrollers.

3.4 Timing

Synchronising the two devices using a Real-Time Clock (RTC) so that both devices would be recording at the same time and the information could give live feedback was one possible solution so the data would be comparable. This was found to be unnecessary as the analysis from the event would only happen after it occurred, therefore the data would be recorded sent to the application and could then be graphed. The signals will be sent out to both peripherals at the same time and will record the data with the same timestamps so the readings will be synchronised to a very close margin and further accuracy isn't needed. It was decided that for the data being transmitted it wasn't necessary to have real time transmission. This will also reduce storing errors with limited space on microcontroller.

As the signalling processes data for one header sends it back and saves it to the database the use of real time clocking isn't necessary. If the timing increment of each reading is sent periodically then the results can be stored and graphed with accuracy without the need of a real time clock. For this to work, an application that handles said data correctly is of vital importance. Using a button to turn on/off Bluetooth and connect simultaneously to the two microcontrollers is essential as this commences the connection between the devices and the app. This ensures that the recordings are done under same time frame and can be graphed and analysis comparatively.

When originally coding for the microcontrollers the ‘delay’ function was used to add a time stoppage between readings. This leads to problems if multiple activities, such as measuring more than one device simultaneously, it will delay certain readings and transmission won’t work well. The ‘millis()’ function in Arduino must be used to separate timing of these events so the readings can be read logically.

```
unsigned long timestamp_start = millis();

while(true) {
  if ((millis() - timestamp) >= SAMPLE_TIME) {
    timestamp = millis();
    CurieIMU.readAccelerometer(ax, ay, az);
    data_array[1] = (int)(timestamp - timestamp_start) & 0xFF;
    data_array[0] = ((int)(timestamp - timestamp_start) >> 8) & 0xFF;
```

Figure 3.4.1 Timestamp reading

SAMPLE_TIME is defined as 40ms. Every time the difference between ‘millis()’ and timestamp is greater than 40 a new reading is saved to the data array and the timestamp is taken then and saved to the array also using the variable ‘timestamp_start’. This starts at zero as originally it is the same as the timestamp then for each iteration of the loop it will increase by 40ms. It will record this way on both devices and the readings can be compared graphically.

4 PROJECT STRUCTURE

4.1 Circuit

There are two separate circuits figure 4.1.1 is the main circuit attached to the head as a wearable cap which is connected to the Head Arduino Genuino 101. The second circuit is solo Arduino Genuino 101 which is attached to the offside of the ball. The first circuit operates for the accelerometer with x, y, z as three analogue output pins A3-A5 respectively, with GND going to GND of the Arduino and +3.3V going to the voltage supply in the microcontroller. The three FSRs are connected to the power supply (VCC). The other electrode of each goes to GND via a 10K ohm resistor and to the analogue pin A0-A2. As the FSR is technically a resistor, it is transitive so it doesn't matter which end is attached. There is also a 9V battery plugged into the barrel connector to power the circuit.

The second circuit is much more simple to build as it only has to detect the accelerometer forces x, y, z and transmit them back digital using BLE transmission. Originally it was planned to have an external accelerometer but it was easier to work with the Genuino microprocessor again and therefore will be the same as figure 4.1.2 shown below with a 9V battery barrel connector.

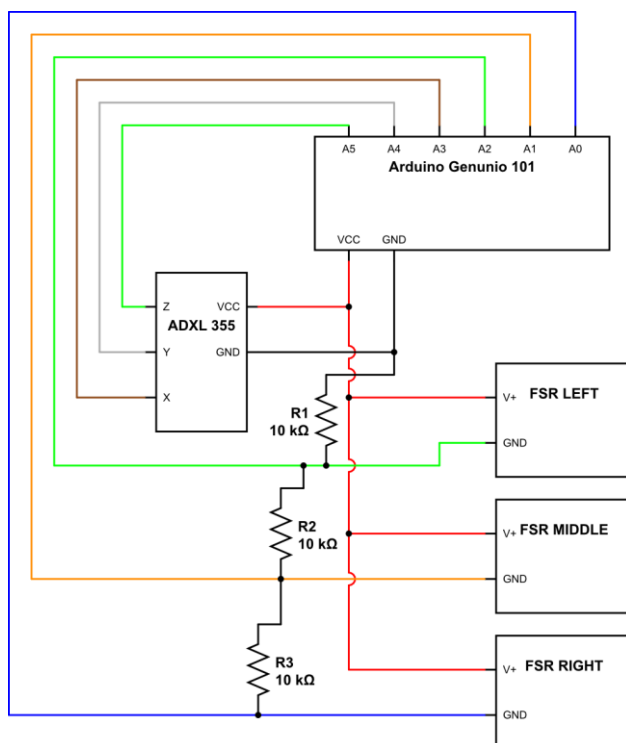


Figure 4.1.1 Genuino 101 Circuit

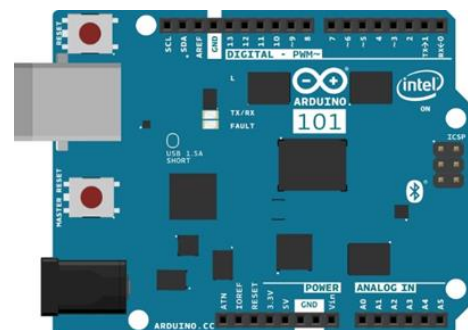


Figure 4.1.2 Genuino 101 pins

4.2 FSR Operation

The FSR works by sensing the pressure applied and changing its resistance based on this. The higher the force the less the resistance which gives a rise in voltage in the circuit outputting an analogue value between 0-1023. FSR's are made of layers of conducting and non-conducting materials, the more force that is applied to the surface, the more conducting particles come into contact and the resistance between electrodes is reduced, but in a non-linear response.

At first for this project it was thought that the FSR could determine the actual force value of the pressure of the ball hitting the head in Newton's. When the FSR's went through initial testing it was clear that this would not be possible as it wasn't sensitive enough to a large range of pressure values. When pressed lightly it would produce a low analogue value and when pressed slightly harder it would reach 1023 quite easily. Thus, using this to differentiate between hard headers would not give any useful information. Their purpose in the project will be to act as a switch to work with the accelerometer axial force measurements to determine the exact point of contact with the ball. This can be used to compare headers although not what it was originally intended for.

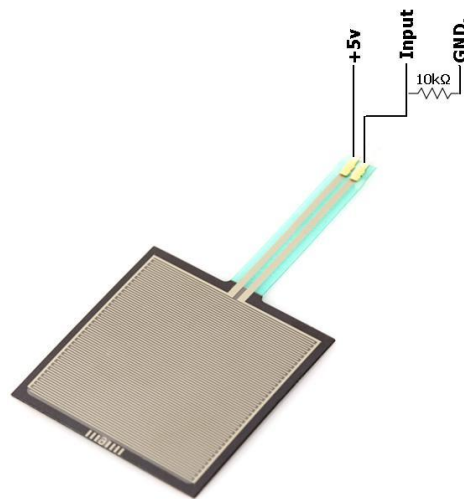


Figure 4.2.1 FSR 406 Circuit Diagram

4.3 Accelerometer Operation

The accelerometer measures forces in the x, y, and z directions. These can be used to compare headers in terms of movement and acceleration forces. The output analogue reading from 0-1023 is given for each axis by sensing the amount of dynamic acceleration. It uses this to find out the direction and speed of the device. The ADXL 335 is a small, thin low powered accelerometer ideal for reading values and consuming small amounts of power. An accelerometer operates using three electrodes in each axial direction and depending on the microscopic distance between the middle electrode and plate it changes its output voltage value i.e. its analogue reading [14]. This procedure is happening constantly in each direction and the values are read back to the microcontroller. The acceleration forces can be determined by change in all three these directions over time. The difference between this and a gyroscope is the accelerometer measures linear motion in three directions whereas the gyroscope measures angular motion so is unaffected by gravity.

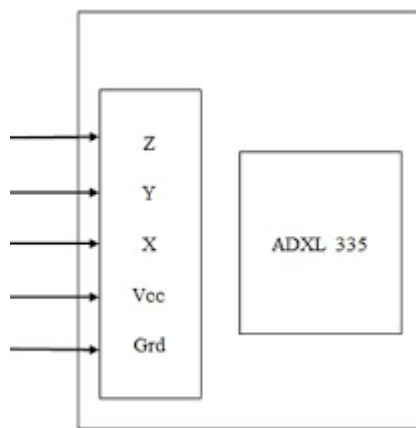


Figure 4.3.1 ADXL 335 pins

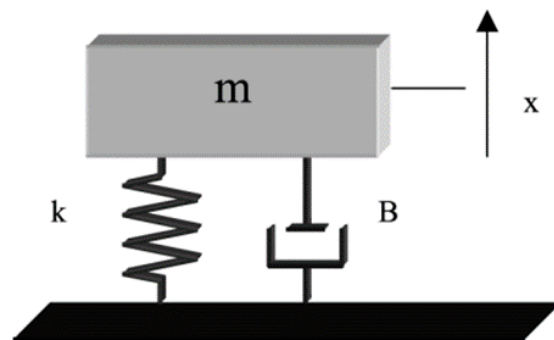


Figure 4.3.2 Simplified Accelerometer operation

Shown in figure 4.3.2 is a simplified diagram of how an accelerometer operates [15]. The mass M is attached to spring of stiffness K and also to dashpot that has damping coefficient B . When the accelerometer moves with linear acceleration the mass moves causing deflection in the dashpot which is sensed and converted to electrical current directly proportional to the acceleration. It uses the basic principles of Newton's 2nd Law which states "Force is equal to the change in momentum per change in time. For a constant mass, force equals mass times acceleration" [16]. From this law the formula $F=ma$ is derived. The operation of the below figure will give a voltage analogue output for one direction of the accelerometer. To operate as a 3-axis accelerometer like ADXL-335 it will need 3 more of these devices built into one. This will then output the analogue voltages for the 3-plane directions X, Y and Z. To convert these to comparable readings further calculations are necessary depending on the inputted voltage. This will be described in the testing section 8.1.

4.4 Data to Microcontroller

The diagram below shows how the analogue readings, from the devices, are converted by the microcontroller, sent by Bluetooth to the mobile application and saved to csv files. The two circuits operate as described previously in section 4.1. The analogue reading from each device is converted to digital readings at the pins of the microcontrollers. Each timestamp, these values are saved to an array of size 20 bytes (packet limit for BLE) when the ‘START’ button on the Android device is pressed. After every cycle these readings are sent to the application until the ‘STOP’ button is pressed. Originally it was planned to save all the readings locally on the Arduino before sending them to the phone but due to the lack of usable memory it was easier to save each reading to an array of 20 bytes, to the RAM of the microprocessor and send it every 40ms. All the packets sent between ‘START’ and ‘STOP’ are then saved to a csv file locally. Figure 4.4.1 displays the full flow of the data transfer from the first recording until being analysed on the mobile phone application.

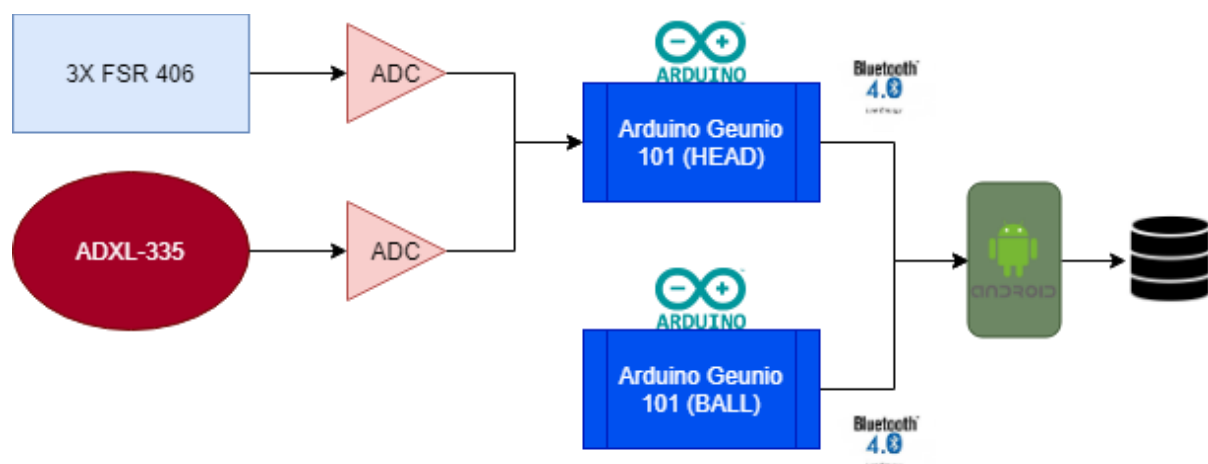


Figure 4.4.1 Dataflow Diagram

Each analogue reading is converted using an ADC which has a 10-bit resolution, 1024 values (2^{10}) that give readings from 0-1023. To convert this to a useable voltage reading the formula for ADC is used, shown in figure 4.4.2 [17].

$$\frac{VCC}{\text{Resolution of ADC}} \times \text{Analogue Reading} = \text{ADC Voltage}$$

Figure 4.4.2 ADC conversion

If for example, $VCC = 3300mV$, $resolution = 2^{10}$, and the analogue reading = 512 the outputted voltage 1650mV which is the maximum it could be for the inputted voltage. This will be converted on the mobile application before it is graphed and compared.

The readings from the devices are saved locally on the microprocessor using 'data_array[20]', this operation is shown below in figure 4.4.3. The size of each integer is 16 bits to send this across the link using 'setValue()' function of the peripheral characteristic 'TRANSFER' shown in figure 4.4.4 which sends the value on the BLE link. The maximum size of the array is 20 bytes each member of the array is one byte in length. Therefore, each integer recorded from the devices must be split from 2 bytes (16 bit) integers into single bytes and saved to the array. It sets the first 8 bits to one element in the array and the second 8 bits to the next element which after transfer are joined again on the mobile end before being saved to the database.

```
data_array[1] = (int)(timestamp - timestamp_start) & 0xFF;
data_array[0] = ((int)(timestamp - timestamp_start) >> 8) & 0xFF;

fsr_data = FSR_CAL(fsrLEFT);
data_array[3] = fsr_data & 0xFF;
data_array[2] = (fsr_data >> 8) & 0xFF;

fsr_data = FSR_CAL(fsrMIDDLE);
data_array[5] = fsr_data & 0xFF;
data_array[4] = (fsr_data >> 8) & 0xFF;

fsr_data = FSR_CAL(fsrRIGHT);
data_array[7] = fsr_data & 0xFF;
data_array[6] = (fsr_data >> 8) & 0xFF;

accl_data = ACC_CAL(xpin);
data_array[9] = accl_data & 0xFF;
data_array[8] = (accl_data >> 8) & 0xFF;
```

Figure 4.4.3 Data array configuration

```
TRANSFER.setValue(data_array, 20);
```

Figure 4.4.4 Set value function

4.5 Bluetooth Connection

The built-in Bluetooth module in the Genuino 101 can transmit using BLE [18]. There is a built in Bluetooth low-energy module in both microprocessors which is the 'Nordic Bluetooth low energy controller NRF51822'. Standard Bluetooth connection is based asynchronously serial connection (UART) see explanation of this in 4.6, BLE acts like a community bulletin board. It is made up of central (Mobile phone) and peripheral (microcontroller) devices.

When using BLE it is important to know how they connect and send and receive data. Generic Attribute Profile (GATT) give hierarchical structure to the BLE state data in layers called Services, Characteristics and Descriptors. It is built on top of Attribute Protocol (ATT) which defines the communication between two BLE devices. GATT defines client and server roles in the communication these can be broken into three groups' discovery, client-initiated and server-initiated [19]. GATT

provides a structural and contextual representation of a device and ATT allows that data structure to be used over a BLE communications link. See the figure 4.5.1 [20] below explains how data are sent between server and client using GATT. The ATT will allow us to request and change characteristics and send notifications and many more features. This will be very useful when data has to be sent between the central and peripheral devices in this project.

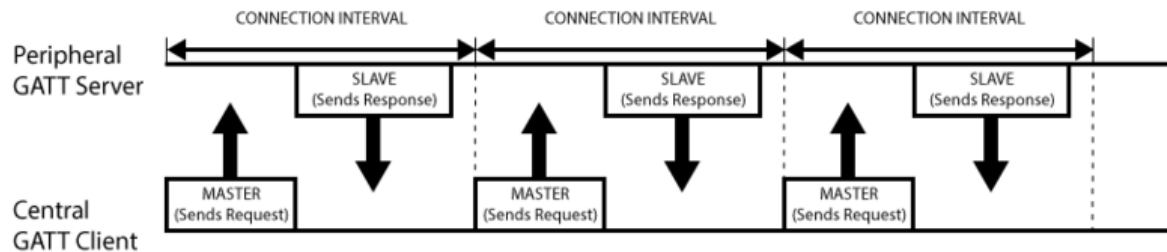


Figure 4.5.1 Client/Server BLE ATT Protocol

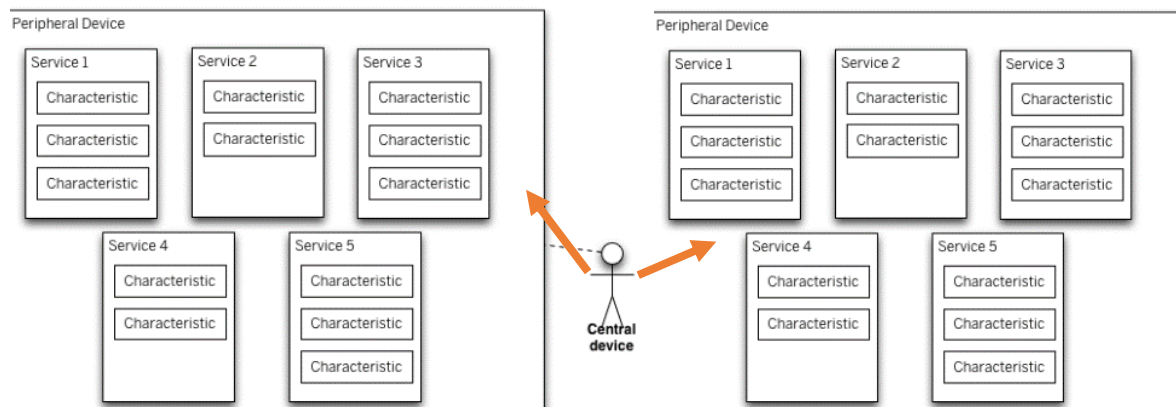


Figure 4.5.2 BLE operation

See figure 4.5.2 [18] above that shows the execution of the BLE. The central node connects to either one or more peripheral devices (two for this project), it connects between these devices using Generic Access Profiles (GAP). There are four possible roles central, peripheral, broadcaster and observer. A Peripheral advertises with small frequent streams of data, inviting and accepting connections from Central devices. The Central device scans for BLE devices, searching for advertising peripheral devices. Although they are not used in this project, a Broadcaster advertises but it does not accept connections and an Observer scans advertising packets but never connects to devices. For this project there will be one central device the mobile phone and two peripheral devices one on the ball and one on the head.

To communicate information between GAP devices using services, characteristics and descriptors. Services are top level containers and typically correspond to some kind of primary feature of a device. For this project there is only one service for all communication, this is called 'HEAD_ARD_SERVICE' and has a specified 16-bit Universal unique Identifier (UUID) and has characteristics to send and receive data. Characteristics are individual data items, have a defined type,

an associated value and support one or more operations. Characteristics belong to a service and for this project there are two characteristics namely ‘RUN’ and ‘TRANSFER’. These are defined in figure 4.5.3 below.

```
// BLE objects
BLEPeripheral blePeripheral;

// GAP properties
char device_name[] = "HEAD_ARD";

BLEService HEAD_ARD_SERVICE("19B10000-E8F2-537E-4F6C-D104768A1214");

BLEIntCharacteristic RUN("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite);
BLECharacteristic TRANSFER("19b10002-e8f2-537e-4f6c-d104768a1214", BLERead | BLENotify, 20);
```

Figure 4.5.3 Code initialising Service and Characteristics

The device once connected waits for command to read in the ‘RUN’ characteristic sent using the ‘START’ button on the application. This will begin to start saving recorded data from the devices to the 20-byte array. The array of data are transferred using the characteristic ‘TRANSFER’ every timestamp and stops when the ‘STOP’ button is pressed on the application.

Descriptors are the last optional layer that belongs to certain characteristics. They give information for a characteristic or give a way to control the behaviour of a characteristic. This project contains no descriptors as they are not necessary for the tasks needed.

4.6 Bluetooth Transmission

As there is a maximum amount of data that can be transferred per second, when comparing BLE and Bluetooth, it isn’t a factor in this project as the collected data is very small, 20 Bytes per frame. The first two bytes are given to the timestamp, the next six are given to the FSR readings (two each) and the last twelve are saved for the two accelerometers (six each). The frame length of 40ms, which is 25 times per second, is sufficient to capture any heading event. This means the transfer speed needed is 500 bytes per second. The Samsung S6 that is being used for the app has a slave latency of 10ms and can transfer the data at 2KB/s therefore the throughput isn’t limited from using BLE so it won’t affect the performance of the system.

The Nordic NRF51822 is compliant with Bluetooth 4.1, built around a 32-bit CPU with 256KB of embedded flash and 16 KB of RAM for improved application performance [21]. There is a 32 kHz oscillating clock which uses I^2C bus protocol for modifying the main bus clock as it allows multiple

masters and slaves on the bus. For serial communication it uses Serial Peripheral Interface (SPI) as it is much faster for transferring data due to it having a simpler protocol. SPI is different to I^2C as it works with only one master and multiple slaves. To communicate with external BLE devices it uses Universal Asynchronous Receiver Transmitter (UART) communication. Although Universal Serial Bus (USB) is the industry standard for connecting peripheral devices and is faster and better for larger volumes of data transfer UART is used for this BLE chip. There are two UART interfaces UART1 is available to the user and UART0 is used for Bluetooth communication.

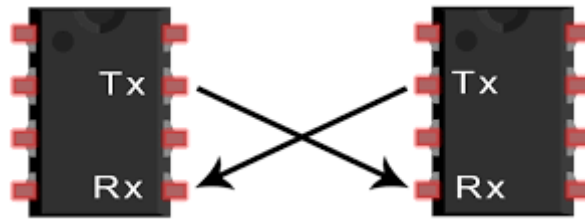


Figure 4.6.1 Two UART devices communicating

UART is used to send the data from the Arduino (peripheral node) to the phone (central node) using Host Control Interface (HCI). The HCI is a thin layer that essentially provides a uniform method of accessing the Bluetooth capabilities [22]. HCI UART Transport Layer is used to allow the use of Bluetooth HCI over a serial interface between two UARTs on the same line. It assumes it's free from line errors.

UART is a serial form of transmission and when the host transmits it sends the data to the UART which then sends it out in single bytes. The UART orders its data using a first-in first-out queue (FIFO) and is transferred according to the order it receives the data from the host.

UART uses direct communication between two devices through the pins of each one for sending data out TX and one for receiving data to the device RX as shown in the diagram above figure 4.6.1 [23]. For the purpose of this project TX is used to send data to the central node from the peripheral device and RX is used to receive data from the central. The data are inputted into each UART in byte form and sent across the channel as bits before being formed into byte format again on the other side of transmission. The TX also adds a start, parity and stop bit to the byte on arrival to create a data packet. The UART on the receiving side reads the packet and removes these added bits and it is finally exported serially along the link. UARTs use asynchronous transmission, which is why these extra bits are added to define start and end of the packet.

This configuration of BLE makes it possible to send information from the Arduino peripheral devices to the Android mobile device. For the Android device to be able to understand this information and make use of the data for analysis, some type of application is needed to handle the data. The next section of the project describes the creation of this application.

5 ANDROID APPLICATION

5.1 Android vs IOS

Android is a mobile operating system developed by Google to power smartphones and tablet PCs. It is based on a modified version of Linux Kernel. It's developed with OHA (Open Handset Alliance) meaning anyone can develop Android on their hardware. This is the reason many manufacturers use Android as they have access through OHA. Android is open source and its development tools are free (Compared to Apple's iOS which is \$100/year to develop applications) and easy to use which is why it became the most popular operating system for smartphones.

IOS was created by apple and is used in iPhones. It is limited as the operating system is Mac OS and is the only place that an app can be developed. Although there is a disadvantage in cost of development and limitation of Mac OS, the IOS market is much quicker to deploy, as the app only has to be applicable to a small number of similar iPhones. Whereas, there are numerous types of Androids in which the application must be designed to work on, which leads to a longer and more costly development time.

As this app wasn't being deployed and free access was with Android studio, which is coded in Java and I have previous experience with, the Android application was chosen over the IOS option.

5.2 Android Studios

To develop an app for this project first the key concepts and applications of Android studios must be understood. All applications in Android studios are compiled using the Android Software Development Kit (SDK). These are a set of development tools used to develop applications for Android including required libraries, a debugger and an emulator. These are all bundled into an Android Package (APK) this contains all the contents of the Android app which can be used by Android devices to install the full usable application.

An Android app is protected by the many security features implementing principle of least privilege where individual apps only have access to their own app's code run on a Virtual Machine (VM). If the app requires to outside sources such as files or Bluetooth features it must request permission from the user. The operating system (OS) is a multi-user Linux system in which each application is a different user, this assigns each device a unique Linux user ID, which gives access to the permissions for the files defined through this ID [24].

There are many key concepts in Android studios that define the structure of the app these are all explained in detail below.

Project

Creating a project is the first setup towards making an application with Android Studios. This defines everything for the workspace in an app including sources, resources and build configuration. All elements can be seen using project view when the app has been created. The name and the module of the project must be chosen on creation. The module is the type of provide essential files and templates depending on the which is chosen from five options 'Phone and Tablet', 'Wear OS', 'TV', 'Android Auto' and 'Android Things'. The Application Program Interface (API) level must be chosen also before the project is made the chosen level will depend on the needs of the application and on the device being used. The lower the API level the more devices the application code can be run on but some features for example BLE require an API level 18. This API level supports one Android platform but lower levels of Android will work on newer API levels, although it is not backwards compatible, newer versions don't work on older devices. This is an important decision to make for an application and the app will be more useful if lower levels of API are used with a threshold high enough to support the majority of features the app will be using. The API level can be changed if needed in a build file but this may cause problems with the app when trying to change it.

Activity

Activities are one of the four possible component building blocks for users in Android Studios. The others are Services, Broadcast Receivers and Content Providers. An Activity's purpose is to interact with the user. Each activity will output a single screen interface that the user can interact with. To create an activity a java class must extend the predefined Activity class and implement the 'onCreate' method as shown below in figure 5.2.1. Activities can link with other activities but each one is independent of each other. They are used to manage the current and previous interfaces and what the app does when the user interacts with it. An Activity is defined in the Android Manifest XML file and the display configuration is defined in the resource layout file. In the example below the activity is created with this overridden method and the view of the file is set to the resource file 'activity_main'. Fragment activity such as AppCompatActivity are subclasses of activity and are used to give a more modular design to activities.

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Figure 5.2.1 Creating Activity

Services

A Service is a background entry point for apps to perform remote and long duration tasks such as monitoring step count for a fitness tracker. Services don't have an interface but can bind with an activity to interact with it. There are two types of Services, namely 'Started' and 'Bound'. A Started Service keeps the system running until the task is completed, such as playing music. Bound Services are run when a system has called it to be used which links processes and services together which will be needed for this project to link BLE and the Activity interfaces. Services are very useful tools to provide a concept application to customers. See figure 5.2.2 below for how a Service is initialised. The 'onCreate' method is empty here but should contain the initial setup of the service. The class extends the java predefined Service class and implements overridden method IBinder which can be used to connected Activities activated with the Intent.

```
public class MainActivity extends Service {
    @Override
    public void onCreate() {

    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

Figure 5.2.2 Creating Service

Broadcast Receivers

This is a component that allows the system to broadcast signals outside of standard flow and receive responses from these signals. This is very useful for connecting applications together with a specific broadcasting message that is set to handle this signal and react to it in a certain way. They don't provide a screen interface but can be set to display a bar notification on the screen to provide information to the user. As there were no apps interacting with each other in this project this component was not used.

Content Providers

This is the only component that doesn't use Intents to run. Its purpose is send data to another application on request. These files can be stored on the Internet, SQL Databases and files locally. These are very useful for centralising data in one location so applications can access it when needed. This kind of access wasn't needed, although could have been used for storing data, for this project. The main reason behind this was the data was stored locally in XML files and an extra component would have

over complicated the process. There is also the risk of security with this component as all applications that use it have access to the data, for sensitive data this can be a concern due to GDPR.

Intent

Available in three of the four components above they are objects used to bind components together. For Activities and Services these are used to define actions to perform and for Broadcast receivers it defines the action message to be broadcasted. They are the link between components see the figure 5.2.3 which shows how the peripheral Activity links with the graph Activity and sends information to be used. A new Intent is created stating current and new Activity in its parameters then inputs data using the intent to send it from one string to another in different Activities.

```
public void onANALYSE(View view) {
    Intent intent = new Intent( packageContext: PeripheralControlActivity.this, GraphActivity.class);
    intent.putExtra(GraphActivity.EXTRA_FILE, previousLog_1.toString());
}
```

Figure 5.2.3 Intent between two Activities

View

To configure the output of the application interface for the user the View must be defined. This is how each Activity's layout is setup, essentially the GUI elements of the screen. These can be held together in containers called ViewGroups which are a collection of Views. Fragments are full user interfaces that usually take up part of an Activity and can be reused in different Activities and can contain Views and ViewGroups. The Id for a resource file can be set like figure 5.2.4 below which links the java Activity to the layout XML.

```
setContentView(R.layout.activity_main);
```

Figure 5.2.4 Call View to Resource Layout file

The initial View layout is empty and for every feature of the screen such as textboxes, scroll sliders, switches and buttons these must be designed using an XML resource file. This can be done using the design widget screen or text XML editor as available in the resource layout file. See below a typical example of this for a textbox. This is a typical textbox using text size 26dp called 'Analyse'. Give that dp or dip (density-independent pixels) are the size relative to a 160dpi screen, one dp is one pixel on a 160dpi screen. It is placed in the centre of the screen and this is all done within the TextView definition parameters. There is a Design emulator which displays the contents of the XML file to see how the layout will look on a typical phone. This is shown in figure 5.2.6 where the textbox 'Analyse' is displayed at the top middle of the phones screen. This makes it easier to edit and using this tool the layout can be altered with predefined widgets.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="30dp"
    android:layout_gravity="center"
    android:gravity="center"
    android:text="Analyse"
    android:textSize="26dp"/>
```

Figure 5.2.5 TextView in XML Resource file

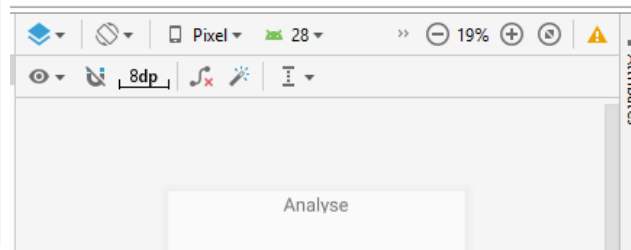


Figure 5.2.6 TextView Design

Android Manifest

Every application contains one and only one Android Manifest File which is the root directory of the project. The Android system must have the information contained in this file before it can run the projects code. It is where the app is setup and where Activities and Services are instantiated. In figure 5.2.7 the 'MainActivity' is setup using intent filters which declare the capabilities of a component. Permissions for certain features are also declare in this file see figure 5.2.8 for an example of how Android asks for permission to use the Location of a device. To create the pop-up that asks for permission to use tools in the device such as using Location this must first be declared along with the configuration of the pop-up in the Java app code.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Figure 5.2.7 Android Manifest File - Activity Setup

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure 5.2.8 Android Manifest File – Permission

Build App Gradle

Gradle is the build tool used by Google to build Android packages it initialised the setup of the Android SDK, any repositories it takes information from and all dependencies it needs to run. It will configure, extend and customise the build process and is an essential tool used to import external sources or packages used. The build needs to be resynchronised every time there are any changes made to this file.

5.3 Heading Ball Application

When designing this application, the aim was to connect to two BLE peripheral devices, receive the data that they sent, store this data and retrieve it for analysis. To accomplish this task and to capture a heading event there were many steps involved. First the app scans for BLE devices and when these two peripheral devices are found, it will attempt to connect both, using the Bluetooth ON button. There is also a 'Bluetooth OFF' button used to disconnect the devices from the phone. When the two microprocessors were connected to the application the option to press the 'START' button would become enabled. This button would be used to send a signal characteristic 'RUN' to the peripheral devices and they would start to record the analogue readings from the sensors connected to them. There would be a timestamp set 25 times every second and this would be sent back to the phone along the 'TRANSFER' characteristic. Once data was received on the phone the app will interpret the data and save the incoming data to a local CSV file, row by row, for each incoming timestamp. This stream would continue until the 'STOP' button on the phone application was pressed. This would stop the microprocessor from recording new readings and send the remaining readings to the created CSV file. This CSV file would be available to find on the device. The final button's purpose labelled 'ANALYSE' was to analyse the recorded waveform graphically. There would be one graph for each accelerometer, three in total, and the FSR values would be shown on each. This would also display a rating for the event determined using a machine learning algorithm to compare headers which would be displayed at the top of the screen. Ease of use was very important when designing this application therefore when creating the layout and structure of the app it had to be displayed in a clear and concise manner. Therefore a meaningful and smooth flow between Activities was implemented.

There was certain conditions that the application needed, to implement the required features. A minimum of API level 14 required for 'MPAndroid' graphing capabilities and API level 18 for was necessary for BLE connectivity. Level 18 corresponds to the Android version 4.3. The phone being used to run the app is a Samsung Galaxy S6 which runs on Android version 7.0 Nougat or API level 24. When creating the app it was decided in the event that extra features were needed, a higher API level would be more likely to have the capabilities to run them. The disadvantage of course is that level 24 would only run on approximately 37.1% of devices. As the project wouldn't be released to the public, the features of the application were of higher priority than compatibility rating of handheld devices. API level 24 was chosen.

5.4 Scanning Activity

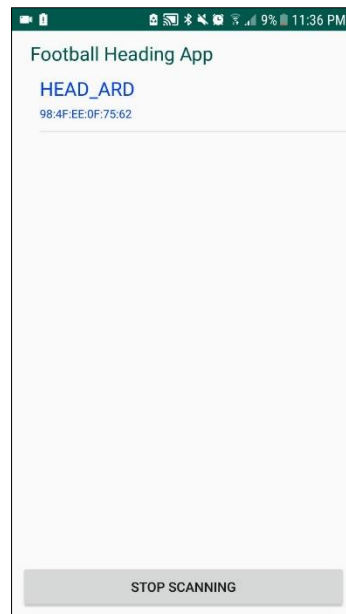


Figure 5.4.1 Scanning Activity Interface

The first activity to be displayed on the application was the scanning activity figure 5.4.1 above shows the displayed interface. This was called ‘MainActivity’ in the Android project. I found many examples of Bluetooth connectivity online but they follow the basic guidelines of connectivity with BLE devices I found the Bluetooth Developer Starter Kit the most useful [25]. The first task was to allow the application to access location and the Bluetooth features of the phone. These permissions were put into the Manifest file see figure 5.4.2 below.

```
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure 5.4.2 Bluetooth and Location permissions

The ‘MainActivity’ class uses a class called ‘BleScanner’ to scan for BLE devices Android has created predefined methods to start, stop and set scanning. To configure the layout at in figure 5.4.1 two resource layout files were used ‘activity_main.xml’ and ‘list_row.xml’. The main layout included a title, a ListView to be filled with BLE devices and a button to initialise the scanning Activity. The List layout was two textViews that would hold a devices name and the second holds the IP address of the device this view was imported into the main layout to list devices. There is also an interface ‘ScanResultsConsumer’ which is implemented by the ‘MainActivity’ used to receive and process data from the Bluetooth scanning objects.

BleScanner

First the variables and constructors are coded including the Bluetooth predefined objects `BluetoothLeScanner` and `BluetoothAdapter` which are key in the configuration of Bluetooth. In the constructor the `BluetoothManager` object is called, shown in figure 5.4.3. This provides an instance of `BluetoothAdapter` which checks if the Bluetooth on the device is currently on/off and has the capabilities to turn it on.

```
public BleScanner(Context context) {
    this.context = context;

    final BluetoothManager bluetoothManager = (BluetoothManager) context.getSystemService(Context.BLUETOOTH_SERVICE);
    bluetooth_adapter = bluetoothManager.getAdapter();
}
```

Figure 5.4.3 BluetoothManager Created

The method `startScanning` allows another object to initiate Bluetooth scanning. Filters are also added in this method to only search for devices with the correct names. These were 'HEAD_ARD' and 'BALL_ARD' set for the head and ball Arduino peripheral devices respectively. This will increase the speed of connection and also filter out unwanted devices, see the filter configuration below in figure 5.4.4. The `ScanSettings` object seen below the filters uses `SCAN_MODE_LOW_LATENCY` will scan aggressively, meaning advertising packets will be collected quickly. This tool is useful for purpose of this project as the devices will be near and are specific to the assignments but in a general sense it isn't recommended as it is expensive in terms of battery consumption.

```
List<ScanFilter> filters;
filters = new ArrayList<ScanFilter>();
ScanFilter filter = new ScanFilter.Builder().setDeviceName("HEAD_ARD").build();
filters.add(filter);
ScanFilter filter1 = new ScanFilter.Builder().setDeviceName("BALL_ARD").build();
filters.add(filter1);
ScanSettings settings = new ScanSettings.Builder().setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY).build();
```

Figure 5.4.4 Scanning Filters

The `startScan()` function is called to start the scan at the bottom of the `startScanning` method. This uses the filters and settings and initialises the `scan_callback` function to retrieve objects that are found when scanning. Figure 5.4.5 shows this function being created.

```
scanner.startScan(filters, settings, scan_callback);
```

Figure 5.4.5 Start Scanning function

The `scan_callback` class defined below in figure 5.4.6 returns the found devices that match the filter criteria. It returns with these devices names, raw byte arrays and their RSSI strength. This class can now scan when needed and give details of the found BLE devices to another object. The `MainActivity` will now need to use this functionality to display and react to actions from the user.

```

private ScanCallback scan_callback = new ScanCallback() {
    public void onScanResult(int callbackType, final ScanResult result) {
        if (!scanning) {
            return;
        }
        scan_results_consumer.candidateBleDevice(result.getDevice(), result.getScanRecord().getBytes(), result.getRssi());
    }
};

```

Figure 5.4.6 Scan Callback Inner-Class

MainActivity

To implement the BLE scan first a BleScanner object is created to scan when the button is pressed this is instantiated using the code shown in 5.4.7. The onCreate method links the activity_main XML file using a View also attaching the ListView to the ble_adapter. When a BLE device is found in the BleScanner Callback it adds it to the Bluetooth ListAdapter and retrieves the name and address of the device. These are updated to the ListView and will show both the name and address for each device found on the list like seen in figure 5.4.1 for 'HEAD_ARD'.

```
ble_scanner = new BleScanner(this.getApplicationContext());
```

Figure 5.4.7 BleScanner Object

This Activity also has methods to request Bluetooth, Location and file access permissions which are needed for newer versions of Android due to the GDPR of mobile phone users. These will pop-up with requests for permissions and also turn on tools needed if they originally disabled.

The method onScan() will start the scanning method for a time period of 5000ms, if all the permissions have been accepted. Depending on the current scanning state the Scan button is updated using the setScanState method seen in figure 5.4.8.

```

private void setScanState(boolean value) {
    ble_scanning = value;
    Log.d(Constants.TAG, msg: "Setting scan state to "+value);
    ((Button) this.findViewById(R.id.scanButton)).setText(value ? Constants.STOP_SCANNING : Constants.FIND);
}

```

Figure 5.4.8 setScanState method

The scanner will stop after 5000ms but and if there are BLE devices found they will display on the screen filtered from the BleScanner class to only show the 'HEAD_ARD' and 'BALL_ARD'. When saving the data recorded from these devices later the order in which these two will connect is important. It will only allow connection if both peripherals are available. A counter is tokened to mark the order in which they arrive. It sets the first intent.putExtra() in both cases to the HEAD_ARD therefore always connecting to this device first and the BALL_ARD second, as seen in figure 5.4.9. This Intent links the MainActivity to the second PeripheralControlActivity which is used to connect and transmit data

between the BLE devices. It attaches the name and address from the devices listed to the intent which is used to differentiate between them in the next Activity.

```
BluetoothDevice device_1 = ble_device_list_adapter.getDevice(position);
BluetoothDevice device_2 = ble_device_list_adapter.getDevice( position: position+1);
if (toast != null) {
    toast.cancel();
}

if (device_1.getName().equalsIgnoreCase( anotherString: "HEAD_ARD") && device_2.getName().equalsIgnoreCase( anotherString: "BALL_ARD")) {
    Intent intent = new Intent( packageContext: MainActivity.this, PeripheralControlActivity.class);
    intent.putExtra(PeripheralControlActivity.EXTRA_NAME_1, device_1.getName());
    intent.putExtra(PeripheralControlActivity.EXTRA_ID_1, device_1.getAddress());
    intent.putExtra(PeripheralControlActivity.EXTRA_NAME_2, device_2.getName());
    intent.putExtra(PeripheralControlActivity.EXTRA_ID_2, device_2.getAddress());
    startActivity(intent);
}

if (device_2.getName().equalsIgnoreCase( anotherString: "HEAD_ARD") && device_1.getName().equalsIgnoreCase( anotherString: "BALL_ARD")) {
    Intent intent = new Intent( packageContext: MainActivity.this, PeripheralControlActivity.class);
    intent.putExtra(PeripheralControlActivity.EXTRA_NAME_1, device_2.getName());
    intent.putExtra(PeripheralControlActivity.EXTRA_ID_1, device_2.getAddress());
    intent.putExtra(PeripheralControlActivity.EXTRA_NAME_2, device_1.getName());
    intent.putExtra(PeripheralControlActivity.EXTRA_ID_2, device_1.getAddress());
    startActivity(intent);
}
```

Figure 5.4.9 Configuring Intent and device order

5.5 BLE Adapter Service

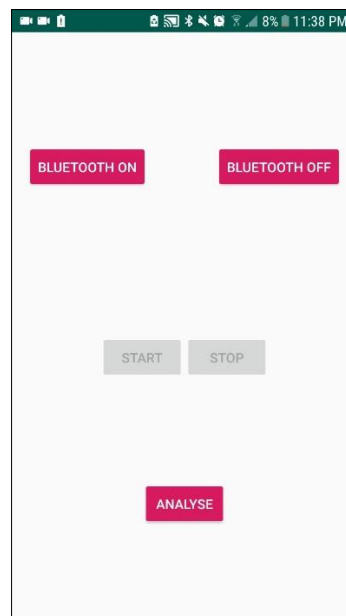


Figure 5.5.1 PeripheralControlActivity Interface

The PeripheralControlActivity is launched which allows the user to interact with the peripheral devices over BLE in various ways, once the scanning phase has been completed. See figure 5.5.1 above which shows the layout of this Activity. In the activity_peripheral_control.xml file this layout is defined using a linear pattern, five button interface. There are two buttons for connection BLUETOOTH ON and BLUETOOTH OFF, two buttons for transmission START and STOP and one button to graph the recorded data, ANALYSE. Similar to the first MainActivity there is a second class that controls the key

actions that it needs to perform. This is an Android Service called `BleAdapterService`. It is important to use a Service that runs in the background as it makes it less likely for the BLE devices to disconnect due to the lifecycle of classes terminating in Android. This Service is called in the `PeripheralControlActivity` `onCreate` method as seen in figure 5.5.2 below. Once the Activity is setup, the resource layout files are defined and the Extra Intents from the scanning Activity are retrieved (the names and the addresses of the BLE devices) this Service is called.

```
// connect to the Bluetooth adapter service
Intent gattServiceIntent = new Intent( packageContext: this, BleAdapterService.class);
bindService(gattServiceIntent, service_connection, BIND_AUTO_CREATE);
```

Figure 5.5.2 Call `BleAdapterService` using Intent

The Service needs to be declared in the Android Manifest as seen in figure 5.5.3 and set enabled. Then the `BleAdapterService` is able to implement the required methods as a Service. The `BluetoothGatt`, `BluetoothManager` and `BluetoothAdapter` objects are created, which will be used in the process of connection and transmission.

```
<service
    android:name=".bluetooth.BleAdapterService"
    android:enabled="true" />
```

Figure 5.5.3 Define `BleAdapterService` in Manifest

For an Activity to use a Service it must be able to “bind” to it. To communicate between these components a handler object is used. There is an `onServiceConnected` method in `PeripheralControlActivity` that needs to be used to connect the Service and Activity, shown in figure 5.5.4. The Service is added to the Activity and `message_handler_1` and `message_handler_2` are setup for communication. There are a number of switch case statements for each handler, an example is seen in figure 5.5.5 whereby depending on the message received by the handler in the `BleAdapterService` there will be a reaction to the interface such as turning on/off buttons or writing messages to the console like in the example.

```
@Override
public void onServiceConnected(ComponentName componentName, IBinder service) {
    bluetooth_le_adapter = ((BleAdapterService.LocalBinder) service).getService();
    bluetooth_le_adapter.setActivityHandler_1(message_handler_1);
    bluetooth_le_adapter.setActivityHandler_2(message_handler_2);
}
```

Figure 5.5.4 `onServiceConnected` Binder


```

@SuppressLint("HandlerLeak")
private Handler message_handler_1 = handleMessage(msg) → {
    Bundle bundle;
    String characteristic_uuid;
    byte[] b;

    switch (msg.what) {
        case BleAdapterService.MESSAGE:
            bundle = msg.getData();
            String text = bundle.getString(BleAdapterService.PARCEL_TEXT);
            showToast(text);
            break;
    }
}

```

Figure 5.5.5 Handler Switch Statements

Connection

The connection to the two BLE peripherals will happen when the BLUETOOTH ON button is pressed. There is a method in PeripheralControlActivity used to act upon this displayed in figure 5.5.6 below. This calls the methods connect_1 and connect_2 from the BleAdapterService class which attempt to connect to the devices.

```

public void onConnect(View view) {
    if (bluetooth_le_adapter != null) {
        if (bluetooth_le_adapter.connect_1(device_address_1)) {
            ((Button) PeripheralControlActivity.this.findViewById(R.id.connectButton)).setEnabled(false);
        } else {
            showToast(msg: "onConnect: failed to connect to device 1");
        }
        if (bluetooth_le_adapter.connect_2(device_address_2)) {
        } else {
            showToast(msg: "onConnect: failed to connect to device 2");
        }
    } else {
        showToast(msg: "onConnect: bluetooth_le_adapter=null");
    }
}

```

Figure 5.5.6 onConnect Method in PeripheralControlActivity

Both connect_1 and connect_2 are the same differentiated with the name of each device. It connects the device found, if it has a name or an address, to the GattService using connectGatt. If this is successful for both devices then the connection is complete and the state will change the BluetoothGattCallback method to “connected”, which is returned to the interface updating the view and disabling the button BLUETOOTH ON.

```

// connect_1 to the device
public boolean connect_1(final String address) {

    if (bluetooth_adapter == null || address == null) {
        sendConsoleMessage_1( text: "connect_1: bluetooth_adapter=null");
        return false;
    }

    device = bluetooth_adapter.getRemoteDevice(address);
    if (device == null) {
        sendConsoleMessage_1( text: "connect_1: device=null");
        return false;
    }

    bluetooth_gatt_1 = device.connectGatt( context: this, autoConnect: false, gatt_callback_1);
    return true;
}

```

Figure 5.5.7 connect_1 method in BleAdapterService

The BleAdapterService will make sure that each connected device has the expected GATT Services before allowing the central node to use the START and STOP buttons. This is also an indicator to the user that the connection has worked correctly as it will display on the screen if the same GATT Services are found from the devices and the central node. This is done using a case switching statement in PeripheralControlActivity which uses the BleAdapterService method GATT Services Discovered, shown in figure 5.5.8 below. This method retrieves each peripheral devices UUIDs for their GATT services. If they use are the same as the UUID for the local GATT services on the phone the statement returns true. Then a message displayed to the console which states “Device has expected GATT services” and the user is allowed to continue to the transmission between devices.

```

for (BluetoothGattService svc : slist) {
    Log.d(Constants.TAG, msg: "UUID=" + svc.getUuid().toString().toUpperCase() + " INSTANCE=" + svc.getInstanceId());
    if (svc.getUuid().toString().equalsIgnoreCase(BleAdapterService.HEAD_ARC_SERVICE)) {
        heading_service = true;
    }
}

if (heading_service) {
    Toast.makeText(getApplicationContext(), text: "Device has expected services", Toast.LENGTH_SHORT).show();
    ((Button) PeripheralControlActivity.this.findViewById(R.id.startbutton)).setEnabled(true);

    if (bluetooth_le_adapter != null && bluetooth_le_adapter.isConnected()) {
        if (!bluetooth_le_adapter.setIndicationsState_1(BleAdapterService.HEAD_ARC_SERVICE, BleAdapterService.TRANSFER_CHARACTERI
            System.out.println("Subscribe failed");
    }
}

```

Figure 5.5.8 Case Statement to compare GATT Services

Pressing either the BLUETOOTH OFF button or the back button on the application will disconnect from these peripheral devices. The disconnect_1 or disconnect_2 disconnects the same way as connect_1 and connect_2 connected to the central device using the BluetoothGattCallback. This concludes the setup of how the application connects and disconnects from BLE devices.

Transmission

For the application to send and receive data from these peripheral devices it must use GATT services as previously stated. There is only one such service used in all transmission called HEAD_ARD_SERVICE which has a UUID of "19B10000-E8F2-537E-4F6C-D104768A1214". This UUID must be identical to the identifier created in each of the peripheral device's GATT services as explained in the Connection section. Once verified the transmission phase can begin which involves the use of the next two buttons, START and STOP.

Inside this GATT service there are two Characteristics used in transmission, namely RUN and TRANSFER. The START button will initialise communication with the peripheral devices by signalling to locally save the readings from their individual sensors. It uses the RUN characteristic to do this which simultaneously writes the RUN_RUN constant (Byte = 01) to the characteristic. The STOP button will also use the RUN characteristic to send the signal but will use the constant RUN_STOP (Byte=00) which stops the transferring of data. These are both shown in figure 5.5.8.

```
bluetooth_le_adapter.writeCharacteristic_1(BleAdapterService.HEAD_ARD_SERVICE, BleAdapterService.RUN_CHARACTERISTIC, Constants.RUN_RUN);
bluetooth_le_adapter.writeCharacteristic_2(BleAdapterService.HEAD_ARD_SERVICE, BleAdapterService.RUN_CHARACTERISTIC, Constants.RUN_RUN);

bluetooth_le_adapter.writeCharacteristic_1(BleAdapterService.HEAD_ARD_SERVICE, BleAdapterService.RUN_CHARACTERISTIC, Constants.RUN_STOP);
bluetooth_le_adapter.writeCharacteristic_2(BleAdapterService.HEAD_ARD_SERVICE, BleAdapterService.RUN_CHARACTERISTIC, Constants.RUN_STOP);
```

Figure 5.5.8 Write to RUN Characteristic enabled and disabled

When the peripheral devices receive the RUN_RUN signal a new timestamp is created and all the current readings are saved to a 'data_array'. Once all readings are saved they are sent back to the central node using the TRANSFER characteristic as shown below in figure 5.5.9. This is a 20 byte array which is received on the application in the switch case statement notification received. To extract the incoming values they must be sorted, ordered and saved to a database. This is the next phase of the application explained in section 5.6.

```
TRANSFER.setValue(data_array, 20);
```

Figure 5.5.9 Write to RUN_STOP Characteristic enabled

5.6 Handling Data

The recorded data are transmitted in packages of 20 bytes over the TRANSFER characteristic. The object bundle is used to transfer the incoming data from the characteristic to other Activities. The data are mapped to String keys (b[0] is first byte in the array) and PeripheralControlActivity retrieves these values. An array of integers are must be unpackaged and converted to usable readings therefore

these packages must be reassembled into their individual recordings. These methods are called in figure 5.6.1 below.

```
case BleAdapterService.NOTIFICATION_OR_INDICATION_RECEIVED:
    bundle = msg.getData();
    characteristic_uuid = bundle.getString(BleAdapterService.PARCEL_CHARACTERISTIC_UUID);
    b = bundle.getBytes(BleAdapterService.PARCEL_VALUE);

    if (characteristic_uuid.equalsIgnoreCase(BleAdapterService.TRANSFER_CHARACTERISTIC)) {
        int timestamp = byteArrayToInt(b[0], b[1]);

        float fsrLEFT = map(byteArrayToInt(b[2], b[3]), ard_voltage_min, ard_voltage_max, out_min: 0, ard_g_max);
        float fsrMIDDLE = map(byteArrayToInt(b[4], b[5]), ard_voltage_min, ard_voltage_max, out_min: 0, ard_g_max);
        float fsrRIGHT = map(byteArrayToInt(b[6], b[7]), ard_voltage_min, ard_voltage_max, out_min: 0, ard_g_max);
    }
```

5.6.1 Case for Receiving Packages

Two bytes make up a full sensory reading or timestamp from the microprocessor. The HEAD_ARD will use all 20 bytes but the BALL_ARD only uses 8 bytes as it has less sensors. As each integer was split into two equal parts on the peripheral side it was only a matter of joining two bytes back together. The method byteArrayToInt in figure 5.6.2 joins each two byte reading. This method joins two bytes each time from the bundle and they are then assigned to an integer such as timestamp shown above in figure 5.6.1.

```
private int byteArrayToInt(byte byte1, byte byte2) {
    return byte2 & 0xFF | (byte1 & 0xFF) << 8;
}
```

Figure 5.6.2 Method byteArrayToInt used to reassemble package

The recordings, with the exclusion of the timestamp, are still in their Analogue forms and must be converted to voltage readings. The map method shown in figure 5.6.3 is used to convert these values which is called in figure 5.6.1. A simple ADC conversion is used, the same as shown in 4.4.2, which is a product of the Analogue resolution and the recorded Analogue reading divided by the voltage range. Each sensor has an Analogue resolution of 1023 with voltage mapped from ± 3.3 Volts for the accelerometer, as it is multi-directional, and 0-3.3Volts for the FSR. The built-in 2g output Accelerometers have digital readings from 0- 64000 and must be mapped in this way. As there is constantly data bytes incoming, while the START button has been pressed, the recordings must be saved to a database.

```
public float map(int x, int in_min, int in_max, float out_min, float out_max){
    return ((float)x - (float)in_min) * (out_max - out_min) / ((float)in_max - (float)in_min) + out_min;
}
```

5.6.3 Map each Analogue value to Voltage

Creating Local Database

A database is used to keep track of all the data recorded. Firebase Database is a Realtime cloud-based NoSQL which lets data be stored and synchronised between users. The data are stored in JSON format and any changes are imported immediately to the system. It supports multiple platforms and is especially noted for connectivity to Android Studios. Originally to store all data the Firebase Database was going to be used but upon consideration this was changed. For purposes of editing and uploading this algorithm to a python script for machine learning like what was needed in this project a filing system of CSV files would act in the same way as a database. Once this decision was made a new class called FileOperation was created to handle the incoming data and export it to local CSV files.

The method createLogFile in the FileOperation class is used to create a directory to save new files. They are all exported into the “Head Ball Logs” folder in the “Documents” directory and saved as CSV files as shown in figure 5.6.4. The name of the file is created using the inputted name and the current time method. This is used to differentiate between the recordings from each peripheral.

```
public static File createLogFile(String title, Context c) throws Exception{
    String name = title + " " + getCurrentTimeStamp();
    File directory = Environment.getExternalStoragePublicDirectory(DIRECTORY_DOCUMENTS);
    File folder = new File(directory, child: "Head Ball Logs");
    folder.mkdir();
    if(folder.exists()){
        System.out.println("Folder Exists");
        return File.createTempFile(name, suffix: ".csv", folder );
    }
}
```

Figure 5.6.4 createLogFile method in FileOperation

The files are called in the peripheral control activity using a File and FileWriter object. When the START button is pressed the file is created shown in figure 5.6.5. This creates a file under the name “Head Data” and the FileWriter object initialises the file listing the title of each column at the top of the CSV.

```
if(currentLog_1 == null) {
    try {
        currentLog_1 = FileOperation.createLogFile( title: "Head Data", getApplicationContext());
        writer_1 = new FileWriter(currentLog_1);
        writer_1.append(titles_1);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figure 5.6.5 Create file when START pressed

When data packages are received are appended depending on column name and all the contents of one package fill the columns of one row. When all the packages have been appended and the STOP button is pressed the remaining data are immediately written to the intended destination using the flush() method and the file is closed as shown in 5.6.6 below. Now that the data has been recorded, transmitted and saved the analysis stage of the application begins using the GraphActivity.

```
try {
    writer_1.flush();
    writer_1.close();
    previousLog_1 = currentLog_1;
    currentLog_1 = null;
}
```

Figure 5.6.6 Close file

5.7 Graphing Activity

GraphActivity is the final Activity used in this application which is called when the Analyse button in PeripheralControlActivity is pressed see figure 5.7.1 below. The putExtra method exports the last CSV files to the GraphActivity to use as coordinates for the Graphs.

```
public void onANALYSE(View view) {
    Intent intent = new Intent( packageContext: PeripheralControlActivity.this, GraphActivity.class);
    intent.putExtra(GraphActivity.EXTRA_FILE, previousLog_1.toString());
    intent.putExtra(GraphActivity.EXTRA_FILE_1, previousLog_2.toString());
    startActivity(intent);
}
```

Figure 5.7.1 Call

This Activity will display three graphs one for each accelerometer labelled Head, Neck and Ball graph. Each will display their individual readings from the accelerometers x, y and z axis and the three FSR readings, a total of 6 plots per graph. To create this graphical interface an external library was imported called MPAndroidChart [26]. The graph interface would look similar to the example shown below in figure 5.7.2 with a scroll view so each of the graphs could be viewed in a single Activity. The graphs would have labelled axis, titles, different coloured plots and a Legend. This library had the capability of implementing all these functions therefore it was chosen.



Figure 5.7.2 GraphActivity interface

First Maven repository was added to the build Gradle file in the application and the dependencies were implemented shown in figure 5.7.3. This allows the application to use of the functionality of these libraries. The activity_graph.xml file was created to setup the layout of this Activity. All three charts were inputted into a scroll view and the layout was set to landscape as opposed to the standard portrait. The graphs required were of type LineChart and all are labelled with their individual titles.

```
repositories {
    maven { url 'https://jitpack.io' }
}

dependencies {
    implementation 'com.github.PhillJay:MPAndroidChart:v3.1.0'
}
```

Figure 5.7.3 MPAndroidChart repository added to Gradle

This layout XML file is called in the onCreate method of GraphActivity. The files from the previous log are imported and the three graphs are created. The setTouchEnabled() method allows the user to zoom in to the graph at certain points of interest, which is enabled for each graph. The background is set to white and the colour of the plots are set to be different from each other. To read the files a BufferedReader is used see figure 5.7.4. It will read each line of the log files and parse each value in the array, to the array list of each plot. Each plot is defined by the timestamp on the x-axis and the voltage reading on the y-axis.

```

try {
    FileInputStream is = new FileInputStream(previousLog1);
    BufferedReader reader = new BufferedReader(new InputStreamReader(is));
    String line = reader.readLine();
    System.out.println(line);
    while (line != null) {
        line = reader.readLine();
        String[] values = line.split(" ");
        int timestamp = Integer.parseInt(values[0]);
        fsrLeft.add(new Entry(timestamp, Float.parseFloat(values[1])));
        fsrMiddle.add(new Entry(timestamp, Float.parseFloat(values[2])));
        fsrRight.add(new Entry(timestamp, Float.parseFloat(values[3])));
        xpin.add(new Entry(timestamp, Float.parseFloat(values[4])));
        ypin.add(new Entry(timestamp, Float.parseFloat(values[5])));
        zpin.add(new Entry(timestamp, Float.parseFloat(values[6])));
        ax.add(new Entry(timestamp, Float.parseFloat(values[7])));
        ay.add(new Entry(timestamp, Float.parseFloat(values[8])));
        az.add(new Entry(timestamp, Float.parseFloat(values[9])));
    }
}

```

Figure 5.7.4 Parse Log files

The plot of data are added to another array list of plots depending on which graph is being configured. All this data are finally added to the graph using the setData command and inputting the array list of plots. This code functionality is shown in figure 5.7.5. Once this is completed the application is ready for testing and has the full capabilities required from the start including scanning for BLE devices, connecting to them, transmitting signals and data between them, saving the data to a filing system and analysing the waveform of the recordings with graphs.

```

ArrayList<ILineDataSet> HeadDataSet = new ArrayList<>();
HeadDataSet.add(setFSRL); // add the data sets
HeadDataSet.add(setFSRM);
HeadDataSet.add(setFSRR);
HeadDataSet.add(setHeadX); // add the data sets
HeadDataSet.add(setHeadY);
HeadDataSet.add(setHeadZ);

// create a data object with the data sets
LineData HeadData = new LineData(HeadDataSet);

// set data
HEADChart.setData(HeadData);

```

Figure 5.7.5 Add datasets to LineChart

6 MACHINE LEARNING

6.1 The Theory

Artificial Intelligence is currently at the forefront of software development. It is the application of computer systems performing tasks which prior to this required human intelligence. A large subsection of this development is Machine Learning which is the study of teaching computers to do certain tasks without being explicitly programmed that overtime autonomously improve their results from real-world observations and testing. This technology can be used in multiple fields across all branches of work including engineering, finance, marketing and data security to name but a few. Machine Learning uses algorithms which are trained from recorded data for a particular system. The performance of an algorithm depicts how accurately it defines this algorithm with comparison to new data. The larger the data sets the less accurate these algorithms become as it is much harder to define trends and patterns between different recordings. To overcome this issue a concept called Deep Learning is implemented which makes sure the performance accuracy keeps improving at the same rate the data size is increase and therefore avoiding inconsistent results. Deep Learning uses artificial neural networks modelled like the human brain capable of learning and making decisions intelligently [27]. These neural networks use multiple algorithms to interpret sensory data through the output. Once the original algorithm is classified new data are imported and the algorithm learns constantly updating itself.

There are four overhead types of algorithms Supervised, Unsupervised, Semi-Supervised and Reinforcement. In each type there are many different kinds of algorithms used for different purposes and accuracy needs. Supervised learning uses training, test and validation datasets to split the data. The training dataset is data that is labelled with the corresponding output which the algorithm learns from. The validation sample set is used as an unbiased evaluation of a model fit on the training dataset which tunes the model's parameters. The test data are used by the algorithm created from the training dataset and compares the results found in terms of accuracy, this can be used to compare different types of algorithms. There are two types of Supervised learning namely classification and regression. Classification algorithms are used to classify data with predefined labels whereas Regression algorithms are used with numeric variables and compares the relationships between them. Supervised learning has many example algorithms such as decision trees, LSTM and Random Forests.

Unsupervised learning uses unlabelled data to cluster similar readings together generating groups for different clusters. Examples include K-means clustering and neural networks. Semi-Supervised algorithm uses a combination of the first two groups using labelled and unlabelled data to train the algorithm which can be useful as it balances the expensive cost of collecting labelled data and the inaccuracy of using all unlabelled data creating optimal results in certain cases. Finally,

Reinforcement learning works with the principals of behaviour and rewards, learning to maximise rewards for a given task. Based on the decisions the algorithm makes it will reward or punish the machine therefore learning how to correct the decisions it makes and improving on them in the future. This can be used, for example, in the game of Chess where a robot plays a human, if the robot's piece gets destroyed because of a movement it is punished, but if it destroys an opponent's piece it is rewarded therefore learning to implement rewarding behaviour and win a game through multiple tests.

All of these algorithms, types of learning and classifications can be incorporated into systems to provide evaluation. For this project, the purpose of machine learning is to improve analysis and the accuracy of these results. The implementation of this is described in detail in section 6.2.

6.2 Implementation

This project collects data from sensory objects generated in a time series format. Each header can be classified with a rating depending on certain factors such as position and force of each event. To incorporate machine learning analysis classified labelled data are needed the method of collecting of this is described in section 9.5. Once this data are collected the machine learning algorithm analysis can take place. The data flow operation of the project is shown below in figure 6.2.1, from collecting sensory data, transmit this data over Bluetooth, save the recorded data locally to csv files and finally analyse this data using graphing and machine learning to rate the quality of the header.

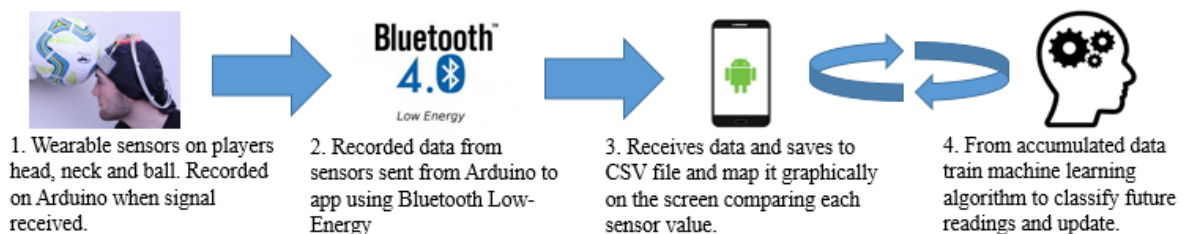


Figure 6.2.1 Dataflow Diagram of project

The dataset received from the sensors is multivariate time-series data. This is a collection of multiple output data collected at constant time intervals. These can be analysed to determine a long-term trend to be able to predict future readings. There are two main algorithms which can be used with such data Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) [28].

RNN

Standard feed-forward neural network are used for pattern recognition but for sequential data handling RNN networks are used as it can remember inputs and make predictions with them. To generate an RNN first the data are trained similar to with a feed-forward neural network that uses one

input at a time and produces a single result but the difference is the previous results aren't independent of each other. Each time a set of data are passed through and outputted the information gained is passed to the next dataset to be trained. See figure 6.2.2 [29] below which shows the process of training the data. The 'X' values are the input, the 'Y' values are the output of each layer and the 'h' values represent the information passed to the next layer. In this way it trains from previous data and learns to improve at each step therefore producing results that can be used to classify time-series data. RNN work very well with short term dependencies but for remembering the context of inputted arguments it fails. This is due to a problem called the vanishing gradient which reduces accuracy of the classification. The cause of this issue stems from the use of the same weights in the algorithm that during each time step calculate the output which is also done during back propagation. Moving backwards causes the error to increase thus for time-series there is an issue as the algorithm will only remember the last inputted data not the whole recording. To overcome this problem there are more powerful models which can use this structure but fix this major issue, for example LSTM.

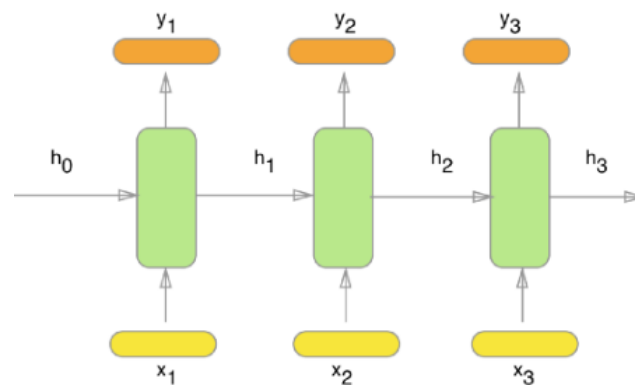


Figure 6.2.2 Dataflow of RNN

LSTM

LSTMs build from the RNN algorithm with modifications to make it applicable to time-series problems. Instead of completely transforming the function for each new dataset making it impossible to differentiate important information like in RNN, LSTM uses cell states which determine if the information is important or not to the algorithm. The algorithm has three dependencies the previous cell state, the previous state output and the current time step. The architecture of a LSTM algorithm is shown in figure 6.2.3 [29]. Different memory blocks are called cells which are the rectangle objects shown, the cell and hidden state are inputted to the next dataset which can be the two streams inputted from 'A' and outputted to 'B'. The memory blocks have to remember and manipulate memory using three Gates all serving different purposes namely Forget, Input and Output Gates. The Forget Gate labelled 'F' in the diagram is responsible to removing irrelevant information for the cell state. This Gate multiplies the previous output and current input using weight metrics decides if the information is similar and discards

it if not. The Input Gate labelled 'I' which is used to add information to the cell state if it is deemed important. There is a sigmoid filter similar to the Forget Gate used with created vector function 'tanh' then it is added to the cell state using the addition operator. The Output Gate chooses the useful information from the current cell state and outputs it. This works similarly to the Input gate using a sigmoid filter and vector function 'tanh' in multiplication which outputs this time-step and inputs this value into the next time-step.

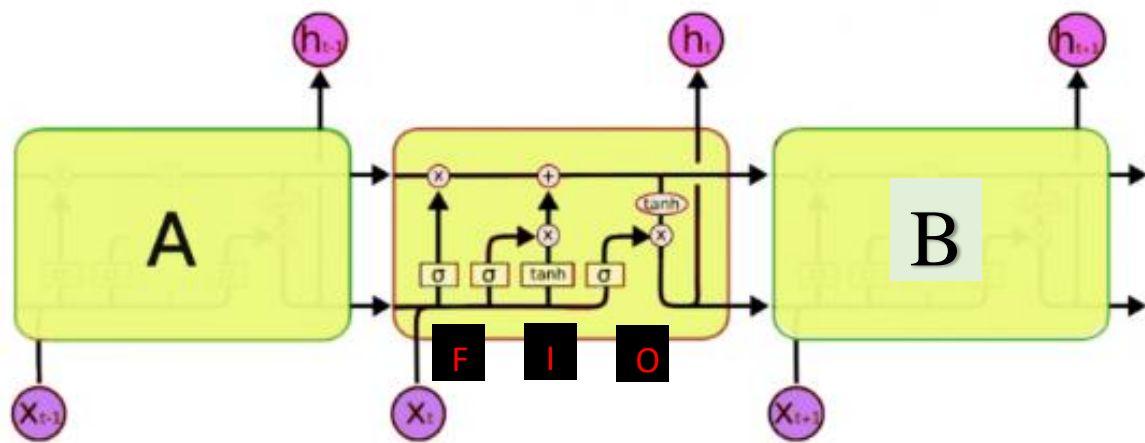


Figure 6.2.3 LSTM Architecture Diagram

The one challenge with LSTM is the difficulty to train them but with sufficient resources in testing this isn't an issue. To implement this algorithm a software development tools must be used that can import the recorded data, analyse it and produce an algorithm which can classify test runs. TensorFlow is a free software library which can be used with Python data prediction and differentiable programming for numerous applications, especially useful with machine learning tasks. As this file had to be run through the mobile application virtual machines couldn't be used to process the python files therefore the command prompt was used to run the code which became quite versatile when importing this algorithm into the application as discussed in section 6.6. Python has particularly useful tools for importing CSV files which is another reason this language was chosen to implement this task but before this could be done the CSV files had to be arranged correctly.

6.3 Preparing data

To import this large volume of CSV files into the python script recorded data needs to be combined, discarded and normalised. There are two CSV files created at each heading event, one from the Head Arduino readings and one from the Ball Arduino readings, which need to be combined and put into a new folder which can be imported directly without any editing. To do so a folder called FYPData was created, when called a method FYP searches in this folder to determine how many CSV files it contains. It then names the file "HeadData" along with the count plus one so the files incremented

upwards. This method is shown in figure 6.3.1 below. To remove irrelevant data a threshold of 0.5 Volt (approximately one sixth of the max reading) is set, which at least one FSR peak values must hit, before a new file is created. If one of the FSR values don't achieve this threshold voltage, then the contact between the ball and head is so low that the recording doesn't classify as a heading event. This is implanted using a maximum peak value for each FSR when being parsed and being compared with the threshold once completed. This maximum voltage is also used to section off the important data needed from each recording which will normalise each data frame making them comparable for the algorithm. The index of first peak FSR reading to occur above the threshold is recorded and fifteen readings before and after are added to a newly created file called head data.

```
File[] list = folder.listFiles();
int count = 0;
for (File f: list){
    String filename = f.getName();
    if (filename.endsWith(".csv"))
        count++;
}
File output = new File( pathname: folder.getPath() + "/" + name + (count + 1) + ".csv");
output.createNewFile();
return output;
```

Figure 6.3.1 FYPData folder created

Using two array lists, one for each dataset, the values are added and combined into one single file between the start and end ranges. A FileWriter is used with the Concat predefined method to concatenate each string into one single CSV file separated by commas to fill each Excel box. Once this is completed the new file is created and is saved in the FYPData folder which can be imported directly to the python script.

```
try {
    File headDataComplete = FileOperation.FYP();
    FileWriter writer = new FileWriter(headDataComplete);
    writer.append(titlesComplete);

    for (int j = start; j <= end; j++) {
        String newLine = new String();

        for (Float f : data_1.get(j)) {
            newLine = newLine.concat(f.toString() + ",");
        }

        for (Float f : data_2.get(j)) {
            newLine = newLine.concat(f.toString() + ",");
        }

        newLine = newLine.concat("\n");

        writer.append(newLine);
    }

    writer.flush();
    writer.close();
}
```

Figure 6.3.2 Approach to Combine Datasets

One extra feature that was added to the GraphActivity in the java code was limit-lines which show the X-axis points where the recording is sectioned to give a more visual representation of the important part of the dataset to the user. An example of this is shown below in figure 6.3.3.

The one issue with this solution is, if the head impacts in less than 600ms from the start, or if the ‘STOP’ button is pressed in less than 600ms after impact, the reading is cut short and causes a problem of different size samples. This can be avoided by introducing a minimum of a one second delay between the header and the Start/Stop action, which gives more consistent data.

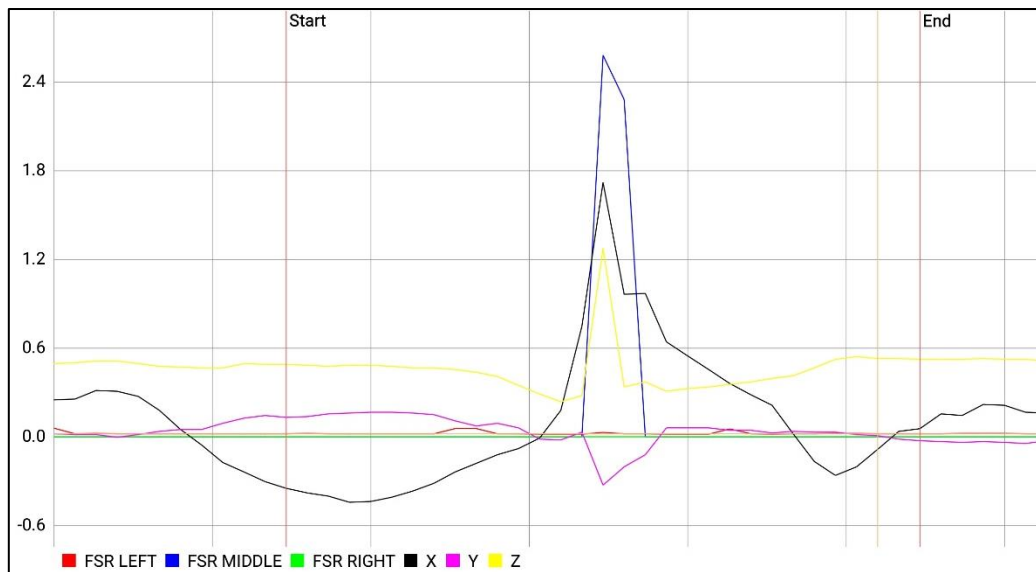


Figure 6.3.3 Start and End Limit Lines in GraphActivity

6.4 Designing Algorithm

To implement an LSTM algorithm first the data set of one hundred and eighty headers, is imported using Pandas, which is a library used for data manipulation and analysis. The data path is set to the folder which contains the CSV files, then a list of samples is created called ‘sequences’. A loop runs through all the files in the folder and reads them using a Pandas import. Each CSV value is added to a sample in the ‘sequences’ list. Figure 6.4.1 below shows the implementation of this. Each sample is now a member of the list, each timestamp is a timestep in each member and each sensory reading is a feature of each timestep. The classifications of each sample are stored in a separate CSV file, which is imported as the target values and each member is connected to its associated sample, from its reference index.

```

path = 'FYData/HeadData'
sequences = list()
for i in range(1,180):
    file_path = path + str(i)+ '.csv'
    print(file_path)
    df = pd.read_csv(file_path, header=0)
    values = df.values
    sequences.append(values)

```

Figure 6.4.1 Panadas import CSV files

The data is then split into three subsets training, validation and test. To do this a split ratio of the samples must be chosen. If the split is too high in favour of training, the model can become too specific and complex to the data and can cause problems of overfitting which will lead to poor performance when tested [30]. K-fold cross validation can help to eliminate this problem by shuffling the data and testing the trained model on many cases and comparing the accuracy of the output. If there are too little training data then underfitting can occur, where a model can't model the training data or classify new data, which is not a suitable model. A ratio of 80:20 for training to test is chosen and that training data it is split into the same ratio of 80:20, with training and validation sets. These ratios give optimum results for first time testing but can be easily edited if the model isn't outputting high results. The split ratio in the script is set to 0.8 and sectioned using the Numpy library. The index of the data is added to the training set, from the first member of the array to the member at the split ratio of the index. The rest of the samples are added to the test dataset which is twenty percent of the total. The same is then done with the training set and with the target data. The functionality of this is shown in figure 6.4.2.

```

split_ratio = 0.8

trainingval = sequences[0:int(np.ceil(len(sequences)*split_ratio))]

test = sequences[int(np.ceil(len(sequences)*split_ratio)):]
train = trainingval[0:int(np.ceil(len(trainingval)*split_ratio))]
validation = trainingval[int(np.ceil(len(trainingval)*split_ratio)):]

```

Figure 6.4.2 Split data into training, validate and test sets

A sequential model is created in which the LSTM import is added defining the size and shape of model and array. This is flattened to reduce the runtime and then the dense layer of three is added to signify the three output classifications. The creation of this is shown in figure 6.4.2 and the output of the models summary is displayed in figure 6.4.4.

```

model = Sequential()
model.add(LSTM(256, input_shape=(31, 14)))
model.add(Flatten())
model.add(Dense(3, activation='sigmoid'))
model.summary()

```

Figure 6.4.3 Create LSTM Model

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 256)	277504
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 3)	771
Total params: 278,275		
Trainable params: 278,275		
Non-trainable params: 0		

Figure 6.4.4 LSTM Model Summary

The datasets can be added to the model but first the learning rate is set to 0.001. This a hyperparameter which sets the rate that new sample's information is added to the model. The learning rate is low as the data fluctuates considerably and the model must learn slowly or it will be over-trained for a specific sample. The Pickle library is used to save a model, which is loaded and tested against a new sample for classification, in section 6.5. For multi-label classification a there is a loss called 'categorical_crossentropy' which uses a function called a 'softmax', which condenses a vector to the range 0-1, with all elements adding up to a total of 1 and combines this with cross-entropy loss. These compare the output classifiers and set the gradient weights, to calculate the error of classification and to determine an accuracy by comparing the training with the validation data. The model is then fit with the datasets and these parameters, an epochs of 200 and batch size of 128. Epochs in a neural network are the amount of runs through the network with the batch size of 128 samples. This outputs an accuracy for each increase in epochs.

Figure 6.4.5 shows the fitting of the datasets to the LSTM.

```

adam = Adam(lr=0.001)
chk = ModelCheckpoint('best_model.pkl', monitor='val_acc', save_best_only=True, mode='max', verbose=1)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
model.fit(train, train_target, epochs=200, batch_size=128, validation_data = (validation, validation_target))

```

Figure 6.4.5 Fit the model with Training dataset and Validate results

Finally, the 'best_model.pkl' is loaded to predict the accuracy of the model using the test dataset. This predicts the test samples without their labels and then compares this to the actual classification. The precision increases for each prediction that was correct and an overall accuracy is given. For the first test an accuracy of 65.7% was outputted. The rest of the results are explained in

detail in section 6.5 and an evaluation of the model is made. Figure 6.4.6 below shows the accuracy predicting function.

```
model = load_model('best_model.pkl')

from sklearn.metrics import accuracy_score
test_preds = model.predict_classes(test)
accuracy_score(test_target, test_preds)
```

Figure 6.4.6 Accuracy of Model using Test dataset

6.5 Results and Output

The validation dataset can be used to produce a performance metrics called learning curves. This compares the accuracy of the data with the increase in the epochs and gives an indication of how the model is generalising the data. The epoch value was incremented in steps from one, to one hundred and eighty, at each step the trained model was validated and the accuracy was outputted. These values were plotted on a learning curve, that compares accuracy in percent on the Y-axis to data size/epochs on the X-axis. This graph is plotted and displayed below in figure 6.5.1. The initial accuracy of the first sample is 50%, but this increases to the maximum accuracy of 78.6%, proving that the model is learning at an increase rate with a larger dataset. To further improve this accuracy more samples would need to be recorded. With the increase in accuracy there is a higher chance of the model correctly classifying heading event and if this was implemented in the real-world a much higher percent would be necessary to avoid misclassification.

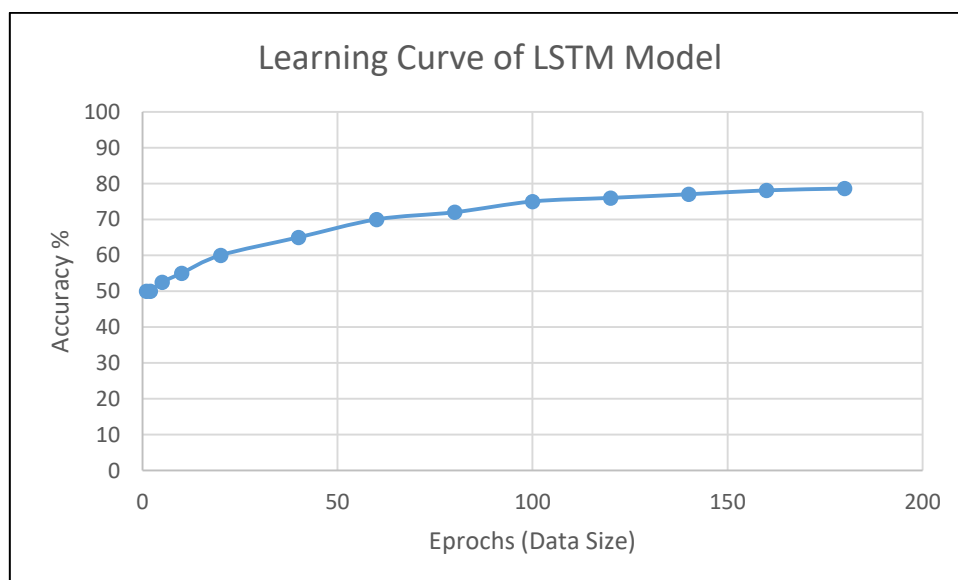


Figure 6.5.1 Learning Curve of LSTM Model

The other test the model uses is with the test data. This measures the overall accuracy of the model, which was 78%, and indicates, from the trained and validated algorithm created, that new data will be predicted to this level of precision. It is a similar statistic to the learning curve but is important to know how well new data is classified.

The final step was to classify a newly recorded CSV file from this loaded model. A second python script called `load_model.py` was created to implement this. This script loads the model as seen with the test data in the previous script and loads in the new CSV file. The types of classifications to output are mapped to the cross-entropy values created in the first model, these are labelled “Good - Middle”, “Bad - Left”, “Bad - Right” and “Poor Middle”. The model prediction function is used to predict the classification of the imported CSV file which outputs this string value. This second script shows that the model has been created and can be tested on new samples of incoming data. It was initially planned to import these scripts into the Android device again and classify the reading when the Analyse button is pressed, but this proved to be too time consuming so the idea was discarded. In section 9.4 the accuracy of the algorithm is compared to recorded and classified readings to see if the model is as precise in the real-world. To do any further testing the circuitry had to be secured, to avoid inconsistent data being transmitted from loose connections, this is examined and implemented in the preceding section.

7 SECURING CIRCUITRY

7.1 Overview

The system would need to be durable to withstand physically testing of hundreds of heading events. To withstand this level of testing, a wearable circuitry system had to be developed, that was robust, durable and flexible. In-order to secure the sensors and the measuring equipment to the head, a sports head mask was used. By attaching the equipment to the cap, it made the testing process a little easier to achieve, as the kit could be attached or removed as a single piece. There were three main sections to implement in securing this headgear system, FSR's, Strip Board Assembly and the Microprocessors. See figure 7.1.1 and 7.1.2 below which show front and side view of the head circuit layout. The FSRs are to the front, the strip board assembly is on the top and the white wire leads to the neck where the microprocessor is held in a pouch.

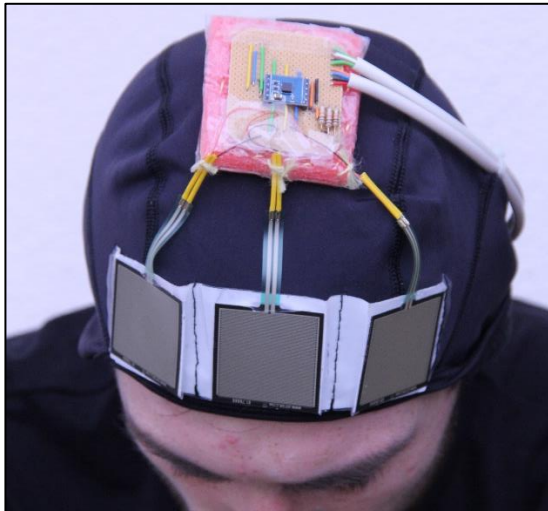


Figure 7.1.1 Frontal view of the full Circuit



Figure 7.1.2 Side view of the full Circuit

7.2 FSR – Force-Sensing Resistors

The FSR's are quite delicate and needed to be kept in their correct positions in order to get accurate measurement results. These were first attached to the cap using good quality double sided tape, however it was found that they came loose after a small amount of testing. Therefore to hold the FSRs in place more permanently it was decided to stitch the tape to the cap and together with the adhesive the attachment was made secure.

The solder tags, where each device is connected to the breadboard was very thin and if put under pressure would break. Some light gauged wire was pre-soldered and then was sweat soldered to the FSR tags. The distance from each of the FSRs to the Printed-Circuit Board (PCB) was measured and the wires were extended to fit to size. PVC sleeving covered the solder tags to offer further

protection. In figure 7.2.1 the FSRs are shown connected to the PCB. The yellow sleeving is visible in this image just above the FSRs.

The circuit for the FSR involved connecting red end to VCC and the other in parallel to GND and to a 10K resistor that goes to analogue pins. This uses a total of 3 analogue pins and the readings from these sensors are recorded to the microprocessor. To assemble this circuit a solderable breadboard was used which is explained in section 7.3.

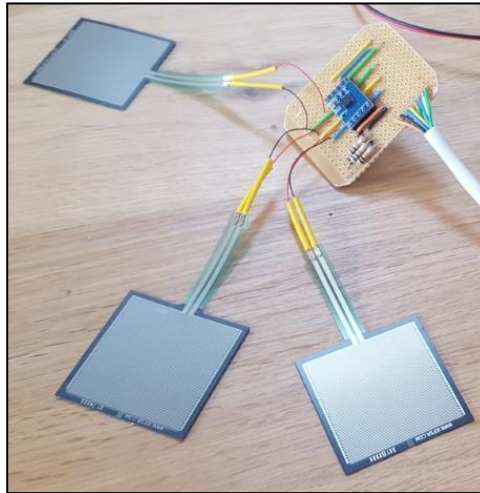


Figure 7.2.1 FSRs connected to Strip Board

7.3 Strip Board Assembly

The strip board is the main interconnecting device and also contains the analogue Accelerometer to measure the head movements. The layout was designed to be as compact as possible, but strong and robust to withstand the testing with multiple headers. The component pins and the connecting wires are folded down and soldered for extra strength. The interconnections are in accordance with the circuit diagram in 4.1.1. The tracks on the strip board, under the ADXL335 were split using a grinding wheel, this to enable a separate connection to each of the ten pins.

A piece of anti-static foam was placed under the strip board, to protect the pins and the components, and to facilitate attachment to the cap material. Again double sided tape was attached and stitched to secure the strip board assembly to the cap.

The heavy gauged wires extend the strip board to the Arduino, 2 wires for the power, 3 for the FSR output and 3 for the accelerometer output, a total of 8 wires. These wires are protected by white PVC sleeving, as shown in figure 7.3.1 on the right-hand side. This sleeving is attached to the cap, for extra reinforcement, using small cable ties.

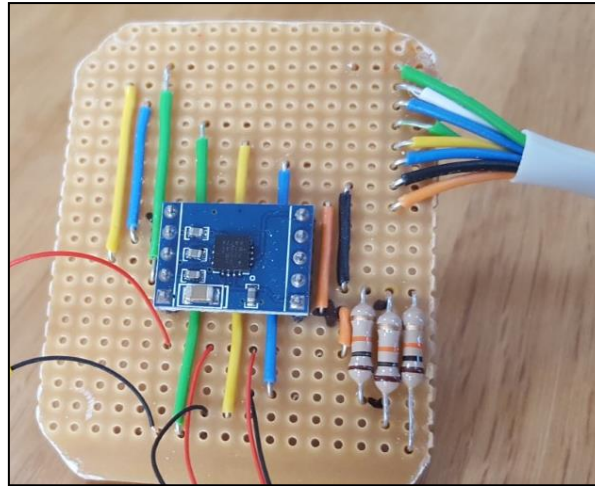


Figure 7.3.1 Strip Board Assembled Circuit

7.4 Microprocessor Placement

A small pouch was made and attached it to the cap using Velcro. This holds the Arduino in the correct position on the neck. The cable from the strip board assembly is extended to the Arduino and the wires were inserted into the slots as per the wiring diagram. On the ball there is a similar connection where a small pouch is attached to a netting that surrounds the ball. This netting is attached to a thread which extends to where the ball is hung from. This was tested at different heights to see the difference in ball movement. At first the ball was hung from a low roof with a short thread which gave inconsistent data due to collisions with external objects and the lack of manoeuvrability. To prevent this the thread attached to the ball was extend and was hung from a high bar, shown in figure 7.4.1, giving the ball more mobility without colliding with outside surfaces and leading to improved results.



Figure 7.4.1 Ball Circuit Setup

To power the circuit a PP3 connector was soldered to the wires of a small power plug which connects to a 9 volt battery for each microprocessor. This makes it easy to connect and disconnect the power to the Arduino. This connection is shown in figure 7.4.2 below. A special casing is used to prevent the microcontroller from damage and which is labelled to differentiate each device.

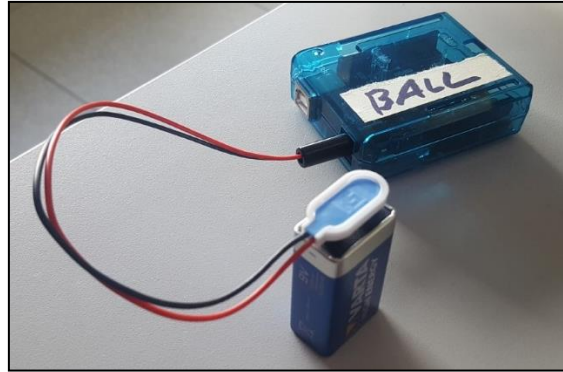


Figure 7.4.2 9V Battery PP3 connection

Once the assembly is complete, as shown in Figure 7.1.1, it is easily placed over the head as a single wearable piece. All the components are kept together and it reduces the risk of having problems with poor connections or component damage during rigorous testing.

8 TESTING

8.1 Accelerometer calibration test

Each axis on the accelerometer outputs a value that must be changed for each sensor as their range of values are different. The calibration was achieved by slowly rotating the accelerometer through 360-degree and noting the current value on the accelerometer for each axis. This was done for each axis and the maximum and minimum values were noted. Before the value is sent to the application a calculation is carried out in the microcontroller. Below is the equation for each axis. On a flat surface such as the top of the head the X-axis and Y-axis start at a value close to zero and the Z-axis will output a value of -1g (-9.81m/s^2) due to gravity. The formula used is similar to the ADC conversion using minimum and maximum input and output values which is coded in the Android app in the Map function shown in figure 5.6.3.

8.2 FSR Test

To test the FSR initial circuit was setup comparing the FSR 402 and 406. The voltage v forces were compared to see the sensitivity of each. The results found that both produced similar outputs but as the 406 is bigger physically it would be more useful for the project. It was also found, through this testing, that the FSR devices couldn't be used for measuring the actual force of impact, of the heading event, as the devices aren't sensitive enough. The conclusion was made that they could be used just as a switch, to show graphically where the impact occurs and comparing it to the axial accelerometer forces. The output of the FSRs was a simple ADC conversion from the inputted 0-1023 analogue reading to an output voltage that is supplied to the circuit of 0-3.3Volts.

8.3 Initial System Testing

Before the system was fully secure and setup initial testing was carried out to make sure the sensors were outputting correct values. This was first done printing statement outputs to the Android application console, which was useful for an indication, but was hard to read and compare. The graphing Activity was implemented and these tests were again carried out. This was a very good test as it showed a miscalculation of the FSR values which were incorrectly mapped between $\pm 3.3\text{V}$ like the accelerometer instead of 0-3.3V.

These tests also showed the conversion of the digital accelerometer values where ranging between 0-4V, as opposed to the desired $\pm 2\text{V}$. To solve this issue the code was retraced. First the readings were gathered from the Arduino microprocessors to make sure they were being recorded correctly over the range of $\pm 32,768$. This was found to be correctly outputted. This output was compared with the values received on the central node which were printed to the console. These were found to be

over the range of 0-65,560 which is exactly double the original value. The recordings were mapped over this region instead and outputted correctly. When the two bytes were added back to integers, in the transfer of these recordings over BLE the receiving end perceived them as all positive whole numbers and inverted the values causing the range to become positive for all readings. Once this issue was resolved the whole system was ready for testing.

9 STUDY AND RESULTS

To develop a wearable football heading system which can analysis and interpret data visually was an important objective when the project was first defined. The aim of the study was to be able to analyse a heading event and develop a platform which would feedback information to the user. Using sensory devices on the head and ball the movement and forces of a heading event were to be investigated and determine what leads to a good header. The pattern the accelerometer values of each test are comparable and the most important sections of each frame are found. From these results, conclusions can be made with regard to the severity of an impact which can be used to inform players, coaches and trainers about the precautions necessary when heading a football.

Over a two day period, on the 28th and the 29th of March 2019, a study was carried out to test the multiple heading events in room 3002 in the Alice Perry Engineering Building NUI Galway. The tests were to be carried out by myself, Ronan Murphy as the event was captured through the use of a smartphone therefore making it possible to head the ball and record the event simultaneously not requiring any extra participants. To set the event up first the ball had to be secured to an iron beam which runs across the top of the ceiling, using a knotted rope. Once this was in place the wires from the head mask were inserted into Head Arduino and a battery supply was connected to both Arduinos. Each microcontroller was then placed and secured with a cord in their pouches on the head and ball. The devices were now turned on and the mobile phone application was used for connection using BLE. Once connected the heading event can commence and the START button on the application is pressed. The ball is then headed and after a slight delay the STOP button is then pressed to stop recording any more readings. The Analyse button can then be pressed which graphs the recorded data and can be used to determine the classification of a header. Any irrelevant data are removed using a threshold FSR voltage explained in section 6.3. Each header is classified depending on where the ball hits the head and the movement leading up to it. This is divided into four sub-sections good-centre, poor-centre, bad-right, bad-left which are documented in training to be inputted into the machine learning algorithm. There were fifty headers for each sub-section completed, a total of two hundred, in a random order so splitting the data when producing the algorithm wouldn't give inconsistent results. Before classifying and concluding on outcomes of a header key points of interest must be highlighted and examined from the graphs.

9.1 Accelerometer Orientation

The directions that the accelerometers are initially set to determine the analogue readings they output therefore it is very important that these are kept constant to make each graph comparable. The Head Arduino was fixed to the strip board and placed on top of the user's head therefore keeping the X and Y axes stationary near zero and the Z-axis at approximately -1 Volt due to gravity. The neck and

ball accelerometers are tilted 90 degrees on their side to fit into the pouches. As they are different types of accelerometer, to the ADXL335 on the Head, they have different orientation of the axes. To keep these starting positions constant at the beginning of each test the pouches are checked to make sure the microprocessors are oriented the correct way. Through the motion of the header these values change quite significantly and the movement of the head, ball and neck can be mapped and analysed at the points of change. These axes values all return to their original positions once the heading event is complete.

9.2 The Heading Event

The heading event can be split into three different sections before, during and after impact and each can be compared using key events from the sensory devices. In figure 9.2.1, 9.2.2 and 9.2.3 three graphs of a sample heading event are shown, Head, Neck and Ball graph respectively. The subsections that follow explain in detail the heading event, using comparisons with the three figures below.

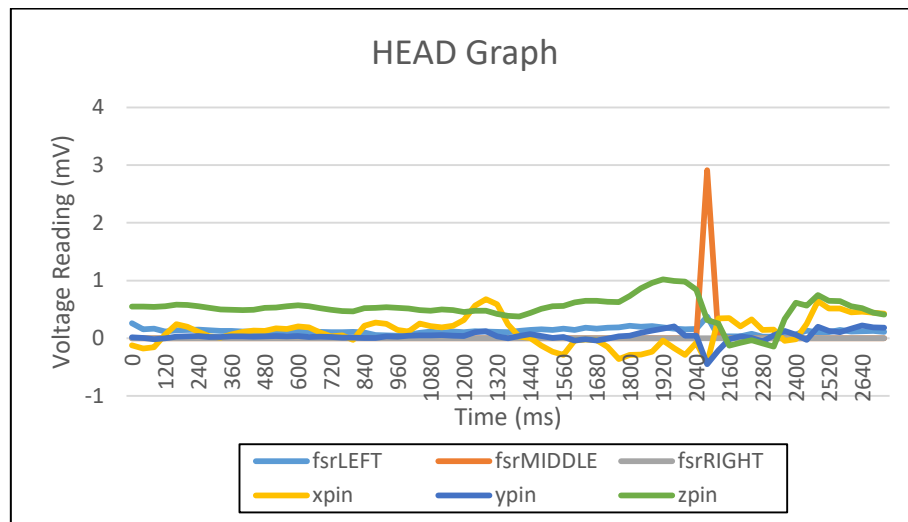


Figure 9.2.1 Head Graph

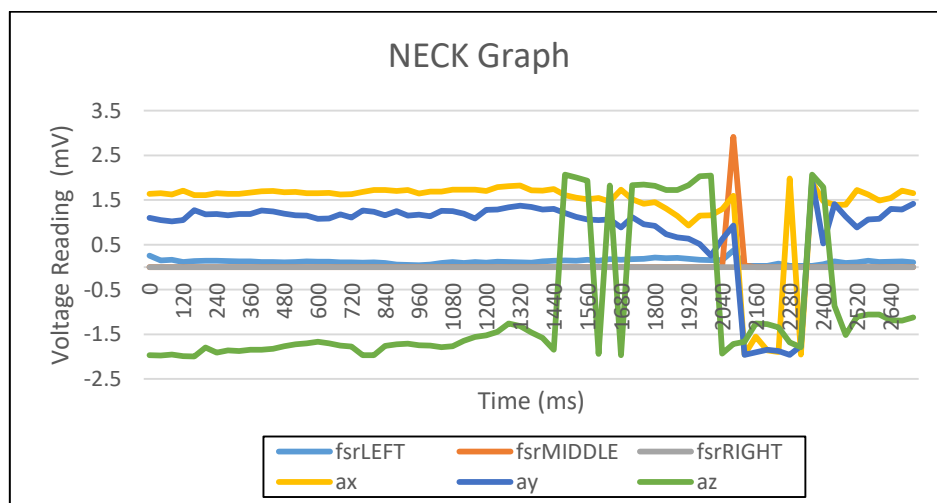


Figure 9.2.2 Neck Graph

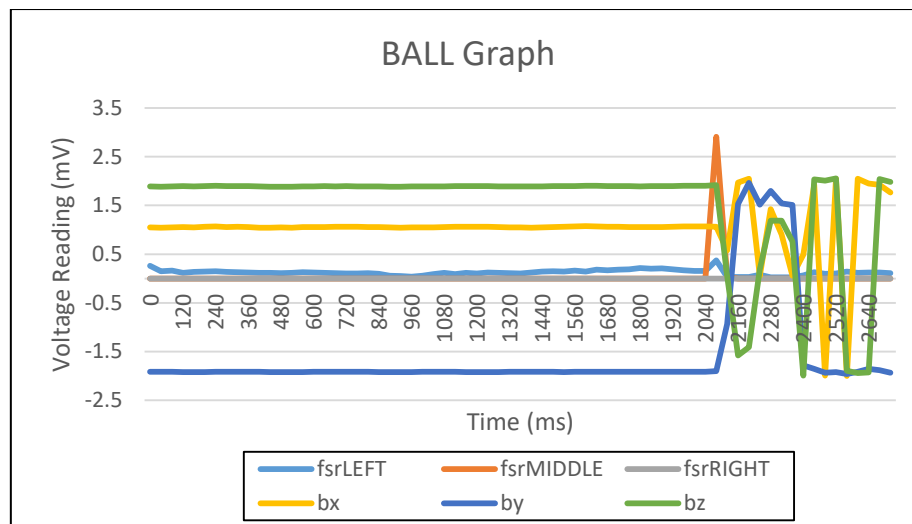


Figure 9.2.3 Ball Graph

Before

The football begins at rest, in the ball netting which is hung from the cord connected to the ceiling. The BALL graph won't have movement in any direction before impact and there is no information to be gained from it. As seen in the HEAD graph, all FSRs will have no change before impact and will remain at zero as there is no force being applied to them. The accelerometer values in each direction remain constant until just prior to the heading event. As the head leans back and moves to head the ball there is a positive increase in the Z direction and reaches its peak just before impact. The X direction has a steady negative movement until just before impact, where a sharp increase that indicates the heads rotation to hit the football. The NECK graph shows the most movement prior to the impact, this is mainly due to the neck being the most active muscle used in a heading event. As seen in figure 9.2.4 the Neck rotates around an axis from leaning back to forward. The first change in NECK graph is in the Z direction, which is due to the lean back phase where there are three sharp changes from negative to positive, as the Arduino pouch isn't fully fixed so upon rotation it bounces off the neck. As the neck moves forward to strike the football there is a steady decline in both the X and Y directions which indicates the change in movement of the neck. The graphs give a very visual representation of the movement and any small changes can be picked up quite easily which shows the accuracy and usefulness of the sensors.

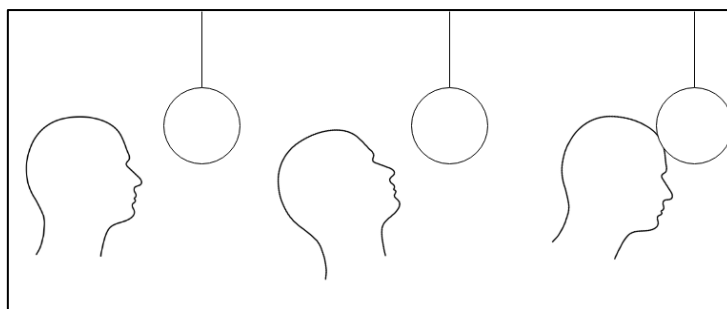


Figure 9.2.4 Head and Ball Action before Impact

The Impact

The Impact between the head and ball is the most important part to be gained from the analysis as where the force and position of the collision are vital in understanding of the negative impacts related to the heading event. The FSR reading will explain the position the header hits by determining which FSR has received an impact. In this case the central FSR has been hit with one peak of around 3 Volts, there is also a very slight increase in the left FSR but as it is such a small value it is deemed insignificant. As the header is central this reduces the harmful effects of heading a football dramatically. The other concern is the football colliding with the lower forehead or higher up, closer to the top of the head, the second leading to a potentially dangerous heading event. This can be determined by the change in accelerometer values. Before and after impact there is a change from positive to negative in direction of the accelerometer. This impact can be seen as the transition period where the change is occurring and the midpoint between the new and old values.

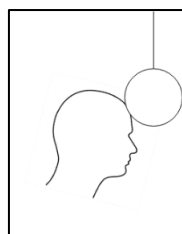


Figure 9.2.5 Collision between Head and Ball

After

After the peak in the FSR readings there is a considerable amount of change in the accelerometer readings in three graphs. The head and neck will follow through after hitting the ball before returning to their original position this movement is portrayed in figure 9.2.6 below. The HEAD graph shows a dip in the X, Y and Z direction as the head moves forward before returning to the original position. The NECK graph has a much sharper and larger decrease in all axes and the X and Z directions has another sharp recoil, symbolising the pouch bouncing off the neck, before all axes returning to their initial readings. Although the pouch was secure and didn't allow for much movement a more stable

pouch could be used in future to reduce any recoil or bouncing effects. The BALL graph has the most movement of the three in this stage of the heading event after being stationary previous to this. As the football moves to its maximum height quickly, as seen in the final image of figure 9.2.6, the Y axis has a sharp increase, a slight delay at the peak and then another sharp decrease. The X and Z axes change quite erratically as the ball spins mid-air as it rises and falls, causing sharp changes in these directions. After the ball has returned to stationary the STOP button is pressed and the heading event is completed. This heading event would be categorised as a good header because of the positioning and movement throughout. The next stage of analysis is to determine the different categories a heading event can be classified as.

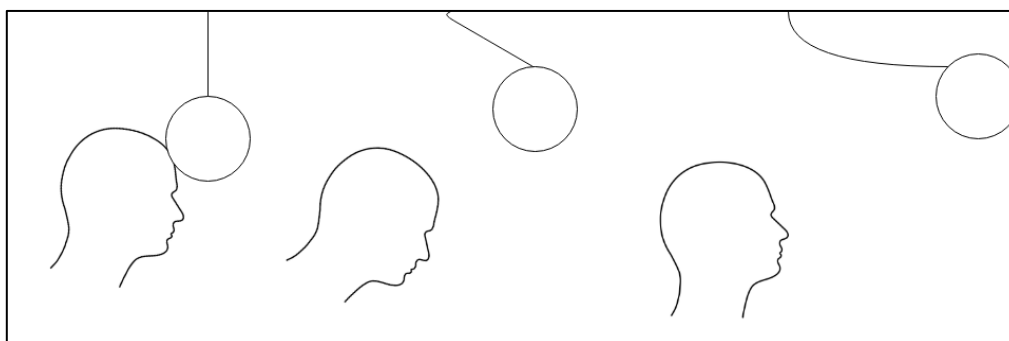


Figure 9.2.6 Head and Ball Action after Impact

9.3 Conclusions from Graph

As seen in section 9.2 each part of the three graphs can be analysed and understood from the change in direction of the different axes. To categorise each heading event there are certain attributes that can be analysed in the graphs. There were many options for how they could be split but the most significant were chosen which dividing the headers into four classifications Central Low Risk, Central Potentially Harmful, Left Dangerous Event and Right Dangerous Event, which are all described in detail below. Each event displays their three graphs with only the window of useful information displayed cut as described in section 6.3, this outputs a zoomed-in view into the major changes of the recordings.

Central Low Risk

This is the category for a good header classified only if the ball collides with the central FSR, the neck moves correctly and the ball moves straight with little spin. The head, neck and ball graphs are included below for the cut section in figures 9.3.1, 9.3.2 and 9.3.3 respectively. This example is very similar to the one described in section 9.2 except the neck graph doesn't include any sharp changes in the Z direction prior to the impact making it a better representation of a good header. In figure 9.3.3 the BALL Graph there are no sharp changes in the X and Z directions as seen in the previous example, this

shows the football doesn't spin after impact which is another indicator of a good header. All these factors lead to the below example being classified as a Central Low Risk heading event.

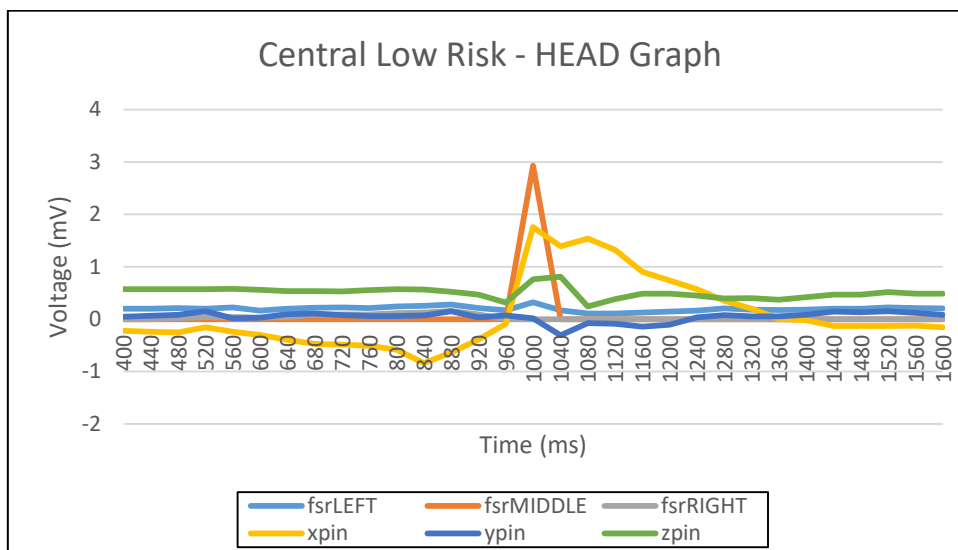


Figure 9.3.1 Central Low Risk Head Graph

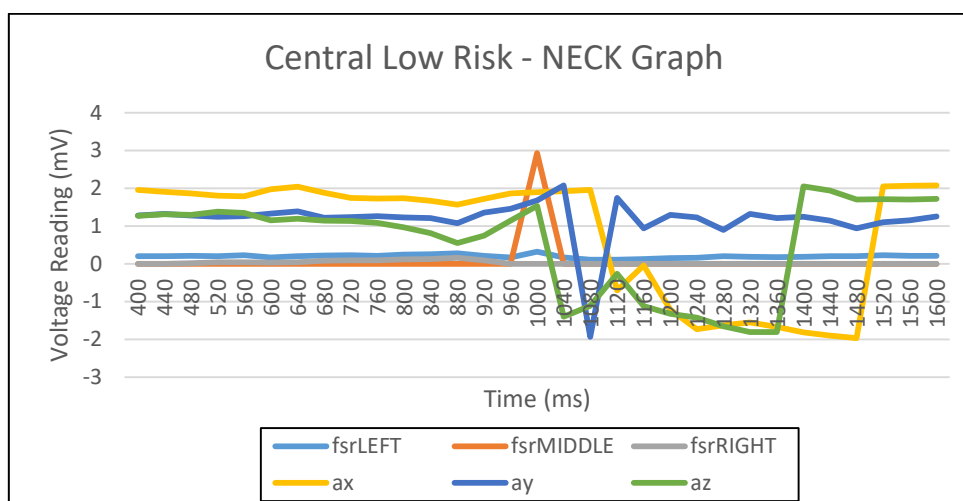


Figure 9.3.2 Central Low Risk Neck Graph

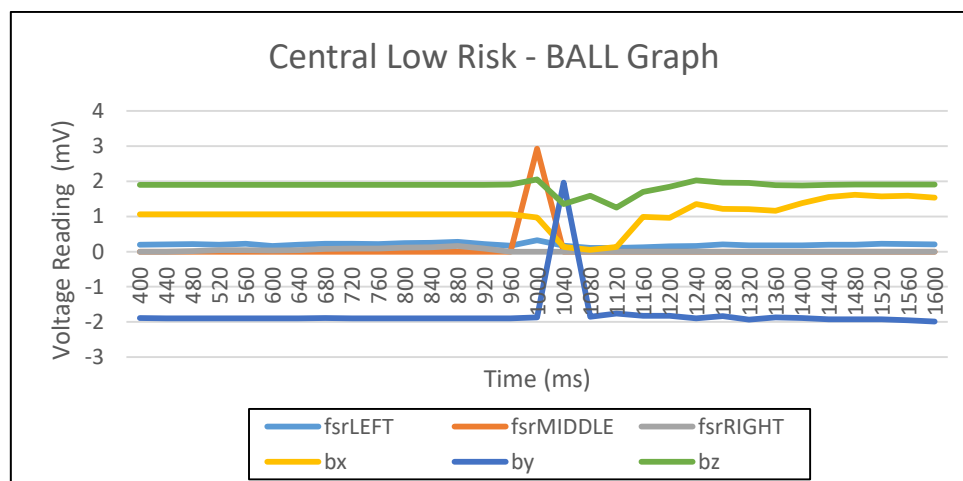


Figure 9.3.3 Central Low Risk Ball Graph

Central Potentially Harmful

When a heading event is central but has other attributes not linked to a good event such as impacting with multiple FSRs, no neck movement, excessive ball spin or impacting with top of head instead of forehead the header is classified as a Central Potentially Harmful event. The example below shown in figures 9.3.4, 9.3.5 and 9.3.6 display the head, neck and ball graphs for a Central Potentially Harmful header for a multitude of reasons. In all three graphs the FSR left and centre are almost equal proving that the header was off centre which is one key factor in its classification. The NECK Graph displays quite unusual X direction towards the start of the reading which is due to the movement of the user's body being off balance and not in correct position rather than the bad heading event which is a misleading reading. The BALL Graph outputs some spin in the X and Z axes due to the football being hit from a sliced side angle with use of the central and left FSR. There is erratic movement in the X and Y axes of figure 9.3.5 also due to the pouch bouncing. The contribution of these three negative effects all give weight to creating a second category for central heading events and labelling this header as Central Potentially Harmful. If the collision was further to the left or right there is a need for more categories as these regions of the skull have softer tissue and impact can cause more significant damage.

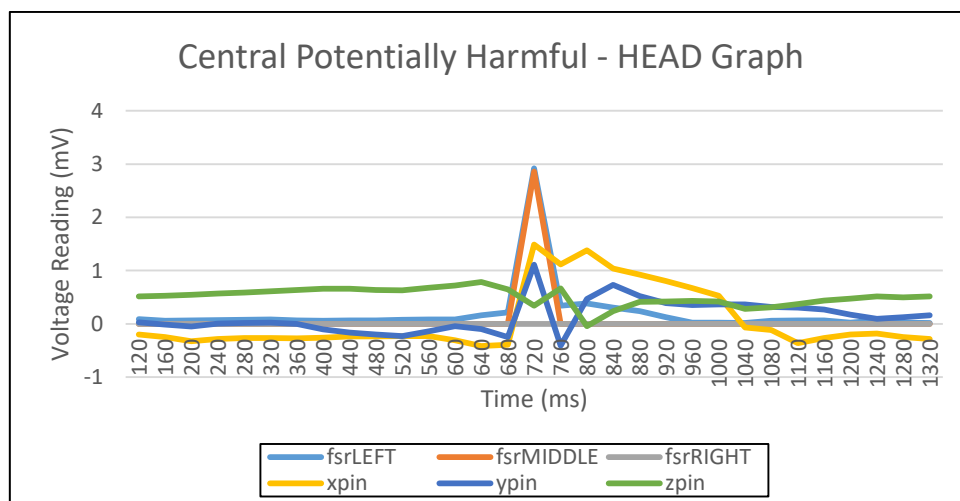


Figure 9.3.4 Central Potentially Harmful Head Graph

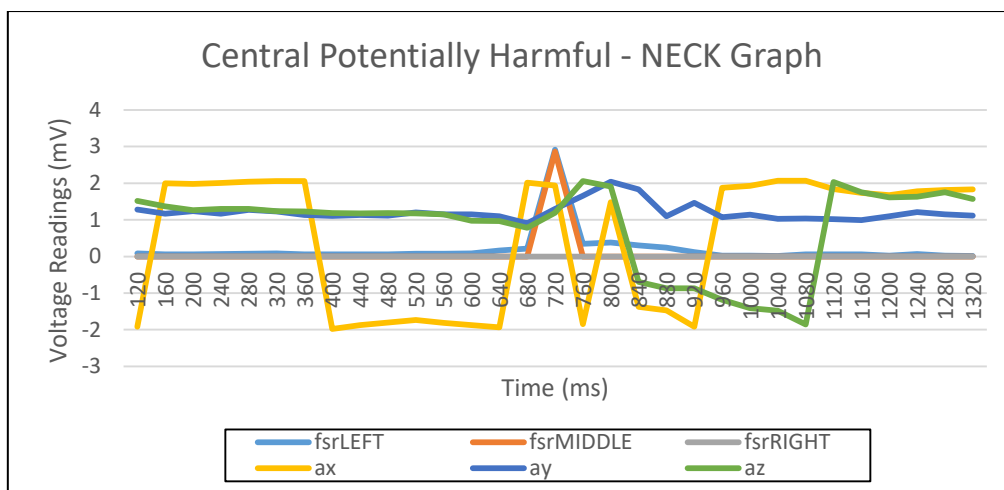


Figure 9.3.5 Central Potentially Harmful Neck Graph

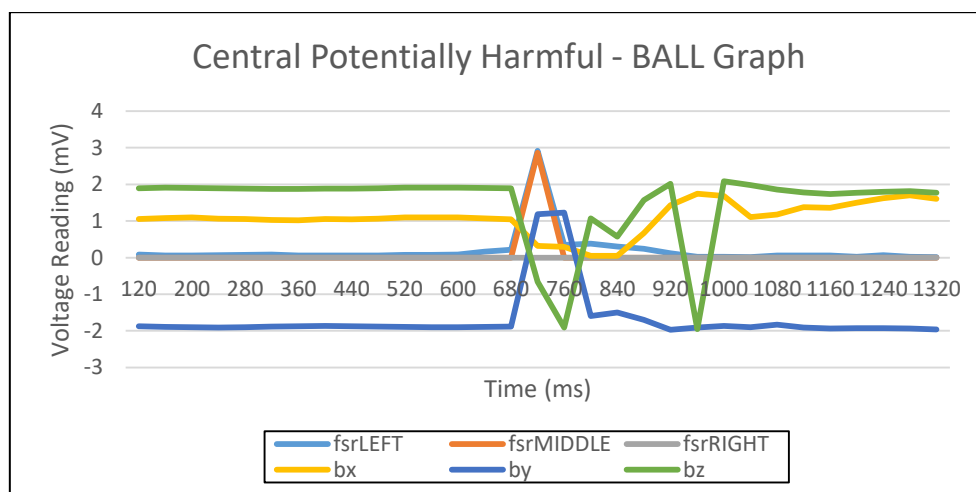


Figure 9.3.6 Central Potentially Harmful Ball Graph

Left Dangerous

If the header is predominantly activation the leftmost FSR it raises concern as to the football impacting with a weaker part of the skull endangering the player which results in a Left Dangerous classification being given. Apart from the activation of only the left FSR the movement of the three graphs shown in figure 9.3.7, 9.3.8 and 9.3.9 below are standard and would be classified as a good header if the impact was central. The only harmful indicators are the spinning of the football in the BALL graph and the bouncing of the pouch in the NECK Graph. The classification of this Left Dangerous event is due to the dangers of the football impacting with the left side of the temple which can have more detrimental effects as it is a softer part of the skull.

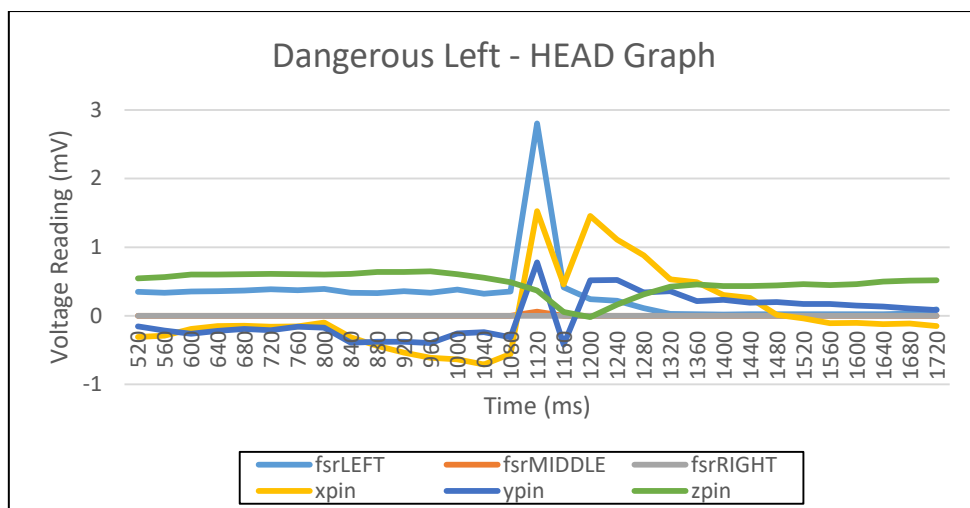


Figure 9.3.7 Dangerous Left Head Graph

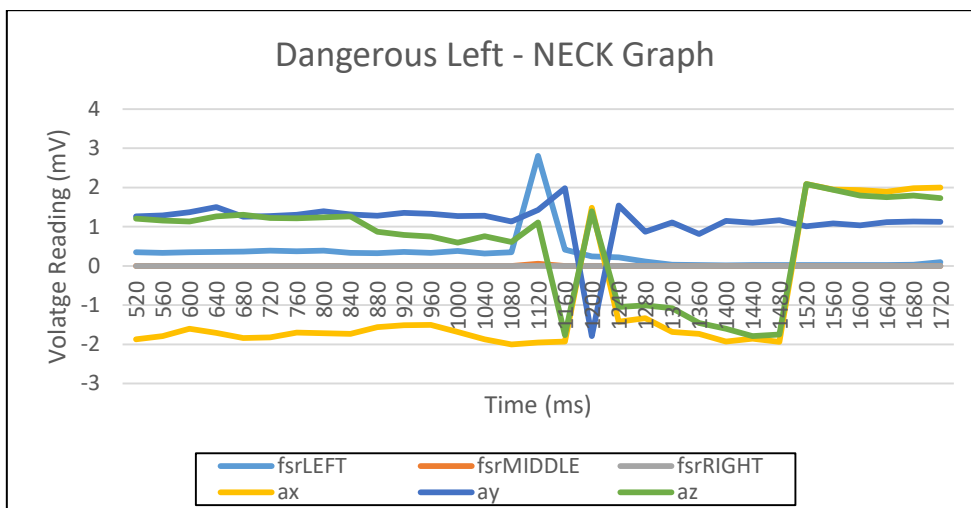


Figure 9.3.8 Dangerous Left Neck Graph

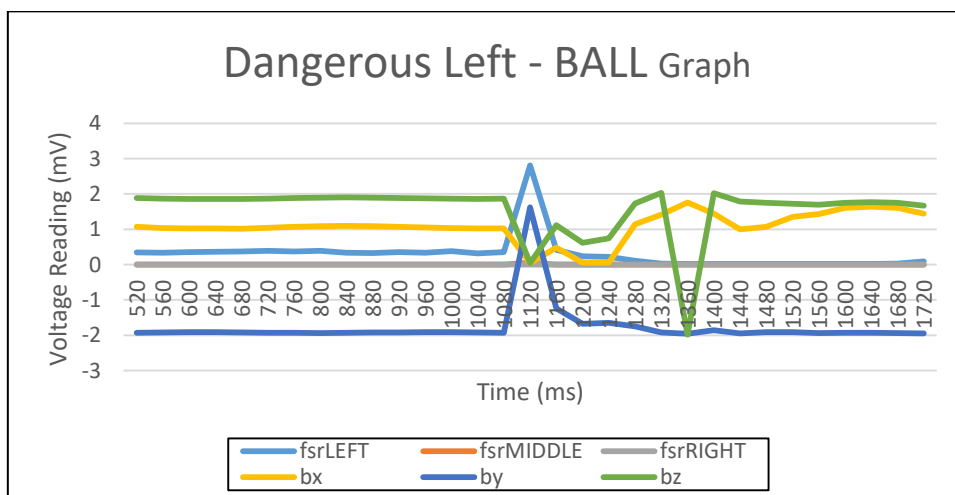


Figure 9.3.9 Dangerous Left Ball Graph

Right Dangerous

Similarly if the header is predominantly activated on the right FSR it raises concern as to the football impacting with a weaker part of the skull endangering the player which results in a Right Dangerous classification being given. The reason there are two separate categories for the football colliding with the pterion or near that region of the skull is to label which side of the head is hit, this information can be used in further study where the neurological effects of players are tested. The three figures 9.10, 9.11 and 9.12 display the graphs of a Right Dangerous heading event. The force of this event is larger than usual as the spike in the X axis of the Head graph displays and much sharper increase and movement of the head to hit the ball. Since the football also impacts with the right side of the skull, there is awkward movement in the neck and there is ball spin after impact there are a many reasons to classify this as a bad header. As the collision is with the right FSR and the football the header is categorised as a Dangerous Right event. This is the final category that was used to classify heading events giving a total of four categories that cover all types of directions, movement and positioning. The final analysis section 9.4 describes the results of the machine learning algorithm used to classify new heading events into these four categories trained on the one hundred and eighty test headers previously categorised using key moments, as in this section.

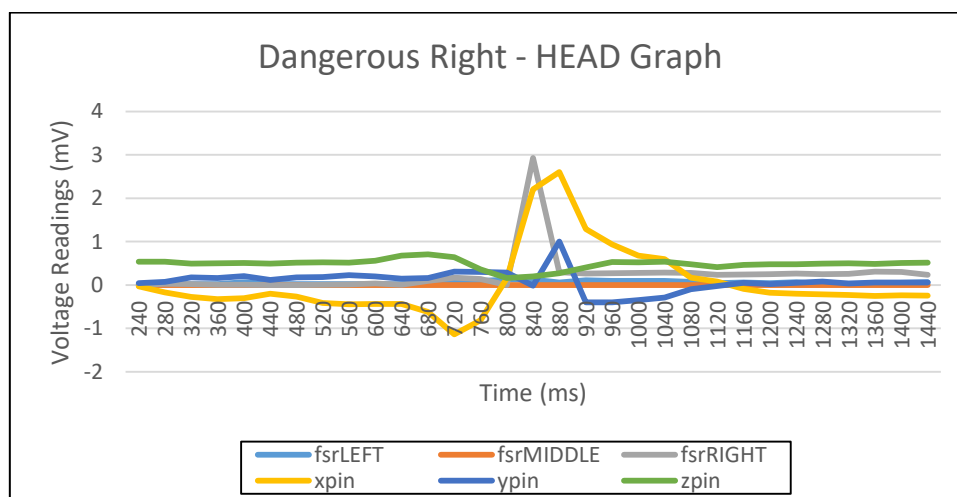


Figure 9.3.10 Dangerous Right Head Graph

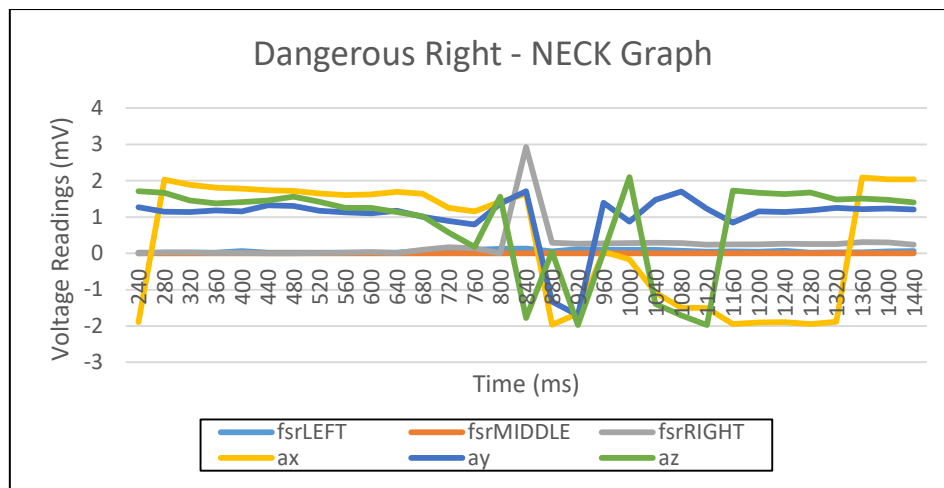


Figure 9.3.11 Dangerous Right Neck Graph

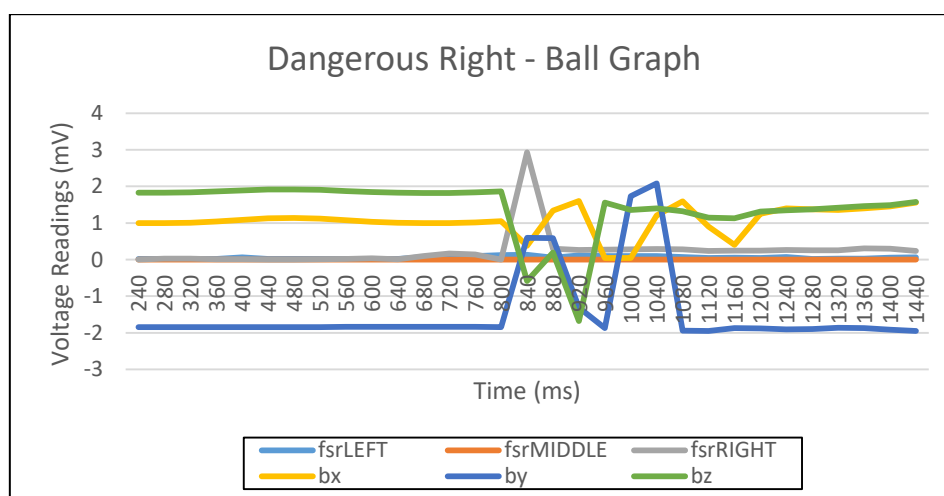


Figure 9.3.11 Dangerous Right Ball Graph

9.4 Overall System Results

To test the full system the machine learning algorithm section had to be added for analysis and 20 more tests were recorded and classified. These tests were then imported to the python load_data script and classified using the LSTM model. The results that the model outputted were compared to the classifications made during testing. Fourteen of the tests were correct and seven were misclassified with “Poor - Middle” being the highest misclassified reading due the similarities this category shares with “Good - Middle”. This give a real-world output accuracy of 70% which only slightly off the overall accuracy of 65.7% found from the test dataset. The difference is due to the small test size of 20 samples, if there were more the accuracy would be more representative of the sequence. The conclusion drawn from these results is that trained model can predict new headers very accurately which gives the player real-time feedback of how they can improve their heading technique and as a consequence reduce the harmful effects of heading a football.

10 CONCLUSION

The aim of this project was design and build a wearable head sensing platform which could perform feedback analysis on heading events. Through this analysis each event can be differentiated using their attributes and key features and categorised into classifications. In doing so a user can learn and improve their heading ability and reduce any associated risks with potential neurological harm by implementing these improvements to their game. The heading events were classified into four categories through much testing and results to prevent the player from head injury in the long-term. There were three bad categories, which could cause damage, which were the use of the temple on the left, use of the right temple, lack of neck movement resulting in the top of head impacting and finally a good central forehead event. The damaging categories should be avoided and the central forehead with steady but firm movement and motion of the neck should be implemented at each event resulting in a straight, effective and accurate header. From these conclusions the project objective has been achieved from implementing this platform to deploying test and results and finally concluding with these classifications. These were then used to implement an algorithm which can classify a new heading event in real-time allowing the user to have graphical feedback but also an accurate rating of the header and how to improve. The idea behind this was that the user can be tested for initial headers without feedback and then compared to the improvement in accuracy with feedback leading to a much higher percentage of accuracy.

The future of this study would be to get more feedback from testing on a larger scale and getting a more precise indication of how the player can improve with each event and even use this system as a regular practising tool. If the circuitry system was made more robust and could add more sensors which would record the actual force of each event this would give added value to the project. The use of an online database to save data rather than using CSV files locally would improve accessibility to recordings for a bigger scale. If a larger dataset was gathered the machine learning algorithm's accuracy could increase and the percentage of incorrect classifications would be minor went testing on new users.

Through the study of this project there were many things learnt including a huge improvement on circuit board soldering and design, communication with BLE devices and the study of this, Android app development, graphical sensory analysis and machine learning implementation. The broad range of topics, the experience of working on a challenging project and the knowledge gained was very beneficial to my development. The system could be implemented on a much larger scale and be used, for example, in kids soccer camps, so the knowledge of the proper heading technique can be learned at an early stage, therefore leading to less negative long-term effects.

BROADER SOCIETAL IMPACT

If this project was developed and implemented into the real-world there would be much to consider to improve scalability, usability and safety. Increasing the scale of this project would require an online database where all the results could be feedback to a neural network which could be constantly learning and updating based on new results. There would need to be standardised equipment deployed to multiples soccer training camps around the world which could do testing and give feedback from the system.

To improve usability the sensors could be built inside the ball and the frame supplied as one piece of equipment, that is height adjustable. The system could then be used in various scenarios of the header, which players can learn from, to improve their heading technique, in a game-like situation. The head mask would have to become more robust and the sensors would have to be built-in to the system with an increase in the amount and types of sensors, including a magnetometer to get a pin point accuracy of collision and a much more sensitive force measurement such as a strain gauge or a load cell. Using these changes all the sensors on the head and ball would be much more compact and useable but would also increase the cost of the system.

Increasing the usability goes hand-in hand with improving the safety of the system leading to less external wires and less room for hazard. One safety precaution would be to introduce a spongy head mask which would also offer more protection to the head. This could be deployed across the sport as a whole leading to less damaging effects on the head. This would improve the safety of the game and would lead to unintentional head collisions being less severe.

REFERENCES

- [1] Kunz, M. (2007, July 31st) BIG COUNT.FIFA Magazine [ONLINE]
Available at - https://www.fifa.com/mm/document/fifafacts/bcoffsurv/emaga_9384_10704.pdf
- [2] Carcione, J. (2016 March 5th) Concussions: understanding Brain Trauma [ONLINE]
Available at - https://www.onhealth.com/content/1/concussion_brain_injury
- [3] Sparke, A. (2018 December 9th) Bones of the Skull - TeachMeAnatomy [ONLINE]
Available at - <https://teachmeanatomy.info/head/osteology/skull/>
- [4] Rodrigues, A. (2016 March 21st) Effects of Soccer Heading on Brain Structure and Function [ONLINE]
Available at - <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4800441/#B15>
- [5] FSR Force Sensing Resistors, Interlink Electronics [ONLINE PDF]
Available at - <ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/datasheet/FSR400.pdf>
- [6] Dadafshar, M. (2014 March) ACCELEROMETER AND GYROSCOPES SENSORS [ONLINE PDF]
Available at - <https://pdfserv.maximintegrated.com/en/an/AN5830.pdf>
- [7] ADXL335 Module – ITEAD [ONLINE]
Available at - <https://www.itead.cc/adxl335-module.html>
- [8] SparkFun Triple-Axis Digital-Output Gyro Breakout - ITG-3200 [ONLINE]
Available at - <https://www.sparkfun.com/products/retired/11977>
- [9] GENUINO 101 – Arduino
Available at - <https://store.arduino.cc/genuino-101>
- [10] ARDUINO MEGA 2560 REV3 – Arduino
Available at - <https://store.arduino.cc/mega-2560-r3>
- [11] Bluno Nano - An Arduino Nano with Bluetooth 4.0 - DFROBOT
Available at - <https://www.dfrobot.com/product-1122.html>
- [12] Ray, B. (2015 November 1st) Bluetooth Vs. Bluetooth Low Energy [ONLINE]
Available at - <https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy>
- [13] Through, P. (2016 April 3rd) Maximizing BLE Throughput on iOS and Android [ONLINE]
Available at - <https://punchthrough.com/pt-blog-post/maximizing-ble-throughput-on-ios-and-Android/>
- [14] Woodford, C. (2018 June 28th) Accelerometers [ONLINE]
Available at - <https://www.explainthatstuff.com/accelerometers.html>
- [15] Singh, P. (2014 February 11th) Mems accelerometer designing and fabrication – Slide 5 [ONLINE]
Available at - <https://www.slideshare.net/prashantsingh94651/mems-accelerometer-designing-and-fabrication>

[16] Lucas, J. (2017 September 26th) Force, Mass & Acceleration: Newton's Second Law of Motion [ONLINE]

Available at - <https://www.livescience.com/46560-newton-second-law.html>

[17] Analog to Digital Conversion [ONLINE]

Available at - <https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all>

[18] CurieBLE Library [ONLINE] – Arduino information page on BLE

Available at - <https://www.arduino.cc/en/Reference/CurieBLE>

[19] GATT Overview – Bluetooth [ONLINE]

Available at - <https://www.bluetooth.com/specifications/gatt/generic-attributes-overview>

[20] Townsend K. (2014 March 20th) GATT – AdaFruit [ONLINE]

Available at - <https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt>

[21] Intel (2017 March 1st) Intel Curie Model Datasheet [ONLINE]

Available at - <https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/curie/intel-curie-module-datasheet.pdf>

[22] Host Control Interface (HCI) [ONLINE]

Available at - https://www.amd.e-technik.uni-rostock.de/ma/gol/lectures/wirlec/bluetooth_info/hci.html

[23] Circuit Basics (2016 June 1st) Basics of UART Communication [ONLINE]

Available at - <http://www.circuitbasics.com/basics-uart-communication/>

[24] Developers (2018) Application Fundamentals [ONLINE]

Available at - <https://developer.Android.com/guide/components/fundamentals?hl=en>

[25] Bluetooth (2018) Bluetooth Developer Starter Kit [ONLINE]

Available at - <https://www.bluetooth.com/develop-with-bluetooth/build/developer-kits/bluetooth-starter-kit>

[26] Jahoda, P. (2018) MPAndroidChart [ONLINE]

Available at - <https://github.com/PhilJay/MPAndroidChart>

[27] Tahsildar, S. (2018 October 11th) Machine Learning Basics [ONLINE]

Available at - <https://www.quantinsti.com/blog/machine-learning-basics>

[28] SRIVASTAVA, P. (2017 December 10th) Essentials of Deep Learning : LSTM [ONLINE]

Available at - <https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/>

[29] Colah, C. (2015 August 27th) Understanding LSTM Networks [ONLINE]

Available at - <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[30] Brownlee, J. (2016 March 21st) Overfitting and Underfitting With Machine Learning Algorithms [ONLINE]

Available at - <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>

APPENDICES

A.1 Parts List

Name	Quantity	Material/Type	Source	Cost Per Unit (€)	Total cost (€)
Arduino Genuino 101	2	Microprocessor	Lab Technicians	30	60.00
9 Volt Battery	2	Lithium Ion	Lab Technicians	3	6.00
Arduino Case	2	ABS	Arduino Website	4	8.00
Battery Connector	2	Barrel to 9V Head	Lab Technicians	1	2.00
FSR 406	3	Conductive Polymer	Interlink Electronics	4	12.00
ADXL335	1	Accelerometer	ITEAD	5.5	5.50
Colour Sleeved Wire	≈15	Copper	Lab Technicians	>0.2	1.00
10K Resistor	3	Metal Oxide	Lab Technicians	>0.4	1.00
Head Mask	1	Thermal Fabric	Amazon	5	5.00
Football	1	Synthetic Leather	Amazon	8	8.00
Ball Netting	1	Woven Fabric	Amazon	1	1.00
Rope	1	Polypropylene	Lab Technicians	N/A	0
Thick Wire Sleeving	1	Chloroprene	Home	N/A	0
Solder Breadboard	1	Copper	Lab Technicians	N/A	0
Black Pouch Holder	2	Thermal Fabric	Home	0.5	1.00
Velcro Straps	1	Velcro Tape	Lab Technicians	5	5.00
Total Cost					€115.50

A.2 Code Cheat Sheet

This is a set of instructions to help with running the code for each of the three sections as listed below.

Arduino

There are two Arduino C files, the Ball_Arduino and Head_Arduino, which can be run through the Arduino IDE. The appropriate imports will need to be installed for the Genuino 101 and BLE features. Once updated these files can be loaded to two separate boards which will make them capable of saving the recorded readings from the analogue pins to arrays and transmitting these arrays in packets of 20 bytes using BLE when an Android app sends the transmission signal.

Android

The Android code is run, as explained in detail in section 5, this is the Heading_App application which is broken into many different classes and three Activities. These display the individual interface screens of the application. Once this has been loaded to an Android device, with an Android version greater than 7.0 (nougat), then the phone can connect to the two peripheral devices, transmit and receive data packets, store the recorded packets to CSV files saved locally and analyse the received material using Graph Activity. This Activity needs the import of the MPAndroidChart software. It also joins and saves the data to a separate cut to size CSV file which can be imported to the machine learning algorithm.

Machine Learning

There are two python scripts, one to create the algorithm called time_series_train.py, the other to load the model and test on an unclassified dataset load_model.py. These files are run through command prompt and output an algorithm accuracy and classification respectively. Tensorflow needs to be pip installed using the python installer. Once done the time_series_train.py can be run using the dataset and target data. This will output a Pickle file which can be loaded to the load_model.py script and can classify a new CSV recording.

HEALTH AND SAFETY

RISK ASSESSMENT FOR SOLDERING IN



O'É Gaillimh
NUI Galway

College of Engineering and Informatics, Risk Assessment.

Locations: List ALL Lab Location(s): ENG-3004	Project Title: Soldering	Duration (Dates) : 03/01/2019
---	--------------------------	-------------------------------

Person(s) at risk: Ronan Murphy - Student

Hazards	Likelihood	Severity	Controls
FIRE	1	2	CO2, foam and ABC fire extinguisher in lab. Leave soldering iron in holster and not in use. Don't apply to flammable materials. Turn off when not using.
INHALATION	3	1	Don't work on soldering iron for long period of time. Turn on fume extractor before use.
BURN	2	2	Don't touch tip of iron. Don't touch materials that have just been soldered.
FLUX IN EYES	1	3	Wear safety goggles at all times.
ELECTRIC SHOCK	1	2	Check for broken or loose wiring regularly and report damages.
SOLDER SPLATTER	1	1	Clean solder tip after each appliance. Limit solder on iron.
DAMAGE BENCH	2	1	Don't leave soldering iron on bench. Clean bench once cooled.

Likelihood:

- 1= Very Unlikely/Yearly
- 2= Unlikely/During a Semester
- 3= Likely/ Weekly
- 4= Likely/Daily

Severity:

- 1= Slight Harm
- 2= Moderate Harm
- 3= Extreme Harm

Severity 1 2 3

Likelihood
1
2
3
4

1	X	X	X
2	X	X	
3	X		
4			

Overall Risk assessment is highest risk colour:

Red =High

Yellow = Medium

Green = Low

Overall Risk = Medium

Risk Assessment
with
controls

High

0

Medium

1

Low


4

Prepared By: Ronan Murphy

Approved by (Research/Project Supervisor(s)):

Signed by Researcher/Student(s) –

SOP FOR SOLDERING IN LABORATORY

 O'É Gaillimh NUI Galway		College of Engineering and Informatics		
Subject: Project Soldering		LABORATORY OPERATING PROCEDURE Title: Soldering		Lab Number: ENG-3004
<u>Objective</u>		To solder wires in circuit		
Health & safety Documents		Refer to the Safety Statement for the particular Laboratory		
New Safety Hazards		Fire, inhalation of fumes, burning of skin, flux causing eye damage, electric shock, solder splatter, bench damage.		
Additional PPE Required		Safety goggles.		
Additional Engineering Controls		<ol style="list-style-type: none"> 1. Leave Soldering iron in holster when not in use. 2. Turn fume extractor on before use. 3. Turn off iron when not in use. 4. Wear safety goggles. 5. Clean tip of iron after every use. 6. Don't touch tip of iron or items recently soldered. 7. Ensure no loose wiring in solder before use. 		
Overall Risk Level Low/Med/High		Medium		
Report all accidents to:-		Lab Champion:- Martin Burke, Myles Meehan		
SOP prepared by: Ronan Murphy		Contact Details: r.murphy33@nuigalway.ie	Approved By: Liam Kilmartin	Date: 03/01/2019

#1	<u>Scope of Work/Activity:</u> Soldering components onto circuit board and soldering wires together. Equipment – Soldering iron, Solder, wire cutters, fume extractor, heat shrink, vice.
#2	<u>Reference Documents:</u> The recommended document for soldering safety (from University of Rhode Island): http://www.riccardobevilacqua.com/SolderingSafety.pdf
#3	<u>Environmental or Waste Management Impacts and procedures:</u> Extra wire, solder and damaged components disposed.
#4	<u>Other Equipment:</u> Soldering iron, Solder, wire cutters, fume extractor, heat gun, Vice.
#5	<u>Materials:</u> Solder, metal wires, heat shrink

#6	<p><u>Maintenance:</u></p> <p>- Within Equipment Calibration date – Yes __, No __, n/a _x_.</p> <p>- Within Preventative Maintenance due date - Yes __, No __, n/a _x_.</p> <p>- Within PAT testing due date - Yes _x_, No __, n/a _x_.</p>
#7	<p><u>Procedure:</u></p> <ol style="list-style-type: none"> 1. Put on the required PPE (safety glasses). 2. Turn on the soldering iron and fume extractor. 3. If soldering wires: <ol style="list-style-type: none"> 1. Secure one of the wires in the vice. 2. Apply solder to wire in vice. 3. Hold the end of the second wire to end of wire in vice. 4. Use the soldering iron to melt the solder and attach the two wires. 5. Test the strength of the joint by firmly tugging on the wire. 6. Slide a section of heat shrink over the solder joint to fully cover the joint. 7. Use the heat gun to evenly apply heat to the heat shrink until the solder joint is sealed. 4. If soldering components onto a board: <ol style="list-style-type: none"> 1. Put the components in place on board. 2. Place the board face down on the soldering mat. 3. Press the solder on where wire meets board. 4. Once the solder joint has been made, use the wire cutters to trim excess. 5. Place the soldering iron back into the holster. 6. Turn off the soldering iron and fume extractor. <p>In case of emergency:</p> <ol style="list-style-type: none"> 1. Power off equipment if safe to do so. 2. If required, call the emergency services on 999 or 112. 3. Seek support from first aid person on duty.

RISK ASSESSMENT FOR ATTACHING FOOTBALL



OÉ Gaillimh
NUI Galway

College of Engineering and Informatics, Risk Assessment.

Locations: List <u>ALL</u> Lab Location(s): ENG-3002	Project Title: Attaching Football at a Height	Duration (Dates) : 27/03/2019 28/03/2019
Person(s) at risk: Ronan Murphy - Student		

Hazards	Likelihood	Severity	Controls
UNSAFE USE OF LADDER	2	3	Ladders only used for short duration, placed on solid ground, held by a second person and shoes being worn are slip resistant.
DEFECTIVE LADDER	1	3	Condition and stability of Ladder check fully before use and not used if there is any doubt.
OVERREACHING	2	3	Make sure the Ladder is high enough for overreaching not to be necessary, don't reach out to beam if too far away.
ROPE FRICTION BURN	2	1	Wear gloves when tying a knot in the rope for attachment.
OVERHEAD OBJECTS HAZARD	1	2	Check for loose objects above before climbing the Ladder and remove if possible.
BEAM STRENGTH	1	1	Use a very secure beam that wouldn't have any risk of breaking under small to medium pressure.

Likelihood:

1= Very Unlikely/Yearly
2= Unlikely/During a Semester
3= Likely/ Weekly
4= Likely/Daily

Severity:

1= Slight Harm
2= Moderate Harm
3= Extreme Harm

Severity 1 2 3

Likelihood

1
2
3
4

1	X	X	X
2	X		X
3			
4			

Overall Risk assessment is highest risk colour:

Red =High

Yellow = Medium

Green = Low

Overall Risk = Medium


Risk Assessment with controls	High 2	Medium 1	Low 3
--------------------------------------	-----------	-------------	----------

Prepared By: Ronan Murphy

Approved by (Research/Project Supervisor(s)):

Signed by Researcher/Student(s) –

SOP FOR ATTACHING FOOTBALL

 O'É Gaillimh NUI Galway	College of Engineering and Informatics			
Subject: Project Ladder	LABORATORY OPERATING PROCEDURE Title: Climbing Ladder to Attach Football			Lab Number: ENG-3002
<u>Objective</u>	Attach rope to beam using ladder			
Health & safety Documents	Refer to the Safety Statement for the particular Laboratory			
New Safety Hazards	Defective Ladder, rope friction burn, overreaching, beam strength, overhead object hazard, unsafe use of ladder.			
Additional PPE Required	Stable and secure Ladder, Rope, Solid Beam			
Additional Engineering Controls	<ol style="list-style-type: none"> 1. Ensure Ladder is safe to use, check for stability and cracks 2. Ensure there are no overhead objects that could fall 3. Wear gloves. 4. Secure Ladder to flat ground directly below the beam and have a second person hold Ladder 5. Don't touch tip of iron or items recently soldered. 6. Carefully climb ladder and tie know to beam without overreaching 7. Once secure carefully climb back down 			
Overall Risk Level Low/Med/High	High			
Report all accidents to:-	Lab Champion:- Martin Burke, Myles Meehan			
SOP prepared by: Ronan Murphy	Contact Details: r.murphy33@nuigalway.ie	Approved By: Liam Kilmartin	Date: 03/01/2019	

#1	<u>Scope of Work/Activity:</u> Climb ladder and tying rope to beam overhead Equipment – Ladder, Rope, Solid Beam.
#2	<u>Reference Documents:</u> Cleaning Work at Height (Ladders) – Risk Assessment https://www.hsa.ie/eng/education/managing_safety_and_health_in_schools/new_guidelines_files/cleaning-work-at-height-ladders_%E2%80%93no-9.pdf
#3	<u>Environmental or Waste Management Impacts and procedures:</u> Extra rope cut off.
#4	<u>Other Equipment:</u> N/A
#5	<u>Materials:</u> Propylene fibre rope, steel metal ladder, steel metal beam

#6	<p><u>Maintenance:</u></p> <p>- Within Equipment Calibration date – Yes __, No __, n/a __.</p> <p>- Within Preventative Maintenance due date - Yes __, No __, n/a __.</p> <p>- Within PAT testing due date - Yes __, No __, n/a __.</p>
#7	<p><u>Procedure:</u></p> <ol style="list-style-type: none"> 1. Put on the required PPE (Safety Gloves). 2. Get and secure Ladder to solid ground underneath beam. 3. Get second person to hold ladder in place. 4. Carefully climb ladder to required height. 5. Get someone to pass the rope attached to football. 6. Reach, without overreaching, and tie know to secure beam. 7. Once secure, climb back down ladder carefully. 8. Return all used equipment to original place. <p>In case of emergency:</p> <ol style="list-style-type: none"> 1. If required, call the emergency services on 999 or 112. 2. Seek support from first aid person on duty.