UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Foot strike pattern determination with a battery-powered device and Android application.

Dewaldt Snyman
22706852

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccalaureus in Engineering in the Department of Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: Dr W. A. Smit

31 October 2022

Acknowledgements

I want to thank my family and friends for all the support; without them, this would not have been possible. Finally, I thank Dr. Willem Smit, my skirpsie coordinator, for all the advice and help with this skripsie.

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.
   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

22706852 Studentenommer / *Student number*	 Handtekening / *Signature*
DN Snyman Voorletters en van / *Initials and surname*	October 31, 2022 Datum / *Date*

Abstract

English

This document is a technical report on the design and development of a mobile application capable of using data readings from a prototype pressure-sensing footwear device to display foot strike patterns and other relevant information. This information could help with health analysis, like the gait analysis, on runners and other athletes. This application could be a tool for healthcare professionals and high-performance athletes.

The prototype device can continuously sample ADC measurements from a portable device with 8 FSR cells. This FSR device fits in the sole of a shoe, and ADC measurements from the FSR device are read and transmitted by the Arduino. The transmission happens via BLE (Bluetooth Low Energy). The Android application can then connect to the Arduino, receive these ADC readings, and use the data to display the foot strike patterns of a user. The foot strike patterns are displayed on a heatmap using the power of openGL—ES. The force exerted on each cell can be calculated by calibrating each cell and using the necessary mathematics to formulate a graph that relates the force to the ADC values. The prototype device has battery power and fits in a casing that allows users to strap the device to their legs.

A new casing was designed and printed with a 3D printer to cover the prototype device circuitry.

Afrikaans

Hierdie dokument is 'n tegniese verslag oor die ontwerp en ontwikkeling van 'n mobiele toepassing wat datalesings van 'n prototipe drukwaarnemende skoentoestel kan neem om voetstakingpatrone en ander relevante inligting te vertoon. Hierdie inligting kan help met gesondheidsanalises, soos die Gait analise, op hardlopers en ander atlete. Hierdie toepassing kan 'n hulpmiddel wees vir afrigters en atlete.

Die prototipe toestel kan voortdurend analoog na digitale lesings neem vanaf 'n draagbare toestel met 8 drukkrag sensitiewe resistor selle. Hierdie Ftoestel pas in die sool van 'n skoen, en analoog na digitale word deur die Arduino opgeneem en oorgedra. Die oordrag vind plaas via BLE (Bluetooth Low Energy). Die Android-toepassing kan dan aan die Arduino koppel, hierdie ADC-lesings ontvang en die data gebruik om die voetstakingpatrone van 'n gebruiker te vertoon. Die voetstakingpatrone word op 'n hittekaart(heatmap) vertoon met behulp van die krag van openGL—ES. Die drukkrag wat op elke sel uitgeoefen word, kan bereken word deur elke sel te kalibreer en die nodige wiskunde te gebruik om 'n grafiek te formuleer wat die krag met die analog na digitale waardes in verband bring. Die prototipe toestel het batterykrag en pas in 'n omhulsel waarmee gebruikers die toestel aan hul bene kan vasbind.

'N Nuwe omhulsel is ontwerp en gedruk met 'n 3D -drukker om die stroombaan van die prototipe toestel te bedek.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms and abbreviations

FSR	Force Sensing Resistor
MCU	Micro Controller Unit
ADC	Analog to Digital
BLE	Bluetooth Low Energy
SD	Standard Definition
GATT	Generic Attribute Profile

Chapter 1

Introduction

In the modern day, jogging is one of the most popular physical activities [4]. People all over the world partake in this physical activity. One would think that jogging is a simple exercise and that there are no significant injuries or health concerns. However, there are some major concerns, as jogging has a high injury rate. There is an increasing interest in this subject and studies on how health specialists can analyze these injuries. Strike patterns during running are a possible way of identifying injuries or risk of injury. Methods like the Gait analysis, which also analysis strike patterns, are popularized amongst runners and athletes to prevent or condition injuries. Unfortunately, this type of analysis does acquire sophisticated devices.

1.1. Problem Statement

There is a need for a portable device and a mobile application that can help runners and athletes see their foot strike patterns and use that information for analysis.

1.2. Objectives

Develop an application that can use data from a prototype device and display the foot strike patterns from this data. Do additional calculations to show the ground force applied with each step. The raw data should be processed and saved to be displayed and used in practical manners, such as videos and graphs.

1.3. Scope and Limitations

This project's scope is to design and develop an Android application that displays the foot strike patterns captured by the prototype device. As a predecessor built the device, some design flaws to consider:

- The casing for the device could be more sturdy and practical.
- There is no documentation of the Arduino code on the device.

- The Arduino Nano 33 BLE does not have enough ADC channels, and an ADS1115 had to accommodate the extra two ports needed. The ADS1115 causes the readings to differ somewhat.
- Charging circuit is not in working order

1.4. Overview of the report

Chapter 2 Literature Review
This chapter is the documentation for the necessary research and literature review needed to complete this project. The essential topics are BLE, FSR, Gait analysis and footstrike patterns, and OpenGL.
Chapter 3 System Design Review
This chapter is a brief review of the system design. A previous student had already built the device
Chapter 4 Software Design In this chapter, each section will explain all the Arduino and Java code used for the prototype device and android application, respectively
Chapter 5 Testing and Calibration
In this chapter, we conduct tests to verify that the device and application are in working order. The tests showcase some of the capabilities and observations made with the application and device.
Chapter 6 Summary and Conclusion
This chapter summarizes the device and application's capabilities along with a conclusion on the observation, results, and overall success of the project.
Chapter 6 Further Work
This chapter discusses how the prototype device and Android application can implement improvements and features in the future.

Chapter 2

Literature Review

2.1. Foot strike patterns and Gait analysis

Running is a popular everyday physical activity worldwide, and statistics prove this [4]. Although running is a simple activity, it involves complex movements and integrating muscles, joints, and various body parts. These complex movements cause running to be a common physical activity that typically causes injuries. In-depth studies on foot strike patterns elsewhere [5], [6], [7], [8] review the basics of foot strike patterns and the biomechanics of running. These in-depth studies are beyond the scope of this technical report, but these reviews provide relevant information to understand better what causes these injuries. These studies indicate that considering foot motion during walking and running is crucial to understanding why running can commonly lead to injuries. It has been proven in studies such as [9] that running performance, energy requirements, and musculoskeletal stresses are directly related to foot strike patterns and action-reaction force between the limb and the ground.

Factors like foot strike patterns, footwear conditions, running speed, and environmental conditions can affect a human's biomechanics during running. According to [7] there are three primary foot strike patterns: forefoot, rearfoot (heel strike), and midfoot. Forefoot striking is when the anterior region of the foot strikes the ground first. Midfoot striking is when the posterior and anterior parts of the foot hit the ground simultaneously, and rearfoot or heel strike is when the heel or rear area of the foot strikes the ground initially. According to [10] [7] majority of both mid-distance and long-distance runners are heel strikers. This would explain why running shoes are heavily padded at the heel part of the shoe. This padding makes the landing process more comfortable. See Figure 2.1 for an excellent visual representation of the different strikes. There are many benefits of knowing one's foot strike patterns. For instance, specialists use foot strike patterns in the Gait analysis.

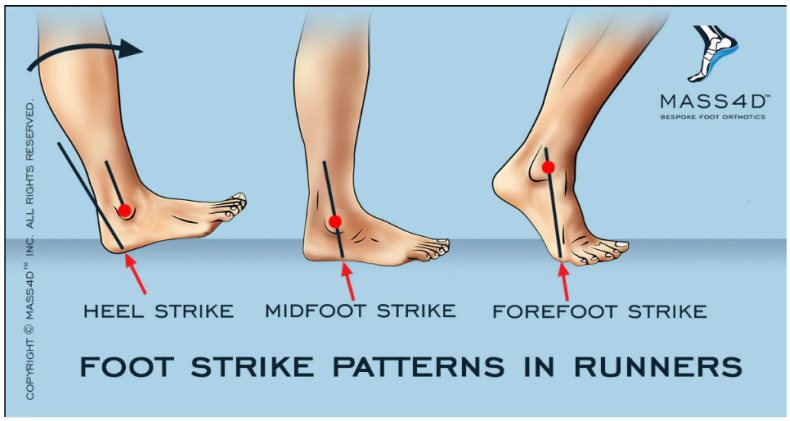


Figure 2.1: Image found in an online article [1] illustrating the Foot Strike Patterns in Runners.

Gait analysis studies human motion using observer instruments to measure body movement, body mechanics, and muscle activity. Gait analysis is commonly used in biomechanics to help athletes prevent or recover from injuries. The gait analysis identifies posture- or movement-related problems that can cause injuries. Identifying these problems allows professionals to advise and train athletes to move more efficiently in their sport, mainly when running or walking. Foot pressures are regularly used in the systematic physical examination of athletes, such as the Gait analysis [11]. Further reading shows that foot strike patterns, step rate, and posture can be combined for gait modifications and to reduce the impact of the load on body parts during running. The study [12] from the Journal of Biomechanics demonstrates this with a series of experiments where runners were analyzed as seen in figure 2.2

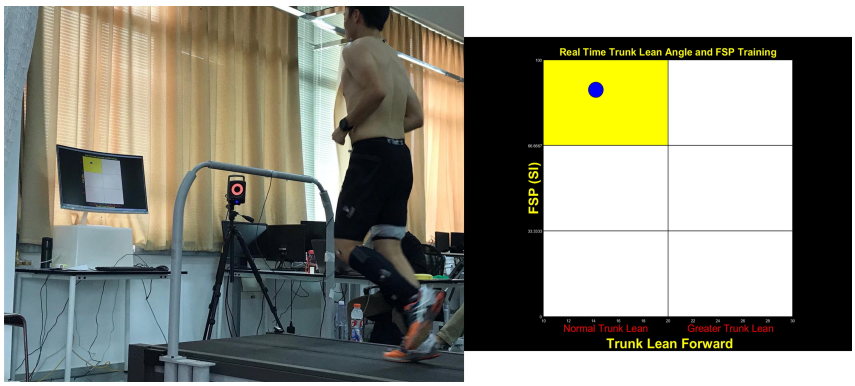


Figure 2.2: Example of how gait analysis can be for runners

2.2. Technology used for Gait analysis

Table 2.1: Summary of running gait parameters that different sensors and systems can measure [3].

Sensor\System	Measurements and outcomes
Motion analysis systems	Linear and angular velocity, and acceleration. The body orientation and positioning.
Force platforms	Ground reaction force and joint moment. This system can measure power by using motion analysis along with force platforms.
Pressure sensors	Pressure distribution of the foot. Vertical force, the center of pressure, and spatiotemporal measurements.
Electromyography	Muscle activation and muscle fatigue.
Accelerometers	Acceleration and orientation.n
Electrogoniometers	Relative joint angles.
Gyroscopes	Orientation, angular velocity, and acceleration.

2.3. Bluetooth Low Energy BLE

2.3.1. How Does BLE Work?

When using Bluetooth Low Energy, it is essential to know the roles of each device. These roles are defined by the Generic Attribute Profile(GATT).In all BLE applications, there are two roles: the **central** and the **peripheral** devices. [13] The peripheral device will be the device that broadcasts or advertises information, and the central device will be scanning for information. A good visual representation of how BLE works is to think of an advertising board where the peripheral device keeps pinning new info onto the board, and the central device scans the board and uses the information available. These two devices have unique addresses. The peripheral will be advertising information to nearby devices, while the central device will look for any devices that advertise information; when the central device finds the advertised information, it attempts to connect to the peripheral device. Once a connection is established, the central device can start reading or writing information from or to the peripheral device.

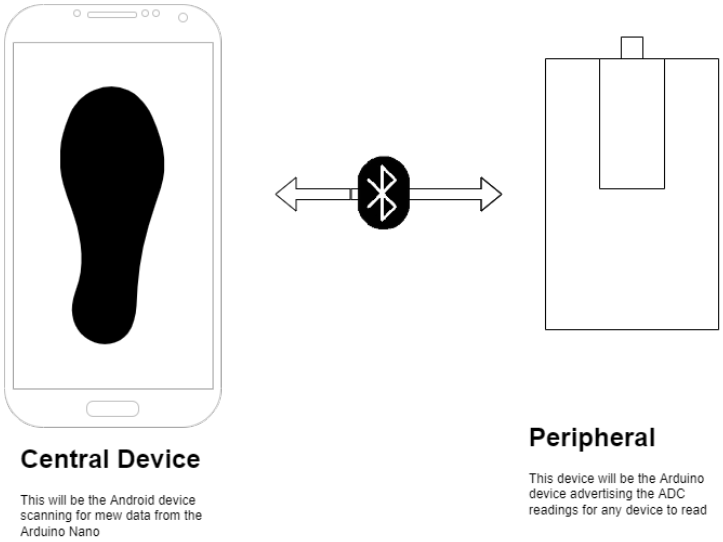


Figure 2.3: Basic BLE overview

2.3.2. Services, Characteristics Descriptor and Properties

The GATT structure for BLE can be explained in the following hierarchical order. A service is a collection of characteristics; each characteristic has properties and a descriptor that describes the characteristic. See the basic illustration below 2.4.

Each **service** has its unique identifying code called a UUID. The UUID allows one peripheral device to have multiple services and can be 128-bit long for each service. A service can also be seen as a group of capabilities. For example, a smartwatch can measure heart rate and temperature and track GPS location. These three capabilities can be

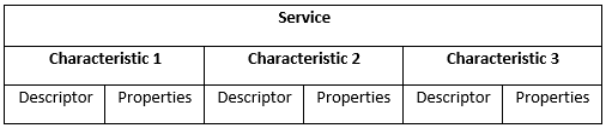


Figure 2.4: Basic hierarchy of GATT structure for BLE

grouped under one service called the activity service. This method of grouping information allows the central device to understand the information that the peripheral is advertising.

The capabilities mentioned in the above example are better known as **characteristics**. Each characteristic has its unique identifying code called, also known as a UUID, which allows one service to have multiple characteristics. A characteristic can be seen as a single capability. For example, the heart rate measurement is one characteristic. This characteristic can be seen as a single capability of the activity service.

A characteristic has other attributes that help describe the value that it contains. These attributes are **properties** and **descriptors**. Properties describe how the characteristic can be used. Properties represent several bits that indicate whether the characteristic is set to read, write, write without response, notify or indicate. The descriptor contains information about the characteristic value and how it should be interpreted. The descriptor can be user descriptions, format, and units of the value or extended properties of the value [14]. For example, the heart rate characteristic can be a range of the heart rate measurement, and the descriptor defines this range.

2.4. Force Sensitive Resistors

A Force Sensitive Resistor (FSR) is a piezoresistive electrical component, meaning a change in the electrical resistivity can be detected when mechanical strain is applied. Most FSR devices specify that they can measure force at temperatures as high as $200°C$. FSRs consists of three layers: a semi-conductive material/semi-conductive ink between two substrates. There are two types of FSRs: Shunt Mode and Thru Mode.

Shunt mode FSRs have polymer thick-film layers that have two separate parts. These parts are separated with a spacer. The shunt mode FSR is the most common and basic type of FSR and is also used by the IEEE foot sensor. This type of FSR would change resistance when pressure is applied, and the ink touches the electrodes. The more pressure applied more ink touches the electrodes and yields the voltage change across the resistor.

Thru Mode FSRs are made of polyester film layers which are positioned on the outer parts, while a conductive silver circle with traces secures the pressure-sensitive layers. An adhesive layer will then laminate the two layers together. See 2.5 for a graphical illustration of the two types of FSRs. Figure 2.5, along with more tutorials on the use of FSRs, was found here [15], [16]

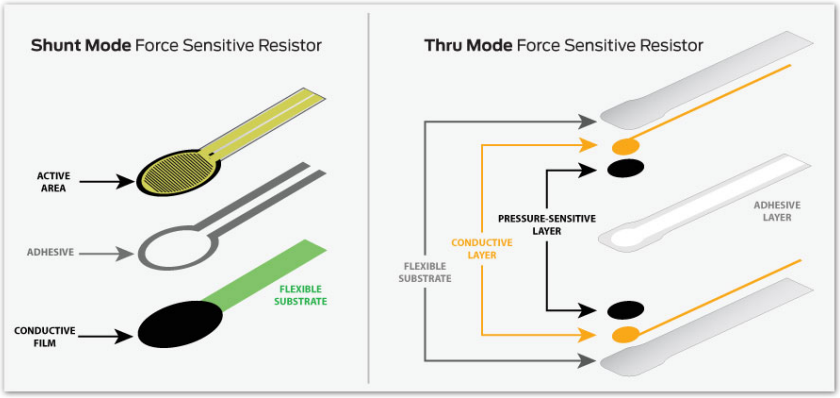


Figure 2.5: An illustration of the two types of FSRs and their different layers.

2.5. Open GLES for android

OpenGL is an open-source graphics library for high-performance 2D and 3D graphics rendering. OpenGL—ES is a flavor of OpenGL intended explicitly for embedded and mobile devices. OpenGL—ES is a cross-platform, high-performance graphics API that Android devices can use. Android supports the framework API and the Native Development Kit (NDK) of OpenGL.

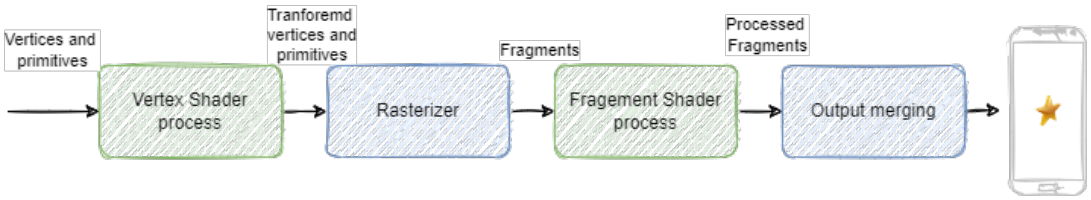


Figure 2.6: OpenGl pipeline diagram adapted from [2]

The primary pipeline of OpenGL and also OpenGLES starts by defining vertices. These are the points to draw a shape in a 2D or 3D space. These raw vertices are passed to the vertex shader to be processed. The vertex shader code will give these points a position on the screen. Next, the processed vertices are passed to the rasterizer. The rasterizer describes a virtual scene. The best way to understand this is to look at the visual explanation in figure **??**. The geometry of what is to be displayed is described with vertices. The rasterizer projects this geometry onto a 2D plane, which is the screen. The vertices are called fragments after they have been projected. These fragments are mapped with textures or colors by the fragment shader, depending on what is to be displayed. After that, all the fragments are merged to represent the final output to the display. All this information is compiled from the video presetion [17] by folkert. Here he explains all the basics, mathematics, and code of OpenGL.

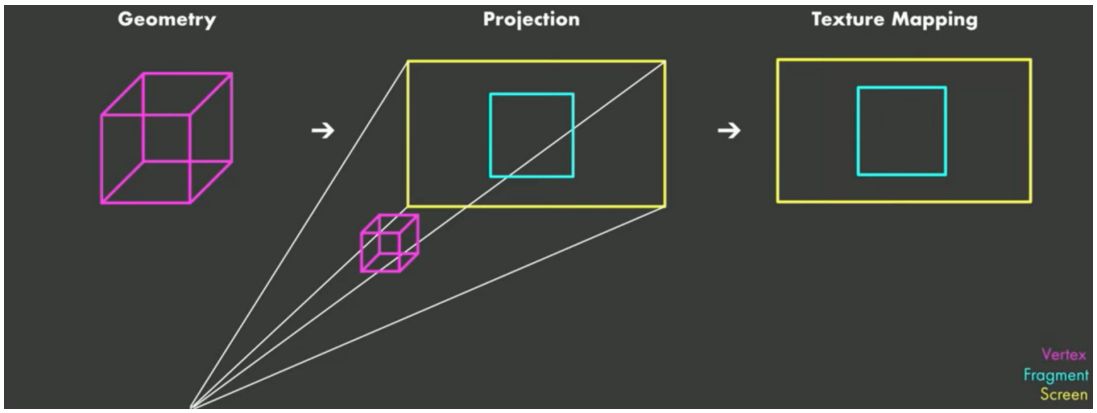


Figure 2.7: Rasterizer illustration

Regarding the Android framework, two foundational classes allow manipulating graphics with OpenGL ES API. These two classes are $GLSurfaceView$ and $GLSurfaceView.Renderer$. It is best to understand the implementation of these two classes to use OpenGL in an Android application. [18]

GLSurfaceView

This class is a View that can draw and control objects utilizing OpenGL API. This class is very similar to the $SurfaceView$ in functionality. To use the $GLSurfaceView$ class, create an instance of it and add a renderer. It is also possible to extend this class to allow for touchscreen functionality.

GLSurfaceView.Renderer

This interface needs to be implemented as a separate class. This class should then be attached to the $GLSurfaceView$ class by using $GLSurfaceView.setRenderer()$. This interface defines the methods required for drawing. These methods are the following:

- onSurfaceCreated(): This method is called when the GLSurfaceView is created. This method contains the actions needed to set up and initialize all the parameters and objects that are needed for an OpenGL ES environment. This method will only be called once.
- onDrawFrame(): This method is called each time, and a graphics object is drawn and redrawn.
- onSurfaceChanged(): This method is called when the system finds that the geometry has changed. The geometry can change when the orientation of the device screen is changed, like changing from portrait to landscape, or when the size of the $GLSurfaceView$ changes.

Chapter 3

System Design Review

3.1. System overview

The prototype foot sensor device was designed and built by Jared Adams, the predecessor of this skripsie topic. Link to Jared Adams's skripsie report. See appendix D for images of the device.

The LiFePO4 battery supplies power to all the components of the prototype device. These components are an Arduino Nano 33 Ble, an ADS115 ADC module, and the IEEE foot sensor. The battery has a nominal voltage of 3.2VDC and is charged by a battery charge management controller, namely, the MCP73123.

The prototype device will then use BLE to transmit ADC data to an Android device. The Android device will make use of this data via an application written in Java. This application will consist of various methods that perform calculations with the data, display the data in various ways, and save the data.

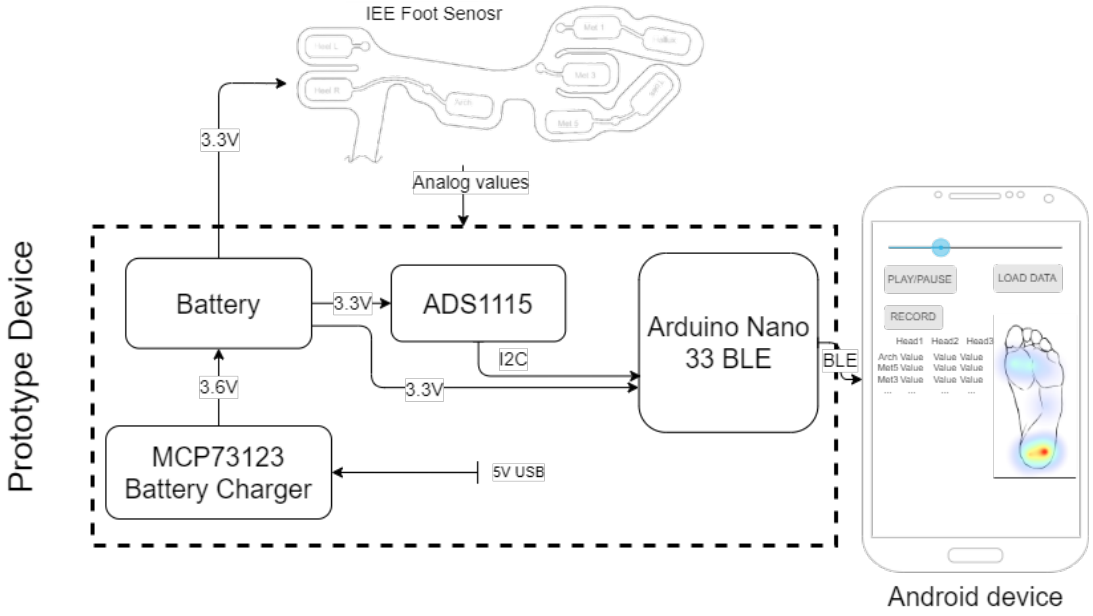


Figure 3.1: System Diagram

3.2. Device components

3.2.1. Arduino Nano 33 BLE

The Arduino Nano 33 BLE is a small Arduino Nano device ideal for wearable devices. The Nano 33 BLE has a powerful processor, the nRF52840 from Nordic Semiconductors, a 32-bit ARM Cortex-M4 CPU running at 64 MHz. The central processor also includes Bluetooth Low Energy, a low-power consumption solution to transmit data wirelessly. The board also contains eight 3.3-volt 12bit analog input pins (two of which are designated for I^2C pins) and 14 digital input/output pins.

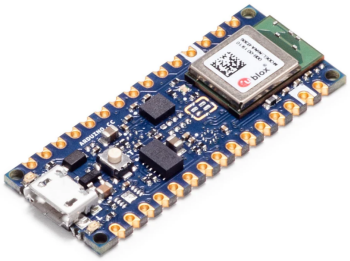
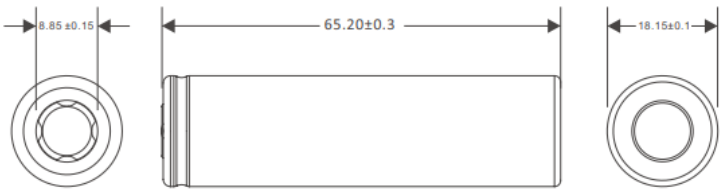


Figure 3.2: Arduino Nano 33 BLE

3.2.2. LiFePO4 Battery

A lithium iron phosphate battery is a type of rechargeable battery. This battery uses LiFePO4 as cathode material and graphite carbon electrodes as the anode and has a maximum charge cut-off of 3.65V and a minimum charge cut-off of 2.5V. LiFePO4 batteries have a flat discharge curve with a nominal output voltage of 3.2V. This battery provided a stable supply voltage for the system and a stable reference voltage for the ADCs. See the datasheet [19] for all the battery characteristics as needed.

(a) Picture of the physical battery



(b) Battery dimensions in mm

Figure 3.3: LiFePO4 Battery

3.2.3. IEE Smart Footwear Sensor

The IEE Smart Footwear Sensor consists of 8 separate high-dynamic FSR cells. The foot sensor has eleven pins. One supply voltage pin, two ground pins, and eight output pins. The structure of the foot sensor is essentially a resistive divider network with a fixed resistor R_{fix}, made from conductive ink with a value of 2 $k\Omega < R_{fix} < 4$ $k\Omega$. The foot sensor is supplied with power by the LiFePO4 Battery.

3.2.4. ADS1115 Precision ADC module

This component increased the number of analog pins in the system. The ADS1115 adds four additional 16-bit analog pins. This component communicates with the Arduino via I^2C and makes the extra analog pins available to the Arduino.

3.2.5. Battery Charger

The charger circuit uses an MCP73123 battery management controller chip with additional conditioning circuitry. The MCP73123 has a regulated 3.6V output which would be sufficient to charge the battery. An internal overvoltage protection circuit monitors the input voltage and shuts down the charger when it exceeds the 6.5V threshold. A constant current and voltage are achieved by controlling an internal P-channel MOSFET in the linear region. This controlled voltage is then connected to the battery terminals. The MCP73123 has charge detection, which allows it to automatically charge the battery once the battery voltage drops below 3.2V. See 3.4 for the circuit diagram.

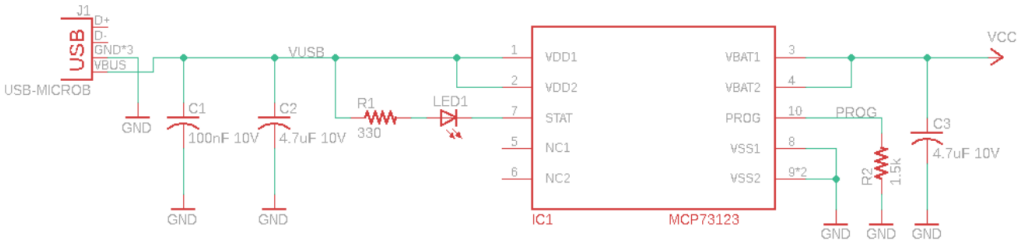


Figure 3.4: Battery charger circuit

Chapter 4

Software Design

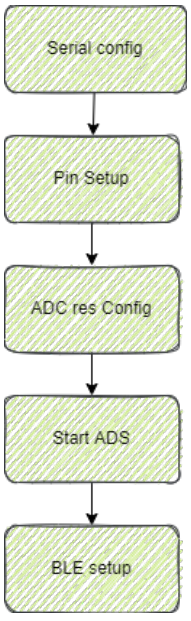
Find the full Arduino, Java, and latex code in this repository. [20]

4.1. Arduino Code

4.1.1. Setup

When using Arduino devices, there will be a setup() method which will run once only when the device starts. The Serial communications and baud rate are specified with the Serial.begin() function. After that, the Arduino pin modes are set to activate the internal pull-up resistors, as this is required for measuring the analog values from the IEE foot sensor. The pin mode can be configured using the pinMode() function. This function takes two arguments which are the pin number and the mode. An object of the Adafruit_ADS1115 class is created to allow the use of the ADC readings sent by the ADS1115 via I2C. Now within the setup() method ads.begin method can be triggered to do all the necessary I2C setup for the ADS1115. Next, the BLE setup commences. Arduino has a well-written library, ArduinoBLE.h, which provides all the needed functions to set up BLE for the Arduino device. See code snippet 4.1

Figure 4.1: Basic illustration of setup function




```
1     BLE.setLocalName("Arduino Nano 33 BLE (Peripheral)")
     ;
2     BLE.setAdvertisedService(gaitService);
3     gaitService.addCharacteristic(gaitCharacteristic1);
4     gaitService.addCharacteristic(gaitCharacteristic2);
5     BLE.addService(gaitService);
6     BLE.advertise();
7     if (!BLE.begin()) {
8       Serial.println("BLE error-Ble could not start");
9       while (1);
10    }
```

Listing 4.1: BLE Setup

The ”gaitService”, ”gaitCharacteristic1” and ”gaitCharacteristic2” objects, used in code snippet 4.1, are created outside the setup() function. The ArduinoBLE library provides the BLEService and BLECharacteristic classes to create these objects. The BLEService class only requires the service UUID as an argument to create an object. The BLECharacteristic class requires the characteristic UUID, the properties of the characteristic, and the size of the value that the characteristic will represent. The properties can be specified by applying an or-operation with predefined constants provided by the ArduinoBLE library.

4.1.2. Main loop

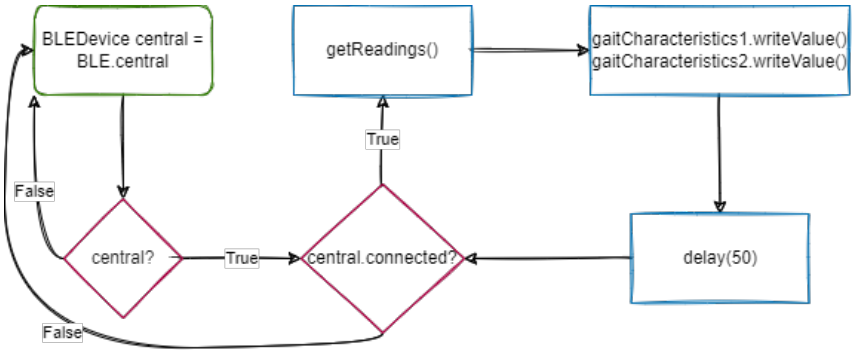


Figure 4.2: Main loop code flow

ADC

Unfortunately, the Arduino NANO 33 only has six ADC pins; therefore, an external ADC module, the ADS1115, was used to send the remaining two ADC values across I2C. The Adafruit library has a class with a method called readADCSingleEnded(). This method is used to get the ADC reading from the ADS1115 module. This method requires the pin number which connects the foot sensor to the ADS1115 as an argument. The ADS1115 has a 16-bit resolution; therefore, it is required to use the map() function to scale the 16-bit ADC value down to 12 bits. The remaining six ADC readings can be retrieved using the analogRead() function and passing the pin number as an argument.

All the ADC operations that need to happen are in the getReadings() function. This function stores eight analog readings in eight floats which are then copied across two byte arrays, each having a length of 16 bytes to minimize the use of characteristics to only two characteristics.

BLE

In the main loop() function, the Arduino will continuously wait for a connection from a central device. An if statement with the BLEDevice object as a condition will wait for

a connection to be established. Within the if statement is a while loop which will loop while the central device is connected. Inside the while loop, the getReadings() function is called to get all the ADC readings. These ADC readings are written to each characteristic with the writeValue() method from the previously created BLECharacteristic class objects. These characteristics are now advertising data to the central device

4.2. Android application code

4.2.1. Overview

The current version of the android application is only for testing purposes. This application is not built for commercial use but rather for the possibility of being commercialized. The basic java application layout can be seen in figure 4.3. The application only has one activity, and that is the main activity. The application has a few custom views and helper classes to assist the main activity in calculating, storing, and displaying the data readings. Each class and view will be discussed and explained to some extent in the coming section of this chapter. All the files have comments to understand the code better. If the reader wishes to use or further the development of the application or would like to use one of the classes, it would be best to refer to the actual code that is in a GitHub repository

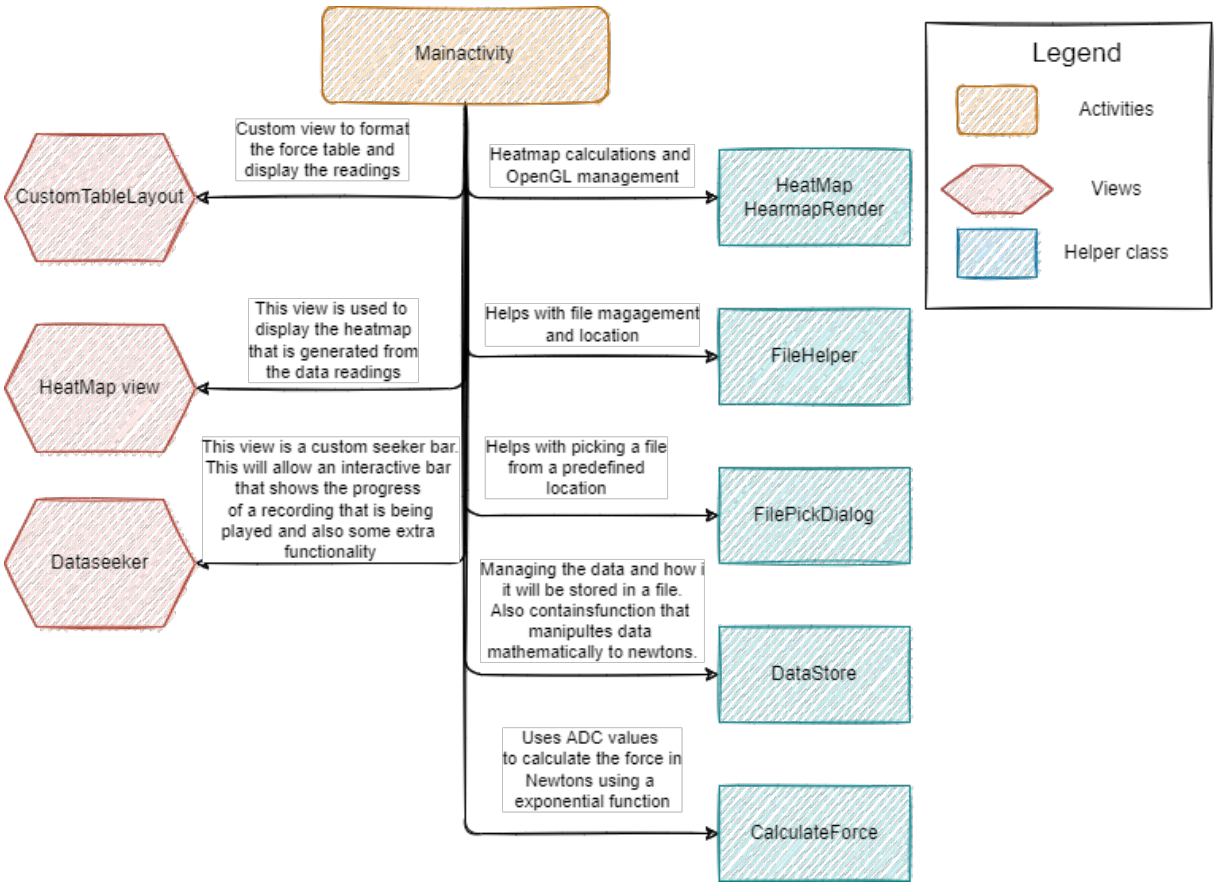


Figure 4.3: Android application basic layout

4.2.2. HeatMap

The heat map is the main focus of this topic, as it is an excellent way of displaying the user's foot strike patterns clearly and visually. For this openGL was used to represent the data graphically. The heat map consists of 3 classes: HeatMap, HeatmapRenderer, HeatmapView.

The HeatMapPoint, in the HeatMap class, contains the high-level code that facilitates interaction between java and OpenGL. Upon creating an instance of this class, the vertex and fragment shader code is loaded from internal resources; this allows for ease of editing, considering the alternative is placing the code into strings. The vertex shader is responsible for supplying OpenGL with any transformations to the drawing surface; this is only used to orient the camera for 2D rendering correctly.

The fragment shader is the more important one, the code contained within $heatmap\_f.frag$ is run for every pixel on the heatmap surface. To determine the color of any particular pixel, the distance to all the heatmap points is calculated and an appropriate color is determined based on this. The HeatMap class also contains the java representation of these points as a 1D float array, with each point consisting of an x, y, and intensity value. Then there are also some constants such as width, height, pointRadius, and heatmax(the maximum value each point can have). The init function is responsible for setting up OpenGL and loading the shader code. To interface with the rest of the program, the heatmap provides the IHeatMappable interface, which is used to retrieve an intensity value for a given index. An "instance" of this interface is included in every instance of the HeatMapPoint class, representing individual points with x and y coordinates. The coordinate system is set up such that coordinates are normalized by the smallest of the width and height measurements. This is very useful because it means that as long as the aspect ratio does not change, any change in width and height will not result in any of the points' coordinates changing. The heatmap contains a list of these points that it then packs into the aforementioned 1D array for sending to OpenGL. The initPoints function creates this 1D array and sets the initial values, but whenever the values of points change dataChanged should be called to update the 1D representation. Points can be added to the heatmap using the addPoint function. Lastly, the draw function is in charge of actually using OpenGL to draw the shader onto the screen. This function also sends the java variables to OpenGL for use within the shader code.

The HeatmapRenderer class contains a reference to its HeatMap instance used for drawing. Also, it contains various matrices used to orient the OpenGL "camera" in order to use it for 2D drawing. It implements onSurfaceCreated, where it calls the HeatMap init code and sets the background color; it also implements onDrawFrame, where it clears the drawing surface, applies previously mentioned transformations, and calls the HeatMap draw code. The onSurfaceChanged function is also implemented to set up the OpenGL

draw area when there is any width or height change. Lastly, a static helper function, loadShader, is included that takes in a shader type and actual shader code as a string and returns the OpenGL handle to the compiled shader.

The HeatmapView class is a custom view created to display the heatmap on the screen of a mobile device. This view includes some OpenGL initialization code and creates and sets the heatmap renderer. It also exposes the actual HeatMap object via getHeatmap to allow external activity code to add points. A separate function is added for updating the HeatMap data; this is required because the HeatMap needs to be updated, and the view itself needs to request a re-render.

4.2.3. DataSeeker

This class is an extension of the base Android SeekBar view, which allows a user to tap and linearly drag a notch between a minimum point and a maximum point. The goal of the DataSeeker is to allow for the animation of an arbitrary object so long as it implements the provided IDataAccessor interface. This interface provides the DataSeeker with the total amount of frames in an animation, the real-time stamp in milliseconds of any frame, and a function that should display any given frame. This class uses an ExecutorService to run a playback loop at non-fixed intervals; the loop will increase the animation frame by one each iteration, looping if enabled or terminating. The frame is displayed using a reference to an IDataAccessor, and then using that same reference determines how long to sleep the thread until the next frame should be displayed.

When this view is created, it sets its own OnSeekBarChangeListener so that it can stop the executor service, skip to the selected frame and restart it again to ensure smooth playback. Something similar is done in the playPause function, stopping or restarting the executor service to toggle playback smoothly.

4.2.4. DataStore

The DataStore class is a helper class that handles the data and the format it will be stored on a file. This class stores timed instance data which consists of all the FSR readings for a given timestamp (stored in milliseconds). The class HeatMap.IHeatMappable can be used directly with the heatmap, using the FSR readings as intensity values. The toString function represents all the stored data as a comma-delimited string for writing to the disk. This exact string can be passed in as a constructor parameter to re-create the original object by parsing the values after splitting by commas.

4.2.5. GattHandler

The GattHandler class mainly handles the BLE setup and connection. This class has an init(), seen snippet 4.2, that takes the current context as an argument and helps initialize the connection with a specific device. The device MAC has to be provided to establish a connection, and as we already know which device we want to connect to, we can provide the Arduino MAC address which is known to us. The content of the init() runs to a separate thread to allow the application to make the BLE connection in the background. The BluetoothAdapter object [21] is used to initiate a BluetoothDevice object [22] using a known MAC address. Then with the BluetoothDevice object, called device, the connectGatt() method is invoked. This method is used to connect to a GATT server hosted by the Arduino device. The caller, the android device, acts as a GATT client. A callback is then used to return results to the caller.

```
1  public static void init(Context context) {
2    // run function on a separate thread
3    new Thread(() ->
4    {
5        BluetoothAdapter bluetoothAdapter = BluetoothAdapter.
   getDefaultAdapter(); // use to perform fundamental Bluetooth tasks
6        BluetoothDevice device = bluetoothAdapter.getRemoteDevice(
   deviceMAC); // specify our device MAC
7        device.connectGatt(context, false, gattCallback);
8
9    }).start();
10 }
```

Listing 4.2: init method in the GattHandler class

The callback mentioned above is a BluetoothGattCallback object. This object is created in the GattHandler class and contains fundamental methods for interacting via Bluetooth. The onServicesDiscovered() method sets up the current GATT service with all the characteristics, notifications, and descriptors. Then there is the onCharacteristicChanged() method which is triggered every time the Arduino device writes to the characteristics. This allows one to capture the characteristic values and store them. The values are temporarily stored in a buffer, and each value is permanently stored in a float for further use. As the format of the data being sent is known, the data can easily be labeled. After that, using the DataStore class, the values are stored in a DataStore object. The values are converted to Newtons with the CalculateForce class before storing them in the DataStore object. Lastly, the runnable is set to run in the background. This allows the data to be continuously updated and updates the TextViews and the heatmap. See the code snippet 4.3 that illustrates the onCharacteristicChanged() method.

```
1  public void onCharacteristicChanged(BluetoothGatt gatt,
    BluetoothGattCharacteristic characteristic) {
2    super.onCharacteristicChanged(gatt, characteristic);
3    // store characteristic values in a Buffer
4    ByteBuffer buffer = ByteBuffer.wrap(characteristic.getValue());
5    // order the buffer
6    buffer.order(ByteOrder.LITTLE_ENDIAN);
7    // check which characteristic triggered this method
8    if (characteristic.getUuid().equals(deviceServiceCharacteristicUuid1
    )) {
9        arch  = buffer.getFloat();
10       met5 = buffer.getFloat();
11       met3 = buffer.getFloat();
12       met1 = buffer.getFloat();
13   }else if(characteristic.getUuid().equals(
    deviceServiceCharacteristicUuid2)){
14       heelR = buffer.getFloat();
15       heelL = buffer.getFloat();
16       hallux = buffer.getFloat();
17       toes = buffer.getFloat();
18   }
19   // Stored data in data store object
20   data.archVal  = CalculateForce.calculateForceADS(arch);
21   data.met5Val = CalculateForce.calculateForceADS(met5);
22   data.met3Val = CalculateForce.calculateForceArduino(met3);
23   data.met1Val = CalculateForce.calculateForceArduino(met1);
24   data.heelrVal = CalculateForce.calculateForceArduino(heelR);
25   data.heellVal = CalculateForce.calculateForceArduino(heelL);
26   data.halluxVal = CalculateForce.calculateForceArduino(hallux);
27   data.toesVal = CalculateForce.calculateForceArduino(toes);
28
29   runOnUIThread(()->{
30       runnable.run();
31       runnableTxt.run();
32   });
33
34 }
```

Listing 4.3: onCharacteristicChanged() method

4.2.6. FileHelper

This class is only a helper class that helps with the file directories to which the recorded data is saved. It consists of 4 methods. The first method is the init() method. This method ensures that the file directories exist and, if not, creates those file directories. The directories are also saved in a File data type. There are two directories, the root and the internal. The data is stored in a root directory to ensure that other applications cannot access the data or when the user has root access. The internal directory allows users to copy the data from their mobile device to another location. Because the data is also stored in the root directory, the data will always stay intact for the application. Then there is the getRoot() and getInternal(), which return the root and internal File data types, respectively. The last method is the getStoredFiles method, which returns all files stored in the root directory. This method is used to allow users to load stored files in-app.

4.2.7. FilePickDialog

This class is simply an extension of the base Android Dialog class; it serves as a basic file picker and uses the base Android ArrayAdapter with a simple list item layout to display files that can be picked in a pop-up. Once a file is selected, if this pop-up has a fileSelectedHandler, it is called with reference to the selected file and closes itself. A custom class, FileItem, was created to store file names and extensions; its toString is overridden because the ArrayAdapter will use this to populate the layout with text.

4.2.8. CustomTableLayout

This class extends the LinearLayout class and allows us to display data in a tabular format. The LinearLayout class allows us to add other Views to a View linearly. This means that Views are easily aligned in columns and rows. The CustomTableLayout class has an onCreateView() which contains the TextViews and the necessary formatting to have the View look as seen in figure 4.7

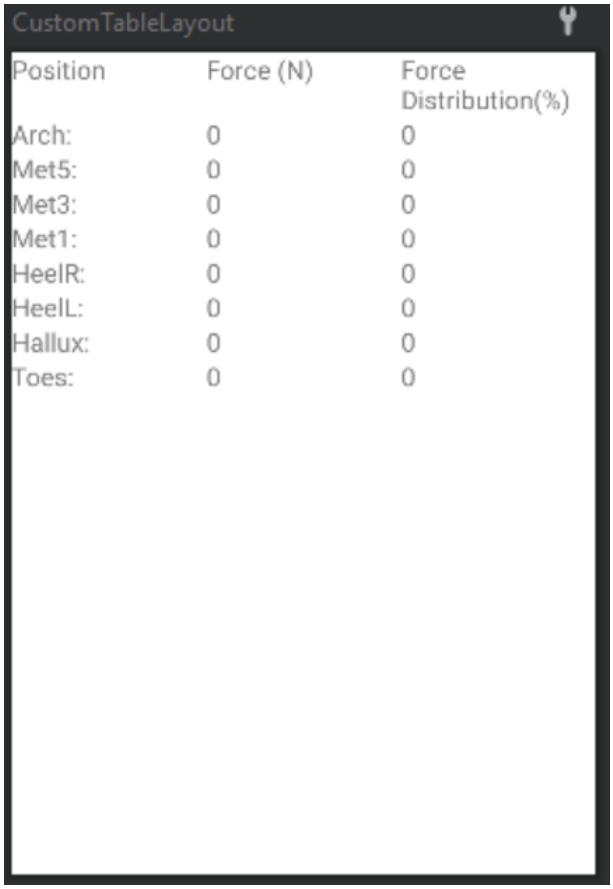


Figure 4.4: Preview of the CustomTableLayout View

The CustomTableLayout class also contains a method called updateData(). This method updates the table values with the values received through the GattHandler class. This method is called in side the MainActivity.

4.2.9. CalculateForce

This class allows for calculating the force applied to each cell using an exponential function. The exponential function was determined using linear regression and from the calibration done in section 5.2. With the tools provided by Desmos [23], all the points could be plotted, and an exponential could be formulated.

x_1	y_1	y_2
0	4096	4096
40	$2500 + \frac{500}{4}$	$3000 + \frac{500}{4}$
110	2000	2437
145	$1500 + \frac{500}{3}$	2032
570	1000 + 187	1500 + 156
750	1000 + 156	1532
1090	1000 + 125	1469
1740	1000 + 62	1470
2200	1000	1469
2860	1000	1440
3550	1000	1440
3980	1000	1420

(a) x and y values read from the calibration graph5.2

2

$y_1 \sim a + be^{(cx_1)}$

STATISTICS RESIDUALS

$R^2 = 0.9849$ e_1 plot

PARAMETERS

$a = 1074.92$
$b = 2915.43$
$c = -0.0119782$

3

$y_2 \sim a + be^{(cx_1)}$

STATISTICS RESIDUALS

$R^2 = 0.9927$ e_2 plot

PARAMETERS

$a = 1485.72$
$b = 2576.55$
$c = -0.0100719$

(b) Resulting exponential functions

Figure 4.5: Linear regression results

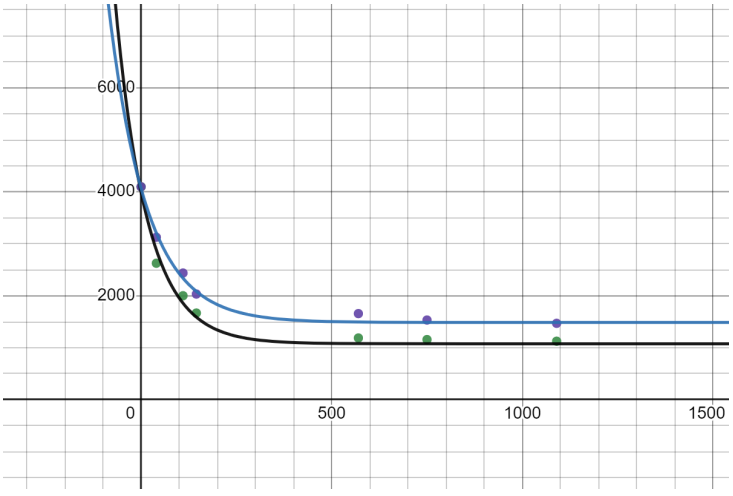


Figure 4.6: The plots of the two resulting exponential functions. The black graph represents the Arduino samples, and the blue graph is the ADS1115 samples

This class has two methods: calculateForceArduino () and calculateForceADS(). These methods take float values, the ADC readings received via BLE, as arguments and calculate the force values with the previously formulated exponential functions. See 4.4 on how the calculateForceArduino() method looks. The calculateForceADS() method looks similar, with only the a, b, and c variables differing.

```
1    public static float calculateForceArduino(float value) {
2      final double a;
3      final double b;
4      final double c;
5      a = 1060.84;
6      b = 2937.21;
7      c = -0.0117044;
8      return (float) Math.max(0, Math.log((value - a) / b) / c);
9  }
10
```

Listing 4.4: Method that calculates the force of the ADC readings from the Arduino

4.2.10. MainActivity

This class is the core of this Android application. It contains most of the initialization and functionality of buttons, TextViews, and other custom views. This class also implements the HBRecorderListener [24], a library that allows the application to record the device screen. The onCreate() is a method in the MainActivity that contains a large portion of the application logic. This logic includes initialization, declarations, and functionalities. The onCreate() starts by initializing all the views, buttons, the screen recorder, and the default runnable. A runnable is used to execute code on a thread.

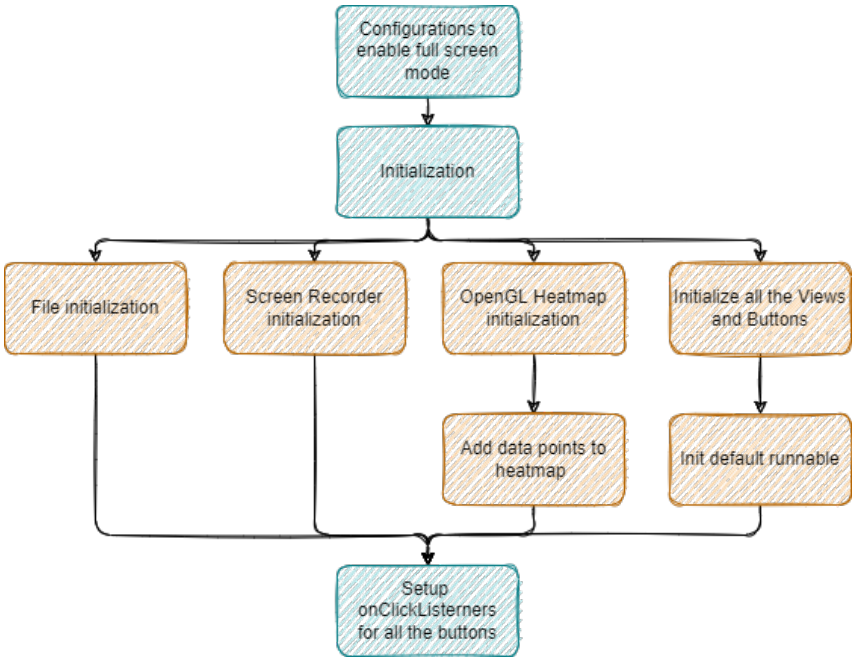


Figure 4.7: Overview of the onCreate() method

There are three buttons in the MainActivity, namely: connect, record, and load. These buttons have a setOnClickListerner() method, which takes a lambda function as an argument and executes the code within that lambda function when the specific button is pressed. When **connect** button is pressed, the init() method from the GattHandler class is executed with the current context of the activity, as its argument and the Android application attempts to connect to the prototype device via BLE. The **record** button, when pressed, executes the code that brings up a dialog with a text field, which is a field that takes the user-typed text and saves it as a string. This text field saves the name of the recording files, and after pressing the save button, the application starts recording the data and the device screen. Then if the record button is pressed while the application is recording, the recording is stopped, and the data is written to a file. The screen recording will be stopped and saved as well. Both data and screen recording files have the previously specified name as their file names. The recording process will be canceled if the cancel button is pressed instead of the save button. The **load** button, when pressed, will create a

FilePickDialog object. The FilePickDialog object spawns a pop-up that allows previously recorded files to be picked. The DataSeeker object created in the initialization will start playing the recorded file back in the application. The **play/pause** button executes the playPause() method from the DataSeeker class. This pauses or plays the recording if one is loaded.
There are also a few helper methods used in the onCreate(). The **startRecordinScreen()** method is used to start recording the device screen. The **startRecdingData()** method is used to copy live data to an Arraylist of type DataStore. This ArrayList will contain all the data to be saved to a text or CSV file. The **quickStteings()** method is used to set up the screen recorder. This method disables the audio and sets the video quality to standard definition(SD). The **createFolder()** method is used to set up the folder directory to which the screen recordings will be saved. This new directory is where the screen recordings will be saved. The **runOnUIThread()** method is a critical method that takes a runnable as an argument and executes the runnable code on the Main Thread without blocking the Main Thread. This allows all the code in the runnable to be continuously executed, and the application runs smoothly.

Chapter 5

Testing and Calibrations

5.1. Overview

Basic tests were conducted to ensure the device and application function as intended. After that, more tests were done by walking on different surfaces. By doing so, the system's capabilities could be displayed, and interesting observations could be made. An important thing to consider is that the shoe used is a running shoe, and running shoes have lifted heels to make a heel strike more comfortable. This can be seen in all tests that the heel sensors experience more pressure than the rest of the sensors. The temperature of the FSR cells was not considered, as FSR cells are not sensitive to temperature conditions, as seen in the specifications of many FSR devices. Please see figure E.1 in appendix E to understand the graphs' legends in this section.

5.2. Calibration

The previous student did the foot sensor calibration by using a compression testing machine, the Instron 3345. According to the previous student, this machine can apply a compression force up to 40kN. The machine was set up to apply a certain amount of force and slowly release. The force was applied to a single cell and the output ADC where recorded for this cell. Due to the machine's slow release all ADC values between the interval $F_N - 1 < F_N < F_N + 1$, where F_N is applied force, had been recorded and an average ADC value was calculated. The same test was done for the cells that are connected to the ADS1115. The results were recorded and plotted as seen in figure 5.2

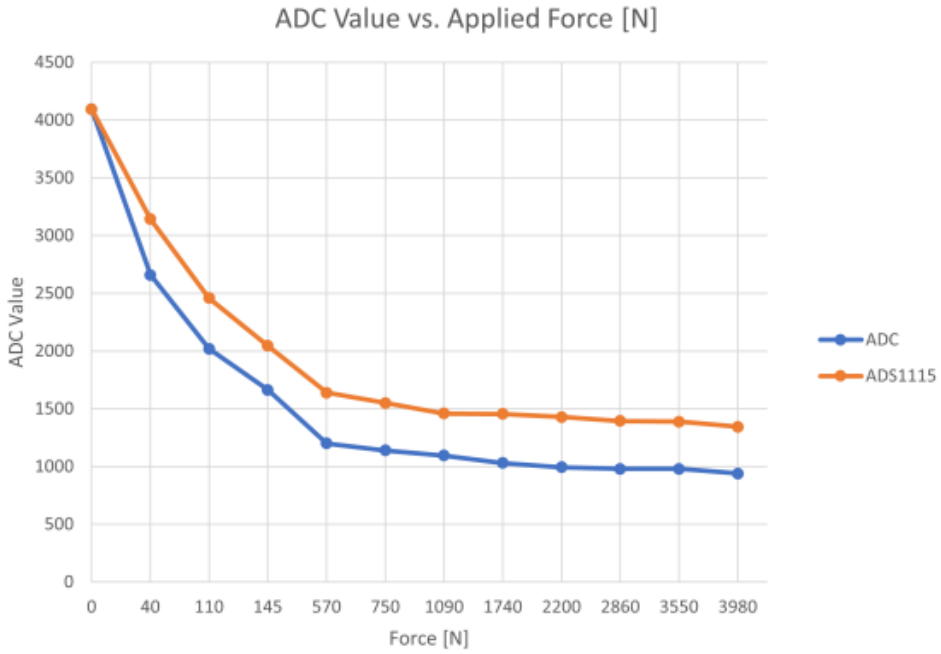


Figure 5.1: Sensor calibration graph

5.3. Applied force accuracy

The test for this section was conducted using two 2.5kg(≈24.5N) weights and putting them on a single FSR cell. When testing the accuracy of the force calculations, it was found that the two cells connected externally were not accurate. Another issue was that the FSR cells had a large surface area, which made it difficult to distribute the weight evenly across a cell. If the weight is not perfectly distributed, the readings are not accurate. It is impossible to distribute the weight evenly when the foot sensor is fitted into a shoe; therefore, the sensor is not very good for accurate force measurements.

Table 5.1: Force readings of each FSR cell when applying 2.5kg and 5kg

Cell	2.5kg applied(N)	5kg applied(N)
Arch	15.33	30.21
Met5	16.12	31.86
Met3	24.93	49.51
Met1	23.29	49.38
HeelR	24.04	48.21
HeelL	24.78	49.78
Hallux	24.44	49.37
Toes	24.44	49.11

5.4. Bare foot

By standing bare foot on the pressure we can see what a mid foot standing patterns should look like. This test will be used as a control as it will become apparent that the shoes influence how the device behaves.

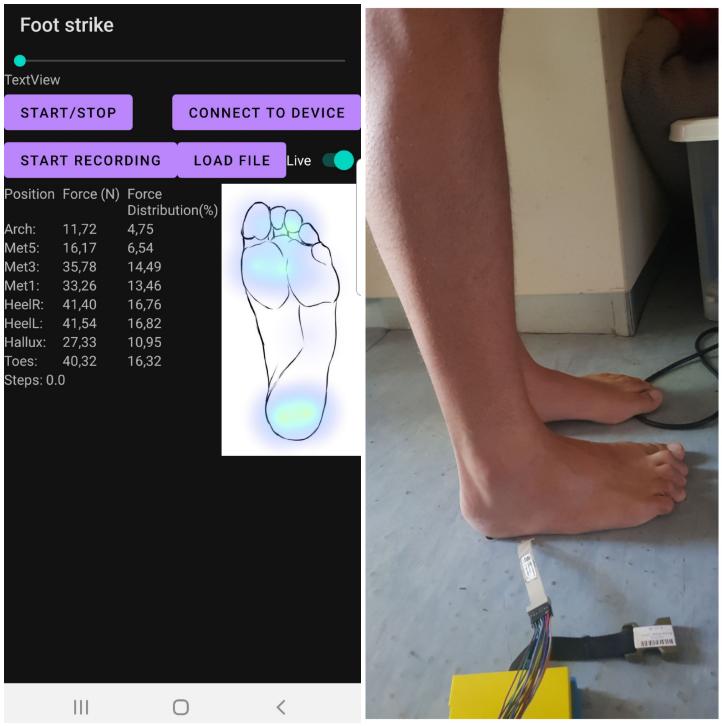


Figure 5.2: Bare foot mid-stance test

5.5. Basic strike patterns

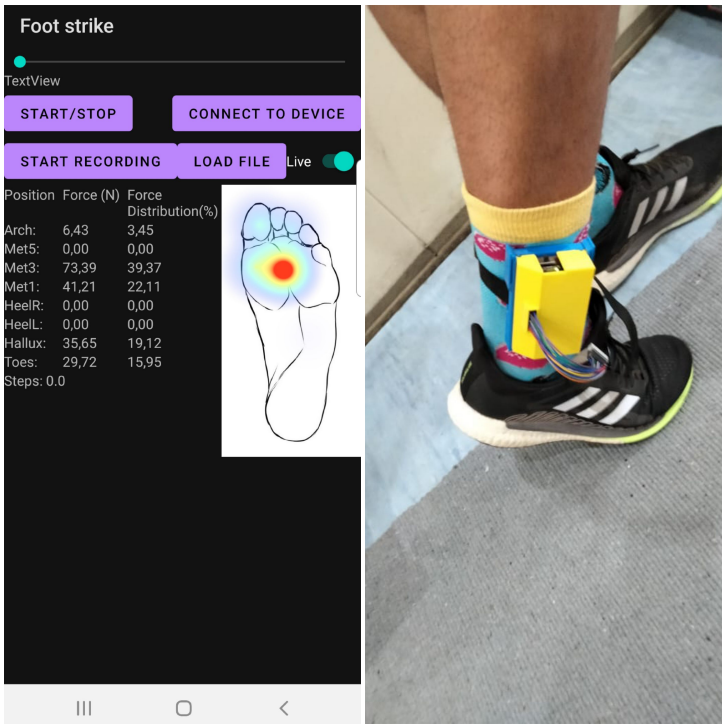


Figure 5.3: Fore foot push-off with shoes

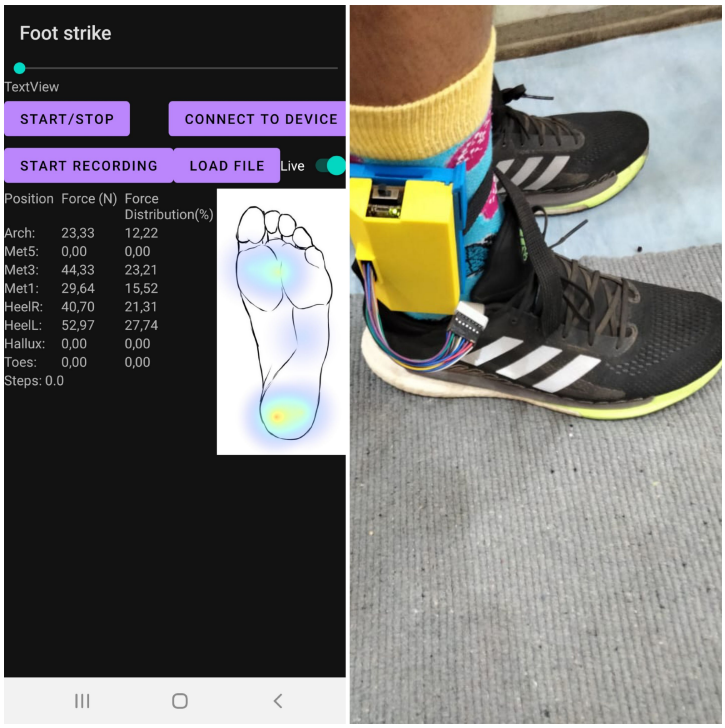


Figure 5.4: Mid-stance with shoes

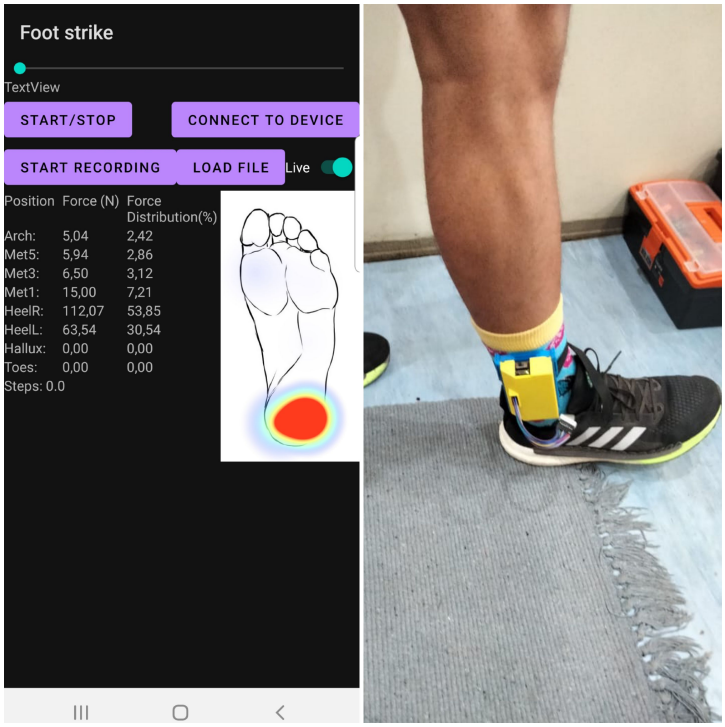


Figure 5.5: Heel-strike with shoes

5.6. Different surfaces

The following test was conducted from a standing start and then, at a steady rate, walking across two different surfaces. When comparing the following two graphs, we can see how the grass causes a damping effect. The damping effect makes sense as the grass is soft and absorbs some force applied to the FSR cells

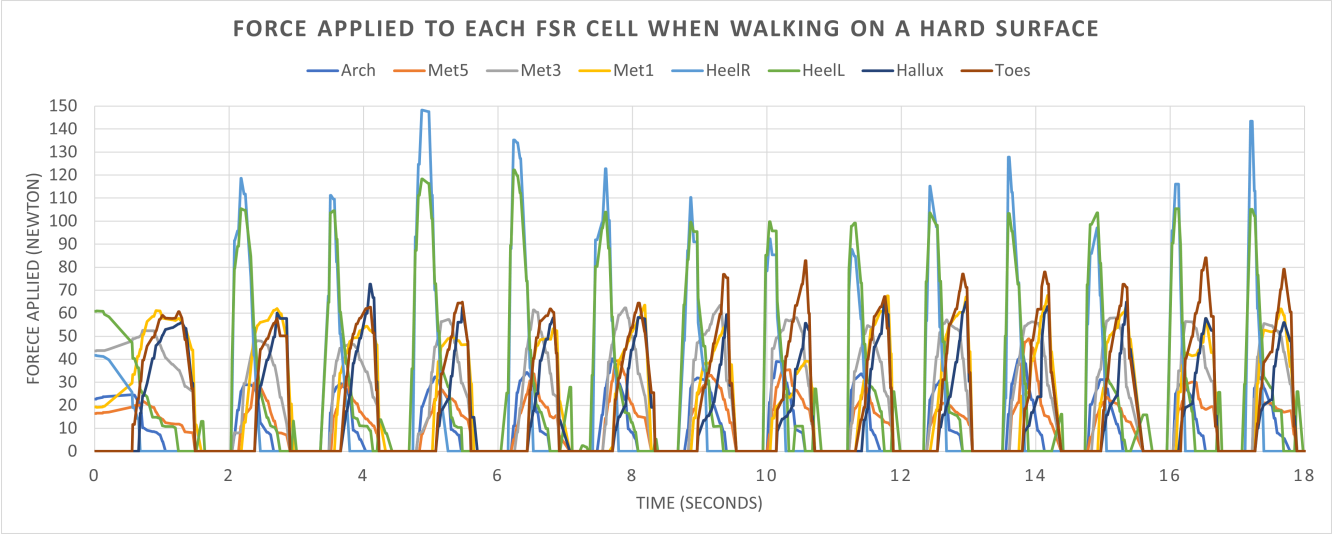


Figure 5.6: Graph that shows the force applied to each FSR cell when walking on a hard surface

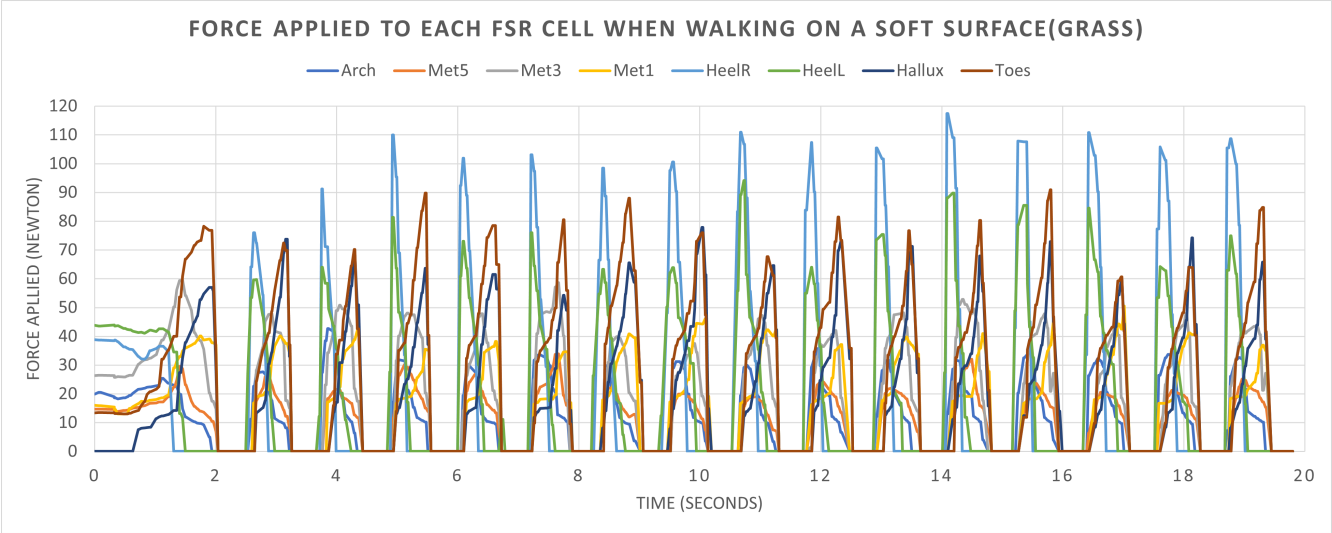


Figure 5.7: Graph that shows the force applied to each FSR cell when walking on a grass/soft surface

5.7. Inclines

The following test was conducted from a standing start and then walking up and down a hill or incline. The surface of the incline was hard and somewhat uniform. There is much less focus on the heel part of the foot when walking up and down. It is also clear that much shorter and more frequent steps was taken when walking downhill.

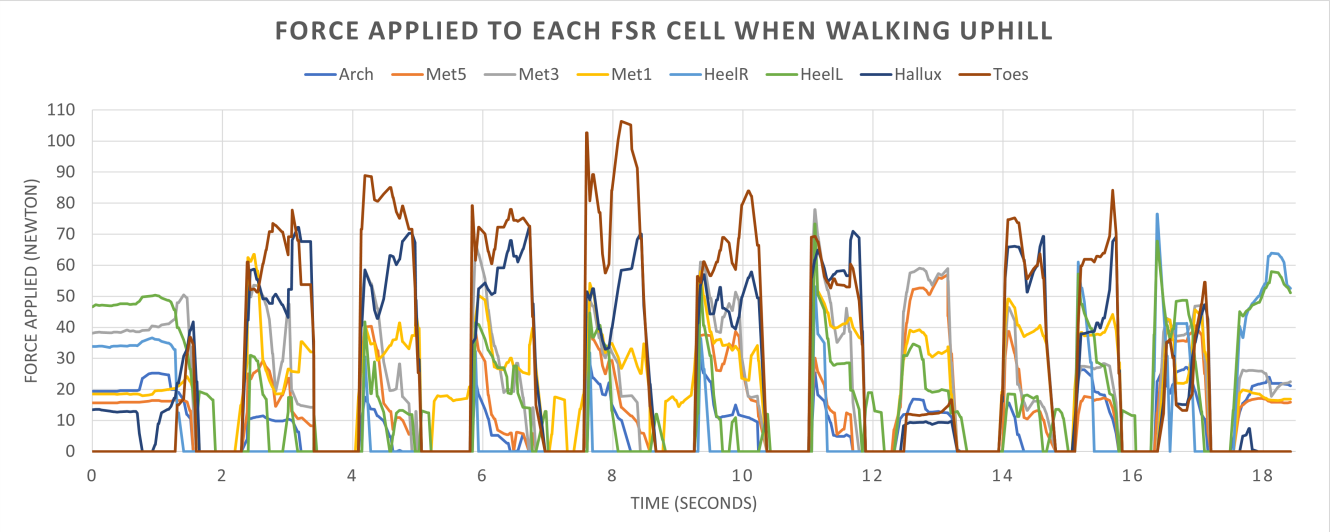


Figure 5.8: Graph that shows the force applied to each FSR cell when walking up a hill/incline

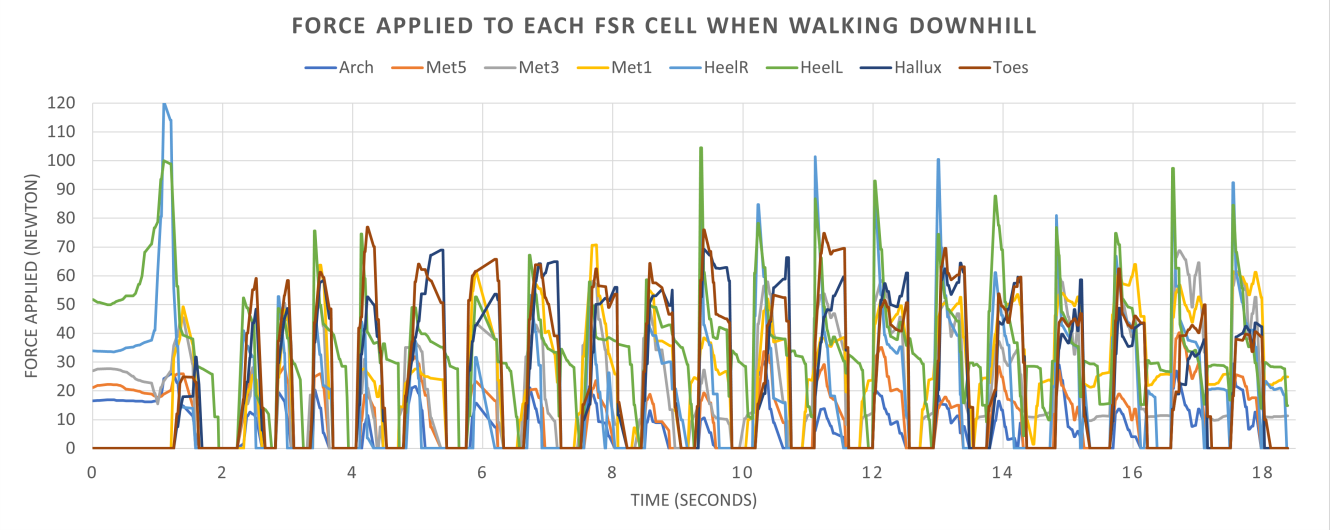


Figure 5.9: Graph that shows the force applied to each FSR cell when walking down a hill/incline

5.8. Stairs

The following test was conducted from a standing start and walking up and down stairs. The stairs had a small surface area; therefore, when walking on them, only a part of the foot would make contact. When walking up the stairs it is apparent that mainly the front area made contact with the ground. It is also interesting that the last step on the graph shows force applied to the heel area again, when the top of the staircase is reached.

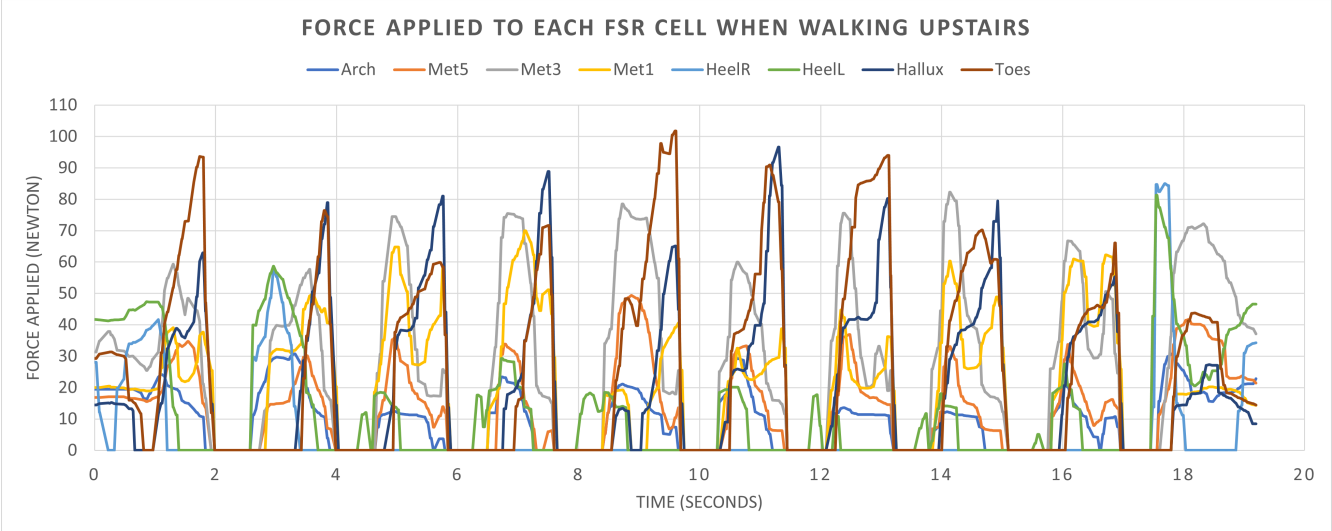


Figure 5.10: Graph that shows the force applied to each FSR cell when walking upstairs

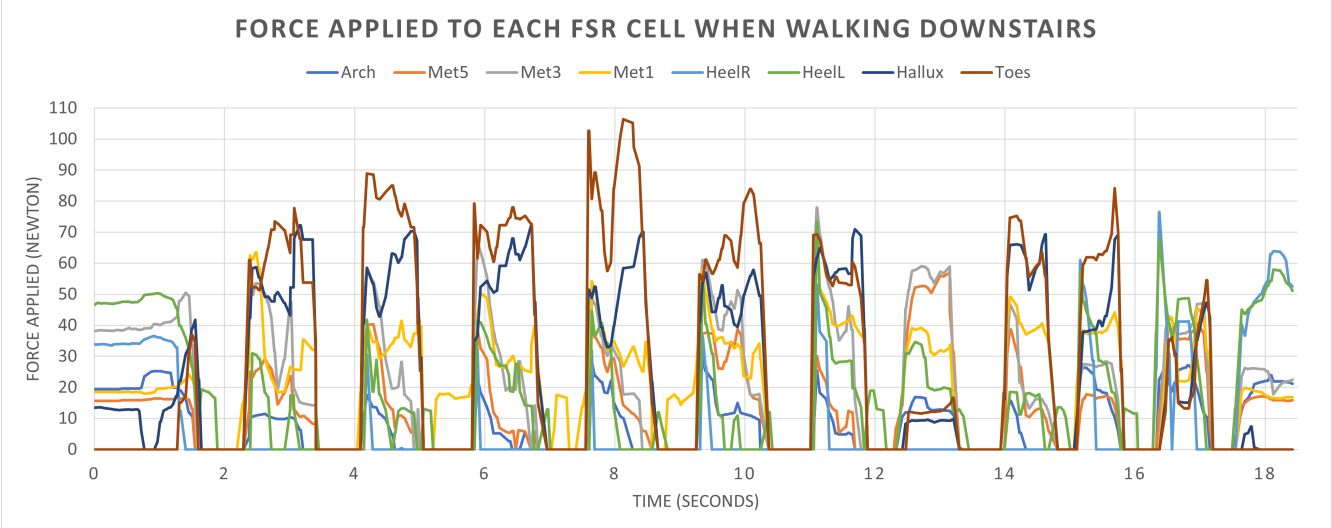


Figure 5.11: Graph that shows the force applied to each FSR cell when walking downstairs

Chapter 6

Summary and Conclusion

An Android application displaying foot strike patterns was successfully designed, developed, and tested. The final application has the following features:

- Bluetooth connection to specific Arduino device
- Heatmap that displays the data from the foot sensor
- Table that displays the calculated force and force distribution
- Ability to record video and comma-delimited data in CSV format. Both files are saved on the user's phone.
- Option to load previously recorded data and play it on the application

The application also had a feature that calculated the user's steps, but there was no use for this in the current state of the application. Not having two devices made it difficult to calculate the user's total steps for both feet. This feature will be added to further work to, for example, calculate cadence.

It was possible to conduct some tests using the device and application features. Some observations could be made, like seeing less pressure on the FSR cells when walking on grass than on a hard surface. It was also seen how walking uphill differs from walking downhill and what type of foot strike patterns all the tests yields. Another interesting observation was that the shoes used for testing had slightly lifted heels, influencing the strike patterns as the heel FSR cells received more pressure when standing in a mid-stance position.

Video recordings demonstrating the features of the device and application were shown to the staff of the Sports Science Faculty, and there was much interest in the device and application. The primary concern was the device, which indicates what is to be done in further work. The link to the videos mentioned follows. The following link is an example video of walking uphill [25], and this link [26] is an example video of walking downhill.

Some functionality could not be achieved due to time constraints and limitations. This is mainly because only one prototype device was available, and the prototype device had design choices that constrained the development of possible features.

Chapter 7

Further Work

7.1. Device

As mentioned in section 1.3 the current prototype does have some limitations. These limitations include that it is difficult to duplicate this device, and the weight and geometry of the device do not allow it to be used for running. The Arduino Nano 33 BLE also lacks the necessary analog pins. Therefore a re-design is recommended for the prototype device. A device like the ESP32 would better fit a prototype device. The ESP32 has a sufficient amount of analog pins, and some models have built-in charging circuits making it easier to build multiple copies of the possible design.

As mentioned in section 2.2 other sensors could be used for further analysis. The Arduino Nano 33 BLE has an accelerometer and gyroscope that are not used in the project's current state. These types of sensors should be considered when designing a new prototype device.

Another recommendation is to re-calibrate or refactor the foot sensor device. There are companies like Stappone [27] who have much more sophisticated devices. This indicates that it is possible to build a better sensor device. One of the foot sensors is faulty and has been marked. The 5 FSR cells at the top of the device are faulty.

7.2. Application

Lastly, exploring the capabilities of OpneGLES would allow for more advanced 3D rendered images or even live 3D movements, as this is all well in the capabilities of OpenGLES. If a new prototype device allows it, the application can have further functionality like:

- Cadence
- Strike length
- Power
- Speed and acceleration

Bibliography

[1] M. 4D. (2017, NA) Recognising foot strike patterns in runners. [Online]. Available: https://www.mass-4d.com/blogs/clinicians-blog/recognising-foot-strike-patterns-in-runners

[2] A. M. B. Monjoie. (2017, may) Opengl, a noob's guide for android developers. Youtube Video. [Online]. Available: https://www.youtube.com/watch?v=fnE8yOkNhv4&t=619s&ab_channel=AndroidMakers

[3] H. B. K, "Methods of running gait analysis," *Current Sports Medicine Reports*, vol. 8, pp. 136–141, 2009. [Online]. Available: https://journals.lww.com/acsm-csmr/Fulltext/2009/05000/Methods_of_Running_Gait_Analysis.10.aspx

[4] S. R. Department. (2020, nov) Running & jogging - statistics & facts. [Online]. Available: https://www.statista.com/topics/1743/running-and-jogging/#dossierContents__outerWrapper

[5] M. O. Almeida, I. S. Davis, and A. D. Lopes, "Biomechanical differences of foot-strike patterns during running: A systematic review with meta-analysis," *Journal of Orthopaedic & Sports Physical Therapy*, vol. 45, no. 10, pp. 738–755, 2015, pMID: 26304644. [Online]. Available: https://doi.org/10.2519/jospt.2015.6019

[6] P. R. Cavanagh and M. A. Lafortune, "Ground reaction forces in distance running," *Journal of Biomechanics*, vol. 13, no. 5, pp. 397–406, 1980. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021929080900330

[7] M. O. A. I. S. D. A. D. Lopes, "Biomechanical differences of foot-strike patterns during running: A systematic review with meta-analysis," *738-755*, vol. 45, no. 10, pp. 738–755, sep 2015. [Online]. Available: https://www.jospt.org/doi/10.2519/jospt.2015.6019

[8] H. F. H. . C. J. B. Laura M. Anderson, Daniel R. Bonanno, "What are the benefits and risks associated with changing foot strike pattern during running? a systematic review and meta-analysis of injury, running economy, and biomechanics," *Sports Medicine*, vol. 50, no. 5, p. 885–917, May 2020. [Online]. Available: https://link.springer.com/article/10.1007/s40279-019-01238-y

[9] K. P. C. L. J. R. P. G. Weyand, "Foot speed, foot-strike and footwear: linking gait mechanics and running ground reaction forces," *Journal of Experimental Biology*, vol. 217,

no. 12, jun 2014, https://journals.biologists.com/jeb/article/217/12/2037/12083/Foot-speed-foot-strike-and-footwear-linking-gait.

[10] M. E. K. X. cheng Liu Kyle G. Roberts and J. M. Valadao, "Foot-strike pattern and performance in a marathon," *Foot-Strike Pattern and Performance in a Marathon*, vol. 8, no. 3, pp. Foot–Strike Pattern and Performance in a Marathon, 1980. [Online]. Available: https://journals.humankinetics.com/view/journals/ijspp/8/3/article-p286.xml?content=abstract

[11] B. Richard, "Gait analysis methods in rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, vol. 3, no. 4, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021929080900330

[12] Y. Huang, H. Xia, G. Chen, S. Cheng, R. T. Cheung, and P. B. Shull, "Foot strike pattern, step rate, and trunk posture combined gait modifications to reduce impact loading during running," *Journal of Biomechanics*, vol. 86, pp. 102–109, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021929019301174

[13] J. Bagur. (2022, oct) Connecting nano 33 ble devices over bluetooth. [Online]. Available: https://docs.arduino.cc/tutorials/nano-33-ble-sense/ble-device-to-device

[14] M. Afaneh. (2017) Bluetooth gatt: How to design custom services & characteristics [midi device use case]. [Online]. Available: https://novelbits.io/bluetooth-gatt-services-characteristics/

[15] Tekscan. (2022) How does a force sensing resistor (fsr) work? [Online]. Available: https://www.tekscan.com/blog/flexiforce/how-does-force-sensing-resistor-fsr-work

[16] G. Seeedstudio. (2020) What is a force sensing resistor (fsr sensor)? [Online]. Available: https://www.seeedstudio.com/blog/2020/10/27/what-is-a-force-sensing-resistor-fsr-sensor/

[17] media.ccc.de folkert. (2019, aug) Introduction to opengles and glsl programming. Youtube Video. [Online]. Available: https://www.youtube.com/watch?v=fL957hgX9kA&t=535s&ab_channel=media.ccc.de

[18] A. Developers. (2022) Opengl es. [Online]. Available: https://developer.android.com/develop/ui/views/graphics/opengl/about-opengl#basics

[19] ANTBATT. (2019) Lifepo4 battery cell 18650-3.2v-1500mah. [Online]. Available: https://www.antbatt.com/wp-content/uploads/2019/09/18650-3.2V-1500mAh-DataSheet.pdf

[20] D. Snyman. (2022, oct) Skripsie. GitHub repository for final year skripsie by Dewaldt Snyman. [Online]. Available: https://github.com/22706852/Project-E--448_skripsie

[21] A. Developers. (2022, jul) Bluetoothadapter. [Online]. Available: https://developer.android.com/reference/android/bluetooth/BluetoothAdapter

[22] ——. (2022, jun) Bluetoothdevice. [Online]. Available: https://developer.android.com/reference/android/bluetooth/BluetoothDevice

[23] Desmos. (2022) Desmos. [Online]. Available: https://www.desmos.com/calculator

[24] J. F. T. Frysinger. (2022, jun) Hbrecorder. GitHub repository for a library used to record a device screen in Android. [Online]. Available: https://www.statista.com/topics/1743/running-and-jogging/#dossierContents__outerWrapper

[25] D. Snyman. (2022, oct) Example video of a foot strike application and device when walking uphill. Youtube video demonstration. [Online]. Available: https://youtu.be/By-dD7hpOh0

[26] ——. (2022, oct) Example video of a foot strike application and device when walking downhill. Youtube video demonstration. [Online]. Available: https://youtu.be/da2boUO0qt4

[27] stAPPtronics. (2022) stappone. [Online]. Available: https://www.stappone.com/en/

Appendix A

Project Planning Schedule

Date	Week	Description
Jul 18-23	1	Project planning and scope
25-30	2	Study prototype devices and how it can be used.
Aug 01-6	3	Literature study on the topic at hand. This is everything that is correlated to running and foot strike patterns
8-13 and 15-20	4 and 5	Start developing the Arduino code for the device. Try to get basic ADC readings from the IEEE foot sensor The previous student could not provide previously written code.
22-27	6	Reach BLE and how it can be used to transmit data from an Arduino device. Start developing the code as soon as I have good understanding of BLE for Arduino
Aug 29 - Sept 3	7	Test Week
5-10	8	Recess
12-17 and 19-24	9 and 10	Start building the Android application and understand BLE regarding Android devices. Display the data transmitted from the Arduino in a text view
26- 1 Oct and 3-8	11 and 12	Build a heatmap view using OpenGL—ES. This requires a good understanding of OpenGL—ES and custom Android views
10-15	13	Use the previous students calibrations to build another custom view that displays the force exerted in newton on each cell
17-22	14	Test the application. This includes using the device on difference surfaces and walking up or down a hill. See if temperature has an effect on the cell readings
24-29	15	Present application to a person who has knowledge of foot strike patterns to see if an application of such would be of use in the industry and reflect on findings
31 October 2022	16	Report Deadline. Prepare for oral examination

Appendix B

Outcomes Compliance

ELO	Chapter	Description
1. Problem solving	1,2,3,4	Chapter 1 demonstrates the identification and formulation of an engineering problem. Chapter 2 and 3 demonstrates the analysis of the problem and previous solutions Chapter 4 demonstrates how the problem was solved by developing an android application
2. Application of scientific and engineering knowledge	3,4,5	Chapter 3 and 4 demonstrates the use of mathematics and engineering fundamentals to design and program an application that can do advance calculations.
3. Engineering Design	3,4	Chapter 3 demonstrates the understanding of engineering design. Chapter 4 demonstrates the design and synthesis of and engineering idea by developing Arduino and Java code.
4. Investigations, experiments and data analysis	5	Chapter5 demonstrates the investigations of different ways to test the device an application. The data could be retrieved from the various tests and the data was analyzed.
5. Engineering methods, skills, and tools, including Information Technology	4,Appendix C	Chapter 4 demonstrates the use of programming skills in C++ and Java. In Chapter 4 mathematical tools such as Desmos was used. Appendix C demonstrates the design and print of a 3D model using ZBrush
6. Professional and technical communications	6	Chapter 6 demonstrates that the device an application were demonstrated to people in the field of sports science. Their feedback can be used for future work and recommendations.
7. Individual Work	All	The whole of the project was conducted individually
8. Independent learning ability	2,4	Chapter 2 demonstrates the ability to learn new concepts and skills and chapter 4 demonstrates how these newly found concepts and skills were implemented.

Appendix C

3D Model of device casing

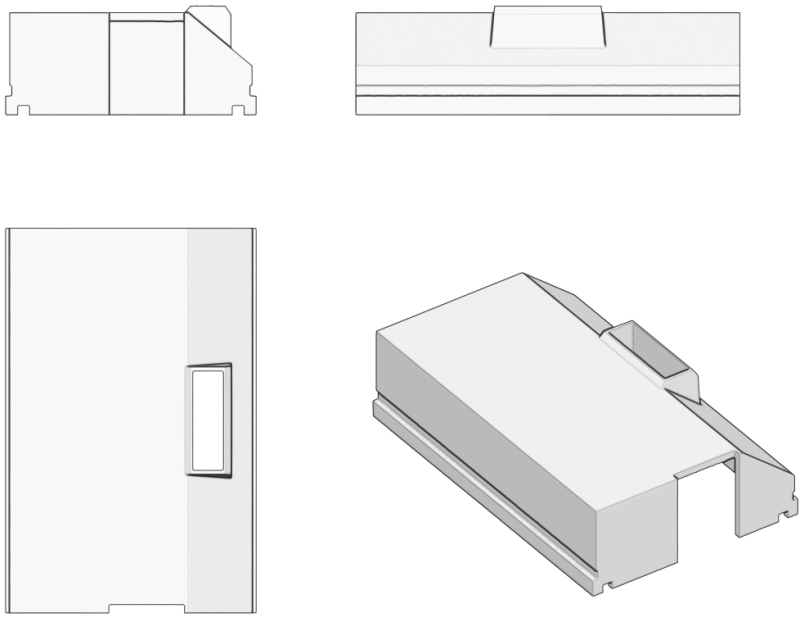


Figure C.1: Top shell 3D model

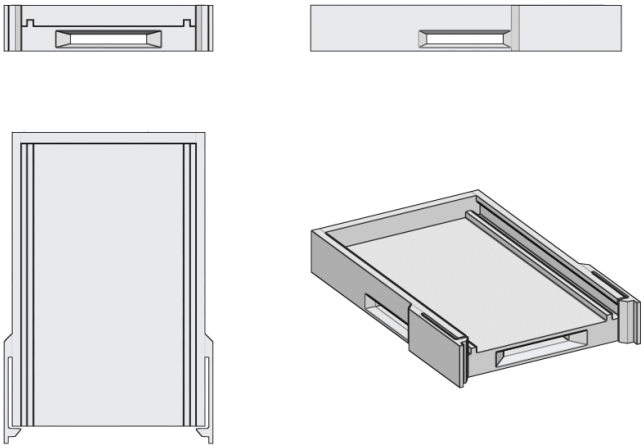


Figure C.2: Base 3D model

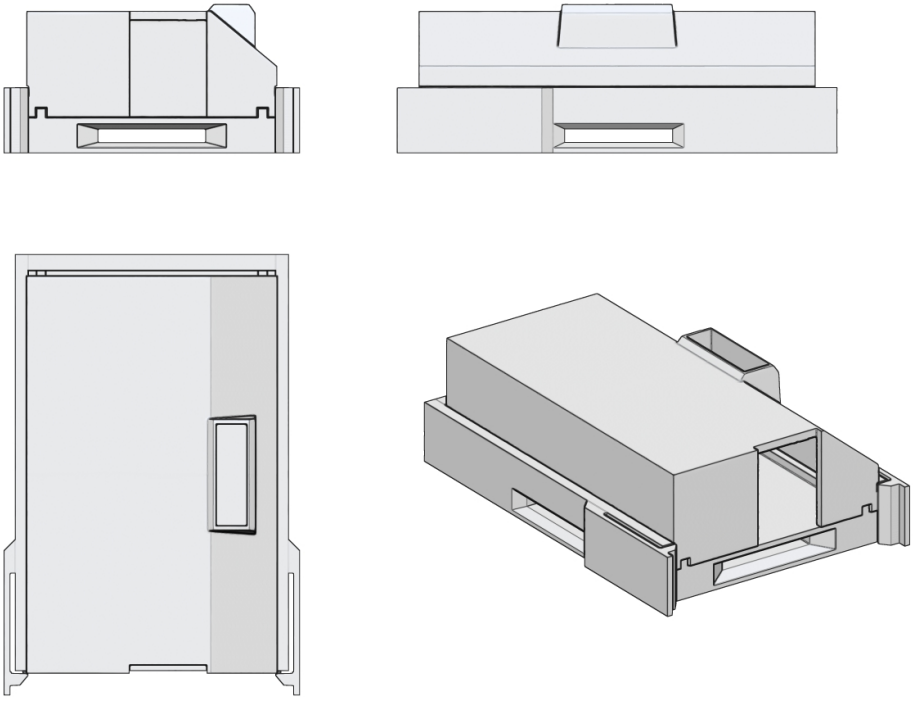


Figure C.3: Assembled base and shell

Appendix D

Prototype Device

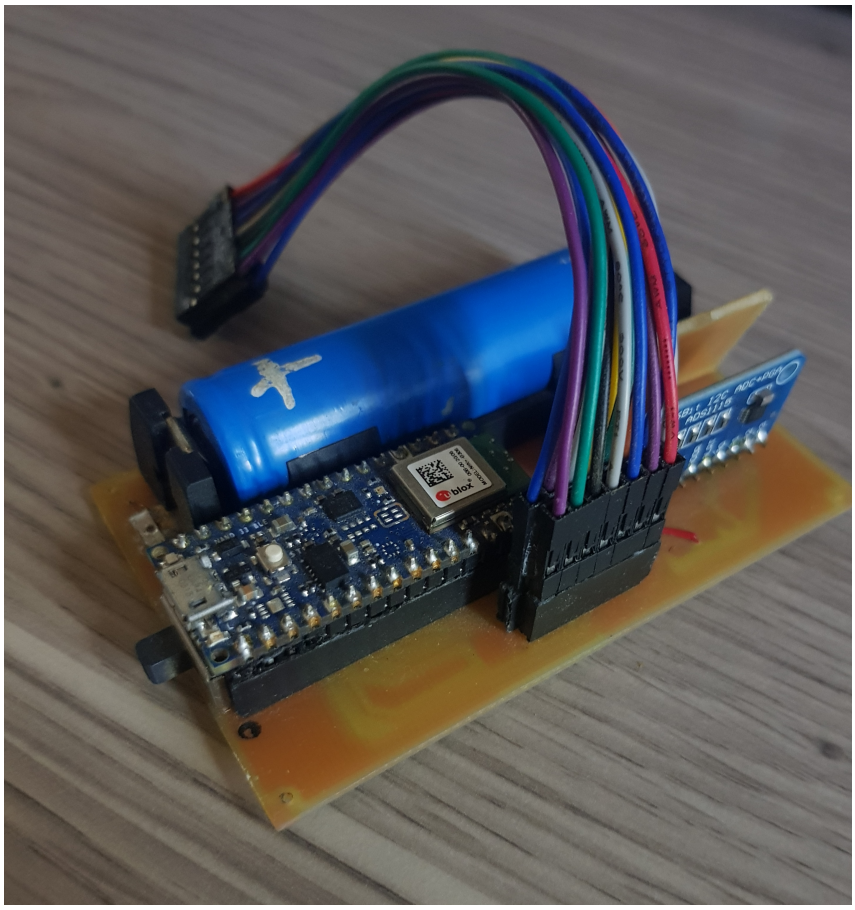


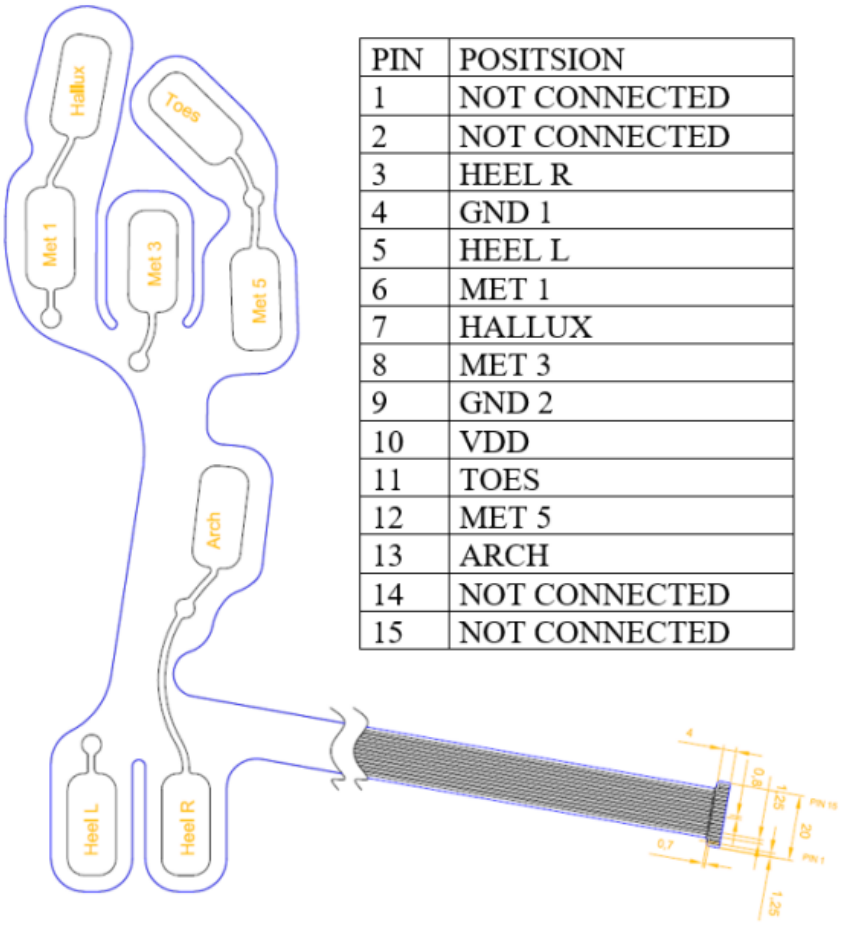
Figure D.1: Prototype device without the foot sensor connected

Figure D.2: Prototype device with the foot sensor connected

Figure D.3: Prototype device fitted in a casing with the foot sensor connected

Appendix E

Smart Footwear Sensing Solutions by IEE



PIN	POSITSION
1	NOT CONNECTED
2	NOT CONNECTED
3	HEEL R
4	GND 1
5	HEEL L
6	MET 1
7	HALLUX
8	MET 3
9	GND 2
10	VDD
11	TOES
12	MET 5
13	ARCH
14	NOT CONNECTED
15	NOT CONNECTED

Figure E.1: IEE foot sensor specifications and labeling for each cell