Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Learning for sensor-based, real-time fall detection for cyclists.

Aidan Mongan
14334583

April 22, 2019

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of
B. A (Mod.) Computer Science

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

Like all extreme sports, mountain biking comes with the potential for serious injury to the rider in the event of an accident. Non fatal injuries can easily become fatal, when one is alone, far from help and potentially incapacitated. A study conducted by Paracelsus Medical University recorded injury rates as high as 16.8 injuries per 1000 hours of riding, with 22 being moderate and 16 being severe, with rider error being the leading cause (1). An automated crash detection system has the potential to be life saving in the worst of circumstances.

Existing discipline-specific solutions e.g., for road only or for mountain use only, on both hardware (Specialized's AnGi) and software (Strava Beacon,Garmin) have inflexible detection algorithms focusing on using thresholds for only one to two data points. For example AnGi records values from its inbuilt gyroscope and accelerometer, while Garmin's system uses only accelerometer values. Such threshold-based solutions pose issues in terms of high false detection rates and a single threshold value is unlikely to be suitable for different users at different skill levels.

This project expands on previous research done in the area of wearable fall detection devices for the elderly, focusing on the design of a software solution for real-time fall detection. Three data points are used: raw sensor data from both a tri-axial gyroscope and a tri-axial accelerometer as well as the rate of change of speed, calculated via GPS. The proposed system utilizes learning techniques to improve detection rates and over time generates a more personalized model. Based on pre-captured training data of both regular riding and crashes, data is classified using a multivariable logistic regression model in real-time to determine whether a crash has occurred. Raw sensor data is captured from the inbuilt sensors present on android smartphones.

This approach is implemented as an android application called "RideSafe" and was evaluated using a user study, comprising of 16 participants at local trail centres over a 2 day period. Crash data was also collected, by means of intentional crashes in a controlled environment for verification. Results show that this system can successfully detect upwards of 80% of crashes with a low rate of 18% false positives.

Acknowledgements

I would like to thank my supervisor Siobhan, for her continuous support over the past 6 months. My sincere gratitude to both family and friends for their invaluable advice and support received over the past 5 years. I also wish to thank my tutor Dr. Hugh Gibbons, without whom, I may not have made it through to my final year. Finally a special thanks to everyone who partook in this study, especially Matt from Glencullen Adventure Park for allowing testing to take place on site.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

Personal experiences were the main driving factor in my motivation to pursue this study. As a mountain biker with 10 years of experience the author has sustained their fair share of minor injuries, but witnessing injuries sustained by more venerable fellow riders are sometimes more impactful. Last summer on a seemingly normal spin with a friend, we discovered a woman lying injured off on the trail side, incapacitated and unable to call for help so we did on her behalf. Multiple phone calls later to aid the first responders in locating us they arrived - around 1 hour after impact. This experience made the author realize how useless your mobile phone is to you in these situations when one is unable to even pick it up. Non fatal injuries can easily become fatal, when one is alone, far from help and potentially incapacitated.

1.2 Aims

The aim of this project was to develop an android application for real-time fall detection for cyclists, automating the process of requesting assistance, and to reduce response time in the event of an accident. Before development of the application the author set himself strict aims to achieve.

Simplistic and Intuiative

After the initial set up process, to carry out the main use case: crash detection would be started and stopped with a single press of a button. Start the service, put your phone in your pocket and enjoy your time on you bike with piece of mind. Simple and convenient to use, removing the possibility of confusion for the end user, as the end users will be members of the general public. A simple user interface is important as the setting to which the app would be outdoors in potentially harsh weather conditions, external factors such as glare from the sun and the possibility of moisture on the screen make high detailed, small user interface elements unsuitable. Less is more in this scenario.

Diverse

Many existing systems are discipline specific, only working for one aspect of cycling i.e., for cross country usage only. The author intends this app to have the potential to work for all disciplines of cycling. Targeting single disciplines would drastically reduce the number of potential users as well as producing highly undesirable, inaccurate results if used for the incorrect discipline.

Standalone

Utilizing android smartphones built in sensors removes the need for extraneous external equipment for ride monitoring. The author intends the app to work as expected with one's phone placed in their pocket or bag, requiring no extra mounting equipment for either the rider or the bike.

Enjoyable user experience

Many existing solutions exhibit deal breaking issues which ultimately causes the end user to stop using the system, the author aimed to eradicate the pitfalls present in other systems leading to a better user experience.

Efficiency

Performance in terms of battery usage is of utmost importance, heavy battery usage would have the potential to kill the phone when one would need it most - in an emergency. Every possible optimization in terms of battery will be made where possible - without impacting performance.

1.3 Personal Goals

In addition to the aims of this project the author had set some personal goals to achieve from undertaking this project.

Develop a fully functional application

Having had brief experience working with android studio before undertaking this project to develop simple applications, most of which were interfaces for arduino circuits connected via bluetooth, the author had never developed such a large scale complex application prior to this project. The author was excited to broaden his skill set and develop an application ready to be published to the google play store.

Work with Embedded sensors

Having experience working with microcontrollers and various sensors, the author was excited to utilize the plethora of available sensors present in android smartphones today. An investigation will be conducted to identify the most suitable sensors to utilize for this project.

Collect and Analyse Real World Data

Datasets for what a bike accident looks like in terms of sensor values are few and far between, the author was excited to conduct their own research with many unknowns to which would need to be investigated and discovered. Having very few similar documented studies available they were very interested to study this particular system in the domain of cycling.

Real world testing

The author was aware before undertaking this study that it would involve a lot of real world data collection, analysis and testing. Being a crash detection application testing could not be simulated sitting at a desk, which meant all testing would need to be conducted in the real world which proved both challenging and exciting.

1.4 Readers Guide

2 - Background

This section will discuss the concept of fall/accident detection, exploring both the existing cycling and medical applications. The main approaches to the problem of fall detection will be discussed and the pros and cons of each type of system will be discussed.

3 - Design

Chapter 3 will discuss the design of RideSafe in terms platform choices, User interface design as well as the design of the systems underlying architecture design.

4 - Implementation

Chapter 4 will discuss the methods which were implemented in the development of RideSafe,on a per activity basis. A detailed explanation of the methodologies used to collect crash data and train the applications model will also be contained in this chapter.

5 - Evaluation

Chapter 5 is a discussion of results from various tests RideSafe was subjected to. Results from both real world on location testing as well as CarPark tests will be displayed and analyzed. Performance metrics for different aspects of RideSafe will be discussed.

6 - Conclusion

In Chapter 6 the author will discuss potential future uses for RideSafe as well as features which could be added and improved with further work. Conclusions from the the project itself will also be discussed.

2 Background

Accidental falls and accidents resulting in injury is a large scale world wide problem, A study conducted by the American Journal of Public Health found that one of the leading causes of death is falling, for people aged 50 or above , a 136% increase over a 30 year period(2). Accidents are not aged biased however certain sports come with a higher probability of accidents occuring. Like all extreme sports, mountain biking comes with the potential for serious injury to the rider in the event of an accident. In 2018 there were 29,000+ registered members of cycling Ireland in over 450 clubs (3) , with this number set to increase year on year. With an increase of the number of people taking up the sport, the amount of reported injuries has also increased significantly. A study conducted by Paracelsus Medical University recorded injury rates as high as 16.8 injuries per 1000 hours of riding, with 22% being moderate and 16% being severe (1). The most common mechanism of acute severe injury for mountain bikers has been found to be falling forward (64.9%), which lead to the most severe injuries, with an ISS [1] of 3.4 out of a possible 6 (currently untreatable) (4)

Existing solutions for fall detection for both medical (Fall detection for the elderly) as well as existing crash detection systems for cyclists will be discussed and explained in this chapter.

[1] Injury Severity Score

2.1 The Concept

Fall detection for the elderly has been around for decades, the original idea came to fruition as a result of the impracticality of having around the clock care for the independent, but fragile members of society. With high success rates and the advancement of technology in recent years the concept has been applied to many different uses cases, such as accident detection in vehicles and sports.

2.2 Existing Solutions - Medical

Fall detection is not a new concept in the medical domain, The first fall monitoring system (PERS[2]) Hausnotruf was developed in the 1970's by Wilhelm Hormann These early active systems were manually triggered, requiring the user to activate them by pressing a button on a transceiver, while relying on an active phone line in the home to contact help. Active systems have been mostly phazed out due to their main weakness - the user must be conscious and capable of triggering the alarm. Passive systems are now commonplace and most relevant to this study. Passive systems monitor the users movement and trigger the alarm without the need for user interaction with the system. Passive systems can be implemented in one of two ways: Vision Based approaches and wearable sensor based solutions. Currently Sensor based solutions are widely available, while computer vision implementations are mostly experimental proof of concepts.

2.2.1 Vision Based Approaches

Computer Vision Approaches are currently the state of the art in terms of fall detection for the elderly, these systems comprise of multiple cameras set up in the home, with a base station analysing the video footage to determine if the user is in need of assistance. Implementing Computer Vision techniques such as optical flow or Gaussian Mixture Models, to focus on moving objects and ignoring the background, allows for recognition of falls or slips. Many environmental issues such as changing lighting and limited view of cameras in the home impede the accuracy of these systems. Unfortunately for these systems it is not possible to constrain the environment enough to allow for near perfect accuracy. Vision based approaches also pose a large risk to the occupants privacy. 24/7 surveillance in the home is not something anyone desires, especially in what is supposed to be the comfort of your own home. Depending on the security utilized, vulnerabilities may exist exposing the stream to the world.

Cost is a large factor in why sensor based solutions are prefered, the cost of initial setup is

[2]Personal Emergency Response System

much higher for a computer vision solution. Adding complexity to an implementation while simple more effective solutions, with similar or in some cases better results is a step in the wrong direction for now, however as technology further improves vision solutions will be more viable to bring to market.

2.2.2 Sensor Based Approaches

Wearable sensor based solutions are very common globally, in both passive and active products. These wearable devices are usually in the form of a small external device.

These devices can be:

- Wrist mounted
- Attached to a belt
- A Pendant worn around the neck.

Housed within these enclosures are embedded sensors, the most common of which is the triaxial accelerometer. The triaxial accelerometer measures proper acceleration in 3 perpendicular axis (g-force). Almost every wearable fall detector factors in accelerometer values while determining if a fall has occurred or not. The second most common sensor used in fall detection is the triaxial gyroscope, used to calculate rotational speed. Gyroscopes have proven to be useful for understanding the direction of a fall. Depending on where the device is worn, some false positives which would occur using only an accelerometer can be disregarded, such as sitting down too quickly. A pendant style device's relative rotation would be more or less unchanged sitting down vs standing. The use of highly sensitive barometric sensors have also been utilized to measure minute changes in atmospheric pressure, the difference between pressure when one is standing vs lying down can be distinguished between. The most popular commercially available fall detection using barometric sensors is lifeline (5). There is no standard passive implementation, solutions include one or more of the above sensors along with an algorithm or formula to determine if a fall has occurred. A comparison of sensors most commonly used can be seen in Table 2.1.

Table 2.1: Comparison Of Sensors Used In Existing Solutions

Sensor Type	ANGI	Garmin	Apple Watch	Philips Lifeline	MobileHelp
Accelerometer	Yes	Yes	Yes	Yes	Yes
Gyroscope	Yes	No	Yes	No	No
GPS (Detection)	No	No	No	No	No
Barometer	No	No	Yes	No	No

2.3 Exsisting Solutions - Cycling

Existing solutions for cycling are in their infancy relative to medical applications, with technology having become much more advanced in terms of efficiency and physical size, more portable and independent systems have become more feasible to implement. Implementations range from additional software provided with existing products to standalone dedicated hardware solutions.

2.3.1 Garmin

Garmin, best known for producing GPS based navigation systems, include incident detection with their cycle computers Edge 1030 and above. Garmins incident detection is one of the most basic solutions available on the market today - A simple threshold based system using only a Gyroscope for detection. " Incident detection should not be relied on as a primary method to obtain emergency assistance. " - Garmin. Incident detection is for road use only resulting in it being next to useless for a large percentage of their customers - mountain bikers. Garmin cycle computers are very expensive, however they perform their main intended use (Navigation) to an excellent standard. The limitations of systems such as this will be discussed in Section 2.4, threshold based solutions.

2.3.2 Specilized's ANGI

In late 2018 Specialized released their crash detection solution - ANGI [3], (6) building upon technology acquired from icedot in 2017 (7).

ANGI is a threshold based system comprising of three components:

- Compatible Helmet
- ANGI Sensor
- Smartphone

[3]ANGI - Angular and G-Force indicator

The ANGI sensor is a small device which mounts to the rear of a compatible helmet, consisting of a bluetooth chip, an accelerometer and a gyroscope. While connected to a compatible smartphone ANGI measures linear and rotational forces present at the riders helmet. When a crash is detected the users mobile phone is used to contact an emergency contact. The ANGI sensor is relatively inexpensive on its own (50 euro), but factoring in the cost of a compatible helmet and the yearly subscription fee (alike many medical solutions) the solution becomes less budget friendly. Utilizing external sensors like ANGI allows for consistent placement relative to the rider, but utilizing bluetooth connectivity adds an extra step needed to use the device as well as introducing latency and reliability concerns.

2.4 Threshold based solutions

The most simple sensor based implementations are threshold based, meaning they compare current sensor readings to a predefined threshold value at a given moment in time. If the current value(s) surpass the threshold(s) a fall or crash will be flagged. This simple implementation focuses solely on a single value from a single point in time, context of what the previous value or the next value is not taken into account, as previously mentioned before garmins incident detection uses this implementation. With relatively low accuracy rates sole threshold based solutions are rarely the ideal choice for a system.

For the application of mountain biking, there are three main reasons to why a threshold based solution is unsuitable:

- Tend to be discipline specific:

To compute a single threshold for multiple disciplines of cycling is infeasible, the forces experienced during a commute are on average far lower than what is experienced while mountain biking. A system intended for road use only would trigger seconds into a mountain bike trail. (maybe image of trail beside road).

- Unsuitable for different riders of different skill levels:

With cycling in general, as the riders skill and experience increases , their speed also increases resulting in higher forces experienced. Single threshold values are unsuitable among different riders and will produce poor results. A comparison of the forces experienced by two different riders of different skill level, on the same section of trail can be observed in Figure 2.1

- High rate of false positives:

Accidental non crash actions such as dropping the device, or shaking it rigorously can easily fool a threshold based systems (see Table 5.1) . false positives can have serious legal repercussions if the system is programmed to contact the local emergency services, potentially diverting limited resources away from a true emergency.

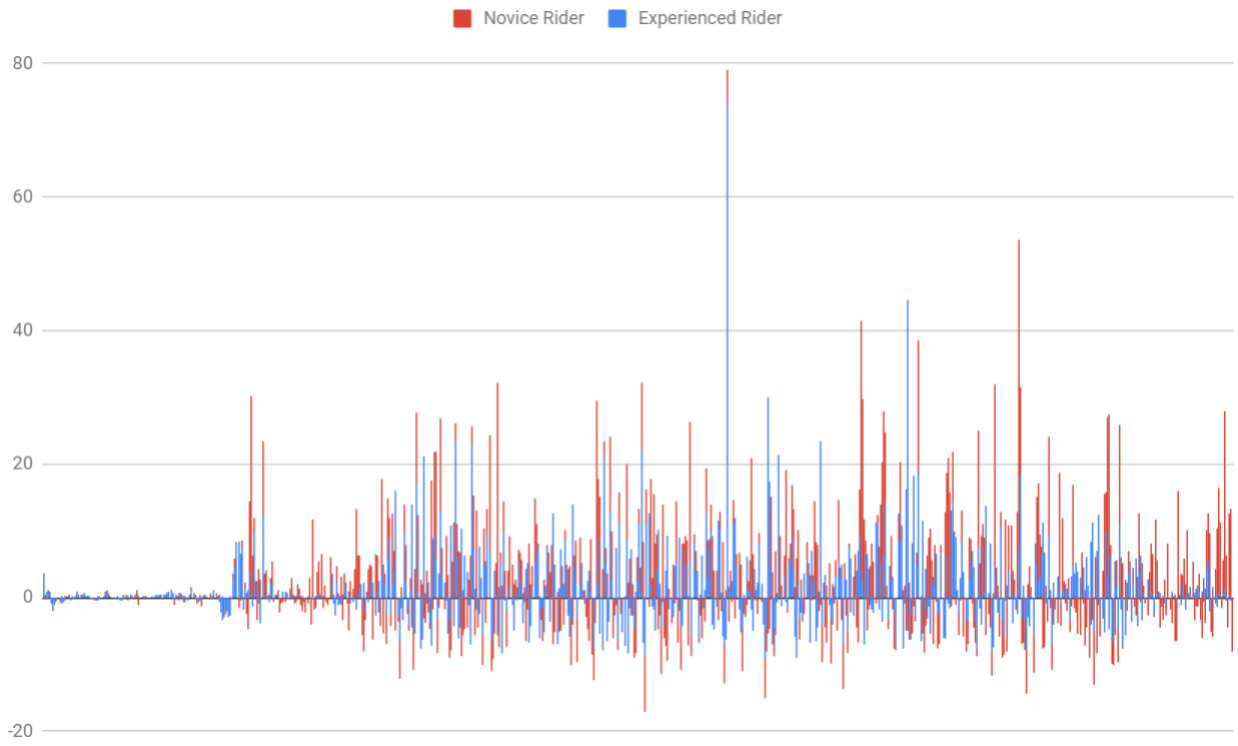

Figure 2.1: G-Force experienced by two different riders on the same trail

2.5 Rule Based Solutions

An extension of threshold based solutions are rule based solutions, a popular method used in fall detection (8) . Rule based solutions utilize a rulebase as a means of determining if an event has occurred. Analysing raw data allows for feature extraction to which a set of rules can be derived. E. g., if accelerometer reading is greater than threshold AND gyroscope reading is greater than threshold THEN a crash has occurred, a similar premise to the rules used in a mamdani fuzzy inference system.

Rule based solutions allow for more in depth understanding of the event that occured. False positives can be greatly reduced by constructing rules targeted to fire iff the sensor reading align with the signature off a fall, ruling out false positives which would be triggered by ADL similar in nature to a fall. These systems work quite well for medical applications, threshold driven rules are actually quite suitable for slow moving humains moving around a home, very few daily tasks would surpass thresholds set for falling over. The forces experienced during ADL have been shown to be far lower than what is experienced during a fall for instance(9).

A review of fall detection methodologies found that the most common implementation consisted of systems using a combination of threshold and rule based methods combined, 42 out of 51 (82%) of the existing solutions were rule based using thresholds (8).

2.6 Machine Learning Techniques

Machine learning techniques are commonly used in the development stages of a system. Using real world test data of known falls or crashes to train the model, the optimal threshold can be computed. For medical applications this is common practice as there isn't a huge variance in fall patterns when one is in the home. Learning is rarely used in real-time as the low cost, basic hardware isn't capable of running complex algorithms in real-time. Experimental studies have shown the benefits of utilizing a combination of live sensor readings in conjunction with learning techniques, for example 96% accuracy has been achieved using k-NN (10) . While Learning can improve accuracy, its complexity makes a simpler solution such as a rule-threshold based solution, a more appealing solution for low cost implementations.

For more advanced systems the most popular machine learning techniques are Support Vector Machines (SVM), Naïve Bayesian classifiers , k-Nearest Neighbors (k-NN) and binary decision trees. Choosing one method over another is not trivial as results vary to which performs the most accurately for fall detection. Findings from existing studies show no clear winner as the test methodologies differ from study to study producing different results in accuracy. As no one sensor make or model is used accuracy can also be dependant on sensor quality.

3 System Design

3.1 Requirements

3.1.1 Functional Requirements

- An accurate method of detecting incidents which may occur while cycling.
- A more personalized detector, allowing better detection for riders of all abilities.
- As lightweight and efficient application as possible to enable compatibility with as many devices as possible, as well as reducing battery usage.
- Automatic SMS functionality to allow one to be located in the event of an accident.
- Locate the rider accurately in the event of an accident as quickly as possible.

3.1.2 Non-Functional Requirements

- Useful statistics computed and displayed for the user (Average Speed, Distance Travelled)
- Google Map integration providing an accident heatmap to warn of potential dangers as well as plotting journeys.
- A compass to help with navigation if both gps and internet connectivity are unavailable.

3.2 Interface Design

As simplicity and usability were key in the development of this application, the design of RideSafe was heavily influenced by Google's standard layout recommendations. For RideSafe functional requirements had a higher precedence than a fancy looking user interface. Inspiration was drawn from popular apps utility applications. As shown in figure 3.1 the main function of the application is the first screen the user will land on, removing extraneous steps required for the user to reach their goal - in this case change the channel. A simple uncluttered layout with only the necessary buttons required on the screen.

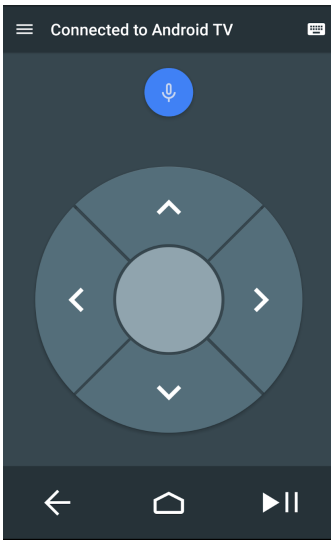

Figure 3.1: Simple Home Screen Layout

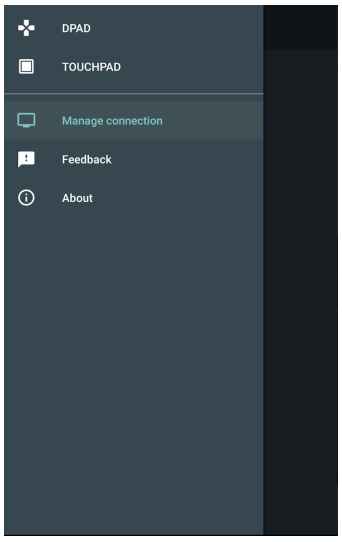

Figure 3.2: An Example implementation of A Navigation Drawer

For features less often used, Google's navigation drawer AKA the hamburger menu is a popular choice. As shown in figure 3.2 they are simple to use and allow for easy navigation to secondary functions of the application. Implementing a navigation drawer minimizes wasted space which would be taken up by adding a button for every secondary feature on the home screen, allowing the most important functionality to use up more screen real estate. As the hamburger menu is so common in every category of application on the play store, there is a high probability users will have encountered it before - resulting in a smaller learning curve to navigate through the application.

Colour and contrast of screen elements were a simple choice, having experience using my phone trailside in all conditions ranging from sweltering hot days to snowstorms.

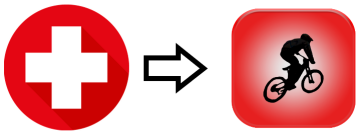

Figure 3.3: App icon Inspiration

Simplicity is of utmost importance, glare on
the screen makes any app impossible to see,
especially apps featuring a dark theme. My
experience led me to favour a bright white theme with dark icons for maximum contrast. Accent colors were also an easy choice, red was chosen as it is the color associated with "emergencies" and "hospitals" figure 3.3. Complementary shades of red were chosen using a material design palette generator (11).

3.3 System Architecture Design

This section will discuss in detail the method used to implement the crash detection logic used in RideSafe

3.3.1 System Type

Examining the vast amount of research done on fall detection in the medical domain, implementations which are threshold based could immediately be ruled out, as discussed in (section 2) threshold based solutions pose multiple issues in multiple areas, leading to the author not considering using this method.

A rule based implementation, with their proven accuracy and reliability was ultimately chosen to be the underlying architecture of ridesafe. Implementing a rule based system allows for more context to be considered measuring different data points at separate time frames, allowing the decision process to be based off a time interval rather than one moment in time. As will be discussed in the following subsection the author's choice of data points require a short period of time to determine if a crash has occurred rather than a single moment in time leading to a rule based solution to be the optimal solution.

3.3.2 Sensors & Measurements

Rather than just using the same data points used in existing solutions, the author launched an investigation into the suitability of different data points to determine if an accident has occurred. RideSafe Data Collection as it is now called is a logging application which was developed by the author to record multiple data points in multiple formats. While using the data collection application mountain biking over a period of a month at local trail centres. With a dataset containing tens of thousands lines of data points collected the data was visualized and analized. For RideSafe my findings of the most relevant data points and sensors to utilize were chosen.

- Triaxial Accelerometer

Like virtually every crash/fall detection solution, the triaxial accelerometer measuring linear acceleration of a body in three different axis as shown in figure 3.4 in m/s^2. From my research the accelerometer was found to be a key sensor in determining if a crash has occurred. Using the well known Pythagorean theorem:

$$= \sqrt{x^2 + y^2 + z^2}$$

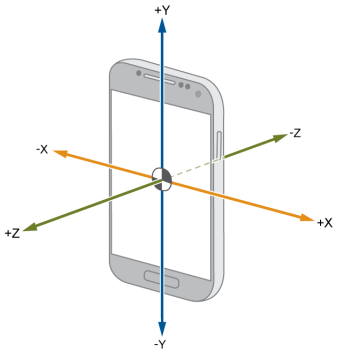

Figure 3.4: Different Axis of an Accelerometer (Source Mathworks)

and accounting for the force of gravity on earth by subtracting 9.8 m/s^2, directionless g-force can be calculated. As the phones intended orientation is purposely unspecified directionless gforce is used to not require a certain axis to require a certain orientation.

- Triaxial Gyroscope

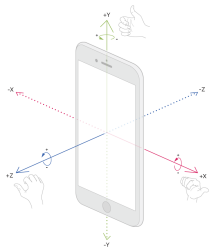

Figure 3.5: Different Axis of a Gyroscope (Source Mathworks)

The use of android's built in gyroscope allows for measuring rotational acceleration in three different axes as shown in figure 3.5. To measure rotational force in a certain direction the phone's orientation must be known, in this case where the orientation is unspecified total change in rotational acceleration was chosen, measuring the change in total rotation rather than on a per axis basis. The difference of the combined change in rotational acceleration is what the author found to be the most suitable reading.

- Speed - GPS

From the authors investigation the change of speed was of great importance as a datapoint to consider. Many existing solutions for cycling re-use the logic found in fall detection for the elderly, a system designed to work indoor where speed isn't much of a factor. The author found that change of speed was directly related to a crash occurring as well as other activities.

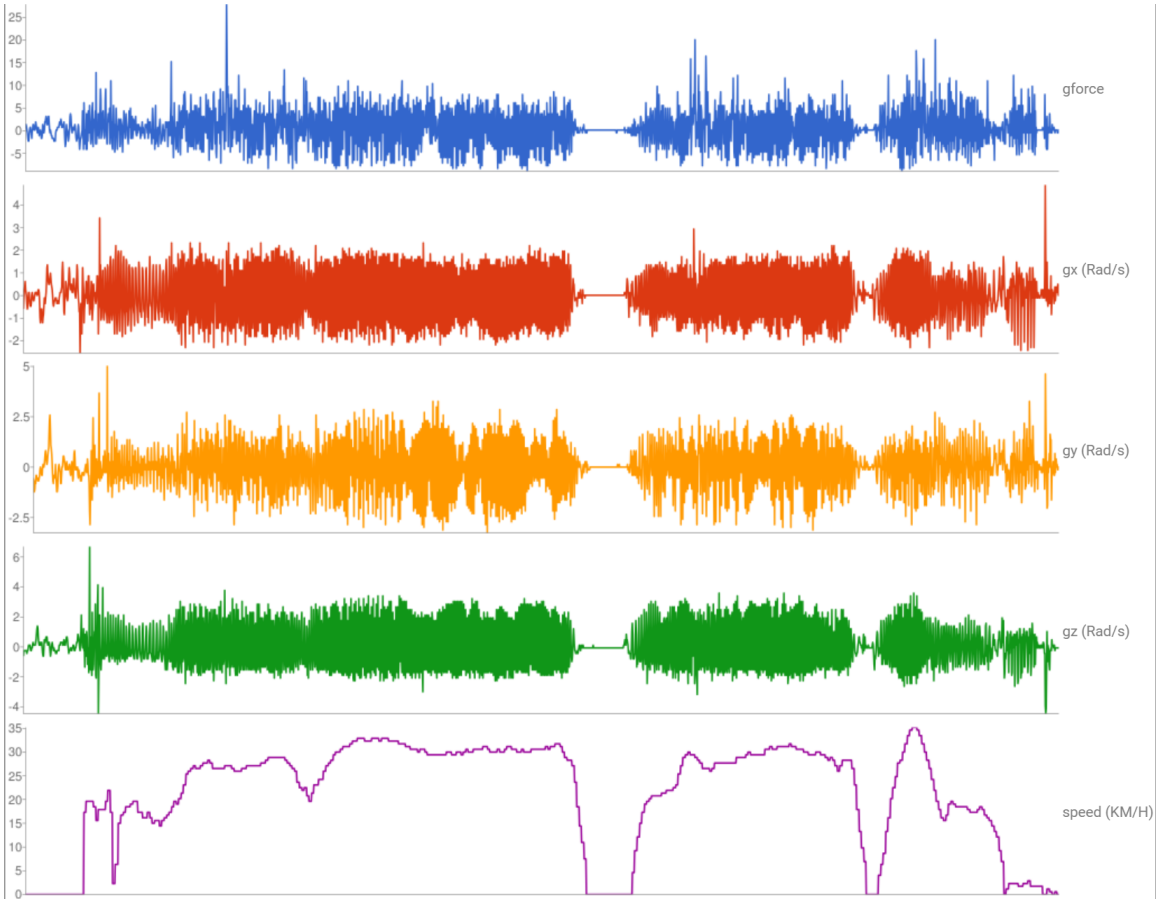

Figure 3.6: G-Force, Angular Velocity and Speed over time

Figure 3.6 shows data collected from RideSafe data collection during a cycle in the Dublin mountains. The data represents a normal mountain bike spin on metro 1, spikes of g-force can be observed, as a result of landing jumps or simply a rough section of the trail. The large decreases in speed throughout the data are a result of extremely tight corners and or quick rests in between.

3.4 Machine Learning

With this design we have multiple continuous variables to consider to determine if a crash has occurred. Given the multiple current data values a probability needs to be computed. Using supervised machine learning outcome predictions can be made for given values once the model is trained with sufficient examples of both positive and negative examples. Researching commonly used machine learning techniques used in fall detection (8), the author considered two possible methods.

- Support vector machines (SVM)
- Logistic Regression(LR)

Both SVM and (LR) are algorithms commonly used for classification, both performing comparably in practice. The main goal of both (LR) & (SVM) are to calculate the best fitting line of divide in a given dataset, allowing for predictions to be made using similar data. SVM's maximize the "line" of fit using support vectors maximizing the margin producing a class divide, outputting a binary value 1 or 0 depending on class membership. LR using a sigmoid function outputs probabilities of a value belonging to class 1 between 0 and 1. LR allows for a manual class divide line to be set enabling a desired probability to set for the class divide. Logistic Regression's accuracy has been previously evaluated with both balanced and imbalanced data leading to accuracy exceeding 90% in both cases (12). Having previously studied Logistic regression paired with promising accuracy with test values, the author decided on basing the implementation on Logistic Regression due to the flexibility of the algorithm as well its simplicity to implement in java.

4 Implementation

This chapter is split into three main sections:

- Technologies Used

Both the software and hardware used in the development for both RideSafe & the companion data collection variant.

- Main Activities

The main activity implementations will be discussed in detail along with their underlying purpose.

- Training

The methods used to train RideSafe's Multivariate logistic regression model will be explained in terms of how and where the data was recorded.

4.1 Technologies Used

This section will discuss the technologies used in the development of RideSafe.

4.1.1 Android

Android was chosen as the target platform for numerous reasons:

- previous Experience

The author currently owning multiple
android devices as well as having previous experience developing small scale android apps, Android was the clear target for development. Usually deemed as less intuitive to develop the author was unfazed due to previous experience.

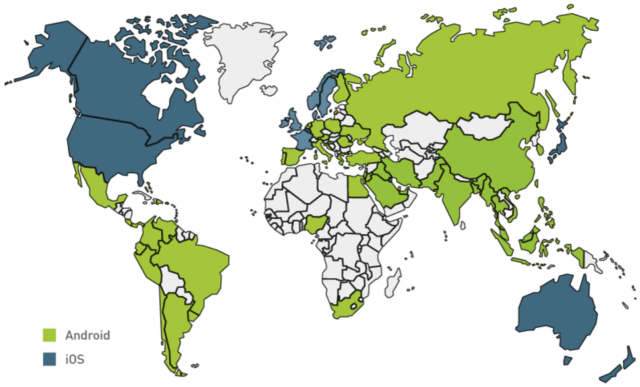

Figure 4.1: Market share (source: App Annie)

- Market Share/Target Audience

Currently there are more active android devices than there are ios devices globally. Knowing real world tests would be required from willing participants, the likelihood of them owning an android device would be marginally higher.

- Access to hardware

The most important factor, Android allows for much better access to the hardware of the device it is installed on, crucial for an application relying on accurate and reliable sensor measurements. Being a less restrictive unix based operating system features such as file management are also possible. The offloading of sensor readings is a simple process through any pc or another android device.

4.1.2 Android Studio - Java

Android Studio, being the official IDE for developing android applications and being provided by Google, it was the clear choice. As well as including Github integration for easy version control, Android studio includes useful debugging features such as logcat to view exactly what line of code in which file caused an error. The built in AVD manager allows for testing of code in an emulated android environment, this proved useful for ui changes but does not include sensors or gps leading me to do most of my testing on my own device. Android Studio's profiler proved useful to help me stay on track in terms of power consumption. Resources such as memory and cpu usage can be monitored on a per activity basis, which ultimately led to some code restructuring to improve device responsiveness in some situations. Android runs java natively, allowing some aspects such as the Logistic Regression class to be developed, tested and optimized on a desktop computer, which with little effort

can be copied into the application where it will perform as expected. Investigating the ideal learning rate and number of iterations were also conducted on a computer, which saved a lot of time compiling the application and installing it every time a change was made.

4.1.3 SQLite

Databases are crucial to the functionally of RideSafe, using SQLite databases in android allow for secure reliable data Storage. Separate tables contain data such as training data and profile information, crucial datasets for the apps operation. The usage of cursors allow for easy navigation and data retrieval. The use of a databaseHelper class allowed for the use of simple java functions to perform complex SQL commands, which removed the need to write raw SQL commands more than once.

4.1.4 Github

GitHub

Github was used for version control for both RideSafe and the data collection app. Github served as a secure backup of work and allowed the author to switch between desktops and laptops in seconds, continuing from exactly where they left off. Branching allowed for major changes with the option to revert back to a working state easily if necessary. Both RideSafe and the data collection version both share a large portion of code, copying the repo and uploading it under a different name when development required allowed for both apps to be developed essentially in parallel and when the time came both apps could be developed separately.

4.1.5 Python

The source code for RideSafe does not contain a single line of python, however the functions provided in google sheets were limiting what operations that could be done to sort the raw sensor data that was collected. Using python thousands of lines of raw sensor data could be processed and categorized in an instant, ultimately allowing the charastics of crashes to be learnt by analysing sorted data. File combinations also were handled in python, depending on use RideSafe Data Collection App would export multiple files of different runs or one single file with hours of non related data also included. The use of python allowed for hours of manual sorting to be done automatically in seconds.

4.2 Main Activities

This section will discuss in detail the main activities and classes used in the main operation of RideSafe

4.2.1 Splash Screen & Logistic Regression

To the
user the splash screen looks to be simply a loading image as seen in Figure 4.2 , but this is not the case. In the background some operations crucial to operation are being carried out. Firstly RideSafe checks if the application has previously been set up by means of a SQLite query, by querying the number of entries of the user profile table, the requirement for setup being required can be determined. If setup is required RideSafe directs the user through the setup process.

Figure 4.2: A Screenshot of the Slash Screen

Listing 4.1: Querying database entries

```
SQLiteDatabase
 db = mDatabaseHelper.getWritableDatabase();
int tableRows = (int) DatabaseUtils
.queryNumEntries(db, "profile_info");
if (tableRows < 1)
```

For performance
and stability reasons two major tasks are also run:

- Loading and updating training data.

In the case of either the app is run for the first time or
an updated version of the training data is pushed, the training
data will be reloaded. The training data supplied with ridesafe is stored as an asset as a text file, the text file, shown in figure 4.3, contains the known sensor values of both crashes and normal riding. The final column is the label for the data, used to indicate whether the data corresponds to a crash (1) or a non crash (0). Loading data from a file is expensive in terms

of time, so this process is only performed when absolutely necessary. The training data is read in line by line and stored in a SQLite database table. The data provides secure storage and quick access to the data when needed.

```
30.5413934609707 16.61 1
30.1479080838027 18.63 1
30.1379763920411 18.15 1
-1.791605051376399 0.15 0
-0.215585800588665 2.43 0
-0.513353745576568 1.01 0
```

Figure 4.3: An Example Of RideSafe Training Data

Listing 4.2: Loading Training Data from a file and storing in a database table

```
try (BufferedReader reader = new BufferedReader(
        new InputStreamReader(getAssets().open("TrainingData.txt"))))
            {

        // loop untill eof
        String mLine;
        while ((mLine = reader.readLine()) != null) {
                String[] columns = mLine.split("\\s+");
                double[] data = new double[columns.length];
                for (int i = 0; i < columns.length; i++) {
                data[i] = Double.parseDouble(columns[i]);
                }

                ContentValues values = new ContentValues();
                values.put("gforce", data[0]);
                values.put("rotation", data[1]);
                values.put("label",data[2]);
                mDatabaseHelper.addRow(values, "TrainingData");
```

- Training the logistic regression model.

Once the data has been loaded for the first time or updated the logistic model needs to be re trained. Testing had shown a very small learning rate of 0.0005 combined with 500 iterations calculated accurate weights for each variable in a training data set containing circa 300 entries. Training data used for Ridesafe will be discussed in section 4.3. A combination of a small learning rate paired with a large number of weight estimate iterations ensures accurate weights can be calculated. In this implementation the weights are along a u-shaped curve,

using the learning rate as essentially a step size increases the accuracy of weight estimation. The learning rate can be observed as linearly dependant in respect to the iteration count. If a very large learning rate was used the algorithm could overstep or undershoot the most accurate weight estimation. The implications of using too few iterations would result in the system terminating prematurely, resulting in incomplete weight estimations.

Listing 4.3: Training/updating the Model

```
public void train(DatabaseHelper mDatabaseHelper, int col) {
        Cursor dataCursor = mDatabaseHelper.getData("td");
        int numRows = 0;
        while (dataCursor.moveToNext()){
            numRows++;
        }
        for (int n=0; n<numIterations; n++) {
            dataCursor.moveToFirst();
            for (int i=0; i<numRows; i++) {
                double[] x;
                x = new double[1];
                x[0] = dataCursor.getDouble(col);
                double predicted = classify(x);
                double DataLabel = dataCursor.getInt(3);
                weights[0] = weights[0] + LearningRate * (DataLabel -
                    predicted) * x[0];
                dataCursor.moveToNext();
            }
        }
        System.out.println( Arrays.toString(weights) );
        dataCursor.close();
    }
```

Figure 4.3 shows an example of RideSafes training data, the first column is g-force, followed by change in rotational velocity and finally the assigned label. For each row in the training data training table of the database, the value is classified based on the current calculated weight (var predicted). Then the weight for that sensor reading is recalculated based on what the system has already learnt, with the exception of iteration 0 where the predicted value will be 50% likely. weights[i] = weights[i] + Learning Rate * (DataLabel - predicted) * x[i]; The current weight for sensor data i = the current weight for sensor data i + the specified learning rate(0.0005) * (The known class of the variable either 1 for a crash or 0 otherwise - what it currently classifies as) * the sensor reading associated with that weight. This process is repeated until the weights are as accurate as possible and no longer change.

The final calculated Weights for both g-force & total rotational velocity change from the training data used can be seen in figure 4.4. The regression model provides an answer to the following in real-time, Given current sensor values x , what is the probability of (1|x) where 1 is the class label belonging to data pre-learnt of accidents.

```
[0.5713683138694551, -0.6388561706923281]
```

Figure 4.4: Weights calculated by the Multivariate Logistic Model

The function used for classification as shown below is an implementation of the formula for the sigmoidal Logistic curve

$$P = \frac{1}{1 + e^{-}(a + bX)}$$

For Each Row of data the probability P is calculated using the logistic curve.

Listing 4.4: Classify Function

```
public double classify(double[] x) {
        double logit = .0;
        for (int i=0; i<weights.length;i++) {
            logit += weights[i] * x[i];
        }
        return 1.0 / (1.0 + Math.exp(-logit));
    }
```

As seen in figure 4.5 green points on the graph are plotted with a probability near 0, these points represent the sensor values associated with a non crash. The red points are plotted probabilities of sensor values representing crashes. The blue point represents classified sensor values with no known truth, using the learnt weights the model calculates the probability of the likelihood the readings match those of an accident, e, g,. The sensor values related to the blue dot are more similar to what the system knows is a crash, resulting in a high probability that a crash has occurred.

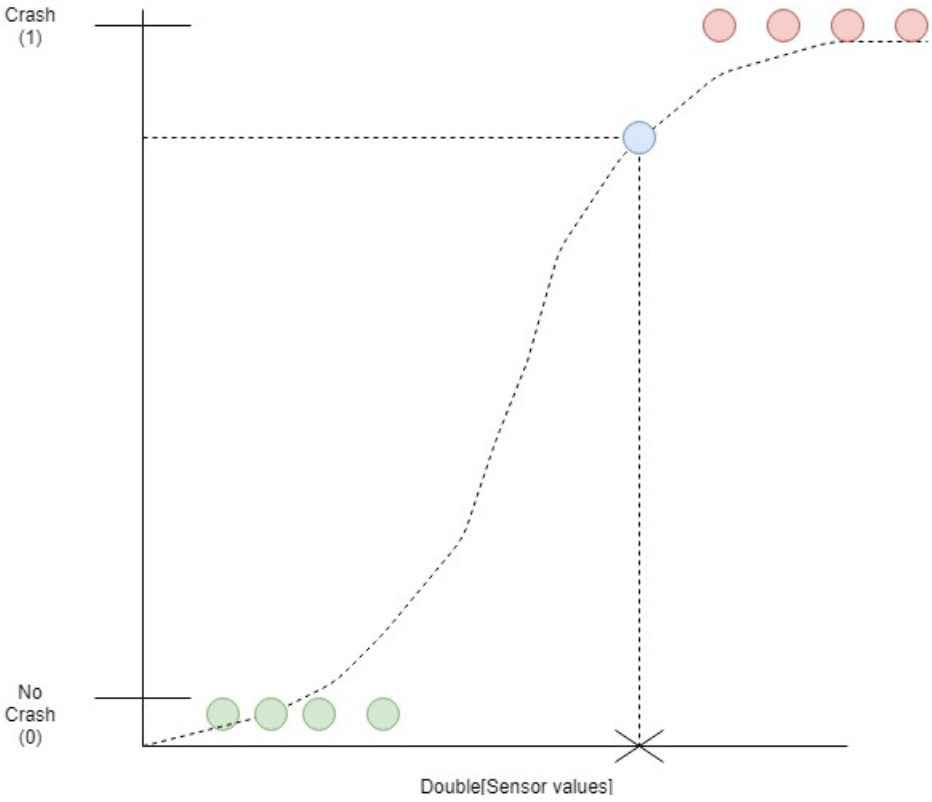

Figure 4.5: Sigmoid curve produced by a logit function

Figure 4.6 shows example test cases of both crash and non crash data being classified, Crash data is quite unique versus standard riding data leading to crashes being identified to a high degree of certainty, Normal Riding varies quite a lot leading to a range of output probabilities.

```
 Sensor Readings similar to recorded crash data
should be closer to 1  :     0.9901961080072567
should be closer to  1  :     1.0
should be closer to 1  :     0.9999999999417124
should be closer to 1  :     0.9999962211872435
Sensor Readings from Normal Riding
should be closer to 0  :     0.5128529548881319
should be closer to 0  :     0.48016984335307333
should be closer to 0  :     0.06181141774528422
should be closer to 0  :     0.2663067821954863
```

Figure 4.6: Classifying Example Test values With A Trained Model

4.2.2 Home Screen

The author kept the home screen as simple and user friendly as possible. By utilizing large black icons against a plain white background the screen remains viewable in extremely bright and dark environments. Buttons were kept large for ease of use while wearing gloves as many cyclists do. Towards the bottom of the screen are some useful analytics for the user such as their max recorded speed & distance traveled.

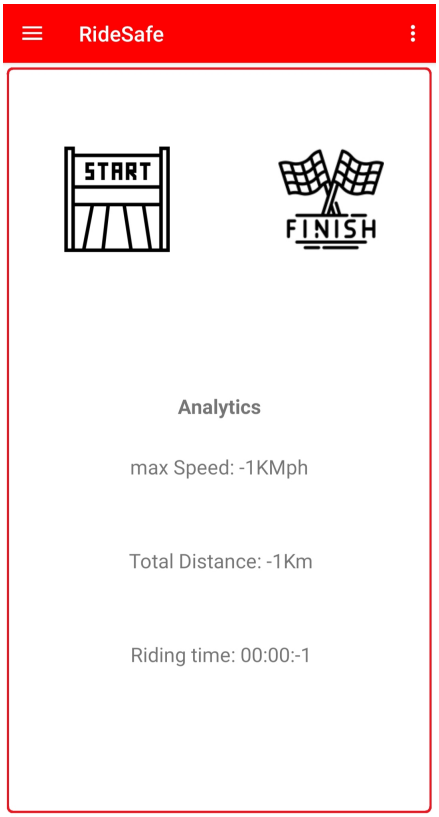

Figure 4.7: A Screenshot of the home Screen

Listing 4.5: Starting the Crash Detection Service

```
public void startService(View v) {

        locationManager = (LocationManager
) getSystemService(LOCATION_SERVICE);
        gpsManager
 = new GPSConfig(main_Home.this);
        isGPSon
 = locationManager.isProviderEnabled
(LocationManager.GPS_PROVIDER);

        if (isGPSon) {
            String init = "running";
            Intent serviceIntent
 = new Intent(this, RideSafeService.class);
            serviceIntent.putExtra("inputExtra"
                , init);
            startService(serviceIntent);
```

```
            RUNNING = true;
        } else {
            showSettingsAlert();
        }
    }
```

On Pressing the Start Button RideSafe
performs a check to determine if location services are enabled
for the device. If GPS is not enabled the user is prompted to
do so in their device settings, otherwise an intent is launched
to start the background service which will be discussed
in detail in the next section. The Stop service button is simply
the opposite of the start button bar the gps check, An intent
is launched on pressing the to end the background service.
The analytics section is also updated on pressing finish.

4.2.3 RideSafe Background Service

RideSafes background service is where both all the sensor
monitoring takes place and the rule based system is implemented. On
pressing the start button on the home screen the background service is
launched. This background service has no view or user interface layout,
the user is reassured the system is running with a persistent device
notification which can be seen in figure (ref), Google recently required
any background service to have a notification displayed to inform
the user operations are being performed which cannot be seen.

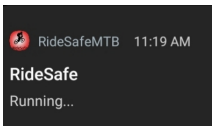

Figure 4.8: A Screenshot of the service notification

Listing 4.6: Taking Sensor Readings

```
@Override
    public void onSensorChanged(SensorEvent event) {

        sensorType = event.sensor.getType();
        if (sensorType == 1 && !GAquired) {      // accelerometer
            X = event.values[0];
            Y = event.values[1];
            Z = event.values[2];
            ACCELEROMETER = Math.sqrt(X * X + Y * Y + Z * Z) - 9.807;
            ACCELEROMETER = (double)Math.round(ACCELEROMETER * 100d)
                /100d;
```

```
            GAquired = true;

            if(RAquired) {
                GAquired = false;
                RAquired = false;
                ClassifyX();
            }
        } else if (sensorType != 1 && !RAquired) {

            GYROX = event.values[0];
            GYROY = event.values[1];
            GYROZ = event.values[2];

            GYROX = (double)Math.round(event.values[0] * 100d)/100d;
            GYROY = (double)Math.round(event.values[1] * 100d)/100d;
            GYROZ = (double)Math.round(event.values[2] * 100d)/100d;
            currentRot = GYROX + GYROY + GYROZ / 3;
            currentRot = (double)Math.round(currentRot * 100d)/100d;

            RAquired = true;
            if(GAquired){
                GAquired = false;
                RAquired = false;
                ClassifyX();
            }

        }
    }
```

Listing 4.7: Senor Setup

```
SM.registerListener(this, myAccelerometer, SensorManager.
    SENSOR_DELAY_NORMAL);
        SM.registerListener(this, myGyroscope, SensorManager.
            SENSOR_DELAY_NORMAL);
```

The recording of sensor values is performed as shown above in lstlisting Sensor Readings In the onCreate method both the accelerometer and gyroscope are initialized as well as registering device listeners. As both sensors are required to use the same onsensorchanged

method booleans are used as triggers similar to the operation of a basic state machine. When readings from both sensors have been acquired they are converted into the required format as discussed previously in the design section. The next reading required is the current speed.

Listing 4.8: Calculating Speed & Distance

```
@Override
    public void onGPSUpdate(Location location) {
        speed = location.getSpeed();
        currentSpeed = round(speed, 3, BigDecimal.ROUND_HALF_UP);
        SPEEDCURR = round((currentSpeed * 3.6), 3, BigDecimal.
            ROUND_HALF_UP);
        if((int)SPEEDCURR > MaxSpeed){
            MaxSpeed =  (int)SPEEDCURR;
        }
        if(prev.getLatitude() != -9999 && prev.getLongitude() !=
            -9999){
            current.setLatitude(location.getLatitude());
            current.setLongitude(location.getLongitude());
            distance = (int) current.distanceTo(prev);
            distance = distance / 1000;
            TotalDistance = TotalDistance + distance;
            ContentValues values = new ContentValues();
            values.put("lat", current.getLatitude());
            values.put("long", current.getLongitude());
            values.put("speed", SPEEDCURR);
            mDatabaseHelper.addRow(values, "locationData");
            prev.setLatitude(current.getLatitude());
            prev.setLongitude(current.getLongitude());
        }
        else{
            prev.setLatitude(location.getLatitude());
            prev.setLongitude(location.getLongitude());
        }
    }
```

Speed is calculated using android's location services, utilizing location services speed is calculated by means of distance over time, where distance is the straight line distance

between two locations, each location is in the form of a latitude-longitude pair and factoring in the curvature of the earth a suitably accurate speed and distance can be calculated. As RideSafe is intended to be used in areas which may not have the best signal, the location manager will fall back to cell tower triangulation to compute a location if no GPS signal is available. SPEEDCURR is speed converted to kilometer per hour and rounded to three decimal places as a huge degree of accuracy is not required. Latitudes and longitudes as well as the speed to which was being travelled is also computed and recorded for the purpose of journey tracking which is an option available in the maps section of the application. Both the polling for the accelerometer and gyroscope is far quicker than calculating speed, speed is on average three times slower to calculate than g-force and rotational velocity are, leading to 3 calculations of each sensor corresponding to a single speed.

Listing 4.9: RideSafe's Rule Based System

```
public void ClassifyX() {

        if (ACCELEROMETER != nullNum && GYROX != nullNum && GYROY !=
            nullNum && GYROZ != nullNum && SPEEDCURR != nullNum) {

             RotChange = currentRot + totalRotPrev;
             RotChange = (double)Math.round(RotChange * 100d)/100d;
             double[] currentGFORCEValue = {ACCELEROMETER};
             double[] currentROTValue = {RotChange};

if(currentGFORCEValue != null && currentROTValue != null) {

    if (!isMoving()) {

        if (LOGISTICG.classify(currentGFORCEValue) < Threshold  &&
            LOGISTICR.classify(currentROTValue) < Threshold)
             Log.d("Answer from service", "No Crash Detected");

        else {
             Log.d("Answer from service", "Crash Detected " );
             monitorSpeed();


        }
    }
```

}

Once all three parameters are obtained RideSafes system rule is queried to see if an accident may have occurred. The code above implements the following rule: IF the rider is moving THEN Classify current sensor readings, IF the probability computed based on current readings is less than A threshold THEN a crash has not occured. IF the probability exceeds the the threshold THEN monitor speed. All sensor values are classified in real time with a pre trained model. The current speed is not classified in the logistic model as a given speed is valid for both crashes and non crashes, from my testing including speed as a variable in the model produces a calculated weight close to 0.5 as it can be associated with both classes ,leading to its removal. If the threshold is exceeded MoniorSpeed() is called. In this case the rider has experienced a large impact and or excessive rotation. monitorSpeed() Starts a timer rechecking the speed of the rider. If the riders speed has dropped significantly after a large impact from my research one can come to the conclusion an accident has occurred. If the timer elapses the crash handler which will be discussed in the next section will be launched.

When the service is stopped from the home screen parameters such as riding time and max speed are updated using shared preferences. Sensors are unregistered and the background service is destroyed.

4.2.4 Crash Handler

figure 4.9 shows the layout of the crash handling screen, The crash handler is launched by means of an intent from the background service previously discussed as shown below. The location and speed of where the crash occurred is passed via a bundle for usage in both the accident heatmap and for content in the automatic SMS which is sent.

Listing 4.10: Launching the Crash Handler

```
void LaunchCrashHandler(){
        Intent intent =
new Intent(RideSafeService.this, Crash_Handler.
class).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        Bundle b = new Bundle();
```

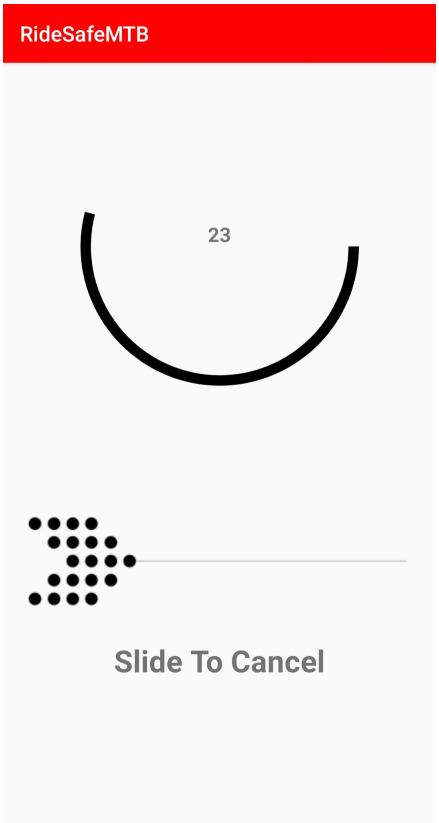

Figure 4.9: A Screenshot of the Crash Handler

```
        b.putDouble("latitude", current.
            getLatitude());
        b.putDouble("longitude", current.
            getLongitude());
        b.putDouble("speed", SPEEDCURR);
        intent.putExtras(b);
        startActivity(intent);

    }
```

As the crash handler is launched while the users phone is out of hand and the device locked, extra flags were set for the activity so that it can be launched and displayed from a locked device. By passing the users lockscreen the crash handler is displayed an an audible alarm is played on repeat.Password,pin entry or fingerprint scanning is all bypassed when the activity is launched. The alarm is for the scenario where a false positive my occur to alert the rider to cancel the timer before the sms is sent. The flags used were as follows:

Listing 4.11: Flags to allow device to RideSafe to display the crash handler

```
 this.getWindow().setFlags
(WindowManager.LayoutParams.FLAG_FULLSCREEN |
                                WindowManager
.LayoutParams.FLAG_DISMISS_KEYGUARD |
                                WindowManager
.LayoutParams.FLAG_SHOW_WHEN_LOCKED |
                                WindowManager
.LayoutParams.FLAG_TURN_SCREEN_ON,
                    WindowManager
.LayoutParams.FLAG_FULLSCREEN |
                                WindowManager
.LayoutParams.FLAG_DISMISS_KEYGUARD |
                                WindowManager
.LayoutParams.FLAG_SHOW_WHEN_LOCKED |
                                WindowManager
```

```
. LayoutParams.FLAG_TURN_SCREEN_ON);
```

While the alarm is played, a countdown timer is displayed
on screen along with a slide bar used to cancel the countdown.
In the event of a false positive or an accident where the rider
does not require assistance, sliding the cancel slider will cancel the timer and relaunch the background service and re locking the device. A slider is used as it is much more unlikely to accidentally slide to cancel compared to just pressing a button.

Once the timer expires the users chosen emergency contact is automatically sent an SMS e, g,. I have had an accident. my location is :https://maps.google.com/?q=" + Latitude + "," + Longitude "I Crashed while traveling at : " + Speed + "kmph Sent Automatically by RideSafe". The users speed and location which were passed from the service intent are used to automatically generate a link to their location on google maps, allowing the emergency contact to relay the information to the authorities, or simply by clicking the link receive directions to accident site themselves.

Listing 4.12: Automatic SMS Sending

```
private void sendSms() {
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage(PhoneNumber, null, message, null,
            null);
        Toast.makeText(getApplicationContext(), "SMS␣sent␣to␣" +
            ContactName,
                Toast.LENGTH_LONG).show();


    }
```

4.2.5 Settings

The settings screen as seen in Figure 4.10, provides a few useful functions to the end user:

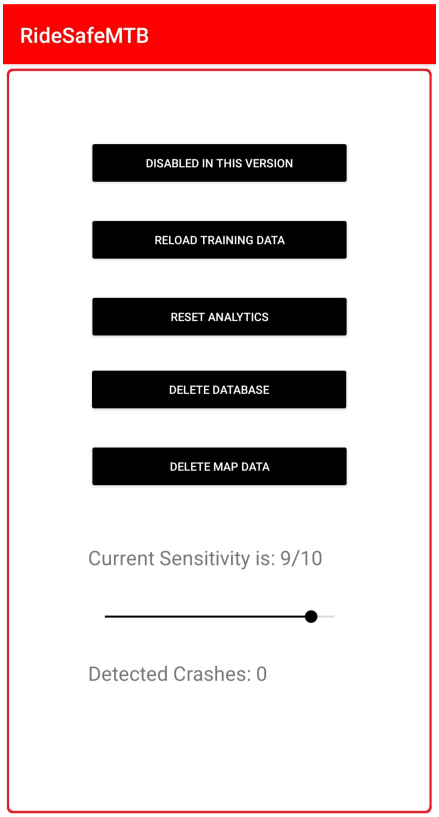

Figure 4.10: A Screenshot of the Settings Screen

As this screenshot is from the version currently listed on google play the function "export data to the developer" has been disabled in the store version. The button was originally implemented for allowing quick transfering of sensor data. Upon pressing the button the full database of RideSafe was saved as a database file which was sharable in the android operating system. Data could easily be transferred via email for debugging purposes. Delete Database is closely related and was used in conjunction with the export button to allow small samples of data to be recorded and then deleted, allowing for small separate files rather than a single large file to be exported.

Reload Training data is a manual override to reload and re train the system, When crashes occur the values associated are added to the systems database, at a convenient time for the user the values are added to the classifier and the system can be retrained.

Reset Analytics and Delete map data are both implemented to clear user specific data such as previous journey latitudes and longitudes used to plot journeys on the map or the max speed displayed on the home screen.

As mentioned in the discussion of the background service, there exists a threshold value for the probability required to trigger a detected crash, the settings screen includes a sensitivity slider to adjust the threshold value used. Each notch on the slider increments or decrements this value by .05, allowing users slight adjustment to improve their experience while using the supplied training data.

The detected crash counter, used for testing purposes is a counter of how many times the application has detected an accident.

4.3 Non-Functional Requirements Implemented

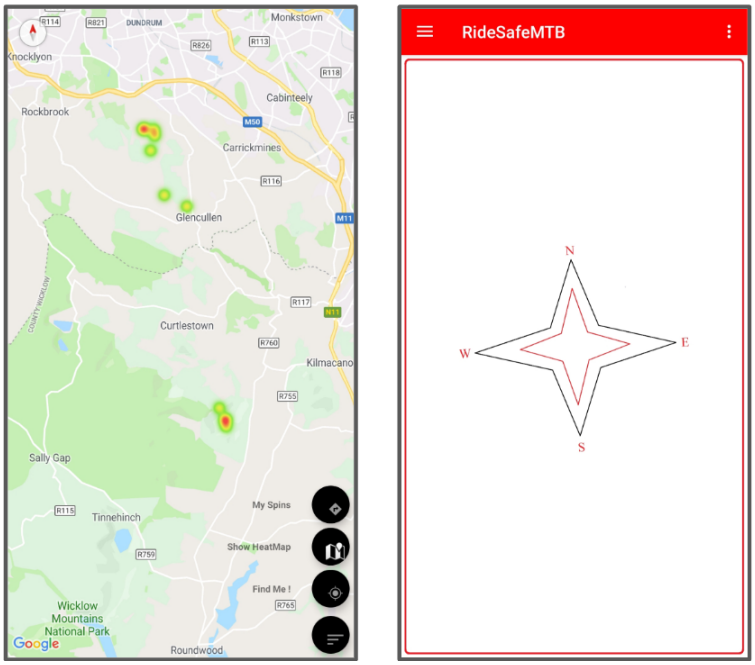

Figure 4.11: Accident HeatMap (left), Compass (Right)

4.3.1 Google Maps

Google's map API was taken utilized to implement two additional features:

- Accident heatmap

Users can currently see potential danger zones where previous accidents have occured. Once a crash has been detected the users latitude and longitude of where the crash has occurred is plotted on RideSafes built in map. As the size and colour of the spot on the map increases the likelihood of present danger increases. By viewing the map users could decide to take extra caution while passing through the area, or avoid it entirely, potentially preventing an accident occurring. Currently the heatmap is user specific, future plans for the heatmap will be discussed in future work.

- Ride Tracking

A Simple way to view trails and journeys ridden by the user. By storing the latitudes and longitudes where the rider has travelled in a database table, these points can be plotted on

the map allowing the user to view places they have been and passed by. The speed of which the rider was travelling at the certain locations was also recorded with the intention to change the colour of the plotted journey according to the recorded speed however, due to time constraints dynamic line colouring has yet to be implemented.

4.3.2 Compass

With Certain trails being situated in very remote areas, navigation can sometimes be an issue, especially when a connection to the internet cannot be obtained. Using the orientation sensor the direction of magnetic north can be calculated. By simply rotating an image with offsets calculated to point north a very simple yet reliable compass has been implemented. A small but useful feature.

4.4 Training

The Logistic model utilized in RideSafe has been trained by collecting training data in two seperate ways:

- Logging of ride Data

In the early stages of development data was collected solely by the author while mountain biking at local trail centres. Similar to medical solutions the "ADL" of mountain biking had to be identified and the patterns learnt. Globally trails are allocated a rating to how technical and difficult they are, ranging from a "green" trail to "double black diamond". Thousands of points of data were collected from "blue" "red" & "black" trails in the Dublin/Wicklow area. This data proved invaluable in terms of learning what are considered "normal" sensor readings while mountain biking. Upon receiving ethical approval data has been collected from numerous riders of varying skill level and experience.

- Crashing in a controlled environment

For the model not to be biased an equal number of instances of both crash data and non crash data must be present in the training data, With few "real World" major accidents occurring during the data collection phase action was taken to ensure the system could be trained correctly. Alike the training performed in medical applications where controlled falls are purposely done onto a soft surface such as a mattress, the author conducted similar tests at the local indoor skate park. Three hours of intentional crashes were performed in one of two main ways: The most common cause of severe mountain biking injury - over the handlebars as well as side impact which is commonly caused by loss of traction.

Figure 4.12 shows stills of a video taken with accompanying sensor readings closely aligned. The sample rate used was 3 readings of each sensor per second bar speed which is slower to

calculate (roughly once per second). The data readings are labelled and are displayed over time. The first section closely resembles data collected from normal riding on a green trail, slow speed, low g-force and marginal rotation of the device. Section 2 closely resembles data collected from riding blue/red trails, spikes in both g-force and rotation of the device, more movement in terms of rotational velocity on all axis as well as spikes in g-force. In this instance the spikes are caused by the small direction change associated riding the takeoff ramp. Section 3 & 4 are of most interest, the flatline from the sensor values are caused by gliding through the air, completely off the ground. Section 4 is both impact and aftermath. Big spikes of both g-force and rotational velocity can be observed as impact occurs. As clear from the graph the change in speed is a huge anomaly compared to standard riding. The results of this testing allowed for the development of RideSafe's rule base.

After countless intentional falls enough data was collected to both identify the characteristics and pattern of the events occuring during an accident.

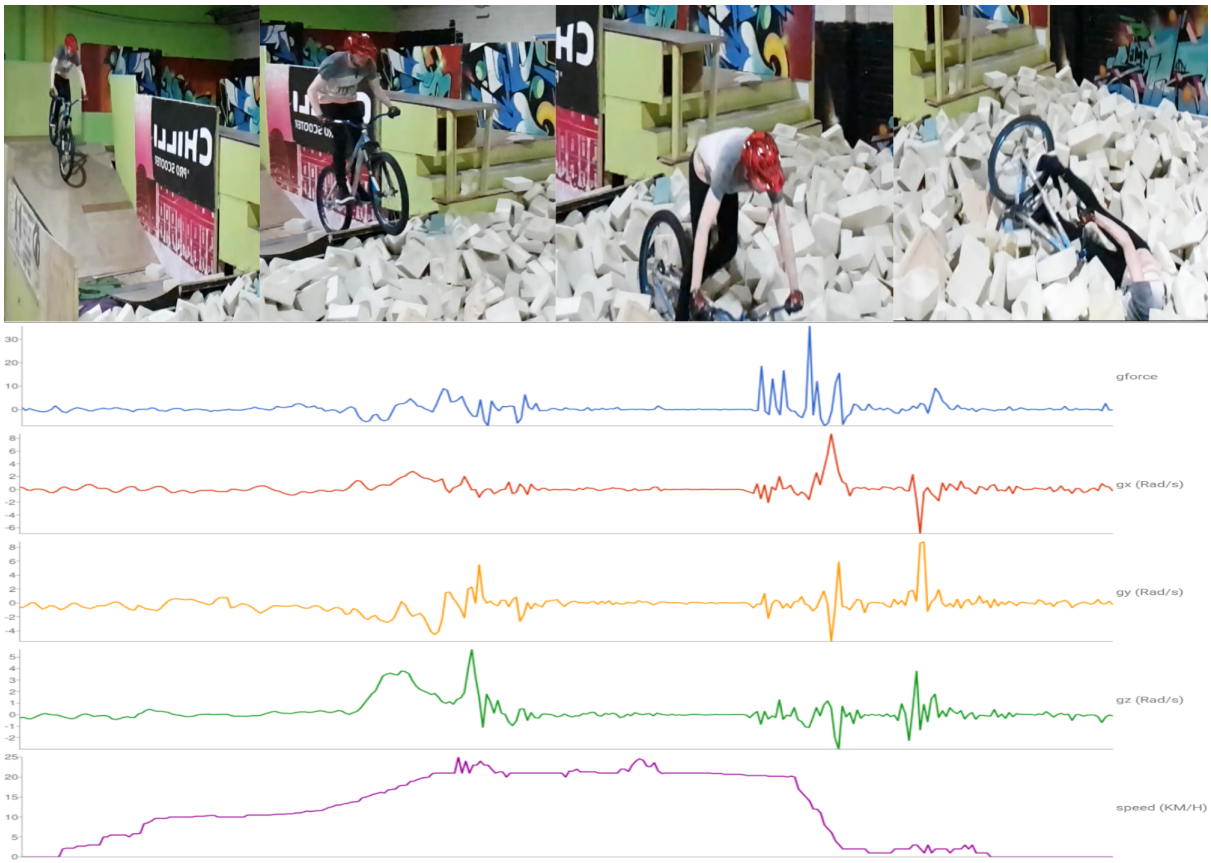

Figure 4.12: Analysis of A Crash

4.5 Application Flow Diagram

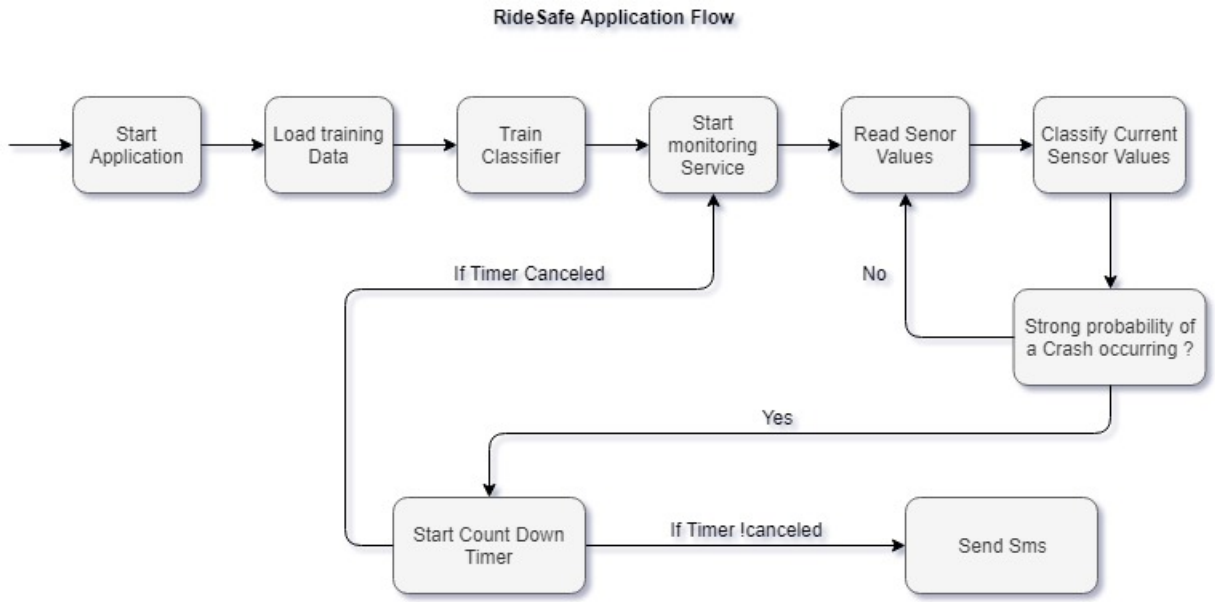

Figure 4.13: RideSafe Application Flow

5 Evaluation & Results

5.1 Carpark Tests

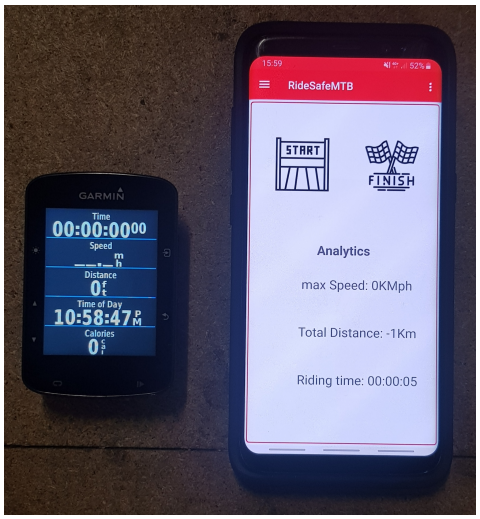

Figure 5.1: Garmin Edge 520 Vs RideSafe (Samsung Galaxy s8)

From analyzing user reports of existing solutions three common issues arose, issues impacting the user experience enough for users to abandon using the systems altogether. RideSafe installed on a Samsung Galaxy s8 was tested side by side with Garmin Edge 520. Each test was carried out 10 times.

- Drop Test

The drop test consisted of dropping both devices side by side from hip height, onto a wooden floor.

- Hard Braking

The hard braking test was performed with each device in a trouser pocket, pedaling up to an average speed of 25km/h and coming to a complete stop as quickly as possible.

- Shake Test

The shake test was conducted by putting both devices in hand and shaking vigorously for a

Table 5.1: Carpark Tests

Test Type	RideSafe	Garmin Edge 520
Drop Test	0	6
Hard Braking	1	4
Shake Test	0	10

random amount of time each run.

The results of these "carpark Tests" are shown in table 5.1 . The number present in each cell is the number of times the system failed and detected a crash.

5.2 Real World Testing

The main real world tests of the system were conducted on two seperate days,in two seperate locations. Ridesafe was evaluated by recording the number of accidents reported by the application vs the true number of accidents which may have occurred. Participants with a personal relationship to the author may also be included as participants. The procedure for the testing was as follows:

- 1 Install the application.
- 2 Before commencing a run start the background service by pressing "start" on the home screen.
- 3 upon completion of the cycle the service was stopped.
- 4 The number of detected crashes displayed on the settings screen was noted.
- 5 The true number of crashes detected was recorded.

Testing Day one

The First test of RideSafe were conducted by 4 participants of varying skill including the author. Testing was conducted on "Red" "Black" and uncategorized trails located on Ticknock mountain Co. Dublin. Testing took place over a 2 hour period consisting of on average 6 runs per rider. The test results from day one are shown in Table 5.2.

Findings from testing day one provided interesting insights to the operation of the performance of RideSafe, Rider #2 briefly lost control and veered off the trail for a couple of moments, without actually falling of the bicycle. As the rider considered this a crash (albeit

Table 5.2: Testing Day 1: Results

Rider #	No of Actual Crashes	Number of Detected Crashes
1	0	0
2	1	0
3	0	0
4	0	0

very minor) the result was recorded. As RideSafe is trained to detect more severe crashes where there is a potential for injury I can conclude RideSafe correctly did not flag this as a crash. No false positives were detected in day one of testing.

Testing Day Two

For testing day two more participants were required to accurately evaluate the system. With permission to recruit participants having being granted by the owners, Testing Day 2 took place at Glencullen Adventure Park. Glencullen Adventure Park offers a more diverse selection of trails, more indicative to what is commonly ridden around ireland ranging from smooth flowy trails (left) to intense rough technical trails(Right) as shown in figure 5.2.

Figure 5.2: A Smooth trail (left) vs A steep technical trail (Right)

Being restricted to recruit participants over the age of 18, who have a compatible android phone and cycle with their phone on their person the author managed to recruit 12 willing participants in total. Participants recruited were of all skill levels and ages. The author while personally participating exported RideSafes database upon completing a full run while top to bottom, the total count of exported databases was used to calculate the average number of

Table 5.3: Testing Day 2: Per Rider Results

Rider #	No of Actual Crashes	Number of Detected Crashes	False Positives	Missed Detections
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	3	2	0	1
6	0	0	0	0
7	0	0	0	0
8	1	1	0	0
9	0	0	0	0
10	0	2	2	0
11	1	1	0	0
12	na	na	na	na

Table 5.4: Per Rider System Results

True Detection Accuracy	False Positive Percentage	Missed Detection
80%	18.1%	9%

runs completed per rider. Results have been calculated on both a per rider basis as well as per total estimated run count. Results are listed separately and clearly labeled.

Out of 12 willing participants one participant left without debriefing so as a result their results are not available(Rider #12).

- Out of 5 reported crashes occurring RideSafe correctly identified 80% of them (True Detections).

Rider #5 reported 3 accidents occurring while using the application, 2 of which were detected. Rider #8 reported 1 accident occurring which was correctly identified. Rider #11 reported 1 accident occurring which was correctly identified.

- Rider #10 reported no accidents had occurred while using RideSafe however two incorrect detections were made resulting in two False Positives.

The total number of recorded runs as completed by the author were 9 compete runs. Using this number

$$9 \pm 1$$

the total number of runs was estimated as :

$$\frac{((11 \cdot 8) + (11 \cdot 9) + (11 \cdot 10))}{3} = 99$$

With one rider experiencing two false positives the percentage can be calculated as

$$\frac{2}{99} \cdot \frac{100}{1} = 2.02\%$$

Concerning the crashes reported vs detected crashes ,the author is led to believe the test methodology is partly to blame, What is the definition of a crash? Is an injury required to constitute a crash? Leaving these questions unanswered led to participants having their own definition of what exactly is considered a crash, this is the number which was reported to the author. RideSafe being trained on more serious crashes which would cause an injury by design, should not detect very minor incidents leading the author to believe the missed detections are from very minor incidents.

In relation to the false positives detected while Rider # 10 was testing, the author believes they were caused by one of two reasons:

Unsatisfactory hardware / Gps connectivity

As RideSafe rule based system is heavily influenced by the requirement of the riders current speed, in the case of both no GPS reception and bad network signal speed will not be possible to compute. The combination of an impact paired with the loss of being able to calculate speed could trigger the crash detection algorithm.

Unsuitable Training Data

As the system was mainly trained solely by the author, it is probable the model is unsuitable for every type of rider, the model will evolve over time as crashes are recorded by individuals but all the testing was conducted using the same model. Riders riding bikes without adequate suspension dampening would also experience more forces than riders with adequate suspension.

5.3 Application Performance

The performance of RideSafe is of utmost importance. For the target audience the issue of battery life can result in the difference between life and death. With ones mobile phone being rendered utterly useless with a dead battery, the efficiency of ridesafe has been thoroughly tested.RideSafe's performance in terms of power consumption was evaluated using Android studio's built in profiling tool. Figure 5.3 highlights the performance of RideSafe during the training phase and when idle. Spikes of medium power consumption can be observed in the "splash Screen" where in this case firstly the training data is being read from a file and stored in a database, followed by the training of the logistic regression model. As this process is rarely called the impact on battery life is not concerning. As the application is left idle it uses negligible amounts of system resources (>2% CPU usage & circa 200MB of RAM).

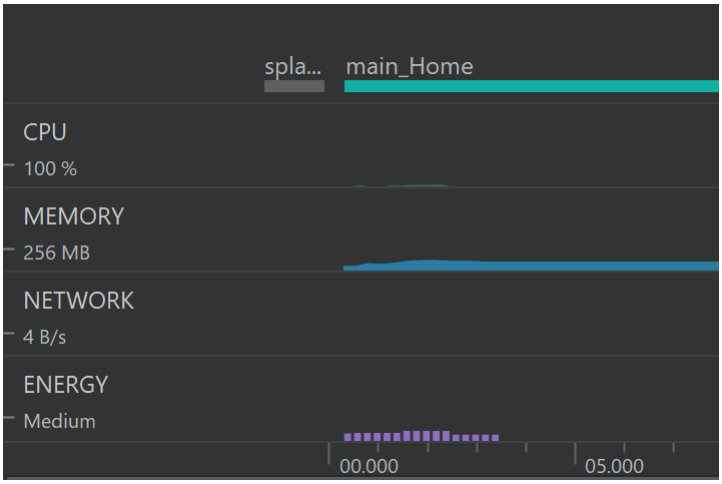

Figure 5.3: Resource utilization during Training / idle

Figure 5.4 shows the performance of RideSafe when the background service is running. CPU and Memory usage remain consistently low while the service is running. Optimisations such as keeping the contents of the service to only what is absolutely necessary for the functional requirements to be implemented. The use of a background activity itself with no visualizations on screen has been shown to conserve power itself as on of the biggest battery drains on an android device is the screen itself (13). Spikes of energy can be observed over time as the service runs due to the calculation of speed via GPS. The calculations required for GPS are unfortunately expensive in terms of battery consumption. As speed is crucial to the operation of RideSafe, its removal is currently not possible, but as will be discussed in the following chapter a possible solution does exist. Overall the power consumption of ridesafe is considered to be classified as Low-Medium.

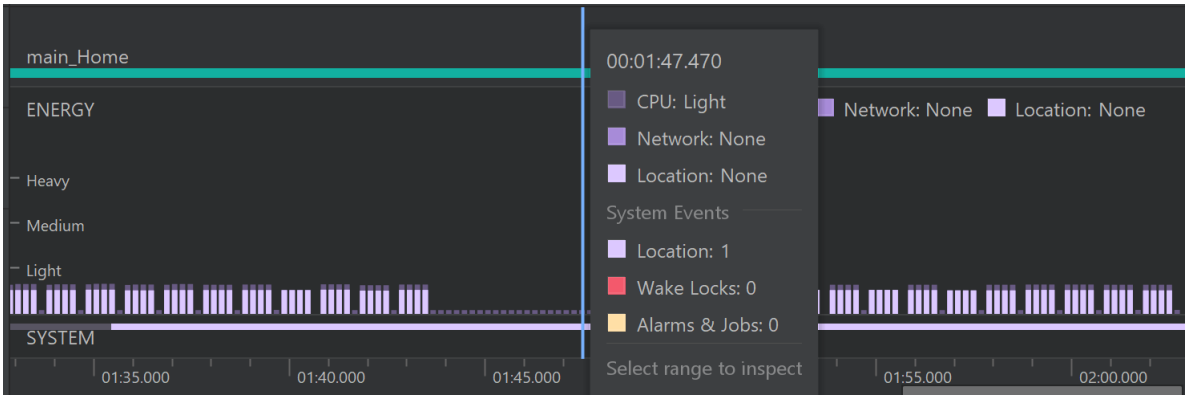

Figure 5.4: Background Service Performance

6 Conclusion

6.1 Future Work

6.1.1 Training

The author believes that the accuracy of RideSafes detection algorithm could greatly be improved with more comprehensive training data. Due to time constraints the data used for training the machine learning model is not as diverse as intended. The training data used for ride safe during tested was collected only from two riders. As discussed throughout even with the most common types of crashes displaying similar patterns, no two riders are identical. With RideSafe performing as it does one can only deduce performance would improve with a more generalized model to begin with eventually leading to a more personalized model over time.

6.1.2 Collaboration

Currently on the market popular ride tracking applications severely lack any safeguards for the users safety, popular cycling oriented fitness tracking applications such as Trailforks or Strava actively monitor the users speed and location. Without major redesigns for either party the crash detection logic present in RideSafe could be implemented as a supplementary feature for the benefit of the users. Since the most power intensive feature of RideSafe is GPS functionality, the algorithm could be added to Strava or Trailforks adding only marginally higher power consumption then they already have.

6.1.3 Inter-Device communication

RideSafe would greatly benefit with a companion server. Three major improvements could be made with server-device communication

Communal Accident HeatMap

Crash locations collected from all users around the world could be collected to populate a global heatmap accessible to all users. All accident sites from all users could produce an accurate depiction of where danger may be present, potentially saving users from riding dangerous trails or locations. The map could easily be split into an all time accident heatmap as well as showing only recent accidents which could indicate a trail which recently became hazardous, such as a fallen tree or a dislodged rock.

Push Notifications

Once a crash is detected fellow local users of RideSafe could be sent a distress notification. By locating the nearest users, a distress call could be put out. As the emergency services can take quite a while to arrive, perhaps a local first aider cold reach the scene in advance.

Shared Training Data

Crash data collected from all users automatically pushed to a server could be used to create a shared dataset of training data, benefiting all users and improving RideSafes generalized model.

Performance Improvements

As suggested to the author external speedometers for bicycles are readily available and inexpensive. As GPS is currently used to calculate speed in Ridesafe improvements in terms of battery performance and accuracy can be made by using an external speedometer. These Bluetooth enabled devices communicating using ANT+ can be fitted to any bicycle to calculate speed and if utilized device battery life would greatly improve.

6.2 Conclusion

The purpose of this project was to develop a simple to use crash detection application using machine learning in real-time, and by doing so potentially save someone's life. As shown from the results, this goal was achieved. The author is delighted to have been able to achieve all aims originally set in the planning stage of RideSafe, as well as achieving all the personal goals set in chapter 1. RideSafe may not be ready for a full release at time of writing but the author was adamant to make their presence on the google play store, as shown in figure 6.1 RideSafe is currently available for open beta testing. Working in an area where specific relevant sensor data is not publically available has been a challenging experience, discovering unknowns, alone has been a very rewarding experience to which the author is extremely proud to have achieved. To the best of the authors knowledge RideSafe is the first sensor based application to use speed as a factor in determining whether a crash has occurred as well as classifying live sensor readings in real-time on a single device.

All functional requirements have been accomplished as well as laying down the foundations for many non-functional requirements. With time constraints as well as the logistics involved with real world testing and training, development of RideSafe has not been an easy feat to complete, however it has been a truly rewarding experience. Choosing to undertake my own idea for a project was daunting at first, with so many unknown variables at play at the beginning I had my doubts at the feasibility of completing this project. My motivation to create an application which one day may save a life kept me focused to reach my end goal.

By researching the innovations made in the medical domain as well analysing the pitfalls of existing solutions, the development of RideSafe has shown that existing solutions may not always be the optimal solution.

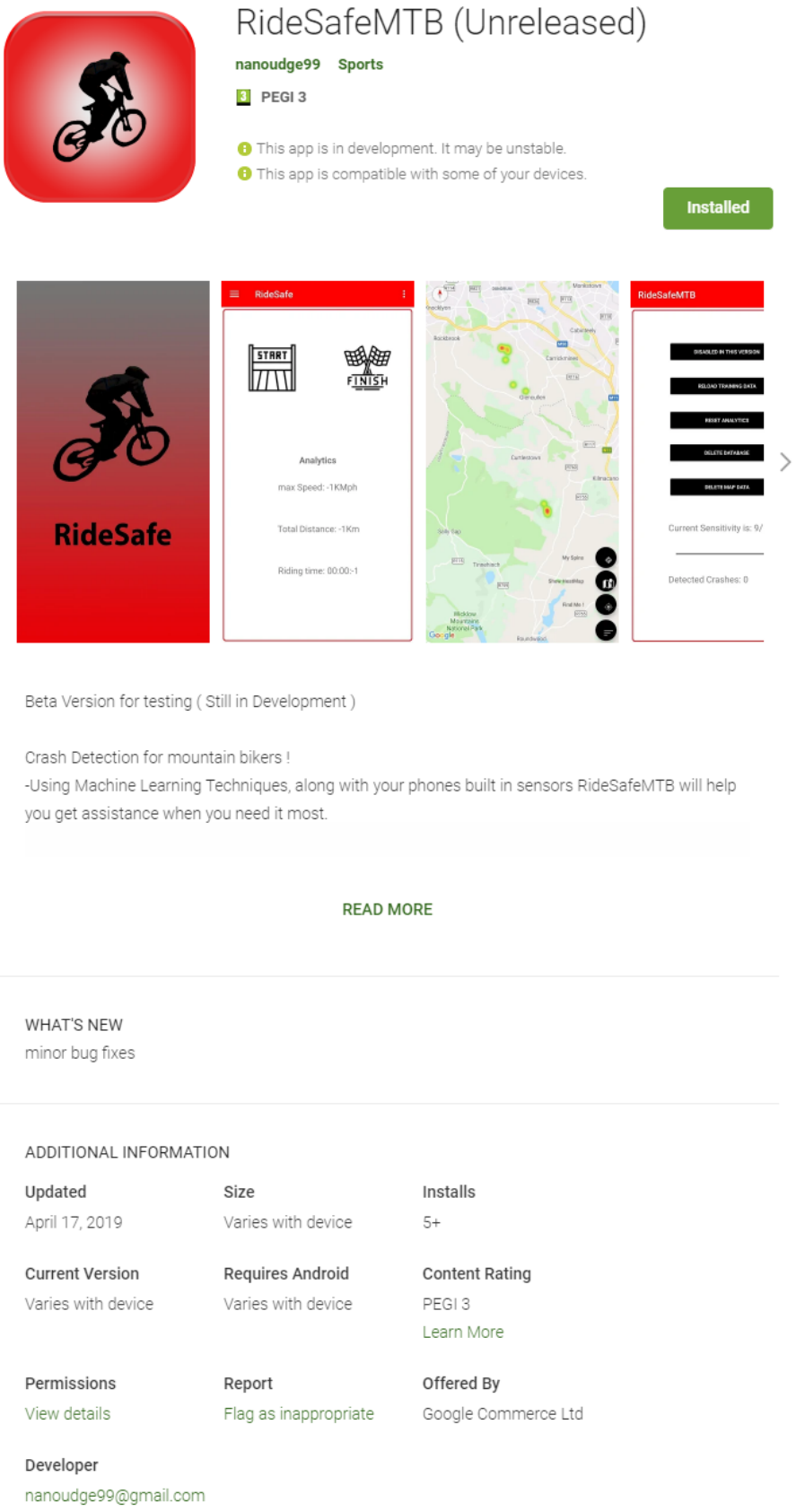

Figure 6.1: Play Store Listing For RideSafe

Bibliography

[1] Johannes Becker, Armin Runer, Daniel Neunhäuserer, Nora Frick, Herbert Resch, and Philipp Moroder. A prospective study of downhill mountain biking injuries. *British journal of sports medicine*, 47(458-462):1, 2013.

[2] Pekka Kannus, Seppo Niemi, Mika Pavanen, and Jari Parkkari. Fall-induced deaths among elderly people. *American Journal of Public Health*, 95(3):422–424, 2005.

[3] The Irish Times. Mountain-biking injuries go with the territory. `https://www.irishtimes.com/life-and-style/health-family/fitness/mountain-biking-injuries-go-with-the-territory-1.3384717`.

[4] Tony K Chow MD and Robert L Kronish MD. Mechanisms of injury in competitive off-road bicycling. *Wilderness and Environmental Medicine*, 13(1):27–30, 2002.

[5] Koninklijke Philips N.V. . Philips Lifeline, Automatic Fall Detection. `https://www.lifeline.philips.com/medical-alert-systems/fall-detection.html`.

[6] Specialized. ANGI FAQS. `https://www.specialized.com/us/en/faq/angi`.

[7] ICEdot. ICEdot Crash Sensor. `http://site.icedot.org/site/crash-sensor/?lang=en`.

[8] Natthapon Pannurat, Surapa Thiemjarus, and Ekawit Nantajeewarawat. Automatic fall monitoring: A review. *sensors (Basel)*, 14-7(12900-12936):1–4, 2014.

[9] Panagiotis Tsinganos and Athanassios Skodras. A smartphone-based fall detection system for the elderly. *Proceedings of the 10th International Symposium on Image and Signal Processing and Analysis*, 1:1–4, 2017.

[10] Chia-Yeh Hsieh, Chih-Ning Huang, Kai-Chun Liu, Woei-Chyn Chu, and Chia-Tai Chan. A machine learning approach to fall detection algorithm using wearable sensor. *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, 2016.

[11] Color Tool.
https://material.io/tools/color/#!/?view.left=0&view.right=0.

[12] Maher Maalouf, Dirar Homouz, and Theodore B. Trafalis. Logistic regression in large rare events and imbalanced data: A performance comparison of prior correction and weighting methods. *Computational Intelligence*, 34:161–174, 2018.

[13] Amit Singhai, Joy Bose, and Nagaraju Yendeti. Reducing power consumption in android applications. *2014 IEEE International Advance Computing Conference (IACC)*, 1:4–6, 2014.

A1 Appendix

A1.1 Other Resources

A beta version of Ridesafe is currently availible on the Google Play store which can be accessed by folloing this link:
https://play.google.com/store/apps/details?id=com.release.nanou.ridesafe

The source code for RideSafe is availible on the authors Github account :
https://github.com/monganai/RideSafe

A1.2 Supplied With This Report

Supplied with this report is a rescorce containing:

- The source code for RideSafe as well as a pre-compiled version of the application (.APK File)
- A subset of collected sensor values used for analyzing and training RideSafe

A1.3 Application Screenshots

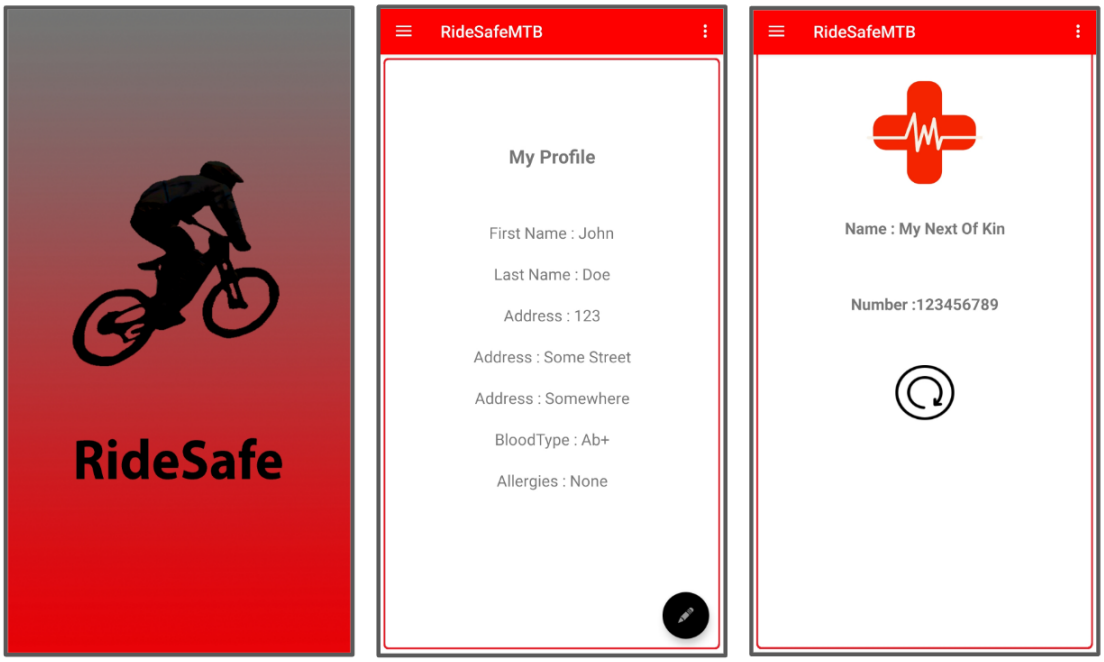

Figure A1.1

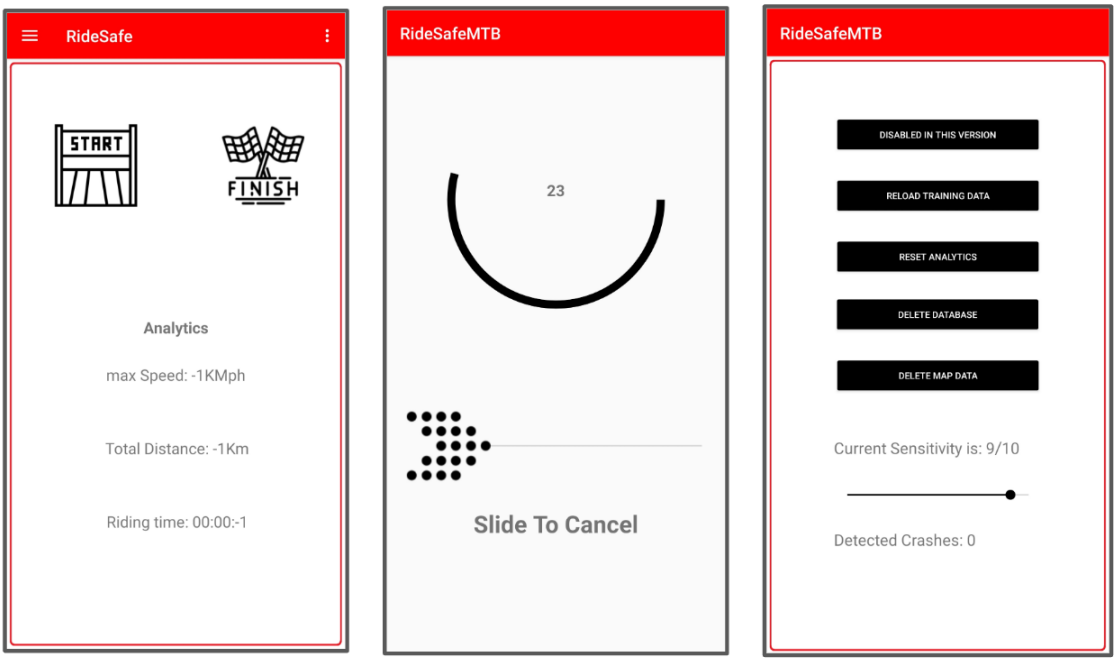

Figure A1.2

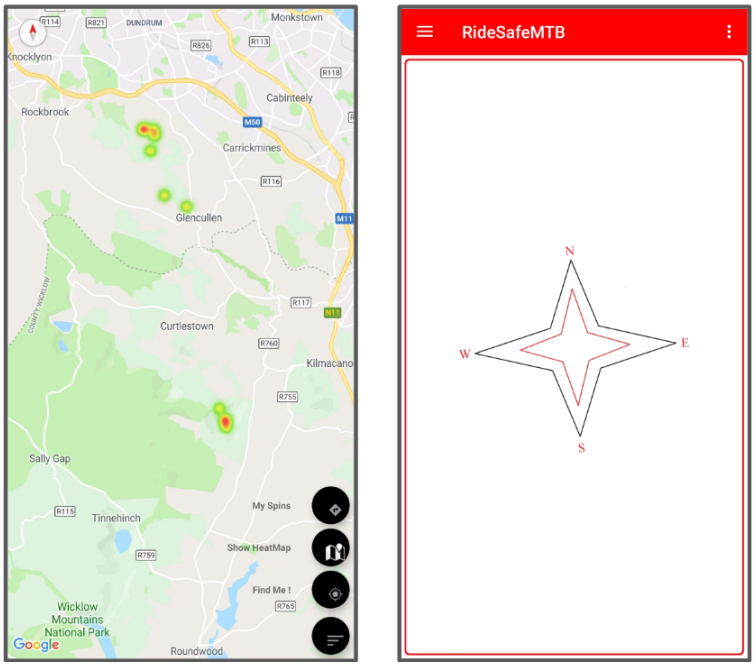

Figure A1.3