ASHESI UNIVERSITY COLLEGE

AUTOMATED COLLECTION AND VISUALIZATION OF ROAD QUALITY DATA TO AID DRIVER NAVIGATION

APPLIED PROJECT

B.Sc. Management Information Systems

Kwabena Gyang Boohene

2017

ASHESI UNIVERSITY COLLEGE

Automated Collection and Visualization of Road Quality Data to Aid Driver Navigation

APPLIED PROJECT

Applied project submitted to the Department of Computer Science, Ashesi University College in partial fulfilment of the requirements for the award of Bachelor of Science degree in Management Information Systems

Kwabena Gyang Boohene

April 2017

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Project laid down by Ashesi University College.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

Acknowledgement

I would first like to thank God Almighty for the strength given me to embark on this project. I would like to express my gratitude to all the individuals who aided me to complete this project. I would like to acknowledge Delali Vorgbe and Antoinette Doku for the advice and support they gave me in pursuing this project.

My special gratitude goes to my supervisor, Dr. Ayorkor Korsah who has supported and aided me tremendously from the beginning to the end of this project. It was through her that my love for computer science grew, and for that, I am sincerely grateful.

Finally, many thanks to the Computer Science Department of Ashesi University College, for providing a structured curriculum which has greatly supported my growth in many aspects of computer science.

Abstract

In nations worldwide, roads play a major part in the livelihood of individuals. They aid in the transport of both people and goods from one location to another. In developed countries, most roads are tarred making travel easier and quicker. In developing countries, this is not the case. Quite an amount of road work and development is still needed in these regions. These roads normally have different grades of road quality and have different effects on the individuals that use them. With the introduction of mapping applications such as Google Maps, individuals can place information about various locations on a mapping interface. In this project, an application is built to allow individuals collect data about the quality of roads around them. This data will be viewed on a map interface. With such data, drivers would be able to choose they type of roads they would want use.

Table of Contents

Chapter 1: Introduction and Background

Road transportation looks at the movement of individuals or goods from one location to another via a specific route. In Ghana, vehicles that travel via road carry the most passengers in the country as compared to those via rail, river, and air combined (Lamptey, 2016). A study held by a group of researchers in Indonesia revealed that road improvement could be a stepping stone for economic development through the opening of labor markets as well as decrease the income gap. On the downside of things, the deterioration of road quality could also be a setback to nations with unwarranted government expenditure (Gertler et al., 2016). It could lead to an increase in health care budgets, because of accidents that may occur due to the poor quality of roads.

Road accidents have had a great impact in Ghana. According to the National Road and Safety Commission, a total of 13,133 road accidents were recorded in 2015, in which 1,856 people lost their lives. This put the economic cost associated with road accidents at 1.6% of Ghana's GDP (Lamptey, 2016). On the positive side of things, these figures are an improvement on the number of road accidents recorded over the past three years. The rate of road accidents in Ghana significantly reduced by 17% in 2015 as compared to the figure recorded in 2014. (Botchway, 2016). To reduce this further, there is the need to continuously improve the infrastructure that supports our road systems in this country. A first step could be for the governing authority to find a way to constantly receive updates on the quality of roads in an area, thereby enabling a quick response to the deterioration of any of this infrastructure.

Such information could be easily collected with a tool that is easily accessible to a large percentage of the population. Today, one of many such tools is the smartphone. As of 2015, a median value of 54 percent of individuals across 21 emerging and developing

countries are reported to owning a smartphone or using the internet (Poushter, 2016). This rise has largely influenced the use of navigation tools, some of which include Google Maps and Apple Maps. These tools aid individuals in navigating into known areas as well as unknown ones but fail to provide users with road quality data about roads in those areas. This limits the decisions of road users as to which routes to use based on the quality of these roads.

1.1 Proposed Solution

My solution is to build an application which will allow drivers to collect road quality information on the different roads they use. Users will be able to view the data gathered on other roads as well, to aid them when navigating these roads. Once this major aspect of the application is completed, research will be done into how to frequently update the information on a map interface using the aspect of crowdsourcing. Crowdsourcing is the process of engaging a group of people for their skills, ideas, and participation in aiding in the development of products (Goodrich, 2016). This project builds on research done by Delali Vorgbe (Vorgbe, 2014) and Antoinette Doku (Doku, 2014), two Ashesi alumnae.

1.1.1 Prior Work

Delali Vorgbe's project involved designing an algorithm which was used to classify the surface quality of a road, from sensor data collected from a smartphone. The classification classes determined by the algorithm were three These classes were good, fair, and bad. Furthermore, he built an Android application to record sensor data needed to test the algorithm. Both the algorithm and Android application will serve as the basis on which my application will be built. The sensor data collected includes GPS values (longitude and latitude points on the Earth's surface) and linear acceleration data. Linear acceleration is the

measurement of the acceleration effect on the movement of a device, excluding the effect of the Earth's gravity (Innovations Inc, n.d.).

Antoinette Doku's project on the other hand involved researching into whether embedding road surface quality data on a map interface would be of use to drivers. The results from her research provide some grounds on which this project is built. My goal is to develop a working application that allows individuals to be able to view the data collected by other road users, which will aid them in route navigation.

1.2 Related Work

1.2.1 Participatory Sensing

As stated earlier, the smartphone industry has grown tremendously especially in emerging economies (Poushter, 2016). This growth has provided individuals, firms, and industries a new medium through which they can collect and deliver information to many users. Consequently, it has pushed the boundaries of crowdsourcing data, which could initially only be done with computers and has introduced a new concept known as participatory sensing (Kanhere, 2013).

Participatory sensing can be characterized into two categories; people-centric or environment-centric. People-centric sensing looks at collecting information about individuals and using this information to their benefit. Examples include health monitoring applications and social media applications. Environment-centric sensing on the other hand is the use of mobiles phone to capture information about the immediate surroundings. This information is usually used to tackle issues or problems in a community (Kanhere, 2013).

This collection of information is possible through the inbuilt sensors that allow these smart gadgets to track, amongst other things movement, orientation, and positioning. More precisely, these smart gadgets are fitted with sensors such as the accelerometer, gyroscopes,

GPS (Global Positioning System) and location sensors. (Rosario et al., 2015). This project focuses entirely on the environment centric sensing approach. This is because a major aspect of the project involves individuals using an Android application to record vibrations generated by the vehicles they use when traversing different roads. Users will also be required to record their GPS coordinates when using these various roads. This will involve the use of a linear acceleration sensor and a GPS sensor which are embedded in most Android devices.

1.2.2 Data Recording and Transfer

Furthermore, this paper looks at how to collect and display road quality data to users. In tackling this issue, some important areas to consider include, the type and volume of data being sent to the database server, and how all this information would be accumulated. Consideration should also be given to the frequency of data transfer between the Android device and the server, whether real time or within specific intervals. Based on related research, there were several approaches found with regards to the collection and sending of sensor data from smartphones to servers.

A first approach looked at performing all the filtering of sensor data on the Android device. This data was then directly transferred to a server which was displayed on a map interface. Vittorio et al. (2014) use this approach. The system they had built also involved some aspects of road surface data collection and classification, but their classification was based on the number of road anomalies on a specific road. In their implementation, immediately their application began, two filters were applied to the acceleration data collected on their smartphone device. This data was then sent in real time to a central server. If a consistent number of events (vertical acceleration impulses) higher than a given

threshold happen in a specific location over a period, that location was marked with the title, “road anomaly” on the mapping application being used.

The second approach to data recording and transfer looked at sending compressed raw acceleration values collected by a smartphone device to a server for processing. An example is an algorithm developed by Lauer et al. (2013). This looked at collecting and sending compressed acceleration vector values collected by a smart phone to a server via a web service. This web service then stored the data, processed it, and displayed it on a mapping application titled OpenStreetMaps (OSM). OSM is an online platform that enables online masses to create a free, editable map of the world. This map is constantly updated by thousands of volunteers around the world (Buczkowski, 2015).

Considering one of the goals of this project is to reduce the size and amount of data sent back and forth between the mobile phone and the server, a similar approach is considered in this project.

1.2.3 Spatial Database Management

Another important feature to look at is the architecture involved in adding spatial data to the database of the system proposed in this project. Spatial data or geospatial data is the use of numerical values to represent the location, size, and shape of an object on the Earth’s surface. An example of spatial data types includes, the longitudinal and latitudinal degrees on the Earth’s surface (Rouse, 2013).

Consequently, Goetz et al. (2012) looked at this task using a tool called OSMOSIS. This is a command line Java tool that can read OSM (OpenStreetMap) map formats and store the extracted data in an SQL format. Due to its (OpenStreetMap) crowdsourced nature, data on the platform is updated frequently. Thus, to keep their (Goetz et al.) system updated, the new OSM data was imported using the OSMOSIS tool and inserted into an SQL

database. They (Goetz et al.) could now include any data they would like to be displayed on a map interface.

This data is then displayed using an open source tool called Geoserver. Considering this, their database is structured using a concept known as “views”. The use of “views” allowed them to easily manipulate and determine the kind of data to be displayed on the Geoserver interface. Despite this, one major drawback of the system was, it took quite some time for the database to be updated. The initial import of OSM data took about 8 hours, and regular updates could take as long as 3 hours.

Alessandroni et al. (2014) have a different approach in keeping their database updated. With their approach, an Android device was used to collect and compute data about the different roads an individual might use. After this data was computed on the Android device, it was sent to a Linux machine running ASP.net and stored in a PostgreSQL database. To improve the accuracy of the collected data points, the data was then mapped to the closest road using a geographical database. A background process then used an aggregation method to compute finalized data which was then pushed to an external CartoDB server. CartoDB is an online service that allows easy visualization and handling of maps with rich data overlays (Alessandroni et al., 2014). This entire approach cuts out the aspect of importing and storing large amounts of map data. This reduces the cost involved hardware and software maintenance, and reduces the amount of time used in updating any spatial database system as compared to the first approach.

Considering both alternatives to keeping a spatial database updated, the second approach has been considered. Instead of using the CartoDB server, Google’s mapping application “Google Maps” will be used to display the roughness data. Also, a MySQL

database would be used instead of a PostgreSQL database. This is because using the first approach would require a lot of resources just to keep the system updated.

1.2.4 Visualization of Spatial Data

Furthermore, many approaches have been used for visualizing data on a map interface. One of the most popular is the usage of colors and shapes to depict different objects on the map interface. An example is how “blue” is often used to display the ocean or fresh water (Rosenberg, 2017). This also applies to other aspects on a map interface such as roads, building, countries and regions. Google Maps uses such an approach by giving objects related to vegetation different shades of green. Also, objects involving human activity (different types of roads – freeways, highways) are given different shades of orange (Li, & Bailiang, 2016).

Consequently, Google Maps also uses colors when displaying the amount of traffic on a specific road. Green means no traffic delays, orange means medium amounts of traffic and red means heavy traffic (Google Inc., n.d.). Unfortunately, Google Maps fails to provide this color distinction for roads of different kinds of surface quality. This project is looking to solve this issue. It is intended that three different colors will be used to differentiate roads of varying surface quality. These include green for a road segments of good quality, orange for a road segments of fair quality and dark grey for poor-quality road segments.

1.2.5 Related Technology

Applications directly related to this project include SmartRoadSense (Alessandroni et al., 2014) and Roadroid (Forslof et al., 2013). Both applications present road condition monitoring with Android phones but apply different algorithms in determining road surface quality. SmartRoadSense for example uses Linear Predictive Coding to estimate roughness data from sample points. Linear Predictive Coding allows the prediction of certain values

collected from an analog signal by a linear combination of past values and the signal itself. Linear Productive Coding was used to remove acceleration data not attributed to irregularities on the road surface (Alessandroni et al., 2014).

Consequently, upon doing some research on the Roadroid platform, there was very little insight as to what kind of classification algorithm was being used on the platform. Despite this, some insights worth mentioning include a regression analysis on data collected by a physical accelerometer attached to a rear axle of a car. The results from this analysis showed that, the system could classify correctly 70% of the data collected as compared to an average of subjective visual expert inspections. This was the original version of the Roadroid application before being ported to the Android platform. Unfortunately, on the Android version, there is no evidence of the afore mentioned regression analysis being used.

Another notable fact is that, the creators of Roadroid have developed a correlation index between their road condition data algorithms and IRI (International Roughness Index) values (Forslof et al., 2013). This correlation index is known as the eIRI value. The IRI has become a worldwide standard for evaluating and managing road systems (Karamihas & Sayers, 1996). The eIRI on the other hand is an estimated form of the IRI. This correlation increases the validity of their surface quality results. Roadroid also uses colors to distinguish the various kinds of surface quality.

In this paper, the approach used is more similar to the Roadroid application than SmartRoadsense. A major difference between the examples mentioned above and the approach used in this paper is the difference in the classification algorithms. As stated earlier, the classification algorithm used is based on a logistic regression algorithm developed by Francis Delali Vorgbe (2014). On the other hand, SmartRoadSense uses

Linear Predictive Coding, whilst the original version of Roadroid also used some form of regression analysis.

Other differences include the different map interfaces used in displaying the surface quality data. Roadroid uses OpenStreetMaps, and SmartRoadSense uses the CartoDB server. In this project, Google Maps will be used to display the road surface quality data. The major reason is because of its wide popularity on smartphone devices as it ranked 7th out of the top ten apps downloaded on Apple devices in 2016 (Bell, 2016). Also, Google Maps has an extensive API (Application Programming Interface) that is supported by a wide range of developers.

1.3 Overview of Chapters

This paper spans six chapters. In this chapter, readers are given an introduction and background to the project. This chapter also discusses the motivation behind the project, prior research done in relation to the proposed system and existing applications that are somewhat similar to the proposed system. Chapter 2 discusses the unique features that the system must possess and the target audience of the system. It also provides different scenarios of how this application will be of value to targeted users. Chapter 3 presents readers with a detailed explanation of the system design and architecture. It provides an explanation of the different modules that will be used to create the system and designs of the different external views that the application will contain.

In chapter 4, a description of the implementation process used to develop the system is given. This involves the different technologies and tools used in creating the system. Chapter 5 shows the different tests carried out on the system, and the results from these tests. It also gives a description of the solutions to the problems identified from the tests

carried out. Finally, chapter 6 concludes the paper by providing a summary of the entire project, limitations of the system and some further work required to improve the system.

Chapter 2: Requirements Specification

The goal of this chapter is to provide a detailed description of the functionality that the road surface quality system will possess. It describes the project's target users, as well as different scenarios in which the application would be useful to these users.

2.1 Project Overview

The road surface quality system consists of two main components. The first component is an Android based client which collects raw accelerometer sensor data and uses this data to determine the surface quality of the road (good, fair, poor). The data is then reported to a server to be aggregated for access by users.

The second component is a server-side application which displays the collected data on a map interface for users to view. This data is displayed using three different colors. Dark grey signifies a road segment of poor quality, orange signifies a road segment of fair quality, and green signifies a road segment of good quality.

2.2 User Classes and Characteristics

The road surface quality system is built to aid drivers when navigating known or unknown areas. From various conversations with different drivers, it was realized that the individuals that would tend to use this application regularly are the drivers of private vehicles. This is because, since these drivers own their vehicles they tend to be more careful about the routes they use, to reduce the cost of maintenance and repair on their vehicles. This category also includes taxi drivers who own their vehicles as well as Uber drivers. Drivers that provide public transport on the other hand have no choice about route selection. The routes they use are fixed thus, the surface quality of the road is not a major factor to them.

2.2.1 Use-case Scenarios

Private car owner (Travel to an unknown location):

A group of university graduates are planning a vacation trip to the Central Region of Ghana, to celebrate their graduation. They have never been to this part of the country before and are unsure about which of their vehicles they should travel in. To check the distance to this destination and the quality of the roads that they might use, one of these friends opens the application and is immediately brought to the area of his current GPS position. He then enters the destination he wants to travel to, and the map interface populated with lines of different colors, representing the quality of various roads. Thus, the group decides to go with their friend's Toyota Land Cruiser since the roads within the Central Region are of fair and poor quality. This is because, the Land Cruiser's four-wheel drive system will allow them to easily navigate roads of such quality.

Private car owner (Travel to a location visited long ago)

A private car owner is looking to buy some new clothes for himself. He remembers a shop that he used to visit a long time ago, but he has not been there recently. Rumors have it that the different roads to his destination have deteriorated quickly, but he wants to be sure. He opens the application and inputs his destination. On the map interface, lines whose color indicate the road surface quality are superimposed on various road segments. He thus decides to use a different route from what he is used to because of the poor surface quality of his route.

Taxi Driver (Declining a ride)

A taxi driver is stopped by a passerby who wants to go to a specific destination that the driver doesn't know but the passenger does. The driver quickly opens the application, searches for the area the passenger is going to, and is given results. He sees that roads going

to the passenger's destination are shown to be of poor quality. He decides to decline the passengers request since the road would negatively affect the vehicle he is driving.

2.3 Product Features

Below is a list and brief description of the features and functions of the road quality system. This has been divided into main features and additional features. The main features are necessary and must function to their best capacity. The additional features will only be implemented when the core features are functioning without a doubt.

2.3.1 Core Features

1. Data recording & classification:
    - The user clicks the record button
    - The smartphone device records raw sensor data in the background.
    - The application classifies the recorded data in the background periodically after a sizeable amount of this data (recorded data) has been collected on the smartphone.
    - The user stops recording and clicks the post button to transfer data to the central server.
2. Data processing and storage
    - The central server receives data from the smartphone device and inserts this data into a MySQL database.
3. Data visualization
    - The user opens the application and navigates to map interface.
    - The user enters their destination.
    - The map interface displays colored lines which show surface quality of different roads within a given boundary.

2.3.2 Additional Features

4. GPS navigation:
    - This enables individuals to navigate from one GPS location to the other on the map interface.
    - It provides a system that warns users of road surface quality whiles navigating.
5. Voice aided navigation:
    - This voice aided navigation feature will notify users ahead of time about the surface quality of the road on which they would be driving on. This notification prevents them from losing concentration when they are driving.

Regarding the core features, the following page contains activity diagrams, that gives a vivid description of the flow of information.

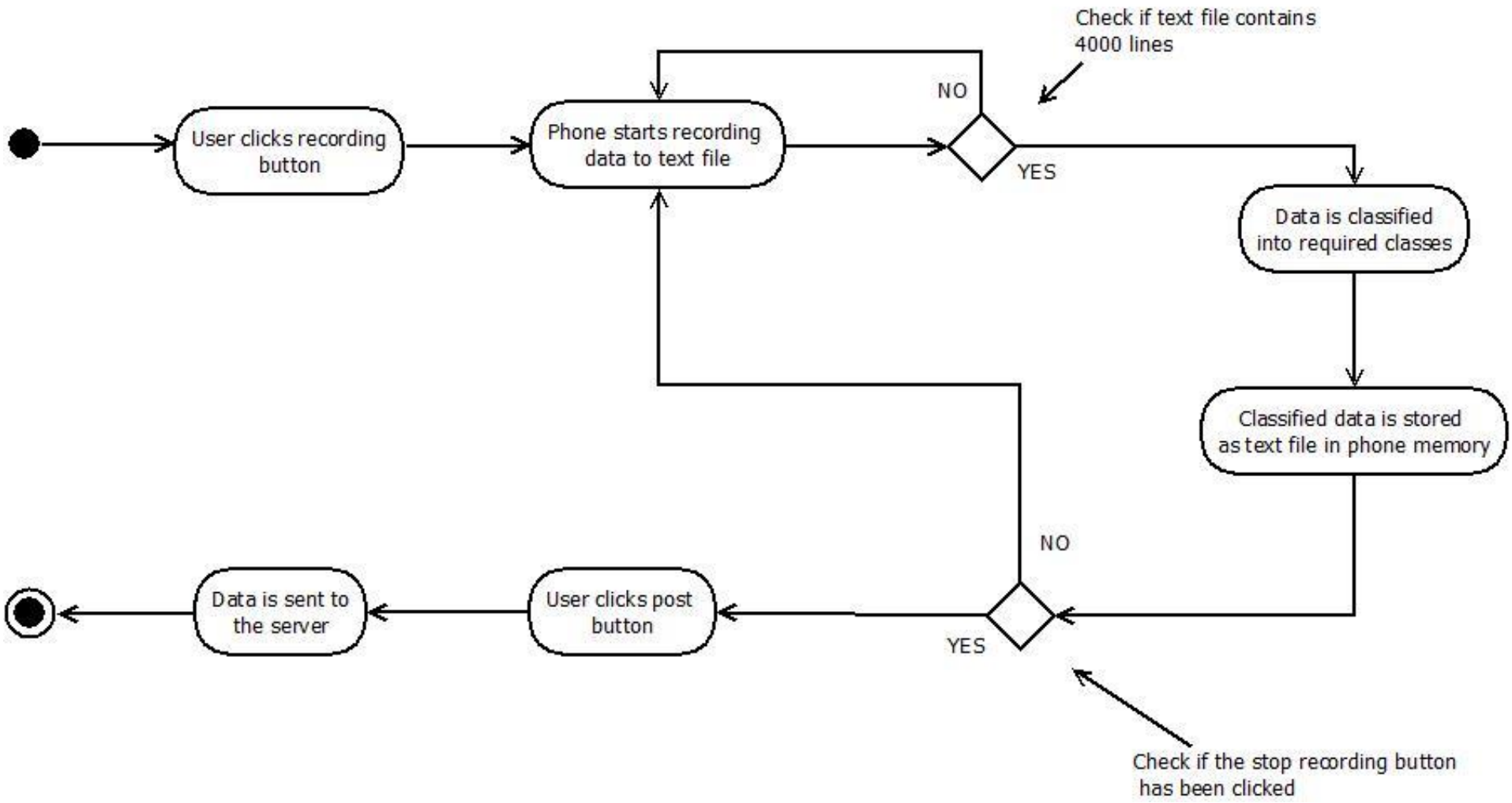


Figure 2.1: Data recording and classification

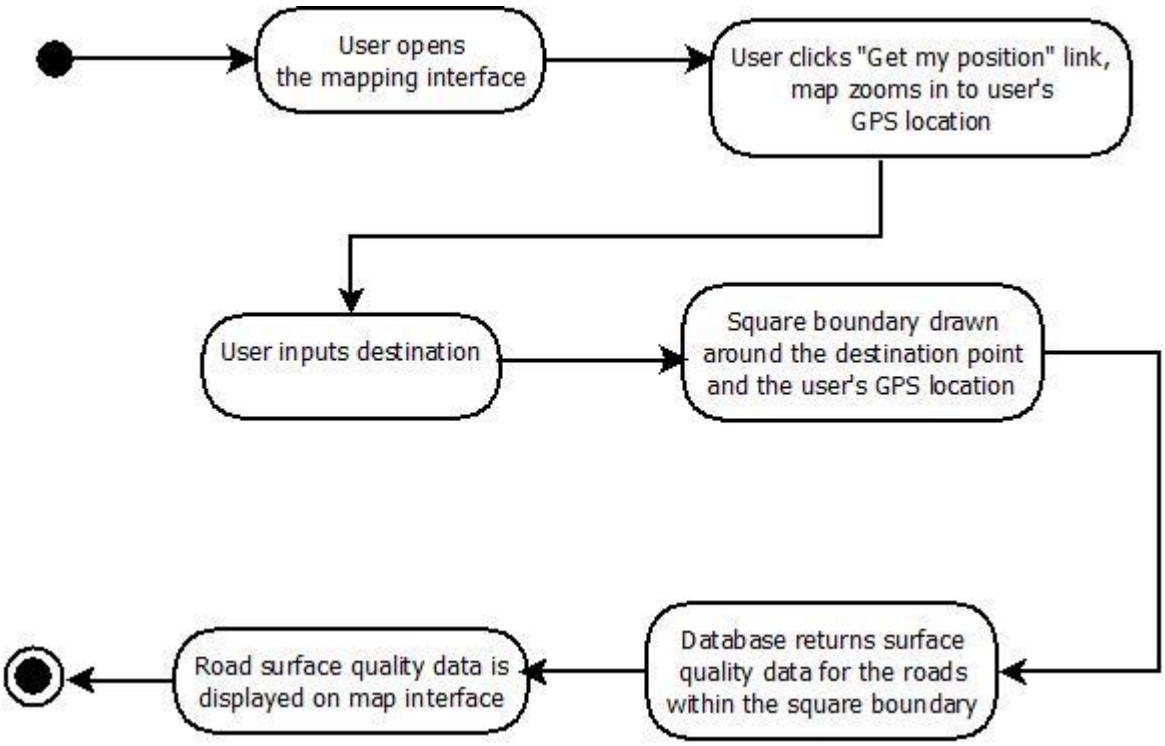


Figure 2.2: Data visualization

2.4 Non-functional Requirements

The major security concern will be on the collection of GPS data from users. No personal user information is collected that connects users to the GPS data collected. Also, the original sensor data will not be viewable to anyone using the application. This is because as soon as a file is classified, the original data file is destroyed. Furthermore, the files containing the road surface quality data will be deleted from the user's device after it has been uploaded to the server.

Chapter 3: Architecture and Design

The system architecture of an application is the overall blueprint of the application. It gives an overview of how an entire system works. This includes the modules or components that make up the system, how these modules are structured and the interactions between these modules. The modules of a system are primarily built to meet the functional requirements, whilst the arrangement of these components take care of the non-functional requirements (Sommerville, 2011). Given this, the architecture styles to be used in this project are that of a Client Server architecture and a Model View Controller architecture.

3.1 Client-server Architecture

In the Client-server architecture, different services running on a system are split into two main parts, the client, and the server. The server contains the resources needed to run a system, whilst the client sends requests to the server to use these resources (Sommerville, 2011). In this system, XAMPP is the application going to be used to run the web server. This application comes with a version of the Apache Web server, which allows the running of server scripts, and a database server that stores information to be used on the application. The database server is run using MySQL which is My Standard Query Language, and is managed using a graphical interface known as phpMyAdmin.

The main client in this application is a web browser, examples include Google Chrome, Mozilla Firefox or Opera. The Android device also serves as a Client, but only when sending data to be stored in the MySQL database.

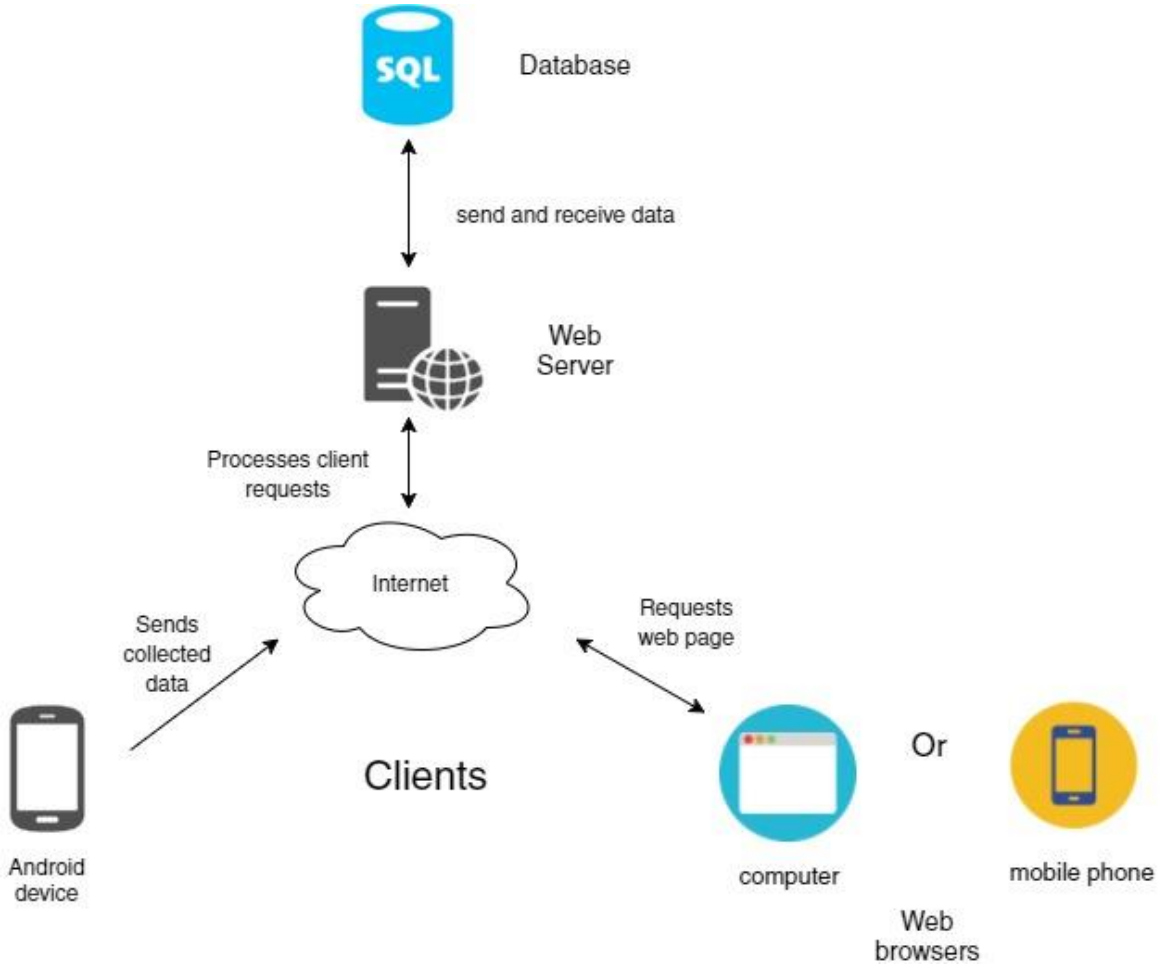


Figure 3.1: General system overview

In a Client-server architecture, there is the need for a clear separation between the presentation of data and the computations done on this data. This is done by separating the application into a set of logical layers with different interfaces. These layers are four in number namely, the presentation layer, data management layer, application processing layer and database layer (Sommerville, 2011).

3.1.1 Presentation Layer

The presentation layer contains the views that users are presented with. It is the only layer that users come to contact with when using an application. In designing this layer for this project, factors that were taken into consideration include:

Web software versions: Users would be able to access data only via browsers that support geolocation.

User experience: The user interface on both the web and Android platform were developed to make navigation on both interfaces as simple as possible.

3.1.2 Data Management and Application Processing Layer

The data management and the application processing layer oversee the type of data being shown to the user. It is here that the functionality of the application is worked on to meet the user's requirements. Here, different web technologies are used to interact with the database to format the data in a presentable manner to be viewed.

Server side scripts are used to send queries both to store and select road surface quality GPS points from the database. These queries are based on location inputs from the user via the presentation layer. Appendix A shows a code snippet of the 'coordinates' class that contains functions used to insert data into the database. In implementing this application, scripting languages are used to create data structures such as arrays and binary trees to reorganize these GPS data points to be drawn on a map interface.

3.1.3 Database Layer

The last layer is the database layer. It is the foundation on which the entire system functions. This handles the storing of data in the database, as well as processing requests written in Structured Query Language. The data stored are mainly different GPS points, the association grade of road quality (good, fair, bad), an identification number for a specific route and a pointer to the next GPS point along a given route. The diagram below provides an Entity Relationship diagram containing the schema of the database.

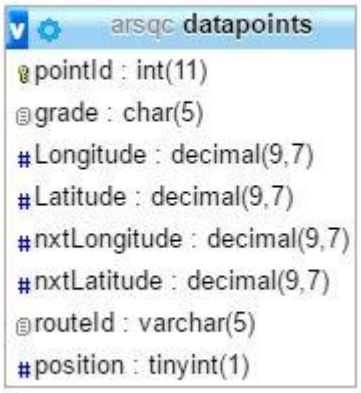


Figure 3.2: ER diagram

3.2 Model View Controller (MVC) architecture

This architectural pattern involves the splitting of an application into three parts: The Model, the View, and the Controller. This architectural approach was used in designing the Android aspect of the entire system.

3.2.1 Model

The Model refers to the aspect of an application in which data is stored. It could also be termed as the database of the application. In this scenario, the Model used is a basic text file. Lines containing road surface quality data are written to a text file and stored temporarily in folders in the memory of the mobile device.

3.2.2 Controller

The Controller in an application works in the background. It controls the flow of information or data between the model and the view of an application. For the application, the controllers are called ‘services’. Services are classes that perform computations in the background of an application and provide results to either the Views or the Model data. The ‘services’ used in this application include a classifier service and a post service. The

classifier service reads raw text data stored as a Model and outputs text files containing road surface quality data. The post service sends all the text files containing the road surface quality data to a central server via the use of an Android HTTP client library.

3.2.3 View

The Views in an application are the different user interfaces that are presented to the user. Here, information is presented in a format that is pleasing to the user. User data is also collected through these interfaces. The Views in the Android application are known as 'activities'. Activities are written in both Java and XML programming languages, but the XML aspect performs the display function. Some 'activities' in this project include the home page and recording page.

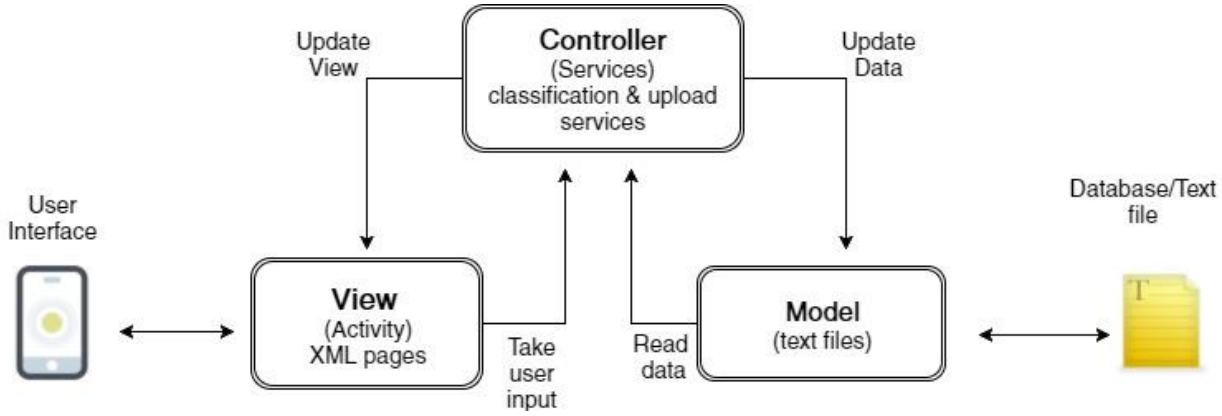


Figure 3.3: Display of MVC model

3.3 External Interface Requirements

The interfaces built for this platform are meant to be simple, interactive and welcoming for the user. There are interfaces for both the web and Android applications. The interfaces on the Android platform are divided into four main pages: the home page, the recording page, and the web view. The home screen displays a simple welcome message for the user. It also contains two buttons that allow users to move to either the recording page or the web view. The recording page allows users to record road quality data and post

this data to a central server. The web view contains a mobile view of the map interface. This allows users to use the map interface on their mobile devices.

The desktop web view contains a simple web page that displays a Google Maps interface. Users can view road surface quality data on this interface. Appendix E contains an image of the desktop web view.

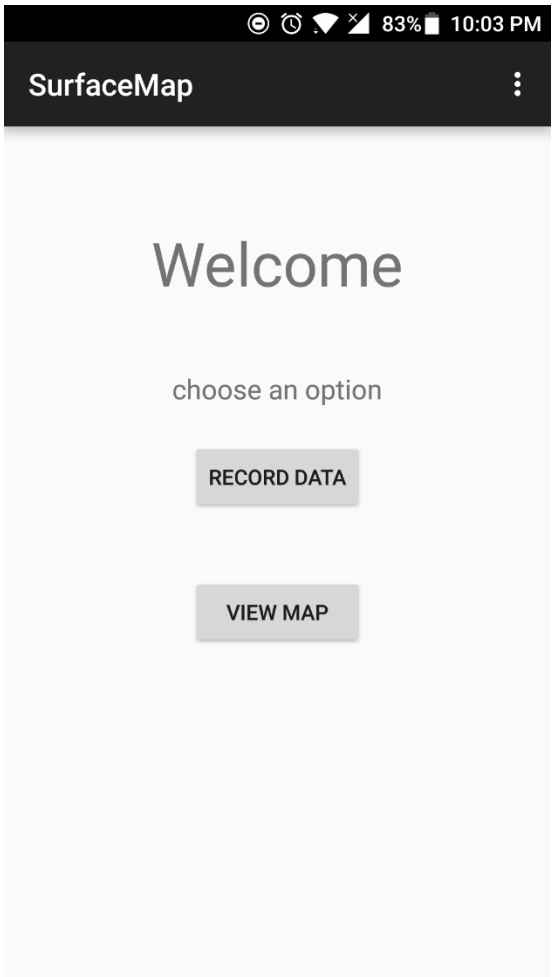


Figure 3.4: Home page

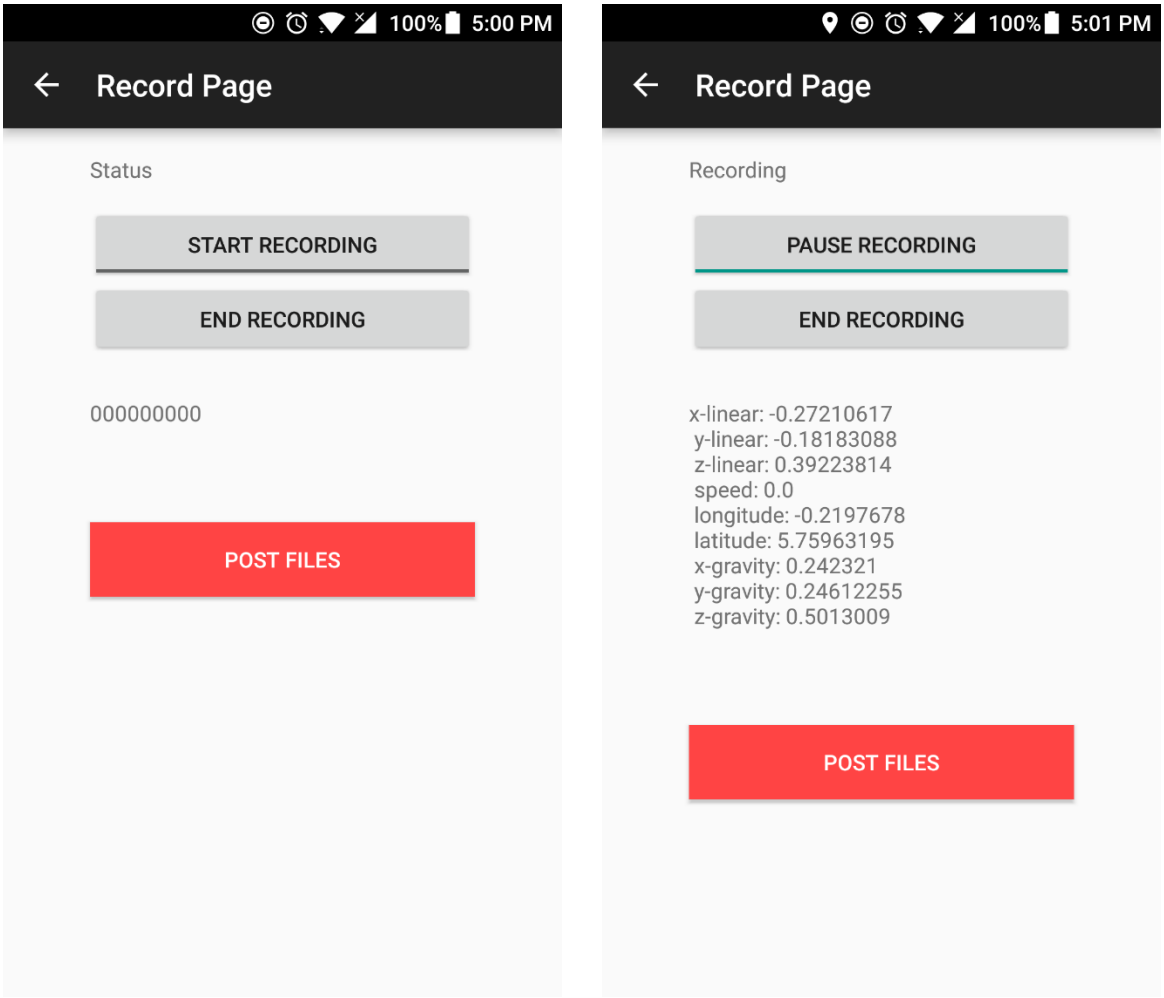


Figure 3.5: Recording page

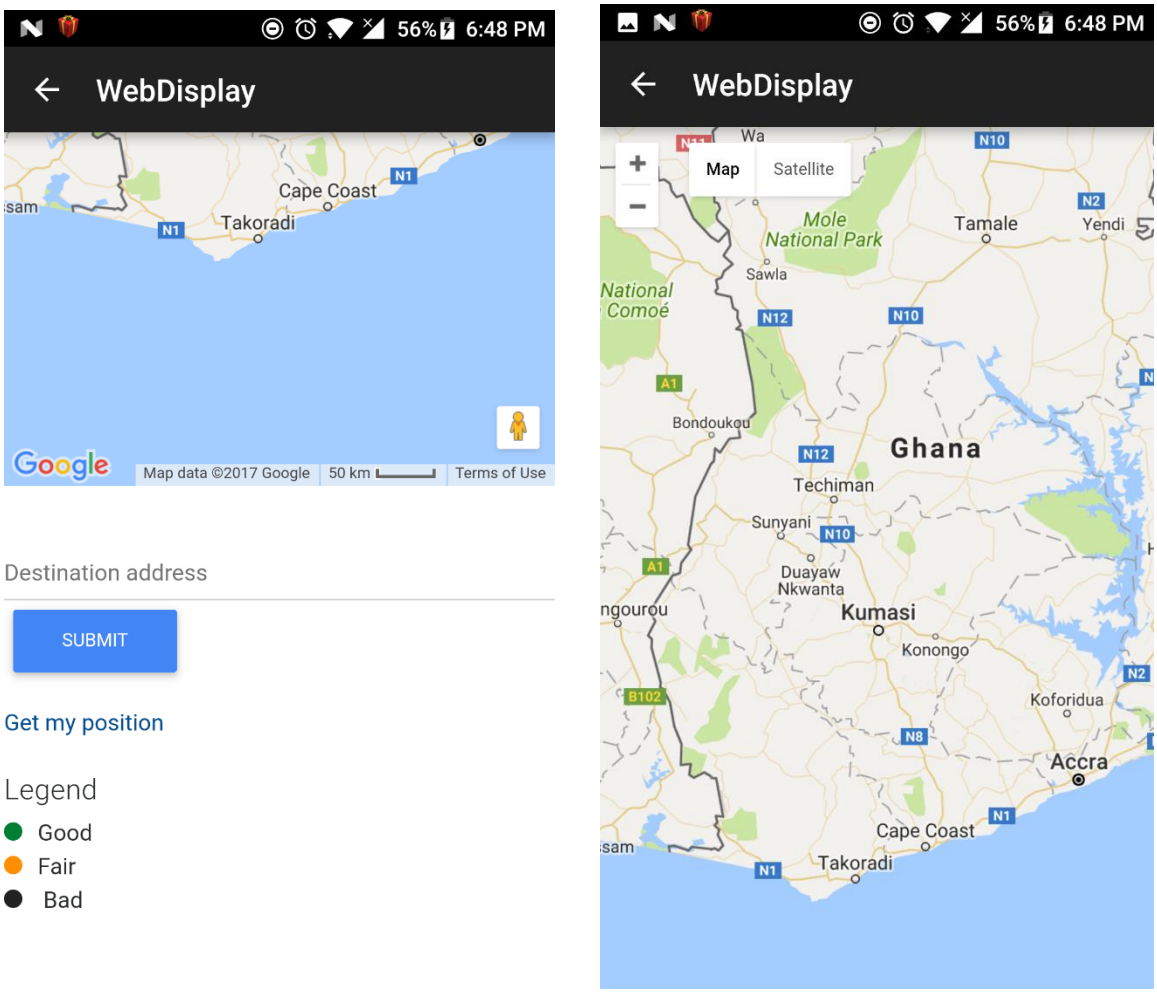


Figure 3.6: Mobile web view

Chapter 4: Implementation

This chapter gives a detailed explanation of how the road quality application works. As mentioned earlier, the work of the system is done in two parts, an Android application, and a web server. This section gives a detailed explanation on how both platforms work. We will first examine the hardware and software needed to run the system, and the various programming languages used in building the entire application. This will be followed by an explanation on how the Android application collects and classifies road quality data, and how this data is manipulated to be efficiently stored in a database. The chapter concludes with the approach used in displaying this collected data on a Google Maps interface.

4.1 Hardware Requirements and Software Requirements

With regards to the Android application, there is the need for an Android smartphone enabled with a linear accelerometer, and a GPS sensor. This device must also be equipped with the capability of accessing the internet. Secondly, this device should be running an Android OS (Operating System) version of API 15 or above. This system will be a native application developed using Java and XML programming languages on the Android studio platform.

On the server side, the computer hardware should be able to run either a Windows or Linux operating system. The server should also be running a browser that supports geolocation. Examples of such browsers include Chrome version 5+, Firefox version 5+, Opera version 10.6+ and many others (Pilgrim, 2010). The system will also be required to run an SQL database to store the transferred data, and should support RESTful Web services. The programming languages to be used include PHP, JavaScript, HTML, CSS, and SQL. The visualization of data will be done using the Google maps JavaScript API.

4.2 Android Application

4.2.1 Road Quality Data Collection

This entire approach used to collect road quality data is based on an original configuration developed by the pioneer of this project, Delali Vorgbe. The data collected is based on vibrations generated by a moving vehicle. These are collected using the linear accelerometer on the Android device. The main constraint with the Android application comes from use of the phone's sensors. The orientation and position of the smartphone device affect how the values are recorded. Moreover, the smartphone must be in a fixed position to record the required sensor information correctly. After the device has been positioned correctly, the application is started a few minutes before the vehicle starts moving. Lines of raw data containing certain features are written to a text file. These features include:

<X-Linear acceleration, Y-Linear acceleration, Z-Linear acceleration, Longitude, Latitude>

As stated earlier, linear acceleration is the effect of acceleration on a moving device, excluding the effect of the Earth's gravity. This effect is measured along the three axial planes namely, X, Y and Z (Innovations Inc, n.d.) using a linear accelerometer. The longitude and latitude coordinates are recorded via the GPS sensor on the device.

The linear accelerometer records at a frequency of 4HZ and the GPS sensor records at a frequency of 1HZ. This means that four lines of data are written to the text file every second. Also, the classification algorithm classifies data within a ten second window. Due to such restrictions, it was decided that a text file can contain a maximum of 4000 lines.

To ensure that both accelerometer and GPS data can be narrowed down to one second, the total number of lines chosen must be firstly divisible by four, and then divisible by ten. The number "four" is because of the rate at which the accelerometer records data to

a text file (4HZ). Ten is because of the size of the window used by the classification algorithm.

4.2.2 Road Data Classification

The algorithm used for classification was originally built using a logistic regression equation implemented through GNU Octave (A powerful programming language used for complex mathematical equations). Before values are passed to the logistic regression equation, a set of feature vectors are extracted from the raw data stored in the text files. The features are based on the linear acceleration values recorded per a ten second window.

These features include:

Z-Mean– The mean value of all Z values in each window.

Z-Variance- The variance of all Z values in each window.

Z-Standard deviation -The standard deviation of all Z values within the 10 second window.

Z-HPeak – The highest Z value within each window.

Z-LTrough. – The lowest Z value within each window.

The last feature vector is the integer "1".

Without going into too much detail, the result of the computations done in GNU octave resulted in an equation in the form:

$$\theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Eq. 4.1: Logistic regression algorithm

The $\theta^T{}_x$ symbol is the hypothesis function that determines the quality grade of a specific road segment (good, fair, bad). The elements θ_0 through θ_n refer to values of individual parameters (Theta values), whilst those from X_0 to X_n are the feature vectors extracted from the raw data. The GNU octave platform provides the values for the various individual parameters. This classification algorithm also enables multiclass classification. This means one class can be placed against the others to determine a road section's quality. In this project, this approach of classification was used to determine the road quality. This resulted in three hypothesis functions, with three different set of individual parameters, two of which can be seen below.

Good vs Bad/Fair

Theta Values: 38.388669, -1.658378, 4.517424, -23.416034,0.305482, -1.711021.

Bad vs Good/Fair

Theta Values: -0.641961, -1.009224, -3.973086, 17.536464, -0.565364, 0.681880.

Once these were obtained, a Java class was created perform the rest of the classification. In this class, the theta values are multiplied by the extracted features in a fashion similar to that illustrated in Eq. 4.1 ($\theta_0 x_0 + \ldots \ldots + \theta_n x_n$). The result is then passed through a sigmoid function which ensures that the result is a value between 0 and 1. Unfortunately the classifier used is not full proof, there is still some amount of work needed to improve it. Below is a list containing the three versions of the multiclass classifier as well as their accuracies.

Good vs Bad/Fair Classifier: 87 percent accurate.

Bad vs Good/Fair Classifier: 52 percent accurate.

Fair vs Good/Bad Classifier: under 52 percent accurate.

Given the above list, the values obtained by the last classifier are disregarded due to its high inaccuracy. To finally determine the quality of the road, the values obtained by the first two classifiers (Good vs Bad/Fair and Bad vs Good/Fair) are compared against the threshold value of 0.5.

For example, if classifier one (Good vs Bad/Fair) gave us a value of 0.8 and classifier two (Bad vs Good/Fair) gave us a value of 0.4, the grade given to that point is “good”. In another instance, classifier one may give a value of 0.3 and classifier two may give a value of 0.6. We would rate the road section as “bad”. A last instance might be when both provide values below 0.5. In this instance, we rate the road as “fair”. These grades are then written to a text file along with their longitude and latitude values, an identification number recording the route, and variable determining the position of a point (start, middle, end).

There was the need to convert this implementation done on the GNU octave platform to a format that could run on an Android application. Thus, the Java programming language was used. Some important classes developed for this implementation include:

ClassifierService.java- This performs the comparison and multiplication of theta and feature values to determine the road quality grade.

FeatureExtactor.java- This extracts the features from the raw data collected.

Code snippets of both classes are represented in the Appendix C and B respectively.

After all this, the user clicks a button that pushes the data to the webserver via a background HTTP service. Appendix D contains a sequence diagram that gives a visual explanation of the entire process.

4.3 Web Server

The role of the web server is to store the data sent from the Android application in a database, and secondly facilitate the display of this data on a map interface for viewers to use. The webserver was built using four specific tools. These were the Google Maps API, a Geotree JavaScript class, the PHP programming language, and an SQL database. The Google Maps API controls the display of map data on the Google Maps web platform. It provides GPS services, geolocation, routing, and direction services. The API used is based on the JavaScript programming language

The Geotree JavaScript class, is very central to the functioning of this application. It is an open source data structure built by a company named Salsita Software. Salsita Software is a software company located in Prague, Czech Republic. This company is made up of computer scientists and engineers who are experts in the latest web languages and technologies (Salsita Software, n.d.). This data structure enables the quick storage and retrieval of GPS coordinate data.

This class uses a combined implementation of a Red and Black tree data structure and a Z-order curve computation to provide these results. The Red and Black tree data structure is an implementation of a binary search tree, but each node is colored either red or black. This tree has three characteristics that make it unique. The first one is the “Ordering invariant”. Here, the keys of all nodes to the left of given parent node are smaller than the key of the parent node. Also, the keys of all nodes to the right of the same parent node are larger than the key of the parent node.

The second characteristic is the “Height invariant”. Here, the number of black nodes on every path from the root node to a leaf must be equal in number. The last characteristic is the “Color invariant”. This simply means that no two consecutive nodes are red. Insertion

and deletion of nodes require either left or right rotations to maintain these three attributes. The runtime for all operations in a red and black tree is O(log n) (Pfenning, 2010). The Z-order curve implementation performs some computations that provide each node with a key before insertion into the Red and Black tree.

The SQL database stores the permanent GPS coordinates generated by the Android device. The database titled "arsqc"**,** holds one table titled "datapoints". This table has seven columns namely: longitude, latitude, nxtlongitude, nxtlatitude, grade, routeid and path. Figure 3.2 shows an Entity Relationship diagram giving a vivid representation of the database.

*nxtlongitude- Next longitude point on a given route

*nxtlatitude- Next latitude point on a given route

*grade- The road surface quality grade (good, bad, fair)

*path- A value used to determine position along a given route (start, middle, end)

The PHP programming language holds the entire program together. It provides functions that allow both storage and retrieval of data from the SQL database. It also provides server manipulation and control.

4.3.1 Road Data Storage and Display

After the text file containing the road classification is pushed to the webserver, a PHP script is called to read data from the file into the database. The data is read line by line and inserted into the database with the following parameters: longitude, latitude, nxtLongitude, nxtLatitude, grade, routeId and path. Thus, each database entry has a record of the next GPS point along a specific route. After a file has been completely read, it is permanently deleted from the webserver.

The second aspect requires the display of the stored data in a web browser. Once the webpage is loaded, all the GPS points in the SQL database are inserted into the Geotree data structure. The parameters of the insert function in this class are: latitude, longitude, and data. The "data" variable in this instance, is all the information contained in a single record in the database (longitude, latitude, nxtlatitude, nxtlongitude, routeid, and path).

When a point is inserted into the Geotree, the latitude value is added to 90 and then multiplied by 100,000 (to ensure the value falls between the latitude range of (+/-) 90). The longitude value is added to 180 and multiplied by 100,000 (to ensure the value falls between the longitude range of (+/-) 180). These two results are now passed to the Z-curve class. This class then finds the maximum of the two values and performs bitwise operations using this maximum value to generate a unique key. This key and its associated data are now inserted into the Red and Black tree, and the appropriate rotations are performed to maintain the three attributes stated earlier on.

Once the above is done, a Google Maps interface appears on the webpage, with an input box that allows the user to insert their destination. The user's destination is converted to a GPS point via the "geolocation" feature of the Google Maps API. This data is then used to construct a rectangular boundary around the user's source and destination GPS points. Figure 4.1 on the next page shows the results, with "Berekuso" being the origin point, and "Sakumono" being the destination point.

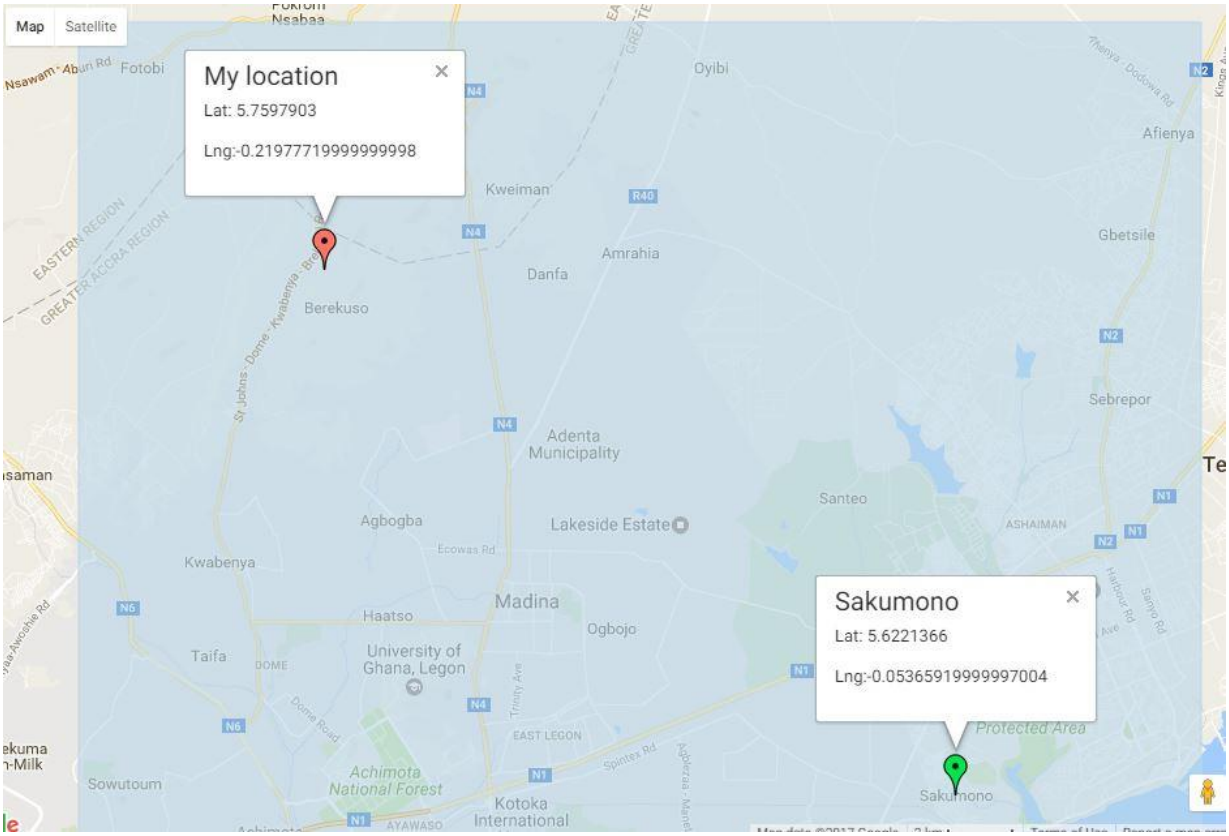


Figure 4.1: Polygon drawn boundary

Two GPS points at the corners of this rectangular boundary are passed as arguments to the "find" function of the Geotree data structure. These points must be diagonally opposite each other i.e. top right, bottom left corners or the top left, bottom right corners. In the "find" function, the maximum and minimum values of the GPS points which were passed as arguments are computed. The minimum latitude and longitude points are passed to the Z-curve class to return a start index. The maximum latitude and longitude points are also passed to the Z-curve class to return an end index. Both indexes are passed to the "find" function of the Red and Black tree which returns an array of GPS points within these boundaries. Each index in the array contains a JSON object containing data similar to that of a single record in the "arsqc" database. The data points returned are in no specific order, thus they are rearranged.

Each index in the array is considered as a specific GPS point. To draw these lines, there is the need to use a tool in the Google Maps API known as a “polyline”. A “polyline” is used to draw lines on the mapping interface given a source point and a destination point. The GPS points are rearranged based on the route id, the next GPS point, and the path (start, middle or end) of the point. These are then passed to the “polyline” function which loops through the data and draws the lines on the mapping interface. The lines drawn are colored based on the road quality between each GPS segment. Appendix D provides a sequence diagram showing the entire process. Figure 4.2 below gives a visual representation of the result. As you can see, majority of the road section is dark grey, meaning this road is relatively of poor quality.

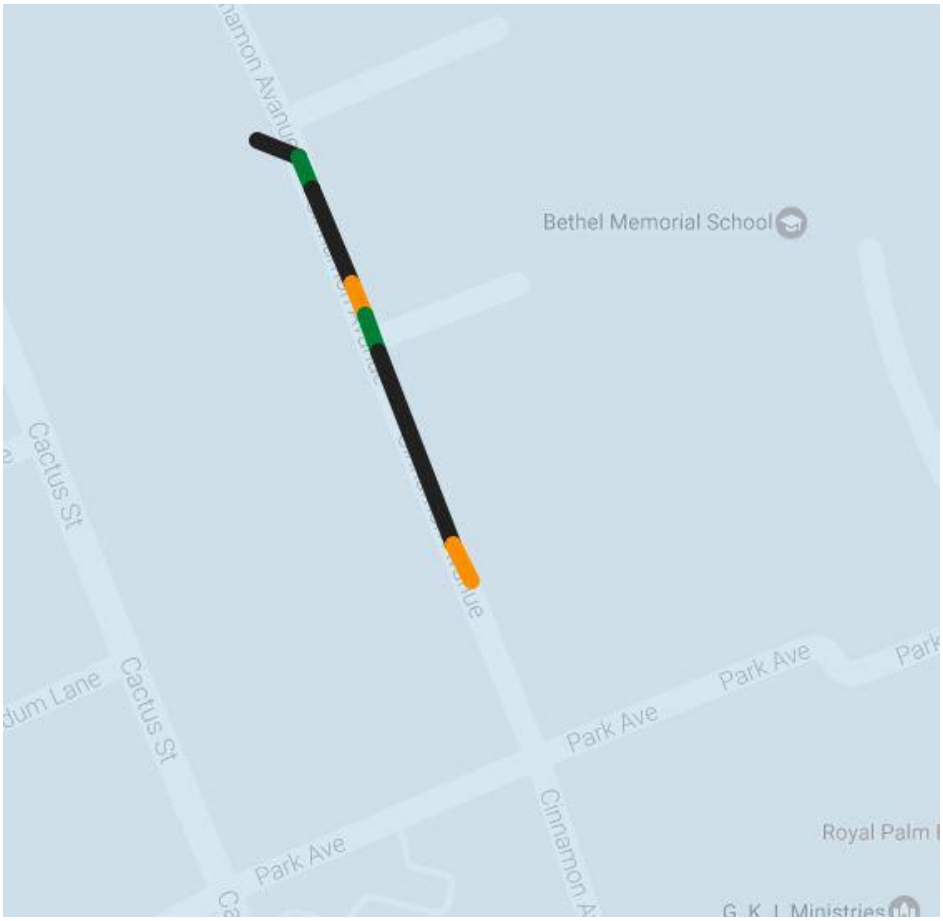


Figure 4.2: Road quality visualization

Chapter 5: Testing and Results

Testing is a requirement needed to make sure that an entire system is functioning to its utmost capacity, and to correct any errors that may occur. In this paper two types of testing procedures were conducted, component testing and user testing.

5.1 Component testing

Component tests were performed on many aspects of the application, but the most important involved testing the “arsqc” database. This is a core part of the entire project, thus needs to be as robust as possible. This test was done by trying to insert an estimated 1,400 road surface quality points into the database.

Consequently, the results showed that the MySQL configuration class threw an error message. It stated that the default timeout limit of 30 seconds for executing an SQL command had been reached, thus the process had timed out. This meant that some data points were not added to the database. Approximately 497 data points out of the estimated 1,400 were added to the database. Furthermore, it was realized that the interface used to manage the database (phpMyAdmin) started to slow down. This also caused an increase in the amount of time spent to return results from SQL queries made on the database.

The solution to this problem was to reduce the number of data points being sent to the database. It was realized that some consecutive data points contained the same GPS and road quality values. This meant that the same data points were being added to the data base multiple times. Two filters were added to solve this issue. Instead of the data points being directly added to the database, they are first filtered and then stored temporarily in an array data structure.

The first filter checks to see if the data point already exists in the temporary array. If it does, exclude that point else pass the point to the next filter. The second filter checks to see if the immediate data point has the same road quality grade (good, fair, bad) as the previous point added to the temporary array. If it does, exempt it from the array, if not add it to the temporary array. After running all the data points through these filters, the data in the temporary array is inserted into the database.

5.2 User testing

The second type of testing done was user testing. This meant giving the application to an individual to test. The application was installed on a Google Nexus tab 7 running the Android OS version 4.4.2 (API level 19). The user run the application over the weekend to collect data on the different roads they used.

Some insights gained from this test were, a huge chunk of data collection was lost due to loss in battery power. Also, after the application was started the Android device was put into the glove compartment of the moving vehicle left to record data. Upon reaching the destination, when the device was checked, its screen had timed out. The user then turned the screen on and clicked the end recording button.

Upon analysis of the user's feedback, it was realized that the device had stopped recording data as soon as the screen timed out. It was realized that whenever the screen timed out, the entire application went on a stand still. The solution was to use an Android permission component termed "Wakelock". A "Wakelock" gives the user control of the power state of the host device (Michelon, 2013). This will prevent the screen from timing out when a user is recording data.

Chapter 6: Conclusion and Recommendations

The chapter looks at a summary of the entire project. It looks at how extensive the functional requirements of the system were met, the limitations of the project as well as the future work to be done.

6.1 Summary

The goal of the project was to provide a system to automate the collection and visualization of road quality data. With regards to the core functionalities, all were accomplished. In the first chapter, it was stated that research done on crowdsourcing would be done as soon as the major aspects of the projects were completed. Unfortunately, this could not be achieved due to the amount of time spent on the core functionality.

6.2 Limitations and Challenges

One major challenge was the classification algorithm. The original classification algorithm was written using the GNU Octave platform. Trying to understand the output from the application was an initial problem, especially in the case of multi-class classification. This is because, no clear explanations were given on how to use the original classifier to classify data. The only instructions given were on how to validate the accuracy of the classifier. This made implementing the classification algorithm in the Java programming language quite tedious.

Furthermore, one limitation to note is that the web view page cannot be used on Google Chrome version 50 and above. This is because Chrome version 50 prevents the use of the Geolocation API host servers that do not possess an SSL certificate (O'Donoghue, 2016). This restriction was realized in the later part of the project when testing the web display on a mobile device. Running the application on a local server works perfectly, but

accessing it through the internet prevents the Geolocation functionality from working. Apart from Chrome, it functions properly on every other browser.

6.3 Future Work

Crowdsourcing server:

One major improvement that could be done to this system would be to create a crowd sourcing server that performs data management. This server should be able to compare new and old data of a given road segment. It should be able to decide whether to update the database with this new data, or discard it. This server could also employ techniques such as latency control, quality control, and cost control. These form a major part of crowdsourced data management (Li et al., 2016).

Classification algorithm:

This is the core of the application and needs to be improved. The optimum outcome would be to have a classifier that could provide an accuracy close to a hundred percent in all three classification areas. Different approaches such as K-means clustering or another form of regression analysis can be looked at.

Data collection:

Unfortunately, the approach to record data at this point is very rigid. The application must be placed in a fixed position, if not the data collected could be invalid. The optimum solution would be to make the positioning of the mobile device flexible and easily accessible, but still maintaining the accuracy of the data recorded.

Data retrieval:

In the implementation of this application, all the data is loaded from the database and stored in the Geotree data structure. This is costly and inefficient. This is because as the

number of data points in the SQL database increase, the length of time needed to load this data into the Geotree would greatly increase. This could affect the efficiency of the entire application. A future approach would be to build a hybrid system comprised of the Geotree data structure and the SQL database. This would cut out the time needed to load the data.

Battery power consumption:

The use of sensors such as the GPS sensor and accelerometer quickly drain the battery power of Android devices. Also, the use of the "Wakelock" mechanism has an adverse effect of draining battery power. These make the entire usage of the application quite costly. Research needs to be done to find methods to reduce the amount of battery power consumed by the application

Application sustainability:

For this application to be of great use to drivers in developing countries, there is the need to ensure its sustainability. The application will only benefit individuals if there is a constant supply of data from various sources. In Ghana, some ideas for this include selling this application to companies that might benefit greatly from its use. Examples include Easy Taxi and Uber Ghana. Another idea would be to market this idea to the Ministry of Transport. This could serve as a tool in monitoring the condition of roads in the country.

6.4 Conclusion

Altogether, this project sets the foundation for the easy visualization of road quality data. It contains a clean interactive interface, which makes it simple to use and understand. Despite not being market ready, it is a step in the right direction to aiding drivers reduce the cost involved in maintaining their vehicles, and reduce the probability of the occurrence of road accidents.

References

Alessandroni, G., Klopfenstein, L., Delpriori, S., Dromedari, M., Luchetti, G., Paolini, B., ... & Bogliolo, A. (2014). Smartroadsense: Collaborative road surface condition monitoring. *Proc. of UBICOMM-2014. IARIA.*

Bell, K. (2016). *Apple just revealed the most-downloaded app of 2016*. *Mashable*. Retrieved from http://mashable.com/2016/12/06/most-downloaded-apps-2016/#5EuC5Qjne5qh

Buczkowski, A. (2015, October 23). *Why would you use OpenStreetMap if there is Google Maps?* Retrieved from Geoawesomeness: http://geoawesomeness.com/why-would-you-use-openstreetmap-if-there-is-google-maps/

Botchway, K. (2016). *Road accidents down by 17% in 2015*. Retrieved from http://citifmonline.com/2016/01/01/road-accidents-down-by-17-in-2015-infographic/

del Rosario, M. B., Redmond, S. J., & Lovell, N. H. (2015). Tracking the evolution of smartphone sensing for monitoring human movement. *Sensors*,*15*(8), 18901-18933.

Doku, A. (2014). *Embedding information about road surface quality into Google maps to improve navigation.* Ashesi University College

Forslöf, L., & Jones, H. (2013, June). Roadroid: continuous road condition monitoring with smart phones. In *IRF 17th World Meeting and Exhibition, Riyadh, Saudi Arabia. Available from http://www. roadroid. com/common/References/IRF* (Vol. 202013).

Gertler, P. J., Gonzalez-Navarro, M., Gracner, T., & Rothenberg, A. D. (2016). *Road Quality, Local Economic Activity, and Welfare: Evidence from Indonesia's Highways.* UC Berkeley: Center for Effective Global Action. Retrieved from: https://escholarship.org/uc/item/0vs9p5mb

Goetz, M., Lauer, J., & Auer, M. (2012, January). An algorithm based methodology for the creation of a regularly updated global online map derived from volunteered geographic information. In *Proceedings of the Fourth International Conference on Advanced Geographic Information Systems, Applications, and Services, Valencia, Spain* (Vol. 30, pp. 50-58).

Goodrich, R. (2013). *What is crowdsourcing?* Retrieved from http://www.businessnewsdaily.com/4025-what-is-crowdsourcing.html

Google Inc. (n.d.). *View traffic, satellite, terrain, biking, and transit*. Retrieved from https://support.google.com/maps/answer/3092439?co=GENIE.Platform%3DDesktop&hl=en&oco=1

Innovations Inc. (n.d.). *Linear acceleration in smartphones and tablets*. Retrieved from Rotoview: http://www.rotoview.com/linear_acceleration_sensor.htm

Kanhere, S. S. (2013, February). Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In *International Conference on Distributed Computing and Internet Technology* (pp. 19-26). Springer Berlin Heidelberg.

Karamihas, S. M., & Sayers, M. W. (1996). *Interpretation of Road Roughness Profile Data*. FHWA/RD-96/101. FHWA, US Department of Transportation, Washington, DC.

Lamptey, E. (2016). *Regulatory Body for Road Transport: …a necessary policy prescription*. Retrieved from http://thebftonline.com/features/opinions/16831/regulatory-body-for-road-transport-a-necessary-policy-prescription.html

Lauer, J., Billen, N., & Zipf, A. (2013, November). Processing crowd sourced sensor data: from data acquisition to application. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science* (p. 43). ACM.

Li, G., Wang, J., Zheng, Y., & Franklin, M. J. (2016). Crowdsourced data management: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *28*(9), 2296-2319.

Li, M., & Bailiang, Z. (2016). *Discover the action around you with the updated Google Maps*. *Google*. Retrieved from https://blog.google/products/maps/discover-action-around-you-with-updated/

Michelon, P. (2013). *Wakelocks: Detect No-Sleep Issues in Android* Applications*. Retrieved from https://software.intel.com/en-us/android/articles/wakelocks-detect-no-sleep-issues-in-android-applications

O'Donoghue, R. (2016). *No HTTPS? Then say goodbye to geolocation in Chrome 50!* Retrieved from https://mobiforge.com/news-comment/no-https-then-bye-bye-geolocation-in-chrome-50

Paterson, W. (1986). International roughness index: Relationship to other measures of roughness and riding quality. *Transportation Research Record*, (1084).

Pfenning, F. (2010, October 21). Retrieved from Carnegie Mellon University School of Computer Science: https://www.cs.cmu.edu/~fp/courses/15122-f10/lectures/17-rbtrees.pdf

Pilgrim, M. (2010). *Geolocation - Dive into HTML5*. Retrieved from http://diveinto.html5doctor.com/geolocation.html

Poushter, J. (2016). S*martphone ownership and internet usage continues to climb in emerging economies*. Retrieved from http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/

Rosenberg, M. (2017). *The Role of Colors on Maps*. Retrieved from https://www.thoughtco.com/colors-on-maps-1435690

Rouse, M. (2013). *What is spatial data?* Retrieved from http://searchsqlserver.techtarget.com/definition/spatial-data

Salsita Software. (n.d.). *Get to know Salsita*. Retrieved from Salsita Software: https://www.salsitasoft.com/about-us

Sommerville, I. (2011). *Software engineering* (9th ed.). Boston: Pearson.

Vorgbe, F. D. (2014). *Classification of road surface quality using android smartphone devices.* Ashesi University College

Vittorio, A., Rosolino, V., Teresa, I., Vittoria, C. M., & Vincenzo, P. G. (2014). Automated sensing system for monitoring of road surface quality by mobile devices. *Procedia-Social and Behavioral Sciences, 111*, 242-251.

Appendices

Appendix A: Code Snippet of Coordinate Class

```
/*
* @author: Kwabena Boohene
* @date:03/03/2017
* Contains functions to move classified data to sql database
*/
include_once("adb.php");
include_once("datapoint.php");

class coordinates extends adb{
      //Array containing existent grades
      var $GradeList=array( );
      //Counter object
      var $place=-1;
      //Array containing GPS values
      var $GPSvalues=array( );
  //Constructor
  function coordinates( ){
  }

  /*
  * @params $dataline- Line in a text file
  * Splits lines in a text file into array format
  * @return True or False after insert into database
  */
  function splitLine($dataLine){
    $textArray = explode(",",$dataLine);
    return $textArray;
  }
```

```
  /*
  * @params $grade, $longitude, $latitude
  * @return query result, True or False
  * Inserts data into database
  */
function addCoordinate(
$grade,$longitude,$latitude,$RouteID,$status,$NxtLong=false,$NxtLa
      t=false){
if($NxtLong==false){
 $strQuery="INSERT into DataPoints SET GRADE='$grade',
      LONGITUDE='$longitude',
      LATITUDE='$latitude', ROUTEID='$RouteID',
      POSITION='$status'";}
else{ $strQuery="INSERT into DataPoints SET GRADE='$grade',
      LONGITUDE='$longitude',
      LATITUDE='$latitude', NXTLONGITUDE='$NxtLong',
      NXTLATITUDE='$NxtLat',
      ROUTEID='$RouteID', POSITION='$status'";}
      return $this->query($strQuery);
  }
 $strQuery="SELECT * FROM datapoints ORDER BY position DESC LIMIT
      1";
       return $this->query($strQuery);
}

//Updates latitude and longitude points
function updateNxt($nxtLat,$nxtLng,$pointId){
$strQuery="UPDATE datapoints SET nxtLatitude = '$nxtlat',
      nxtLongitude = '$nxtLng' WHERE pointId='$pointId'";
return $this->query($strQuery);
      }
}
  /*
  * @params road quality file
  * @return nothing
  * Reads the lines of a text file
  */
```

```
//Reads file and does appropriate checks
function readFile($dataFile){
      $RouteID=sha1(basename($dataFile));
      $RouteID = substr($RouteID, 0, 5);
      $this->checkLast();
      $check=$this->fetch();
      /*if($check['position']==2){
            $checkFile = fopen($dataFile,"r");
            $checkLine = fgets($checkFile);
            $fLine = $this->splitLine($checkLine);
            $newPoint = new
      datapoints($fLine[1],$fLine[2],$fline[0]);
            $Exist=$this->pointExists($newPoint);
            if($Exist==false){
                  $this-
      >updateNxt($fLine[2],$fLine[1],$check['pointId']);
                  array_push($this->GPSvalues,$newPoint); }
                  fclose($checkFile);
            }*/

      $tempFile = fopen($dataFile, "r");
      if ($tempFile) {
            while(($line = fgets($tempFile)) !== false){
                  //Gets current GPS points
                  $currentPoint=ftell($tempFile);
                  $textArray = $this->splitLine($line);
                  $nxtLine =fgets($tempFile);
                  $nxtArray=$this->splitLine($nxtLine);
                  if($nxtLine != false){
                  //Makes sure duplicate points aren't added
                  if($nxtArray[1]!=$textArray[1]){
                  $startPoint = new
      datapoints($textArray[1],$textArray[2],$textArray[0]);

                        //Check if the grade point has already
      been stored
```

```
                $verify = $this->pointExists($startPoint);

                if($verify!=true){
                //Checks grade point
                $veriGrade=$this->shortenPath($startPoint);}
                else{$veriGrade=false;}

                if($veriGrade==true){
                array_push($this->GPSvalues,$startPoint); }
                }
}
                else{
                $point = new
      datapoints($textArray[1],$textArray[2],$textArray[0]);
                //Check if point has already been stored
                $verify = $this->pointExists($point);
                      if($verify!=true){
                            //Checks grade point
                            $veriGrade=$this-
      >shortenPath($point);}
                           else{$veriGrade=false;}
                           if($veriGrade==true){
                                 array_push($this-
      >GPSvalues,$point); }
                      }
                      fseek($tempFile,$currentPoint,SEEK_SET);
                }
                fclose($tempFile);
           } else {
                // error opening the file.
           }
      }
//Prevents sequential GPS points of the same grade
//being added to the database
function shortenPath($point){
```

```
      if($this->place==-1){
      array_push($this->GradeList,$point->getGrade());
            $this->place++;
            return true;
            }
            else{
                  if($point->getGrade()!=$this->GradeList[$this-
      >place]){
                        array_push($this->GradeList,$point-
      >getGrade());
                       $this->place++;
                       return true;}
                  else{return false; }
                       }
        }

//Check if point has been stored already
function pointExists($datapoint){
     $validPoint=false;
      for($i=0;$i<sizeof($this->GPSvalues);$i++){
             if($datapoint->getLng()==$this->GPSvalues[$i]-
      >getLng()){
            $validPoint=true;
              break; }
            else{$validPoint=false; }
      }
      if(sizeof($this->GPSvalues)==0)
         $validPoint=false;
      return $validPoint;
}

//Adds data to database
function insertData($RouteID){
       for($i=0;$i<sizeof($this->GPSvalues);$i++){
         if($i+1<sizeof($this->GPSvalues)){
```

```
            $this->addCoordinate($this->GPSvalues[$i]->getGrade(),
                    $this->GPSvalues[$i]->getLng(), $this-
      >GPSvalues[$i]->getLat(), $RouteID,2,
                  $this->GPSvalues[$i+1]->getLng(), $this-
      >GPSvalues[$i+1]->getLat() );
      }
      else{
      $this->addCoordinate($this->GPSvalues[$i]->getGrade(),
      $this->GPSvalues[$i]->getLng(),
       $this->GPSvalues[$i]->getLat(), $RouteID,2);
      }
       }
}

  /*
  * @params nothing
  * @return all the gps data stored in the database
  */
  function fetchAllData(){
    $strQuery="Select grade, Longitude, Latitude, nxtLongitude,
     nxtLatitude, routeId, position from datapoints";
    return $this->query($strQuery);
  }

//Checks the last entry into the database
function checkLast(){
      $strQuery="SELECT pointId, position FROM datapoints ORDER BY
      position DESC LIMIT 1";
      return $this->query($strQuery);
}

//Updates null entries in the database
function updateNxt($nxtLat,$nxtLng,$pointId){
      $strQuery="UPDATE datapoints SET nxtLatitude = '$nxtLat',
      nxtLongitude = '$nxtLng' WHERE pointId='$pointId'";
      return $this->query($strQuery);}
```

Appendix B: Code snippet of Feature Extractor class

```
public class FeatureExtractor{
    private BufferedReader br;
    private String currentLine;
    private int numWindows;
    private String fileName="";
    private DataCrawler crawler = new DataCrawler();
    private DataPoints testDataPoints = new DataPoints();
    private boolean start=false, end=false;

    public FeatureExtractor(String name){
        fileName=name;
    }

    // Gets file location to classify
    File sdcard= Environment.getExternalStorageDirectory();
    File directory = new
     File(sdcard.getAbsolutePath()+"/ARSQC/rawData");
    String testDataInputFile;
    FeatureComputer testDataComputer = new
     FeatureComputer(testDataPoints);

    //Computes the number of windows to be used
    public void getNumWindows (){
        int count= 0;
        int requiredLines=0;
        try {
            testDataInputFile = new File(directory,fileName)+"";
            br = new BufferedReader(new
     FileReader(testDataInputFile));
            while ((currentLine = br.readLine()) != null) {

     if((currentLine.contains("Start")!=true)||(currentLine.conta
     ins("End")!=true))
                {
```

```
                    count++;
                }
                if(currentLine.contains("Start")==true){
                    start=true;
                    System.out.println("Start exists");
                }
                else if(currentLine.contains("End")==true){
                    System.out.println("End exists");
                    end=true;
                }
                if(count%4==0){
                    requiredLines=count;
                }
                }
            br.close();
        }catch (IOException e) {
           e.printStackTrace();
        }
        //Check integer division
        int numSecs = requiredLines/4;
        numWindows=numSecs/10;
    }

//Extracts data into multidimensional array
    public double [][] extract (){
        getNumWindows();
        crawler.read(testDataPoints,testDataInputFile);
        testDataComputer.computeLong();

        double [][] features = new double[numWindows][8];
        for(int i=0;i<numWindows;i++){
            testDataComputer.window();
            features[i][0]=testDataComputer.computeZMean();
            features[i][1]=testDataComputer.computeZVariance();
```

```
features[i][2]=testDataComputer.computeZStandardDeviation();
        features[i][3]=testDataComputer.highestZPeak();
        features[i][4]= testDataComputer.lowestZTrough();
        features[i][5]=testDataComputer.computeLong().get(i);
        features[i][6]=testDataComputer.computeLat().get(i);
        if(i==0){
            if(start==true){
                System.out.println("Start Exists");
                features[i][7]=1;
            }
            else{features[i][7]=2;}
        }
        else if(i==(numWindows-1)){
            if(end==true) {
                features[i][7] = 3;
                System.out.println("End exists");
            }
        }
        else {
            features[i][7]=2;
        }
    }
    //Deletes raw file data
    File rawfile = new File(directory,fileName);
    if(rawfile.exists()){
        rawfile.delete();
        System.out.println("File deleted");
    }
    else{
        System.out.println("File not deleted");
    }       return features;}}
```

Appendix C: Code Snippet of Classifier Class

```
public class ClassifierService implements Runnable {
    private double [][] exFeatures;
    private PrintWriter printer;
    private String roadGrade, filename;
    private double class1,class2;
    private FeatureExtractor extractor;
//Constructor
    public ClassifierService(String name){
        filename=name;
    }

    //Runs as soon as the thread starts
    public void run() {
        extractor = new FeatureExtractor(filename);
        runClassification();
    }
    // Good vs Bad/Fair theta values
    public double classify1(double val1, double z_mean, double
      z_var,
                            double z_D, double z_peak, double
      z_trough){

        double theta1, theta2, theta3, theta4, theta5, theta6;
        theta1=38.388669;
        theta2=-1.658378;
        theta3=4.517424;
        theta4=-23.416034;
        theta5=0.305482;
        theta6=-1.711021;

        double result =
      (val1*theta1)+(z_mean*theta2)+(z_var*theta3)
                  +(z_D*theta4)+(z_peak*theta5)+(z_trough*theta6);
```

```
    return result;
}

//Bad vs Good/Fair theta values
public double classify2(double val1, double z_mean, double
  z_var,
                        double z_D, double z_peak, double
  z_trough){

    double theta1, theta2, theta3, theta4, theta5, theta6;
    theta1=-0.641961;
    theta2=-1.009224;
    theta3=-3.973086;
    theta4=17.536464;
    theta5=-0.565364;
    theta6=0.681880;

    double result =
  (val1*theta1)+(z_mean*theta2)+(z_var*theta3)
            +(z_D*theta4)+(z_peak*theta5)+(z_trough*theta6);
    return result;
}

//Determines the grade of road
public String grade(double classify1, double classify2){
    String grade=" ";
    if((classify1<0.8)&&(classify2<0.8)){
        grade="Good";
    }
    else if((classify1>=0.8)&&(classify2>=0.8)){
        grade="Bad";
    }
    else{
        grade="Fair";
    }
```

```
    return grade;
}

//Sigmoid function of probability
public double sigmoid(double prob){
    prob = (1/(1+Math.pow(Math.E,(-1*prob))));
    return prob;
}

//Writes grade along with required parameters to a text file
public void runClassification(){
    exFeatures = extractor.extract();
    Boolean checkExternal = canWriteOnExternalStorage();
    if(checkExternal==true) {
        File sdcard =
 Environment.getExternalStorageDirectory();
        File dir = new File(sdcard.getAbsolutePath() +
 "/ARSQC/Classification");


        try {
            if (!dir.exists()) {
                dir.mkdirs();
            }
            File file = new File(dir, filename);
            if (!file.exists()) {
                file.createNewFile();
            }
            printer = new PrintWriter(new BufferedWriter(new
 FileWriter(file, true)));
            for (int i = 0; i < exFeatures.length; i++) {
                class1 = sigmoid(classify1(1,
 exFeatures[i][0], exFeatures[i][1],
                        exFeatures[i][2], exFeatures[i][3],
 exFeatures[i][4]));
```

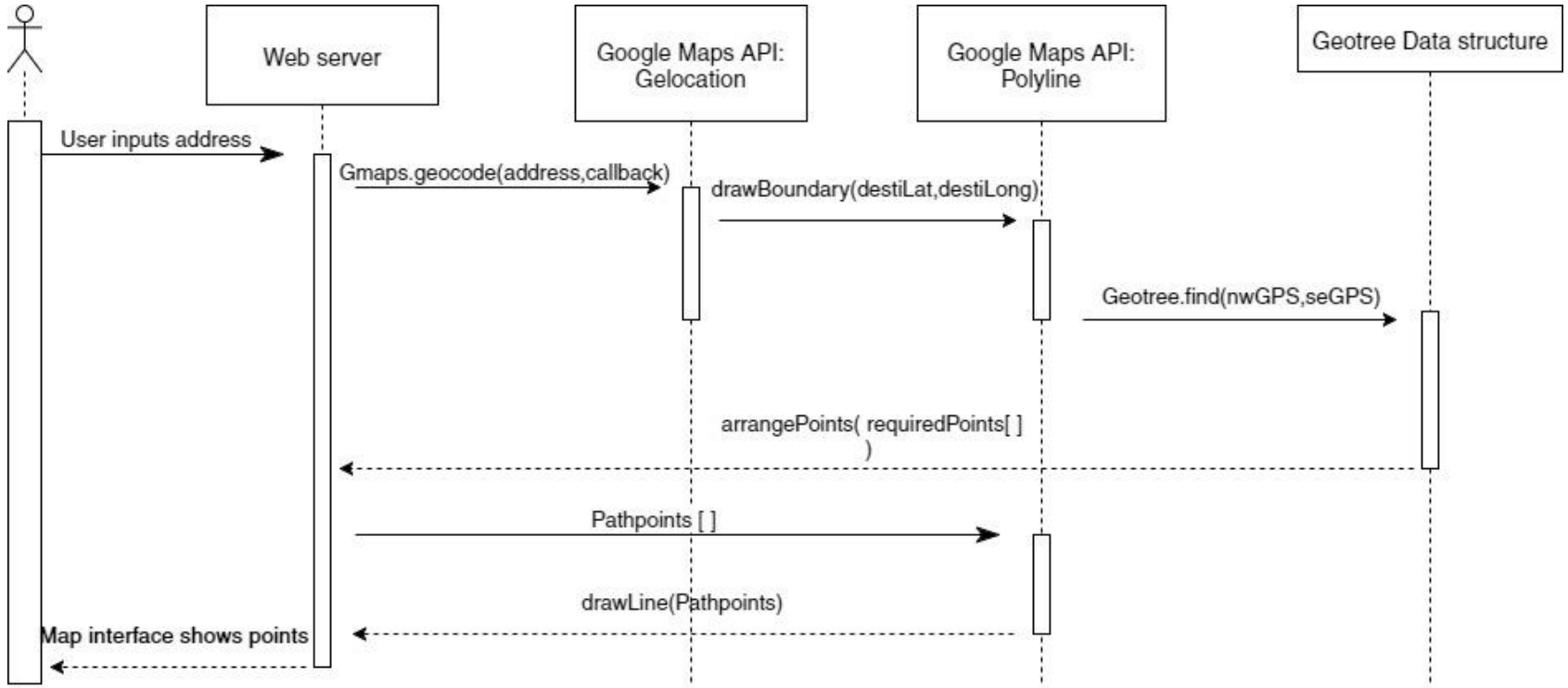
```
                class2 = sigmoid(classify2(1,
    exFeatures[i][0], exFeatures[i][1],
                        exFeatures[i][2], exFeatures[i][3],
    exFeatures[i][4]));


                roadGrade = grade(class1, class2);
                printer.println(roadGrade + "," +
    exFeatures[i][5] + "," + exFeatures[i][6]+

    ","+class1+","+class2+","+exFeatures[i][7]);
            }
            printer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Appendix D: Sequence Diagrams of Data Classification and Data Display

Sequence Diagram of Data Display



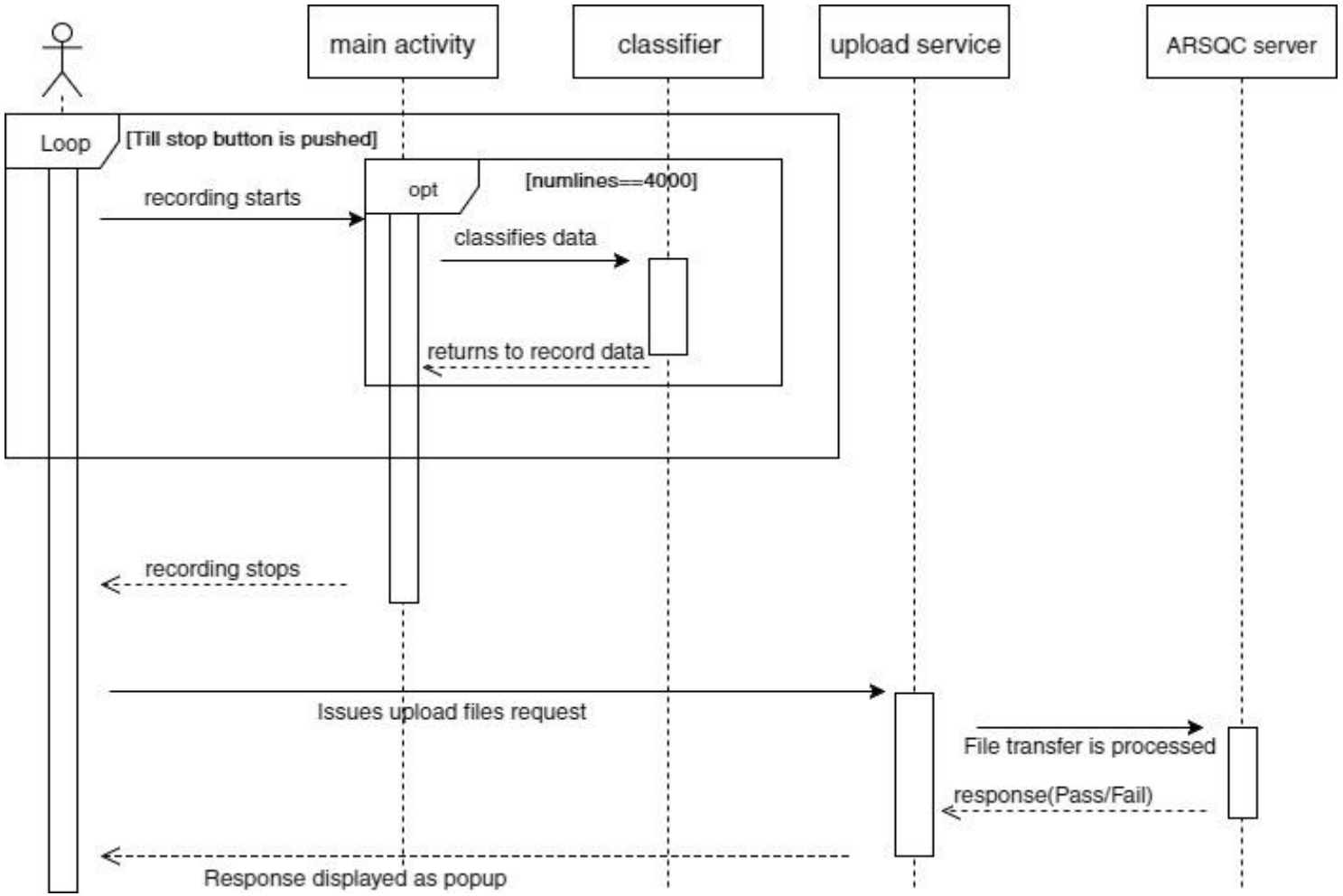
Address: Location typed in by user

Callback: function run after the running function is completed

requiredPoints[ ]: Array containing GPS points within square boundary

Pathpoints [ ]: Final GPS points to be drawn on map interface

Sequence Diagram of Data Collection and Classification



Appendix E: Web View