MEASUREMENT OF ROAD PAVEMENT SURFACE UNDULATIONS USING A LOW-COST ACCELEROMETER SENSOR

A.W.C. Chamikara

(168325U)

Degree of M.Eng in Highway & Traffic Engineering

Transport Division, Department of Civil Engineering

University of Moratuwa

Sri Lanka

July 2020

MEASUREMENT OF ROAD PAVEMENT SURFACE UNDULATIONS USING A LOW-COST ACCELEROMETER SENSOR

A.W.C. Chamikara

(168325U)

Degree of M.Eng in Highway & Traffic Engineering

Transport Division, Department of Civil Engineering

University of Moratuwa

Sri Lanka

July 2020

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

The above candidate has carried out research for the Masters/MPhil/PhD thesis/ Dissertation under my supervision.

Name of the Supervisor: Dr. H.R. Pasindu

Signature of the supervisor: Date:

ABSTRACT

Pavement roughness measurement is one of the four parameters of measuring the pavement condition evaluation, i.e., Pavement roughness measurement, distress condition evaluation, skid resistance, and structural capacity evaluation.

This research aims to improve IRI measurement accuracy by smartphone method using a low cost, off the shelf accelerometer without compromising the cost aspect. This method collects data from an accelerometer fixed to a vehicle axel. Since the vehicle's shock absorbers do not damp the measurement, the readings are much more realistic. Data is then forwards to a machine-learning algorithm to analyze the collected data and predict the road condition. This algorithm should be trained using a training data set before using, which involves collecting and labelling data according to prior knowledge and previously collected data. The training was done by collecting data using a smartphone application and manually marking the data points. Then this data was separated as training and testing data as appropriate, and training data was fed into the algorithm with the manually labelled data as a reference. After training the algorithm, the testing dataset was provided to the model to measure the accuracy.

The second part of the research was carried out to train the algorithm on detecting potholes without human involvement. For this, the data collection application was slightly modified to label the pothole data points. Then the previous training and the testing method were carried out.

Accurate results were observed during both instances regarding the labelled data. It was found that more training data makes the prediction model more accurate.

Since this is a low-cost method to determine the road surface condition, local road authorities can implement this as a network to collect real-time data and carry out future road maintenance works effectively.

ACKNOWLEDGMENT

IRI data for this research was obtained from the Road Development Authority, Sri Lanka, and Mr. Chamikara R., who supported android application development. And Dr. H.R. Pasindu and staff of Transport Division, Department of Civil Engineering, University of Moratuwa.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Road roughness is a crucial parameter that used in road maintenance planning and managing works. The International Roughness Index (IRI) considered the primary measurement related to road roughness. IRI value measured using the quarter- car math model with the units of meters per kilometre. The main drawback of this method is the cost associated with the measurement accuracy and data collection speed. Highly accurate IRI measuring devices like laser-based vehicle-mounted devices are more reliable than the Bump Integrator Devices, which generate less accurate measurements. This research uses an accelerometer device to measure the road surface undulations and uses that data to predict the actual IRI value with machine learning assistance.

An accelerometer is a device that can identify the intensity of physical acceleration (or the rate of change of velocity) experienced by an object. A common multi-axis digital accelerometer can measure the linear acceleration in 3 axes (x, y, and z-direction) and the rotational acceleration around the three axes. Most commonly, acceleration measured by the gravitational force equivalent units known as the g-force. Therefore, an accelerometer resting on the Earth's surface indicates a 1g-force value. Today, digital accelerometers can be purchased off the shelf anywhere in the world for an affordable cost. Integrated accelerometers in smartphones are generally less accurate and have lower data output rates.

With a smartphone application, collected accelerometer data is combined with the GPS location data from the smartphone. GPS data used to calculate the speed and position of the data collection vehicle. Then the dataset is processed using a machine learning algorithm." Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence" (Samuel, 1959). There are many branches of machine learning used in the world today. A classification based machine-learning algorithm used in this research to classify accelerometer data into IRI values.

This research's main objective is to improve the indirect IRI measuring methods' accuracy, such as the smartphone accelerometer-based approach while maintaining cost-effectiveness. An external accelerometer was used in this research instead of a smartphone's built-in accelerometer to improve the test's accuracy. A machine-learning algorithm was used to improve the result interpretation accuracy. All the components and tools used in this method are selected to minimize the cost.

The scope of this proposed study includes the following components,

- Study the feasibility of using an external accelerometer and machine learning with a smartphone to improve the data collection accuracy.
- Evaluate the possibility of identifying the road distresses such as potholes and road bumps using the accelerometer without any human involvement and
- Evaluate the accuracy of IRI values' prediction compared to commercially available solutions such as laser-based IRI measuring devices.

.

2 LITERATURE REVIEW

2.1 Early Adoption of IRI

The *International Roughness Index (IRI)* is an essential parameter for pavement management and decision-making in the road development sector. Before having a standard measurement system, different parts of the world have been using various methods to assess the pavement condition. In 1982, the World Bank initiated the *International Road Roughness Experiment (IRRE)* to establish a standard measurement unit. As a result, it was found that different roughness measurements were not correlated with one another, partially due to how the measuring instruments respond to the road profile and partly due to how the data is processed (Paterson, 1986). Hence it was suggested that to have a standard measurement scale to improve the reliability of exchanging information related to road roughness.

(Bennett, 2006) has identified five *Information Quality Levels* in road surface undulation measurement. The decision-making priority decreased with the information quality level from having the highest accuracy of the collected data in level 1. The main concern in data collection is the repeatability of the data that was collected. A research study in 2006 using 68 various road surface profiles found that the accuracy varies significantly even among the same types of devices, and none of the profilers that evaluated has met all the current IRI bias standard requirements (Wang, 2006). That means it is hard to cross-compare the data collected from one device with a different kind of device.

2.2 Road Roughness Measuring Devices

There are several devices available in the market for roughness data collection. The following were found as the most common devices.

2.2.1 Bump Integrator

There are two main types of bump integrators available in the market. The Central Road Research Institute (India) developed the fifth wheel bump integrator (Fifth Wheel Bump Integrator, 2012). The bump integrator towed using a vehicle at 32 kmph speed to take the measurements. The other type is the vehicle-mounted bump

Figure 2.1: Bump Integrator

integrator, which has two separate sensors for calculating bumps and distance travelled. This instrument also has a data collection speed of 32 kmph.

- Strength - Fast, simple, and reliable in taking measurements. Less human involvement is needed. No in-depth knowledge of the subject is necessary.
- Limitations - Both have a data collection speed limitation of 32 kmph. Also, vehicle vibrations can be an issue for these devices.

Figure 2.2: Rolling Straight Edge

2.2.2 Rolling Straight Edge

A 3-meter width structure supported by rubber tires, which has a similar tire in the middle of the device, can move freely in a vertical direction. The intensity of bumps is recorded in a graph sheet with the distance measurement. Rolling straight edge is a push-on type device. So, the typical data collection speed is 1-2 kmph.

- Strength - The accuracy of this device is very high. The reproducibility of this device is excellent.
- Limitations - 3m distance of this device is a limitation. Also, it is not directly related to the road roughness that road users are experiencing in the real world.

2.2.3 Road Surface Profiler (Laser-based)

This equipment is known as the high-speed profiler. The road profile is digitally measured using a combination of laser and accelerometer sensors. This device is capable of collecting data at a speed of 100 kmph. However, typically, data is collected at 80 kmph as a rule of thumb.

- Strength - Data collection speed and accuracy are the main plus points of this device. Also, the data is collected automatically, and little human involvement is necessary.
- Limitations - High cost associated with the device is the main limitation. Because of that, developing countries like Sri Lanka are having trouble accessing modern decision-making tools. Also, to operate this machine, it must have high operating skills.

2.2.4 Walking Profiler

Walking profiler is a high precision instrument when it comes to road roughness measurement. Usually, an inclinometer sensor fixed between 2 supported wheels is used to measure the road surface undulations. Since it is a push-on instrument, it has a data collecting speed denoted as 800 meters per hour.

- Strength - High accuracy of the data is the main strength of this device

- Limitations - Low data collection speed and skills needed to collect the data is the main limitation of this device.

Figure 2.3: Walking Profiler

2.2.5 Smartphone-Based Measurements

There are many smartphone applications available for both Android and iOS operating systems to measure road roughness. (Abeywardana, Abeywikrama, Amarasinghe, & Kumarasinghe, 2018) Even homegrown applications are available, which are more polished for the Local conditions. Smartphone-based measurement is typically carried out using the inbuilt accelerometers in the smartphone and converting it into an IRI value. In the data collection, the phone must be rigidly fixed to the vehicle to measure accelerometer data accurately. Data collection speed can be varied since the data could be normalized with the speed data collected using an in-built GPS sensor. The relationship between pavement distress and IRI is often expressed as the root mean square value of the accelerometer data. (Firoozi, Mahmoudzadeh, Azizpour, & {others}, 2017) Indicate that the IRI = 4.19RMS + 1.73, where RMS is the root mean square value of acceleration data. (Sandamal & Pasindu, Applicability of smartphone-based roughness data for rural road pavement condition evaluation, 2020) has formed a relationship with IRI and pavement surface distresses such as ravelling, cracking, and edge gap. They have introduced two submodels to account for the ageing of the pavement. Submodel 1, IRI = 2.538 + 0.095RAV + 1.545EDG + 1.158PAT .

Submodel 2, IRI = 6.135 + 0.107RAV + 11.353POT + 0.25CRA. (RAV - Raveling as a percentage, EDG - percentage of the edge gap more than 10 cm in linear length along both side of the pavement, PAT- Patch area as a percentage, CRA - Cracking area as a percentage, POT - Pothole area as a percentage)

- Strength – Data collection speed can be varied. Anyone who has a decent smartphone can access this technology.
- Limitations - Reproducibility is questionable since measurements can depend on the smartphone model, vehicle type, and speed.

2.3 Sri Lanka in Road Roughness Measurement

Even though Sri Lanka is a middle-income country, the road density of Sri Lanka is higher when compared with similar countries. So, road maintenance is one of the challenges that Sri Lanka is facing right now. The high cost associated with road maintenance and high road density makes it crucial to identify the best road rehabilitation period and level. Because of this, Sri Lanka has looked into the IRI-based road maintenance prediction models. Currently (as of 2019), the Road Development Authority has one laser-based road profiler to cover more than 12,000 kilometres of A and B class roads and about 160 kilometres of expressways.

A research study was carried out to adopt smartphone-based road roughness measurement for low-volume roads in Sri Lanka (Gamage, Pasindu, & Bandara, 2016). Moreover, a methodology for road maintenance planning for low-volume roads was proposed based on the smartphone-based roughness measurement data (Silva & Pasindu, 2017). An undergraduate research study (Abeywardana, Abeywikrama, Amarasinghe, & Kumarasinghe, 2018) conducted extensive research on smartphone-based roughness measurement data collection.

2.4 Using Accelerometer in Road Roughness Measurement

There are many methods proposed to measure road roughness over the past years. However, the use of an accelerometer seems to be highly popular mainly due to the availability and the low cost. Smartphone-based accelerometers are commonly used to determine road roughness. *AndroSensor* is one of the applications that use the inbuilt accelerometer sensor in a smartphone to assess the road roughness (Douangphachanh & Oneyama, 2013). Similar studies have been carried out in Sri Lanka to evaluate

smartphone-based roughness data collections' usability in Sri Lankan conditions (Abeywardana, Abeywikrama, Amarasinghe, & Kumarasinghe, 2018) (Gamage, Pasindu, & Bandara, 2016). Nevertheless, using an external accelerometer to determine the road roughness was not evident in this literature review.

2.5 Use of Machine Learning Algorithms

Even though it is not very clear when the first idea of a machine learning algorithm was introduced in History, the name '*Machine Learning'* was surfaced in 1959 by Arthur Samuel (Samuel, 1959). In his study, he was researching to develop a digital counterpart for a professional checker player. The traditional approach to this problem was to use a decision tree for each move that can play throughout the game. In a game of checkers, there are $5x10^{20}$ possible search spaces. Because of this enormous possibility number, the traditional path was too much for computers in 1959. Therefore, he proposed two methods, namely the reward-based approach and the supervised learning method. A model is required to perform a machine learning task. In this research, more attention was given to artificial neural network models.

Also, there are three main methods available to train these models to predict future outcomes.

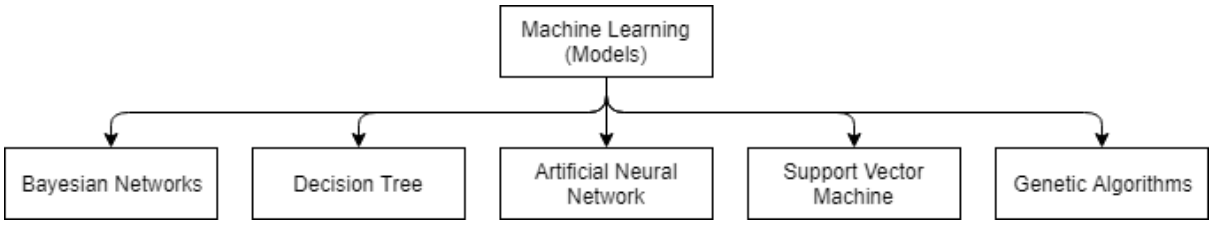

Figure 2.4: Machine Learning Models

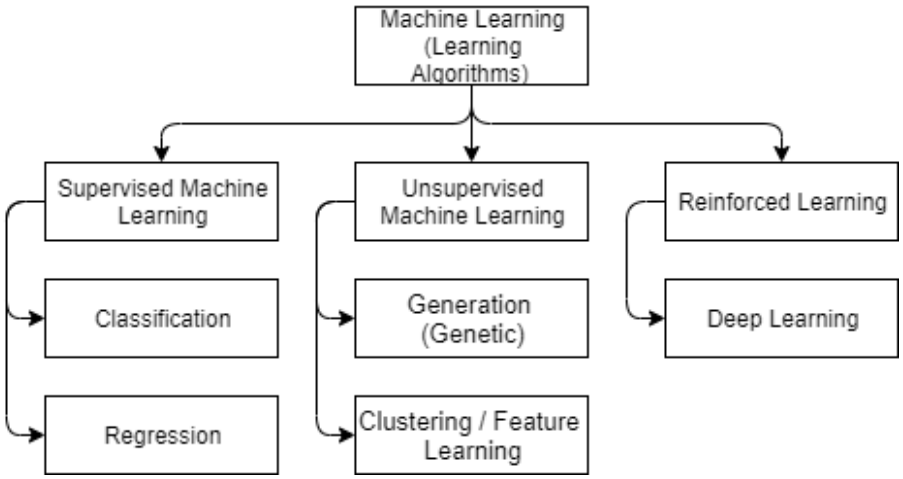

Figure 2.5: Machine Learning Algorithms

2.5.1 Artificial Neural Network

Artificial neural networks are similar to the biological counterparts consisting of inputs and outputs. An artificial neural network is an interconnected group of nodes with inputs and outputs. Each link between nodes represents a weightage and a bias. The initial data is multiplied by the weightage. Then the bias is added. This whole number goes through an activation function.

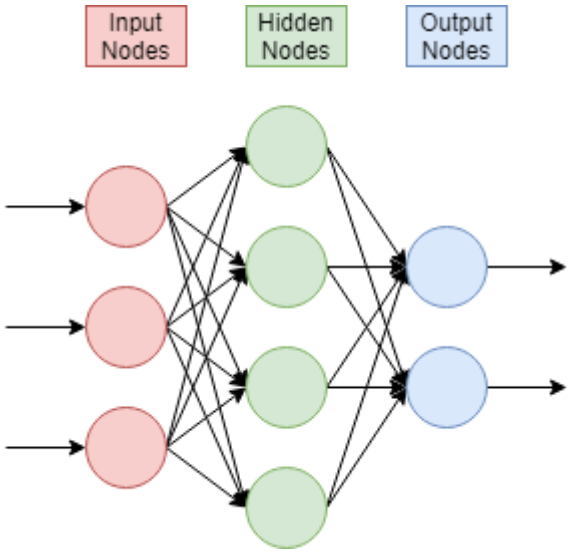

Figure 2.6: Neural Network Layers

2.5.2 Supervised Machine Learning

Supervised machine learning is carried out by training an algorithm with previously labelled data or with input data that the outcome is known. For example, if we take a

population with known age, gender, weight, and height, we might be able to train an algorithm to predict an individual's height by inputting the age, gender, and weight.

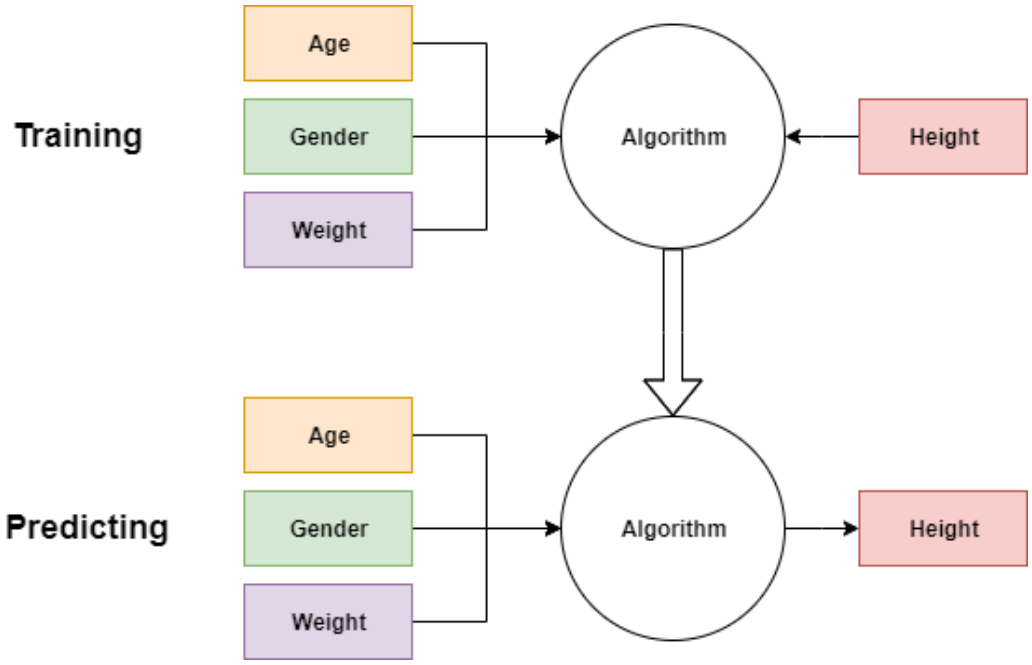

Figure 2.7: Supervised Learning Example

Recent breakthroughs of these supervised machine learning algorithms can be found with companies like *Google Deepmind* and *OpenAI*. *Google* has been working on the board game '*GO'* to develop an algorithm to defeat that game's champions (Silver, et al., 2016). Moreover, the computer game '*Dota 2'* was mastered by the *OpenAI five* using the available human gameplay as training data to train the algorithm (Nandy, Abhishek, Biswas, & Manisha, 2018).

2.5.2.1 Classification Model

In classification models, clusters of data are classified into two or more categories using an algorithm. Classification is considered an instance of supervised machine learning. Clustering is the counterpart of classification, which is an instance of unsupervised machine learning where the main difference is that in clustering, it is done based on inheritance similarities rather than on known expected classification groups. In this particular research, classification models are used to classify the accelerometer input patterns as IRI values.

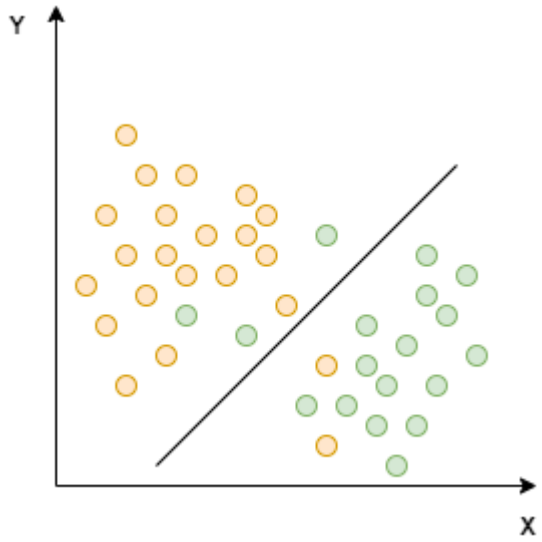

Figure 2.8: Classification Model

2.5.2.2 Regression Model

A regression model is used where it is necessary to predict an outcome based on past data. Such as upcoming weather anomalies prediction using previous weather data.

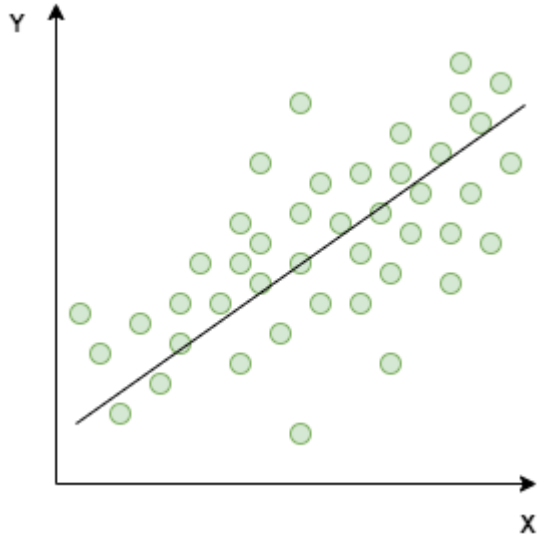

Figure 2.9: Regression Model

2.5.3 Unsupervised Machine Learning

Unsupervised learning has three main models. Unsupervised machine learning is similar to natural evolution. Like natural evolution, unsupervised learning is done by going through several iterations to evolve the model into a better stage than previously.

2.5.3.1 Generation (Genetic) Model

This model is heavily based on the natural selection process occurring in nature. It selects the best outcome and forwards the best features to the next iteration while other features are naturally ignored in the process. Genetic models were introduced as computer models in 1975 by John Holland (Holland, 1992). These models are often used where it is hard to identify the boundaries of a problem, such as where the number of possible next moves in the board game *'GO'*.

2.5.3.2 Clustering / Feature Learning

Clustering is the unsupervised version of the classification. While classification is done where the problem could be clearly understood, clustering is done often to classify unknown datasets. Clustering is commonly used in image processing to identify objects where there could not be previously predicted. As mentioned in (Blashfield & Aldenderfer, 1988), cluster classification was initially suggested by anthropologists Driver and Kroeber in 1932.

2.6 Summary

In the past, there were many solutions presented to calculate the IRI value reliably and fast. This research explores the possibility of calculating IRI value reliably while lowering the cost associated with the process. Moreover, to remove the barriers to access the IRI test.

3 METHODOLOGY

3.1 Data Collection

Data collection was carried out using an *Android* smartphone and an accelerometer sensor. Smartphones also used to save the collected accelerometer data while travelling on the road.

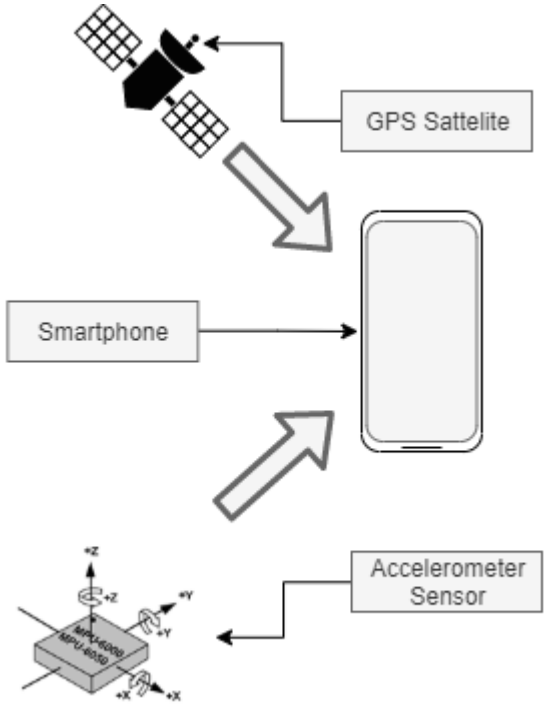

Figure 3.1: Data Collection Methodology

3.1.1 Accelerometer Sensor

The Accelerometer sensor was purchased locally. The main difficulty of this method is to extract data from the accelerometer while travelling on the road. Because of that, an accelerometer with a Bluetooth device is used to collect data. The chosen accelerometer sensor has the following specifications.

Table 3.1: Accelerometer Specifications

Feature	Description
Output data	Linear acceleration - x, y, and the z-axis direction Angular acceleration - around x, y, and z-axis Attitude - Euler angle
Output frequency	100 Hz
Connectivity	Bluetooth / USB
Required voltage	3.3 V to 5 V
Size (W x H x L)	15.24 mm x 3 mm x 15.24 mm

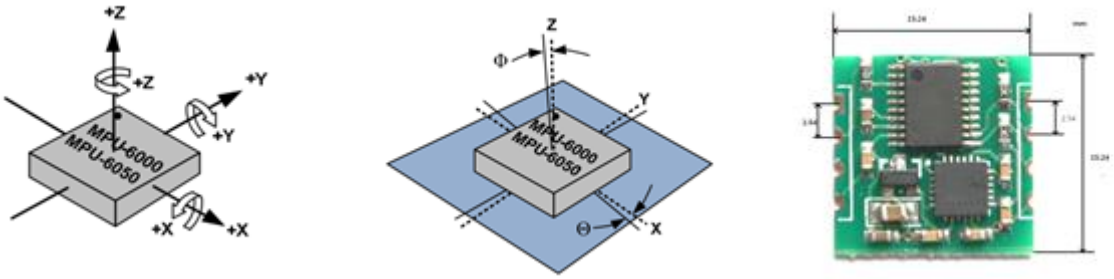

Figure 3.2: Accelerometer Sensor

This accelerometer sensor was attached to the driven wheel of a vehicle, as in the following diagram.

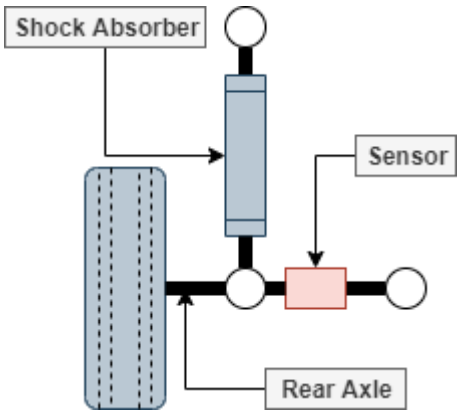

Figure 3.3: Sensor Placement

The sensor was placed as close as possible to the wheel to get the vibrations as much as possible.

3.1.2 Smartphone

In this research, a conventional mid-range Android device was used since the application that collects the data can is compatible with the Android operating system. The phone used in this research has the following specifications.

Table 3.2: Smart Phone Specifications

Feature	Description
Operating System	Android version 8.1
Bluetooth	Version 2.0
Global Positioning System	With a 2 Hz refresh rate

Figure 3.4: Mobile Phone

3.1.3 Smartphone Application

According to the date, a smartphone application was developed to read the accelerometer data and log these data in a text file. This application was developed using *Android Studio* with *Python* language. Also, this application is responsible for pulling the global positioning data from the built-in GPS module of the smartphone and matches that with the instantaneous accelerometer data. In the training phase, the same application was modified to collect pothole data with user input.

This data was then imported into *Microsoft Excel* for the initial data cleanup.

```
| Za:     -73.87| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       1.21 | Y:       0.34 | Z:      -0.14 | X1:       1.22 | Y1:     -10.31 | Z1:      12.57 | Xa:     179.99 | Ya:     -95.56
| Za:     -82.82| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       0.68 | Y:      -0.14 | Z:      -0.26 | X1:       0.00 | Y1:     -42.05 | Z1:      42.11 | Xa:     179.99 | Ya:     -97.49
| Za:     -70.29| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       0.79 | Y:      -0.02 | Z:      -0.33 | X1:      -3.54 | Y1:      -0.92 | Z1:     -12.45 | Xa:     179.99 | Ya:     -96.40
| Za:     -60.34| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       1.01 | Y:       0.06 | Z:      -0.32 | X1:      -4.58 | Y1:      25.45 | Z1:     -40.83 | Xa:     179.99 | Ya:     -96.31
| Za:     -59.87| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       1.27 | Y:      -0.07 | Z:      -0.37 | X1:      -0.79 | Y1:      42.72 | Z1:     -39.00 | Xa:     179.99 | Ya:     -96.34
| Za:     -73.37| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       1.29 | Y:      -0.09 | Z:      -0.29 | X1:      -0.92 | Y1:      19.47 | Z1:     -19.47 | Xa:     179.99 | Ya:     -96.75
| Za:     -77.67| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       0.95 | Y:      -0.23 | Z:      -0.26 | X1:       2.62 | Y1:     -21.30 | Z1:      30.70 | Xa:     179.99 | Ya:     -96.44
| Za:     -76.54| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: true|
10:37:52 | X:       1.03 | Y:      -0.14 | Z:      -0.33 | X1:       3.05 | Y1:      -2.99 | Z1:       8.85 | Xa:     179.99 | Ya:     -96.55
| Za:     -70.25| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:52 | X:       0.51 | Y:      -0.15 | Z:      -0.31 | X1:       4.09 | Y1:     -38.21 | Z1:      34.24 | Xa:     179.99 | Ya:     -96.05
| Za:     -67.11| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:52 | X:       1.69 | Y:      -0.01 | Z:      -0.40 | X1:      -2.81 | Y1:      20.81 | Z1:     -50.66 | Xa:     179.99 | Ya:     -95.94
| Za:     -66.81| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:52 | X:       0.54 | Y:      -0.37 | Z:      -0.25 | X1:       6.96 | Y1:     -22.89 | Z1:      93.81 | Xa:     179.99 | Ya:     -97.34
| Za:     -58.13| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:53 | X:       0.77 | Y:      -0.08 | Z:      -0.36 | X1:       0.85 | Y1:      -7.32 | Z1:      13.31 | Xa:     179.99 | Ya:     -96.36
| Za:     -61.00| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:53 | X:       1.10 | Y:      -0.03 | Z:      -0.38 | X1:      -1.65 | Y1:     -11.60 | Z1:     -10.86 | Xa:     179.99 | Ya:     -95.83
| Za:     -58.63| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:53 | X:       0.72 | Y:      -0.13 | Z:      -0.38 | X1:       0.55 | Y1:      -4.15 | Z1:       2.62 | Xa:     179.99 | Ya:     -96.09
| Za:     -49.00| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:53 | X:       1.44 | Y:       0.18 | Z:      -0.43 | X1:      -9.83 | Y1:      46.26 | Z1:    -111.63 | Xa:     179.99 | Ya:     -94.72
| Za:     -51.99| Latitude: 5.98006079 | Longitude: 80.42620566 | Pothole: false|
10:37:53 | X:       0.99 | Y:      -0.03 | Z:      -0.32 | X1:      -0.98 | Y1:      10.19 | Z1:     -15.20 | Xa:     179.99 | Ya:     -95.24
```

Figure 3.5: Data Record Format

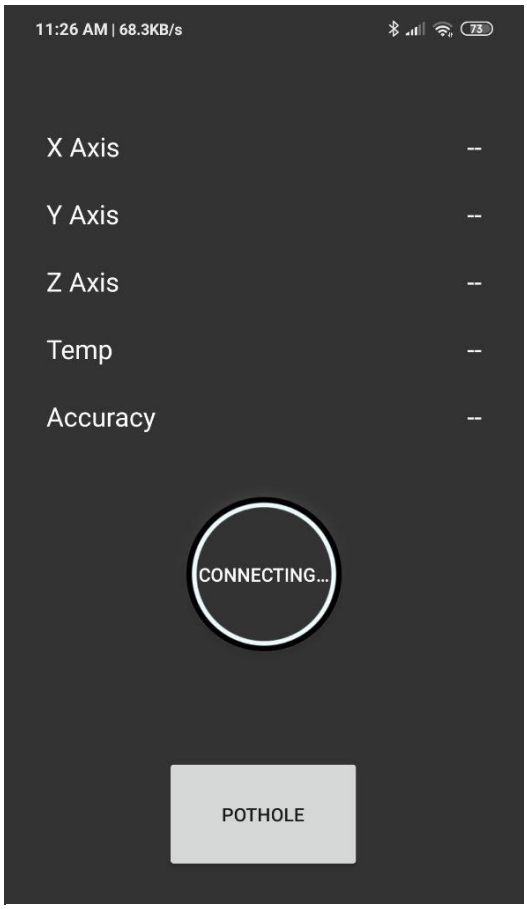

Figure 3.6: Mobile Phone Application

3.2 Data Pre-processing

Collected data using this method consist of;

- Time in seconds
- GPS data from the smartphone - latitude and longitude
- Linear acceleration - 3 components
- Angular acceleration - 3 components
- Pothole - true or false

For this research, we need to convert these data into more simple inputs for data analysis. Because of that, speed and distance were calculated using time and GPS data. Then three components of linear acceleration and angular acceleration were combined into one parameter. Then each combined acceleration is normalized by the speed to eliminate the speed factor of the data collection.

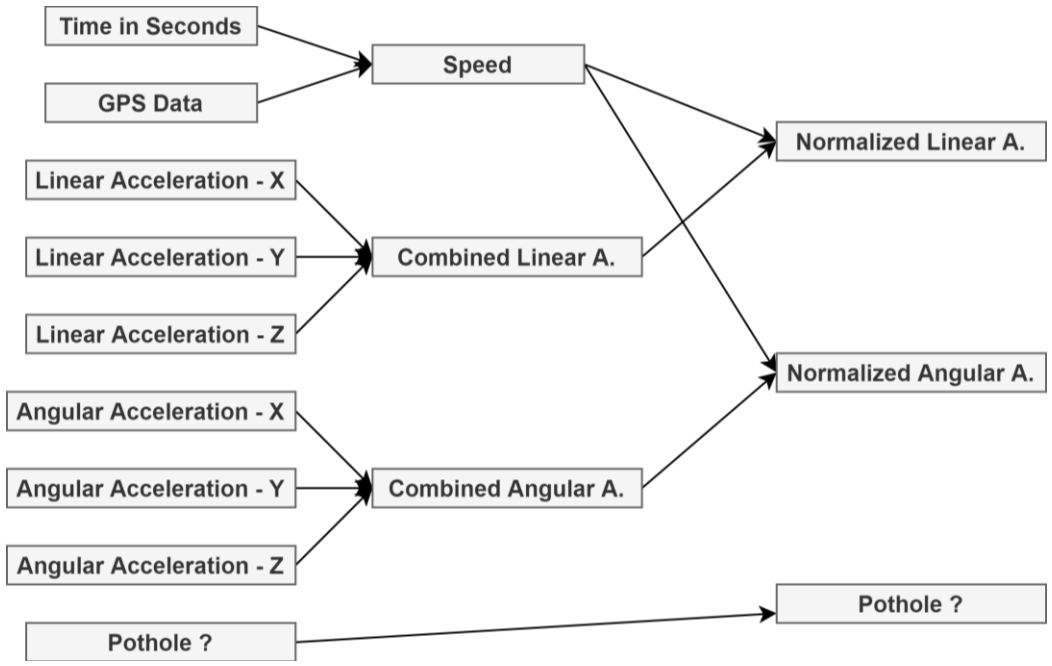

Figure 3.7: Data Preprocessing Procedure

3.2.1 Speed Calculation

Speed is calculated using the recorded time and distance calculated from two GPS coordinates. The distance calculated with the following formula;

$$Distance(m) = acos(\ sin(lat1) \times sin(lat2) + cos(lat1) \times cos(lat2) \times cos(lon2 - lon1)\ ) \times 6371000$$

Then the speed calculated by this simple equation;

$$Speed = \frac{Distance}{(time\ 2 - time\ 1)}$$

3.2.2 Combined Acceleration Calculation

Accelerometer orientation is different each time when it is fixed to the vehicle axle. Thus, both linear and angular acceleration components are combined using the following formula to minimize the inconsistency of accelerometer orientation on the vehicle axle.

$$Combined\ Acceleration = \sqrt{(X\ Axis)^2 + (Y\ Axis)^2 + (Z\ Axis)^2}$$

This calculation was performed in the *Microsoft Excel* package.

3.2.3 Data Arrangement

In general, the IRI data is noted down for 100-meter sections. Because of that, the above data had to be separated into 100-meter sections. This calculation was done by using the distance calculated in 3.2.1.

3.3 Data Analysis

A machine-learning algorithm based on an artificial neural network was used in this project to analyze data. *Tensorflow* is a free and open-source library that can be repurposed easily. *Weka* is a software that was developed by the University of Waikato, New Zealand. Both software was used to analyze and predict the IRI values taken from the sensor and road profiler.

Weka software was used to identify the parameters for machine learning algorithms. Since many parameters have been fine-tuned before implementing those parameters in the *Tensorflow, t*his was done because even though *Tensorflow* is much more flexible than *Weka,* it is easy to fine-tune and get the result that we want.

3.3.1 Using Weka Software

Software Information;

Table 3.3: Weka Software Specifications

Name	Waikato Environment for Knowledge Analysis
Version	3.8.3
Year	2018
Developer	The University of Waikato Hamilton, New Zealand

A Multilayer perceptron analysis used to analyze the preprocessed data.

3.3.2 Using Tensorflow Library

Tensorflow JavaScript API 2.0.0 was used in this research to make a neural network with two inputs, one hidden layer with ten nodes, and one output layer. These changes made based on the initial analysis done on the *Weka* software.

The pothole detection analysis carried out initially. The following neural network was trained based on the previously collected data on the pothole with the accelerometer data.

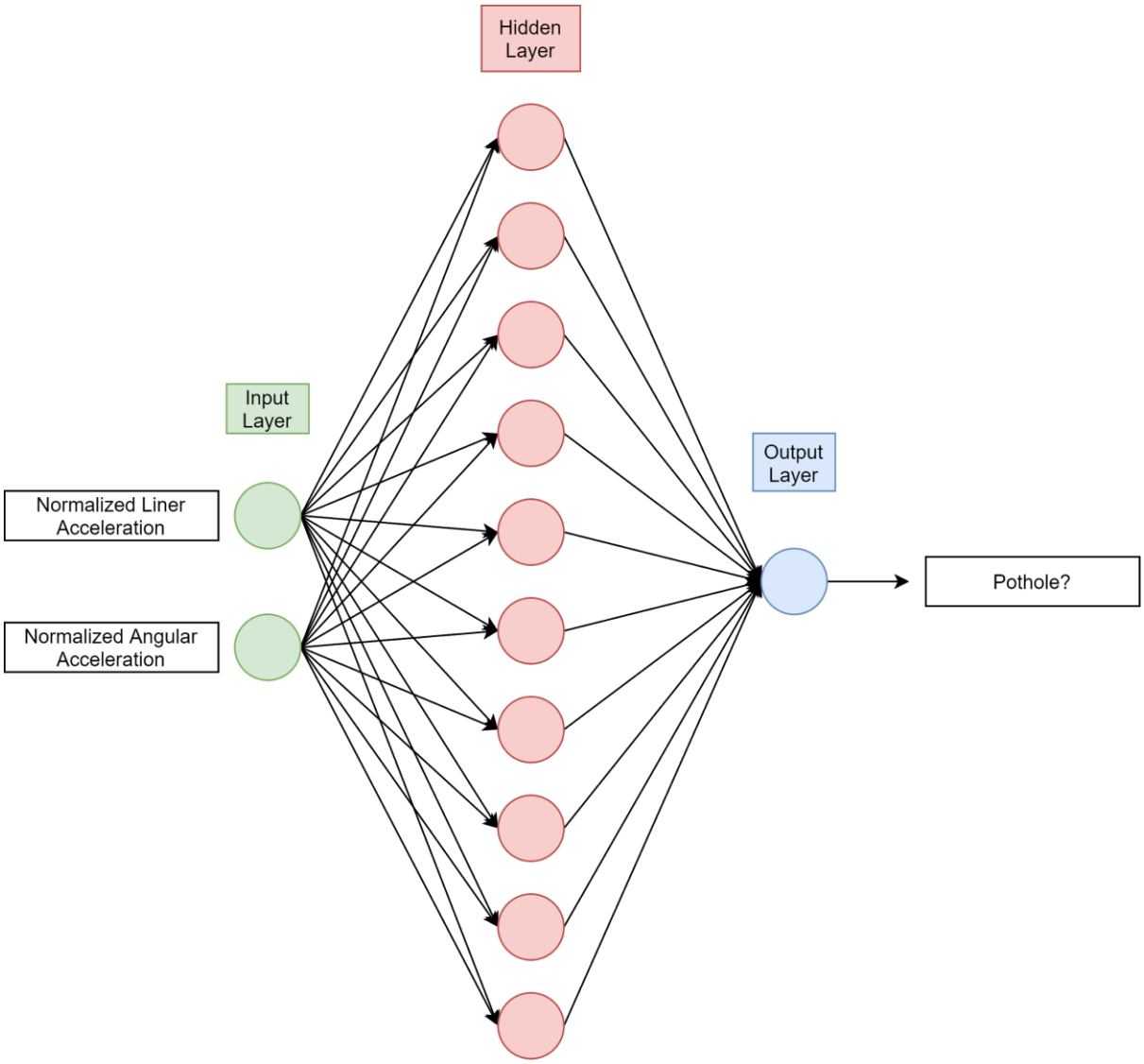

Figure 3.8: Neural Network Model

Moreover, the same neural network has been used to calibrate the accelerometer data with the previously collected IRI values by the *Road Development Authority* on the same road section.

This library includes several options to fine-tune the outcome of the neural network, such as the following.

```
function preload(){
  xs_data = loadTable('xs.csv','csv','header');
  ys_data = loadTable('ys.csv','csv','header');

}

function setup() {
  noCanvas();
  //console.log(xs_data.getRowCount());

  const inputx = (xs_data.getArray());
  const shapex = [29,3];
  const inputy = (ys_data.getArray());
  const shapey = [29];

  const xs = tf.tensor(inputx, shapex, 'float32');
  const ys = tf.tensor(inputy, shapey, 'float32');

  console.log(ys);
  ys.print();

  const model = tf.sequential();
  const configHidden = {
    inputShape : [3],
    units : 10,
    activation : 'sigmoid'
  }
  const hidden = tf.layers.dense(configHidden);

  const configOutput = {
    units : 1,
    activation : 'sigmoid'
  }

  const output = tf.layers.dense(configOutput);
```

Figure 3.9: JavaScript for Tensorflow

3.3.2.1 Tensors

Tensors represent inputs or outputs in a neural network. In the *Tensorflow* library, tensors can initiate as a scalar, vector, or matrix with multiple dimensions. The type of tensor has chosen according to the inputs given to these tensors. Humans are good at identifying the relationships between two variables (in the x and y-axis). However, machines are better at predicting the relationships with more than two variables. In this research, the input tensor is initiated as a matrix of '1 x 2 x number of data points'. Furthermore, data is feed as floating-point values.

3.3.2.2 Layers

In a neural network, there are three significant layers involved.

- Input layer
- Hidden layers
- Output layer

Form this, input layers and output layers are self-explanatory. In a neural network, inputs taken by the input layers and pass it to hidden layers by multiplying them with a bias. Links between nodes represent the bias values. Then the hidden layer transfers those values to the outputs with a similar bias value. The final value is compared with the given actual values. The difference between the actual value and the output value is then minimized in the training process.

In the *Tensorflow*, these layers should be initiated with the following configurations.

- Input shape - number of inputs to the current layer from the previous one
- Units - the number of nodes in the current layer

- Activation function - use to pass the data to the next layer when the value meets the activation function criteria. *Rectified linear units*, *sigmoid*, *softmax*, and few other functions are used in this research. It found that for the pothole detection neural network, *sigmoid* and *softmax* functions work better, and in IRI prediction, the linear activation function works better.

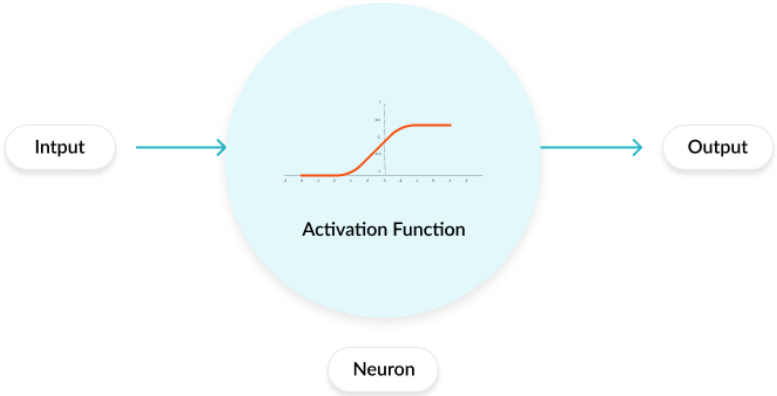

Figure 3.12: activation Function Process

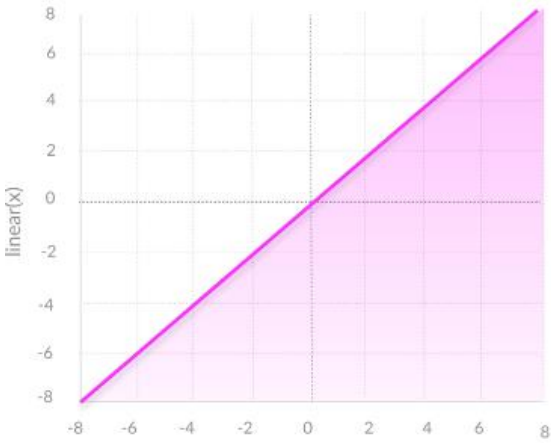

Figure 3.12: Linear Activation Function

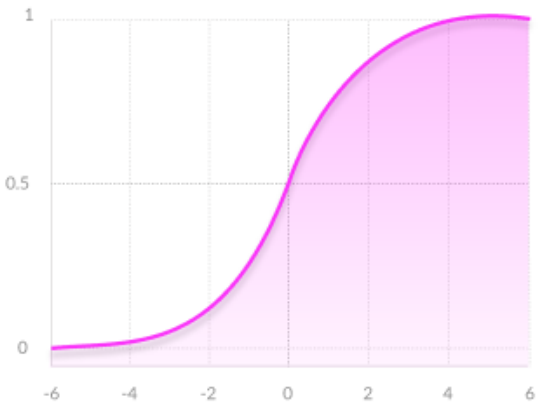

Figure 3.12: Sigmoid Activation Function

3.3.2.3 Compiler

The compiler's function is to bind the inputs and actual outputs in the neural network. It minimizes the loss between the predicted output and the actual output. Loss can be calculated as a mean squared error, cousin distance, or absolute difference between predicted values and actual values.

4 RESULTS

4.1 Pothole Detection

The following data were obtained using the accelerometer sensor along the A2 road in the Matara area. A section of the data is included here.

Table 4.1: A2 Road Accelerometer Data

Time	Angular Acceleration			Linear Acceleration			Latitude	Longitude	Pothole
	X	Y	Z	X	Y	Z			
10:18:56	0.92	-0.16	-0.3	1.28	-9.03	16.66	5.93974088	80.47991128	FALSE
10:18:56	0.68	0.03	-0.25	-1.77	11.35	-15.93	5.93974088	80.47991128	FALSE
10:18:56	1	-0.1	-0.23	0.61	11.29	-4.15	5.93974088	80.47991128	FALSE
10:18:56	0.7	-0.16	-0.31	0.43	0	2.87	5.93974088	80.47991128	TRUE
10:18:56	5.32	2.18	0.03	-46.39	157.47	-390.5	5.93974088	80.47991128	TRUE
10:18:57	1.28	-0.49	-0.52	32.78	-91.06	274.17	5.93974088	80.47991128	TRUE
10:18:57	-1.58	-0.57	0.26	34.06	-53.65	95.15	5.93974088	80.47991128	TRUE
10:18:57	0.08	-0.8	-0.59	28.69	-70.62	343.14	5.93971337	80.47978609	FALSE
10:18:57	0.36	-0.56	-0.01	14.28	-15.14	70.19	5.93971337	80.47978609	FALSE
10:18:57	0.16	-0.82	-0.36	27.47	-162.96	282.04	5.93971337	80.47978609	TRUE
10:18:57	0.78	-0.49	-0.44	17.52	-34.67	158.94	5.93971337	80.47978609	TRUE
10:18:57	1.01	-0.12	-0.33	2.08	20.14	9.83	5.93971337	80.47978609	FALSE
10:18:57	0.93	-0.17	-0.24	4.52	-13.85	38.7	5.93971337	80.47978609	FALSE
10:18:57	0.74	-0.14	-0.35	2.26	-6.96	12.39	5.93971337	80.47978609	FALSE
10:18:58	1.05	-0.23	-0.35	1.77	22.4	8.97	5.93971337	80.47978609	FALSE

10:18:58	1.15	-0.19	-0.43	9.4	-46.14	68.97	5.93971337	80.47978609	FALSE
10:18:58	0.85	0.05	-0.47	0.92	-25.63	0	5.93969077	80.47964384	FALSE
10:18:58	1.54	0.19	-0.35	-2.99	24.84	-38.21	5.93969077	80.47964384	FALSE
10:18:58	0.37	-0.23	-0.41	4.46	-5.07	51.7	5.93969077	80.47964384	FALSE
10:18:58	1.44	0.09	-0.31	-10.38	43.58	-108.22	5.93969077	80.47964384	FALSE
10:18:58	-0.26	-0.16	-0.22	23.5	-108.95	105.77	5.93969077	80.47964384	TRUE

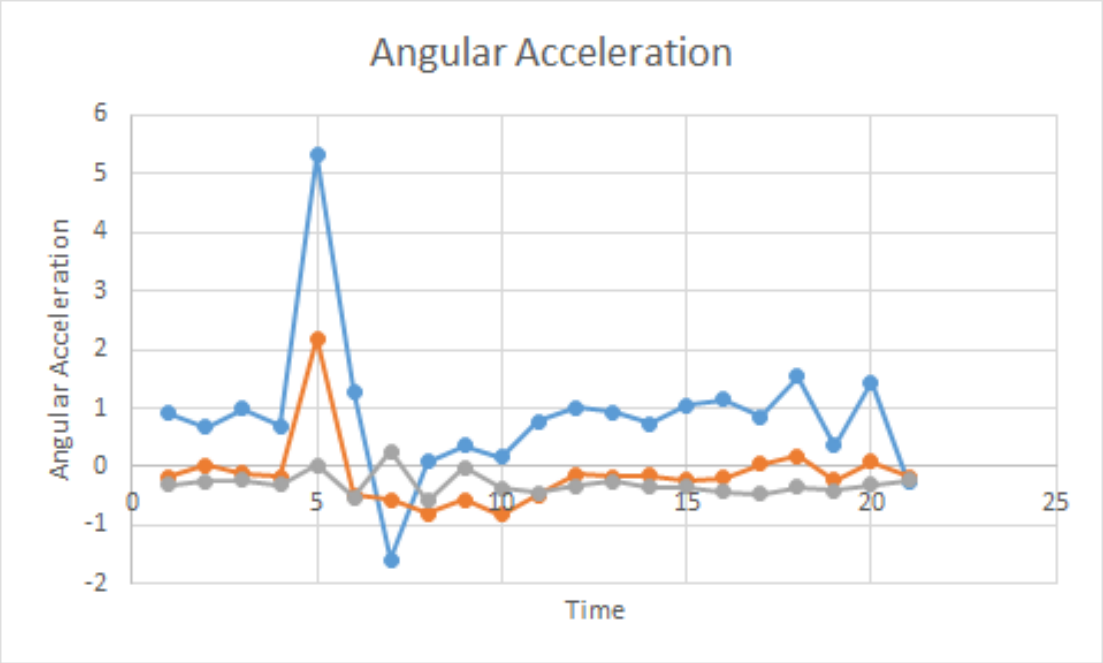

Figure 4.1: Angular Acceleration

In these graphs, we can see the spike of linear and angular accelerations due to vehicles travelling on a pothole. Similar data have been fitted into a machine learning algorithm. The following results from the algorithm have been observed when the training has been completed. One thousand nine hundred thirty-four data points have been used in this training.

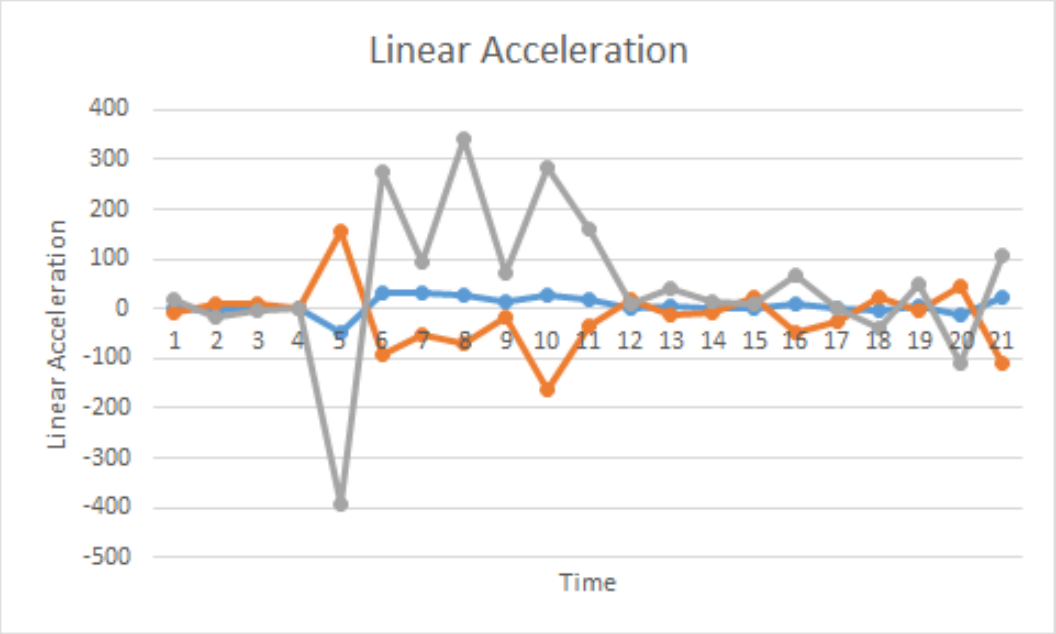

Figure 4.2: Linear Acceleration

Table 4.2: Pothole Detection Results

Correctly classified instances	94.83%
Incorrectly classified instances	5.17%
Mean absolute error	0.0483
Root mean squared error	0.142

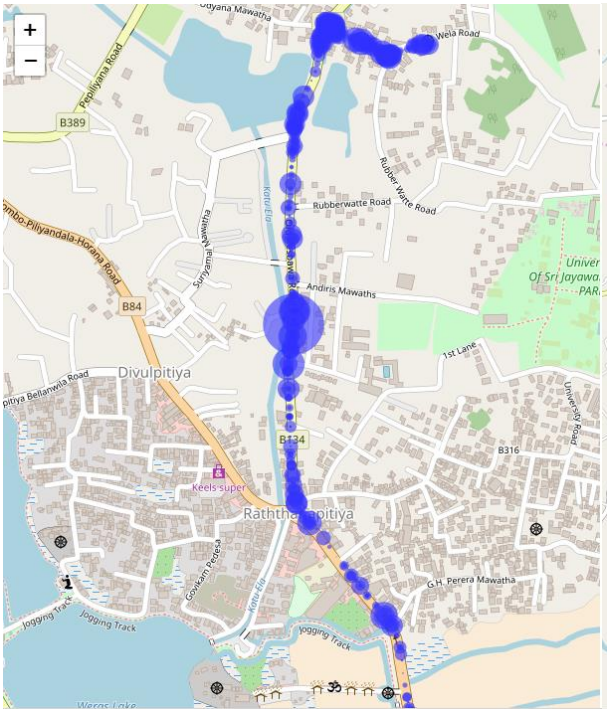

Figure 4.4: Boralasgamuwa Road

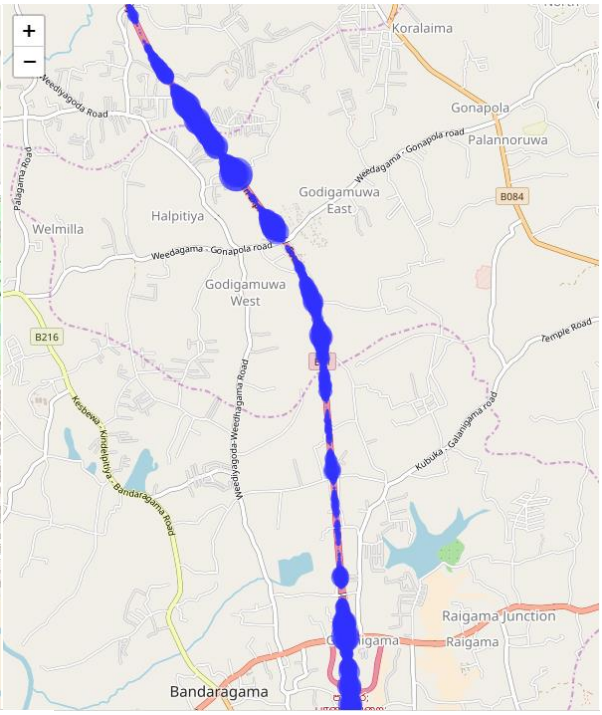

Figure 4.4: Southern Expressway

The left image is a map of the Boralasgamuwa area, and the right image is a map of the Southern Expressway section. Linear accelerations are mapped into that image using collected latitude and longitude data. The size of the circles represents the acceleration intensity.

The neural network used the following settings;

Table 4.3: Pothole Detection Neural Network Settings

Number of inputs	2 (Normalized linear acceleration, Normalized angular acceleration)
Number of hidden layers	1
Number of hidden nodes	10
Activation function in hidden nodes	sigmoid
Number of outputs	1

Activation function in hidden nodes	sigmoid
Optimizer	Stochastic gradient descent
Loss calculation function	mean squared error

4.2 Calibration to IRI Data

IRI data for the Piliyandala bypass road was obtained from the Road Development Authority, Sri Lanka.

Table 4.4: IRI Data for Piliyandala Bypass Road

Chainage (km)	IRI Left	IRI Average
0.10	3.38	4.2
0.20	2.61	3.14
0.30	2.22	2.43
0.40	2.31	2.75
0.50	2.29	2.4
0.60	2.59	2.78
0.70	2.16	2.41
0.80	X	X
0.90	2.27	2.65
1.00	1.75	2.09
1.10	1.96	2.14
1.20	1.96	2.13

1.30	1.9	1.95
1.40	2.51	2.78
1.50	2.29	2.6
1.60	2.4	2.27
1.70	2.7	2.6
1.80	2.12	2.19
1.90	X	X
2.00	1.86	1.96
2.10	1.73	1.93
2.20	1.79	1.59
2.30	2.06	2.03
2.40	1.84	1.72
2.50	1.54	1.66
2.60	1.28	1.39
2.70	2.4	2.25
2.80	1.83	1.92

Accelerometer data is collected on the same road section during the same survey period.

Table 4.5: Accelerometer Data for Piliyandala Bypass Road

Chainage (km)	Speed (m/s)	Average Liners Acceleration	Average Angular Acceleration	Corresponding IRI Value
0.00	8.33	5.52	1.00	3.38
0.10	11.11	3.84	0.97	2.61
0.20	12.50	2.84	0.98	2.22
0.30	10.00	3.82	1.04	2.31
0.40	11.11	4.26	0.99	2.29
0.50	11.11	4.59	1.01	2.59
0.60	12.50	3.89	1.02	2.16
0.80	9.09	3.98	1.01	2.27
0.90	10.00	3.62	0.97	1.75
1.00	9.09	3.74	1.01	1.96
0.00	8.33	11.97	0.99	3.38
0.10	12.50	5.45	0.98	2.61
0.20	12.50	7.70	0.98	2.22
0.30	12.50	6.09	1.02	2.31
0.40	12.50	5.65	1.00	2.29
0.50	12.50	5.52	1.01	2.59
0.60	10.00	4.26	1.01	2.16
0.80	12.50	3.66	1.01	2.27
0.90	10.00	2.80	0.98	1.75
1.00	12.50	3.06	0.97	1.96
1.10	9.09	4.60	1.01	1.96
1.20	11.11	4.29	1.01	1.9

1.30	6.67	4.99	0.99	2.51
2.20	7.14	3.00	0.99	2.06
2.30	12.50	4.79	0.99	1.84
2.40	12.50	4.51	0.96	1.54
2.50	12.50	6.00	1.00	1.28
2.60	12.50	5.78	0.98	2.4
2.70	12.50	6.91	1.02	1.83

The above data was fed into the neural network. The following parameters were observed during the training process.

Table 4.6: Results for IRI - Accelerometer Relationship

Mean absolute error	0.1378
Root mean squared error	0.1889

These are the measurement between results and the prediction data. The mean absolute error and the root mean squared error is higher than the previous pothole detection calculation. High error values are due to a lower number of training data compared to the previous case.

Following settings were used in the neural network,

Table 4.7: IRI- Accelerometer Neural Network Settings

Number of inputs	2 (Normalized linear acceleration, Normalized angular acceleration)
Number of hidden layers	1
Number of hidden nodes	10
The activation function in hidden nodes	Linear
Number of outputs	1

The activation function in hidden nodes	Linear
Optimizer	Stochastic gradient descent
Loss calculation function	mean squared error

4.3 Real-time Data Processing

For the practical usage of this model, the model should be export as a *Tensorflow lite* package. This package can be used in an *Android* or *iOS* application to indicate the IRI value in real-time while collecting the data through an accelerometer.

Another option is to collect the data in the field and run the model on a separate computer after the data collection. There is no difference between running the model on a separate computer or having it in a mobile application package.

5 DISCUSSION

The machine-learning algorithm is a useful tool to identify patterns and correlations when there are multi-dimensional data and large datasets. It can identify Multi-dimensional data patterns that a human mind could not identify. In the sense of accelerometer data against the IRI value, 3- axis accelerometer data and GPS data along the time dimension generate a large amount of non-linear data that has an indirect correlation with the IRI value. In cases like this, machine learning can identify patterns such as a wheel of a vehicle travels on a pothole or a road bump. Also, it can be used to predict the IRI value by training the machine-learning algorithm with a known dataset. These algorithms mostly rely on the quality of the training dataset. The quality of the training dataset can be expressed by reliability, feature representation and, skewness.

Reliability refers to the accuracy of the training data. If the dataset is not reliable, the trained model will output the unreliable predictions as well. It is "Garbage in, garbage out".

Feature representation is another essential aspect of the training data. First, the training data should represent all the scenarios that intend to predict by the model. Furthermore, it should not be skewed as well. A skewed dataset can bias the output predictions of the model depending on the dataset size and skewness. (Google, 2021)

In IRI prediction, the corresponding accelerometer data's predicted values are not closely associated with the laser-based mobile unit's data. Typically, the laser-based mobile unit assesses the total lane width and gives a collective result based on the measurements. However, the accelerometer-based calculation is calculated based on the wheel path undulations. This fundamental difference in the data collection can cause very different results. This difference can lead to an inaccurate machine learning model in the training phase. A lower variation of the result obtained from the laser-based IRI measurement and the accelerometer measurement leads to lower absolute error and lower mean squared error.

Further research should be carried out to determine the feasibility of using the machine-learning algorithm for IRI prediction using similar measurement methods such as walking profiler meter or travelling beam method. Also, the laser-based IRI

measurement gives the IRI value for every 100m section. However, the accelerometer has a data output rate of around 50Hz. Due to this measurement difference, gathering the data for training the machine learning algorithm is difficult. One way around this problem is to get the mean acceleration values corresponding to every 100m section of a particular road and compare it with the IRI measurement on the same road.

The pothole detection phase in this research can be categorized under the reinforced learning. It is similar to the features like step counting applications used in smart devices. In this research, accelerometer data gathered from several potholes used to train the machine-learning algorithm. Moreover, a separate road section was used to test and verify the machine-learning algorithm. The outcome of the testing data suggests that this algorithm can identify the potholes to an acceptable level. However, in general principles of machine learning, there is always an opportunity for improvements through more training. Even though it is evident that the trained model can identify the potholes to a fair degree of accuracy, it can be suggested to train the algorithm further with different road sections, different speeds, and using different vehicles.

The training phase is the most crucial stage of machine learning models. The number of training cycles with the available data should be carefully selected. While a large number of training cycles can show very accurate results within the training dataset, the model may underperform when given a new set of data. Overfitting or overtraining is a statistical phenomenon that picks up the training dataset's residual variations and transfers them to the machine learning model (Overfitting, 2021). Thus, it is essential to find a good balance between the number of training cycles and the training data variation. Overfitting can be identified with a new set of testing data that is separate from the training data.

6 CONCLUSION

This research was conducted to develop a method to find pavement undulation detection within a low budget level. The method's accuracy would be at a sufficient level to detect the distresses prevailing in the roadway. Off-the-shelf accelerometer sensors with reasonable accuracy can be found in almost every electronic store for a low price. The main problem of this method is to convert the accelerometer data to pavement undulations. It is ubiquitous to use mobile phone accelerometer data for pavement undulation measurement as a cheap alternative to IRI measurement. However, the data conversion to the IRI values in those researchers is not conducted thoroughly.

This research can be divided into two major sections. 1. Pothole detection in the road surface and 2. IRI value predictions using accelerometer data.

In the first section of this research, the results indicate that pothole detection using accelerometer and machine learning can have promising results. The low mean absolute error in pothole detection is a direct indication of the accuracy of the results. This research was carried out with a limited number of data points (1934). Machine learning algorithms can be improved sharply by using more massive data sets. The main reason to use machine learning in this research is to find the accelerometer data patterns, which the regression models could not highlight. Pattern detection with machine learning is not a new concept. It has been used in many cases, such as step counters in fitness devices and mobile phones. The same concept can be used to detect the potholes on the road surface.

The second section of this research was conducted to formulate a machine learning algorithm that can predict the IRI value using the accelerometer data. The outcome of the training indicates somewhat accurate results, mainly due to the lack of test data. This machine-learning algorithm was trained upon 29 test points due to the lack of available public domain IRI data, which were obtained recently. Lack of variations and limited test data points profoundly affected the mean absolute error in these results. Even though the undulations are identified with the accelerometer data, conversion to the IRI should be further researched before implementation. However, these results

provide new insight to improve the accuracy of mobile phone undulation detection methods using the machine-learning algorithm as well.

Due to the lack of test data for the accelerometer measurement - the IRI conversion model could not be validated further. Quality of the training data, machine learning settings such as the number of hidden nodes, type of the activation function used defy the accuracy of the prediction. Nevertheless, having a large number of hidden nodes and training cycles could lead to overfitting of the model. This problem could be minimized with fewer hidden nodes, and a testing data set separate from the training data set. A perfect data set should represent all the variables on a measurement, such as weather conditions, the measuring vehicle's speed, vehicle type, and model. Using such a dataset, this machine learning model can be improved vastly.

Pothole detection with this method has an acceptable accuracy with a root mean square error of 0.142. Moreover, the IRI relationship with the accelerometer data also has an acceptable accuracy with a root mean square error value of 0.1889.

Machine learning is a flexible and valuable tool in the transportation sector, given that the algorithm is trained with a satisfactory amount of data and quality. Specifically, the pothole direction of a paved road can be successfully detected using a machine learning algorithm.

These machine learning algorithms can be improved on the go by having more training data. Rather than using regression models for IRI and distress relationship, machine learning can improve the accuracy of those predictions.

It can conclude that with the aid of machine learning combined with a low-cost accelerometer sensor, road pavement undulations such as potholes can be accurately identified on the go.

7 REFERENCES

Abeywardana, H. M., Abeywikrama, U. M., Amarasinghe, P. T., & Kumarasinghe, R. P. (2018). *iRoads - Smartphone-Based Road Condition Monitoring.* University of Moratuwa, Computer Science & Engineering.

Bennett, C. R. (2006). *Data Collection Technologies for Road Management.* Retrieved from https://books.google.com/books/about/Data_Collection_Technologies_for_Road_Ma.html?hl=&id=tSvNwQEACAAJ

Blashfield, R. K., & Aldenderfer, M. S. (1988). The Methods and Problems of Cluster Analysis. In J. R. Nesselroade, & R. B. Cattell, *Handbook of Multivariate Experimental Psychology* (pp. 447-473). Bostan,MA. doi:10.1007/978-1-4613-0893-5_14

Douangphachanh, V., & Oneyama, H. (2013). *Estimation of road roughness condition from smartphones under realistic settings.* doi:10.1109/ITST.2013.6685585

Fifth Wheel Bump Integrator. (2012, March). Retrieved September 09, 2019, from CSIR - Central Road Research Institute: https://crridom.gov.in/content/bump-integrator

Firoozi, Y. S., Mahmoudzadeh, A., Azizpour, M. A., & {others}. (2017). Validation of Smartphone-Based Pavement Roughness Measures. Retrieved from https://www.sid.ir/en/Journal/ViewPaper.aspx?ID=600013

Gamage, D., Pasindu, H. R., & Bandara, S. (2016, July 27). Pavement Roughness Evaluation Method for Low Volume Roads. doi:10.3850/978-981-11-0449-7-199-cd

Google. (2021, 01 05). *Data Preparation and Feature Engineering for Machine Learning*. Retrieved from Google Developers: https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press. Retrieved from https://books.google.com/books?hl=en&lr=&id=5EgGaBkwvWcC&oi=fnd&

pg=PR7&dq=Adaptation+in+Natural+and+Artificial+Systems&ots=mImr8-Ojwq&sig=UvO64uPE2cEcw4j2D3i8C68SBXE

Merry, K., & Bettinger, P. (2019, July 18). Smartphone GPS accuracy study in an urban environment. *PLoS One, 14*(7). doi:10.1371/journal.pone.0219890

Nandy, Abhishek, Biswas, & Manisha. (2018). OpenAI Basics. doi:10.1007/978-1-4842-3285-9_3

Overfitting. (2021, 03 29). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Overfitting

Paterson, W. D. (1986). *Relationship of the International Roughness Index to Other Measures of Roughness and Riding Quality.* Retrieved from https://books.google.com/books/about/Relationship_of_the_International_Roughn.html?hl=&id=b_ahtgAACAAJ

Russell, S., & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach, Global Edition.* Retrieved from https://books.google.com/books/about/Artificial_Intelligence.html?hl=&id=XS9CjwEACAAJ

Samuel, A. L. (1959, July). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development, 3*(3), 210-229. doi:10.1147/rd.33.0210

Sandamal, R. M., & Pasindu, H. R. (2020). Applicability of smartphone-based roughness data for rural road pavement condition evaluation. *International Journal of Pavement Engineering*, 1-10. doi:10.1080/10298436.2020.1765243

Silva, M., & Pasindu, H. R. (2017, October 01). Development of a methodology for road maintenance planning of low volume roads based on roughness data.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., . . . Sutske. (2016, January 28). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484-489. doi:10.1038/nature16961

Wang, H. (2006, 07). Road Profiler Performance Evaluation and Accuracy Criteria Analysis.

ANNEX 01 – DATA PRE-PROCESSING WITH JAVASCRIPT (P5)

sketch.js (JavaScript in P5)

```
let data;
var json = {};
var table;

function preload(){
  data = loadStrings('0292018.txt');
}

function setup(){
  background(200);
  table = new p5.Table();
  var newRow = table.addRow();

  table.addColumn('id');
  table.addColumn('time');
  table.addColumn('x');
  table.addColumn('y');
  table.addColumn('z');
  table.addColumn('x1');
  table.addColumn('y1');
  table.addColumn('z1');
  table.addColumn('xa');
  table.addColumn('ya');
  table.addColumn('za');
  table.addColumn('latitude');
  table.addColumn('longitude');
  table.addColumn('speed');

  for(var i =0; i< data.length; i++){
      let time = data[i].substring(0, 8);
      let x = data[i].substring(19, 24);
      let y = data[i].substring(35, 40);
      let z = data[i].substring(51, 56);
      let x1 = data[i].substring(68, 73);
      let y1 = data[i].substring(85, 90);
      let z1 = data[i].substring(102, 107);
      let xa = data[i].substring(117, 124);
      let ya = data[i].substring(134, 141);
      let za = data[i].substring(151, 158);
      let latIndex = data[i].indexOf("Latitude");
      let longIndex = data[i].indexOf("Longitude");
      let lat = data[i].substring(latIndex+9,latIndex+20);
```

```
      let long = data[i].substring(longIndex+10,longIndex+22);
      var newRow = table.addRow();
      newRow.setNum('id',i+1);
      newRow.setString('time',time);
      newRow.setString('x',x);
      newRow.setString('y',y);
      newRow.setString('z',z);
      newRow.setString('x1',x1);
      newRow.setString('y1',y1);
      newRow.setString('z1',z1);
      newRow.setString('xa',xa);
      newRow.setString('ya',ya);
      newRow.setString('za',za);
      newRow.setString('latitude',lat);
      newRow.setString('longitude',long);
      if(i>= 11){
        let t1 = table.get(i,'time');
        let t2 = table.get(i-10,'time');
        let lat1 = table.get(i,'latitude');
        let lat2 = table.get(i-10,'latitude');
        let long1 = table.get(i,'longitude');
        let long2 = table.get(i-10,'longitude');
        let finalspeed = speedCalculation(t1,t2,lat1,lat2,long1,long2);
        newRow.setString('speed',finalspeed);
      }
  }
  saveTable(table, 'new.csv');
}

function speedCalculation(t1, t2, lat1, lat2, long1, long2){
  let hour1 = t1.substring(0,2);
  let hour2 = t2.substring(0,2);
  let min1 = t1.substring(3,5);
  let min2 = t2.substring(3,5);
  let sec1 = t1.substring(6,8);
  let sec2 = t2.substring(6,8);
  //angleMode(RADIANS);
  let distance = acos(cos(radians(90-lat2))* cos(radians(90-
lat1))+ sin(radians(90-lat2))* sin(radians(90-
lat1))* cos(radians(long2-long1)))* 6371000;
  let timeDiff = (hour1-hour2)*3600+(min1-min2)*60+(sec1-sec2);
  let speed = distance/timeDiff;
  return speed;
}
```

ANNEX 02 – ACCELEROMETER DATA WITH IRI/POTHOLE PREDICTION (MACHINE LEARNING WITH TENSORFLOW)

sketch.js (JavaScript in P5)

```
function preload(){
  xs_data = loadTable('xs.csv','csv','header');
  ys_data = loadTable('ys.csv','csv','header');

}

function setup() {
  noCanvas();

  const inputx = (xs_data.getArray());
  const shapex = [29,3];
  const inputy = (ys_data.getArray());
  const shapey = [29];
  const xs = tf.tensor(inputx, shapex, 'float32');
  const ys = tf.tensor(inputy, shapey, 'float32');
  console.log(ys);
  ys.print();

  const model = tf.sequential();
  const configHidden = {
    inputShape : [3],
    units : 10,
    activation : 'sigmoid'
  }
  const hidden = tf.layers.dense(configHidden);

  const configOutput = {
    units : 1,
    activation : 'sigmoid'
  }

  const output = tf.layers.dense(configOutput);

  model.add(hidden);
  model.add(output);

  const optimizer = tf.train.sgd(0.1);
  const configCompiler = {
    optimizer : optimizer,
```

```
    loss : 'meanSquaredError',
    epoch: 10
  }
let history;

train().then(()=>{
  let output = model.predict(xs);
  xs.print();
  output.print();
  console.log('Training Complete!');
})
  async function train(){
    for(let i =0; i < 1000; i++){
      model.compile(configCompiler);
      const response = await model.fit(xs, ys);
      console.log(response.history.loss[0]);
    }
  }


}
```

index.html (P5)

```
<!DOCTYPE html><html lang="en"><head>
    <script src="p5.js"></script>
    <script src="p5.sound.min.js"></script>
    <script src="tf.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">

  </head>
  <body>
    <script src="sketch.js"></script>


</body></html>
```

ANNEX 03 - DATA COLLECTION ANDROID APPLICATION WITH KOTLIN

MainActivity.kt

```
package com.typeiii.bth

import android.Manifest
import android.databinding.DataBindingUtil
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import com.typeiii.bth.helpers.BluetoothSerial
import com.typeiii.bth.databinding.ActivityMainBinding
import android.graphics.drawable.AnimationDrawable
import android.view.MotionEvent
import android.view.View
import com.karumi.dexter.Dexter
import com.karumi.dexter.MultiplePermissionsReport
import com.karumi.dexter.PermissionToken
import com.karumi.dexter.listener.PermissionRequest
import com.karumi.dexter.listener.multi.MultiplePermissionsListener
import com.karumi.dexter.listener.multi.SnackbarOnAnyDeniedMultiplePerm
issionsListener
import com.karumi.dexter.listener.multi.CompositeMultiplePermissionsLis
tener
import com.typeiii.bth.helpers.LocationManager
import kotlinx.android.synthetic.main.activity_main.*
import android.R.attr.button
import android.content.Context
import android.content.Intent
import android.widget.EditText
import android.widget.Toast
import com.typeiii.bth.helpers.Recorder
import android.content.SharedPreferences

class MainActivity : AppCompatActivity() {

    lateinit var binding: ActivityMainBinding
    private var bluetoothSerial: BluetoothSerial? = null
    private var allPermissionsListener: MultiplePermissionsListener? =
null

    private val deviceName = "HC-06"
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView(this, R.layout.activit
y_main)
        supportActionBar?.hide()
        checkPermissions()
        connectToBluetoothDevice()
        getAccuracy()
        setClickListener()
        onClickListener()
    }


    private fun checkPermissions() {

        val permissionListener = object : MultiplePermissionsListener {
            override fun onPermissionsChecked(report: MultiplePermissio
nsReport) {
                for (response in report.grantedPermissionResponses) {
                    if (response.permissionName == Manifest.permission.
ACCESS_FINE_LOCATION ) {
                        getAccuracy()
                    }
                }
            }

            override fun onPermissionRationaleShouldBeShown(permissions
: MutableList<PermissionRequest>?,
                                                            token: Perm
issionToken?) {
            }

        }

        allPermissionsListener = CompositeMultiplePermissionsListener(p
ermissionListener,
                SnackbarOnAnyDeniedMultiplePermissionsListener.Builder.
with(binding.view,
                        R.string.all_permissions_denied_feedback)
                        .withOpenSettingsButton(R.string.permission_rat
ionale_settings_button_text)
                        .build())


        Dexter.withActivity(this)
                .withPermissions(Manifest.permission.BLUETOOTH,
```

```
                        Manifest.permission.ACCESS_COARSE_LOCATION,
                        Manifest.permission.ACCESS_FINE_LOCATION,
                        Manifest.permission.WRITE_EXTERNAL_STORAGE)
                .withListener(allPermissionsListener)
                .check()
    }

    private fun connectToBluetoothDevice() {
        bluetoothSerial = BluetoothSerial(this, object : BluetoothSeria
l.MessageHandler {
            override fun connected() {
                binding.button.isEnabled = true
                binding.button.text = getString(R.string.record)
                binding.button.setBackgroundResource(R.drawable.ic_btn_
connected)
            }

            override fun disConnected() {
                binding.button.text = getString(R.string.disconnected)
                binding.button.setBackgroundResource(R.drawable.ic_btn_
normal)
            }

            override fun read(x: Float, y: Float, z: Float, temp: Float
, x1: Float, y1: Float, z1: Float, xa: Float, ya: Float, za: Float) {

                runOnUiThread {
                    binding.tvXvalue.text = String.format("% 10.2fg", x
)
                    binding.tvYvalue.text = String.format("% 10.2fg", y
)
                    binding.tvZvalue.text = String.format("% 10.2fg", z
)
                    binding.tvTempValue.text = String.format("% 10.2f℃
", temp)
                }
            }

            override fun read(bufferSize: Int, buffer: ByteArray): Int
{
                return 0
            }
        }, deviceName)
        bluetoothSerial?.connect()
    }
```

```
    private fun getAccuracy() {
        LocationManager(object : LocationManager.PositionListener {
            override fun getAccuracy(accuracy: Float) {
                binding.tvAccuValue.text = String.format("%.2fm", accur
acy)
            }

        }, true).connect()
    }


    private fun setClickListener() {
        binding.button.setOnClickListener {

            if (bluetoothSerial != null && bluetoothSerial!!.isRecordin
g()!!) {
                binding.button.text = getString(R.string.record)
                binding.button.setBackgroundResource(R.drawable.ic_btn_
normal)
            } else {
                binding.button.text = getString(R.string.stop)

                val drawable = AnimationDrawable()
                val image1 = getDrawable(R.drawable.ic_btn_normal)
                val image2 = getDrawable(R.drawable.ic_btn_recoding)
                drawable.addFrame(image1, 1000)
                drawable.addFrame(image2, 1000)
                drawable.isOneShot = false

                binding.button.background = drawable
                drawable.start()
            }


            bluetoothSerial?.setRecord()
        }

    }

     fun onClickListener(){

        var btnPress = false

        button2.setOnTouchListener { v, event ->
```

```
            val mypref = this.getSharedPreferences("mypref", Context.MO
DE_PRIVATE)
            val editor =mypref.edit()
            if(event.getAction()== MotionEvent.ACTION_DOWN){
                btnPress = true
                editor.putString("btnValue",btnPress.toString())
                editor.apply()
            }
            if(event.getAction() == MotionEvent.ACTION_UP){
                btnPress = false
                editor.putString("btnValue",btnPress.toString())
                editor.apply()
            }
            textView.text = btnPress.toString()
            btnPress
        }
    }

}
```

MyApplication.kt

```
package com.typeiii.bth
import android.app.Application
import android.content.Context
class MyApplication: Application() {

    init {
        instance = this
    }

    companion object{

        private var instance: MyApplication? = null

        fun applicationContext() : Context {
            return instance!!.applicationContext
        }

        lateinit var sApplication: Application

        private fun getApplication(): Application? {
            return sApplication
        }
```

```
        fun getContext(): Context? {
            return getApplication()?.applicationContext
        }
    }

    override fun onCreate() {
        super.onCreate()
        sApplication = this

    }

}
```

BluetoothSerial.kt

```
package com.typeiii.bth.helpers

import android.annotation.SuppressLint
import android.bluetooth.BluetoothSocket
import android.bluetooth.BluetoothDevice
import android.support.v4.content.LocalBroadcastManager
import android.content.Intent
import android.os.AsyncTask
import android.bluetooth.BluetoothAdapter
import android.content.IntentFilter
import android.content.BroadcastReceiver
import android.content.Context
import android.util.Log
import java.io.IOException
import java.io.InputStream
import java.io.OutputStream
import java.util.*


class BluetoothSerial(internal var context: Context, internal var messa
geHandler: MessageHandler, devicePrefix: String) {

    internal var connected = false

    internal var bluetoothDevice: BluetoothDevice? = null

    internal var serialSocket: BluetoothSocket? = null

    internal var serialInputStream: InputStream? = null
```

```
    internal var serialOutputStream: OutputStream? = null

    private var serialReader: SerialReader? = null

    internal var connectionTask: AsyncTask<Void, Void, BluetoothDevice>
? = null

    internal var devicePrefix: String


    private val MAX_BYTES = 125

    internal var mConnectedThread: ConnectedThread? = null

    /**
     * Listens for discount message from bluetooth system and restablis
hing a connection
     */
    private val bluetoothReceiver = object : BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) {
            val action = intent.action
            val eventDevice = intent.getParcelableExtra<BluetoothDevice
>(BluetoothDevice.EXTRA_DEVICE)

            if (BluetoothDevice.ACTION_ACL_DISCONNECTED == action) {
                if (bluetoothDevice != null && bluetoothDevice == event
Device) {
                    Log.i(BMX_BLUETOOTH, "Received bluetooth disconnect
 notice")

                    //clean up any streams
                    close()

                    //reestablish connect
                    connect()

                    LocalBroadcastManager.getInstance(context).sendBroa
dcast(Intent(BLUETOOTH_DISCONNECTED))
                }
            }
        }
    }

    init {
        this.devicePrefix = devicePrefix.toUpperCase()
```

```
    }

    fun onPause() {
        context.unregisterReceiver(bluetoothReceiver)
    }

    fun onResume() {
        //listen for bluetooth disconnect
        val disconnectIntent = IntentFilter(BluetoothDevice.ACTION_ACL_
DISCONNECTED)
        context.registerReceiver(bluetoothReceiver, disconnectIntent)

        //reestablishes a connection is one doesn't exist
        if (!connected) {
            connect()
        } else {
            val intent = Intent(BLUETOOTH_CONNECTED)
            LocalBroadcastManager.getInstance(context).sendBroadcast(in
tent)
        }
    }


    /**
     * Initializes the bluetooth serial connections, uses the LocalBroa
dcastManager when
     * connection is established
     *
     */
    fun connect() {

        if (connected) {
            Log.e(BMX_BLUETOOTH, "Connection request while already conn
ected")
            return
        }

        if (connectionTask != null && connectionTask!!.status == AsyncT
ask.Status.RUNNING) {
            Log.e(BMX_BLUETOOTH, "Connection request while attempting c
onnection")
            return
        }

        val bluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
        if (bluetoothAdapter == null || !bluetoothAdapter.isEnabled) {
```

```
            return
        }

        val pairedDevices = ArrayList(bluetoothAdapter.bondedDevices)
        if (pairedDevices.size > 0) {
            bluetoothAdapter.cancelDiscovery()

            /**
             * AsyncTask to handle the establishing of a bluetooth conn
ection
             */
            connectionTask = @SuppressLint("StaticFieldLeak")
            object : AsyncTask<Void, Void, BluetoothDevice>() {

                internal var MAX_ATTEMPTS = 30

                internal var attemptCounter = 0

                override fun doInBackground(vararg params: Void): Bluet
oothDevice? {
                    while (!isCancelled) { //need to kill without calli
ng onCancel

                        for (device in pairedDevices) {
                            if (device.name.toUpperCase().startsWith(de
vicePrefix)) {
                                Log.i(BMX_BLUETOOTH, attemptCounter.toS
tring() + ": Attempting connection to " + device.name)

                                try {

                                    try {
                                        // Standard SerialPortService I
D

                                        val uuid = UUID.fromString("000
01101-0000-1000-8000-00805F9B34FB")

                                        serialSocket = device.createRfc
ommSocketToServiceRecord(uuid)
                                    } catch (ce: Exception) {
                                        serialSocket = connectViaReflec
tion(device)
                                    }

                                    //setup the connect streams
                                    serialSocket!!.connect()
```

```
                                    serialInputStream = serialSocket!!.
inputStream
                                    serialOutputStream = serialSocket!!
.outputStream

                                    connected = true
                                    Log.i(BMX_BLUETOOTH, "Connected to
" + device.name)

                                    return device
                                } catch (e: Exception) {
                                    serialSocket = null
                                    serialInputStream = null
                                    serialOutputStream = null
                                    Log.i(BMX_BLUETOOTH, e.message)
                                }

                            }
                        }

                        try {
                            attemptCounter++
                            if (attemptCounter > MAX_ATTEMPTS)
                                this.cancel(false)
                            else
                                Thread.sleep(1000)
                        } catch (e: InterruptedException) {
                            break
                        }

                    }

                    Log.i(BMX_BLUETOOTH, "Stopping connection attempts"
)

                    val intent = Intent(BLUETOOTH_FAILED)
                    LocalBroadcastManager.getInstance(context).sendBroa
dcast(intent)

                    return null
                }

                override fun onPostExecute(result: BluetoothDevice) {
                    super.onPostExecute(result)

                    bluetoothDevice = result
```

```
                    //start thread responsible for reading from inputst
ream
                    serialReader = SerialReader()
                    serialReader!!.start()

                    //send connection message
                    val intent = Intent(BLUETOOTH_CONNECTED)
                    LocalBroadcastManager.getInstance(context).sendBroa
dcast(intent)

                    if (serialSocket != null) {
                        mConnectedThread = ConnectedThread(serialSocket
!!, messageHandler)
                        messageHandler.connected()
                        mConnectedThread!!.start()
                    }

                }

            }
            connectionTask!!.execute()
        }
    }

    @Throws(Exception::class)
    private fun connectViaReflection(device: BluetoothDevice): Bluetoot
hSocket {
        val m = device.javaClass.getMethod("createRfcommSocket", *array
Of<Class<*>>(Int::class.javaPrimitiveType!!))
        return m.invoke(device, 1) as BluetoothSocket
    }

    @Throws(IOException::class)
    fun available(): Int {
        if (connected)
            return serialInputStream!!.available()

        throw RuntimeException("Connection lost, reconnecting now.")
    }

    @Throws(IOException::class)
    fun read(): Int {
        if (connected)
            return serialInputStream!!.read()
```

```
            throw RuntimeException("Connection lost, reconnecting now.")
        }

        @Throws(IOException::class)
        fun read(buffer: ByteArray): Int {
            if (connected)
                return serialInputStream!!.read(buffer)

            throw RuntimeException("Connection lost, reconnecting now.")
        }

        @Throws(IOException::class)
        fun read(buffer: ByteArray, byteOffset: Int, byteCount: Int): Int {
            if (connected)
                return serialInputStream!!.read(buffer, byteOffset, byteCou
nt)

            throw RuntimeException("Connection lost, reconnecting now.")
        }

        @Throws(IOException::class)
        fun write(buffer: ByteArray) {
            if (connected)
                serialOutputStream!!.write(buffer)

            throw RuntimeException("Connection lost, reconnecting now.")
        }

        @Throws(IOException::class)
        fun write(oneByte: Int) {
            if (connected)
                serialOutputStream!!.write(oneByte)

            throw RuntimeException("Connection lost, reconnecting now.")
        }

        @Throws(IOException::class)
        fun write(buffer: ByteArray, offset: Int, count: Int) {
            serialOutputStream!!.write(buffer, offset, count)

            throw RuntimeException("Connection lost, reconnecting now.")
        }

        private inner class SerialReader : Thread() {

            internal var buffer = ByteArray(MAX_BYTES)
```

```
        internal var bufferSize = 0

        override fun run() {
            Log.i("serialReader", "Starting serial loop")
            while (!isInterrupted) {
                try {

                    /*
                     * check for some bytes, or still bytes still left
in
                     * buffer
                     */
                    if (available() > 0) {

                        val newBytes = read(buffer, bufferSize, MAX_BYT
ES - bufferSize)
                        if (newBytes > 0)
                            bufferSize += newBytes

                        //Log.d(BMX_BLUETOOTH, "read $newBytes")
                    }

                    if (bufferSize > 0) {
                        val read = messageHandler.read(bufferSize, buff
er)

                        // shift unread data to start of buffer
                        if (read > 0) {
                            var index = 0
                            for (i in read until bufferSize) {
                                buffer[index++] = buffer[i]
                            }
                            bufferSize = index
                        }
                    } else {

                        try {
                            Thread.sleep(10)
                        } catch (ie: InterruptedException) {
                            break
                        }

                    }
                } catch (e: Exception) {
```

```
                    Log.e(BMX_BLUETOOTH, "Error reading serial data", e
)
                }

            }
            Log.i(BMX_BLUETOOTH, "Shutting serial loop")
        }
    }

    /**
     * Reads from the serial buffer, processing any available messages.
  Must return the number of bytes
     * consumer from the buffer
     *
     * by @author jpetrocik
     */
    interface MessageHandler {
        fun connected()
        fun disConnected()
        fun read(bufferSize: Int, buffer: ByteArray): Int
        fun read(x: Float, y:Float, z:Float, temp:Float , x1: Float, y1
: Float, z1: Float, xa: Float, ya: Float, za: Float)
    }

    fun close() {

        connected = false
        messageHandler.disConnected()

        if (serialReader != null) {
            serialReader!!.interrupt()

            try {
                serialReader!!.join(1000)
            } catch (ie: InterruptedException) {
            }

        }

        try {
            serialInputStream!!.close()
        } catch (e: Exception) {
            Log.e(BMX_BLUETOOTH, "Failed releasing inputstream connecti
on")
        }
```

```
        try {
            serialOutputStream!!.close()
        } catch (e: Exception) {
            Log.e(BMX_BLUETOOTH, "Failed releasing outputstream connect
ion")
        }

        try {
            serialSocket!!.close()
        } catch (e: Exception) {
            Log.e(BMX_BLUETOOTH, "Failed closing socket")
        }

        Log.i(BMX_BLUETOOTH, "Released bluetooth connections")

    }

    fun setRecord() {
        mConnectedThread?.setRecord()
    }

    fun isRecording(): Boolean? {
        return  mConnectedThread?.getRecord()
    }

    companion object {
        private val BMX_BLUETOOTH = "BMXBluetooth"

        var BLUETOOTH_CONNECTED = "bluetooth-connection-started"

        var BLUETOOTH_DISCONNECTED = "bluetooth-connection-lost"

        var BLUETOOTH_FAILED = "bluetooth-connection-failed"
    }

}
```

ConnectedThred.kt

```
package com.typeiii.bth.helpers

import android.bluetooth.BluetoothSocket
import android.os.Bundle
import android.widget.Toast
import java.io.IOException
```

```
import java.io.InputStream
import java.io.OutputStream
import java.util.*

internal class ConnectedThread(private val mmSocket: BluetoothSocket,
                               var messageHandler: BluetoothSerial.Mess
ageHandler) : Thread() {
    private val mmInStream: InputStream?
    private val mmOutStream: OutputStream?

    private val fData = FloatArray(31)
    private var strDate: String? = null
    private var strTime: String? = null


    private val queueBuffer = LinkedList<Byte>()
    private val packBuffer = ByteArray(11)

    private var isRecord = false
    private val RecordTimeDifference = 0.5 // seconds

    init {
        var tmpIn: InputStream? = null
        var tmpOut: OutputStream? = null

        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = mmSocket.inputStream
            tmpOut = mmSocket.outputStream
        } catch (e: IOException) {
        }

        mmInStream = tmpIn
        mmOutStream = tmpOut
    }

    override fun run() {
        val tempInputBuffer = ByteArray(1024)
        var acceptedLen: Int
        var sHead: Byte
        // Keep listening to the InputStream while connected
        var lLastTime = System.currentTimeMillis() // 获取开始时间
        var lLastRecoderTime = System.currentTimeMillis()
        while (true) {

            try {
```

```
                acceptedLen = mmInStream!!.read(tempInputBuffer)
                //Log.d("BTL1", "" + acceptedLen)
                for (i in 0 until acceptedLen) queueBuffer.add(tempInpu
tBuffer[i])


                while (queueBuffer.size >= 11) {
                    if (queueBuffer.poll() != 0x55.toByte()) continue
                    sHead = queueBuffer.poll()
                    for (j in 0..8) packBuffer[j] = queueBuffer.poll()

                    when (sHead) {
                    //
                        0x50.toByte() -> {
                            val ms = packBuffer[7].toInt() shl 8 or (pa
ckBuffer[6].toInt() and 0xff)
                            strDate = String.format("20%02d-%02d-
%02d", packBuffer[0], packBuffer[1], packBuffer[2])
                            strTime = String.format(" %02d:%02d:%02d.%0
3d", packBuffer[3], packBuffer[4], packBuffer[5], ms)
                        }
                        0x51.toByte() -> {
                            fData[0] = (packBuffer[1].toInt() shl 8 or
(packBuffer[0].toInt() and 0xff)) / 32768.0f * 16
                            fData[1] = (packBuffer[3].toInt() shl 8 or
(packBuffer[2].toInt() and 0xff)) / 32768.0f * 16
                            fData[2] = (packBuffer[5].toInt() shl 8 or
(packBuffer[4].toInt() and 0xff)) / 32768.0f * 16
                            fData[17] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)) / 100.0f
                        }
                        0x52.toByte() -> {
                            fData[3] = (packBuffer[1].toInt() shl 8 or
(packBuffer[0].toInt() and 0xff)) / 32768.0f * 2000
                            fData[4] = (packBuffer[3].toInt() shl 8 or
(packBuffer[2].toInt() and 0xff)) / 32768.0f * 2000
                            fData[5] = (packBuffer[5].toInt() shl 8 or
(packBuffer[4].toInt() and 0xff)) / 32768.0f * 2000
                            fData[17] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)) / 100.0f
                        }
                        0x53.toByte() -> {
                            fData[6] = (packBuffer[1].toInt() shl 8 or
(packBuffer[0].toInt() and 0xff)) / 32768.0f * 180
```

```
                            fData[7] = (packBuffer[3].toInt() shl 8 or
(packBuffer[2].toInt() and 0xff)) / 32768.0f * 180
                            fData[8] = (packBuffer[5].toInt() shl 8 or
(packBuffer[4].toInt() and 0xff)) / 32768.0f * 180
                            fData[17] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)) / 100.0f
                        }
                        0x54.toByte() //magnetic field
                        -> {
                            fData[9] = (packBuffer[1].toInt() shl 8 or
(packBuffer[0].toInt() and 0xff)).toFloat()
                            fData[10] = (packBuffer[3].toInt() shl 8 or
 (packBuffer[2].toInt() and 0xff)).toFloat()
                            fData[11] = (packBuffer[5].toInt() shl 8 or
 (packBuffer[4].toInt() and 0xff)).toFloat()
                            fData[17] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)) / 100.0f
                        }
                        0x55.toByte() //port
                        -> {
                            fData[12] = (packBuffer[1].toInt() shl 8 or
 (packBuffer[0].toInt() and 0xff)).toFloat()
                            fData[13] = (packBuffer[3].toInt() shl 8 or
 (packBuffer[2].toInt() and 0xff)).toFloat()
                            fData[14] = (packBuffer[5].toInt() shl 8 or
 (packBuffer[4].toInt() and 0xff)).toFloat()
                            fData[15] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)).toFloat()
                        }
                        0x56.toByte() //Pressure, height
                        -> {
                            fData[16] = (packBuffer[3].toLong() shl 24
or (packBuffer[2].toLong() shl 16) or (packBuffer[1].toLong() shl 8) or
 packBuffer[0].toLong()).toFloat()
                            fData[17] = (packBuffer[7].toLong() shl 24
or (packBuffer[6].toLong() shl 16) or (packBuffer[5].toLong() shl 8) or
 packBuffer[4].toLong()).toFloat()
                            fData[17] /= 100f
                        }
                        0x57.toByte() //Latitude and longitude
                        -> {
                            val longitude = packBuffer[3].toLong() shl
24 or (packBuffer[2].toLong() shl 16) or (packBuffer[1].toLong() shl 8)
 or packBuffer[0].toLong()
```

```
                            fData[18] = (longitude.toFloat() / 10000000
 + (longitude % 10000000).toFloat().toDouble() / 100000.0 / 60.0).toFlo
at()
                            val latitude = packBuffer[7].toLong() shl 2
4 or (packBuffer[6].toLong() shl 16) or (packBuffer[5].toLong() shl 8)
or packBuffer[4].toLong()
                            fData[19] = (latitude.toFloat() / 10000000
+ (latitude % 10000000).toFloat().toDouble() / 100000.0 / 60.0).toFloat
()
                        }
                        0x58.toByte() //Altitude, heading, ground speed
                        -> {
                            fData[20] = ((packBuffer[3].toLong() shl 24
 or (packBuffer[2].toLong() shl 16) or (packBuffer[1].toLong() shl 8) o
r packBuffer[0].toLong()) / 10).toFloat()
                            fData[21] = ((packBuffer[5].toInt() shl 8 o
r (packBuffer[4].toInt() and 0xff)) / 10).toFloat()
                            fData[22] = ((packBuffer[7].toInt() shl 8 o
r (packBuffer[6].toInt() and 0xff)) / 1000).toFloat()
                        }
                        0x59.toByte() //Quaternion
                        -> {
                            fData[23] = (packBuffer[1].toInt() shl 8 or
 (packBuffer[0].toInt() and 0xff)) / 32768.0f
                            fData[24] = (packBuffer[3].toInt() shl 8 or
 (packBuffer[2].toInt() and 0xff)) / 32768.0f
                            fData[25] = (packBuffer[5].toInt() shl 8 or
 (packBuffer[4].toInt() and 0xff)) / 32768.0f
                            fData[26] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)) / 32768.0f
                        }
                        0x5a.toByte() //Number of satellites
                        -> {
                            fData[27] = (packBuffer[1].toInt() shl 8 or
 (packBuffer[0].toInt() and 0xff)) / 32768.0f
                            fData[28] = (packBuffer[3].toInt() shl 8 or
 (packBuffer[2].toInt() and 0xff)) / 32768.0f
                            fData[29] = (packBuffer[5].toInt() shl 8 or
 (packBuffer[4].toInt() and 0xff)) / 32768.0f
                            fData[30] = (packBuffer[7].toInt() shl 8 or
 (packBuffer[6].toInt() and 0xff)) / 32768.0f
                        }
                    }

                }
```

```
                val lTimeNow = System.currentTimeMillis() // Get start
time
                if (lTimeNow - lLastTime > 80) {
                    lLastTime = lTimeNow
                    val bundle = Bundle()
                    bundle.putFloatArray("Data", fData)
                    bundle.putString("Date", strDate)
                    bundle.putString("Time", strTime)

                    messageHandler.read(fData[0], fData[1], fData[2], f
Data[17],fData[3], fData[4], fData[5], fData[6], fData[7], fData[8])
                    if (isRecord && (lTimeNow - lLastRecoderTime > (Rec
ordTimeDifference * 1000))) {
                        Recorder.getInstance().saveToTextFile(fData[0],
 fData[1], fData[2], fData[17], fData[3], fData[4], fData[5], fData[6],
 fData[7], fData[8])
                        lLastRecoderTime = lTimeNow
                    }
                }

            } catch (e: IOException) {
                //connectionLost()

                break
            }

        }
    }

    fun write(buffer: ByteArray) {
        try {
            mmOutStream!!.write(buffer)
        } catch (e: IOException) {
        }

    }

    fun cancel() {
        try {
            mmSocket.close()
        } catch (e: IOException) {
        }

    }

    fun setRecord() {
```

```
        isRecord = !isRecord
    }

    fun getRecord(): Boolean {
        return isRecord
    }
}
```

LocationManager.kt

```
package com.typeiii.bth.helpers

import android.content.Context
import android.location.Location
import android.location.LocationListener
import android.location.LocationManager
import android.os.Bundle
import android.os.Looper
import android.util.Log
import com.typeiii.bth.MyApplication

class LocationManager(var listener: PositionListener, var needAccuracy:
 Boolean) {

    interface PositionListener {
        fun getLatLong(latitude:Double, longitude:Double) { }
        fun getAccuracy(accuracy:Float) { }
    }

    fun connect() {
        val locationListener = object: LocationListener {
            override fun onStatusChanged(p0: String?, p1: Int, p2: Bund
le?) {
            }

            override fun onProviderEnabled(p0: String?) {
            }

            override fun onProviderDisabled(p0: String?) {
            }

            override fun onLocationChanged(location: Location?) {

                if (location != null){
```

```
                    listener.getLatLong(location.latitude, location.lon
gitude)
                }
            }
        }
        val lm = MyApplication.getContext()?.getSystemService(Context.L
OCATION_SERVICE)
                as LocationManager
        try {
            val location = lm.getLastKnownLocation(LocationManager.GPS_
PROVIDER)
            lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 200
0, 1f,
                    locationListener, Looper.getMainLooper())

            if (location != null) {
                listener.getLatLong(location.latitude, location.longitu
de)
                if (needAccuracy) listener.getAccuracy(location.accurac
y)
            }
        } catch (e: SecurityException) {
            Log.e("ERROR", "location permission error")
        } catch (e: NullPointerException) {
            e.printStackTrace()
        }
    }
}
```

Recorder.kt

```
package com.typeiii.bth.helpers

import android.content.Context
import android.content.Intent.getIntent
import android.content.Intent.parseIntent
import android.os.Environment
import android.widget.EditText
import java.text.SimpleDateFormat
import java.util.*
import java.io.*
import android.content.SharedPreferences
import com.typeiii.bth.MainActivity
import com.typeiii.bth.MyApplication
import com.typeiii.bth.R
```

```
import kotlinx.android.synthetic.main.activity_main.*

class Recorder {

    companion object {
        private var instance : Recorder? = null

        fun  getInstance(): Recorder {
            if (instance == null)
                instance = Recorder()

            return instance!!
        }
    }

    private val fileName: String
        get() = "BTH_$currentDate.txt"

    private val currentDate: String
        get() {
            return SimpleDateFormat("ddMyyyy", Locale.US).format(Date()
)
        }

    private val currentTime: String
        get() {
            return SimpleDateFormat("H:mm:ss", Locale.US).format(Date()
)
        }

    private var locationLatitude: Double = 0.0
    private var locationLongitude: Double = 0.0

    private fun btnCame():String{
        val mypref = MyApplication.getContext()?.getSharedPreferences("
mypref", Context.MODE_PRIVATE)
        val send = mypref?.getString("btnValue","false")
        return send.toString()
    }


    private fun convertValesToString( x: Float, y: Float, z: Float, tem
p: Float, x1: Float, y1: Float, z1: Float, xa: Float, ya: Float, za: Fl
oat) : String {
        val xStr = String.format("% 10.2f", x)
        val yStr = String.format("% 10.2f", y)
```

```
        val zStr = String.format("% 10.2f", z)
        //val tempStr = String.format("% 10.2f℃", temp)
        val x1Str = String.format("% 10.2f", x1)
        val y1Str = String.format("% 10.2f", y1)
        val z1Str = String.format("% 10.2f", z1)
        val xaStr = String.format("% 10.2f", xa)
        val yaStr = String.format("% 10.2f", ya)
        val zaStr = String.format("% 10.2f", za)

        return "$currentTime | X: $xStr | Y: $yStr | Z: $zStr | X1: $x1
Str | Y1: $y1Str | Z1: $z1Str | Xa: $xaStr | Ya: $yaStr | Za: $zaStr" +
                "| Latitude: $locationLatitude | Longitude: $locationLo
ngitude " + "| Pothole: ${btnCame()}|"
    }

    fun saveToTextFile( x: Float, y: Float, z: Float, temp: Float, x1:
Float, y1: Float, z1: Float, xa: Float, ya: Float, za: Float) {

        getLocation()

        try {

            val path = Environment.getExternalStorageDirectory().absolu
tePath + "/BTH"
            val folder = File(path)
            if (!folder.exists()) folder.mkdirs()

            val file = File(folder, fileName)
            if (!file.exists()) file.createNewFile()
            //open file for writing
            val out = OutputStreamWriter(FileOutputStream(file, true))

            out.write(convertValesToString(x, y, z, temp, x1, y1, z1, x
a, ya, za))
            out.write("\n")

            //close file
            out.close()

        } catch (e: java.io.IOException) {
            e.printStackTrace()
        }

    }

    private fun getLocation() {
```

```
        LocationManager(object : LocationManager.PositionListener{
            override fun getLatLong(latitude: Double, longitude: Double
) {
                locationLatitude = latitude
                locationLongitude = longitude
            }

        }, false).connect()
    }
}
```