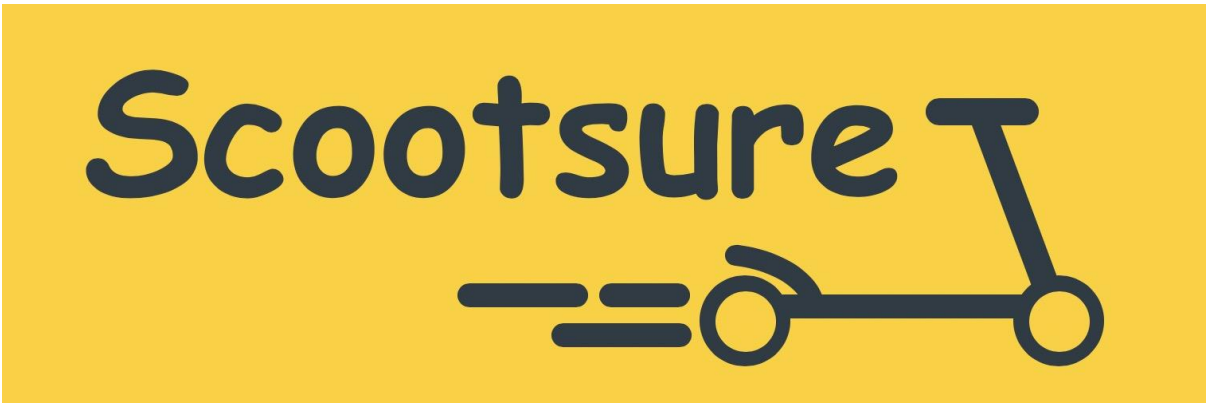


ScootSure

Making Electric Scooters Safer & Insurable Using AI

Author: Niall Meade

Year: 4th Year

School: Ardscoil Rís, Limerick

Date: January 2020

Contents

Comments Page

Introduction

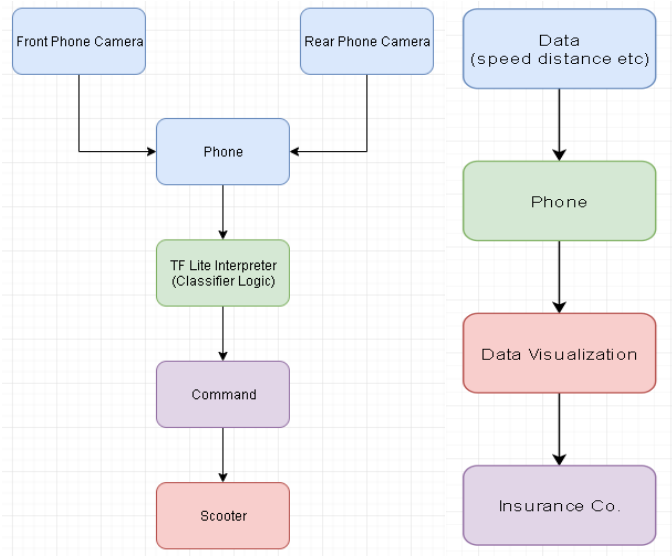
The idea came after my friend fell off his scooter and injured himself as he travelled on stones and from reading an article on Medium about scooters legislation. This put electric scooters on my radar and specifically how dangerous they were. With this in mind, I began to think of different ways the scooter's safety could be improved. What I came up with was ScootSure, a system that would be installed in the scooter to prevent usage of the scooter without a helmet and would also stop the user driving on unsuitable terrain such as grass, sand or loose stones. While using the scooter I found I would lose control while on grass or stones etc. The system would recognise if the driver had a helmet on or not using a convolutional neural network, the type of terrain would be classified using a convolutional neural network also. After further researching electric scooters, and specifically electric scooters in Ireland I found that they were required to have insurance since they had an electric motor. Despite this insurance companies in Ireland are refusing to insure scooters due to lack of understanding of the risk of them and lack of legislation. I thought that my project ScootSure could also be helpful in allowing the insurance companies to gain a greater understanding of the scooters. This is done by collecting data from the scooters on the frequency of journeys.

Proposal

My project aims to create an electric scooter AI based platform to help make scooters safer and more insurable. Electric scooters are popular in cities. These scooters have sparked safety concerns because of the lack of legislation around them and concerns relating to insurance. ScootSure will help make these vehicles safer and more insurable in the following ways

1. Helmet classification
2. Terrain classification
3. Facial recognition or driving pattern analysis Anti-Theft System (If time allows)
4. Data presentation (for insurance companies)

The idea came after my friend fell off his scooter and injured himself as he travelled on stones and from reading an article on Medium about scooters legislation. Main problem was insurance companies refused to insure them due to lack of regulation and lack of data regarding scooters. This SccootSure platform will help with both, data can be collected for use by the insurance industry, while ensuring the driver wears a helmet and stops the scooter on inappropriate terrain. If someone steals the scooter it won't drive, if the driver doesn't have a helmet they won't be allowed drive and if unsuitable terrain is ahead they will be stopped

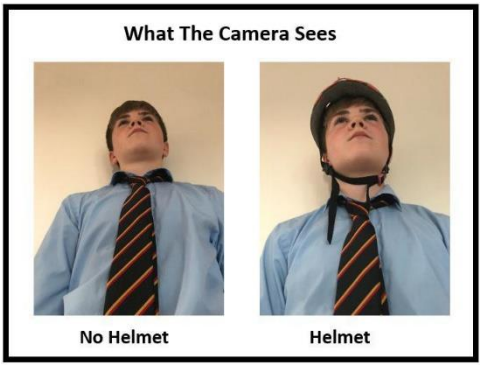


Project Plan

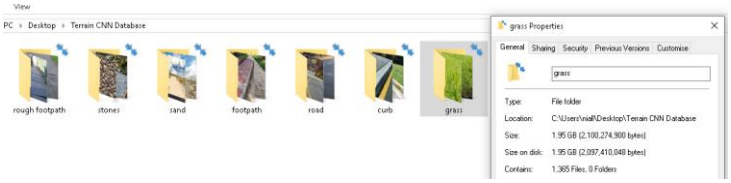
- ScootSure will be developed as an app using the Ionic-cross-platform app-development framework.

- I will train CNN models using keras (helmet, terrain classifiers). These models will then be converted to TfLite for use on my phone.
- I will use the Xiaomi-M365 Scooter bluetooth development kit to send and receive commands to and from my scooter.
- My app will take a picture of the driver and terrain ahead, it then uses TF-Lite-interpreter to run these pictures through the classifiers

- After the models return results the app will react accordingly, e.g don't start if no helmet
- This command is then sent to the scooter to be executed



I'm developing my project using Tensorflow machine-learning library for neural-networks(a neural-network is a way of making a computer think like a human and be able to notice complex patterns), Tensorflow is cross-platform, E.g Android. For this project I have been researching edge detection. This is used in convolutional-neural-networks(CNN) to get an idea of shape for the object being classified. I've researched transfer learning which is more efficient and accurate. Transfer learning takes knowledge gained from solving a different yet similar problem and adapting it to a new model. I've obtained a dataset to train my terrain classifier. I used a web-scraper to download images from google search. I will implement my system on a scooter with my phone as the camera which classifies the images locally using TF-Lite. This is a version of Tensorflow that runs on phones. The neural network is trained on a PC and then the TF-Lite converter is used to convert the model into an optimized format for use on the phone with TF-Lite



Project Objectives/Steps

1. Train my CNN for the terrain classifier using the dataset I have
2. Obtain a dataset for the helmet classifier and train another CNN
3. Convert both models to TF-Lite format for use on phone
4. Implement this system on an electric scooter for presentation at the RDS
5. Look into how the scooter data can be presented and used

Proposal Conclusion

The legalisation of electric scooters has become more topical over the last year due to people wanting to try and avoid the early morning traffic. These scooters can often travel at 30km/h+ making them a threat to safety. Because of this the government has made them illegal because of lack of insurance. This system is a step towards the legalisation of scooters.

Aims

1. Investigate electric scooter legislation in Ireland and insurance for them.
2. Develop two convolutional neural networks, one to classify terrain and another to classify if the driver of the scooter is wearing a helmet or not.
3. Investigate how my project could help insurance companies gain a better understanding of electric scooters.
4. Develop a web interface/dashboard for an insurance company to display all their customers statistics regarding their journeys on their scooter.

App Design

When I set out to start creating ScootSure, I wanted to understand the logic behind it. To do this I decided I would make some diagrams to explain what will be happening in the background. I did some research and I found flow and Jacobson use case diagrams. Flow diagrams are used to explain how the code or app flows in terms of different states. Jacobson use case diagrams are used mainly to show what a user or the app can do. This helps understand functionality needed for the app and user.

Helmet Classifier

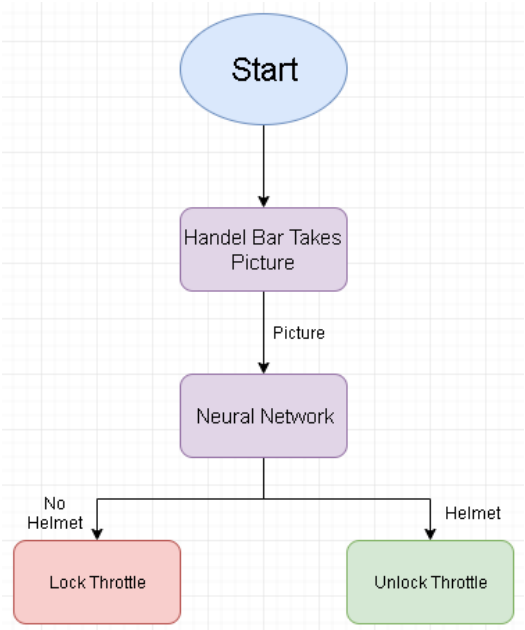


fig x. Helmet classifier flow/state diagram

Save A Journey

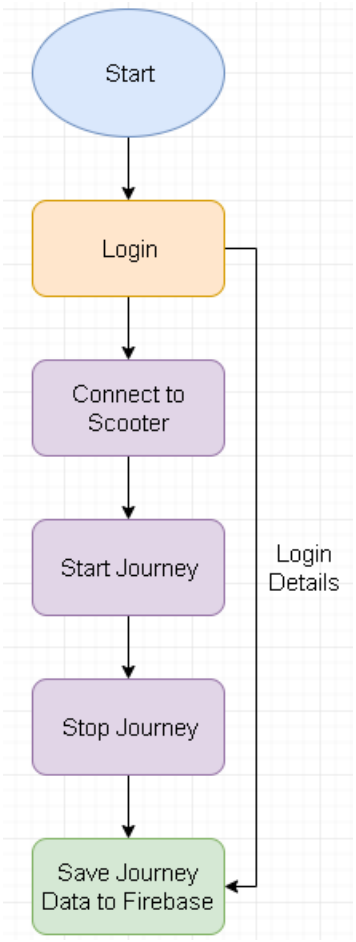


fig x. Save a journey flow/state diagram

View User on Insurance Web Page

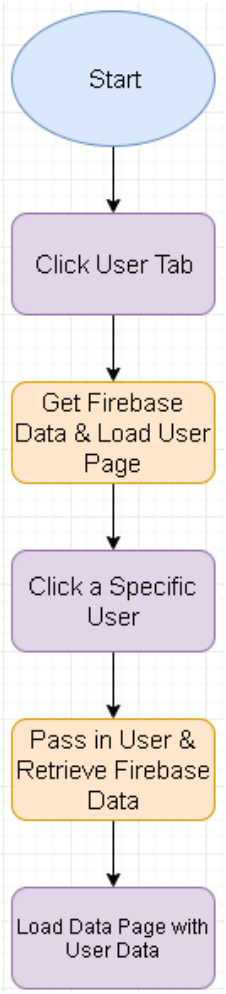
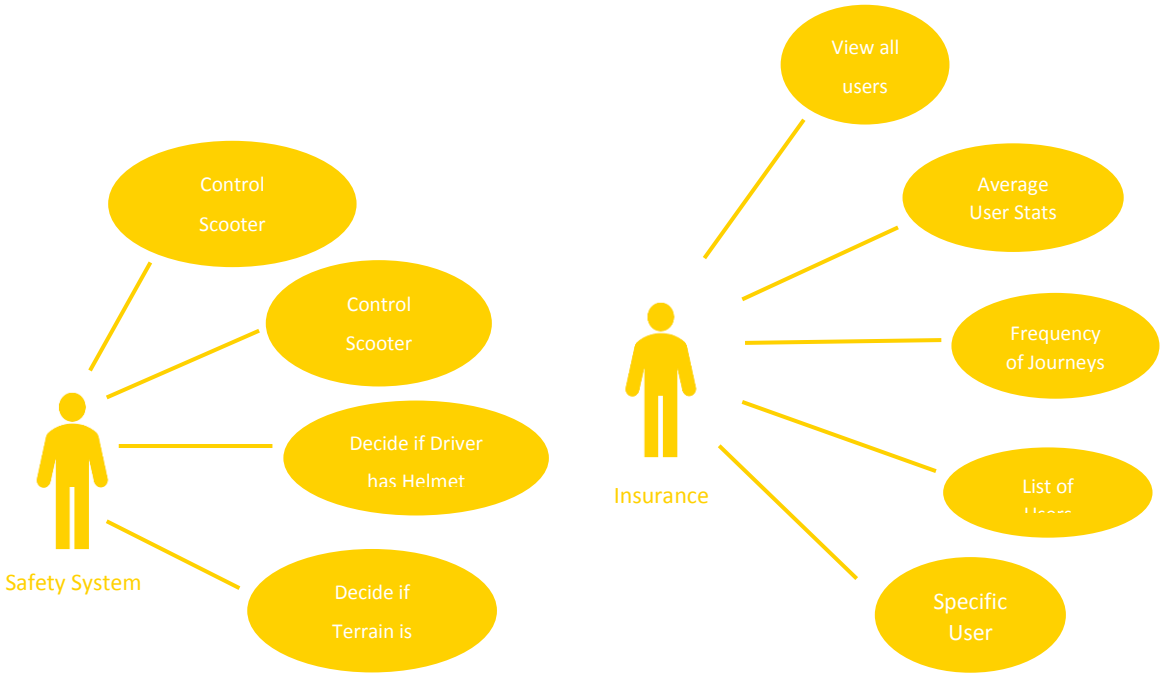


fig x. View user on insurance web page flow/state diagram

Jacobson Use Case Diagrams

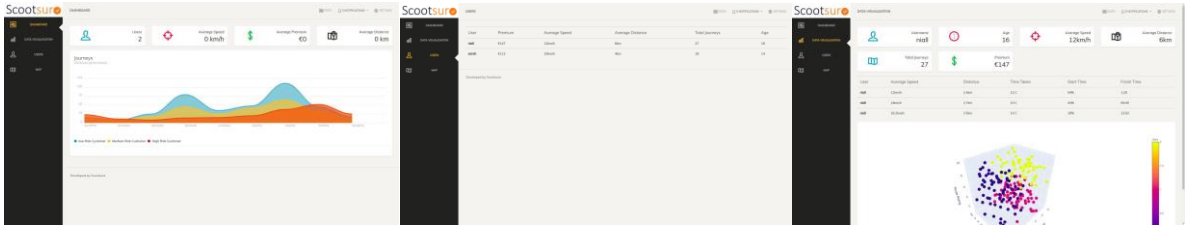
These use case diagrams helped me to understand what functionality both the insurance company and scooter safety system should have when I started development. These diagrams helped me to focus on the way I was developing my app and make sure I stayed on the right track.



Technology

Vue JS

Vue JS is an open source JavaScript framework for building single page web applications and interfaces. In my project I used it to build my web interface for the insurance side of my project. The way Vue works is it compiles all your code into a single page on the “app.vue” page. I used router to direct the “app.vue” page to the various pages in the interface. I also used node package manager (npm) with Vue to install different library’s such as Firebase.



Firebase

Firebase is a database/mobile and web app development platform that I will be using for my back-end for user data storage. I use Firebase in my insurance page web app and my android app which is used by the user to log journeys. Setting firebase up in the android app is very straight forward as there is a tool in the SDK that connects to your firebase/google account and links your project. This means you just have to declare firebase as “db” in the

pages of your app that you want to use it in. Using firebase in Vue JS is less straight forward but still intuitive. Here I created two Vue files in my components folder, the first imports firebase and the second file has the API key and all the relevant information that links to your project. Your API key is unique to you. Firebase works using collections with documents within those collections and those documents are populated with various fields of strings, integers, booleans etc. In my database I've two collections, journeys and users. The users collection contains every user and a master user document which makes it easy to get general statistics on the insurance dashboard. The journeys collection is a list of all journeys that have been saved by the various users. The documents in the journeys collection are linked to a user document in the users collection by their username.

```
var config = {
  apiKey: "<API_KEY>",
  authDomain: "<PROJECT_ID>.firebaseapp.com",
  databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
  storageBucket: "<BUCKET>.appspot.com",
};
firebase.initializeApp(config);
```

```
created(){
 db.collection("users").where("master", "==", true).get().then((querySnapshot) => {
     querySnapshot.forEach((doc) => {
         console.log(doc.id, " => ", doc.data());
         this.statsData = doc.data();
         this.localtotalUsers = this.statsData.totalUsers;
         this.localaveragePremium = this.statsData.averagePremium;
         this.localaverageDistance = this.statsData.averageDistance;
         this.localaverageSpeed = this.statsData.averageSpeed;
         console.log("data collected");
     });
 });
```

```
     db.collection("journeys").where("user", "==", localStorage.user).get().then((querySnapshot) => {
    querySnapshot.forEach((doc) => {
        this.journeys.push(doc.data());
        console.log(doc.id, " => ", doc.data());
        console.log(this.journeys);
    });
});
```

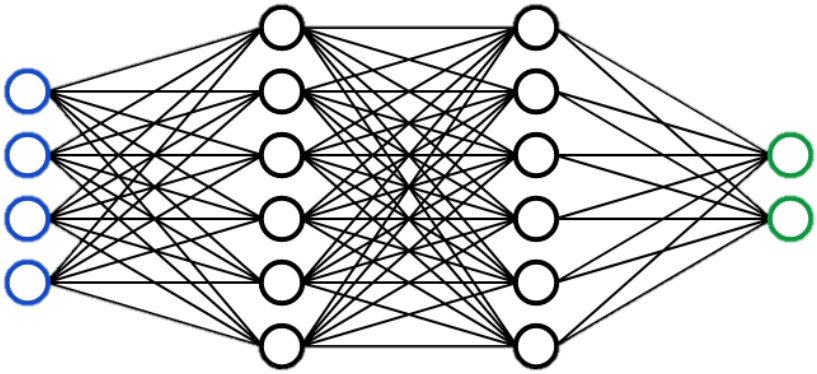
BLE

BLE stands for Bluetooth Low Energy. BLE is a form of wireless communication designed for short range and small data communications between devices. BLE is different from normal Bluetooth as it prioritises battery life over speed and size of the data being transmitted. BLE is used in the Xiaomi M365 scooter to conserve battery life, this works perfectly as the size of the data the

scooter has to transmit is very small. BLE is a one way communication. The device that is broadcasting data is called a BLE beacon. The beacon broadcasts data at regular intervals. In the case of the scooter, the scooter is the BLE beacon and broadcasts it's data such as speed and distance at regular intervals for devices like smartphones to pick up on this data

Neural Networks

Neural networks are a form of artificial intelligence that was partially inspired by the biological neural network that are in our brains as well as other animals. A neural network can be trained with a given dataset to be able to identify patterns and classify the given input to an output. They consist of an input layer, output layer and multiple hidden layers. There are various types of neural networks, each one is better suited to solving a particular problem. In my project I used convolutional neural network models which are suited for classifying images. In a convolutional layer a kernel/filter works it's way through the image with essentially a view finder. For example it might look at 5 by 5 section of pixels at a time. It also has a stride length, this decides how many pixels it moves at a time and decides how much overlap there is. Convolutional networks are different from other networks because of this kernel, this method is best for images because it loses no data about the image versus the method of taking the picture as a matrix of pixel values and placing them in a vector which results in loss of data.



Android SDK

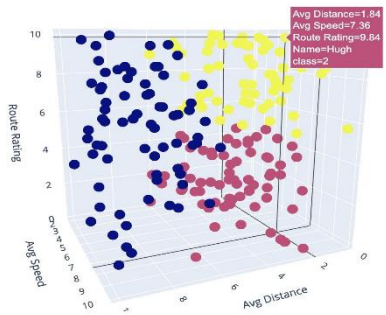
I used the Android SDK (Software Development Kit) to create my android app for use with the scooter in my project. This SDK was designed for the use of making android apps. These apps can be written in Java, Kotlin or C++. It also uses XML files to layout various user inputs and outputs such as buttons and text views.

Libs

Libraries are files of code that were made to be shared. Static libraries differ slightly from normal libraries because a directory to the library must be contained in the program wishing to use it.

K-Means Algorithm

In my web app I wanted to put users in a category to try and predict their premium in some way. I decided to use a k means cluster algorithm to predict where each user would fall, i.e. which cluster they would be in or how high risk of a scooter driver they are. The way a k-means clustering algorithm works is it takes k as the number of clusters in needs to make, in this case 3 (low, medium and high risk). It makes k random reference points (a blue, yellow and red point), each user data point is then assigned to the closest random point. The next stage takes the average position of all the blue points and creates a new blue reference point, this is done for all 3 reference points in my case. Each data point is then re-assigned to the closest reference point, this process is continuously looped through until the variance or ratio of each cluster to the next is of appropriate proportion. I used plotly to create an embedded frame of a 3d scatter plot on my web app. This is linked in the html code of the data visualisation page.



Prototyping

Getting Data from Firebase (Insurance Web App)

The main functionality of the insurance web app is to display data from the users and their journeys. To give the Vue app Firebase functionality I had to install it using node package manager or npm.

```
sudo npm install firebase –save
```

Next, I had to setup Firebase in the app with

```
created(){
 db.collection("users").where("master", "==", true).get().then((querySnapshot) => {
     querySnapshot.forEach((doc) => {
         console.log(doc.id, " => ", doc.data());
         this.statsData = doc.data();
         this.localtotalUsers = this.statsData.totalUsers;
         this.localaveragePremium = this.statsData.averagePremium;
         this.localaverageDistance = this.statsData.averageDistance;
         this.localaverageSpeed = this.statsData.averageSpeed;
         console.log("data collected");
     });
 });
```

the api key and import the relevant libraries. I created two vue.js files, firebaseConfig.vue and firebaseinit.vue. “firebaseinit.vue” imports all the relevant libraries within the firebase library to allow all of them to be imported in each of the pages code with one line of code to make everything look cleaner. “firebaseConfig.vue” contains the API key and other info that directs the app to my firebase project when firebase is called within the app. On the dashboard page of my web app I retrieve data from firebase in a created function. This means when the page is loaded everything in that function is ran. This function retrieves a document in the users collection called master users. I could have retrieved this data by looping through all the documents in the data collection but this would have made the query slower, I decided to structure my database as efficiently as possible when I created it to future proof it to make queries and posts as easy and straight forward as possible. Initially I console logged this data just to show that it was coming back but I then progressed to setting it strings for use on the HTML side to display it.

Posting Journey Data

I wanted to post all the journey data from each journey on the scooter from my android app to be able to get an even better picture of each user and they’re journeys on the web app for the insurance side of the project. I decided that I should post the current speed, average speed, distance, coordinates, user, time and a journey specific journey id. I looked at posting all this data in one document with an array for each data point I wanted to record however after looking at if I could add an element to an already existing array on firebase I found out I would have to first take each array down, save it locally, add another element to it locally and then post it again. I felt this was very messy and could potentially be riddled with bugs. I decided that every 5 seconds I would post these data points under a new document each time but have

```
Timer t = new Timer();
t.schedule(() -> {

        double longitude = 0.0;
        double latitude = 0.0;
        if(locTrack.canGetLocation()){
            longitude = locTrack.getLongitude();
            latitude = locTrack.getLatitude();
        }

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "dd-MM-yyyy-hh-mm-ss");
        String time_stamp = simpleDateFormat.format(new Date());


        Map<String, Object> journeyData = new HashMap<>();
        journeyData.put(KEY_USER, "Niall Meade");
        journeyData.put(KEY_JOURNEYID, journeyID);
        journeyData.put(KEY_SPEED, Statistics.getCurrentSpeed());
        journeyData.put(KEY_DISTANCE, Statistics.getDistanceTravelled());
        journeyData.put(KEY_LONG, longitude);
        journeyData.put(KEY_LAT, latitude);
        journeyData.put(KEY_BAT, Statistics.getBatteryLife());
        journeyData.put(KEY_AVG_SPEED, Statistics.getAverageSpeed());
        journeyData.put(KEY_TIME, time_stamp);




        db.collection( collectionPath: "journey data").document(time_stamp).set(journeyData).addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                Toast.makeText( context: DeviceActivity.this, text: "working", Toast.LENGTH_SHORT).show();
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Toast.makeText( context: DeviceActivity.this, text: "runtime error" + e.toString(), Toast.LENGTH_SHORT).show();
            }
        });
}, delay: 0, period: 5000);
```

them all linked via a journey id. This loops every 5 seconds once the start journey button is pressed.

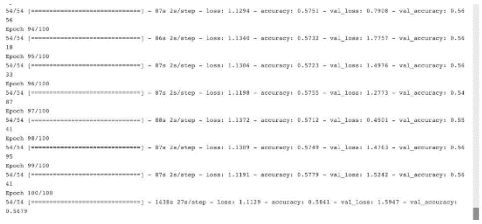
Terrain Classifier

In my project I used convolutional neural network model which is suited for classifying

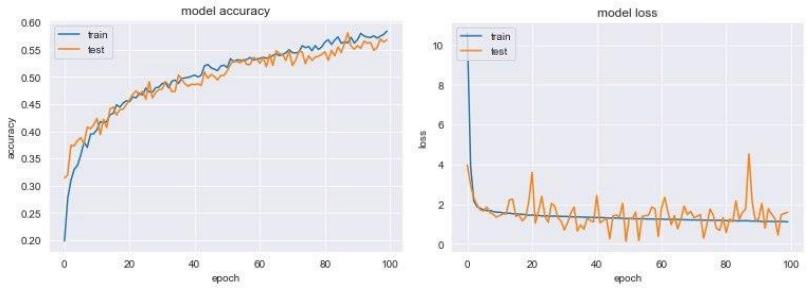
images. I used the Keras neural network library on top of the TensorFlow machine learning library. I trained my terrain CNN using a dataset I acquired from google images. I downloaded this dataset using a google images web scraper library which I ran in terminal.

```
googleimagesdownload --k “grass” –limit 1000
```

This command calls the “googleimagesdownload” library, passes grass in as the keyword parameter and 1000 images as the limit for the max number of pictures to be downloaded. My model is a CNN or convolutional neural network which is better suited for image classification. I trained it on 8 different categories: curb, footpath, grass, loose stone path, road, sand, snow on path and wet path. I trained the neural network with the dataset for

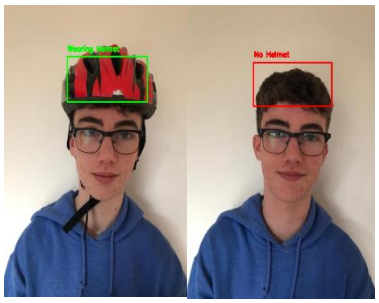


100 epochs. An epoch is a cycle so 100 epochs means it looped through the whole dataset 100 times. For the model to be able to learn all the images had to be the same size ie the pixel width and height had to be uniform across all the pictures. This was a parameter I played around with because the larger the picture the more information you give the model but the longer it will take to proccess. I achieved an accuracy of 58.4%, this is something I want to improve on in the future.



Helmet Classifier

In my project I used a neural network to make the decision of whether the driver of the scooter was wearing a helmet or not. I couldn't find a dataset of people wearing helmets and not wearing them, so I decided to use facial recognition as an aid to this problem. The reason facial recognition is of use is because what we're looking for is the driver, a person, wearing a helmet so the facial recognition narrows the scope down. Then it crops to the 30% of the head where the helmet will be. In my prototype the program is just looking to see how much of the colour red is in the top 30% of this image. It then puts a box around the helmet and displays the relevant text, green box for helmet and a red box when there is no helmet present.



Code

Pulling User Data from Firebase

The purpose of this function is to call general user statistics down from firebase to be displayed in the dashboard of the insurance web app in the form of stats cards. The function is called when the page is loaded hence the "created()". The first thing that happens is I call db which is a reference to Firebase. Next is collection which refers to the collection on Firebase, in this case I want to access the users collection. This is followed by where and get which returns all documents where the field "master" is equal to true. I setup my database with a master users collection to make queries like this as straight forward as possible. This query snapshot is then passed into an arrow function where I set the appropriate data to the relevant strings for use locally in the html side of this page. In JavaScript in an arrow function all variables are referred to using this dot. I have console logs here for debugging different problems I had in the past. The strings are called in the html in double curly brackets.

```
created(){
 db.collection("users").where("master", "==", true).get().then((querySnapshot) => {
     querySnapshot.forEach((doc) => {
         console.log(doc.id, " => ", doc.data());
         this.statsData = doc.data();
         this.localtotalUsers = this.statsData.totalUsers;
         this.localaveragePremium = this.statsData.averagePremium;
         this.localaverageDistance = this.statsData.averageDistance;
         this.localaverageSpeed = this.statsData.averageSpeed;
         console.log("data collected");
     });
 });
```

```
<div class="col-md-6 col-xl-3">

  <stats-card>
    <div class="icon-big text-center" slot="header">
      <i class="ti-user text-info"></i>
    </div>
    <div class="numbers" slot="content">
      <p>Users</p>
      {{localtotalUsers}}
    </div>
  </stats-card>

</div>
```

Posting Journey Data

The purpose of this function is to post data regarding any journeys taken. All this data is stored on Firebase. This function is called when the start journey button is pressed to begin a journey. The first few lines of code setup the timer I use to loop through the code to post every 5 seconds, I also setup the longitude and latitude references so I can post them as coordinates later. Next, I setup the data and time variable that is used later as a time stamp to order the documents rather than have randomly assigned document names for them. I then prepare all the data into a hash map to post to firebase. I declare the hash map and put various data points such as current speed, battery life and coordinates in the map. The key will be the name of the field when it's posted to Firebase. Next I post to firebase, I call firebase with db and give the collection name journey data. The document gets called whatever time it is based off the time stamp it got earlier. This document is then set to the hash map set up earlier with all the different data points to go off. An on success and on failure listener are then added to help with debugging different problems.

```
Timer t = new Timer();
t.schedule(() -> {

        double longitude = 0.0;
        double latitude = 0.0;
        if(locTrack.canGetLocation()){
            longitude = locTrack.getLongitude();
            latitude = locTrack.getLatitude();
        }

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "dd-MM-yyyy-hh-mm-ss");
        String time_stamp = simpleDateFormat.format(new Date());


        Map<String, Object> journeyData = new HashMap<>();
        journeyData.put(KEY_USER, "Niall_Meade");
        journeyData.put(KEY_JOURNEYID, journeyID);
        journeyData.put(KEY_SPEED, Statistics.getCurrentSpeed());
        journeyData.put(KEY_DISTANCE, Statistics.getDistanceTravelled());
        journeyData.put(KEY_LONG, longitude);
        journeyData.put(KEY_LAT, latitude);
        journeyData.put(KEY_BAT, Statistics.getBatteryLife());
        journeyData.put(KEY_AVG_SPEED, Statistics.getAverageSpeed());
        journeyData.put(KEY_TIME, time_stamp);



        db.collection( collectionPath: "journey data").document(time_stamp).set(journeyData).addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                Toast.makeText( context: DeviceActivity.this,  text: "working", Toast.LENGTH_SHORT).show();
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Toast.makeText( context: DeviceActivity.this,  text: "runtime error" + e.toString(), Toast.LENGTH_SHORT).show();
            }
        });
},  delay: 0,  period: 5000);
```

Terrain Classifier Web Cam Program

To run my terrain classification model on my laptop I made a python program that takes advantage of the webcam. The first few lines of the program import the relevant libraries such as imutils, numpy, cv2 and keras. Next the message “starting video stream” is shown as the webcam is turned on and the video stream begins. Some references and constants are then setup such as the image height and width and the various terrain classes. While the camera is on it loops through each frame and sends it to a “classify_image(pic)” function and passes the frame in as the pic. The frame is changed to the correct size of 50 by 50 pixels and converted to a numpy array which is then sent to the terrain classifier model to decide what category it falls under.

Building & Testing

While building the classifiers I used multiple documentations online from the developers of the libraries such as TensorFlow. I used various tutorials on YouTube to learn about neural networks and how to build them, I also looked at multiple examples online. The documentation online is good and all the problems I came across had documentation on or someone else had experienced the same problem and had made a post about it on stack overflow or git hub which had since been answered. If I was really having trouble I would call on my cousin Ciaran to give me a hand or direct me in the correct direction. While developing the web app I was able to test it live by running the following command in terminal.

```
npm run serve
```

This made it really easy and quick my web app and quickly diagnose problems. I also have experience using this type of an environment from last year's project TagBuddy so that also helped. Developing the android app was a new experience using the Android SDK, the cycle of testing new code was longer because each time I would have to load the app onto my phone, while it would only take 2 minutes this process can get tedious. I looked at using an emulator locally on my laptop but it appeared to be overly complicated to get Bluetooth running in the emulator. Android is a very well documented SDK so if you come across a problem chances are its been experienced before and the answer is readily available.

Conclusion

I have built a protype of my project Scootsure. I have a working terrain classifier that can identify suitable terrain for the scooter to drive on. I have also developed a helmet classifier that can successfully make the decision if the driver is wearing a helmet or not. I have built an android app and web app that make up a centralised database for electric scooter data which at the moment there is nothing like. This idea has the potential to help insurance companies gain a better understanding the risk of scooters. In the future I would like to implement the classifiers directly into the phone app but that's a project in itself. I would also like to explore other options for getting a terrain dataset as the dataset I created had a lot of images that weren't related to there category and as a result caused the classifier to be less accurate. I would also like to change my backend from Firebase or at the very least the way it is structured because currently it might not scale properly. I believe that Scootsure as a whole could be of great use for scooters, but I also think that the two safety aspects of my project, i.e. terrain and helmet classifier could also be put to great use in other ways such as on bikes.

Appendix

Sponsorship Letter/Email

Dear …….

My name is Niall Meade, I'm a TY (Transition Year) student in Ardscoil Ris, Limerick. This year I have been accepted into the BT Young Scientist Exhibition for the 3rd time. This year's

project is one of 550 projects selected by the screening judges to compete in the finals at the RDS, Dublin from 8th to the 11th January 2020. Previously in 1st and 3rd year my technology based projects were successful in winning both category and special awards. In addition, I was one of the 30 individuals selected to participate in the BT Business Bootcamp out of over 1000 students at the exhibition.

My project relates to the safety and legalisation of electric scooters. I am currently making an automated system that prevents the use of scooters on unsuitable terrain such as grass. It will also force the user to wear a helmet at all times, via a safety mechanism. These are the two main features I hope to implement in my project.

To do this I require an electric scooter, specifically the Xiaomi M365 scooter, as it's control-board is customizable allowing me to give it commands such as stop/start from my phone which allows me to implement the two features described above.

I'm approaching your organisation asking for sponsorship of €400 for the purchase of the scooter or a contribution towards it. In return, my project would provide publicity for your organisation via my poster, report book and media coverage in the RDS with potential interviews from RTE, Tg4 etc.

Thank you for taking the time to read my email, looking forward to your favourable response.

Regards,

Niall Meade

Web App

Dashboard page

```
<template>
  <div>



  <div class="row">
```

```
    <div class="col-md-6 col-xl-3">

      <stats-card>
        <div class="icon-big text-center" slot="header">
          <i class="ti-user text-info"></i>
        </div>
        <div class="numbers" slot="content">
          <p>Users</p>
          {{localtotalUsers}}
        </div>
      </stats-card>

    </div>

    <div class="col-md-6 col-xl-3">

      <stats-card>
        <div class="icon-big text-center" slot="header">
          <i class="ti-target text-danger"></i>
        </div>
        <div class="numbers" slot="content">
          <p>Average Speed</p>
          {{localaverageSpeed}} km/h
        </div>
      </stats-card>

    </div>

    <div class="col-md-6 col-xl-3">

      <stats-card>
        <div class="icon-big text-center" slot="header">
          <i class="ti-money text-success"></i>
        </div>
        <div class="numbers" slot="content">
          <p>Average Premium</p>
          €{{localaveragePremium}}
        </div>
      </stats-card>

    </div>

    <div class="col-md-6 col-xl-3">

      <stats-card>
        <div class="icon-big text-center" slot="header">
          <i class="ti-map-alt text-error"></i>
        </div>
```

```
        <div class="numbers" slot="content">
          <p>Average Distance</p>
          {{localaverageDistance}} km
        </div>
      </stats-card>

    </div>
  </div>




    <!--Charts-->
    <div class="row">

      <div class="col-12">
        <chart-card title="Journeys"
                    sub-title="24 Hours performance"
                    :chart-data="usersChart.data"
                    :chart-options="usersChart.options">

          <div slot="legend">
            <i class="fa fa-circle text-info"></i> low Risk Customer
            <i class="fa fa-circle text-warning"></i> Medium Risk Customer
            <i class="fa fa-circle text-danger"></i> High Risk Customer
          </div>
        </chart-card>
      </div>


    </div>

  </div>
</template>
<script>
import { StatsCard, ChartCard } from "@/components/index";
import Chartist from 'chartist';
import "firebase/app";
import db from "@/components/firebaseinit.js";

import 'firebase/firestore';

export default {
  components: {
    StatsCard,
    ChartCard
  },
```

```
   created(){
    db.collection("users").where("master", "==", true).get().then((querySnapsh
ot) => {
        querySnapshot.forEach((doc) => {
            console.log(doc.id, " => ", doc.data());
            this.statsData = doc.data();
            this.localtotalUsers = this.statsData.totalUsers;
            this.localaveragePremium = this.statsData.averagePremium;
            this.localaverageDistance = this.statsData.averageDistance;

            this.localaverageSpeed = this.statsData.averageSpeed;
            console.log("data collected");
        });
    });


   },

  data() {


    return {


      journeysToday: Number,
      dateToday: Number,

      secondSeries: [],
      statsData: [],

      localtotalUsers: "",
      localaveragePremium: "",
      localaverageDistance: "",
      localaverageSpeed: "",


      statsCards: [

        {
          type: "info",
          icon: "ti-user",
          title: "Users",
          value: this.localtotalUsers,
          footerText: "Updated now",
          footerIcon: "ti-reload"
        },
```

```
      {
        type: "success",
        icon: "ti-money",
        title: "Average Premium",
        value: this.localaveragePremium,
        footerText: "Updated an hour ago",
        footerIcon: "ti-timer"
      },
      {
        type: "danger",
        icon: "ti-target",
        title: "Average Speed",
        value: this.localaverageSpeed,
        footerText: "Updated now",
        footerIcon: "ti-reload"
      },
      {
        type: "error",
        icon: "ti-map-alt",
        title: "Average Distance",
        value: this.localaverageDistance,
        footerText: "Updated now",
        footerIcon: "ti-reload"
      }
    ],
    usersChart: {
      data: {
        labels: [
          "00:00PM",
          "03:00AM",
          "06:00AM",
          "09:00AM",
          "12:00AM",
          "3:00PM",
          "6:00PM",
          "9:00PM",
          "00:00PM"
        ],
        series: [
          [15, 8, 25, 80, 35, 48, 112 , 45, 17],
          [12, 7, 18, 45, 25, 42, 65 , 32, 12],
          [23, 11, 8, 14, 15, 20, 38 , 52, 25]
        ]
      },
      options: {
        low: 0,
        high: 115,
        showArea: true,
```

```
        height: "245px",
        axisX: {
          showGrid: false
        },
        lineSmooth: Chartist.Interpolation.simple({
          divisor: 2.5
        }),
        showLine: true,
        showPoint: false
      }
    },
    activityChart: {
      data: {
        labels: [
          "Jan",
          "Feb",
          "Mar",
          "Apr",
          "Mai",
          "Jun",
          "Jul",
          "Aug",
          "Sep",
          "Oct",
          "Nov",
          "Dec"
        ],
        series: [
          [542, 543, 520, 680, 653, 753, 326, 434, 568, 610, 756, 895],
          [230, 293, 380, 480, 503, 553, 600, 664, 698, 710, 736, 795]
        ]
      },
      options: {
        seriesBarDistance: 10,
        axisX: {
          showGrid: false
        },
        height: "245px"
      }
    },
    preferencesChart: {
      data: {
        labels: ["62%", "32%", "6%"],
        series: [62, 32, 6]
      },
      options: {}
    },
```

```
    };
  }


};
</script>
<style>
</style>
```

Users Page

```
<template>
    <div>

      <div>
        <mdb-tbl>
          <mdb-tbl-head>
            <tr>
              <th>User</th>
              <th>Premium</th>
              <th>Average Speed</th>
              <th>Average Distance</th>
              <th>Total Journeys</th>
              <th>Age</th>
            </tr>
          </mdb-tbl-head>
          <mdb-tbl-body>
            <tr v-for="user in users" :key="user" v-
on:click="getAllUserData(user)">
              <th>{{user.user}}</th>
              <td>{{user.premium}}</td>
              <td>{{user.averageSpeed}}km/h</td>
              <td>{{user.averageDistance}}km</td>
              <td>{{user.totalJourneys}}</td>
              <td>{{user.age}}</td>
            </tr>
          </mdb-tbl-body>
        </mdb-tbl>
      </div>

    </div>
</template>
<script>
import { PaperTable } from "@/components";
import "firebase/app";
import db from "@/components/firebaseinit.js";
import 'firebase/firestore';
import { mdbTbl, mdbTblHead, mdbTblBody } from 'mdbvue';
```

```
import { json } from 'body-parser';

const tableColumns = ["User", "Premium", "Average "+"Speed", "Average "+"Dista
nce", "Total "+"Journeys", "Age"];

export default {

  methods: {

    getAllUserData(user){
      console.log("success" + user.user)
      localStorage.setItem("user" ,user.user)
      this.$router.push("data-visualisation")
    }
  },

  components: {
    PaperTable,
    mdbTbl,
    mdbTblHead,
    mdbTblBody
  },

   created(){
    db.collection("users").where("master", "==", false).get().then((querySnaps
hot) => {
        querySnapshot.forEach((doc) => {
            this.users.push(doc.data());
            console.log(doc.id, " => ", doc.data());
            console.log(this.users);
        });
    });
  },

  data() {

    return {

      users: [],

      table: {
        title: "User's Journeys",
        subTitle: "",
        columns: [...tableColumns]
      }
    };
  }
};
```

```
</script>
<style>
</style>
```

Data Visualisation Page

```
<template>
    <div>
        <div class="row">

            <div class="col-md-6 col-xl-3">

                <stats-card>
                    <div class="icon-big text-center" slot="header">
                        <i class="ti-user text-info"></i>
                    </div>
                    <div class="numbers" slot="content">
                        <p>Username</p>
                        {{userData.user}}
                    </div>
                </stats-card>

            </div>

            <div class="col-md-6 col-xl-3">

                <stats-card>
                    <div class="icon-big text-center" slot="header">
                        <i class="ti-info-alt text-danger"></i>
                    </div>
                    <div class="numbers" slot="content">
                        <p>Age</p>
                        {{userData.age}}
                    </div>
                </stats-card>

            </div>

            <div class="col-md-6 col-xl-3">

                <stats-card>
                    <div class="icon-big text-center" slot="header">
                        <i class="ti-target text-danger"></i>
                    </div>
                    <div class="numbers" slot="content">
                        <p>Average Speed</p>
                        {{userData.averageSpeed}}km/h
                    </div>
```

```
            </stats-card>

        </div>

        <div class="col-md-6 col-xl-3">

            <stats-card>
                <div class="icon-big text-center" slot="header">
                    <i class="ti-map-alt text-error"></i>
                </div>
                <div class="numbers" slot="content">
                    <p>Average Distance</p>
                    {{userData.averageDistance}}km
                </div>
            </stats-card>

        </div>

        <div class="col-md-6 col-xl-3">

            <stats-card>
                <div class="icon-big text-center" slot="header">
                    <i class="ti-map text-info"></i>
                </div>
                <div class="numbers" slot="content">
                    <p>Total Journeys</p>
                    {{userData.totalJourneys}}
                </div>
            </stats-card>

        </div>

        <div class="col-md-6 col-xl-3">

            <stats-card>
                <div class="icon-big text-center" slot="header">
                    <i class="ti-money text-success"></i>
                </div>
                <div class="numbers" slot="content">
                    <p>Premium</p>
                    {{userData.premium}}
                </div>
            </stats-card>

        </div>

    </div>
```

```
      <div>
        <mdb-tbl>
          <mdb-tbl-head>
            <tr>
              <th>User</th>
              <th>Average Speed</th>
              <th>Distance</th>
              <th>Time Taken</th>
              <th>Start Time</th>
              <th>Finish Time</th>
            </tr>
          </mdb-tbl-head>
          <mdb-tbl-body>
            <tr v-for="journey in journeys" :key="journey">
                <th>{{journey.user}}</th>
                <td>{{journey.averageSpeed}}km/h</td>
                <td>{{journey.distance}}km</td>
                <td>{{journey.timeTaken}} minutes</td>
                <td>{{journey.start}}</td>
                <td>{{journey.finish}}</td>
            </tr>
          </mdb-tbl-body>
        </mdb-tbl>
      </div>
          <iframe id="igraph" scrolling="no" style="border:none;" seamless="se
amless" src="https://plot.ly/~niall_scootsure/1.embed" height="525" width="100
%"></iframe>

    </div>
</template>
<script>
import EditProfileForm from "./UserProfile/EditProfileForm.vue";
import UserCard from "./UserProfile/UserCard.vue";
import MembersCard from "./UserProfile/MembersCard.vue";
import "firebase/app";
import db from "@/components/firebaseinit.js";
import 'firebase/firestore';
import { StatsCard, ChartCard } from "@/components/index";
import { mdbTbl, mdbTblHead, mdbTblBody } from 'mdbvue';
import { PaperTable } from "@/components";

export default {

  components: {
    StatsCard,
    ChartCard,
    PaperTable,
    mdbTbl,
```

```
    mdbTblHead,
    mdbTblBody
  },

  created(){
      console.log(localStorage.user)

          db.collection("users").where("user", "==", localStorage.user).get().
then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            this.userData = doc.data();
            console.log(doc.id, " => ", doc.data());
            console.log(this.userData.user);
        });
    });

          db.collection("journeys").where("user", "==", localStorage.user).get
().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            this.journeys.push(doc.data());
            console.log(doc.id, " => ", doc.data());
            console.log(this.journeys);
        });
    });

  },

    data() {

    return {

      userData: [],
      journeys: []

    };
  }

};
</script>
<style>
</style>
```

Android App

Login Page

```
package maisi.M365.power.main;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.firestore.FirebaseFirestore;
import com.google.firebase.firestore.QueryDocumentSnapshot;
import com.google.firebase.firestore.QuerySnapshot;

public class LoginFragment extends Activity {

    private FirebaseFirestore db = FirebaseFirestore.getInstance();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        EditText username_field = findViewById((R.id.et_email));
        EditText password_field = findViewById((R.id.et_password));
        Button login_btn = findViewById(R.id.btn_login); // reference to your
button ID in your layout xml file
        login_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                String username_input = username_field.getText().toString();
                String password_input = password_field.getText().toString();
                db.collection("users").whereEqualTo("user", username_input).ge
t()
                        .addOnCompleteListener(new OnCompleteListener<QuerySna
pshot>() {
                            @Override
                            public void onComplete(@NonNull Task<QuerySnapshot
> task) {
                                if (task.isSuccessful()) {
                                    for (QueryDocumentSnapshot document : task
.getResult()) {
                                        if(document.getData().get("password").
equals(password_input)){
                                            try {
```

```
                                                Intent k = new Intent(LoginFra
gment.this, ScanActivity.class);
                                                startActivity(k);
                                            } catch(Exception e) {
                                                e.printStackTrace();
                                            }
                                        }
                                        else{
                                            new AlertDialog.Builder(LoginFragm
ent.this)
                                                    .setTitle("Error Logging i
n")
                                                    .setMessage("Username or p
assword incorrect")

                                                    // Specifying a listener a
llows you to take an action before dismissing the dialog.
                                                    // The dialog is automatic
ally dismissed when a dialog button is clicked.
                                                    .setPositiveButton(android
.R.string.yes, new DialogInterface.OnClickListener() {
                                                        public void onClick(Di
alogInterface dialog, int which) {
                                                            // Continue with d
elete operation
                                                        }
                                                    }).setNegativeButton(andro
id.R.string.no, null)
                                                    .setIcon(android.R.drawabl
e.ic_dialog_alert)
                                                    .show(); ;}
                                    }
                                } else {
                                    new AlertDialog.Builder(LoginFragment.this
)
                                            .setTitle("Error Logging in")
                                            .setMessage("Username or password
incorrect")

                                            // Specifying a listener allows yo
u to take an action before dismissing the dialog.
                                            // The dialog is automatically dis
missed when a dialog button is clicked.
                                            .setPositiveButton(android.R.strin
g.yes, new DialogInterface.OnClickListener() {
                                                public void onClick(DialogInte
rface dialog, int which) {
```

```
                                                    // Continue with delete op
eration
                                                }
                                            }).setNegativeButton(android.R.str
ing.no, null)
                                            .setIcon(android.R.drawable.ic_dia
log_alert)
                                            .show();
                                    }
                                }
                            });
                }
            });
    }


}
```

Connect to Scooter Page

```
package maisi.M365.power.main;

import android.Manifest;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;

import maisi.M365.power.util.BleUtil;
import maisi.M365.power.util.ScannedDevice;
import permissions.dispatcher.NeedsPermission;
import permissions.dispatcher.RuntimePermissions;

@RuntimePermissions
```

```
public class ScanActivity extends Activity implements BluetoothAdapter.LeScanC
allback {
    private BluetoothAdapter mBTAdapter;
    private DeviceAdapter mDeviceAdapter;
    private boolean mIsScanning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.activity_scan);

        init();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        stopScan();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        if (mIsScanning) {
            menu.findItem(R.id.action_scan).setVisible(false);
            menu.findItem(R.id.action_stop).setVisible(true);
        } else {
            menu.findItem(R.id.action_scan).setVisible(true);
            menu.findItem(R.id.action_stop).setVisible(false);
        }
        return super.onPrepareOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int itemId = item.getItemId();
        if (itemId == android.R.id.home) {
            // ignore
            return true;
        } else if (itemId == R.id.action_scan) {
```

```
            ScanActivityPermissionsDispatcher.startScanWithPermissionCheck(thi
s);
            return true;
        } else if (itemId == R.id.action_stop) {
            stopScan();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onLeScan(final BluetoothDevice newDeivce, final int newRssi,
                         final byte[] newScanRecord) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                mDeviceAdapter.update(newDeivce, newRssi, newScanRecord);
            }
        });
    }

    private void init() {
        // BLE check
        if (!BleUtil.isBLESupported(this)) {
            Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHOR
T).show();
            finish();
            return;
        }

        // BT check
        BluetoothManager manager = BleUtil.getManager(this);
        if (manager != null) {
            mBTAdapter = manager.getAdapter();
        }
        if (mBTAdapter == null) {
            Toast.makeText(this, R.string.bt_unavailable, Toast.LENGTH_SHORT).
show();
            finish();
            return;
        }
        if (!mBTAdapter.isEnabled()) {
            Toast.makeText(this, R.string.bt_disabled, Toast.LENGTH_SHORT).sho
w();
            finish();
            return;
        }
```

```
        // init listview
        ListView deviceListView = (ListView) findViewById(R.id.list);
        mDeviceAdapter = new DeviceAdapter(this, R.layout.listitem_device,
                new ArrayList<ScannedDevice>());
        deviceListView.setAdapter(mDeviceAdapter);
        deviceListView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterview, View view, int
 position, long id) {
                ScannedDevice item = mDeviceAdapter.getItem(position);
                if (item != null) {
                    Intent intent = new Intent(view.getContext(), DeviceActivi
ty.class);
                    BluetoothDevice selectedDevice = item.getDevice();
                    //intent.putExtra(DeviceActivity.EXTRA_BLUETOOTH_DEVICE, s
electedDevice);
                    intent.putExtra(DeviceActivity.EXTRAS_DEVICE_NAME, selecte
dDevice.getName());
                    intent.putExtra(DeviceActivity.EXTRAS_DEVICE_ADDRESS, sele
ctedDevice.getAddress());
                    startActivity(intent);

                    // stop before change Activity
                    stopScan();
                }
            }
        });

        stopScan();
    }

    @NeedsPermission(Manifest.permission.ACCESS_COARSE_LOCATION)
    void startScan() {
        if ((mBTAdapter != null) && (!mIsScanning)) {
            mBTAdapter.startLeScan(this);
            mIsScanning = true;
            setProgressBarIndeterminateVisibility(true);
            invalidateOptionsMenu();
        }
    }

    private void stopScan() {
        if (mBTAdapter != null) {
            mBTAdapter.stopLeScan(this);
        }
        mIsScanning = false;
        setProgressBarIndeterminateVisibility(false);
        invalidateOptionsMenu();
```

```
    }
}
```

Journey Dashboard Page

```
    public void startHandler(View view) {
        if (!handlerStarted) {
            if (!isConnected()) {
                doConnect();
                Handler handler = new Handler();
                handler.postDelayed(() -> {
                    handler1.post(process);
                    handler.post(runnableMeta);
                }, 5000);
            } else {
                handler1.post(process);
                handler.post(runnableMeta);
            }


            startHandlerButton.setText("Finish Journey");
            handlerStarted = true;
            Log.i("LOG", "Got to here");
            LocationTrack locTrack = new LocationTrack(DeviceActivity.this);



            Timer t = new Timer();
            t.schedule(new TimerTask() {
                @Override
                public void run() {

                    double longitude = 0.0;
                    double latitude = 0.0;
                    if(locTrack.canGetLocation()){
                        longitude = locTrack.getLongitude();
                        latitude = locTrack.getLatitude();
                    }

                    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("
dd-MM-yyyy-hh-mm-ss");
                    String time_stamp = simpleDateFormat.format(new Date());


                    Map<String, Object> journeyData = new HashMap<>();
                    journeyData.put(KEY_USER, "niall");
                    journeyData.put(KEY_JOURNEYID, journeyID);
                    journeyData.put(KEY_SPEED, Statistics.getCurrentSpeed());
```

```
                    journeyData.put(KEY_DISTANCE, Statistics.getDistanceTravel
led());
                    journeyData.put(KEY_LONG, longitude);
                    journeyData.put(KEY_LAT, latitude);
                    journeyData.put(KEY_BAT, Statistics.getBatteryLife());
                    journeyData.put(KEY_AVG_SPEED, Statistics.getAverageSpeed(
));
                    journeyData.put(KEY_TIME, time_stamp);




                    db.collection("journey data").document(time_stamp).set(jou
rneyData).addOnSuccessListener(new OnSuccessListener<Void>() {
                        @Override
                        public void onSuccess(Void aVoid) {
                            Toast.makeText(DeviceActivity.this, "working", Toa
st.LENGTH_SHORT).show();
                        }
                    }).addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Toast.makeText(DeviceActivity.this, "runtime error
" + e.toString(), Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            }, 0, 5000);

        }
        else{
            stopHandler();
            handlerStarted=false;
        }
    }

    public void reset(View view) {
        Statistics.resetPowerStats();
    }

    public void stopHandler() {
        Log.d(TAG,"Stop Handler called");
        handler.removeCallbacksAndMessages(null);
        handler1.removeCallbacksAndMessages(null);
        requestQueue.clear();
        logWriter.writeLog(true);
        Toast.makeText(this, "Logs in:"+logWriter.getPath(), Toast.LENGTH_LONG
).show();
        startHandlerButton.setText("Start Handler");
```

```
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("hh:mm");
        String time_stamp = simpleDateFormat.format(new Date());

        String anothertest = "test2";

        Map<String, Object> finalJourney = new HashMap<>();
        finalJourney.put(KEY_AVG_SPEED, Statistics.getAverageSpeed());
        finalJourney.put(KEY_DISTANCE, Statistics.getDistanceTravelled());
        finalJourney.put(KEY_USER, "niall");
        finalJourney.put(KEY_FINISH, time_stamp);
        finalJourney.put(KEY_BAT, Statistics.getBatteryLife());
        finalJourney.put(KEY_TEMP, Statistics.getBatteryTemperature());

        db.collection("journeys").document(time_stamp).set(finalJourney)
                .addOnSuccessListener(new OnSuccessListener<Void>() {
                    @Override
                    public void onSuccess(Void aVoid) {
                        Toast.makeText(DeviceActivity.this, "success", Toast.L
ENGTH_SHORT).show();
                    }
                })
                .addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Toast.makeText(DeviceActivity.this, "error", Toast.LEN
GTH_SHORT).show();
                    }
                });



    }
```

Terrain Classifier Web Cam Program

```
# import the necessary packages
from imutils.video import VideoStream
import numpy as np
import argparse
from matplotlib import pyplot as plt
import imutils
import time
from PIL import Image
import cv2
import numpy
from keras.models import load_model


# initialize the video stream and allow the camera sensor to warm up
```

```
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

IMG_W = 50
IMG_H = 50
terrain_model = load_model('terrain_model.h5')
terrain_classes=['Curb', 'Footpath', 'Grass', 'Stones', 'Road', 'Sand', 'Snow'
, 'Wet']

def classify_image(pic):

    pic = cv2.resize(pic, (IMG_W, IMG_H))
    pic = np.expand_dims(pic, axis=0)
    class_predicted = terrain_model.predict_classes(pic)
    return class_predicted.max()

# loop over the frames from the video stream
while True:

    frame = vs.read()
    (h, w) = frame.shape[:2]
    frame = imutils.resize(frame, width=400)
    pred = classify_image(frame)
    frame = cv2.flip(frame,1)
    prediction = terrain_classes[pred]
    cv2.putText(frame, prediction, (20, 20),
        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
    cv2.imshow("Terrain Classifier", frame)

    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Helmet Classifier Web Cam Program

```
# import the necessary packages
from imutils.video import VideoStream
import numpy as np
import argparse
from matplotlib import pyplot as plt
import imutils
```

```
import time
from PIL import Image
import cv2
import numpy

avg_color = [0,0,0]
#deep neaural net model for face recognition
prototxt = './deploy.prototxt.txt'
model = './res10_300x300_ssd_iter_140000.caffemodel'
confidence_threshold = 0.6

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(prototxt, model)

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=1).start()
time.sleep(2.0)

# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

    # grab the frame dimensions and convert it to a blob
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
        (300, 300), (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the detections and
    # predictions
    net.setInput(blob)
    detections = net.forward()



    # loop over the detections
    for i in range(0, detections.shape[2]):
        # extract the confidence (i.e., probability) associated with the
        # prediction
        confidence = detections[0, 0, i, 2]

        # filter out weak detections by ensuring the `confidence` is
        # greater than the minimum confidence
        if confidence < confidence_threshold:
```

```
            continue

        # compute the (x, y)-coordinates of the bounding box for the
        # object
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        h_ = endY-startY
        w_ = endX-startX

        #get the top of the head
        endY = endY - int((h_)/1.1)
        startY = startY - int((h_)/3)
        startX = startX - int((w_)/10)
        endX = endX + int((w_)/10)

        #crop the top of the head
        try:
            cropped = frame[startY:endY, startX:endX]
            avg_color_per_row = numpy.average(cropped, axis=0)
            avg_color = numpy.average(avg_color_per_row, axis=0)
            print(avg_color)
        except:
            print("whoops")

        #check if red intensity is above threshold (for helmet)
        if(avg_color[2]>120):
            text = "Wearing Helmet"
            col = (0,255,0)
        else:
            text = "No Helmet"
            col = (0,0,255)
        y = startY - 10 if startY - 10 > 10 else startY + 10
        #draw the bounding box
        cv2.rectangle(frame, (startX, startY), (endX, endY),
            col, 2)


    #show the output frame
    try:
        frame = cv2.flip(frame,1)
        if startX > w/2:
            pos = int(w/2) - (startX - int(w/2))
        else:
            pos = int(w/2) + (int(w/2) - startX)
        cv2.putText(frame, text, (pos-(endX-startX), y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, col, 2)
        #frame = cv2.resize(frame, (w*3,h*3))
        cv2.imshow("Helmet Classifier", frame)
```

```
    except:
        cv2.imshow("Helmet Classifier", frame)

    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

References

On-Semiconductor Scooter Sponsorship

I would like to give a special thanks to On-Semiconductor for sponsoring the cost of €400 for my scooter. I completed my work experience here and after speaking to Gerard O'Brien he said he would speak to management about sponsoring the full cost.