4th Year Project

Technical Document

Project: Which Way?

Name: Toba Toki

Student Number: 15349476

Supervisor: Stephen Blott

Course: Computer Applications

Contents

Abstract

“Which Way?” is a technology direction system. Lights called Neopixels are mounted on the bicycle which directs the user of which way to turn along the journey towards their destination. “Which Way?” consists of an Android application that the user uses to select their desired destination, and a mount which has a round, sewable, Arduino-compatible microcontroller called a Flora Board that is connected to Bluetooth and GPS receivers. The system is designed to be used in two forms. Untethered and tethered. Untethered is where the destination is sent to the Flora board and the GPS module is used to instruct the user of when they are at a turn location. It can also be used tethered, where the phone sends each direction as the user approaches a turn.

Two Neopixels LED lights on top of the bike signals the user to turn left or right. They flash to signal to the user that the route is being sent, also to inform when the phone has been disconnected from the Flora board, and to inform the user when the board’s GPS has gotten a fix.

1. Motivation

While current navigation apps such as google maps are fine for driving or walking, their interfaces are too complicated to be interpreted at a glance. The ability to quickly read directions is essential for cyclists who need to maintain concentration on their surroundings, especially while sharing the road with cars. The aim of this project was to provide an effective yet safe solution for cyclists by insuring they can navigate easily while retaining their focus on the road.

I cycle quite frequently myself and I'm no stranger to the frustration of constantly needing to take my phone out of my pocket when I am cycling to a destination I have never been to. The idea of phone stands on the handlebars provide some solution to this, but cycling can be very dangerous and taking your attention away from the road even for a few seconds to process the user interfaces increases the risk of accidents that could potentially lead to serious injuries or even death.

With "Which Way?", the lights being mounted on the handle bars means by the use of a user's peripheral vision, the user never has to take their attention away from straight ahead on the road.

Another benefit of this project is that if the cyclist wanted, they can even leave their phones at home and still navigate to their destination. A smartphone is often the most valuable item a person carries with them on a daily basis and mounting it on the front of a bicycle means that even minor collisions or hard braking risks causing serious damage. It also avoids the phone from being stolen should the rider be forced to stop at a traffic light.

2. Research

For this project, I had to research a lot of different topics that I had not come across before. A great deal of research went into the Flora board and its components to see how they interact with each other and what can be made with them.

During the starting phase of the project, the first thing I had to do was research how to use the Arduino Studio. Since I had never used Arduino before, I knew this was going to be challenging. For this, I used Adafruit's website which was also the website I purchased the hardware of my project from. Their website was filled with very good tutorials and guides on how to get started on using Arduino and how to use it for the Flora board.

I spent a lot of time researching what Flora board could do and the types of different connections it could give. I found out what RX and TX pins were, and which pins were going to be useful for which parts of the Flora board. I also had to research how to sew the Flora board and accessories together with conductive wire, and I found this part to be very challenging, it took me a few tries and a lot of reading to get a method that worked well and didn't result to inconsistent or broken connections.

When I moved on to developing the Android Application, the first task that I set myself before doing anything else was setting up a Bluetooth connection between the phone and the Flora. This was a lot trickier than I had originally thought and was one of the most challenging parts of the project. I did a lot of research on Bluetooth LE (Low Energy) and on GATT (Generic Attribute Profile), which defines the way Bluetooth LE device transfer data back and forth using concepts called Services and Characteristics.

My project also required a lot of research in finding out why the Bluetooth and GPS modules could not work together, as this was a huge issue with my program that affected functionality and code. I was unable to come to an absolute connection as to why this was happening but, in my attempt, to find the answer, I ended up learning a lot about the Flora's memory, disk space, data pins, hardware and software serial which ended up in me understanding my overall knowledge of the Flora and of microcontrollers in general.

I also spent time researching how to convert latitudes and longitudes to decimal format, and how to write a distance formula between two sets of latitudes and longitudes.

A key piece of my research was discussions with my project supervisor. He loved the idea from the very beginning and him being an enthusiastic cyclist was able to fuel me with ideas of how to go about the project. He was also in agreement that the LED display would be more preferable to a smartphone being mounted on the bike. The ability to consult with someone that I found to be a perfect target user was crucial through the project.

3. Design & Planning

3.1 System Requirements

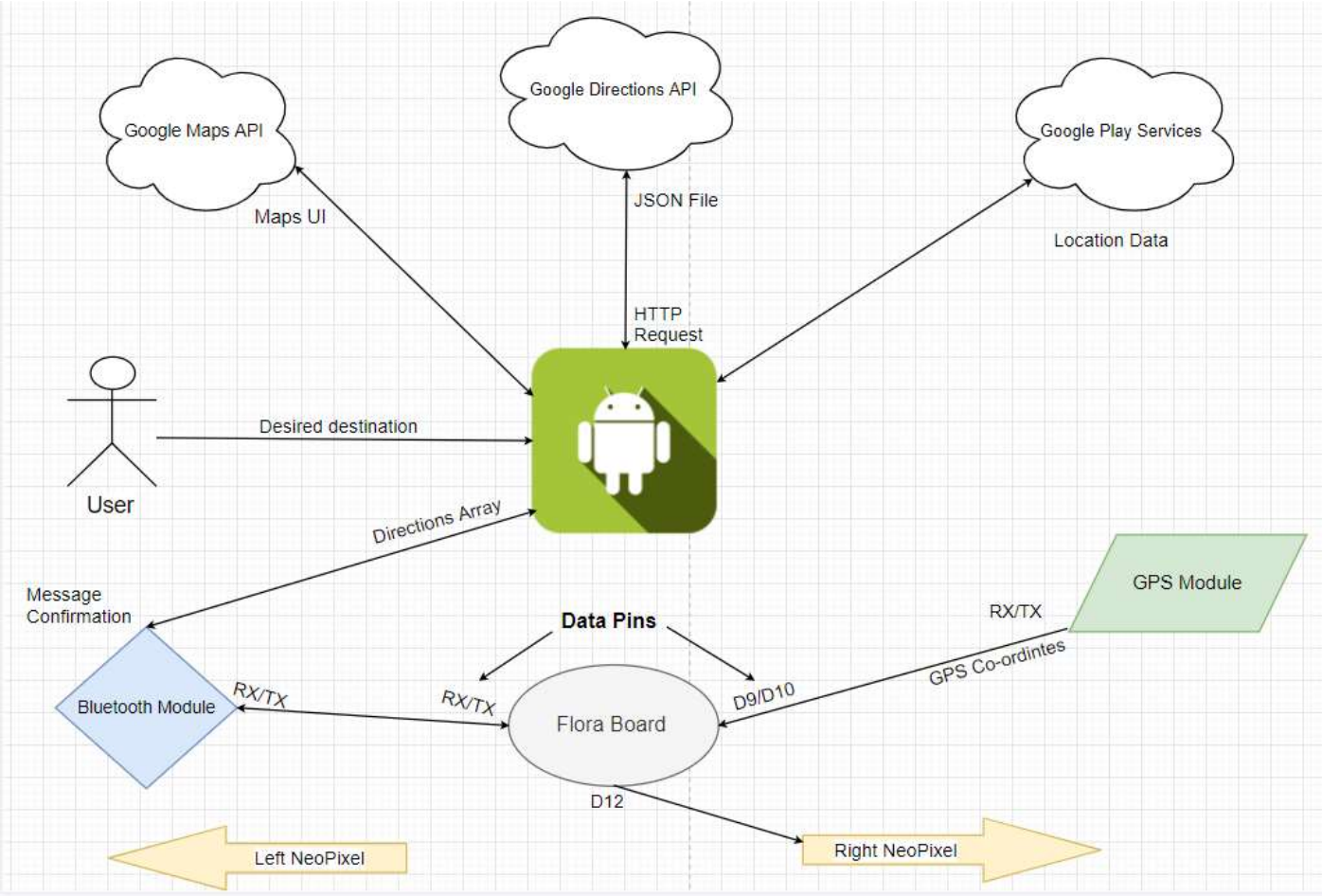
Flora

- Receive data from the Android app via Bluetooth
- Send data back to the Android app
- Store data sent from the Android application e.g. Longitudes, Latitudes and information on turns
- Accommodate both situations where phone is and is not connected
- Use GPS module when untethered from phone to get current location
- Convert GPS module's DD:MM:SS format to decimal
- Light up the appropriate NeoPixel when receiving a left or a right signal (this will be received either from the phone or on the form the Flora board itself)
- Calculate the distance between current location and destination
- If phone disconnects suddenly during journey, switch to on-board navigation
- Use flashing LEDs in different colours to indicate to the user different situations e.g Flash red when the destination is reached

Android Application

- User Interface allows user to easily input a destination
- Uses the user's input along with Google Directions API to get a structured set of data with route details. E.g. Latitude and Longitude coordinates and directions at turns
- Parse the data to make it usable
- Use the mobile device's GPS location system to pinpoint current user's location
- Connect to the Flora board via Bluetooth
- Send direction data to the Flora board
    - A) Turning data. i.e. "Right", when approaching a turn
    - B) Route. i.e. array of latitudes, longitudes and directions (left or right)
- Receive data from Flora (via Bluetooth)

3.2 System Architecture



In the above diagram is the layout of the full system. This begins with the user inputting their desire locations into the UI. This data is then added to a HTTP request that is sent to the Google Directions API which returns a JSON file with the route data of the user's current location to their destination.

This data is then parsed and sent via Bluetooth to the Flora board where it is stored as an array. The Bluetooth module and the Flora Board send data between each other through RX/TX pins.

When the Flora and the mobile device are tethered, the phone sends directions to the Flora board every time a turn is coming up. This in turn causes either the left or the right NeoPixel to light up.

The GPS is needed for when the Flora and the phone become untethered, and the Neopixels are then triggered by the GPS module's coordinate data.

3.3 Android Application Design

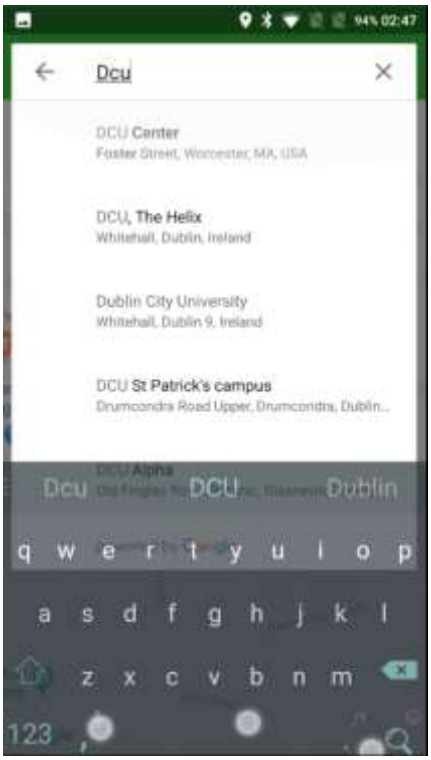
User Interface

I made the android application to be simple for the user to understand. It is the only point of contact between the user and the system. The end goal of the UI is to simply allow the user to be able to input their destination as fast as possible which is why the design of this application is purposely minimal. When the app is opened, the user is faced with a single screen that has a map in it. The applications UI contains a Bluetooth button that connects the application to the Bluetooth module; a search bar which is where the user's input goes into; and a bicycle button that when pressed, sends the route of the destination to the Flora board microcontroller that is on the bike.

The first step is the user selecting their destination in the search bar. This is done using the google autocomplete. As the user is typing, predictions of where they want to go begin to display in a scroll view manner where they can scroll down to find their destination. They are also able to just finish typing where they want and searching for it themselves if they don't want to make use of the autocomplete.

The second step is connecting the app to the Bluetooth module, so data can be passed to and from the app to the board.

Once this is done, we move on to the final step which is when the user selects the button with the person on a bicycle icon and the route is retrieved from Google Directions API, parsed and sent to the Flora board.



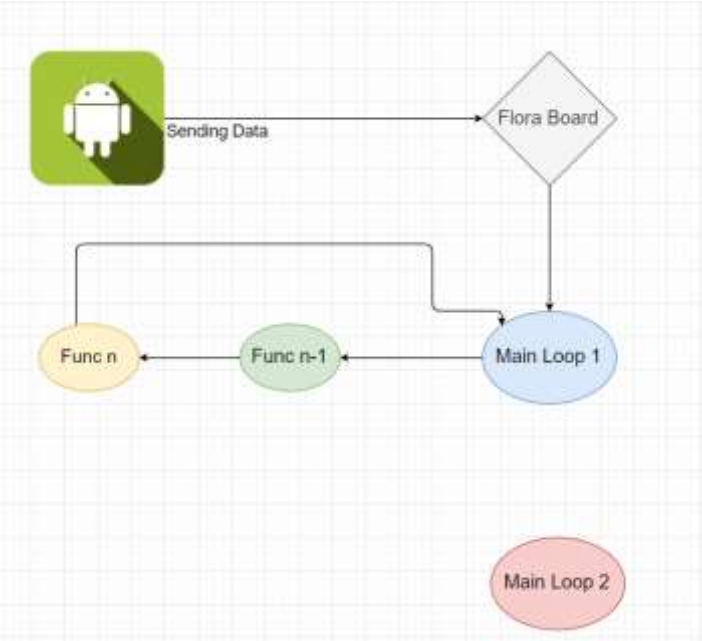
Services

Most of the Android Application's use will be done whilst the user is cycling. After the initial interaction with the application when the user selects their destination, they will no longer need the phone in their hands. This means that the main classes of the app that do all the work needs to be in services instead of the activities.

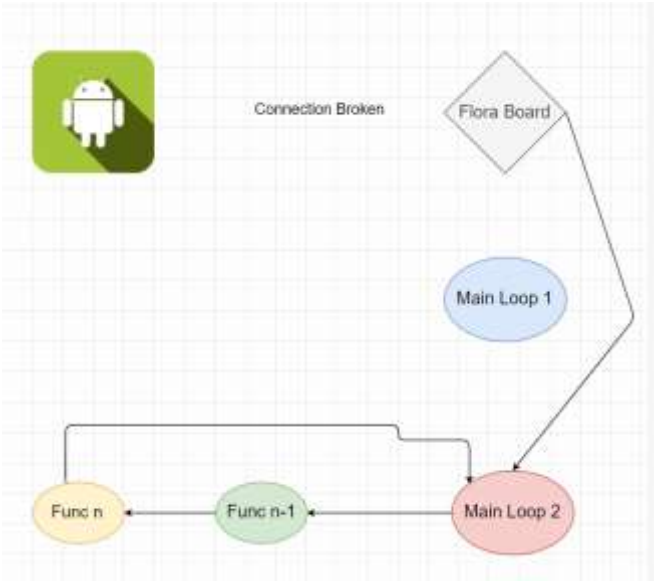
The application has two services. It has a GPS service and a Bluetooth service. The GPS service is in charge of retrieving the user's current location and processing the distance between the user and how close they are to a turn. If the user is in the specified distance away from a turn, the GPS service sends the direction of the turn whether it be left or right to the Bluetooth Service. The Bluetooth service receives this date and then sends it to the Flora board.

3.4 Flora Board Code Design

Tethered



Untethered



The code for the Flora was designed in such a way that there was two different but similar sets of functionalities. I divided the code into functions when the Flora is tethered to the user's mobile device and functions for when it is untethered.

This was done with two main loops. The first loop will continuously keep running until Bluetooth is disconnected. If the Bluetooth is disconnected from the phone, then the second loop is called. This second loop calls for the on-board navigation functions. Such as calculating the distance between two locations and getting a GPS on the user's location. Etc.

Bluetooth and GPS Interfaces

An unanticipated aspect of the project that I found was that the GPS and Bluetooth do not behave well when functioning at the same time on the Flora Board. The Flora only has one set of dedicated hardware inputs and outputs (RX/TX) and if you want to make use of these modules together, it is adamant that you set up custom Inputs and Outputs pins using the SoftwareSerial library on Arduino itself. What I didn't expect was that the SoftwareSerial does not interact well with the hardware serial on the board, and so it was necessary to separate out the use of the GPS and Bluetooth.

Hence, the GPS only comes into use when the Bluetooth is disconnected. In fact, the library is not even initialized until then as even this will cause the GPS to completely stop.

Interference occurs between the NeoPixels and the GPS on SoftwareSerial, as the NeoPixels turns off pin interrupts which is what the SoftwareSerial uses to get the location data from the GPS. This doesn't affect the system in any real way as the number of Neopixels is low since I only need two and the frequency they emit is also low.

Timer interrupt for flashing lights

The Flora system makes use of flashing lights for a lot of different scenarios. I programmed it so that blue flashing lights indicate that the route data is being sent to the Flora, purple flashing lights indicate that the Flora board and the android application have disconnected from one another, and red flashing lights indicate that the use has gotten to their desired destination. The issue with the flashing lights is that they need to flash consistently with respect to time. An easy way to do this was to insert delays into a program so that the light will only flash every n milliseconds, but using delay() on the Flora and other Arduino boards also means that the lights are waiting to flash, the rest of the program is also doing nothing.

For this very reason I decided to use a timer interrupt, so that my program will still be running as the lights are flashing. The interrupt is basically a function in the main loop of the program that is called every time the main loop executes, and given the variables for the LEDs are set, it will either flash on or off for a specified interval, a specified number of times.

4. Implementation

The Android application is divided into 9 different classes. It has one main activity, two services and six helper classes.

App

The classes implemented for the app are as follows:

- *MapsActivity* – Main Activity and UI
- *GPS* – This runs in the background, deals with location, distance between coordinates. Etc.
- *BluetoothLeService* – This runs in the background, connects with the Flora board, sends and receives data
- *BluetoothResultReceiver* – Receives the communications from the Bluetooth in the MapsActivity
- *JSONParser* – Parses the JSON file returned from the Google Directions API
- *DirectionsURL* – Creates a URL to send current location and destination to Google Directions API
- *Step* – An object that represents the way along the route
- *UUIDS* – contains all the UUIDs needed to connect, send and receive data from the Flora board
- *DataRetrieval* – MapsActivity receiver for GPS

MapsActivity

The MapsActivity class is the main activity on the app and is the UI class. The UI features include a map that is generated from google maps API, a search bar and buttons to turn on Bluetooth and to send the route of the user's current location to their desired destination to the Flora board. There is also a button to refresh the screen. In the MapsActivity, there are two services, GPS and Bluetooth which are started up here. When they are started up there is also ResultReceiver class connected to them which acts as a communication channel from Service back to the MapsActivity. This is useful for getting the routeline data from the GPS service to draw the route on the screen, and also the Bluetooth state data to change the Bluetooth display logo to blue when it is connected to the Flora board. The user's current location is enabled in this class as it zooms in on wherever the user is. If the user moves, the blue indicating the user's position also moves.

GPS

This GPS service is one of the classes that runs in the background while the user is cycling. The main role of this class is to get an up-to-date location on the user every half a second. It also calculates the distance between the user and the destination. It calculates if the user goes under the threshold distance. It sends the appropriate direction to turn to the Flora. When the GPS service starts up, it first extracts the URL that is sent with the intent that made it from MapsActivity, as seen below.

```
if (null != intent.getExtras().getString( key: "URL")) {

    // Extract URL for the directions request
    urlString = intent.getExtras().getString( key: "URL");

    // Extract destination selected by user
    destination = intent.getExtras().getString( key: "Destination");

    // This function retrieves the directions using the URL, from the
    // google direction API
    getDirections();
}
```

It then calls the getDirections() method that executes an AsyncTask (a class that runs on a separate thread) that uses the URL previously extracted from the intent to send a HTTP request to the Google Directions API.

```
// Makes a web request for directions from the Google APU, thi is an asynchronous class
// So it doesn't run on the main thread
private class DownloadDirections extends AsyncTask<URL, Integer, Long> {

    @Override
    protected Long doInBackground(URL... params) {
        BufferedReader buffer;
        InputStream stream = null;
        HttpsURLConnection connection = null;
        URL url = params[0];
        try {
            connection = (HttpsURLConnection) url.openConnection();
            connection.setReadTimeout(3000);
            connection.setConnectTimeout(3000);
            //set HTTP method to GET.
            connection.setRequestMethod("GET");
            connection.setDoInput(true);
            //Open communications link.
            connection.connect();
```

From the HTTP request, a JSON file is downloaded which is then passed by the JSONParser class and it is stored in an array.

The GPS class then checks if the user is about to turn using the following code. If the user is, then the appropriate light is turned on.

```
void checkUserIsBeginningTurn(){
    //If user crosses threshold approaching turn, light comes on to
    //indicate turn
    if(getDistanceToTurn() < 60.0f && !aboutToTurn) {
        Log.d(TAG, msg: "inside about to turn");
        lightUpTurnSignal();
        aboutToTurn = true;
    }
}
```

When the light has gone on, the following code checks to see if the user is mid turn

```
private void checkUserIsMidTurn() {
    //Set flags for user who is moving on to the next step,
    //increment counter for steps array
    if(getDistanceToTurn() < 20.0f && aboutToTurn) {
        aboutToTurn = false;
        justTurned = true;
        step++;
        isLastStep = checkIsLastStep();
    }
}
```

Then the following code turns the light off when the user goes back above a threshold from the turn.

```
private void checkUserHasMadeTurn() {
    //If the user has fully made the turn
    if(getDistanceToTurn() > 20 && justTurned){
        justTurned = false;
        turnOffTurnSignal();
    }
}
```

BluetoothLeService

The BluetoothLeService is the part of the app that interacts directly with the Flora. Once the Bluetooth button is clicked by the user, the Bluetooth service is started from the MapsActivity. When the service is started up, it immediately starts scanning for devices by calling scanForDevices() from its initialisation function onStartCommand(). It creates a UUIDS object which contains all the UUIDS needed to connect to the Flora.

```
@Override // Function called when service is started from activity
public int onStartCommand(Intent intent, int flags, int startId) {
    adapter = BluetoothAdapter.getDefaultAdapter();
    handler = new Handler();
    resultReceiver = intent.getParcelableExtra( name: "resultReceiver");
    uuids = new UUIDS();
    //Scan for Bluetooth devices
    scanForDevices();
```

scanForDevices() contains a handler which will scan for a set period which is called SCAN_TIME. To scan for a device, the callback function ScanCallback is called. If this function detects a device, it compares the device's UartUUID to the UARTUUID in the UUIDS class. If the ID's are the same, the device is connected to the Bluetooth service.

After a connection has been established, the BluetoothGattCallback function detects when a connection state changes. It also detects when services are discovered and when characteristics (data) changes.

OnCharacteristicsChanged() is where the confirmation is received from the Bluetooth module and allowing the next item in the route array to be sent.

SendMessage() is the function where the direction or latitude/longitude values get sent to the Bluetooth.

Flora

The code for the Flora board is split into two parts. One part contains the functionality for the tethered element of the system and the other part provides the functionality for the system when it is untethered from the phone.

The first section is the setup. This is where the NeoPixels, GPS and Bluetooth libraries are initialized.

Tethered

After this is the first main loop, which calls the function tetheredFunc(). This loop will run until the Bluetooth disconnects.

```
//This is the first main loop. This loop will continue
//as long as the app is tethered to the flora
while(ble.isConnected()){
  //If the journey has not been completed, call tetheredFunc
  if(!journeyComplete) tetheredFunc();
}
```

Inside tetheredFunc(), first the Bluetooth module is checked for new data, then a new local char array is created and sent as an argument to the function readFromBuffer(). The function then checks if the array contains anything valuable, and if it does, it sends it to the processData() function. tetheredFunc() also calls the interrupt function flashingSignal().

```
void tetheredFunc(){
  //Check for incoming characters from Bluefruit
  ble.println("AT+BLEUARTRX");
  ble.readline();
  ble.waitForOK();
  char bufferData[21];
  readFromBuffer(bufferData);
  flashingSignal();
}
```

Inside processData() the char array it takes as a parameter (the data from the buffer) is checked against a number different codes with different outcomes. “rgt” for instance, calls a function that tells the right light to turn on. “fin” tells the NeoPixels to flash red to indicate the end of a journey.

The first thing that processData() interacts with when the program begins is the latitude, longitude and turn values sent from the app. Latitudes and longitudes are sent from the phone in the form “lat53.23167” and “lng6.67345”, this is done to easily differentiate between latitudes and longitudes when sending the data. The “lat” and “lng” parts are

stripped from the values before they are converted to floats and added to an array. This turns are also added to an array.

After the route data has been fully stored on the Flora, the user sets off on their journey. As long as the phone is tethered, the Flora will now receive either, “lft” which will turn on the left right, “rgt” which will turn on the right light, “off” which will turn off the light after the turn is complete, or “fin” which will flash red to indicate the user has reached their destination. As this is done, the dedicated latLngArrayPointer and turnsPointer are incremented to keep track of where the user is on the route.

Untethered

When the Flora becomes untethered from the phone the program moves to the next main loop. Inside this loop two functions are called, the timer interrupt flashingSignal() as in the last loop, and readFromGPSModule(). In this function the goal is to get a fix on the user’s GPS location with the on-board GPS module, and to use this information to calculate the distance between the user and the destination.

```
float gpsLat = latLngToDecimal(GPS.latitude, GPS.lat);
float gpsLng = latLngToDecimal(GPS.longitude, GPS.lon);
float targetLat = latLngArray[latLngArrayPointer];
float targetLng = latLngArray[latLngArrayPointer + 1];
// was cast as a double
targetDistance = calculateLatLngDist(gpsLat, gpsLng, targetLat, targetLng);
```

In the code above the current latitude and longitude, and the destination’s latitude and longitude are sent to the calculateLatLngDist() function which calculates the distance between them.

```
if(targetDistance < 60.0){
  printTurnDetails(targetLat, targetLng, targetDistance);
  onApproachingTarget();
```

If the distance is less than 60 meters the function onApproachingTarget() is called which sets the correct light on and increments the pointers of the two arrays.

Timer Interrupt for Flashing Light

An essential part of my code is the timer interrupt. This is what is used to flash lights without having to use delay() and keep the whole program busy waiting in the meantime. The interrupt is activated by setting a set of global variables and deactivated by setting them back to zero/false. Below is the setFlashVariables() function and an example call for setting a red flash 15 times at 500 millisecond intervals

```
void setFlashVariables(uint8_t r, uint8_t g, uint8_t b,
            uint8_t f, uint8_t fNum, unsigned long fI, boolean wFJ,
            uint8_t l, uint8_t ri){
   red = r;
   green = g;
   blue = b;
   flashes = f;
   flashNum = fNum;
   flashInterval = fI;
   waitingForJob = wFJ;
   left = l;
   right = ri;
}
```

```
setFlashVariables(255, 0, 0, 15, 0, 500, false, 0, 1);
```

In the code below, currentMillis is the number of milliseconds since the program started. flashInterval is the number of milliseconds specified between flashes. So, from say, the first significant iteration of this function, currentMillis is say 501, previousMillis is 0 (because it has only been initialized), so if flashInterval is set to 500, the result of the if statement is true. previousMillis is now assigned to currentMillis so that in another 500 milliseconds the if statement will pass again.

```
//Timer interrupt to flash neopixels
void flashingSignal(){
  //Get the amount of milliseconds since the program started
  unsigned long currentMillis = millis();
  //Every n milliseconds (dictated by flashinterval)
  //the interrupt will fire
  if((currentMillis - previousMillisLights > flashInterval)){
    previousMillisLights = currentMillis;
```

There are two separate checks in the next if statement, flashNum is the current flash number, and flashes is the desired amount of flashes. So, if flashNum is less than flashes

there are more flashes to go and the if statement passes. On the other side of the || operator, waitingForJob is simply a flag that is set when flashes need to happen for an indeterminate amount of time. This is used for when the route is being sent to the Flora, and the NeoPixels inform the user that the data transfer is in process. Once inside the if statement, if the lightOn flag is false, the lights are turned on a specified colour, otherwise they are turned off.

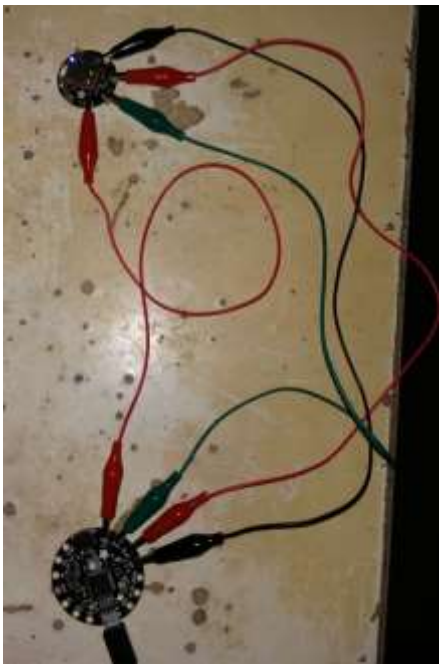
```
//While there are flashes left of still waiting for
//latlng array to fill up
if(flashNum < flashes || waitingForJob){
  flashNum++;
  if(!lightOn){
    //If lights are off, turn them on
    changePixel(left, red, green, blue);
    changePixel(right, red, green, blue);
    lightOn = true;
  }
  else{
    //If lights are on, turn them off
    changePixel(0, 0, 0, 0);
    changePixel(1, 0, 0, 0);
    lightOn = false;
```

If flashNum is equal to flashes, that means that the desired number of flashes has been reached, and the lights now need to stay off. This is achieved by setting all the parameters in setFlashVariables() to zero and false, and making sure that the lights are turned off before leaving the interrupt.

```
//When flashNum reaches the specified flashes, set
  //the flashVariables and pixels back to zero
  else if(flashNum > 0 && flashNum == flashes){
    setFlashVariables(0, 0, 0, 0, 0, 0, false, 0, 0);
    changePixel(0, 0, 0, 0);
    changePixel(1, 0, 0, 0);
    if(lastFlash) journeyComplete = true;
```

5. Testing

Unit Testing

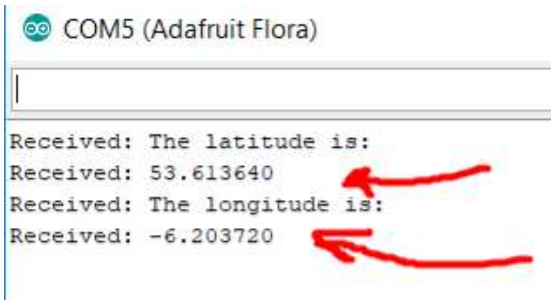


Bluetooth Confirmation Test

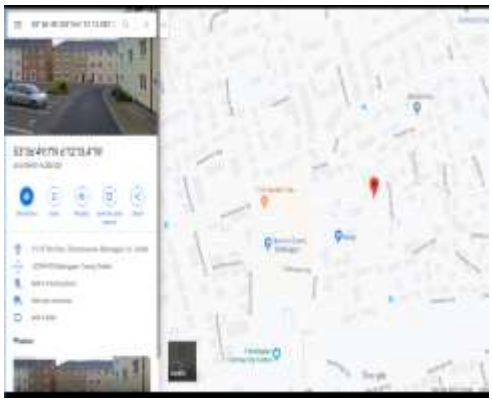
This test was written to check that the Flora's Bluetooth module could receive a message from the phone. I used the Adafruit's Bluefruit app to send the message. As can be seen in the images below, the Serial Monitor registers that the bluefruit is sending the confirmation message "0" after receving a message from the phone, and the app (left) receives the confirmation message.

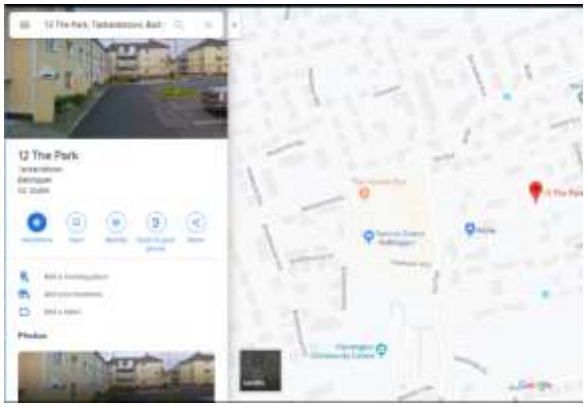
GPS Fix Test

This test checks that the GPS can get a fix on the current location. Below is a screenshot of the Serial Monitor showing the output of my test program, with the latitude and longitude of my location at my home.



Below on the left is an image of the latitude and longitude values inserted into google maps and their position on the map, and on the right is my address input to google maps.



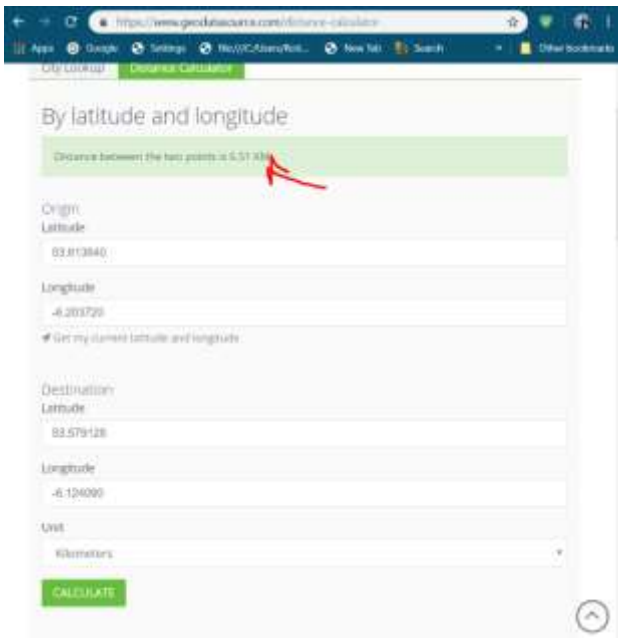


As is evident from the photos, the GPS can fix location to a good degree of accuracy.

LatLng Distance Test

This test checks the distance function for calculating the numbers of meters between two latitude and longitude values. In my test program, I used the latitude and longitude values of the my house and the garda station in my town. I printed out the result of the distance between the two locations in the Serial Monitor.

This result from my test program matched well with an online calculator for latitude and longitude distance, which can be seen in the image below.



Compatibility Testing

I tested the android application by using it on other people's phones to see if it would work as well as it should. The application is to launch and fixate on the user's current location, I tested to see if I could type in the search box and select the predictions from the google autocomplete. I then tested pressing the Bluetooth button to see if the app on the different phones will connect to the Flora Boards Bluetooth Module. Then I pressed the cycling logo to see if the route would be displayed.

I approached this by asking some of my friends that had Android devices if I could use their phones to test my app. Their phones ranged from top tier phones that were only just released in the market while some of them had phones that had been released from years ago. They also had different software versions as some of them were not on the latest Android version. This was good for me because I could see how well it would work on a range of software versions.

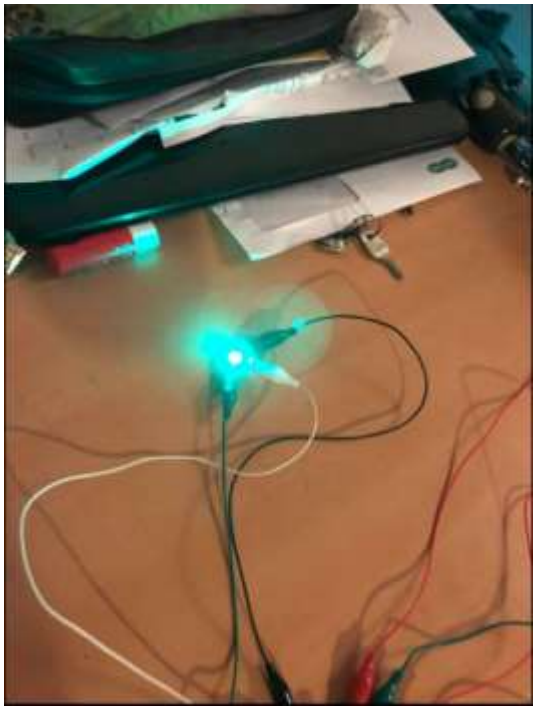
The test was a success. Each phone was able to accept my android application and it worked well on all of them. There was no lag while moving the map around. Each button did according to what they were meant to do for connection and displaying the route.

Functional Testing

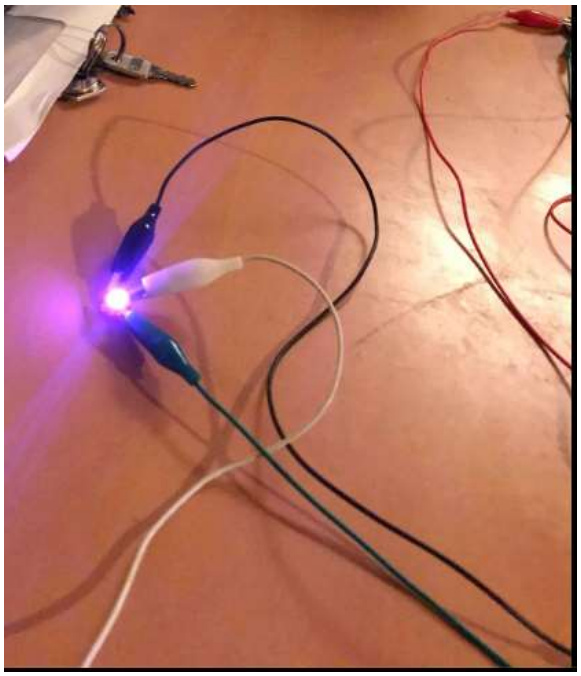
The items to be tested for this was to see if the Neopixel would light up the correct colour when a functionality was being carried out.

I tested to see if it flashed blue when the route was sending.

I tested to see if it flashed green when the route had finished sending.



I tested to see if it flashed purple when the route had finished sending.



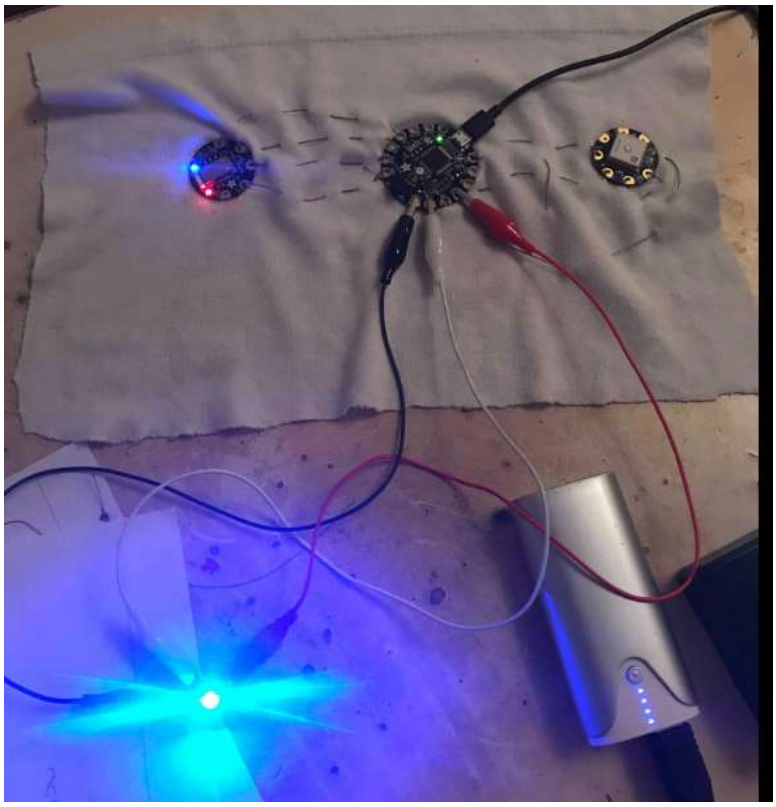
All of the tests passed for functionality.

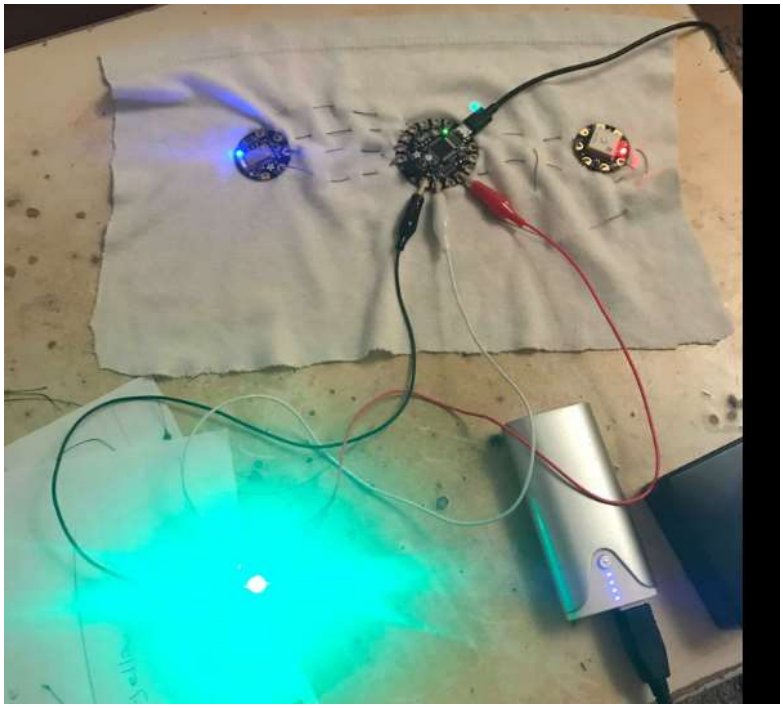
Regression Testing

I did this to check if any risks would arise after my changes to my code and to the hardware. I did this the entire time as I was coding the java ad Arduino. In combination with the unit tests, this is an automated approach and a manual one. Every time I run the program after a change, all tests were ran automatically. I check myself from time to time to make sure everything is the same when doing large changes.

When I was sewing with the conductive thread, I sewed on the Bluetooth module to the Flora board first then tested the rest with the testing wires to see if I sewed it on well and that the project was still working the way it was supposed to.

I then sewed on the GPS module to the Flora board and used testing pins to check the Neopixels to see if the array was being sent and to see if it had finish sending.



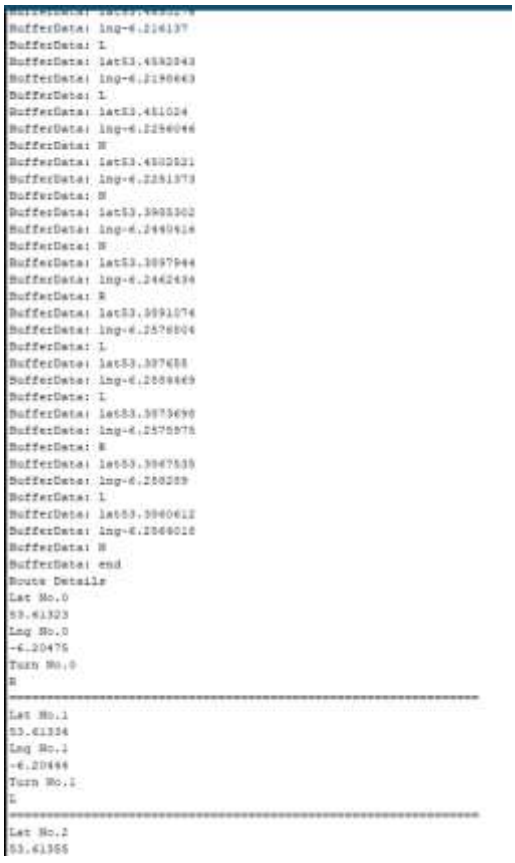


System Testing

I tested the both systems. This was done by seeing if the route downloaded was the one being sent to the Flora.

This is the route being downloaded to the phone that is then being parsed and sent off.

This is what the Arduino receives from the app



6. Problems Solved

Bluetooth and GPS interference

I spent a lot of time, multiple weeks trying to figure out a problem between the GPS and the Bluetooth modules with the Flora, I could not understand why they worked independently of one another but when used together the GPS in particular would be completely unusable. I wrote multiple programs isolating every variable I could think of, and this eventually led me to the realisation that the Bluetooth and GPS could not be instantiated with each other at the same time. This hindered my design somewhat as my code was set up with all modules, Bluetooth, GPS and the NeoPixels running at the same time. I resolved to try and separate out the modules so that at any one time there would only be either the Bluetooth or the GPS running at any one time. This led me to the design as it is now, with one half being tethered to the phone and the other half only activated when the phone disconnects.

Sending Array to Flora via Bluetooth

This problem was difficult to solve because there were so many things that could go wrong when sending data to the Bluetooth module. When I started working on this problem I wasn't aware that the Bluefruit could only accept 20 bytes at a time, could not accept space characters, and was unpredictable in how it received data. I was able to fix these problems by limiting the size of the strings I sent to under 20 characters, using a regular expression to remove all space characters from the data to be sent, and instead of trying to send data at a fixed rate and relying on the bluefruit to run smoothly, only sending the first member of the array, and then waiting for confirmation of receipt of the message, and letting that trigger the next entry in the array

Testing the application

Initially the project could only be tested on an emulator. This was sufficient for developing the logic behind finding the location of the next point in the route and what instructions needed to be displayed. However, it proved to be limited for several reasons. The emulator had no Bluetooth capabilities which meant I was unable to test it with the display. It was also not suitable for testing the location tracking of the user. To get around this I decided to get into the developer options of the phone, which is hidden by default and must be found online. In my case it required hitting the OS version number in the settings five times for it to be revealed. It also requires a specific permission to be included in the android manifest.

7. Future Work

After this I would love to achieve the full functionality I had aimed for with my design and code through using better hardware on a more robust system. In the future I have aspirations to develop “Which Way?” into more than just a direction system for bicycles, but to also include motorcycles in the target user base. I also see it being used on other types of OS’s not just Android. I see it being a website too if you don’t have a phone. The design would be a lot more slicker.