**Technology Report**

**For**

**DigitalDashboard**

**Version 1.0 in progress**

**Prepared by:**

**Karan Raj Kanwar**

**Zhill Patel**

**Darren Prong**

**Humber College**

**School of Applied Technology**

**February 6, 2019**

# Declaration of Group Authorship

We, Karan Raj Kanwar, Zhill Patel and Darren Prong confirm that this work submitted for assessment is our own and is expressed in our own words. And uses made within it of the works of any author, in any form (ideas, equations, figures, texts, tables, programs), are properly acknowledged at the point of use. A list of references used is included. Please refer to chapter 2 Overview for the work breakdown.

**Signed:**

**Student Number:**

**Signed:**

**Student Number:**

**Signed:**

**Student Number:**

**Date:**

# Abstract

In today’s society people enjoy knowing statistics for their travelling periods, but having a bike or a scooter can’t provide real-time information or entertainment. A store-bought scooter or bike doesn't come with a tracker or entertainment but with the DigitalDashboard you can track your speed and listen to music at the same time. Just by simply mounting the portable device on the handle or the body of the source of transportation you can start receiving the real-time travelling speed, location, and music for entertainment purposes. We will demonstrate how our device gets mounted on a bike and scooter, how to use the DigitalDashboard device and its mobile application counterpart.

Table of Contents

[Declaration of Group Authorship 2](#_Toc7103181)

[Abstract 3](#_Toc7103182)

[Revision History 5](#_Toc7103183)

[1.Introduction 7](#_Toc7103184)

[2. Overview 9](#_Toc7103185)

[2.0.1 Database and work breakdown (2019-01-24) 9](#_Toc7103186)

[2.0.2 Application and work breakdown (2019-01-24) 9](#_Toc7103187)

[2.0.3 Hardware and work breakdown (2019-01-24) 10](#_Toc7103188)

[2.0.4 Database and work breakdown revised (2019-03-06) 10](#_Toc7103189)

[2.0.5 Application and work breakdown revised (2019-03-06) 11](#_Toc7103190)

[2.0.6 Web/Hardware and work breakdown revised (2019-03-06) 12](#_Toc7103191)

[3. Build Instructions 13](#_Toc7103192)

[3.0.1 System Diagram 13](#_Toc7103193)

[3.0.2 Bill of Materials/Budget 15](#_Toc7103194)

[3.0.3 Time Commitment 18](#_Toc7103195)

[3.0.4 Mechanical Assembly 19](#_Toc7103196)

[3.0.5 PCB/Soldering 22](#_Toc7103197)

[3.0.6 GUI Creation and Programming 30](#_Toc7103198)

[3.0.7 Setting Up the Adafruit 2.8 Inch PiTFT Screen 32](#_Toc7103199)

[3.0.8 Power Up / Raspberry Pi setup 36](#_Toc7103200)

[3.0.9 Acrylic Case Enclosure 37](#_Toc7103201)

[3.0.10 Unit Testing 44](#_Toc7103202)

[3.0.11 Android Application Setup 46](#_Toc7103203)

[3.0.12 Google Firebase Real-Time Database Setup 51](#_Toc7103204)

[4. Results 52](#_Toc7103205)

[5. Data 53](#_Toc7103206)

[7. Conclusion 55](#_Toc7103207)

[8. Bibliography 57](#_Toc7103208)

[9. Appendix A: Glossary 59](#_Toc7103209)

[10. Appendix B: Code 62](#_Toc7103210)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Karan, Zhill & Darren | 2019-02-10 | Added declaration of authorship, abstract and introduction | 0.1 |
| Karan, Zhill & Darren | 2019-03-07 | Added the work breakdown | 0.1 |
| Karan, Zhill & Darren | 2019-04-24 | Completed the technology Report | 1.0 |

# 1.Introduction

The industry related to our device is the health, fitness and personal transport. The reason why it is considered for health and fitness is because of the adaptability of the device and its small portable form factor. As a personal transportation device it can be easily mounted/installed onto a cylindrical surface. This report includes the creation of the DigitalDashboard, device creation procedure, software used and gathering/analyzing results.

The solution we came up with is a portable mounting digital dashboard, this is fairly small to carry and mounts onto almost all circular surfaces some being the handle of a bike or scooter. This device can tell you your speed, location, altitude, longitude and can play music/radio all while being mounted on a surface.

The main problem is that children’s bikes and scooters don’t come with a digital screen for tracking or any source of entertainment. This we solve by allowing users to mount the DigitalDashboard onto their scooter or bike for tracking and or playing music, this can be use on consumer or professional basis.

# 2. Overview

### 2.0.1 Database and work breakdown (2019-01-24)

For our database we will be using the Firebase Real-time database. After some research it seems the Raspberry Pi and the mobile phone can both utilize Firebase and can be connected to move data around. Karan will be handling setting up the required entities from both mobile and the Raspberry Pi device to the database. Zhill will be making sure that the communication of data between the Raspberry Pi and the mobile application is working and that the data is stored in the database. Darren will be creating and modifying the FM radio (online/offline) functionality and hardware connectivity (TEA5767+amplifier connecting to our database and respond to CRUD operations accordingly). As well, a built in MP3 player; all to be used in conjunction with the LCD application.

### 2.0.2 Application and work breakdown (2019-01-24)

For our mobile application we will be working with Android Studio version 3.3. There will be 3 major components of creating the app. First one being creating a clean layout for the UI and having an aesthetically pleasing icon. The second component being the logic working correctly and having a mobile device communicating with the Raspberry Pi. Lastly getting the mobile device to push and pull data from or to the database. Karan will be handling database creation and the implementation in the application source code. Zhill will be handling some of the functionality of the application, for example, making sure the application will show the data that it got from the sensors. Darren will be handling loading online radio streams when available, and receiving, displaying and saving all local radio stations on the FM band; as well, loading potential MP3’s for the users’ device directly to the application. Our main system feature would be grabbing the speed from the accelerometer and displaying it on the screen, and having the functionality to play music all in device. For the application to run at its best performance an Android device running on 7.0.0 is required.

### 2.0.3 Hardware and work breakdown (2019-01-24)

For setting up our hardware we are going to need a practical sized enclosure for our devices which still holds its purpose of being a portable screen, also we need to attach everything together to get it all to function correctly. We are also required to create the GUI and program on the Raspberry Pi to get our application to work. Karan will be working on the enclosure and the program. Zhill will be working on the hardware portion of the sensors, making sure that it will be able fit inside the enclosure properly. Darren will testing the prototype at each stage of development and also assisting Zhill in fabricating a new PCB with all three sensors and necessary accessories (antenna, AUX port, speakers, etc.) integrated together into a single component. For the application to run at peak performance the latest raspberry pi 3b is required.

### 2.0.4 Database and work breakdown revised (2019-03-06)

For our database we are utilizing Google Realtime Firebase database for both the mobile application and the Raspberry Pi B database. The mobile application allows for reading and writing into the database, our plan was to implement a music database data structure but we ended up discarding that idea as it isn’t really need. We have a user registration and sensor data structure which saves the UID and allows users to access their own information and sensor readings, the sensor reading just stores the UID and the sensor information and can be pulled up when required by user. As for the Raspberry pi database we are in the works on creating database for sensor and user, the code will be written in python as it is easier to create the database with python. Karan has completed the app and has got it working also communicating with the firebase database. Zhill will be working with the new amplifier design in Fritzing; integrating all three PCB designs and sensors unto the final PCB. The components will be supplied by Darren, and tested by both team members. Next step for Darren is to finalize the design with Zhill and fabricate a new PCB with all three sensors and necessary accessories (antenna, AUX port, speakers, etc.), then test the connection with our Firebase Database via our application on the Raspberry Pi. The final enclosure is still to be designed, as this needs the final PCB dimensions in order to be accurate.

### 2.0.5 Application and work breakdown revised (2019-03-06)

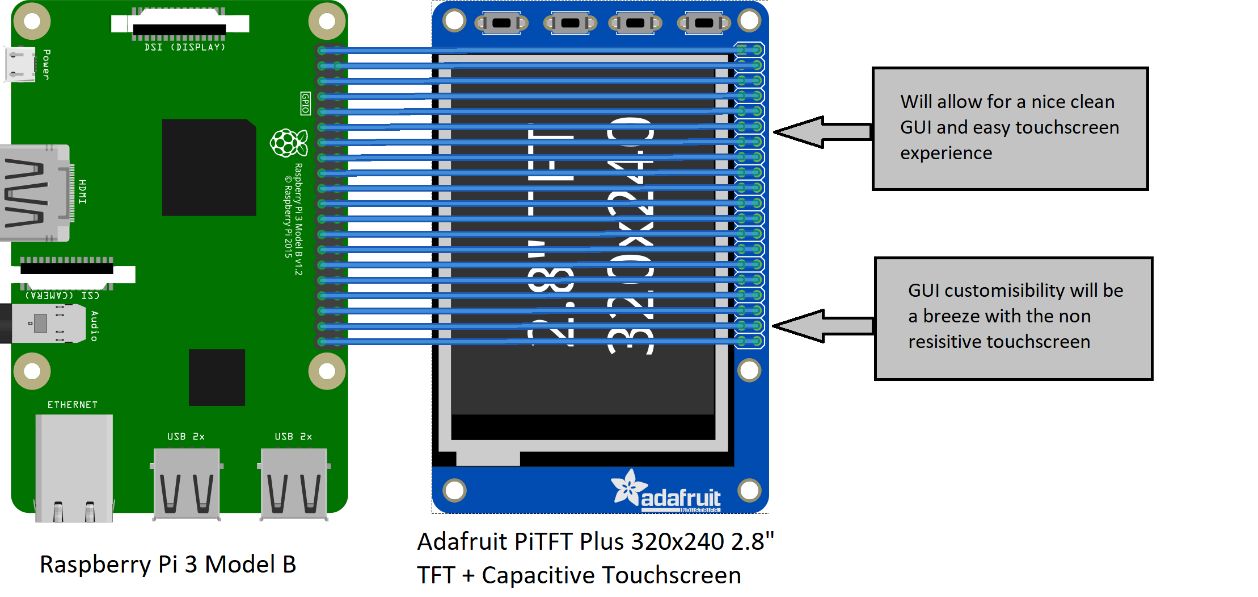
Our device has several pieces which can be separated into small work projects for each team member. Karan has completed the majority of the applications functionality, however, our main display GUI (the one is mainly used on the LCD) is still in development. Zhill will be working on the hardware portion, integrating Darren’s design into the final PCB, as well, design of the mounting points will have to be taken into consideration. Darren has to complete the database design, so that when functional, data is received, perceived and acted upon by the application (i.e. tuning the FM radio sensor). As well, Darren has to assist Zhill in designing and testing the final PCB designs and making sure they conform to the enclosure.

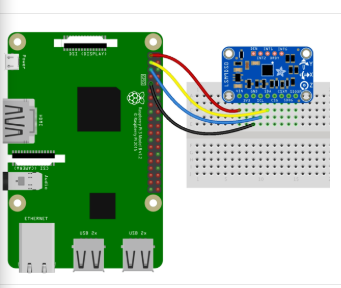
### 2.0.6 Web/Hardware and work breakdown revised (2019-03-06)

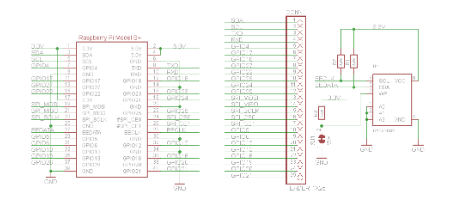
For our hardware at this point of the semester we have full connectivity between the mobile application and the Real-time Firebase database but the Raspberry pi still needs a case which can hold the pi, global positioning sensor, accelerometer sensor, speaker and the custom built PCB. So far Karan has started on the GUI creation for the raspberry pi and is trying to get it to communicate to the Firebase database. Both Zhill and Darren have assisted Karan in the applications development into a functional software platform for our device.

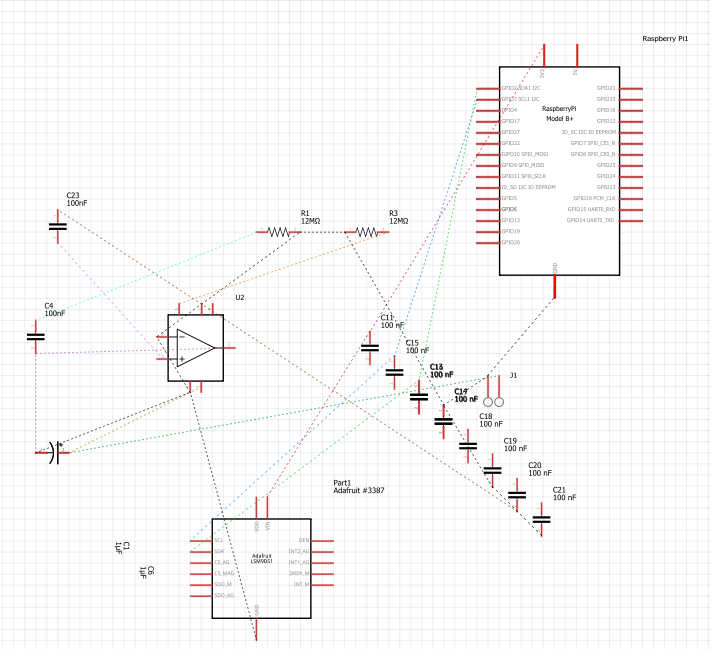
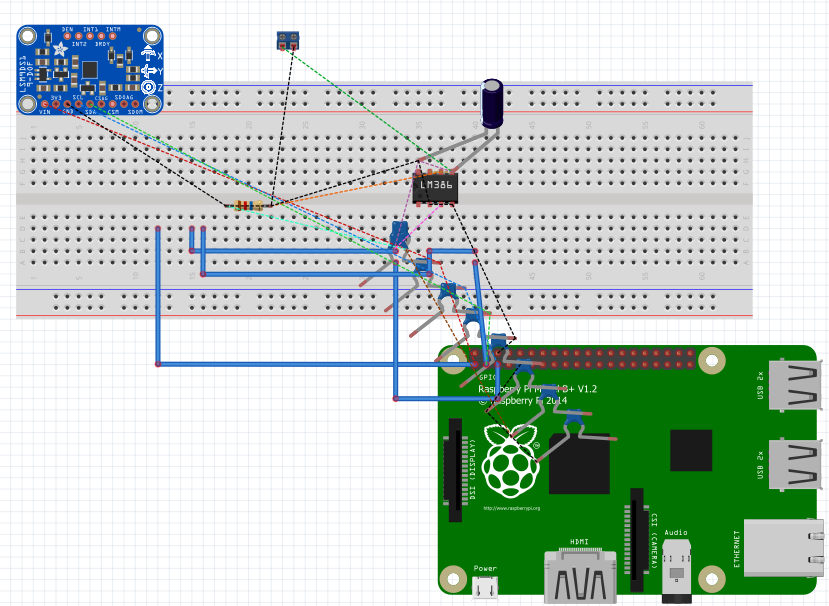
# 3. Build Instructions

# 3.0.1 System Diagram

[](https://raw.githubusercontent.com/KaranRajKanwar/DigitalDashboard/master/Blog%20Images/SystemDiagram.png)







# 3.0.2 Bill of Materials/Budget

| **Part Name** | **Part Number** | **Source** | **Price with shipping & tax** |
| --- | --- | --- | --- |
| Raspberry Pi 3 Model B | B01CD5VC92 | [Amazon](https://tinyurl.com/yd8zt33y) | $55.99 |
| Raspberry Pi Acrylic Case | N/A | [Prototype](https://tinyurl.com/y7lkpouc) Lab | N/A |
| Raspberry Pi 5V Power Supply | B06Y431Y27 | [Amazon](https://tinyurl.com/ya4kcvr3) | $19.13 |
| SanDisk 32gb microSDXC card | B073JWXGNT | [Amazon](https://tinyurl.com/ydhm22or) | $19.65 |
| 2 Mini Raspberry Pi Heatsinks | B014KKY19G | [Amazon](https://tinyurl.com/yawe5tvz) | $11.98 |
| HDMI Cable | B014I8SSD0 | [Amazon](https://tinyurl.com/y6ups5wd) | $12.15 |
| Adafruit PiTFT Cap Touchscreen | B00XW2OI6A | [Amazon](https://tinyurl.com/ybco5so3) | $68.20 |
| Adafruit Ultimate GPS Hat for Raspberry Pi | 2324 | Adafruit | $72.44 |
| TEA5767 FM Radio Receiver | N/A | Odd Wire | $19.00 |
| Adafruit 9-DOF Accel/Mag/Gyro+Temp Breakout Board - LSM9DS0 | 2021 | Amazon | $42.63 |
| Wireless Mouse | N/A | [Home](https://tinyurl.com/ycb737cn) | N/A |
| LM386 Amplifier 8 pin | N/A | Amazon | $3.00 |
| Wireless Keyboard | B0173QNVT0 | [Amazon](https://tinyurl.com/ydh8q29g) | $20.00 |
| Portable Power Bank | N/A | Home | N/A |
| Total price |  |  | $430.00 |

The parts list above is a collection of parts from all three members of the group. The Adafruit Ultimate GPS Hat price before shipping and tax was $44.95, but due to shipping fees from DHL and taxes the cost went up to $72.44. Your budget will vary from mine, as I had a number of components and tools and access to a prototyping lab which greatly decreased my overhead costs. I also had to buy additional components that were not originally on the budget or parts. As well, my major mistake of frying a Raspberry PI (which could set you back another $50.00) wasn't included in the budget. The amplifier originally was planned to be the Class-D amp supplied by buyapi.ca, however I had to put together the LM386 due to time constraints. Also, in the first iteration of my device, the amplifier and speaker were both attached to an external breadboard; this amplifier was modified slightly for the capstone project and added to the internal circuitry.

Looking back however, the Class D amp would be a better option; that’s why I left it on the final parts list. My total budget by the end of the project was $122.68CAD. I had initially proposed a $108 total budget. This cost shouldn’t be considered in our total cost for our capstone project as only one Raspberry Pi is needed, and we were able to share or trade for components.

These are the costs to be expected to create a standalone FM radio receiver, at least at a minimum.

This was the first attempt at wrangling up some of these cheap sensors and my second attempt at putting them into working order, mistakes were made and lessons learned. Often the cheapest route gets you the wrong item or you get nothing at all.

Therefore, go for the one that works, and is quick. I bought the back up at oddwires.com for $15.25USD shipping included. The Pi 3B+ is the preferred platform in my opinion, the others are slightly more cost effective but have slower boot times/compilations/responsiveness etc.

Again, these prices are location dependent, as your geographical situation could greatly increase shipping costs or parts availability. As well, in our prototyping lab, we have access to a $20,000 laser cutter, a lot of spare components and tools, the expertise of professionals, and a great deal more time.

# 3.0.3 Time Commitment

This project can be completed in 7 hours with ease considering you have all the code and design files provided.

* Settings up the raspberry pi with Raspbian OS, getting updates and required software (2 hours)
* Create the PCB which would hold the accelerometer, stereo FM and amplifier (2 hours)
* Solder on the components to the board as instructed (30 mins)
* Attach all the components onto the Raspberry Pi and confirm everything is functioning correctly (30 mins)
* Creating the .cdr file for acrylic file(2 hours)
* Laser cut an acrylic case with the given .cdr file (10 mins)
* Put the acrylic case together with the pi inside and see if everything fit snug (1
* hour)
* Run the Python scripts with the Tkinter library to confirm the GUI is working as required (20 mins)

**Total estimated time**: It would take approximately 9 hours to complete this project if all the files were given before hand.

# 3.0.4 Mechanical Assembly

Adafruit Ultimate GPS Hat:

There was not that work that went into the mechanical assembly of the Adafruit Ultimate GPS Hat, as it did not require a custom Printed Circuit Board (PCB) to be made, it just required some soldering work to be done for it to interact with the Raspberry Pi. After the soldering work is done, you can mount the GPS Hat to the Raspberry Pi, by mounting it on top of the Raspberry Pi’s General-Purpose Input Output (GPIO) pins.

Once the GPS Hat is mounted to the Raspberry Pi, you can power on the Raspberry Pi and do some start-up procedures to make sure that the GPS Hat will work on the Raspberry Pi. When the Raspberry Pi powers on, you can go on the command line terminal and do the following steps:

Step 1: On the command line, run the sudo raspi-config command, which will open up a configuration page.

Step 2: In the configuration page, select Interfacing Options

Step 3: In the interfacing options, find and select on Serial, on the first prompt select No and then on the second prompt select Yes, this will enable the Pi’s Serial ports

Step 4: Exit the configuration page and then run the command sudo shutdown -h now to shut down the Pi safely

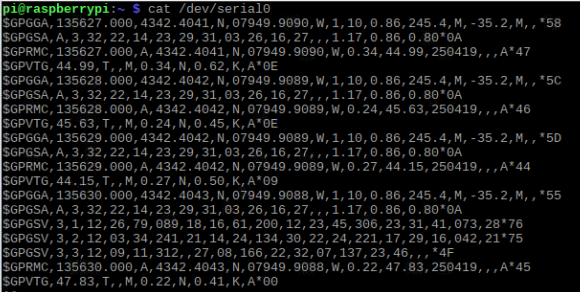
Step 5: Re-power the Raspberry Pi and the GPS Hat will start to work

After the configuration for the GPS Hat and Raspberry Pi, you can start to test the sensor, but first you have to wait for the GPS to catch a signal. Depending on your location, satellite configuration and other factors, the GPS Hat can catch a signal as fast is 45 seconds to as slow as half an hour or longer to get a signal. You can tell if your GPS Hat has a signal or not by the red Light-emitting Diode (LED) the sensor has, if it is blinking every other second the GPS does not have a fix, if it is blinking every 10-15 seconds the GPS has a signal and you can start to run the basic test for the GPS. The basic test for the GPS Hat is fairly simple to do, on the command line you can do these following commands to test the GPS:

Stty -F /dev/serial0 raw 9600 cs7 clocal -cstopb

Cat /dev/serial0

The following commands will set up the GPS and start to read its raw National Marine Electronics Association (NMEA) data that it is outputting. The screenshot below will be the output you will be getting from the sensor.



Adafruit 9-DOF Accel/Mag/Gyro+Temp Breakout Board - LSM9DS0:

For the Adafruit 9-DOF Accel/Mag/Gyro+Temp Breakout Board - LSM9DS0, there was some work that went into the mechanical assembly of the sensor, but there was little configuration that went with the breakout board. The breakout board requires some soldering work to do by soldering some pins onto the board. After the soldering work is complete, you can move on to setting up the breakout board by connecting it to the pins of the Raspberry Pi. For this, you would need to create a custom PCB or for testing purposes you can use a breadboard. Once you have set up the breakout board with the Raspberry Pi you can have to configure the Raspberry Pi to have the Circuit Python library and the LSM9DS0 library. You can do this using the following command on the command line terminal:

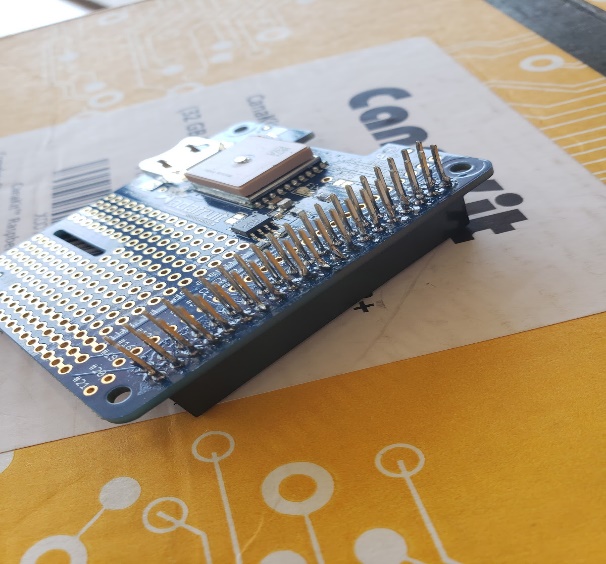
sudo pip3 install adafruit-circuitpython-lsm9ds0

After running this command on the command line, you can begin to code to get the sensor data using programming languages like Python and C Programming. You can find some test code from the Adafruit website under the Adafruit LSM9DS0 Accelerometer + Gyro + Magnetometer 9-DOF Breakouts page.

# 3.0.5 PCB/Soldering

Adafruit Ultimate GPS Hat:

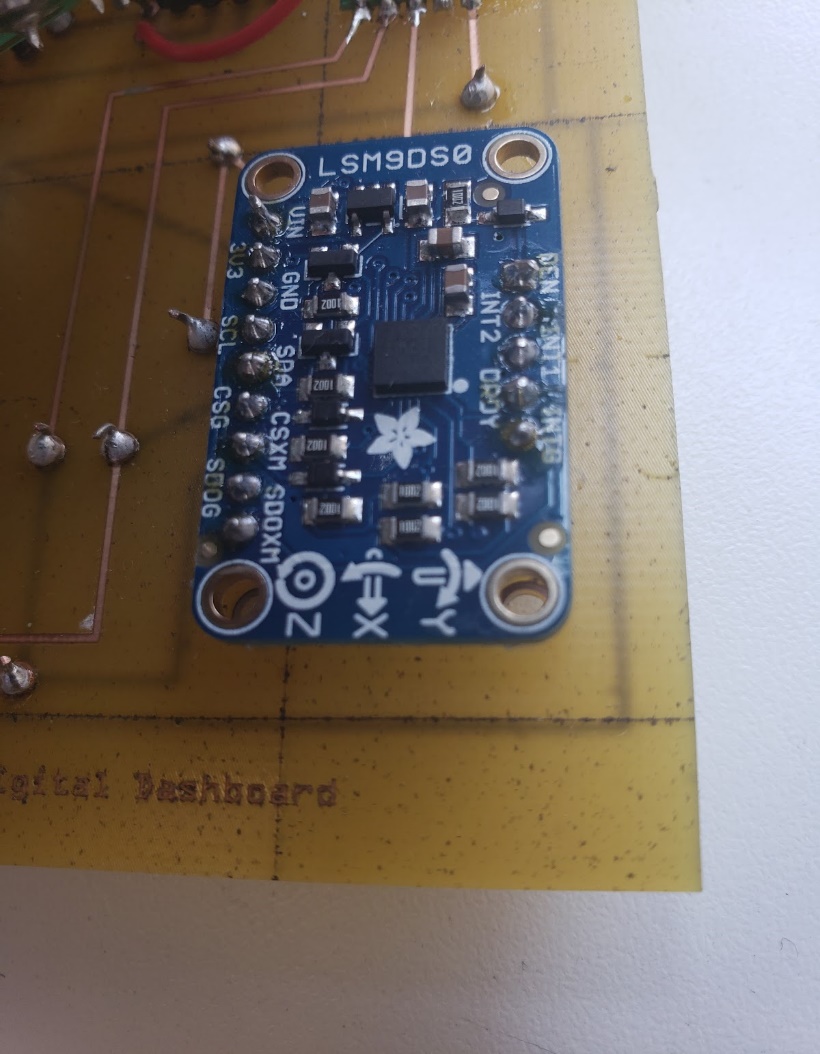
Since the Adafruit Ultimate GPS Hat does not require a custom-made PCB for it, there is no PCB to be made here, but there is some female to male connector header pins that are needed to be soldered onto the GPS Hat. You will need a 2x20 GPIO header pins to solder on to the GPS Hat. you will need to solder all of the pins to the GPS Hat. The figure below is how the solder is going to be on the GPS Hat.



For the project, we decided to not to cut off the excess pins of the GPIO header because we needed the pins to connect to the PCB and the Adafruit Capacitive Touch Screen.

Adafruit 9-DOF Accel/Mag/Gyro+Temp Breakout Board - LSM9DS0:

The Adafruit breakout board required a custom-made PCB to be made, along with some soldering to do. For the breakout board, you would need some IDE header pins to solder on to the board. The figure below shows how the IDE header pins is supposed to be soldered on to the board.

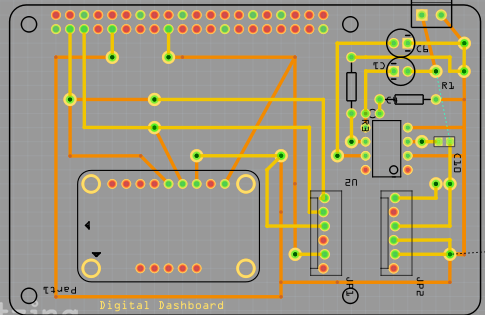


The following PCB was designed was created for the breakout board on the application Fritzing, this PCB design gave us an idea of how the breakout board should be connected to the GPIO pins of the Raspberry Pi. The breakout board uses 4 of the Raspberry Pi’s GPIO pins, as it uses the Voltage in (Vin), Ground (GND), the SDA and SLC pins.



Digital Dashboard PCB:

The following PCB was designed and used for the project, the Gerber files for this PCB design is in the Digital Dashboard repository on GitHub, which can be accessed by others to use and make changes of their own to the project. This version of the PCB design has the Adafruit breakout board, the TEA5767 FM Stereo Radio Module and LM386 Amplifier that were needed for the project.



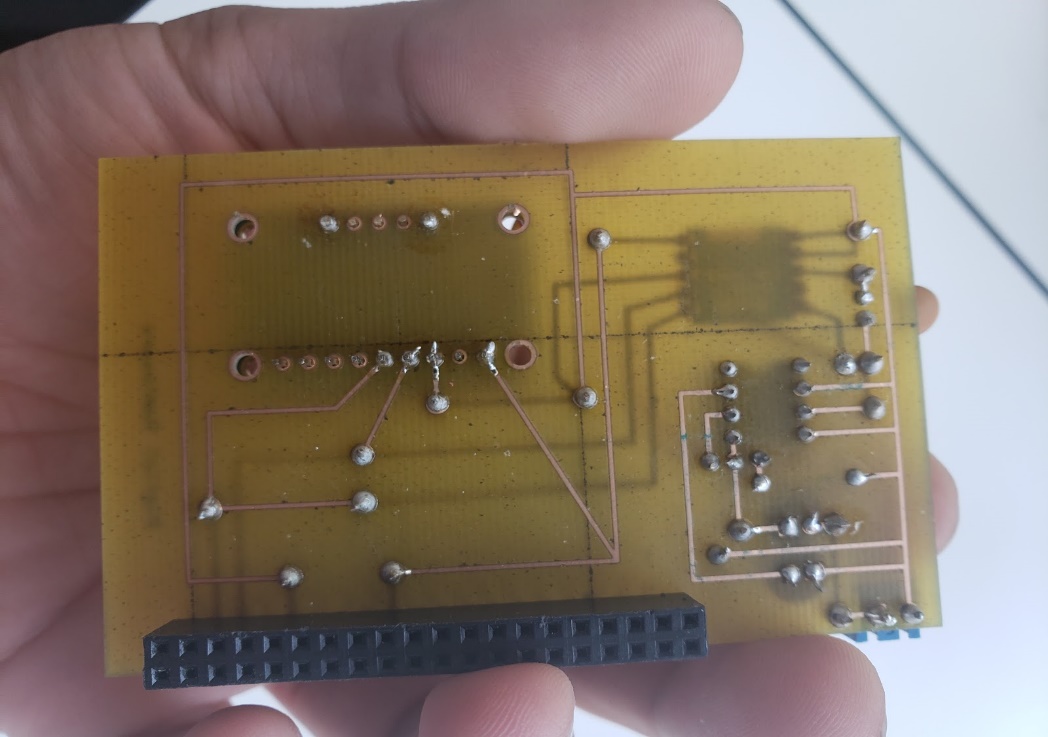
The construction of the PCB that was used for the project happened at the Prototype Lab that is located at Humber College. Although, the creation of the PCB can be made at any third-party production facility, as the Gerber files are accepted universally, as they are the industry standard.

After the creation of the custom-made PCB, you will need to solder the components on to the PCB. you will need to solder on the breakout board to the PCB, the TEA5767 FM Stereo Radio Module, a screw terminal, and the components of the LM386 Amplifier which are resistors and capacitors. Once all of the components are soldered on to the board, you have to solder the VIAs to fix all of the connections together. To solder the VIAs, you will need to strip a piece of wire and solder the wire from the top of the PCB and the bottom of the PCB. The figure below shows the solder the connections from the top of the PCB and the bottom of the PCB.

Top of the PCB:



Bottom of the PCB:



Once the parts are in hand, the design files printed and assembled, here’s what you should have in hand:

Start by attaching the sensor to the PCB, I used hot glue the first time around as Fritzing doesn’t have the TEA5767 included within the parts list. I was able to compromise with a crucial bit of advice from the prototype lab staff here at Humber. What you can do in the program is set the scale down to a few millimeters per square, this allows fine tuning for your component spacing that would normally be impossible with the default scale of 1cm squared. The solution is this, use a series (5 on each side) of surface mount capacitors all spaced at half millimeter apart (in the y-axis), 11 and ½ millimeter in the x-axis. These will act as your trace points for the rest of the circuit. Trying to keep the sensor still as you solder the first connection, it’s difficult so you might need a second pair of hands. Also, be doubly sure of your orientation, the solder point on the sensor is extremely small and is difficult to correct afterwards.

Next will be to solder the via's. I recommend that you take a small amount of 20gauge wire (1cm in length) strip only one end and then insert into all the via's. This will hold it inside the via briefly while you solder the flip side. Solder the entire side with exposed wire first, then flip, remove the insulation, and solder that side for each via.

Double check the shape of each via and how the solder "flows" from the wire onto the copper pad. This ensures a solid connection. It should appear like a sloping hill, or a "Hershey Kiss".

Next, the 40-pin stackable header, along with the screw terminal to be used with your speaker.

Soldering all 40-pins isn't necessary, I only soldered the pins I was using, and two on the far end of the header.

Next up is the LM386 amplifier circuit. Fritzing in this case does have an appropriate through hole template for the LM386, so find it and add it to you board. Here, I received another piece of valuable advice, instead of soldering the amp directly to the board, solder an eight-pin socket the you can insert the amp inside. If the amp fries for whatever reason, you can just pop it out and put a new one in its place. Next, solder your four capacitors into the proper positions (you only need to solder the backside). I have only one resistor on my design, however, if you intend to increase the gain from the amp, you’ll need a second one along with a 10microF cap from pin 1 to pin 8. The volume was a little to high for me at this gain (I believe a gain of 200) so I omitted this part.

In my last design I had an external power supply pump in 12v for the LM386, however in the Digital Dashboard I wanted it to be self-contained. I purchased two NOMA A23 12v batteries from Canadian Tire for $12CDN, and harvested a battery terminal from an old doorbell button. I scratched off the tracing from its ground and positive solder points at the back of the PCB, a soldered on two color correct jumper wires, these then were solder into the positive and negative through holes on my PCB. I haven’t been able to secure it down properly to the board and as a result, these solder joints don’t exactly stay connected for very long if you don’t baby it. Luckily the doorbell button was designed for a 12v battery, so no problem with the fit, however its susceptible to be jostled around and pushed out of its terminals.

You can now solder on a length of wire (up to you how long) to act as the antenna. Solder this in a similar way as you did with the via's. An improvement to this device would be to fabricate your own retractable antenna (which is still just a length of wire). here’s some instructions I found on wikihow.com

*Gather the necessary materials. In order to make a vertical antenna from a coaxial cable, you will need the following materials; 50 ohm (or 75 ohm) coaxial wire with copper shielding, 3/8-inch copper tubing. Calculate the length of your antenna. This will determine both how much of the coaxial cable you have to strip and how long your copper tubing should be. Divide 468 by the frequency to which you want to connect (e.g., 468/108MHz would become 4.3). Divide the resulting number by 2 (e.g., 4.3/2 would become 2.15). Multiply the resulting number by 12 inches to find the antenna length (e.g., 2.15\*12 inches would become 25.8 inches).*

*Cut off one end of the coaxial cable. While you'll want to leave one end of the coaxial cable intact in order for it to serve as the connector, the other end will need to be removed. Attach the tube to the coaxial cable. Slide the copper tubing onto the coaxial cable's stripped end, then slide it down. Solder the coaxial cable's shielding to the tubing. You can do this by removing the PVC (black) shielding from around an inch of the coaxial cable directly below the unshielded part, peeling it back with a pair of pliers to form a lip, and then using your soldering pen to connect the lip to the copper tubing. Connect the coaxial cable to your audio receiver. The remaining coaxial connector should plug into the receiver's coaxial antenna port, which makes the rest of the antenna placement fairly simple. Place the antenna. Once the antenna is plugged in, angle it toward the nearest station and secure it in place if necessary. The fewer obstructions between your antenna and the nearest FM station, the stronger your signal will be. Your coaxial cable may be stiff enough to stand on its own without needing support, but you can use stables or any adhesive to prop up your antenna as needed. (wikihow.com/Make-an-FM-Antenna)*

Next and final step for assembly, attach your PI to the base plate of the acrylic enclosure with the ordered hardware and printed standoffs.

At this stage, you should be able to grab your speaker and plug it into the screw terminals. I’m not sure if my connections weren’t quite perfect or if the speaker had sustained some damage but I heard static from my speaker at this point which I took as a good sign.

You can now attach all parts of the enclosure to the base, and plug in the PCB/device combo into the GPIO pins.

# 3.0.6 GUI Creation and Programming

To create the GUI for our Digital Dashboard we utilized a Python library called Tkinter. In order to use this library, you must confirm that Python is installed on your Raspberry Pi, just use the command “sudo apt-get install python 3” in terminal. Once Python is install please install the dependencies that are required, use the code below in terminal.

sudo apt-get update

sudo apt-get install libatlas3-base libffi-dev at-spi2-core python3-gi-cairo

pip install cairocffi

pip install matplotlib import tkinter as tk

Now to run Tkinter and create a GUI with ease just write this code

# Create the main window

root = tk.Tk()

root.title("My GUI")

# Create label

label = tk.Label(root, text="Hello, World!")

# Lay out label

label.pack()

# Run forever!

root.mainloop()

https://cdn.sparkfun.com/assets/learn_tutorials/8/0/3/screen_01.png

# 3.0.7 Setting Up the Adafruit 2.8 Inch PiTFT Screen

1. Make sure your Raspberry Pi is loaded with an OS, if you aren’t sure how to install the OS please go to sub chapter “3.0.8 Power Up/ Raspberry PI Setup”.
2. Attach the Adafruit 2.8 Inch PiTFT screen to the Raspberry Pi B, but also keep the Raspberry Pi plugged into its primary screen for installing the drivers with ease.
3. Run this installer script in terminal and please follow the steps.

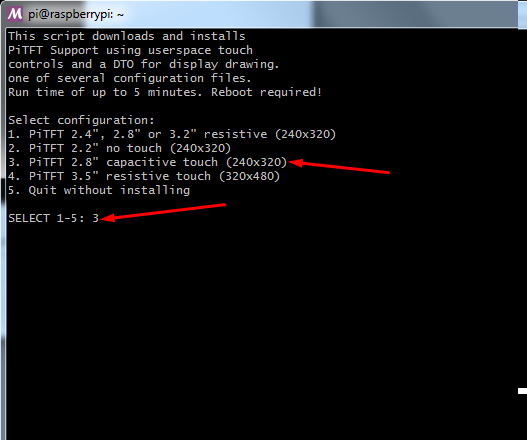
cd ~

wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/adafruit-pitft.sh

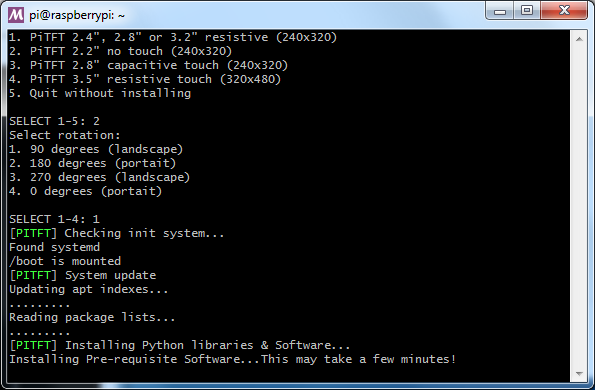
chmod +x adafruit-pitft.sh

sudo ./adafruit-pitft.sh

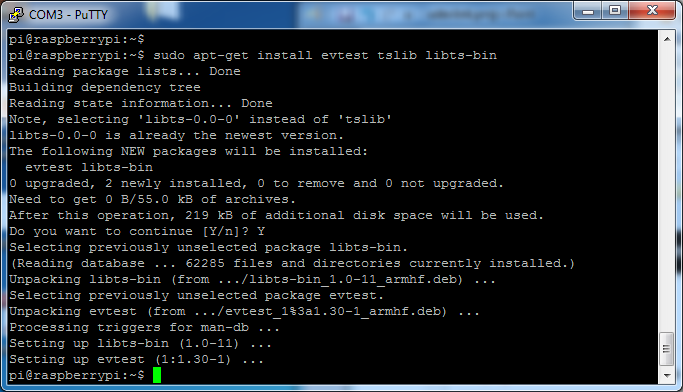
1. You will be sent to a PiTFT Configuration page. Please select the PiTFT screen you are using, in our case we will select “PiTFT 2.8 capacitive touch (240x320).



1. Next select the rotation amount of the screen like below.



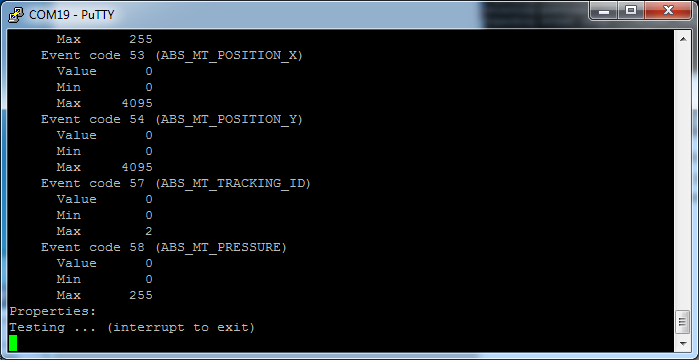
1. After selecting the type of screen you will be asked what you would like to be displayed on the screen, 3 options will be given. PiTFT as Text Console which is recommended for Raspbian Lite, PiTFT as HDMI Mirror which is recommended for Raspbian Full/Pixel or PiTFT as Raw Framebuffer Device.
2. As we have Raspbian Full we will use PiTFT as HDMI Mirror to simulate a full desktop on the screen, if for any reason you would like to change the configuration just run the installer script again to safely configure to your needs.
3. Now the screen should power up the desktop and should replicate the other display, but we still have to configure the pi reading information from the screen.
4. To test if the touchscreen is working correctly you can install the event tester and touchscreen library packages with the code in terminal.

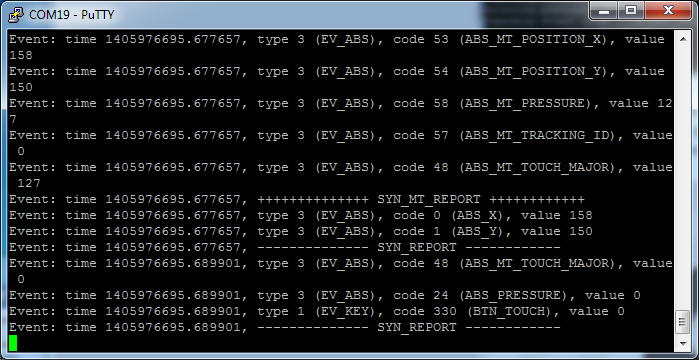


sudo apt-get install evtest tslib libts-bin

1. To use the event tester, use the code below in terminal.

sudo evtest /dev/input/touchscreen





# 3.0.8 Power Up / Raspberry Pi setup

1. Download Raspbian disc image on a computer via ZIP download or torrent.
2. Unzip the downloaded file using any software of choice, I prefer WinRAR.
3. Write the disc image to the micro SD card (must be a minimum of 8 GB), I prefer Win32 Disk Imager.
4. Plug in your choice of mouse, keyboard, display via HDMI, and power supply.
5. Install Raspbian on the Raspberry Pi by inserting the micro SD card into the slot and giving it power.
6. On first boot you should be taken to a configuration screen, you don't have to do anything here, but I recommend changing your account password for security. If for some reason you were not able to see the configuration screen type the following command in terminal.

sudo raspi-config

1. Update the repositories on the Raspberry Pi by running this command in the terminal.

sudo apt-get update

Sudo apt-get upgrade

# 3.0.9 Acrylic Case Enclosure

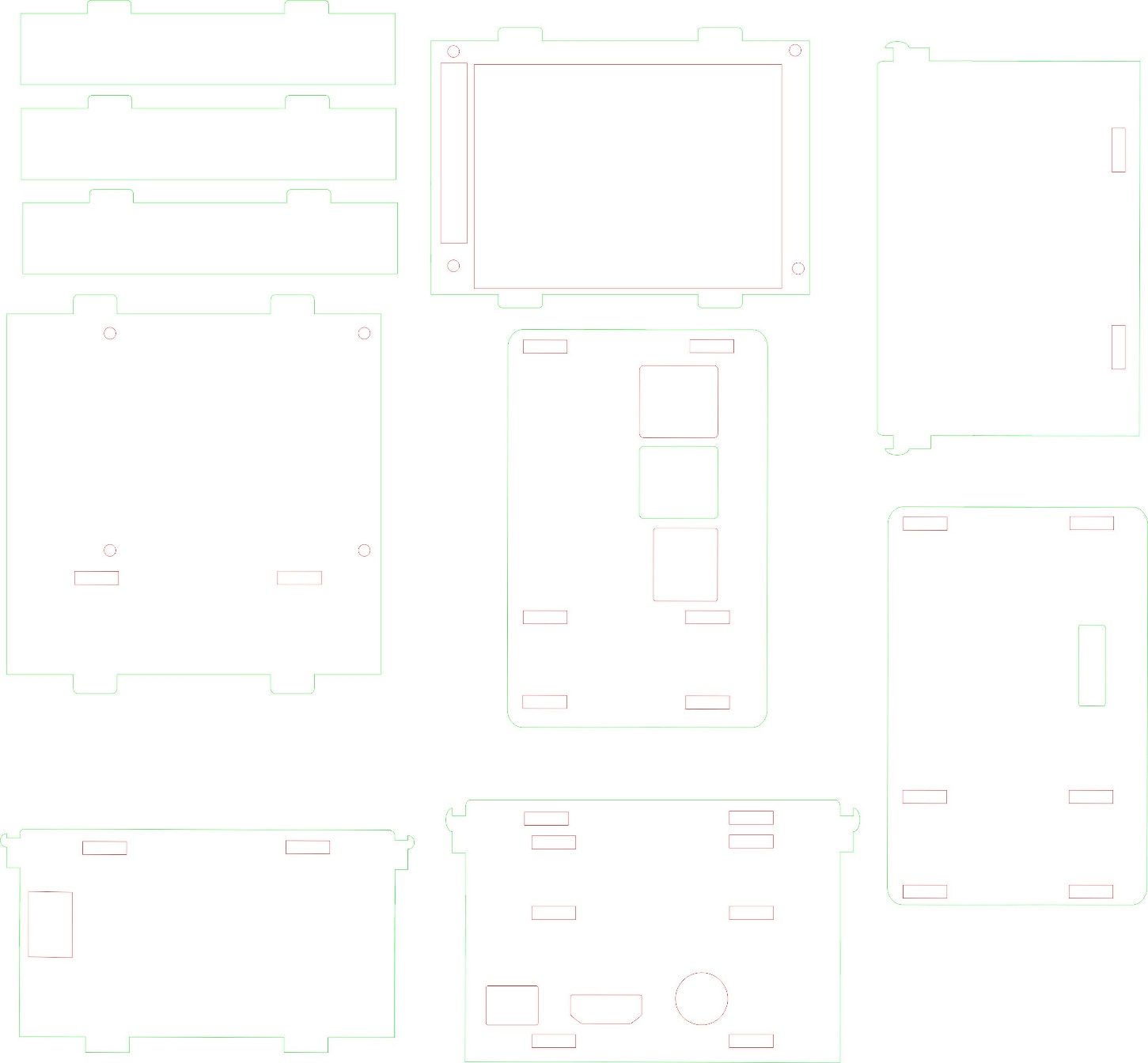
We have acquired some hardware dimensions. Below we have attached some of the required dimensions and their values. You may also find these measurements on our GitHub at <https://github.com/KaranRajKanwar/DigitalDashboard-Final/tree/master/Enclosure%20Measurements> inside the Enclosure Measurements folder. The images state what each measurement is for or shows it within the image. We used the caliper to measure the dimensions of the raspberry pi 3b, PCB, GPS sensor and screen. We got really precise measurements for the screen and Raspberry Pi Ports. Since we were also installing a power bank on the side of the case we got the measurements of the power bank and added that width to the side to extend the case to fit the device snug.

To create out enclosure we used Corel Draw 2019, and the template file that was provided to us in our fifth semester hardware production class. We worked off of that template to create our own raspberry pi case which fits our requirements. We created a higher and a wider case, higher because it requires more head room to fit the PCB and GPS sensor. We had to widen it to give it space for the portable battery that would give it juice for when not plugged into the wall.

Designing this next two major parts will require some trial and error if you haven't used these programs before. Also, if you've ordered your parts and awaiting the sensor in the mail, this can be done in the interim. To design the device PCB, I used the freeware "Fritzing" (which is in BETA at the time of this publication.)

I recreated the PCB layout to include my amplifier circuit alongside the TEA5767. My colleague’s sensor and tracing were also added to the final design. The only change from last semesters fritzing was omitting one screw terminal as I have only one functioning speaker, and our enclosure would only have been able to practically fit a single 3 and ½ inch speaker.

And for the enclosure, CorelDRAW was used on a free 15-day trial as well as access to the program in our lab. I wasn’t the main designer of the enclosure so ill point you to Karan’s or Zhill’s build instructions where you can read more on how that was designed.





# 3.0.10 Unit Testing

Adafruit 2.8-inch capacitive screen:

If the touchscreen works and responds correctly with touches, it's confirmed that it’s been installed correctly but to make sure we can check our i2c address by running this command in the terminal.

Sudo i2cdetect -y 1

Adafruit Ultimate GPS Hat:

If the Adafruit Ultimate GPS is working and responds with the correct NMEA data, then it should be working perfectly fine, but to make sure that the Raspberry Pi can see the i2c address by running the command on the command line terminal:

sudo i2cdetect -y 1

Adafruit 9-DOF Accel/Mag/Gyro+Temp Breakout Board - LSM9DS0:

If the Adafruit breakout board is working, you can run the test code found on the Adafruit website, then it should be working perfectly fine, but to make sure that the Raspberry Pi can see the i2c address by running the command on the command line terminal:

sudo i2cdetect -y 1

FM Radio Stereo :

Credit for the above code goes to "halfluck" on the Raspberry Pi forums. <https://www.raspberrypi.org/forums/viewtopic.php?t=53680&p=419429>

The command to be run here is:

sudo i2cdetect -y 1

This will display all open addresses and will display a hexadecimal value representing the module. We need 0x60. If you see any other address, (other then the other devices you added) there may be some other issue regarding your device; I would look into this guide:

<https://raspberrypi.stackexchange.com/questions/76072/inconsistent-i2c-bus-address-listings-causing-sensor-reading-issues>

or many others like it across the web.

If everything is looking good, you can continue. Now we're ready to play music but most likely you haven't plugged in your speakers. I had two 4.3ohm desktop speakers from an old set I found at my house. One was broken and couldn't play anything, and the other was working just fine. Test them first in your amplifier. Supply a low power frequency to the input of the LM386, shown here:

Here is it breadboarded virtually using fritzing: Once you test has gone through with your speaker, test it out while running the PI hooked up to the amp.

sudo ./radio xx.x //your radio station frequency of choice

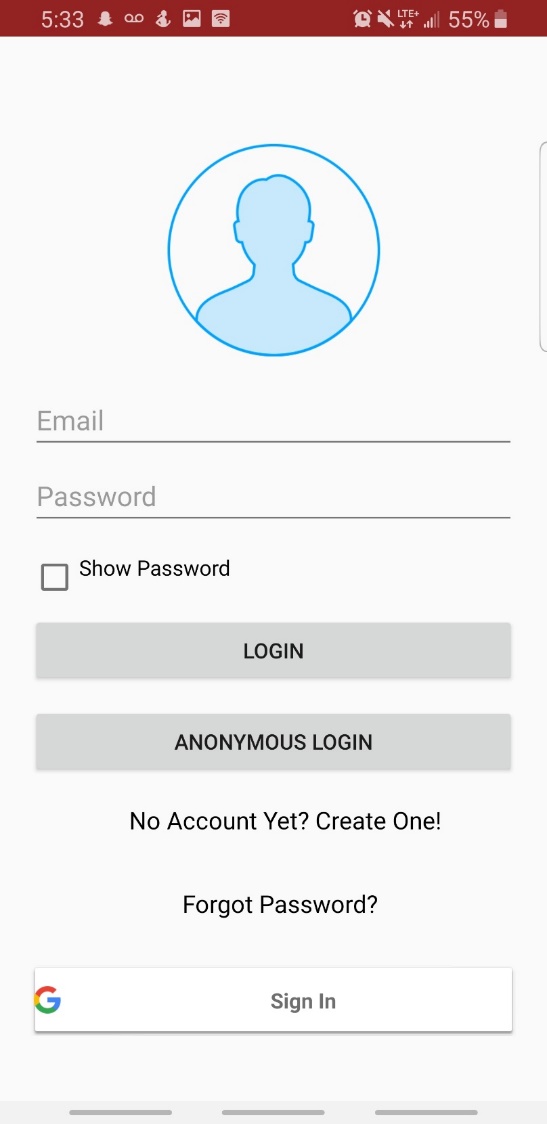
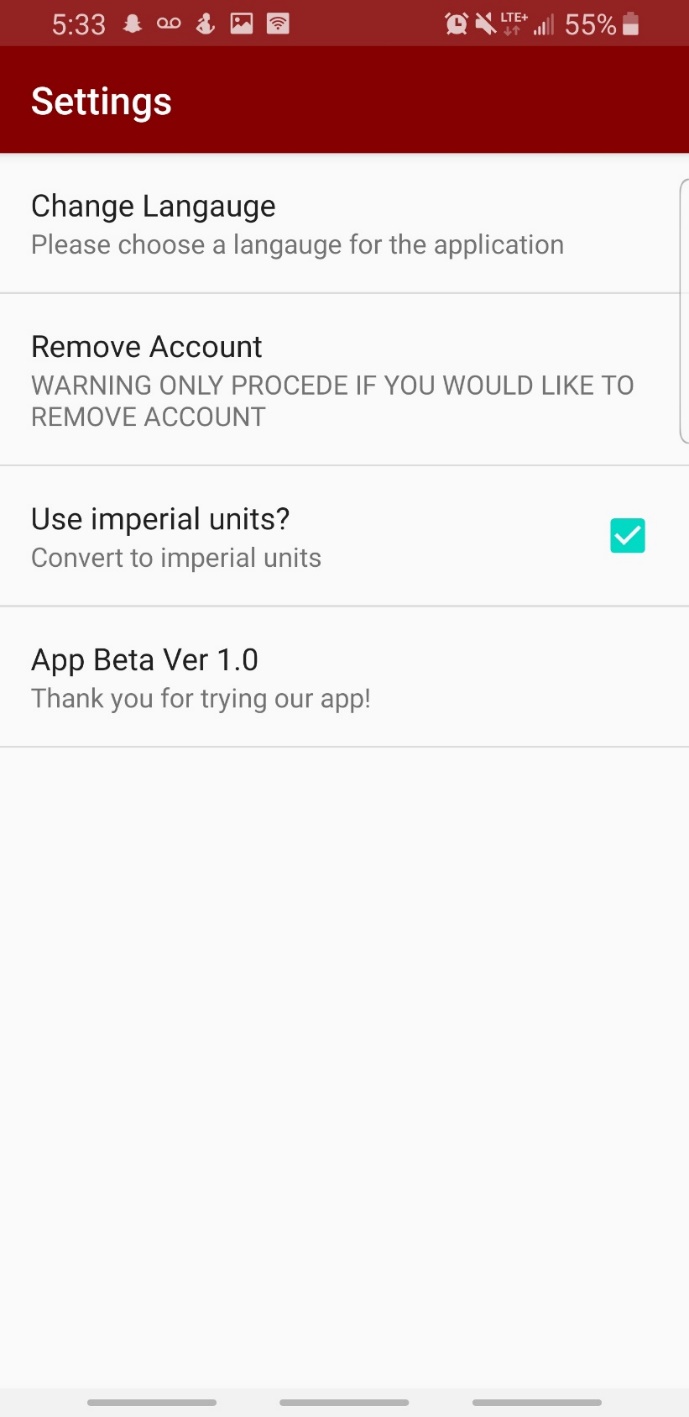
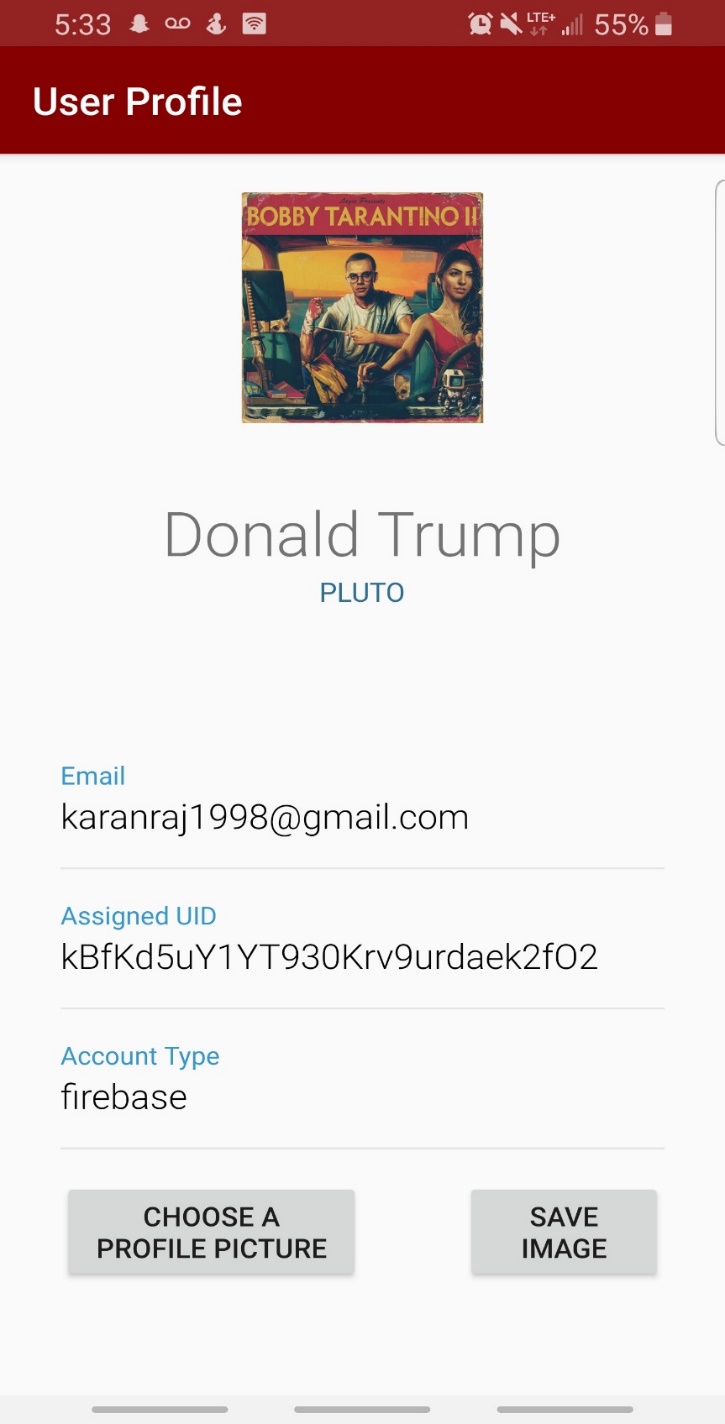
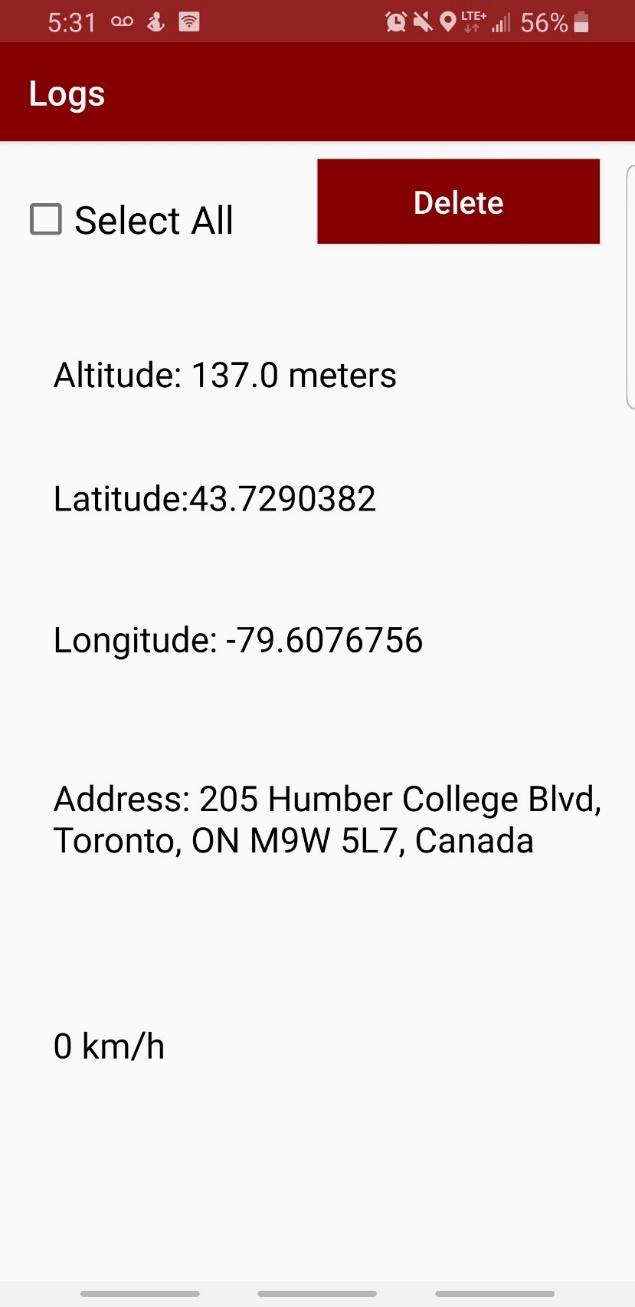
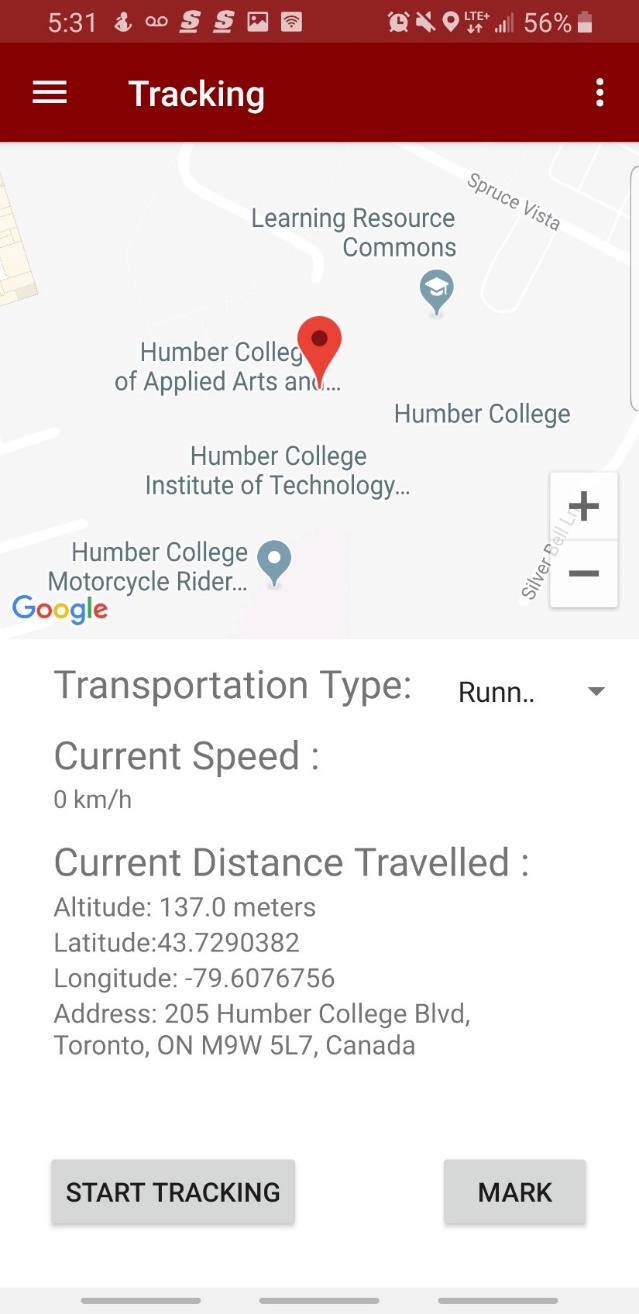
It should come out sounding a little something like this: <https://www.youtube.com/embed/gSB8UnMIgSc>

# 3.0.11 Android Application Setup

The main requirements for this application is API 28 which is the latest android OS build called Pie. For the most optimum performance experience it is recommended this app be used on a Samsung Galaxy s9 on API 28. There are permissions which are required to be enabled by the android system. This is done when the app if first launched, if you don’t enable these permissions the app will not function as it is supposed to. The app will not install on anything lower than API 28. To get the app source code and APK go to <https://github.com/KaranRajKanwar/DigitalDashboardFinal/tree/master/Mobile%20Application/DigitalDashboard>.

The mobile application provides a Digital Dashboard like experience right on your mobile phone, but much more eye pleasing and with many more features. The app allows the user to create an account, recover the password for a pre-created account. You can sign in through the in app created account, your Google account or anonymously. Once logged in a system notification will be placed in your notification drawer telling you that you are logged in, clicking it will allow you to go straight into the tracking page as the user. The tracking page has the Google Maps Api so you can pin point your location, place markers, change map type such as terrain to satellite. Also you pull all the sensor readings on that page by hitting the “TRACK” button, you will be given the speed, altitude, latitude, longitude and address. And after receiving these reading hit the button “MARK” all that information will be saved on a Firebase Real-Time database and placed into the logs page on the app. The app also comes with a built in FM radio and Music player for entertainment purposes, the music player plays music downloaded in the phone. We also have a user profile page which displays all the user’s information and allows you to choose a profile picture which gets uploaded to the firebase storage so it can be displayed for that account on sign in. There is also a settings page implemented within the app allowing users to change the app’s language in runtime, remove account from the Firebase database in app, convert to imperial unit’s checkbox and a developer page. Lastly the app gives the user the option to either sign out or exit the app safely.

The creation of the application had its problems too, some of them just completely crashing the entire app, some making the app function but not push data to the database. An issue we dealt with is spinners in the application, we have a spinner in the application and that spinner allows the user to choose what source of transportation they are using. Where the problem lies is throwing that spinner value to the database, for some reason it doesn’t read the string that’s in the spinner it places the same placeholder in every database push.



# 3.0.12 Google Firebase Real-Time Database Setup

We started with the mobile application database, which utilizes the Googles Real-Time Database. The main information which gets stored in the database is Sensor Information and user registration information. The Sensor information consists of the address, altitude, latitude, longitude, speed and the transportation spinner, all of these values are pushed from the tracking page when the button “TRACK” and “MARK” is clicked. Alternatively, the user registration information consists of the country, email, first name and last name, these values are pushed onto the database from when a user successfully registers within the app. When it comes to pulling that information from the database, we pulled the sensor information into a logs page which displays the latest sensor information. And for the user information we push the values of the logged in user into the user profile page, that page also allows the user to pick a user profile picture and upload that to Googles Firebase Storage. This means whenever that user logs onto the app that picture will pop up for that user. And for the Raspberry Pi Device we configured it to push similar data as the mobile application except in a more barebones format. The Entity in the Real-Time Database is called Raspberry Pi and the values it holds is acceleration, altitude, latitude, longitude, speed. We pull this information into the mobile app by opening the device logs pages and run the device simultaneously. Firebase has a issue which is really worrying, sometimes when we log in the database all the information goes missing. The entity is still there but it stays empty. This is very scary as it compromises the information of our users.

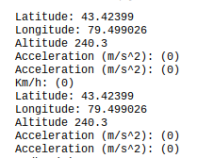
# 4. Results

The Outcome of our Digital Dashboard Device was fairly good but some of the features didn’t come out as planned, for example the GUI is exactly how we wanted it but the functionality is very limited due to the fact that some of the code would not work together. The setup for the creation of our acrylic case was a bit confusing as we didn’t have much experience with the software Corel Draw, we created about 5 different acrylic cases and every time something would give us an issue. We would focus on using Solid Works instead of Corel Draw as it is more informative and graphically pleasing to use. For example, one time the side were too long, the next time the sides were too short just small irregularities which wouldn’t all the case to be perfect. But to state our improvements we can do a better job creating a more attractive functioning enclosure, that would actually mount on top of a handle bar or the vent clip on a vehicle. The device itself is supposed to be a portable mounting device, our device is a bit clunky which makes it hard to mount it on certain surfaces without having it sag down with its height or weight. If we were to recreate this project, we would for sure try and utilize the Raspberry Pi Zero W which is not only a very cheap alternative it is very small and lightweight. That would work in our favor. Also getting a more accurate and speedier GPS would be an improvement we would look at. Our current GPS takes quite a while to load up if its inside a building which wouldn’t be a good feature to have in a working product, it should load up in 5 seconds whereas the current sensor takes 10+ minutes in certain scenarios.

# 5. Data

Raspberry Pi Sensor Readings

For the Digital Dashboard project, we had two sets of data that were retrieved and pushed into the firebase database. The first set of data that we gathered is from the Raspberry Pi, the Raspberry Pi is gathering the data from the Adafruit breakout board and the Adafruit Ultimate GPS Hat, from these sensors the Raspberry Pi gathers data of the Latitude, Longitude, Altitude and Speed from the Ultimate GPS Hat, while from the Adafruit breakout board, the information that is gathered from the sensor is the acceleration. The figure below shows the data that is gathered from both the Adafruit breakout board and the Adafruit Ultimate GPS Hat.



The data shows the current latitude, longitude, altitude, acceleration and speed that the figure was taking in. The latitude and longitude is the correct for the location that we were in and the altitude is correct too. The accelerations and speed are at zero in the figure, due to the fact that we were not moving at the time of the figure was taking, but if you were moving on a transportation vehicle such as a bicycle, then the acceleration and speed would have been the current accelerations and speed values that you are moving at. Also, while you are moving the longitude and latitude will start to change frequently giving the current locations that you are moving at.

Phone Sensor

The second set of data that we were gathering and pushing into the firebase database, was the information that was gathered from the sensors on the phone which is coming from the mobile application that was made for the project. Since the phone’s sensors are a lot better than the Raspberry Pi’s sensors, so for the information that was gathered from the mobile application. The data that is gathered from the phone sensor, is address, altitude, latitude, longitude and speed. The figure below shows the data that is gathered from the phone’s sensor.



The data that is shown in the figure, shows the current address, altitude, longitude, latitude and speed. Like the Raspberry Pi’s sensors, the longitude, latitude, altitude data of the current location that we were in. It also gives us the speed, but it has the same conditions as the Raspberry Pi, as we were not moving at the time, so the speed is going to be zero. If we were moving in a transportation vehicle such as a bicycle or a scooter, then it will show the current speed that you are moving. Along with the current speed, the longitude, latitude and address will start to change as well, as it will start to track the current location that you are in.

# 7. Conclusion

The DigitalDashboards main objective stays to help people track their movement when they need so. The main function of this device is providing accurate speed, longitude, latitude, and a speaker for personal entertainment such as the radio or MP3 music. If the user desires more information, the mobile application makes it’s possible to get a more detailed location utilizing the phones sensors and the correct permissions. The GUI for the device makes it really easy to understand the information given on the screen. The GUI isn’t fully functioning but we can demonstrate exactly how its supposed to look and how its supposed to grab the sensor information from the sensors and place them in the values within the graphical interface.

# Bibliography

**Raspberry Pi Raspbian**

Download Raspbian for Raspberry Pi. (n.d.). Retrieved April 25, 2019, from <https://www.raspberrypi.org/downloads/raspbian/>

**Adafruit PiTFT Capacitive Screen**

Adafruit Industries. (n.d.). Adafruit PiTFT Plus 320x240 2.8" TFT Capacitive Touchscreen. Retrieved April 25, 2019, from <https://www.adafruit.com/product/2423>

**Google Firebase**

Sign in. (n.d.). Retrieved April 25, 2019, from <https://console.firebase.google.com/u/0/project/digitaldashboard-f7a33/over>

**Adafruit Ultimate GPS Hat**

Adafruit Industries. (n.d.). Adafruit Ultimate GPS HAT for Raspberry Pi A /B /Pi 2/Pi 3. Retrieved April 25, 2019, from <https://www.adafruit.com/product/2324>

**Adafruit 9-DOF Accel/Mag/Gyro+Temp Breakout Board - LSM9DS0**

Adafruit Industries. (n.d.). Adafruit 9-DOF Accel/Mag/Gyro Temp Breakout Board - LSM9DS1. Retrieved April 25, 2019, from <https://www.adafruit.com/product/3387>

**Adafruit Speaker 3”**

Adafruit Industries. (n.d.). Speaker - 3" Diameter - 4 Ohm 3 Watt. Retrieved April 25, 2019, from <https://www.adafruit.com/product/1314>

**Corel Draw**

CorelDRAW Graphics Suite 2019 | Try it Free. (n.d.). Retrieved April 25, 2019, from <https://www.coreldraw.com/en/>

**TEA5767 Stereo Sensor**

Instructables. (2017, October 04). How to Use the TEA5767 FM Radio Module - Arduino Tutorial. Retrieved April 25, 2019, from <https://www.instructables.com/id/How-to-use-the-TEA5767-FM-Radio-module-Arduino-Tut/>

**Raspberry Pi B**

Foundation, R. P. (n.d.). Teach, Learn, and Make with Raspberry Pi. Retrieved April 25, 2019, from <https://www.raspberrypi.org/>

**Power Juice power bank**

Power Banks. (n.d.). Retrieved April 25, 2019, from <https://tzumi.com/collections/power-banks-1>

# Appendix A: Glossary

Tkinter: is a free easy to use Python binding GUI toolkit. It comes standard with Linux, Windows and Mac OS X install of Python.

Python: is a high-level, general-purpose programming language that requires dynamic typing and has standard libraries which is also known as the “included batteries”.

Raspbian: is a Debian-based computer operating system for Raspberry Pi. It uses PIXEL which is the Pi improved X-Window Environment Lightweight as the main desktop environment.

Acrylic: is a transparent thermoplastic often used in sheet form as a lightweight or shatter-resistant alternative to glass. The same material can be used as a casting resin, in inks and coatings

GPIO: is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Pi Zero and Pi Zero W). Prior to the Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header.

GPS: is a Global Positioning System it utilizes satellites and gets the position of the device

Accelerometer: an instrument for measuring acceleration, typically that of an automobile, ship, aircraft, or spacecraft,

LED: is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons. This effect is called electroluminescence

Capacitive: is based on capacitive coupling, that can detect and measure anything that is conductive or has a dielectric different from air.

NMEA Data: is to give equipment users the ability to mix and match hardware and software. NMEA-formatted GPS data also makes life easier for software developers to write software for a wide variety of GPS receivers instead of having to write a custom interface for each GPS receiver.

Circuit Python: is designed to run on microcontroller boards. A microcontroller board is a board with a microcontroller chip that's essentially a small all-in-one computer.

Amplifier: is an electronic device that can increase the power of a signal (a time-varying voltage or current). It is a two-port electronic circuit that uses electric power from a power supply to increase the amplitude of a signal applied to its input terminals, producing a proportionally greater amplitude signal at its output.

PCB: mechanically supports and electrically connects electronic components or electrical components using conductive tracks, pads and other features etched from one or more sheet layers of copper laminated onto and/or between sheet layers of a non-conductive substrate. Components are generally soldered onto the PCB to both electrically connect and mechanically fasten them to it.

FM Stereo: is a method of radio broadcasting using frequency modulation (FM) technology.

Breakout Board: are a common electrical component that take a bundled cable and “breaks out” each conductor to a terminal that can easily accept a hook-up wire for distribution to another device. They are a common item in electronic projects and enable easy, clean installation of electronic devices

Terminal: is a program that emulates a video terminal within some other display architecture. Though typically synonymous with a shell or text terminal, the term *terminal* covers all remote terminals, including graphical interfaces. A terminal emulator inside a graphical user interface is often called a terminal window.

# Appendix B: Code

Code for Amplifier and FM Stereo

The first step in making sure your PI will be able to play music will be its ability to first boot up, then to enable I2C devices, then to detect that the PI can recognize the sensor.

First, enable the I2C bus on your PI.

Go to: /boot/config.txt

add these lines to the bottom of the file:

dtparam=i2c\_arm=on

dtparam=spi=on

dtparam=i2s=on

dtparam=i2c1=on

Next, type:

gpio -v

Make sure you have WiringPi libraries installed If no version is available, please try the following commands:

sudo apt-get install upgrade

sudo apt-get install update

sudo apt-get install i2c-tools

sudo apt-get install wiringpi

sudo apt-get install smbus

Now we need to edit the file

sudo nano /etc/modules

Add these lines here:

i2c-bcm2708

and here: i2c-dev

at the bottom of the file, and remember to write to file before exiting.

Also, we need to enable i2c at boot. So now we run the:

sudo raspi-config

Next, navigate through the first menu

found this at raspberrypi.stackexchange.com//questions/63076/advanced-options-i2c-not-showing.\

After the I2C bus has been enabled, the /modules file edited and saved, any missing libraries installed; open a new file for editing in VIM. Name the file radio.c.

radio.c

#include <wiringPi.h>

#include <wiringPiI2C.h>

#include <stdio.h>

#include <stdlib.h>

int main( int argc, char \*argv[]) {

printf ("RPi - tea5767 Philips FM Tuner v0.3 \n") ;

unsigned char radio[5] = {0};

int fd;

int dID = 0x60; // i2c Channel the device is on

unsigned char frequencyH = 0;

unsigned char frequencyL = 0;

unsigned int frequencyB;

double frequency = strtod(argv[1],NULL);

frequencyB=4\*(frequency\*1000000+225000)/32768; //calculating PLL word

frequencyH=frequencyB>>8;

frequencyL=frequencyB&0XFF;

printf ("Frequency = "); printf("%f",frequency);

printf("\n"); // data to be sent

radio[0]=frequencyH; //FREQUENCY H

radio[1]=frequencyL; //FREQUENCY L

radio[2]=0xB0; //3 byte (0xB0): high side LO injection is on,.

radio[3]=0x10; //4 byte (0x10) : Xtal is 32.768 kHz

radio[4]=0x00; //5 byte0x00)

if((fd=wiringPiI2CSetup(dID))<0){

printf("error opening i2c channel\n\r");

}

write (fd, (unsigned int)radio, 5) ;

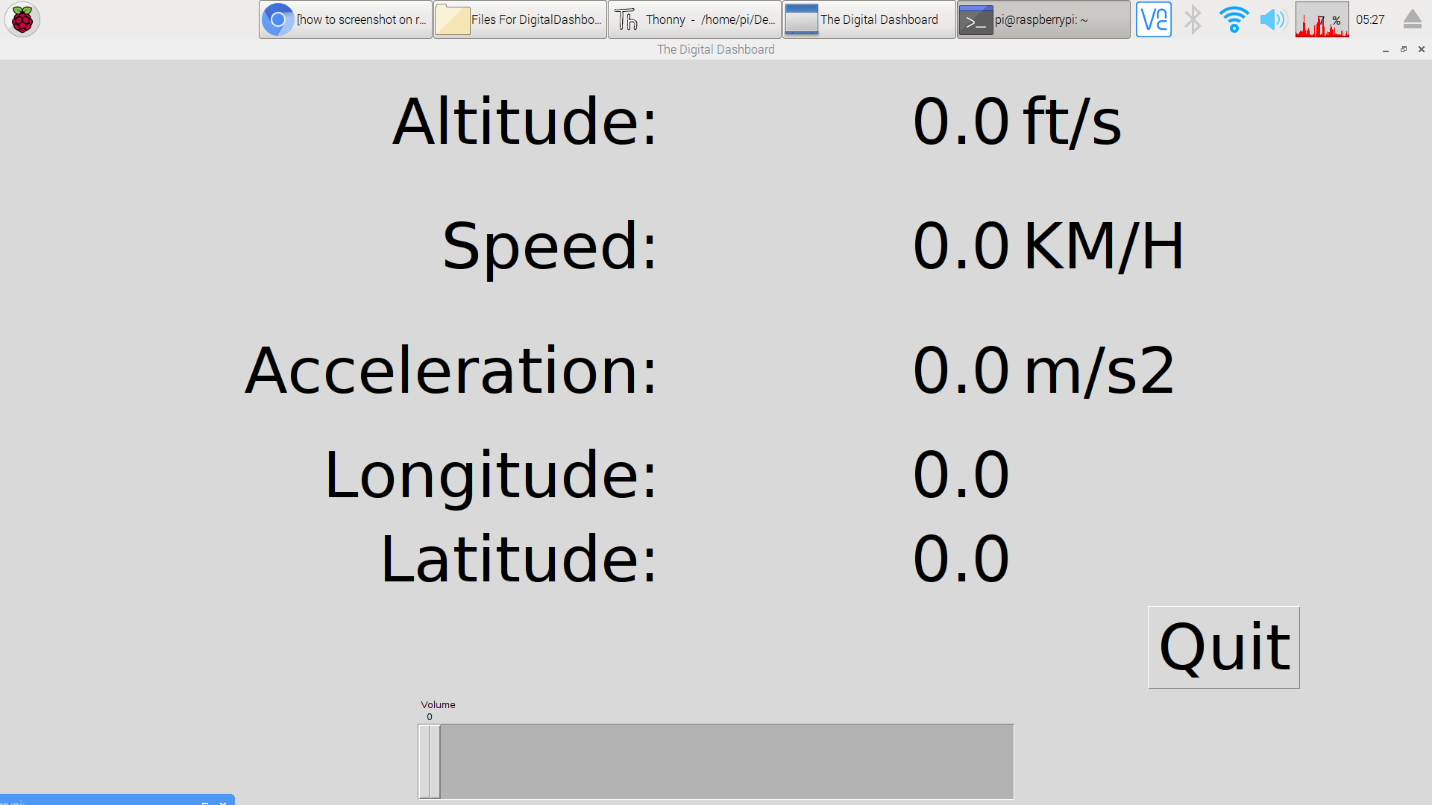
return 0;

}

Once saved as radio.c,

gcc -o radio radio.c -lwiringPi

Dashboard Code



Top of Form

Bottom of Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | |  |
| |  | | --- | |  | |  |  |

import tkinter as tk

import tkinter.font as tkFont

#import test

#import apds9301

###############################################################################

# Parameters and global variables

# Declare global variables

root = None

dfont = None

frame = None

Altitude = None

speed\_1 = None

ms2 = None

Longitude = None

Latitude = None

# Global variable to remember if we are fullscreen or windowed

fullscreen = False

###############################################################################

# Functions

# Toggle fullscreen

def toggle\_fullscreen(event=None):

global root

global fullscreen

# Toggle between fullscreen and windowed modes

fullscreen = not fullscreen

root.attributes('-fullscreen', fullscreen)

resize()

# Return to windowed mode

def end\_fullscreen(event=None):

global root

global fullscreen

# Turn off fullscreen mode

fullscreen = False

root.attributes('-fullscreen', False)

resize()

# Automatically resize font size based on window size

def resize(event=None):

global dfont

global frame

# Resize font based on frame height (minimum size of 12)

# Use negative number for "pixels" instead of "points"

new\_size = -max(12, int((frame.winfo\_height() / 10)))

dfont.configure(size=new\_size)

# Read values from the sensors at regular intervals

def poll():

global root

global Altitude

global Speed

global Acceleration

global Longitude

global Latitude

# Update labels to display sensors information

try:

val = round(test, 2)

altitude.set(val)

val = round(tmp102.read\_temp(), 2)

speed.set(val)

val = round(apds9301.read\_lux(), 1)

acceleration.set(val)

val = round(tmp102.read\_temp(), 2)

longitude.set(val)

val = round(tmp102.read\_temp(), 2)

latitude.set(val)

except:

pass

# Schedule the poll() function for another 500 ms from now

root.after(500, poll)

###############################################################################

# Main script

# Create the main window

root = tk.Tk()

root.title("The Digital Dashboard")

# Create the main container

frame = tk.Frame(root)

# Lay out the main container (expand to fit window)

frame.pack(fill=tk.BOTH, expand=1)

# Variables for holding address,speed,acceleration

Altitude = tk.DoubleVar()

Speed = tk.DoubleVar()

Acceleration = tk.DoubleVar()

Longitude = tk.DoubleVar()

Latitude = tk.DoubleVar()

# Create dynamic font for text

dfont = tkFont.Font(size=-24)

# Create widgets

label\_altitude = tk.Label(frame, text="Altitude:", font=dfont)

label\_altitude\_1 = tk.Label(frame, textvariable=Altitude, font=dfont)

label\_fts = tk.Label(frame, text="ft/s", font=dfont)

label\_current\_speed = tk.Label(frame, text="Speed:", font=dfont)

label\_speed\_1 = tk.Label(frame, textvariable=Speed, font=dfont)

label\_kmph = tk.Label(frame, text="KM/H", font=dfont)

label\_current\_acceleration = tk.Label(frame, text="Acceleration:", font=dfont)

label\_acceleration\_1 = tk.Label(frame, textvariable=Acceleration, font=dfont)

label\_ms2 = tk.Label(frame, text="m/s2", font=dfont)

label\_longitude = tk.Label(frame, text="Longitude:", font=dfont)

label\_longitude\_1 = tk.Label(frame, textvariable=Longitude, font=dfont)

label\_latitude = tk.Label(frame, text="Latitude:", font=dfont)

label\_latitude\_1 = tk.Label(frame, textvariable=Latitude, font=dfont)

button\_quit = tk.Button(frame, text="Quit", font=dfont, command=root.destroy)

scale = tk.Scale(root, orient=tk.VERTICAL)

# Lay out widgets in a grid in the frame

label\_altitude.grid(row=0, column=0, padx=5, pady=5, sticky=tk.E)

label\_altitude\_1.grid(row=0, column=1, padx=5, pady=5, sticky=tk.E)

label\_fts.grid(row=0, column=2, padx=5, pady=5, sticky=tk.W)

label\_current\_speed.grid(row=1, column=0, padx=5, pady=5, sticky=tk.E)

label\_speed\_1.grid(row=1, column=1, padx=5, pady=5, sticky=tk.E)

label\_kmph.grid(row=1, column=2, padx=5, pady=5, sticky=tk.W)

label\_current\_acceleration.grid(row=2, column=0, padx=5, pady=5, sticky=tk.E)

label\_acceleration\_1.grid(row=2, column=1, padx=5, pady=5, sticky=tk.E)

label\_ms2.grid(row=2, column=2, padx=5, pady=5, sticky=tk.W)

label\_longitude.grid(row=3, column=0, padx=5, pady=5, sticky=tk.E)

label\_longitude\_1.grid(row=3, column=1, padx=5, pady=5, sticky=tk.E)

label\_latitude.grid(row=4, column=0, padx=5, pady=5, sticky=tk.E)

label\_latitude\_1.grid(row=4, column=1, padx=5, pady=5, sticky=tk.E)

button\_quit.grid(row=5, column=2, padx=5, pady=5)

scale.pack(padx=500, pady=20)

# Make it so that the grid cells expand out to fill window

for i in range(0, 3):

frame.rowconfigure(i, weight=1)

for i in range(0, 3):

frame.columnconfigure(i, weight=1)

# Bind F11 to toggle fullscreen and ESC to end fullscreen

root.bind('<F11>', toggle\_fullscreen)

root.bind('<Escape>', end\_fullscreen)

# Have the resize() function be called every time the window is resized

root.bind('<Configure>', resize)

# Initialize our sensors

#tmp102.init()

#apds9301.init()

# Schedule the poll() function to be called periodically

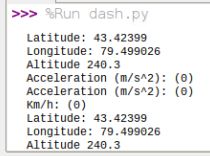
root.after(500, poll)

# Start in fullscreen mode and run

toggle\_fullscreen()

root.mainloop()

Sensor Code



import time

import json

from firebase import firebase

import board

import busio

import serial

import adafruit\_lsm9ds0

import math

i2c = busio.I2C(board.SCL, board.SDA)

sensor = adafruit\_lsm9ds0.LSM9DS0\_I2C(i2c)

while True:

#Runs the linux command that gets the raw NMEA data from the GPS

#Sends the output to the file

os.system('timeout 5 cat /dev/serial0 >> DDData.txt')

#Connects to firebase

myfirebase = firebase.FirebaseApplication('https://digitaldashboard-f7a33.firebaseio.com/')

f= open("DDData.txt","r") #Opens the file to be parsed for data

for line in f.read().split('\n'):

if line.startswith('$GPGGA'):#If the NMEA data line starts with GPGGA then parse the longitude, latitude and altitude

lat, t,lon, = line.strip().split(',')[2:5]

alt = line.strip().split(',')[9]

try :

lat = float(lat)/100

lon = float(lon)/100

alt = float(alt)

print('Latitude:', lat)

print('Longitude:', lon)

print('Altitude', alt)

#Post the data into firebase

postdata = myfirebase.put('Raspberry Pi','Latitude',str(lat))

postdata = myfirebase.put('Raspberry Pi','Longitude',str(lon))

postdata = myfirebase.put('Raspberry Pi','Altitude',str(alt))

except:

pass

elif line.startswith('$GPVTG'): #If the NMEA data line starts with GPVTG then parse the speed in KM/H

knots = line.strip().split(',')[7]

try:

speed = float(knots)

print('Km/h: ({0:0.0f})'.format(speed))

DigitalDashboard.poll(speed)

postdata = myfirebase.put('Raspberry Pi','Speed',str(speed))

except:

pass

else:

#Gets the acceleration from the accelerometer from the x-axis

accel\_x, accel\_y, accel\_z = sensor.acceleration;

accel = math.sqrt((accel\_x\*accel\_x)) #Gets a positive acceleration

print('Acceleration (m/s^2): ({0:0.0f})'.format(accel))

postdata = myfirebase.put('Raspberry Pi','Acceleration',str(accel)) #Post the data into firebase