# vBox: Vehicle Blackbox

*Critical Design Review*

Jose C. Garza

Christian Loth

Prisha Srivastava

Lide Su

Department of Computer Science

Texas A&M University

# Table of Contents

# 1 Introduction

The scope of our project entails the development of a vehicle blackbox that is suitable for everyday drivers and allows for responsible driving through real-time sensor-based systems. vBox provides a modern solution to the growing concerns of speeding and safety risks associated with poor driving habits. Below, we will be discussing the needs, approach and procedure relating to the development of vBox.

## I.I Needs Statement

The problem our design highlights is the need for drivers to recognize their bad driving habits to significantly improve safety. vBox utilizes its mechanisms to provide insight into the consequences their driving could cause. Additionally, the drivers need a way to have access to the data in a way it shows them the decisions they took at certain times

## I.II Goals and Objectives

The goal of this project is to reduce driving-related incidents through automatic, real-time monitoring systems. Incorporating a user-specific interface creates an accurate profile such that it increases self-awareness. Using a time and travel based prediction model, we are able to realistically grasp a driver's current driving behavior and through consistent monitoring, reduce their chances of collisions or accidents. Our objective with this project is to utilize quantifiable data into an interpretative summary that is easily understood by users.

## I.III Design Constraints

vBOx consists of several constraints that are needed for the production of our project such as:

- Technical: We are working with four prominent sensors/ GPIO -- IMU, OBD, GPS and Camera. In order to create a blackbox such that allows extension of our sensors to their required spots outside the box, we must consider certain factors that will support the connectivity with Raspberry PI microcomputers. We can achieve this through spaced out device connectivities and proper soldering. Additionally, we will be relying on a combination of wired and wireless sensor implementation (i.e OBD) to reduce wiring issues.
- Physical: We are designing our prototype with compactness in mind, in order to make our product accessible for various vehicles. Our outside "hull" for our microcomputer will be 3-D printed such that it contains ports for outside connections and a space to fit the raspberry pi inside.

Economical: For our project, we are provided with a $600 budget that will be used towards purchase of required materials and unexpected costs in manufacturing, prototyping and testing. Additionally, our remaining funds will be saved for emergency purposes such as software purchases, deployment and testing costs, if necessary.

Time: The estimated completion date for this Project is April 30,2022. We will achieve this timely development through bi-weekly sprint meetings and burn-up charts to help us identify our weekly tasks and necessary insight on how to maintain a realistic and efficient project timeline while complying with Agile methodology,

### I.IV Validation and Testing Procedure

Testing Procedure for vBox is conducted in four different layers. This layered approach is intended to ensure that the causal sequence of functionality is done so accurately. Below is a breakdown of vBox's testing.

1. Hardware Operations are checked through soldering verifications, and I/O connectivities tests through Raspberry PI. We only conduct hardware verification here to check if the physical components are working properly.
2. I/O System: verification conducted through executable files produced from python scripts. Sensor Manuals are used to verify functionality of sensors.
3. Database Verifications: SQL data retrieval commands are used to check if data exists. Additionally, import history is checked to verify if the most recent datasets are pushed into the database.
4. Web-Platform Testing: Running local testing of web applications using a dummy profile and travel history to check the validity of our service. Running a python script that uses sensor-data to identify driving conditions, and reports a score to the front-end.

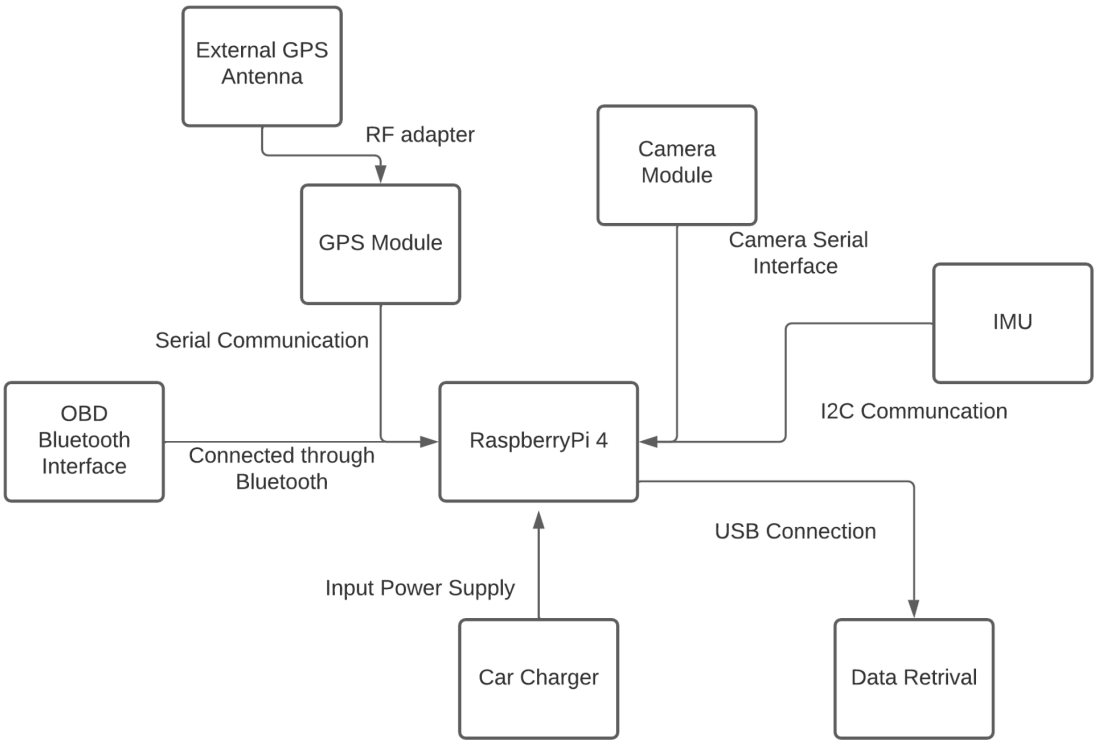
## 2 Proposed design

### 2.1 Updates to the proposal design:

Our design has not changed from the plan originated at the time of the proposal. However, there was an extra device that had been ordered and was a backup. At the time of the proposal a possible challenge was the connection of the bluetooth OBD device to the Raspberry Pi. However, the team achieved a successful connection through bluetooth and the physical OBD interface that could be connected through USB to the Raspberry Pi is no longer required.

## 2.2 System description

The system consists of the following high-level block diagram:




Each of the modules has the following functionalities:

Raspberry Pi: This microcontroller will continuously obtain data from the OBD Interface, IMU, GPS, and the camera module. The data will be stored locally in an SD card until it is transmitted to the web server by the user.

OBD Bluetooth Interface: The OBD port that is present on all cars in the United States will provide data from the ECU to the Raspberry Pi. This port will provide data such as the speed of the vehicle, the rpms of the engine, the throttle position, fuel consumption, among others. These metrics will be important to determine driving behaviors as higher rpms could mean unsafe driving behavior.

IMU (Gyroscope and Accelerometer): The IMU will be connected to the Raspberry Pi and will serve to measure g-forces, orientation and acceleration. This will help determine abrupt change of lanes or high speed corners. In addition, it can be used to determine a hard deceleration or an abrupt acceleration.

The GPS Hat: The GPS hat for the Raspberry Pi will provide us with raw data in the form of NMEA sentences. This will be then parsed to obtain time, date, latitude, longitude, altitude, estimated land speed. The GPS hat will be used in combination with the Google Maps API to map trips and determine if the driver was possibly speeding. In addition, depending on the availability of the information provided by the Google Maps API, it is possible to determine if the driver takes routes that are typically congested.

External GPS Antenna: The external GPS antenna provides a more accurate positioning of the device.

A Camera module: The camera will point towards the outside of the car. Its capability of wide angle view will provide us with a clear view that is as close as possible to the one of the driver. The camera will serve as a constant recording device that will keep track of meaningful situations encountered throughout the drive.

### 2.3 Complete module-wise specifications

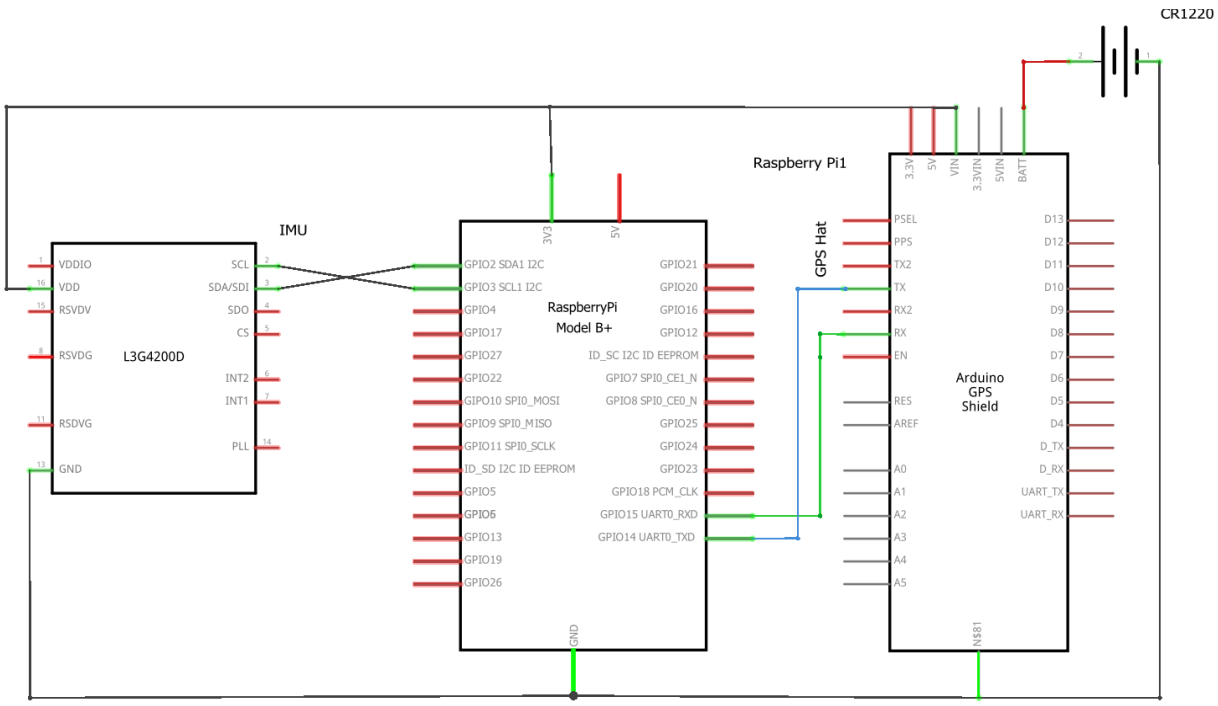
**Parts List and Price:**

The following table contains the parts list and their price

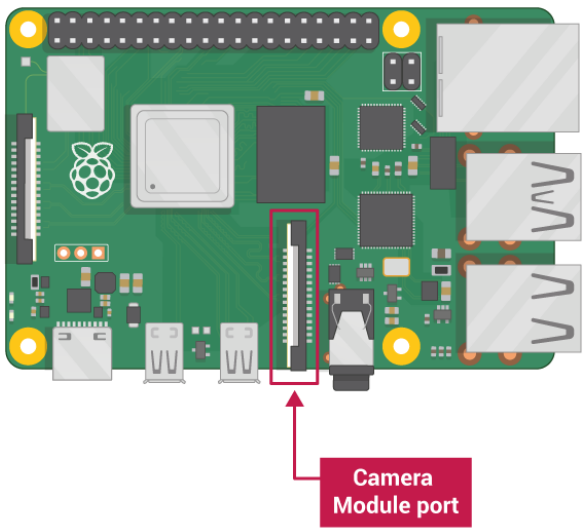
| Description | Price | Quantity |
|---|---|---|
| Raspberry Pi 4 Camera | $24.59 | 1 |
| Adafruit GPS Hat | $29.95 | 1 |
| External GPS antenna | $19.95 | 1 |
| IMU (Accelerometer & Gyro) | $14.95 | 1 |
| Standoffs for Pi HATs | $0.75 | 1 |
| RF Adapter cable | $3.95 | 1 |
| Bluetooth interface OBD port | $13.99 | 1 |
| Raspberry Pi 4 | $167.95 | 1 |
| SD card with 512 GB | $29.99 | 1 |
| Car charger for Raspberry Pi 4 | $14.44 | 1 |
| Suction Cup Mount | $24.99 | 1 |
| 3D Printed Enclosure | $25.00 | 1 |

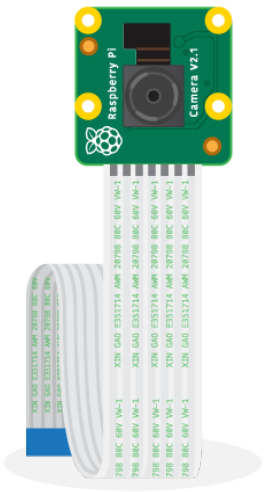
**Circuit and Logic Diagrams:**

The following circuit diagram shows the connections of the IMU and the GPS to the Raspberry Pi, it important to note that that camera connection is not showing in the diagram as it is not connected to the raspberry pi through the GPIO pins, but rather the Raspberry Pi Camera Module port.




The following picture shows the location of the Camera Module port in the Raspberry Pi through where the camera module is connected. The camera module for this project contains a ribbon cable that connects to the port in the microcontroller.

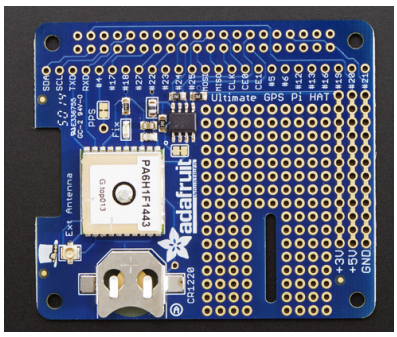


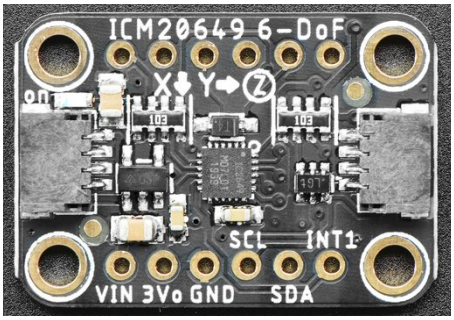
**Interfaces and pin-outs:**

Communication from the modules to the Raspberry Pi occur through the following channels and properties:

1. Adafruit GPS hat:




   a. Power: 3.3 volts supplied the Raspberry Pi & external battery: CR1220
   b. Data communication: Raspberry Pi serial ports:
      i. RXD
      ii. TXD
   c. Additional Accessories:
      i. External Antenna to pin in the GPS hat.
2. External GPS Antenna
   a. Connects physically to the GPS Hat through the external antenna connector
3. Inertial Measurement Unit (IMU):

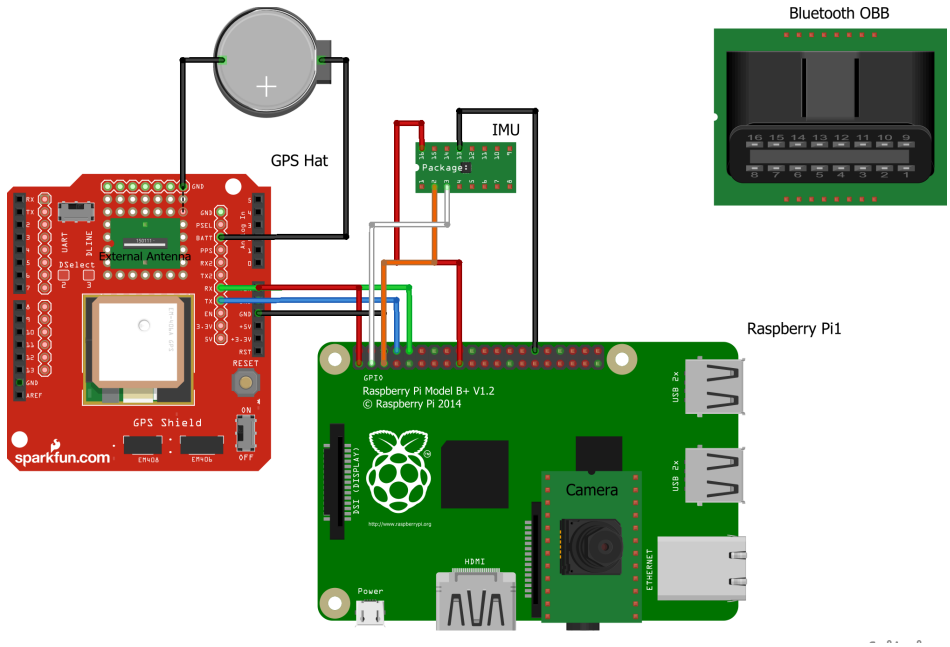



   a. Power: 3.3 volts supplied by Raspberry Pi.
   b. Data communication: Raspberry Pi serial ports:
      i. SCL - I2C clock pin
      ii. SDA - I2C data pin
4. Wide Angle Camera:

   a. Connects and transmits data through the Camera Module Port on the Raspberry Pi.
   b. Contains two LED fill lights that activate depending on the photoresistor for better quality.
   c. No need for external connections or power.
5. Bluetooth OBD

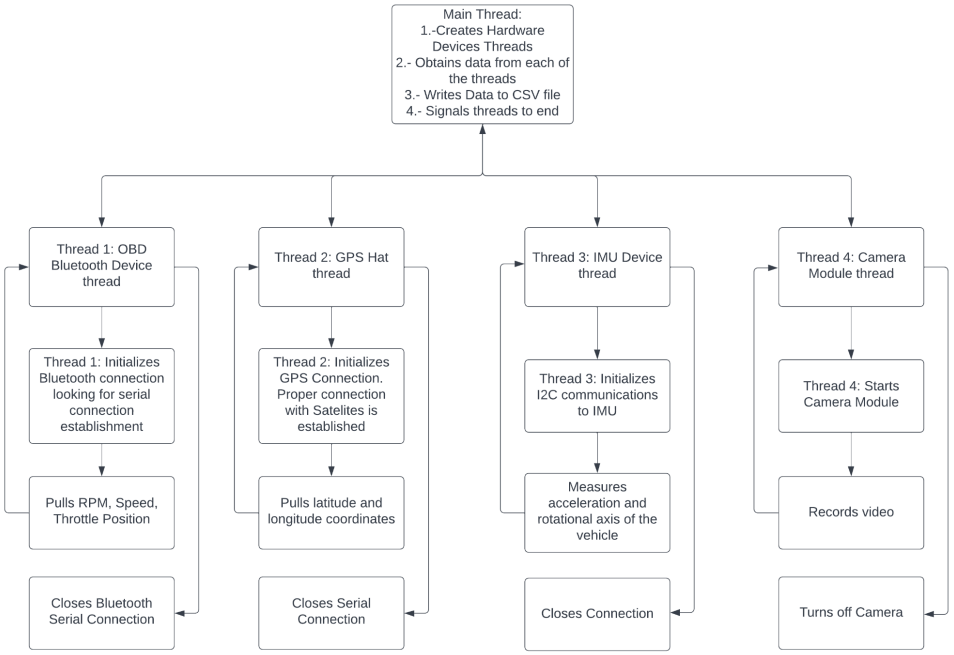

   a. Power is 12 V and is supplied from the battery of the vehicle through the OBD port connection.
   b. Data transmission occurs through the Bluetooth protocol.
   c. The data is received through the second serial port located within the Bluetooth chip in the Raspberry Pi.

The following is a sketch of the interfaces and the connections of the hardware at a high level:

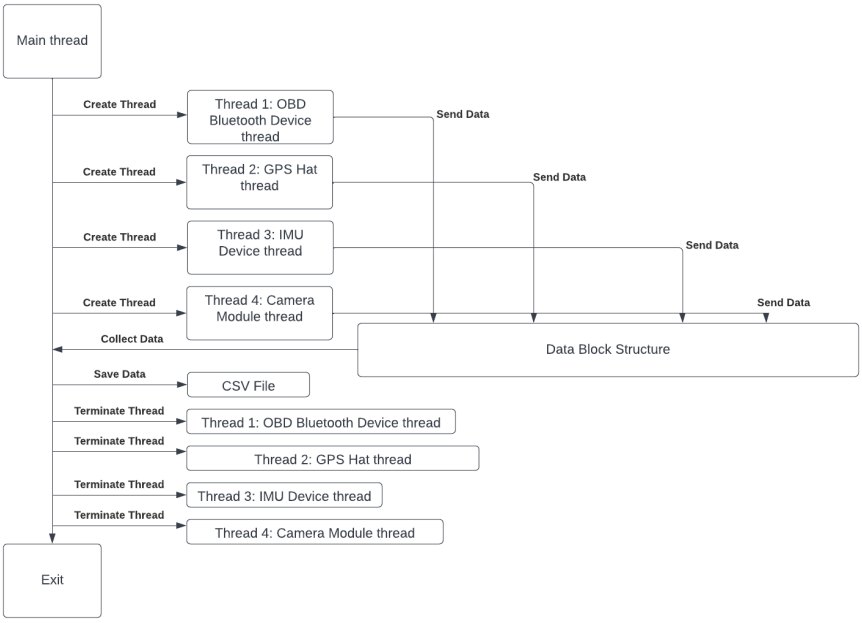



**Embedded Software-Hardware Process Diagram and Timing:**

The following diagram shows the process of data collection from each of the devices. The diagram shows the main process tasks' which include creating a thread for each of the modules or devices, collecting data from them, and saving it to a CSV file.

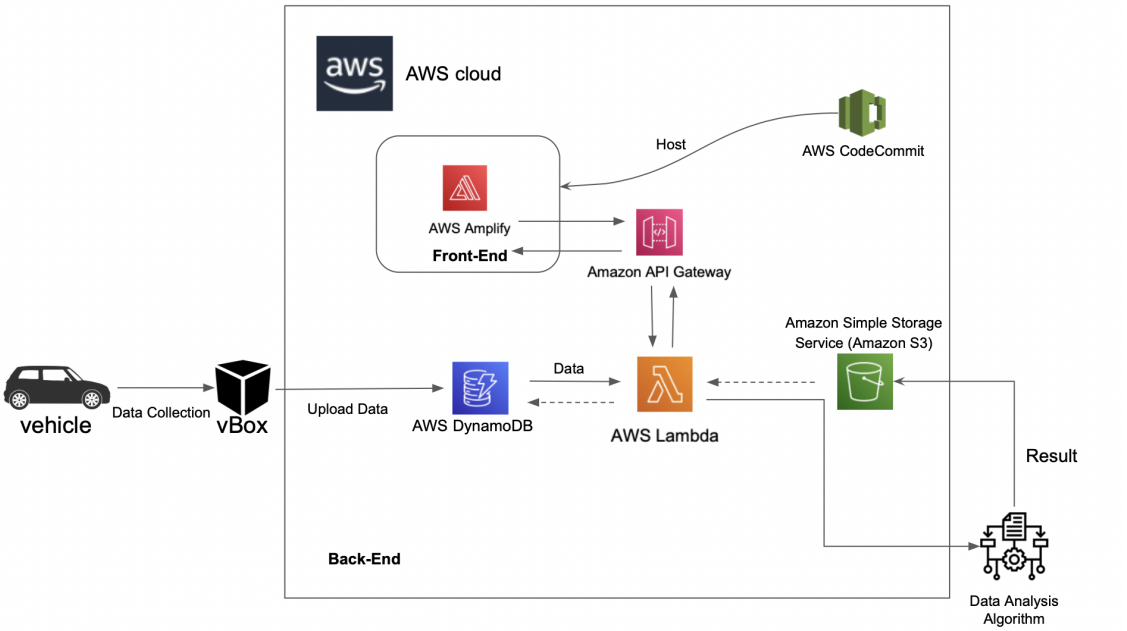


**Timing of the threads:**

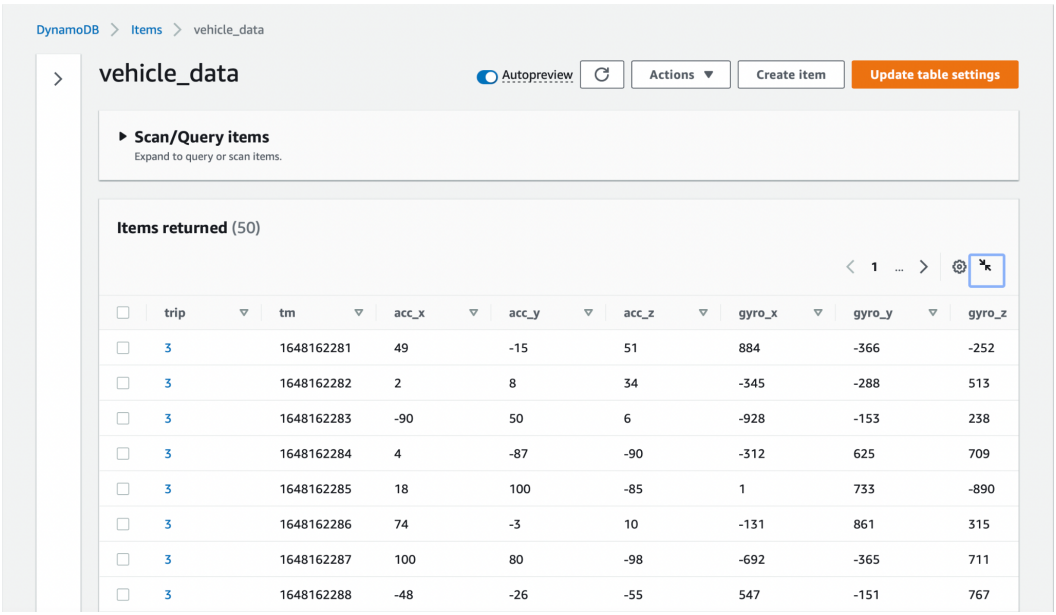



**Software Processes:**

The Software part is composed of back-end and front-end. The back end covers the algorithms required for driver profiling and data management; the front-end is a REACT application that allows the user to navigate through their trips and obtain visualized data about each trip, such as average speed, average engine rpm and throttle position, etc.



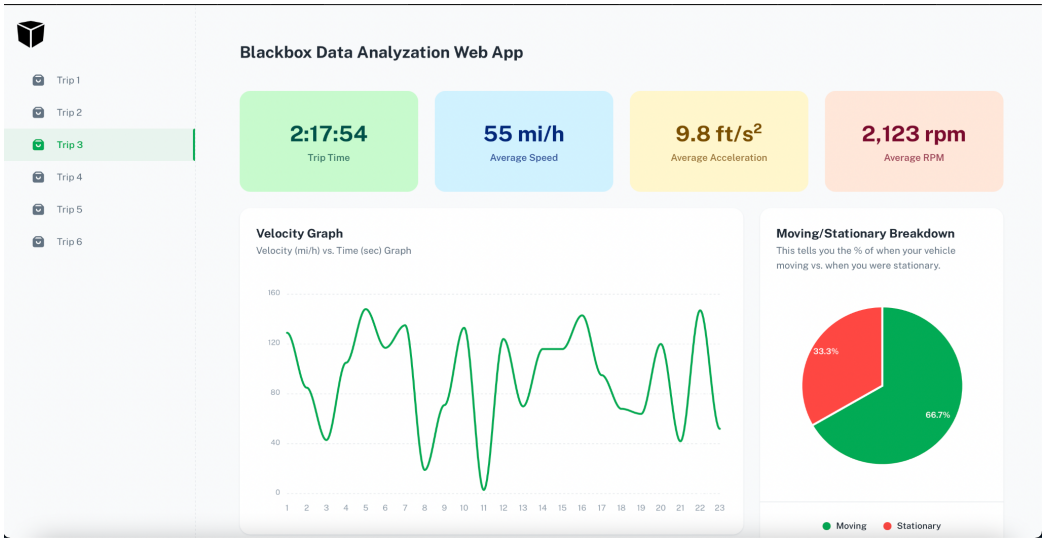
- Back-end: the back-end includes several python scripts that interact with the OBD port of the vehicle and collect the data for further analysis. The collected data is stored locally in the microcomputer and then uploaded to the AWS DynamoDB cloud database via a python script. Since the local storage is practically not able to store the excessive amount of data, it will be cleared after data transfer is complete and set for another round of data collection. So far, we are able to collect and store time, speed, throttle position, acceleration_x, acceleration_y, acceleration_z, gyro_x, gyro_y, and gyro_z. More data such as GPS position data and that obtained from Google Map API will be added later in the next stage. There are also some Lambda functions on the AWS cloud side; their purpose is to interact with the front end via API calls. Each Lambda function corresponds to an AWS API Gateway link. The front end requests for data by calling the API link, then the API Gateway receives the call and invokes the corresponding Lambda function. The Lambda function then extracts data from DynamoDB, processes it, and returns the result as a JSON string, which will be returned by the API Gateway as response to the front end.




| trip | tm | acc_x | acc_y | acc_z | gyro_x | gyro_y | gyro_z |
|---|---|---|---|---|---|---|---|
| 3 | 1648162281 | 49 | -15 | 51 | 884 | -366 | -252 |
| 3 | 1648162282 | 2 | 8 | 34 | -345 | -288 | 513 |
| 3 | 1648162283 | -90 | 50 | 6 | -928 | -153 | 238 |
| 3 | 1648162284 | 4 | -87 | -90 | -312 | 625 | 709 |
| 3 | 1648162285 | 18 | 100 | -85 | 1 | 733 | -890 |
| 3 | 1648162286 | 74 | -3 | 10 | -131 | 861 | 315 |
| 3 | 1648162287 | 100 | 80 | -98 | -692 | -365 | 711 |
| 3 | 1648162288 | -48 | -26 | -55 | 547 | -151 | 767 |

```
Time,RPM,MPH,AX,AY,AZ,GX,GY,GZ
11:53:10 AM,637.5,18.01976457488269,16.470588235294116,0.68,-0.42, 10.25,0.02,0.01,0.01
11:53:11 AM,4208.5,34.17541557305337,31.372549019607842,0.65,-0.46, 10.16,0.01,0.02,0.01
11:53:11 AM,1303.75,0.0,18.431372549019606,0.67,-0.43, 10.17,0.01,0.01,0.01
11:53:12 AM,637.5,25.476218881730695,44.705882352941174,0.66,-0.43, 10.13,0.01,0.02,0.01
11:53:12 AM,1354.5,25.476218881730695,44.705882352941174,0.68,-0.39, 10.23,0.01,0.01,0.01
11:53:13 AM,4062.5,34.17541557305337,18.431372549019606,0.71,-0.36, 10.19,0.02,0.01,0.01
11:53:14 AM,637.5,42.253241072138714,18.431372549019606,0.60,-0.47, 10.17,0.01,0.01,0.01
11:53:14 AM,4062.5,31.689930804104037,26.666666666666668,0.66,-0.45, 10.19,0.02,0.02,0.01
11:53:15 AM,4208.5,0.0,26.666666666666668,0.68,-0.47, 10.25,0.02,0.01,0.01
11:53:15 AM,0.0,8.699196691322676,19.607843137254903,0.36,-0.39, 10.21,0.01,0.01,0.02
11:53:16 AM,1278.75,0.0,18.431372549019606,0.63,-0.42, 10.20,0.01,0.01,0.01
```

- Front-end: the main part of the front end is a REACT web application deployed on AWS Codecommit and hosted on AWS Amplify. It has an authentication interface to protect the data from malicious access. The main page consists of several dashboards that allows the user to select between each trip to see the visualized charts regarding their driving behavior. The web-application calls AWS API Gateway links to obtain data from the cloud database, then presents the results as visualized charts using Javascript.




## 3 Project management

Briefly review the roles and responsibilities of each team member (e.g., team leader, systems design, software design, hardware design, finance and purchases, testing, technical reporting, etc.) Provide any updates on the mechanisms that you are using to manage the project as a team, such as brainstorming sessions, keeping track of progress on every task, etc. Is your management approach working? If not, what types of improvement are you putting in action?

The team consists of four members: Jose C. Garza, Lids Su, Prisha Srivastava and Christian Loth. Each one of the members is a Computer Engineering major and has the qualifications as well as the determination necessary to accomplish this project.

During the project, the team will use the following helper tools to manage the information and progress of the project:

- Github repository: The repository will contain all the documents, code and hardware design material that the team develops throughout the project..

- Google Shared Drive: It will be used as a collaboration tool for documents as it provides a simple way of all members to contribute.
- Slack Channel: It will be our main method of communication for project management decisions such as meeting times, quick questions, suggestions, among others. In this platform the members will communicate daily updates on the tasks they were assigned.

Furthermore, the main oversight of the main components of the project will be divided between the team members in the following way:

Project Team Leader: Jose C. Garza. Throughout the project he has been responsible for turning in any necessary materials before the deadline. He ensures the project is on track for completion by the end of the semester. Additionally, he is the main point of contact for the project in the event of any inquiries by the faculty, ordering parts, etc. During the project, the team leader will keep an updated Gann chart, project budget, and this information will be easily accessible for the other members. Besides management duties, Jose has been responsible for interconnecting all the hardware together with the main microcontroller. Jose has the qualifications necessary to be the team leader as he has experience in previous leadership roles. Additionally, his internship experience consisted of connecting high-level hardware with multiple other modules.

System Designer/Data Analytics: Prisha Srivastava will be responsible for the System Design ensuring the hardware and the software integrate work with each other. Prisha will also work closely with the design and production of the 3D Printed enclosure for the product. Prisha has the analytical skills necessary to understand the system from a higher perspective and identify any issues that could develop due to her internship experience as a Software Engineer. Additionally, her experience in SolidWorks has been essential for the modeling of the enclosure of the Raspberry Pi and its modules. Prisha has also started on tasks regarding the driving behavior analytical algorithm.

Front-End Software Designer/Engineer: Christian Loth will be responsible for working closely with Lide Su to achieve a working web application that displays the information and conclusions gathered from the trip data. Christian has the necessary skills to work in the front-end of the platform as he has developed multiple web applications. He also has experience using API calls and the back-end that will be useful for him to interconnect the front-end with the back-end.

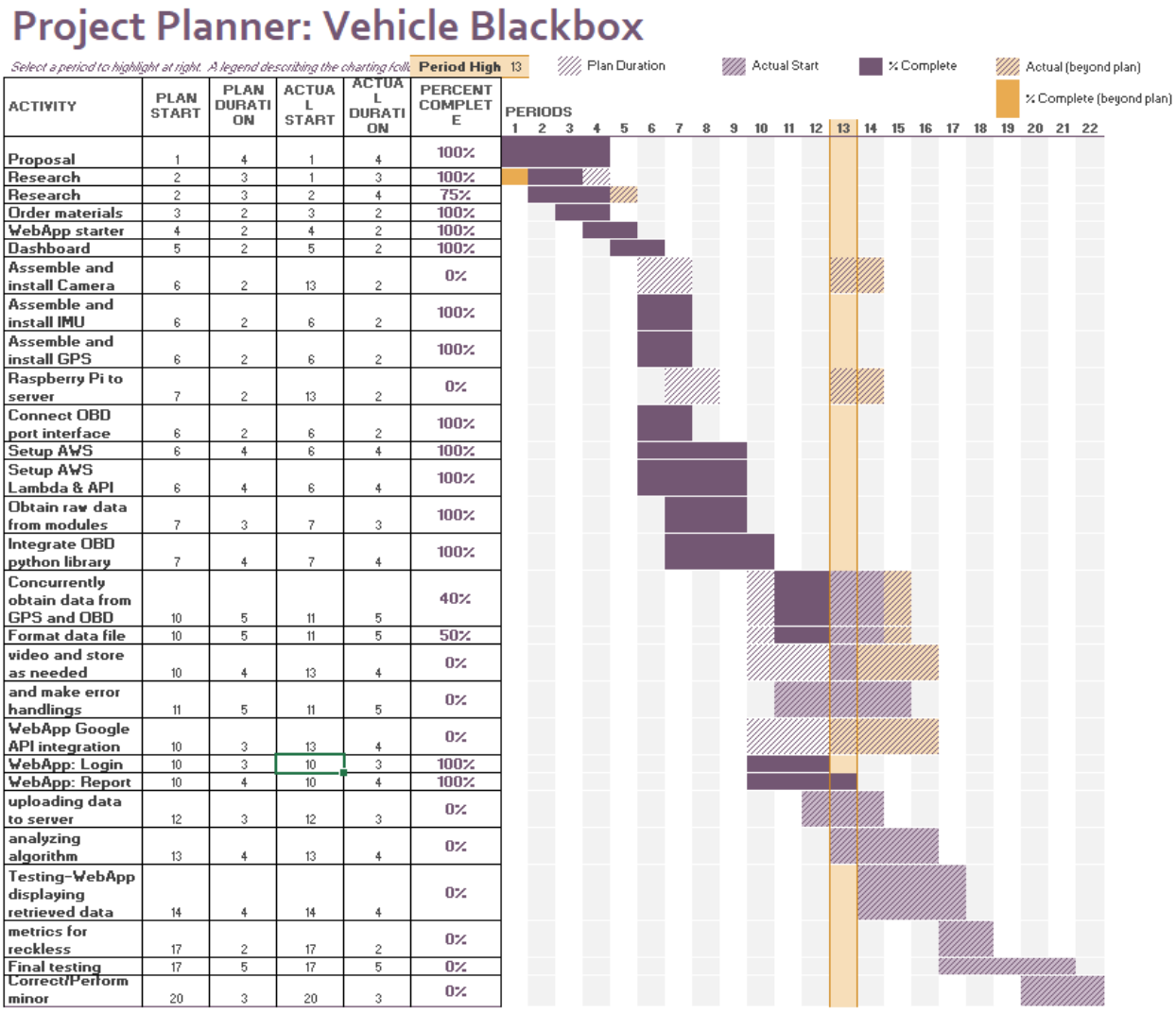
Back-End Software Designer/Engineer: Lide Su has experience using AWS and other Amazon AWS services, therefore he has been responsible for ensuring the back-end software design of the project is in the direction intended to. Some of the tasks he has achieved are: setting up the AWS server, starting

on the creation of lambda functions. His future plans include to ensure the integration with the APIs we choose for the project, as well as working with Christian on how to obtain and analyze the data for the web application

The approach that was taken has successfully moved the team in the right direction and the tasks are being completed on track with the schedule. The major change that has occurred is the addition of tasks to Prisha to start determining the analyzing of data to derive conclusions of the driver.

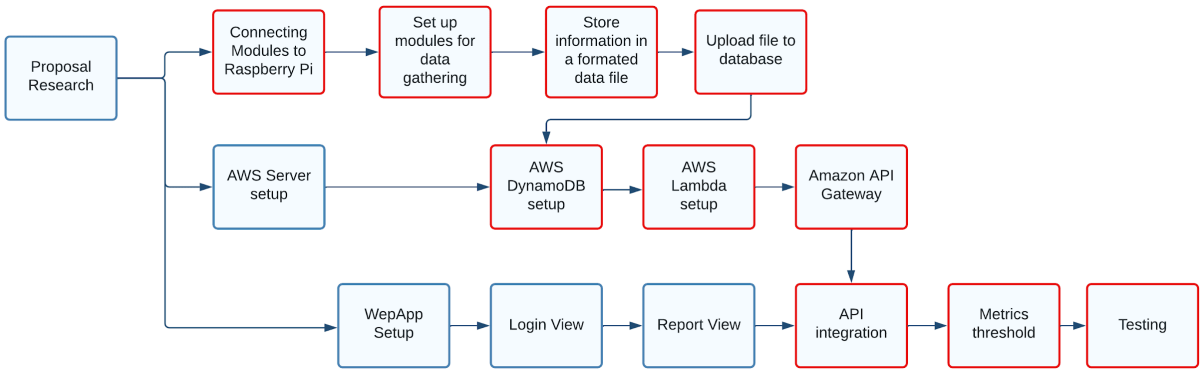
### 3.1 Updated implementation schedule

Our project has been able to stay on schedule with the original plan. There has only been a minor setback due to the late delivery of parts, in specific the Camera module. However, the team tried to work faster in order to catch up to the expected timeline. The updated Gantt Chart is below:




| ACTIVITY | PLAN START | PLAN DURATION | ACTUAL START | ACTUAL DURATION | PERCENT COMPLETE |
|---|---|---|---|---|---|
| Proposal | 1 | 4 | 1 | 4 | 100% |
| Research | 2 | 3 | 1 | 3 | 100% |
| Research | 2 | 3 | 2 | 4 | 75% |
| Order materials | 3 | 2 | 3 | 2 | 100% |
| WebApp starter | 4 | 2 | 4 | 2 | 100% |
| Dashboard | 5 | 2 | 5 | 2 | 100% |
| Assemble and install Camera | 6 | 2 | 13 | 2 | 0% |
| Assemble and install IMU | 6 | 2 | 6 | 2 | 100% |
| Assemble and install GPS | 6 | 2 | 6 | 2 | 100% |
| Raspberry Pi to server | 7 | 2 | 13 | 2 | 0% |
| Connect OBD port interface | 6 | 2 | 6 | 2 | 100% |
| Setup AWS | 6 | 4 | 6 | 4 | 100% |
| Setup AWS Lambda & API | 6 | 4 | 6 | 4 | 100% |
| Obtain raw data from modules | 7 | 3 | 7 | 3 | 100% |
| Integrate OBD python library | 7 | 4 | 7 | 4 | 100% |
| Concurrently obtain data from GPS and OBD | 10 | 5 | 11 | 5 | 40% |
| Format data file | 10 | 5 | 11 | 5 | 50% |
| video and store as needed | 10 | 4 | 13 | 4 | 0% |
| and make error handlings | 11 | 5 | 11 | 5 | 0% |
| WebApp Google API integration | 10 | 3 | 13 | 4 | 0% |
| WebApp: Login | 10 | 3 | 10 | 3 | 100% |
| WebApp: Report | 10 | 4 | 10 | 4 | 100% |
| uploading data to server | 12 | 3 | 12 | 3 | 0% |
| analyzing algorithm | 13 | 4 | 13 | 4 | 0% |
| Testing-WebApp displaying retrieved data | 14 | 4 | 14 | 4 | 0% |
| metrics for reckless | 17 | 2 | 17 | 2 | 0% |
| Final testing | 17 | 5 | 17 | 5 | 0% |
| Correct/Perform minor | 20 | 3 | 20 | 3 | 0% |

Pert Chart:




## 3.2 Updated validation and testing procedures

As mentioned in Section 1, our testing strategies are outlined in several different layers-with each layer including different or all working parts of this product. On a higher level, we test the accuracy of our model by splitting the testing environments--one that includes real test drives, and second, that consists of testing dummy data.

One of the important elements in our product is scripting an algorithm that serves several users, under various driving conditions, to assess their performance. We relied on creating a software program and implementing it in the backend that is connected to the database as well as the web platform. This file pulls data from the database, converts these data points as valid variables, executes the program and outputs the results onto the web platform through API integration. As for the program itself, we sketched out simple switch cases and their respective outputs. For instance, a driver typically drives 15 mph in a 20 mph school zone in Texas. This means the driver is driving 25% under the limit. Now, if the driver is observed to be driving above 20 mph, it is reasonable to assume that the driver is speeding in a school zone, and therefore, acting careless. This observation can be used to identify a “negative” score towards their overall driving score for the trip. There are also certain nuances that this algorithm covers. For example, using the previous scenario, a driver driving below 15 mph would be evaluated less harshly than a driver driving above speed limit. Using a point-based approach that provides a point to a driver depending on the different roads and speed it travels during 1 trip and averaging all points allows to fairly assess a driver and highlight specific instances during the trip that the driver can notice their behavior. However, the average weighted score allows for an assessment that isn’t too sensitive to “slip-ups” that are realistically common in everyday driving. Overall, using a case-by-case approach in crafting our algorithm will allow us to experiment wider with our test cases.

In order to demonstrate that our product works correctly, we will have six different trips that we will complete ourselves. Within each of these trips, we will simulate different driving behaviors to add variance to the data. In each trip, the google API can visualize the trip, whereas the OBD data such as trip time, average speed, average acceleration, average RPM, velocity, moving/stationary breakdown, braking, throttle position, and wheel angle, is displayed.

### 3.3 Updated division of labor and responsibilities

The personal experiences of each team member has been used throughout the project. Since there is major development in both hardware and software, the team splited into two in order to achieve as much progress as possible while using their main skills. Both branches have constantly developed effective progress throughout the project. The team is approaching a point in which the whole team will come together and be responsible for integrating all the different parts together and ensuring proper intercommunication throughout the whole system. After the integration of both branches is completed, the team as a whole will continue working in the development and testing of the platform,

Lide and Christian have worked together in the development of the AWS server and the WebApp. Together they have achieved the setup of the AWS Server and the design of the WebApp. Some of the major accomplishments have been:

1. Established AWS DynamoDB.
2. Implemented Lambda functions and API gateway.
3. Established connection from React app to AWS Amplify to host web app online.
4. Made significant progress with designing the dashboard that the users see of the WebApp.
5. Added different graphs and different attributes to the webpage.

On the other hand, Prisha and Jose have worked together in the hardware side of the project, some of the major accomplishments of the hardware is the following:

1. Receiving the parts ordered and soldering the necessary parts.
2. Establishing a connection between each one of the components.
3. Integrating the module connections together for a more compact device.
4. Establish basic python scripts to test the connection between the devices.
5. Establishing the CSV file format

The following are tasks that the team is expected to accomplish: and their respective dates:

| Task # | Task Description | Responsibility of: | Start Date | End Date |
|---|---|---|---|---|
| 1 | Concurrently obtain data from GPS and OBD | Jose C. Garza | 03/28/2022 | 04/01/2022 |
| 2 | Format data file | Jose C. Garza | 04/01/2022 | 04/05/2022 |
| 3 | Record Camera video and store as needed | Prisha Srivastava | 03/28/2022 | 04/01/2022 |
| 4 | Retrieve data and make error handlings | Prisha Srivastava | 04/10/2022 | 04/15/2022 |
| 5 | WebApp Google API integration | Christian Loth | 03/28/2022 | 04/01/2022 |
| 6 | Design Improvements WebApp: Login View | Chrsitian Loth | 04/01/2022 | 04/05/2022 |
| 7 | Design Improvements WebApp: Report View | Christian Loth | 04/05/2022 | 04/10/2022 |
| 8 | Implement uploading data to server | Lide Su | 04/01/2022 | 04/05/2022 |
| 9 | Develop analyzing algorithm | Prisha Srivastava | 03/28/2022 | 04/15/2022 |
| 10 | Testing-WebApp displaying retrieved data | Lide Su | 04/05/2022 | 04/10/202 |
| 11 | Determine metrics for reckless behavior | Prisha Srivastava | 03/28/2022 | 04/18/2022 |
| 12 | Final testing | All | 04/20/2022 | 04/26/2022 |
| 13 | Correct/Perform minor improvements | All | 04/20/2022 | 04/26/2022 |

The team is confident in the completion of the project in the necessary timeframe. Now that the retrieval of information has been achieved, the probability of something that could stop the project is minimal. The team focus is now in the integration and analysis of the data to present the users with meaningful information.

## 4 Preliminary results

Provide any test results and demo of completed parts of the system at the time of the CDR.

Hardware Testing:

The modules we received required soldering. Therefore after each one was soldered, a continuity test was done for each connection in order to ensure a proper connection.

Following, each of the modules was connected individually to the Raspberry Pi and the required packages were installed to communicate with them. The following are the test performed in order to test the Raspberry Pi communication with each of components:

OBD Bluetooth interface:

A python script was written using the OBD library. The following snippet is of the csv file into which data is written from the OBD port. Keep in mind that the data shown was performed with a stationary car and it is the reason why MPH is at "0.0". The data shown by the OBD port matched the one in the dashboard of the vehicle.

```
Time,RPM,MPH,Throttle,Load,Fuel Status
0:28:36.811282,648,0.0,11.7647058824,28.6274509804,0202
0:28:37.115022,649,0.0,11.7647058824,28.6274509804,0202
0:28:37.375240,651,0.0,11.7647058824,28.6274509804,0202
0:28:37.642446,652,0.0,11.7647058824,28.6274509804,0202
0:28:37.881327,650,0.0,11.7647058824,28.6274509804,0202
0:28:38.148746,650,0.0,11.7647058824,28.6274509804,0202
0:28:38.449984,646,0.0,11.7647058824,28.6274509804,0202
0:28:38.688869,649,0.0,11.7647058824,28.6274509804,0202
0:28:38.930141,649,0.0,11.7647058824,28.6274509804,0202
0:28:39.180201,657,0.0,11.7647058824,28.6274509804,0202
0:28:39.401354,656,0.0,11.7647058824,28.6274509804,0202
```

Inertial Measurement Unit:

Following the OBD, we received the IMU. After soldering the IMU, we tested the communication through the IC2 communication protocol of the Raspberry Pi and wrote a simple python script to achieve data gathering. A snippet of the data is shown below and demonstrates a proper connection with the Raspberry Pi:

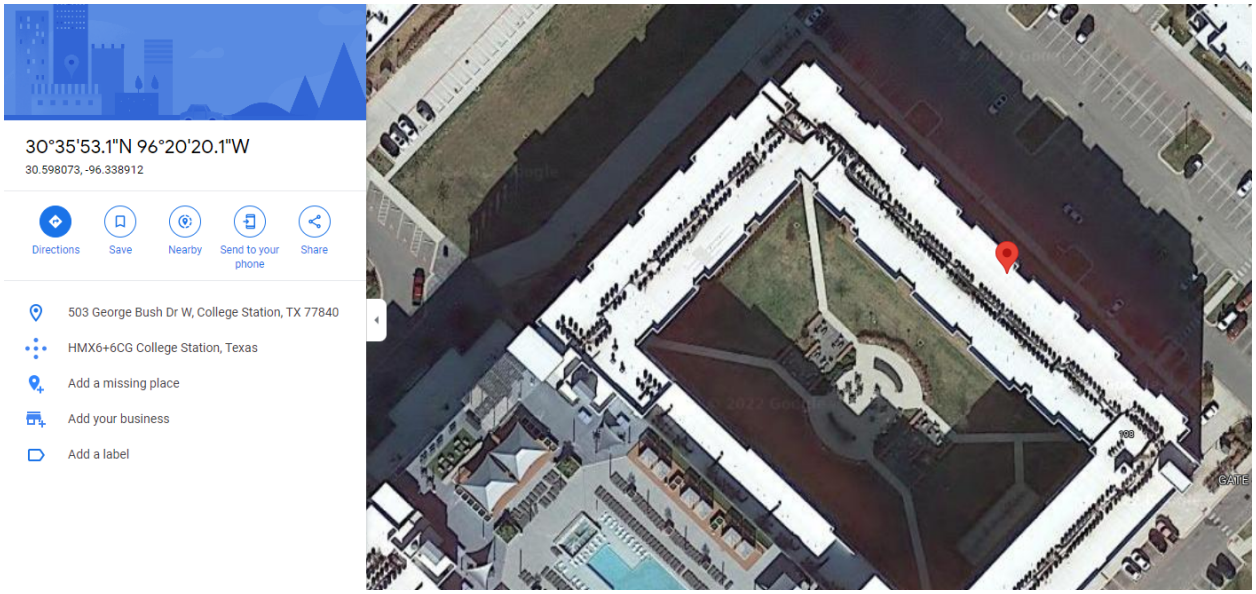
```
Acceleration: X:0.68, Y: -0.42, Z: 10.25 m/s^2
Gyro X:0.02, Y: 0.01, Z: 0.01 rads/s

Acceleration: X:0.71, Y: -0.46, Z: 10.16 m/s^2
Gyro X:0.02, Y: 0.01, Z: 0.01 rads/s

Acceleration: X:0.60, Y: -0.43, Z: 10.17 m/s^2
Gyro X:0.01, Y: 0.01, Z: 0.01 rads/s

Acceleration: X:0.66, Y: -0.43, Z: 10.13 m/s^2
Gyro X:0.02, Y: 0.02, Z: 0.01 rads/s

Acceleration: X:0.68, Y: -0.39, Z: 10.23 m/s^2
Gyro X:0.02, Y: 0.01, Z: 0.01 rads/s

Acceleration: X:0.66, Y: -0.36, Z: 10.19 m/s^2
Gyro X:0.01, Y: 0.01, Z: 0.02 rads/s

Acceleration: X:0.63, Y: -0.47, Z: 10.17 m/s^2
Gyro X:0.01, Y: 0.01, Z: 0.01 rads/s
```

GPS Hat

The GPS communicates through the serial port to the Raspberry Pi which requires enabling certain ports of the Raspberry Pi. The GPS hat by itself requires some time to obtain reliable data from the satellites if not present in the open space. However, after adding the external antenna, we were able to obtain a reliable measurement. The following shows the process of obtaining data and verifying the accurateness of it:

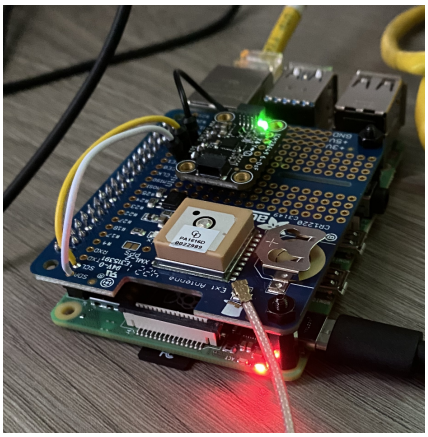
```
raspberrypi:/dev/serial0 9600 NMEA0183>
Time: 2022-03-29T03:24:14.000Z  Lat:  30 35' 52.09920" Non:  96 20' 20.03820" W

 GNGGA GPGSA GLGSA GNRMC GNVTG GPGSV GLGSV
                    Sentences

Ch PRN  Az El S/N   Time:      032414.000      Time:      032415.000
 0  29 229 82  16   Latitude:     3035.8832 N  Latitude:  3035.8835
 1  15 142 63   0   Longitude:   09620.3397 W  Longitude: 09620.3396
 2  13  81 45  25   Speed:     0.71           Altitude:  159.3
 3  18 318 41  22   Course:    250.42         Quality:   1   Sats: 08
 4   5  40 38  23   Status:    A       FAA: A HDOP:      1.12
 5  23 254 33  22   MagVar:                   Geoid:     -23.3
 6   2  98 18  22                RMC                     GGA
 7  20  54 16   0
 8  25 212 13   0   Mode: A3 Sats: 13 18 5 23 2  UTC:          RMS:
 9  26 295  9   0   DOP: H=1.12  V=0.94  P=1.46  MAJ:          MIN:
10  10 242  5  17   TOFF:  0.300015623           ORI:          LAT:
11  24 159  1   0   PPS:                        LON:          ALT:
        GSV                  GSA + PPS                  GST
(64) $GPGSV,3,3,12,25,13,212,13,26,09,295,,10,05,242,,24,01,159,*75
(68) $GLGSV,2,1,06,70,84,217,,85,49,351,22,84,46,097,20,69,30,030,16*62
(42) $GLGSV,2,2,06,71,30,209,19,86,09,323,*60
```

Putting those coordinates into google maps display the current location of the device:




The location is accurate within a 100 ft radius, which is pretty accurate while under a closed roof condition.

After a proper testing of each of the parts and accurateness of the measurements taken. The GPS hat and IMU were united by the breadboard provided in the GPS hat in order to have a more compact unit and less wires into the following:



Proper connections were tested by following the same procedure as indicated above and proper responses were obtained similarly to the ones previously displayed.

Software Testing:

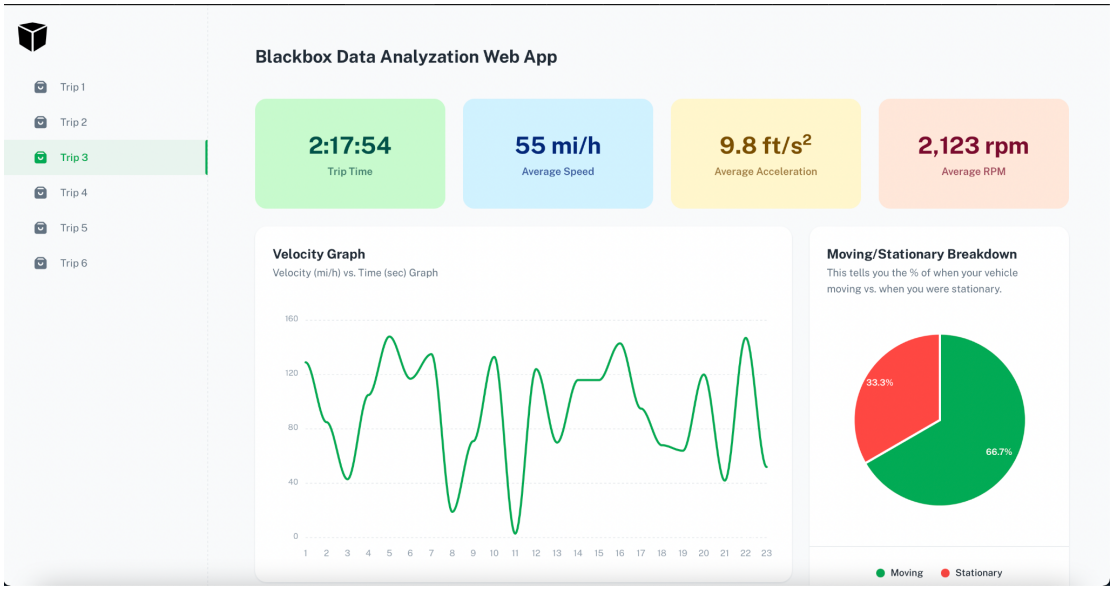
- For the AWS API Gateway call, the response was successful with a return value of **200** and the requested content.


```
{'statusCode': 200, 'headers': {'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Headers': 'Access-Control-
Allow-Origin'}, 'body': {'average_speed': '86.375', 'average_acceleration': '29.673286070976367', 'average_rpm': '4984.75', 'trip_time': '15.0', 'time':
'[1648162265.0, 1648162266.0, 1648162267.0, 1648162268.0, 1648162269.0, 1648162270.0, 1648162271.0, 1648162272.0, 1648162273.0, 1648162274.0, 1648162275.0,
1648162276.0, 1648162277.0, 1648162278.0, 1648162279.0, 1648162280.0]', 'speed': '[70.0, 34.0, 44.0, 117.0, 56.0, 27.0, 119.0, 66.0, 42.0, 128.0, 128.0,
139.0, 71.0, 48.0, 150.0, 143.0]', 'throttle_position': '[68.0, 68.0, 92.0, 67.0, 69.0, 0.0, 66.0, 16.0, 94.0, 87.0, 24.0, 21.0, 62.0, 49.0, 49.0, 66.0]'}}

<Response [200]>
```


- The API response is then parsed using Javascript and embedded into the web application. The velocity graph application requests for timestamp and velocity data for trip 1, and presents it as a line chart.


| acc_y | acc_z | gyro_x | gyro_y | gyro_z | rpm | speed | throttle_... |
|---|---|---|---|---|---|---|---|
| 24 | 87 | 782 | -10 | 966 | 7500 | 129 | 87 |
| -35 | 78 | -171 | -252 | -185 | 6000 | 85 | 98 |
| -15 | -84 | 417 | -222 | -463 | 4500 | 43 | 10 |
| 31 | 57 | -369 | -419 | 387 | 5000 | 105 | 59 |
| 32 | 51 | -514 | 203 | 302 | 6679 | 148 | 18 |
| 56 | -36 | 628 | 951 | -41 | 6440 | 117 | 6 |
| -64 | 41 | -995 | 295 | 809 | 2820 | 135 | 67 |
| -25 | 20 | 330 | 370 | -862 | 1841 | 19 | 28 |
| 65 | -9 | 29 | 969 | 130 | 2728 | 71 | 93 |
| -80 | -14 | 966 | 662 | -558 | 1266 | 133 | 19 |
| 53 | -62 | -732 | 974 | 740 | 8295 | 3 | 62 |
| -59 | -39 | -220 | 989 | -772 | 2917 | 124 | 27 |
| 59 | -30 | 901 | -378 | 990 | 8852 | 70 | 12 |

- The front-end is a REACT web application presenting vehicle data and analysis about the driver.


Blackbox Data Analyzation Web App
Trip 1
Trip 2
Trip 3
Trip 4
Trip 5
Trip 6
2:17:54
Trip Time
55 mi/h
Average Speed
9.8 ft/s²
Average Acceleration
2,123 rpm
Average RPM
Velocity Graph
Velocity (mi/h) vs. Time (sec) Graph
160
120
80
40
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Moving/Stationary Breakdown
This tells you the % of when your vehicle moving vs. when you were stationary.
33.3%
66.7%
Moving
Stationary