Fleet Monitoring System

Sdmay18-18 <u>http://sdmay18-18.sd.ece.iastate.edu/</u> Client/Advisor: Lotfi Ben-Othmane Group Members: Venecia Alvarez, Kendall Berner, Matthew Fuhrmann, William Fuhrmann, Anthony Guss, Tyler Hartsock

Problem Statement

Problem:

- Companies have many vehicles
- Vehicle fleets are inefficient and costly

Solution:

- Allow a fleet manager to see real time data of vehicles
- Allow a fleet manager to see periodic reports about each vehicle and each driver

Functional Requirements

The product shall:

- Gather data from a vehicle's ODB-II port
- Transmit data from the vehicle to the server
- Process raw data from the vehicle on the server
- Record vehicle data into a database
- Display a map with a location of all vehicles in the fleet
- Display historical data for a certain vehicle

Non-Functional Requirements

The product shall:

- Be used by vehicles at any time and location
- Utilize Google Cloud services
- Only allow managers to view fleet data on the dashboard
- Have the server side code made in Node.js
- Use AngularJS on the client side

Market Survey

Other Fleet Management Applications:

- Mobile App vs OBD II
- Live tracking, statistics, vehicle data
- Live map of fleet
- Our Application vs Rest of Market:
 - More useful interpretations of internal data
 - Automate tedious tasks

Basic Design

- 3 Components
 - Microcontroller to retrieve vehicle data
 - Server with database to receive, store, and relay
 - Website to display data to manager

Risk Management

- Limited knowledge of embedded systems → Lots of time spent researching, working with other students with embedded systems experience
- Unable to use original hardware \rightarrow Use a Raspberry Pi
- Difficulty testing with just one device \rightarrow Making fake test data
- Race conditions with the map \rightarrow Research common solutions and redesign

Project Budget

- Original hardware: Android Board, development kit
 - o **~\$500**
- New hardware: Raspberry Pi, GPS, and connector
 - \circ \$50 + \$30 + \$10 = \$90
- Google Cloud services
 - \$40/month * 5 months = \$200

Project Schedule

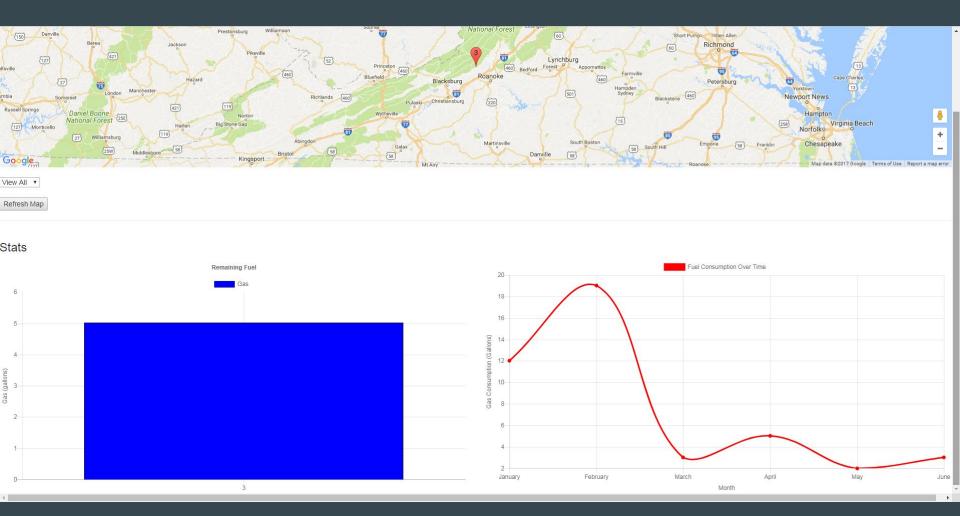
Tasks	10/2-10-8	10/9-10/15	10/16-10/22	10/23-10/29	10/30-11/5	11/6-11/12	11/13-11/19	11/20-11/26	1/22-1/28	1/29-2/4	2/5-2/11	2/12-2/18	2/19-2/25	2/26-3/4
Learn how to put code on hardware														
Learn how to program on the CAN BUS network														
Create the CAN BUS querying code														
> Create generic CAN BUS query libraries														
> Implement queries for specific vehicle components														
> Test accuracy of measured data														
Create the data dissemination code														
> Configure device to use mobile data														
> Create code that sends data in server API format														
Create mockups														
> Determine which data we want displayed														
> Determine which statistics we want to calculate														
Integrate Google Maps to plot locations														
Create graphs and charts using Chart.js														
Learn how to call API to gather data														
Display data in a user friendly manner														
Create API for handling data from the embedded device														
Create data models for incoming data														8
Analyze data and send important information to the database														
Create API for sending data to the front end														
Perform operations on the raw data														1
Implement queries for new needed vehicle components														
Test accuracy of measured data														
Update code that sends data in server API format														
Figure out the packing and installation of the Pi into the vehicle														
Send raw CAN data to server from Pi														3
Check for supported PIDs of vehicle														J
Fix startup/close down of python application to prevent bugs														
Implement register and login functionality														
Implement editing a manager's fleet														
Improve look and feel of website														
Determine and implement more useful/relevant charts														
Implement more AngularJS features														
Convert current api into Swagger api														
Implement server side code for register and login functionality														
Create api functionality for editing fleets														
Update api to take in raw can data														
Create database model for valid PIDs for vehicles														
Implement data processing for new features for 2.0 release														
Analyze current db and data models and make improvements														

Detailed Design - Front End

User Interfaces: Emphasis on visualizing data

Technologies Used: AngularJS, Chart.js, Google Maps API, Bootstrap

Test Plan: UwAmp/XAmpp for local testing



Detailed Design - Server

Technologies Used: NodeJS, MongoDB, Mongoose

Test Plan: Automated API Testing Using Postman, Unit Testing for Data Analysis

Hosted on Google Cloud Compute Engine

Swagger Documentation

Transitioning into a Swagger based API

Detailed Design - Microcontroller

Three Modules: gps_interface, can_interface, server_send

Technologies Used: Python, PiCAN2, Adafruit Ultimate GPS, gpsd, PyCan, requests

Test Plan: unittest.py, verify with simulator, compare with car actuals

Current Project Status

Prototypes for Python application, server, and front-end completed with interoperability.

Group Contributions

Venecia: Client side, emphasis on Google Maps API and AngularJS

Kendall: Client side, emphasis on Chart.js

Matthew: Android application for OBD-II, GPS, sending data. Attempted to fix Android hardware.

William: Android, Python, and ODB-II

Anthony:

Tyler:

Plan for Next Semester

Expand functionality for server and front-end.

Move to real vehicle use for Raspberry Pi.

Increase PID support for Python application.