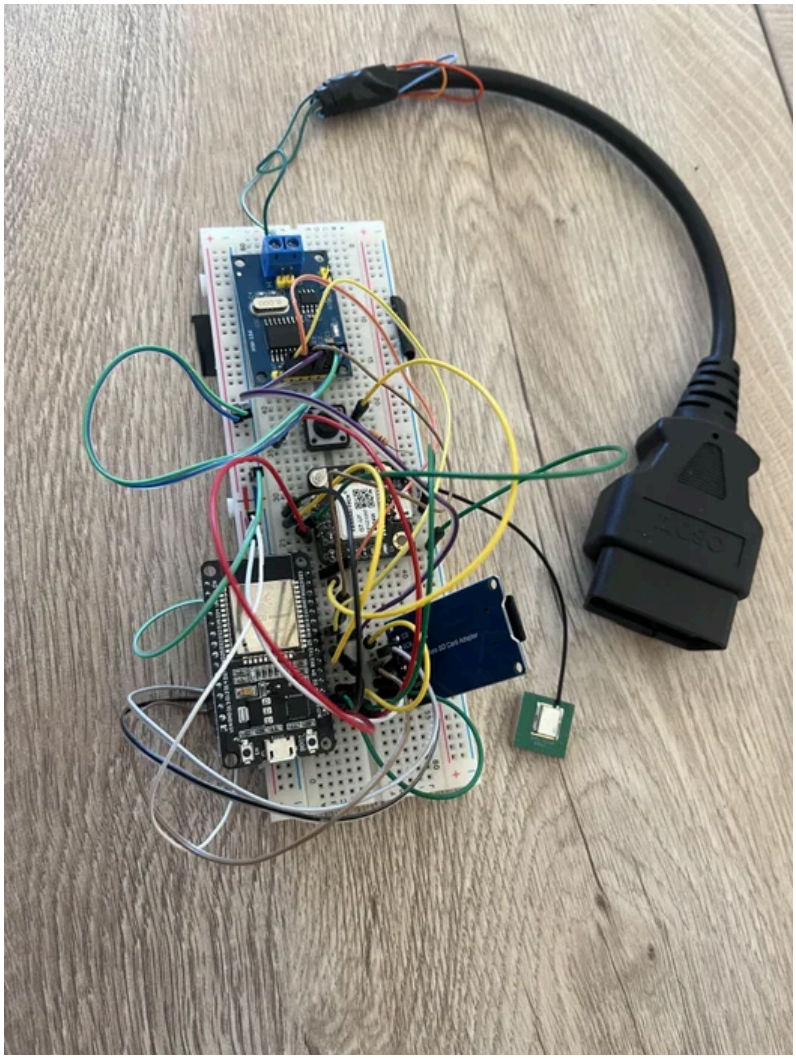
AUTODESK Instructables

ESP32 Vehicle Telematics - OBD2 (CAN) and GPS Reporting System

By bkennar2 in CircuitsMicrocontrollers

Introduction: ESP32 Vehicle Telematics - OBD2 (CAN) and GPS Reporting System



The code for this project can be found at https://github.com/bkennar2/vehicle-telematics

The purpose of this project is to build a device that can track vehicles and read their OBD2 data. Companies like Samsara and Verizon sell devices and monitoring services like this for fleet vehilces. All newer vehicles (except some EVs) have an OBD2 port. The OBD2 port allows you to get data directly from the vehicle using CAN. To learn more about OBD2, read here: https://www.csselectronics.com/pages/obd2-explained-simple-intro

The devices by Samsara and Verizon are internet connected, requiring a monthly cellular plan. My device records trip data on an SD card while you are away from your home. Once, the device returns home, it attempts to connect to your home network and upload the stored data to a local server.

The data handling is performed by an API built on with Python Flask and hosted on a raspberry pi. That data can then be retreived by a webapp that connects to the same API to get trip data. It uses the Google Maps API to create a visualization showing the trip.

HOW IT WORKS:

The ESP32 is connected to the vehicle via an OBD2 breakout cable. Currently, a USB is used to power the setup. Every second, the system reads the GPS data, sends CAN messages to the vehicle, and then read the responses. Then that data is stored in a file called carData.txt. When the user pushes the button on the breadboard or the code detects that the current GPS coordinates are within 30 meters of your preprogrammed home coordinates, it will try to connect to your home WiFi network to upload the data to a Raspberry Pi server. Note: if you start at home, it won't try to upload until you've gone outside of 30 meters.

When the system turns on, a new session is recorded as an "&" in the carData.txt file. When the ESP32 preps the data to upload to the API, this allows it to identify separate the sessions. The session's unique identifiers are VIN and the first timestamp of the session. The data is stored in a csv file on the raspberry pi in the location carData/[VIN]/[start_date]. The ESP32 has a limited amount of memory. So, we break the session into chunks for upload. When the chunk is the first of a session, we add a row in the PostgreSQL database with only the VIN and start date and create the csv file. When it isn't the first chunk, we just append to the previously created csv.

We have a python script that runs once per day that finds all the rows in the database that have no end date. It then processes the csv file to fill out the rest columns of the metadata in the DB. The program gets the session's end date, idle time (seconds spent at 0 km/h), max speed, and average fuel efficiency for the trip. This is where you could add your insights to process and track.

The API allows you to retrieve data as well. I built a very simple Flask app that gets this data and show the trip on a map using the google maps API.

Project Improvements:

1) The circuit could be ran off the 12V from the OBD2 port of the vehicle. You would need a DC-DC converter do drop the 12V down to 5V. However, many vehicle always have the 12V on OBD2 on. You would want a way to put the ESP32 into sleep mode to not drain you vehicle's battery. Some OBD2 ports have an "ignition" pin. Once that goes up to 12V you could come out of sleep.

2) Instead of using a raspberry pi, the api and storage could be ran from the cloud. You would need to add security to you api.

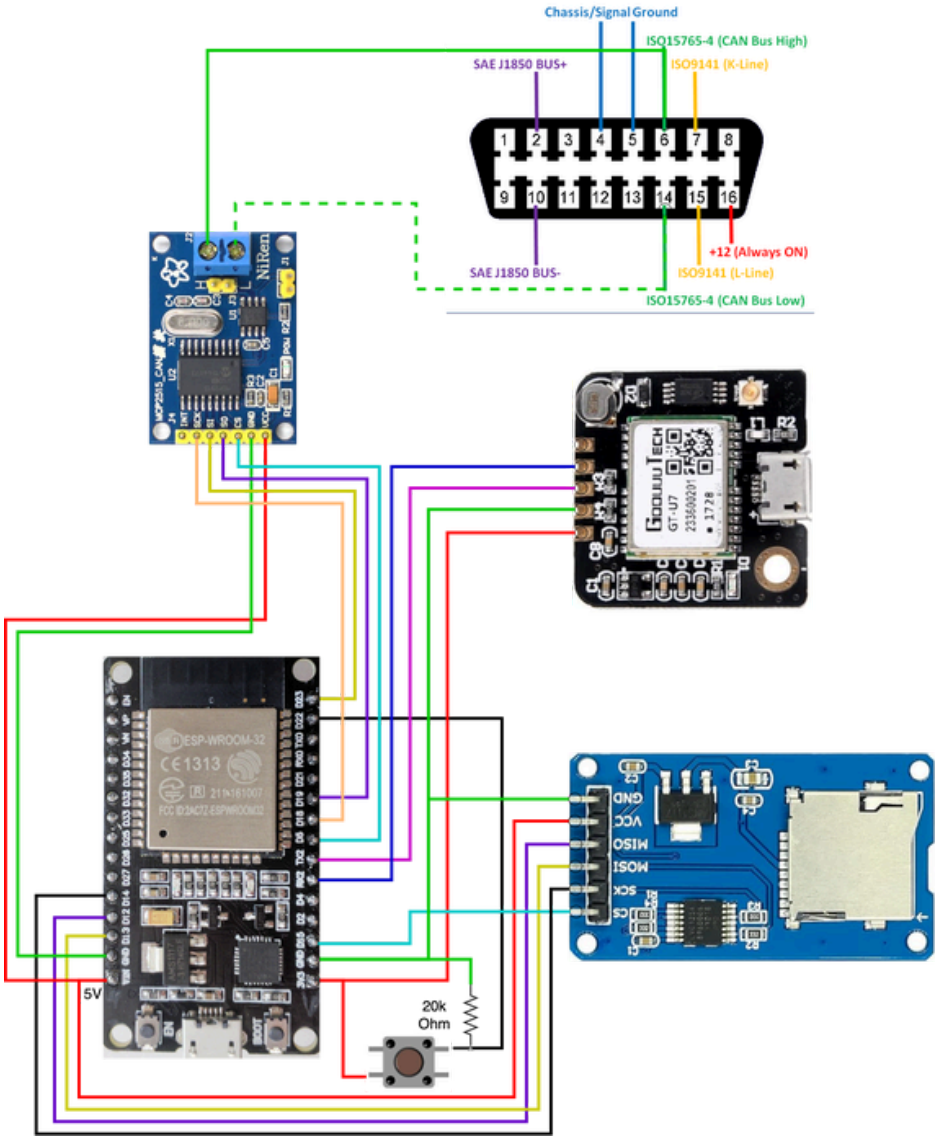
3) The parts from the each of the modules could be intergated into a single board that plugs directly into the OBD2 port. This would make the entire project much cleaner.

4) You could add a section in the ESP32 code that retrieves VIN from the vehicle in the setup.

Supplies

1. ESP32 DevKitV1 board: https://www.amazon.com/Development-Microcontroller-Integrated-Antenna-Amplifiers/dp/B09GK74F7N
2. CAN module: https://www.amazon.com/HiLetgo-MCP2515-TJA1050-Receiver-Arduino/dp/B01D0WSEWU
3. GPS module: https://www.amazon.com/Navigation-Positioning-Microcontroller-Compatible-Sensitivity/dp/B084MK8BS2
4. SD module: https://www.amazon.com/HiLetgo-Adater-Interface-Conversion-Arduino/dp/B07BJ2P6X6
5. micro SD card
6. OBD2 breakout cable: here
7. 20k Ohm resistor
8. Button
9. Breadboard
10. Raspberry Pi

Step 1: Wire the ESP32 and Modules



Wire the ESP32 and modules as shown in the diagram. When powering the ESP32. Make sure to use USB or 5V because some of the modules require 5V input.

Step 2: Set Up the ESP32 Code.

1. install the arduino IDE
2. Under board manager install the ESP32 by Esspressif Systems
3. Under Library manger install
4. TinyGPSPlusPlus by Ress
5. autowp-mcp2515 by autowp
6. Open and save the .ino file from the ESP32 folder from my github repo
7. Change values
8. SSID and password to your WiFi creditials
9. VIN or leave as 12345
10. Server ip address. You could use raspberrypi.local or the static ip address of the pi.
11. Line 27 and 28 add the coordinates of your home or where ever you will park you vehicle
12. Once the ESP32 detects that it's within 30 meters of this, it'll try to connect to the WiFi
13. Line 90 if your CAN module has a 16Mhz crystal you'll need to change the 8 to 16.

Step 3: Set Up Raspberry Pi

Setup your raspberry pi. Here's a good video that shows a headless setup. This means you won't need a monitor, keyboard, or mouse.

Step 4: Set Up Your PostgreSQL Database

You can follow the instuctions here with the following adjustments:

In the section title "Creating your first PostgreSQL Database on the Raspberry Pi"

We create a database named "cardata" with a table named tripData with the following columns. You can add or remove columns based on what data you want to track. This will just be the metadata for your trips. It gives you a summary or the data for that trip.

1. CREATE DATABASE cardata;
2. create table tripData (VIN varchar(17), startTime timestamp, endTime timestamp, idleTime integer, maxSpeed smallint, fuelEff float(1));

Step 5: Set Up API on the Pi

1. go to your home direction "cd ~"
2. type "mkdir Projects" to create a projects folder
3. type "cd Projects" to change directory to the projects folder
4. copy the carDataAPI folder from my github to this location
5. create a python virtual environment for the project
6. python -m env /home/pi/Projects/carDataAPI
7. Go into the carDataAPI folder "cd carDataAPI"
8. Activate the virtual environment "source env/bin/activate"
9. you should see (env) appear next to the command line
10. Install the necessary packages using the requirements file.
11. "pip install -r requirements.txt"
12. Change the username and password on lines 14 and 15
13. Make sure the program runs by using python app.py
14. run "chmod 777 start_flask.sh app.py"
15. This changes the permissions on the files so they can be run from the crontab
16. Run the prgram with "./start_flask.sh" to make sure it works.

Step 6: Set Up Pi Automatically Start the API

We want the API to start whenever the Pi turns on. So, we need to add a line to the crontab. The crontab allows you to run programs automatically. Follow the steps below to get set the API up on the crontab:

1. Run "crontab -e" to edit the crontab
2. Add the below line to the crontab. This line means that on reboot it will run start_flask.sh and send the outputs to flask.log
3. @reboot /home/pi/Projects/carDataAPI/start_flask.sh >> /home/pi/Projects/carDataAPI/flask.log 2>&1
4. Reboot the pi and confirm the Flask app starts up. (sudo reboot now)
5. To see what python files are running you can run "sudo ps -aux | grep python"

Step 7: Test the Setup

1. Connect your ESP32 to your laptop with a USB
2. On the arduino IDE you should see GPS data on the serial monitor
3. Let it run for a 30s and then disconnect the USB
4. Take the SD card out of the module and put it in your laptop.
5. Check to make sure data is being saved.
6. You will only see datetime, longitute, latitude, and altitude.
7. The rest will just be commas because you aren't connected to the vehicle
8. Put the SD card back in the module
9. Connect the ESP32 to the laptop
10. Let the prorgram run for another 30s
11. Press the button on the breadboard
12. The ESP32 should attempt to upload data to the rasperry pi server through the API
13. If you are using the serial monitor, you will see "success"
14. Check that the data is on the raspberry pi
15. ssh into the raspberry pi
16. go to the carDataAPI directory
17. Run "python readData.py" to see if your metadata is there
18. There should also be a folder with your VIN.
19. Go to that folder and check that the complete data is there

Step 8: Test the Process Data File

1. Test the processData.py file by running python processData.py it is also located in the carDataAPI folder
2. If successful, when you run the readData.py file, you will see an endDate in the metaData.
3. Add the a job in the crontab to kick off the run_process.sh file that will kick off the processData.py file.
4. Run "chmod 777 processData.py run_process.sh" to change permissions so the crontab can run the files
5. update crontab "crontab -e"
6. Add: 40 23 * * * /home/pi/Projects/carDataAPI/run_process.sh >> /home/pi/Projects/carDataAPI/process.log 2>&1
7. this file runs at 11:40pm everyday and sends its output to process.log

Step 9: Set Up the Visualization

To run the vizualization, I just set up a Flask App on my computer using either powershell (Windows) or terminal (mac). The steps are very similar to the setup on a raspberry pi. This assumes you already have python on your computer.

1. type "mkdir Projects" to create a projects folder
2. type "cd Projects" to change directory to the projects folder
3. copy the mapApp folder from my github to the Projects folder on your laptop
4. create a python virtual environment for the project
5. python -m env /mapApp
6. Go into the mapApp folder "cd mapApp"
7. Activate the virtual environment "source env/bin/activate"
8. you should see (env) appear next to the command line
9. Install the necessary packages
10. pip install -r requirements.txt
11. Go to https://developers.google.com/maps to get your api-key and map id
12. there are several good tutorial on youtube to complete this step
13. Add your api-key and map id to the file index.html on lines 6 and 104, respectively.
14. You can also change the starting center point of the map on line 102.
15. run the app with "python app.py"
16. Access the app by going to localhost:5000 on your browser