CPE 4750: Introduction to IoT Systems, Spring 2025

Final Project

Vehicle Emergency Alert

Team Members: Bryce Owensby, Julia Johnson, Salini Ambadapudi

Electrical and Computer Engineering

Kennesaw State University

Faculty: Dr. Jeffrey L Yiin

Date of Submission: May 4, 2025

Contents

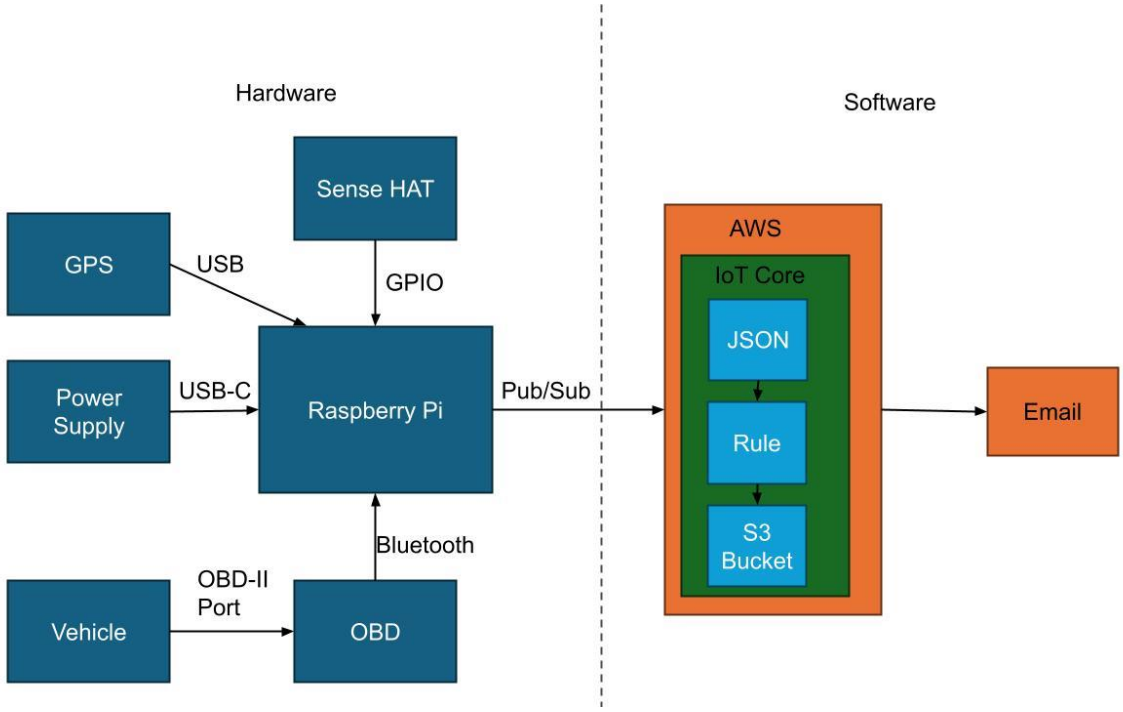
Project Motivation

As a team, we realized that the transportation industry is extremely essential to the economy as a whole. This integral industry is also one of the most dangerous to be a part of, with one of the highest casualty and fatality rates in the country. We decided to contribute to this industry by creating a device that helps prevent accidents and increase emergency response time.

Project Goal

This documentation will provide instructions for creating a cloud-based vehicle alert system. It will utilize a GPS, Bluetooth OBD adapter, and Amazon AWS to provide an alert notification for anytime your vehicle detects an issue or the alarm button is pressed.

Architecture



Hardware & Software

Hardware:

- Raspberry Pi
- Sense HAT
- Power Supply
- USB GPS
- OBD Bluetooth Adapter

- Car

Software:

- Amazon AWS
- obd (Python library)
- pyserial (Python library)
- gpsd (Python library)
- Bluetooth

Instructions/Implementation

Setting up the USB GPS

1. Once you start your Raspberry Pi you will need to install the packages required for interacting with the GPS. In your Raspberry Pi's terminal enter the following command:
   Sudo apt-get install gpsd gpsd-tools gpsd-clients
2. Find device name while the USB GPS is plugged into the Raspberry Pi
   ls /dev/tty*
3. The code above will display any device name that contains /dev/tty. You are looking for /dev/ttyAMC0. Once you confirm that is included in the list, stop the gpsd program.
   sudo systemctl stop gpsd
4. Then start gpsd again but let it receive data from the ACM0 device
   sudo gpsd /dev/ttyACM0 -F /var/run/gpsd.sock
5. You should now be able to display GPS data with either:
   cgps
   Or
   gpsmon
6. If you are starting the GPS for the first time, you will need to allow the device to sit either outside or near a window so that it can receive its initial data. The process may take about 5 minutes.
   a. If no data is received after some time, restart the Pi while the USB GPS remains plugged in.
7. If you want to display the code through a command, you can create a .py file and use the following code:

*NOTE – When creating your .py file, do not name it 'gps.py' or 'run_gps.py'

```
import gps

# Connect to the GPS daemon
```

```
session = gps.gps(mode=gps.WATCH_ENABLE)

while True:
    # Wait for a new GPS data
    report = session.next()

    if report['class'] == 'TPV':
        if hasattr(report, 'lat') and hasattr(report, 'lon'):
            print(f"Latitude: {report.lat}, Longitude: {report.lon}")
        if hasattr(report, 'alt'):
            print(f"Altitude: {report.alt} meters")
```

Setting up the Bluetooth OBD-II adapter connection

1. You will need to update your raspberry pi to ensure the most recent libraries are installed

    sudo apt-get update

2. Installing the packages requires having python-pip installed

    sudo apt-get -y install python-pip

3. To collect and interpret data from the obd, you will now need to install the python obd library.

    sudo pip install obd

4. The last library you will need to install is pyserial. This is what enables the bluetooth obd-II adapter to communicate with python.

    sudo pip install pyserial

5. Upload the code from this github link to your Raspberry Pi. We encourage uploading all files, including the folder, to /home/pi/.

*NOTE - The following code is based on this github and tutorial, but has been updated to fit the needs of this project.

Link to code: https://github.com/JuliaCJ/Vehicle-Emergency-Alert-System/tree/main

6. This script is designed to start at device boot. You will need to set up your Raspberry Pi to auto-login to the user that will be running the script. Use this command to open the

gett@tty1.service file.

sudo nano /etc/systemd/system/getty.target.wants/getty@tty1.service

7. Find the "ExecStart" line in the file and replace it with the following line. This will automatically login the Raspberry Pi to the "pi" user.

ExecStart=-/sbin/agetty -a pi %I $TERM

8. Now, you will set up the daemon script. This will start running the code at device boot. This command will create the daemon script.

sudo nano /lib/systemd/system/vehicle_alert.service

9. Add the following lines to the script. /path/to/scripts/ should be replaced with the actual path to the script. If the code was properly uploaded in step 5, the path should be /home/pi/vehicle_alert.py

*NOTE – The version of python you are using may slightly change these lines. If running the service produces an error related to a missing OBD package, change the word "python" in the "ExecStart" line to "python3."

```
[Unit]
Description=obdPi Service (Print)
After=multi-user.target
[Service]
Type=idle
ExecStart=/usr/bin/python /path/to/scripts/vehicle_alert.py
[Install]
WantedBy=multi-user.target
```

10. After the daemon script is created, it will need to be enabled to run on boot. Input the following commands in your terminal.

sudo chmod 644 /lib/systemd/system/vehicle_alert.service

sudo systemctl daemon-reload

sudo systemctl enable vehicle_alert.service

11. Now you will need to establish the Bluetooth connection with the OBD-II adapter. Insert the following commands to configure your Bluetooth.

sudo bluetoothctl

agent on

default-agent

pairable on

scan on

12. The “scan on” command should show a list of MAC addresses for all available Bluetooth devices. One of the addresses should have a description of “OBD-II” next to it. This is the MAC address you will be referring to in the next steps.

Once you identify the address, input the following commands. For the next steps, replace AA:BB:CC:11:22:33 with your adapter’s MAC address.

*NOTE – If you do not see “OBD-II” next to any of the listed devices, verify that you Raspberry Pi is close enough to the vehicle and that adapter is on. If you still do not see the description, you may need to first connect the adapter to your phone or laptop to see the MAC address.

scan off

pair AA:BB:CC:11:22:33

13. At this point, you may be prompted to enter a pin to connect to the adapter. The pin should be printed on the slip or the box that came with the adapter. Most adapters have a pin of “1234” or “0000.”

16. Once the pin is entered and the device is paired, you will need to connect to it.

connect AA:BB:CC:11:22:33

quit

15. Connection confirmation messages should now be displayed on your terminal. Your Bluetooth adapter is now ready to be used!

If the connection fails, please see the troubleshooting note from step 12.

16. Once the connection is established, it will need to be bound to a serial port to be used with python scripts. You can configure the connection with the following commands. This will only need to be done once.

sudo rfcomm release all

sudo rfcomm bind 0 AA:BB:CC:11:22:33

17. To ensure your vehicle_alert.service script functions properly at boot, the Bluetooth connection should be established at boot too. Open /etc/rc.local with the following command.

sudo nano /etc/rc.local

18. Add the following lines before the end of the file.

sudo rfcomm release all

sudo rfcomm bind 0 AA:BB:CC:11:22:33

19. To update your device with your changes and finish enabling your daemon script, reboot the Raspberry Pi.

sudo reboot

20. Once the device is rebooted, verify that the service is running.

sudo systemctl status vehicle_alert.service

21. You can stop the service at any time with the following command.

sudo systemctl stop vehicle.service

22. The following command will disable the service, which prevents it from running at device boot.

sudo systemctl disable vehicle.service

23. Your service should now be running and functional. If any “FAIL” messages are sent from the service, verify your Bluetooth connection is set up properly. You can manually start the vehicle_alert.py code if the service is interrupted from too many failed connection attempts.

*NOTE – As with step 9, if an error occurs with the OBD package, replace “python” with “python3”

python vehicle_alert.py

24. Data from the OBD-II adapter should be displaying on your terminal. It is updated every 3 seconds and the last 5 minutes of data are stored in a JSON file.

Setting Up AWS Alerts

1. Navigate to the IoT Core service
2. In the new page, select **Connect one device**. Follow the directions provided and ensure you download the provided zip file and have set the SDK language to Python. The zip file will need to be transferred to the Pi for unpacking.
3. Now that your 'thing' is created, you can select it and look into its **Certificates** tab and verify the one listed is set as active. Also verify that the unzipped file that was downloaded to your Pi contained the following files:
   - <ThingName>.public.key
   - <ThingName>.private.key
   - <ThingName>.cert.pem
   - <ThingName>-cert.pem.crt
   - Root-CA.crt
4. You will need to update the policy attached to your certification. In your certification, at the bottom of the page, select your policy.
   - **Edit active version**
   - Change policy action and Policy resource to **'*'**
   - Check the box for **Set the edited version as the active version for this policy**
   - **Save as new version**
5. Back in your Raspberry Pi, enter the following command:

   Pip3 install AWSIoTPythonSDK

6. In the 'vehicle_alert.py' script, find the upload_to_aws() function and replace the default pathways to your pathways for the files from Step 3.

Setting up the email alerts

1. Navigate to the **Amazon SNS console** and then select **Topics** on the right-side column.
2. Create a Topic and set it to be a **Standard** topic type. Name the topic and then create it.
3. Once created, select the topic and click **Create subscription** at the bottom of the page.
4. Set the **Protocol** to **Email** and the **Endpoint** as the intended email address that will be sent the alerts.
5. Navigate to **AWS IoT>Message Routing>Rules** and create a new rule for the alert.
6. After naming the rule the SQL Statement will resemble this template:
   SELECT * FROM 'vehicle/data' WHERE NOT (Alert = ['No Auto-Alerts'])

7. Set the **Action** to be **Simple Notification Service (SNS)**
8. Select the created topic
9. Set message format as **RAW**
10. Create and name a new role for the SNS Alert.
11. Review and create the rule. You should now be able to test the alert code to see if an alert is triggered

Schematics

The raspberry Pi is connected to a power bank (red USB-C cord), A GPS (black USB), and the vehicle's OBD (blue Bluetooth adapter plug-in)

The Bluetooth adapter for the OBD connection while connected to the vehicle

GitHub Link

https://github.com/JuliaCJ/Vehicle-Emergency-Alert-System/tree/main

Lessons Learned

- How to retrieve OBD data from a vehicle and display it in the command line.
- How to retrieve location data using a connected GPS.
- How to use AWS to set up a thing, rule, role, and bucket to receive information from a raspberry pi and respond when set conditions are met.
- Vehicles have various limitations on their OBD data based on make and model. We had difficulties getting the fuel level to properly register. The speed initially output as being in km/h, so we changed it to mph.

Conclusion

The transportation industry is one of the most important, as it fuels national and international economies. Our device, equipped with auto-alert detectors and a manual alert button, serves to enhance safety in this industry. Problems with vehicles can be detected before they cause a real emergency, and fleet managers can view and respond to unusual occurrences even if the driver is unaware or unable to report the problem. When emergencies occur, data with location and time continually send to guide emergency responders to the proper location. We hope that this device will make roads safer for trucks and regular commuters alike.