Problem Statement:

IOT01
Topic: ECO-DRIVING DRIVER ASSISTANCE SYSTEM.

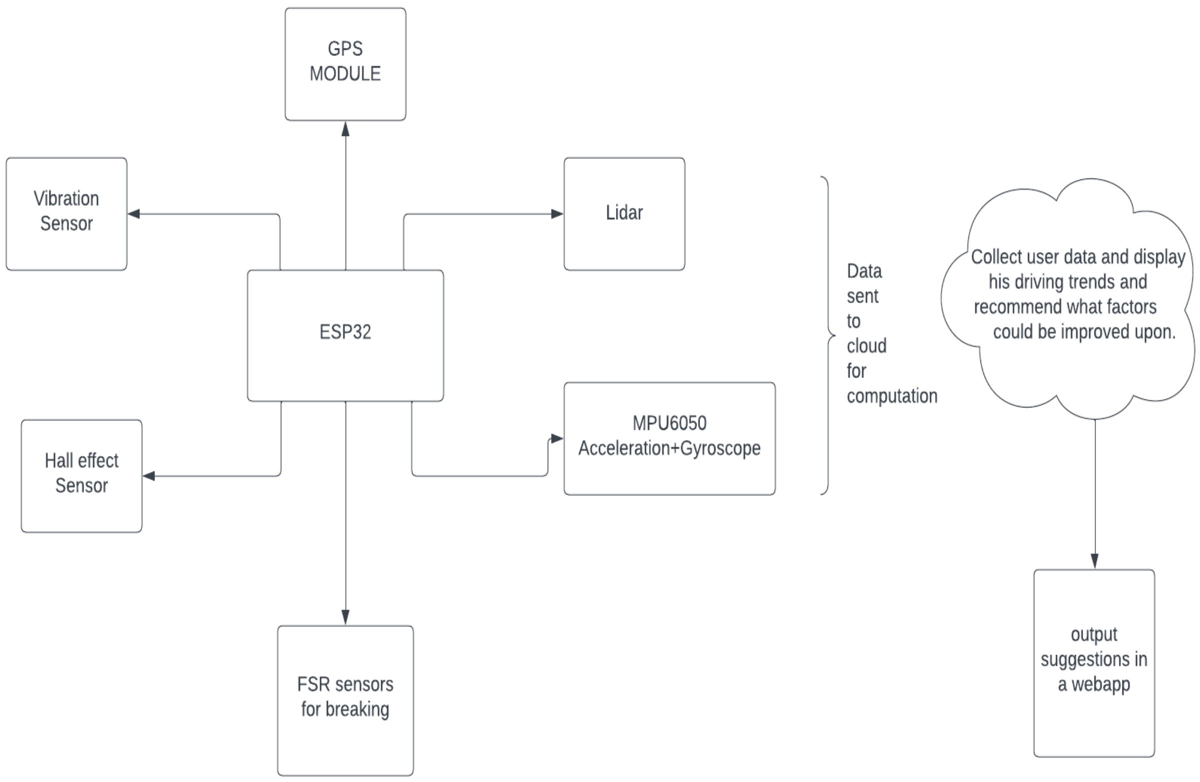
Team- 594092-U9JF25C6
College: NIT TRICHY

Team members:
Kamalesh S
Shraddha Pai
Sreenidhi Balachandran

PROPOSED SOLUTION:(Attempted brownie points)

Our driver assistance system consists of an electronic hub consisting of multiple sensors fetching raw data of parameters such as acceleration , speed, distance , brake detection , vibration and relative distance b/w vehicles on the road .

The driver assistance system at fixed intervals of time, transfers the vehicle's above-mentioned parameters to a remote server **(using Thingspeak)**. After processing the parameters, the
remote server alerts the user when a principle is being violated on an application
and makes the suggestions mentioned above in the form of an app.


GPS MODULE
Vibration Sensor
Lidar
ESP32
Hall effect Sensor
MPU6050 Acceleration+Gyroscope
FSR sensors for breaking
Data sent to cloud for computation
Collect user data and display his driving trends and recommend what factors could be improved upon.
output suggestions in a webapp

Implementation

Initial idea -

We intended to interface a raspberry pi microcontroller with the sensors in proteus, pairing it with ESP 8266 wifi module. We chose the simulation platform after careful consideration and experimentation with other platforms and found that only Proteus provided a very wide selection of sensors. We were able to import several libraries to use the sensors we planned to use. The sensors we initially decided on were:

- **Hall effect sensor for measurement of speed** - we came to the realisation that the best way to measure speed is to use hall effect to measure the rpm of the wheels. The way to implement this in a real life model would be to use a magnet on the wheel and every time the wheel completed one rotation, the sensor would detect changes in the magnetic field as the magnet passed by.
- **MPU6050 for measurement of acceleration** - since the problem statement provided the max limit of acceleration correct up to the fourth decimal digit, we decided that we needed to use a sensor with high accuracy. Thus we decided to go with MPU 6050 which is a Micro Electro-Mechanical Systems (MEMS) which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion related parameters of a system or object..
- **Force Sensitive Resistor (FSR) to detect breaking**- the FSR sensor would be placed on the brake pedal and it would detect breaking by changing its resistance based on the pressure applied.
- **Lidar (Light Detection and Ranging) sensor** - to detect obstacles and other vehicles. It uses laser beams to measure distances. It can provide high-resolution 3D mapping and can accurately measure distances up to hundreds of metres away. It is also not affected by ambient light or fog, making them well suited for outdoor applications where visibility can be an issue.
- **Vibration sensor to detect idling**- a vibration sensor would be placed on the engine and in the case where the speed was 0km/h but the engine was still vibrating, it would be detected as idling.
- **GPS module** - to track the location of the vehicle

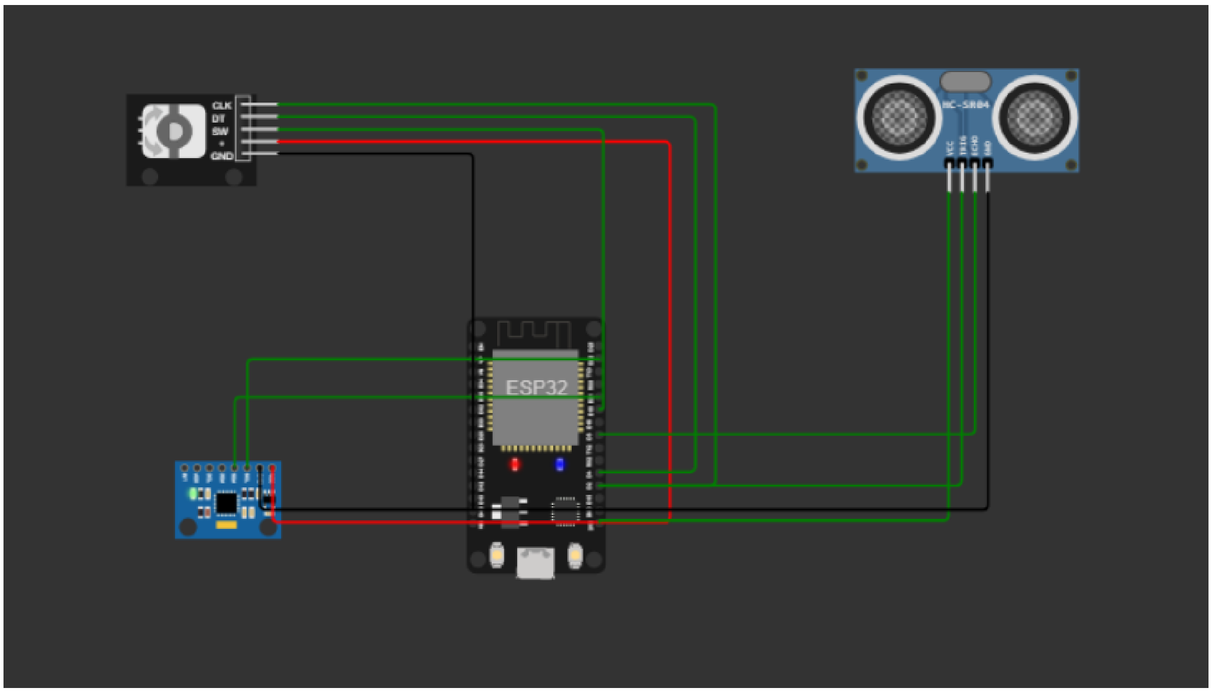
All the data collected by these sensors would be transmitted to the cloud- Thingspeak platform via the Raspberry Pi using the ESP8266. On the cloud, the data would be processed, and subsequently, the driver would receive alerts and suggestions based on the algorithms set .

Assisting the driver by providing him/her with the optimised acceleration data will be useful to control fuel emissions. We initially looked for **ML/DL solutions** to implement this. The plan was to have a LiDAR sensor on the vehicle and map the environment around. A publicly available dataset (provided by Lyft) had multiple data points of real-time LiDAR footage which could have been useful in training the model. However, due to the sheer size of the training dataset and limited time and resources available we couldn't proceed with it.

Unfortunately, we were unable to program the Raspberry Pi in Proteus despite our best efforts. Thus, we decided to shift to the Wokwi platform. We chose to work with ESP 32 with MicroPython firmware. Due to the limited selection of sensors on the platform, we decided to use the following sensors for this project:

- **Ultrasonic sensor to detect obstacles**- ultrasonic sensors use high-frequency sound waves to measure distances. They are relatively cheap and can measure distances up to a few metres.
- **Rotary encoder to measure speed**- It works by generating a series of pulses as the shaft rotates, and the frequency of these pulses can be used to determine the speed of rotation. In this application, the rotary encoder is mounted on the shaft of a motor or other rotating device. As the shaft rotates, the encoder generates a series of pulses, which are sent to a counter or other electronics that can measure the frequency of the pulses. The speed of the shaft can then be calculated by dividing the number of pulses generated in a given time interval by the time interval.
- **MPU6050 to detect acceleration** as specified earlier.

The connections of the 3 sensors were made to the esp32 as follows:



Along with interfacing the sensors to the ESP32, we also connected to wifi using MicroPython.

SOFTWARE PART:

The software part is mainly divided into two parts:

1) Sending data from sensors to cloud
2) Analysing and checking the various parameters involved and the producing output in a Desktop App

1) Sending data from sensors to cloud:

The data from these sensors may be either directly transmitted to the cloud through the ESP module. Due to the lack of the required sensors on Wokwi, we decided to go with making a separate Python script with sample sensor data. Using an URLLib, the data from sensors are added as separate fields. The fields involved are acceleration, velocity, brake, latitude and longitude. Since we are using a **free version of Thingspeak**, we aren't able to transfer this data instantaneously to the cloud. Thingspeak introduces a time delay of 15 s for any data to be sent and stored in the cloud. Hence, we will be able to do real-time communication only if we have access to the paid version of cloud service providing softwares.

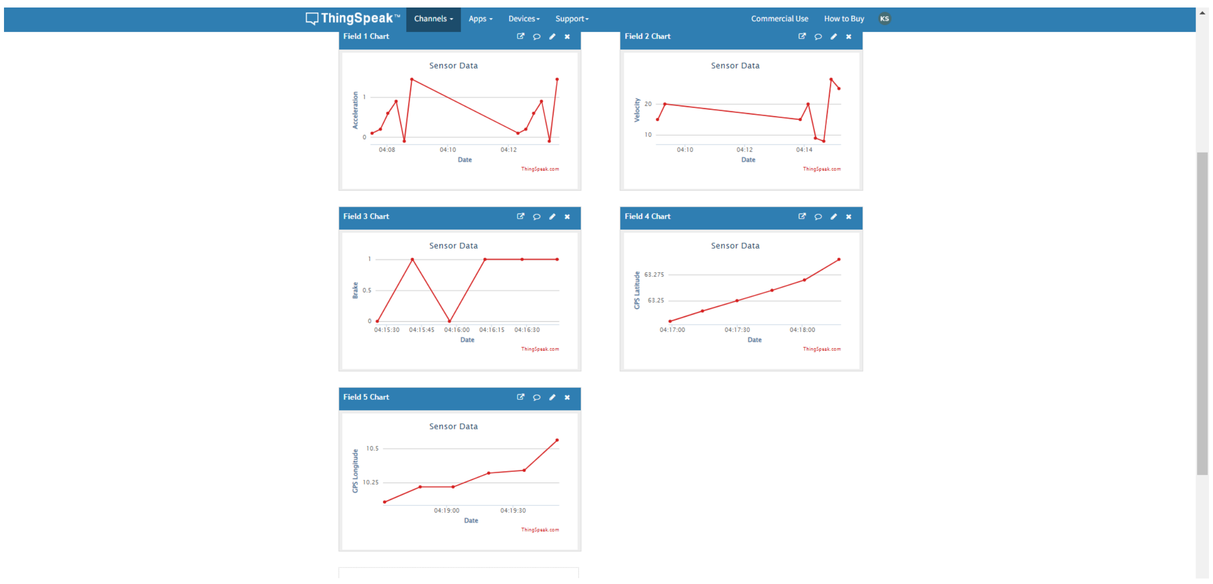


Fig.: How each field shows the data received in cloud (Thingspeak)

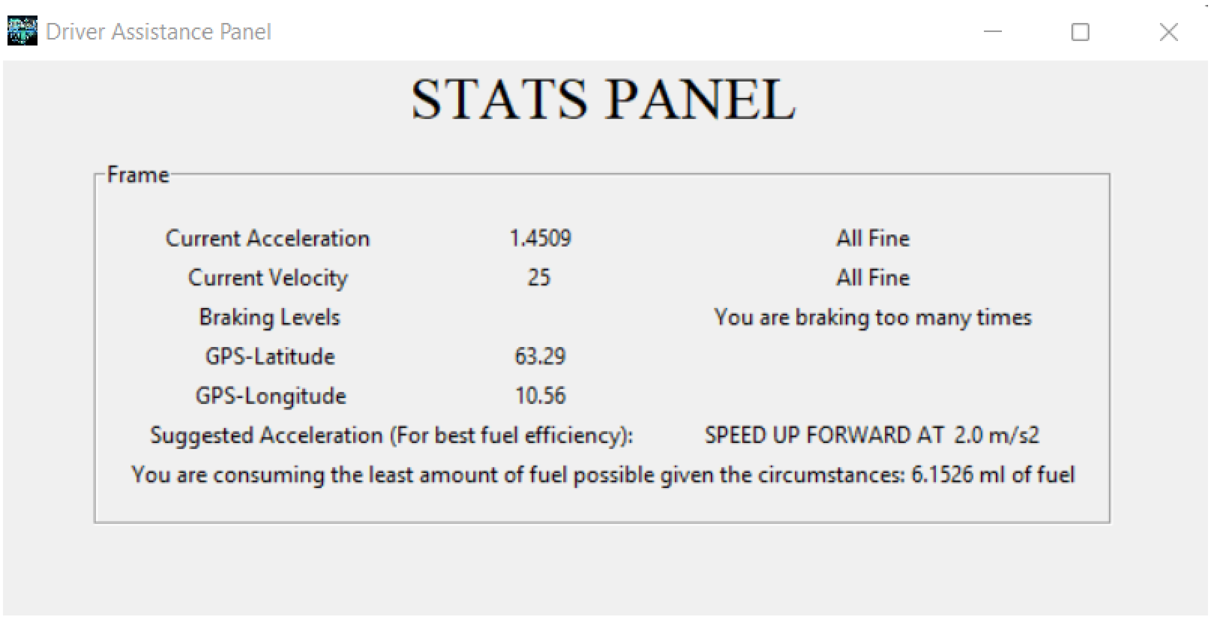
2) Analysing and checking the various parameters involved and the producing output in a Desktop App:

The data from the cloud was retrieved back by sending a request which resulted in a JSON file. This JSON file had all the required data for analysis. Here, we have split the analysis into two parts again,

a) **Simple checking of all parameters given in the problem statement** : Achieved using if-else statements.
b) **State of the art and a Novel method** to minimise fuel consumed and optimise the acceleration while taking into consideration the relative position, velocities and accelerations of surrounding vehicles. To achieve this we have used the SciPy module's minimise function. Here we are **minimising the cost function** and after a lot of deliberation we decided that **Fuel consumed** and **relative velocities** are the

variables that are to be minimised, while all other parameters act as constraints. To optimise acceleration, we have used an optimization algorithm named, **Sequential Least Squares Programming (SLSQP)**. We give an output suggesting the driver to follow an acceleration (along with the direction) that will help conserve fuel.

The Desktop App:
A simple desktop has been made using python's in-built GUI interface called tkinter. The current values of acceleration, velocities, GPS coordinates are shown. An alert is made when any of the parameters are violated. This part of the output updates every 3 seconds. Apart from this, the optimal acceleration along with the fuel consumed to is shown too and this part updates every 18 seconds.



STATS PANEL

Current Acceleration	1.4509	All Fine
Current Velocity	25	All Fine
Braking Levels		You are braking too many times
GPS-Latitude	63.29	
GPS-Longitude	10.56	

Suggested Acceleration (For best fuel efficiency): SPEED UP FORWARD AT 2.0 m/s2

You are consuming the least amount of fuel possible given the circumstances: 6.1526 ml of fuel

Fig.: A screenshot of the final Desktop App showing all necessary stats.