Real-Time Fleet Tracking and Diagnostic System Using CAN Bus and GPS

S V Naga Lakshmi

Assistant Professor, PERI IT, Chennai, India.

Abstract

With the advancement of telematics and intelligent transportation systems, the need for real-time monitoring of vehicle performance and location has become increasingly critical, particularly for large fleet operations. This paper presents the design and implementation of a real-time fleet tracking and diagnostic system that integrates Controller Area Network (CAN) bus communication with Global Positioning System (GPS) technology. The proposed system acquires essential vehicular parameters—including engine speed (RPM), coolant temperature, vehicle speed, fuel level, and diagnostic trouble codes—directly from the vehicle's electronic control units via the CAN protocol. Concurrently, GPS data provides continuous location updates. All information is transmitted to a centralized server using wireless communication (e.g., GSM/4G) for real-time visualization and analysis via a web-based dashboard.

To enhance safety and regulatory compliance, particularly in the Indian automotive context, the system incorporates ARAI-compliant alert mechanisms. These alerts are triggered by predefined thresholds such as over speeding, harsh braking, and engine fault conditions. The system architecture is modular and scalable, allowing integration across various vehicle types and fleet sizes. Experimental validation demonstrates the effectiveness of the system in improving operational efficiency, proactive maintenance, and fleet safety. This work contributes a comprehensive, cost-effective solution for vehicle telematics and diagnostics within the domain of intelligent transportation systems.

Keywords: Fleet management, CAN bus, GPS tracking, vehicle diagnostics, intelligent transportation system (ITS), telematics, real-time monitoring.

Introduction

In the era of intelligent transportation systems and digital transformation, fleet management is undergoing a paradigm shift from traditional tracking methods to advanced, data-driven solutions. Logistics companies, public transport operators, and private fleet managers are increasingly seeking comprehensive systems that not only track vehicle movement but also monitor the internal health of the vehicle in real time. This research addresses these needs through the development of a **Real-Time Fleet Tracking and Diagnostic System** that integrates **Controller Area Network (CAN) bus communication** and **Global Positioning System (GPS) tracking**, analysed and visualized through powerful Python tools.

The **CAN bus**, a robust vehicle communication standard widely used in modern automotive systems, enables access to internal electronic control units (ECUs) without the need for invasive hardware modifications. This allows the system to gather real-time data such as engine RPM, vehicle speed, fuel level, throttle position, coolant temperature, and diagnostic trouble codes (DTCs), providing insights into the operational status of the vehicle. When combined with **GPS data**, which provides precise location coordinates, the system enables synchronized tracking and diagnostics, allowing fleet managers to identify not only *where* a vehicle is, but also *how well* it is performing.

To handle and analyse this data effectively, **Python programming** was used as the backbone of the data processing pipeline. Libraries such as python-can, pandas, and matplotlib were employed to parse CAN messages, filter relevant parameters, and visualize trends such as speed profiles, engine load over time, and frequency of faults. This analysis supports predictive maintenance and performance benchmarking, reducing downtime and repair costs.

On the location-tracking front, **Folium**, a Python library built on top of Leaflet.js, was used to create interactive web-based maps that visualize GPS data. With Folium, vehicle routes are plotted dynamically, enabling route replay, hotspot identification (e.g., frequent stops or idling zones), and geofencing capabilities. This geographic visualization enhances decision-making in route planning, driver behaviour analysis, and real-time alerts for deviations from predefined paths.

This system is designed with modularity in mind, making it adaptable to different vehicle platforms and fleet sizes. Data from vehicles is transmitted over wireless networks (e.g., GSM or Wi-Fi) to a cloud-based server, where it can be accessed via a secure dashboard interface. By merging **hardware-level diagnostics** with **cloud-based telematics** and **Python-powered analytics**, the system delivers a modern solution for fleet oversight, offering significant improvements in efficiency, safety, and cost management.

System Architecture

The system architecture consists of both hardware and software components integrated to provide real-time tracking and diagnostics of vehicles. The design is modular and scalable, allowing deployment across multiple vehicle types and environments.

1.1 Hardware Components

The real-time fleet tracking and diagnostic system is built using a robust selection of high-performance hardware modules to ensure accurate data acquisition, real-time processing, and reliable communication. The components used in the system are as follows:

1. Microcontroller Unit (MCU):

Microchip PIC32MZ1024EFG100-I/PT

This 32-bit MCU belongs to the PIC32 EF family and features a high-performance MIPS core with a dedicated Floating-Point Unit (FPU). It is specifically designed to handle complex tasks such as audio processing and graphical user interfaces. The MCU supports High-Speed USB and Ethernet connectivity, making it suitable for real-time communication with external systems and servers. Its extensive analog capabilities, large flash memory (1 MB), and multiple I/O ports make it the central control unit for managing both CAN and GPS data streams.

2. CAN Transceiver:

Texas Instruments TCAN1042HVDRQ1

This CAN transceiver is compliant with ISO 11898-2:2016 and supports CAN FD (Flexible Data-rate) networks up to 2 Mbps. Variants with a “G” suffix can operate at speeds up to 5 Mbps, allowing higher data throughput for complex diagnostics. The TCAN1042HVDRQ1 is used to establish communication between the microcontroller and the vehicle’s Controller Area Network (CAN) bus, enabling access to various vehicle parameters and diagnostic information.

3. GPS Module:

Telit SL869T3-I

The SL869T3-I is a high-performance GPS module that provides accurate real-time geolocation data, including latitude, longitude, speed, and timestamp. Its small form factor and low power consumption make it ideal for embedded automotive applications. The module supports multiple GNSS systems (GPS, GLONASS, Galileo), enhancing positioning accuracy and reliability even in urban environments.

4. Wireless Communication Module:

AGS2-E

The AGS2-E module facilitates wireless data transmission from the onboard unit to a remote server or cloud platform. It supports various communication protocols over GSM/3G/4G networks, allowing for real-time fleet tracking and remote diagnostics. The module ensures continuous connectivity, even in mobile and geographically dispersed environments.

5. Power Supply Unit (PSU):

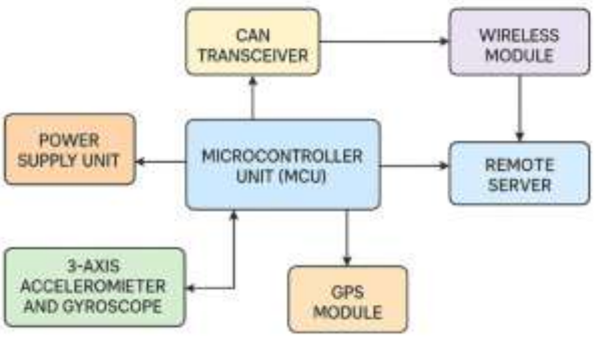
The PSU is responsible for converting the vehicle's standard power supply (typically 12V or 24V) to logic-level voltages (3.3V or 5V) required by the microcontroller and peripheral components. A well-regulated supply ensures protection against voltage spikes, overcurrent, and short circuits, which are common in automotive electrical systems.

6. 3-Axis Accelerometer and Gyroscope:

Bosch SMI130

The SMI130 combines a 16-bit digital gyroscope and a 12-bit digital accelerometer in a compact package. It offers programmable measurement ranges and is capable of detecting motion, orientation, and shock events. This sensor plays a crucial role in driver behavior analysis and accident detection by capturing dynamic motion patterns such as harsh braking, sharp turns, and collisions.

Block diagram of the system:



1.2 Signal/Data Flow Description

The data flow between major components explained below:

1. Vehicle ECU → CAN Bus

The vehicle's Electronic Control Unit (ECU) continuously generates diagnostic and performance data (e.g., RPM, temperature, speed). This data is placed on the CAN bus for communication among onboard systems.

2. CAN Bus → CAN Transceiver

The CAN transceiver listens to the CAN bus and converts the differential CAN signals into standard digital logic levels. It acts as the interface between the vehicle's CAN network and the microcontroller.

3. CAN Transceiver → Microcontroller

The microcontroller receives decoded CAN messages via SPI or UART from the CAN transceiver. It parses the messages, extracts key data fields (e.g., engine temperature, fuel level), and stores them in memory buffers.

4. GPS Module → Microcontroller

The GPS module sends real-time NMEA data strings over a serial (UART) connection. The microcontroller decodes this data to extract latitude, longitude, time, and speed.

5. Accelerometer and Gyroscope → Microcontroller

The SMI130 sensor communicates with the microcontroller via I2C or SPI. It provides motion-related data (e.g., acceleration, angular velocity), used for detecting harsh driving or accidents.

6. Microcontroller → Wireless Module

The microcontroller formats CAN + GPS + IMU data into a packet (e.g., JSON or CSV format). The packet is sent over UART or SPI to the wireless module, which handles the transmission.

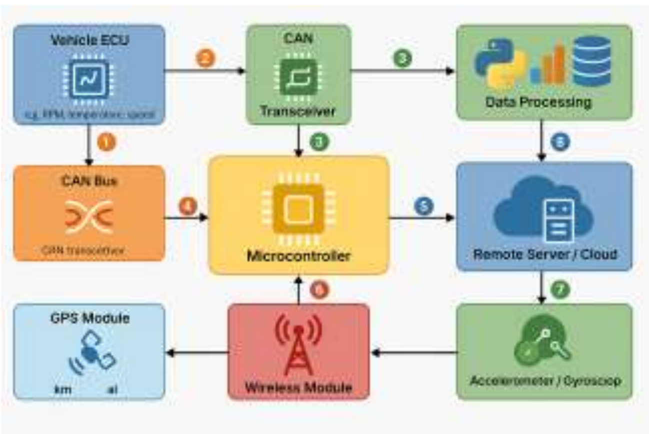
7. Wireless Module → Remote Server / Cloud

The AGS2-E module transmits the data using GSM/4G protocols to a remote server over the internet. Data can be sent using HTTP POST, MQTT, or WebSocket protocols.

8. Remote Server → Data Processing

The backend receives the data and stores it in a relational database (e.g., PostgreSQL). Python scripts use libraries such as: python-can to decode CAN message logs

- pandas for processing and analysing numerical data
- folium for GPS data mapping
- matplotlib for plotting diagnostics and trends



1.3 Software Stack

1. Embedded Software:

Written in C, it handles Reading CAN messages (engine RPM, speed, fuel level, etc.), Parsing GPS coordinates, Formatting and sending data via 2G/4G.

2. Backend Server:

A Python-based server receives data via HTTP or MQTT and stores it in a structured database.

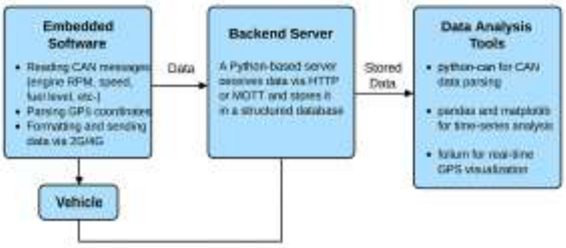
3. Data Analysis Tools:

Python libraries:

- python-can for CAN data parsing
- pandas and matplotlib for time-series analysis

- folium for real-time GPS visualization on maps

4. Software Block diagram



Methodology

1. Data Acquisition

The microcontroller continuously listens on the CAN bus using the MCP2515 interface. Selected CAN Id (Parameter IDs) such as engine RPM, vehicle speed, fuel level, and coolant temperature are polled. Simultaneously, the GPS module provides NMEA sentences (e.g., GPGGA, RMC) from which latitude, longitude, and time are extracted.

2. Data Packaging and Transmission

CAN and GPS data are formatted into CSV structure. This data is transmitted at fixed intervals every 60 seconds) using the GSM module to a remote backend server.

3. Data Storage and Processing

The server receives data and stores it in a time-stamped format. CAN data is cleaned, parsed, and saved as structured tables. GPS data is extracted and stored with geolocation coordinates.

4. Diagnostic Analysis (Python)

CAN Data**:** Fault detection based on thresholds (e.g., high engine temperature). Pattern recognition to identify unusual behaviour (e.g., frequent braking, over-revving). Visualization using matplotlib and seaborn.

GPS Data**:** Vehicle routes are plotted using folium. Markers, polylines, and heatmaps indicate real-time location and movement history. Custom overlays like geofences or stop zones can be added.

5. Alerts and Notifications

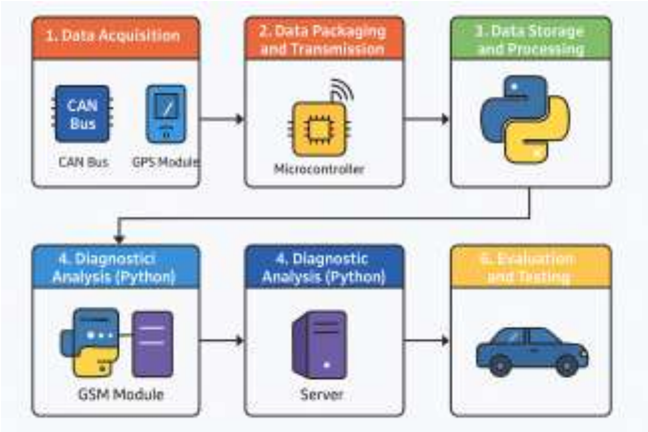
Based on diagnostic results (e.g., high engine load, critical errors), the system can trigger SMS/email alerts. Alerts can be configured using Python scripts with integration to services like Twilio or Firebase.

6. Evaluation and Testing

The system is deployed on test vehicles.

Data is collected during various drive cycles and evaluated for accuracy, latency, and reliability.

Fault conditions are simulated to verify alert and diagnostic functionality.



Packet Formats

1. LOC-Packet GPS Coordinates using Folium

A Location (LOC) Packet in automotive testing, particularly in ARAI or related vehicle certification systems, refers to a data structure that contains real-time or logged GPS-based location data of a test vehicle.

Location Packet

Field	**Description**
Start Character	Start of the packet
Device Id	Unique ID of the Vehicle (IMEI Number)
Sequence Number	Sequence Number of the packet
Latitude	Latitude in decimal degrees - dd.mmmmmm format
Longitude	Longitude in decimal degrees - dd.mmmmmm format
Timestamp	Time value as per GPS date time in UTC format (hhmmss)
GPS Speed	Speed of Vehicle as Calculated by GPS module in VLT. (in km/hrs.) (Upto One Decimal Value)
GPS Odometer	GPS Odometer of the Vehicle Odo
Live	Live or History Packet
Satellites Used	Number of satellites used in the GPS fix

2. Alert -Packets using Python Pandas

Field	**Description**
Start Character	Start of the packet
Header	Identifier for the packet
Vendor ID	Vendor identification header
Firmware Version	Version details of the firmware used
Packet Type	Specifies the type of packet Examples: NR = Normal, EA = Emergency Alert, TA = Tamper Alert, etc.
Packet Status	L = Live, H = History
IMEI	Unique 15-digit identifier of sending unit
Vehicle Reg. No	Mapped vehicle registration number
GPS Fix	1 = GPS fix, 0 = Invalid
Date	GPS date (DDMMYYYY)
Time	GPS time (hhmmss, UTC)
Latitude	Decimal degrees (≥6 places)
Latitude Dir	Direction: N = North, S = South
Longitude	Decimal degrees (≥6 places)
Longitude Dir	Direction: E = East, W = West
Speed	Vehicle speed (km/h, 1 decimal)
Heading	Course over ground (degrees)
No of Satellites	Satellites used in GPS fix
Altitude	Altitude in meters
PDOP	Positional Dilution of Precision
HDOP	Horizontal Dilution of Precision
Network Operator	Name of mobile network operator

Field	**Description**
Ignition	1 = Ignition On, 0 = Off
Main Power Status	0 = Vehicle battery disconnected, 1 = Reconnected
Main Input Voltage	Source voltage (in volts)
Internal Battery Voltage	Internal battery voltage (in volts)
Emergency Status	1 = On, 0 = Off
Tamper Alert	C = Cover Closed, O = Open
GSM Signal Strength	Signal strength (0–31)
MCC	Mobile Country Code
MNC	Mobile Network Code
LAC	Location Area Code (Hex)
Cell ID	GSM Cell ID
NMR	Network Measurement Report – Neighbouring cells' info
Digital Input Status	4 external digital input status
Digital Output Status	2 external digital output status
Frame Number	Sequence number (000001–999999)
Checksum	Validates data integrity (Optional)
End Character	Indicates end of the frame

3. CAN -Packets using Python Pandas

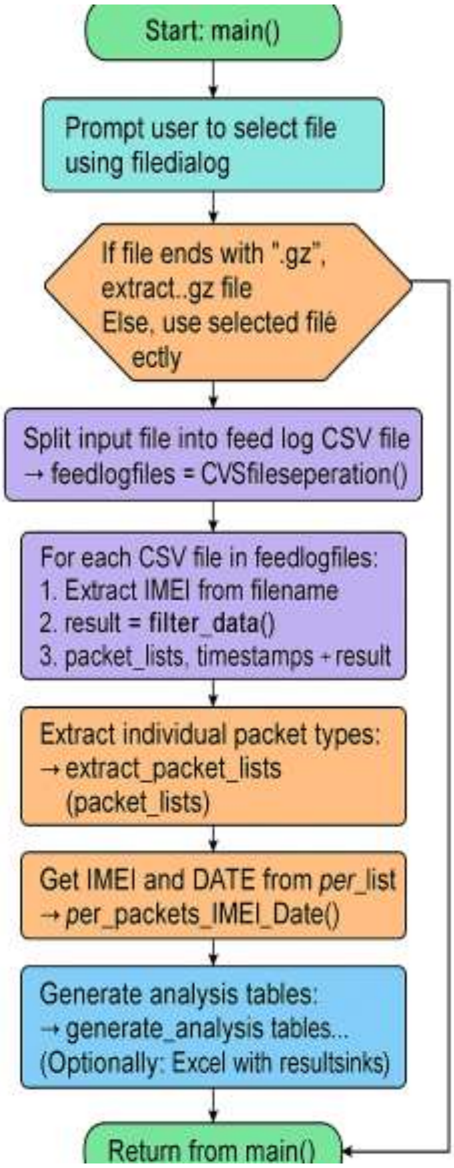
Field	**Description**
Message Type	Type of message
Device ID	Unique ID of the sending device
Sequence Number	Message sequence identifier
Latitude	GPS latitude (decimal degrees)
Longitude	GPS longitude (decimal degrees)
UTC	Coordinated Universal Time (hhmmss)

Field	Description
HRLFC	High Resolution Low Frequency Clock
Sweet Spot	Indicates if engine is in sweet spot RPM
Top Gear	Indicates if top gear is engaged
Sweet Spot Percent	% of operation in sweet spot RPM
Seconds	Second (from timestamp)
Minute	Minute (from timestamp)
Hour	Hour (from timestamp)
Month	Month of transmission
Day	Day of the month
Year	Year
Minute Offset	Time offset in minutes
Hour Offset	Time offset in hours
Total Distance	Cumulative distance travelled (in km)
Fuel Level	Fuel level in percentage or litres
Amber Warning Lamp	1 = On, 0 = Off
Red Stop Lamp	1 = On, 0 = Off
Malfunction Lamp	1 = On, 0 = Off
Flash Mal Function Lamp	1 = Flashing, 0 = Off
SPNLSB	Suspect Parameter Number (Least Significant Byte)
SPN8_2ndByte	SPN Middle Byte
Failure Mode	Failure mode indicator (FMI)
SPN3_MSB	SPN Most Significant Byte
Occurrence Count	Number of times the fault occurred
CCA	Calibration Condition A
CCES	Calibration Condition ES
CCSS	Calibration Condition SS
Engine Speed	Engine speed in RPM
Engine Start Mode	Start mode (Auto, Manual)
Engine Operating Hours	Engine run-time in hours
Power Key Pos	Power key position (Off, On, Start)
Acc Pedal Idel Switch	Accelerator idle switch status
Vehicle Speed	Vehicle speed in km/h
Controller Trim Mode	ECU trim mode

Field	Description
Engine Oil Pressure	Pressure in kPa or bar
Engine Coolant Temp	Coolant temperature (°C)
Acc Pedal Position	Accelerator pedal position (%)
Trip Fuel	Fuel used in trip (litres)
Live	Live packet (1 = yes, 0 = stored)
Ignition Status	1 = On, 0 = Off
End Character	End of message indicator

Results and Discussion

1. Code flow after downloading Server feed Logs and doing analysis.



2. MAP from LOC packets using folium from downloaded Server feed Logs.

The sample data taken from the travel bus from Chennai to Pattukkottai, Tamil nadu and using LOC packet GPS Co ordinates plotted map using python.

3. GPS Data was analysed using PERIODIC packet of ARAI format using from downloaded Server feed Logs.

Positioning Error Rate(PER) Report

PER Parameters	Values
IMEI	['359207067893881']
No of GPS Packets	426
No of GPS Fix Packets	425
No of GPS Invalid Fix Packets	1
No of GPS Satellites(Min)	19
No of GPS Satellites(Max)	55
GSM Signal Strength(Min)	10
GSM Signal Strength(Max)	47
PER Speed (Min)	5.0
PER Speed (Max)	83.0
Ignition Status ON(Packets)	402
Ignition Status OFF(Packets)	24
Network Operator name is	['IND airtel', 'CellOne']
Minimum Internal Battery Voltage	3.7
Maximum Internal Battery Voltage	4.2
Minimum Vehicle Battery Voltage	0.0
Secondary Minimum Vehicle Battery Voltage	24.0
Maximum Vehicle Battery Voltage	27.7
Longitude (Max)	80.259904
Latitude (Max)	13.072935
PDOP(Mean) & Status	0.738, Perfect PDOP

4. Alerts from Vehicle are captured and analysed from downloaded Server feed Logs.

ALT Packet Analysis

Type of Alerts	Description	COUNT	Live	History
PER-NR(1)	Normal Packet(Live)(1)	326	326	0
PER-NR(2)	Normal Packet(History)(2)	100	0	100
----------	--------------------------	-----	----	-------
ALT-ALL	All System ALERTS	1048	880	168
ALT-BD	Battery Disconnect(3)	2	2	0
ALT-BL4	Battery Low(4)	3	2	1
ALT-BH5	Battery High(5)	2	2	0
ALT-BR	Battery Reconnect(6)	1	0	1
ALT-IN	Ignition ON(7)	4	4	0
ALT-IF	Ignition OFF(8)	4	3	1
ALT-TA	Tamper Alert(9)	0	0	0
ALT-EA	EA Trigger(10)	2	1	1
ALT-EA	EA Clear(11)	0	0	0
ALT-HB	Harsh Braking(13)	970	823	147
ALT-HA	Harsh Acceleration(14)	29	21	8
ALT-RT	Rash Turning(15)	25	18	7
ALT-DT	Device Tamper(16)	2	1	1
ALT-GeoIn	Geo-fence In/Entry(18)	0	0	0
ALT-GeoOut	Geo-fence Out/Exit(19)	0	0	0
ALT-CRT	ALT-Tilt(22)	0	0	0
ALT-IMPACT	ALT-IMPACT(23)	0	0	0
ALT-OS	ALT-OverSpeed(24)	4	3	1

5. CAN data like Vehicle Speed, Engine RPM, Total Distance , Fuel Level, and other parameters are analysed using Server feed Logs.

HRLFC (Maximum)	43599.44
EngineStartMode	3
Maximum EngineOperatingHours	4174.95
Second Minimum EngineOperatingHours	4166.4
Total EngineOperating Time (Mins)	513.0
TripFuel (Maximum)	1024.00
WaterInFuelIndicator	3
AmbientBarometricPre	102.00
Ambienttemperature	51.00000
EngCoolantlevel	102.00
ClutchONTime	6000
BrakeONTime	6000
DistanceinPowerMode	141429.00
FuelinPowerMOde	34561.512
DistanceinECOMode	17425.00
FuelinECOMOde	4220.559
DistanceinECOPlusMode	19303.00
FuelinECOPlusMOde	4817.014
DistanceinSweetSpotAcheived	142833.37
FuelinSweetSpotAcheived	32656.002
DistanceinCruiseMode	45.50
FuelinCruiseMode	5.500
TimeinPowerMode	108487.50
TimeinECOMOde	407.80
TimeinECOPlusMOde	459.25
TimeinSweetSpotAcheivedMode	2267.30
TimeinCruiseMode	0.00
Engine_total_idle_fuel_used	1256.00
Engine_Totaltime_Idle_Hours	690.05
AppliedVehicleSpeedLimit	0
Primary_brake_Pressure	1000
Secondary_brake_Pressure	1000
CAN Latitude (Max)	13.07284
CAN Longitude (Max)	80.25980
TotalDistance (Maximum)	178158.11
FuelLevel (Maximum)	76.80
EngineSpeed (Maximum)	1735.50
VehicleSpeed (Maximum)	86.57
EngineOilPressure (Maximum)	408
EngineCoolantTemp (Maximum)	95
Distance_inLowACMode(Max)	N/A
EngineHours_inLowACMode(Max)	N/A
Distance_inHighACMode(Max)	N/A
EngineHours_inHighACMode(Max)	N/A
Distance_inACMode(Max)	N/A
EngineHours_inACMode(Max)	N/A
CDslOxicatIntkGastmp	0.00000
DslOxicatOutkGastmp	390.18750
CatlistUpstrmTmp	383.10001
InletNoxConcentration	3076.75
OutletNoxConcentration	3076.75
Intakeairtemperature	215
TransmissionCurrentGear	0
ReverseGear	0
MultimodeSwitchstatusPOWER_ECO_ECOPULS	0
EngineOilTemperature	111.1875
SoftwareVersion	NA

6. Finally we doing the latency of the server, that no of packets reached to Cloud from the device, ideally with 10s, the data should reach to server.

Server Latency Analysis

Type of Packets	COUNT	Percentage
No Of Packets <5s	3439	51.637
No Of Packets 6-10s	674	10.12
No Of Packets 11-30s	460	6.907
No Of Packets 31-60s	107	1.607
No Of Packets 1-5Min	178	2.673
No Of Packets 6-10Min	168	2.523
No Of Packets 11-15Min	196	2.943
No Of Packets 16-30Min	495	7.432
No Of Packets 31-59Min	388	5.826
No Of Packets >1 Hour	555	8.333
Total No of Packets	6660	100
No of Packets <1min	4680	70.27

The results are also converted into PDF using matplotlib using pdfpages.

Conclusion

Using server feed logs, we can effectively analyze various aspects of vehicle telemetry using Python. This includes decoding and interpreting CAN data, processing periodic data such as speed and odometer readings, tracking the live GPS location of the bus on a map, and measuring server latency to evaluate communication delays. These insights help in monitoring vehicle performance, ensuring timely diagnostics, and improving overall fleet management efficiency.

References

1. https://hmr.araiindia.com/Control/AIS/14201910518PMAIS-140.pdf
2. https://www.geeksforgeeks.org/pandas-tutorial/
3. https://realpython.com/python-folium-web-maps-from-data/