In-Vehicle Communication and Data acquisition system

Ravishankar Holla 1, Akash Shanbhag 1, and Karmugilan Murugu 1

1Affiliation not available

October 31, 2023

Abstract

The paper briefly explains on CAN protocol, On-board Diagnostics and about the establishment of wireless data acquisition in the automotive domain. This paper can really help people to understand the overall network of in-vehicle system and about the STM32 mcu.
In-Vehicle Communication and Data acquisition system

An implementation of wireless On-Board Diagnostics system

Mr. Ravishankar Holla
Dept. of Electronics and Communication
RV College of Engineering
Bengaluru, India
ravishankarholla@rvce.edu.in

Akash Shanbhag
Dept. of Electronics and Communication
RV College of Engineering
Bengaluru, India
akashsshanbhag.ec20@rvce.edu.in

Karmugilan
Dept. of Electronics and Communication
RV College of Engineering
Bengaluru, India
karmugilanm.ec19@rvce.edu.in

Abstract—In the current world scenario, Electronic systems play an important role in automotive field. Due to advanced technologies, automotive systems are more efficient and precisely controlled by electronic systems. This paper, takes a step towards this vision by developing a system that could be easily connected to the Electronic Control Unit of vehicle using Raspberry Pi, STM32 microcontroller, and Wi-Fi, that connects the On-Board Diagnostics of the vehicle with a Smartphone or any localhost devices.

The problem statement is Collection and monitoring of a Vehicle's data for the purpose of Onboard Diagnostics and Remote Diagnostics. This speaks about the data collection from various sub-modules of the car and Integrate it to a single bus for easy and reliable communication. The approach is to integrate a bus network that interfaces with all the sub-modules and collects data to put it on to this network. Further, STM32 microcontroller is used to sniff this data and transmit to Raspberry pi that creates a local network for further transmissions for localhost devices. A very easy and user-friendly display is designed to exhibit all the collected data. Further, This can be easily implemented to all the currently available electric vehicles in the market with minor modifications based on the vehicle parameters.

Index Terms—Controller Area Network(CAN), OBD, STM32 microcontroller, Raspberry Pi, Thingsboard, Fleet management.

I. INTRODUCTION

Due to the microprocessor revolution, which has made automobiles more sophisticated and internet-connected, businesses are being urged to adopt these components for vehicle tracking and monitoring. A modern automobile may have up to 70 electronic control units for different subsystems since it is equipped with a Controller Area Network (CAN) bus. Due to the integration of several electronic control units and sensors in a relatively small system, such as a car, data collecting systems are used to monitor system activities that can be further improved to detect faults. As a result, the idea of an on-board diagnostics system is born.

Monitoring the operations taking place in the vehicle engine has long attracted researchers who have been interested in effectively monitoring, understanding and even improving their usage. In the present world scenario, the On-board diagnostic port is a hardware part and it needs specific tools and connectors to access the information. This creates a problem for a normal user to get info about the car. So this study focuses on data collecting for on-board diagnostics, which provide technicians with access to information gathered by a car’s numerous subsystems for the purposes of performance analysis and repair requirement analysis. The objectives is to establish a controller area network to connect the different modules of vehicle to a single bus. Then, a data acquisition system is developed that collects the data from the CAN Bus and transfers it to Raspberry Pi. These data will be intercepted and transmitted wirelessly to Thingsboard software. The user friendly display model collects these data and gives a visual aesthetic dashboard on Thingsboard.

II. RELATED WORKS

On-Board Diagnostics (OBD) system has been mandatory equipment for diagnostic emission performance and vehicle maintenance. Products for reading information from the OBD interface are common and widely used by the vehicle service engineer. A hands-on project that uses CAN bus equipment to obtain the battery and motor information for a Nissan Leaf is presented in [8]. A detailed look at extracting CAN bus data for the Toyota Prius and Ford Escape has been presented in [4], where the authors demonstrate analysis and injections of CAN packets to control vehicles via a laptop and CAN bus tools. Several interesting applications have been built by research
teams based on OBD data. Using OBD data from combustion vehicles to build an eco-routing system has been shown [5], as using OBD data to create personalized energy consumption prediction models for different types of vehicles[6] [7].

III. WIRELESS ON-BOARD DIAGNOSTIC UNIT

The Wireless On-Board Unit(OBU) module proposed in this paper is designed to acquire the vehicle operation information, e.g., speed, battery voltage, rpm, steering angle, and others from Controller Area Network(CAN) and STM32 microcontroller.

In the following, CAN module is discussed along with the OBD system of the car, the data acquisition system used to transmit the data and its implementation details, and the user panel as an On-Board Display.

A. Controller Area Network

The CAN specifications used in this paper is based on "BOSCH CAN spec 2.0" and follows ISO standards. Here, the physical layer and datalink layer specifications of CAN are implemented. The model for this is shown in figure 2. Initially, all the sub-modules will be connected to CAN Bus as shown in figure 3.

Devices on the CAN bus are called nodes. All nodes on the CAN bus are connected in parallel, meaning that each node is connected to all the other nodes on the network. The advantage of using this type of network is that it gives each node equal priority to transmit the data. The data acquired from all the nodes are put into the bus network and it is broadcast to all the connected devices. The STM32 Microcontroller captures this data from the CAN Bus.

Here, STM32 is specifically selected due to its high performance core and it has an inbult CAN communication pins. This MCU contains two inbult CAN that can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers. And also 256 bytes of SRAM are allocated for each CAN. The pins PA11 and PA12 of MCU gives the information on CAN_RX and CAN_TX respectively. Also there is 120 ohms resistance that needs to be connected at each termination node of the CAN Bus. This is very important, as the CAN bus termination should match the nominal impedance of the cables. Therefore 120 ohms are considered as standard resistance.

B. Interface between the CAN and STM32

To communicate with the CAN Bus, the utilization of CAN transceiver IC is considered. The IC acts as an intermediate transmitter/receiver pair to interface the STM32 to the CAN Bus. Few points have to be considered while establishing the connection between CAN and STM32 MCU:

- The CAN_RX and CAN_TX on the CAN transceiver chip will go to PA11 and PA12 on the STM32. This gives the CAN reception and transmission signals.
- The CANL pin should be connected to the other CANL pins of the other bus nodes. The same goes for the CANH pins.
- The 120 ohms resistor across the CANH and CANL pins is possibly required in the event that the node is a terminal node. This implies that it is at the end of parallel connection wiring. All in all, the CAN transport ought to just have two 120 ohms resistors in it, and they ought to be as far separated from one another as could really be expected.

Now that the transceiver circuit is connected to the STM32, writing messages to the CAN bus is possible. To read from the CAN bus, first the ID of the CAN message has to be known. Every message should have a unique ID, with lower IDs having higher priority. When writing a CAN message, here that CAN messages are of multi-byte. Each written message must have an ID and length.

STM32CubeIDE software is used to program the MCU. This development tool gives a platform for peripheral configuration, code generation, code compilation, and debug features for STM32 MCU. Using this tool the filtering out of messages are executed. The STM32 have the Filters built inside the CAN peripheral.

C. Local hosting through Raspberry Pi

Raspberry Pi board has been utilized for gathering information from the STM32 through USB communication. Here, Raspberry Pi gathers the information and uses Message Queuing Telemetry Transport(MQTT) broker to send data to the Thingsboard server. Data is also sent to a router that creates a local network so that the information can be accessed locally with appropriate authorization. The figure 4 shows the message format that is being forwarded from STM32 MCU to Raspberry Pi over Universal Serial Bus(USB) communication. The message format shows only the data signals and does not include control information which is also transmitted along
The use of USB communication is due to its faster transfer rate over other type of serial communication. It is more efficient and swift compared to RS232 and parallel ports. “VNC viewer software” is used to connect the Raspberry Pi board and access the Adafruit OS present in it. This OS is used to install the necessary files to integrate and establish the USB communication.

Connecting the Raspberry Pi board to the system through LAN connection gives access to Adafruit OS that is running on the Raspberry Pi. Through the Python programming the Raspberry Pi is made to connect with the router to host a network, that will connect to thingsboard server and transmit the captured data. Router plays an important role in transmission of data from Raspberry Pi to the thingsboard database systems. A finite state algorithm similar to the one used in USB interfacing is used to check server-client connection reliability.

A lightweight and efficient protocol like MQTT allows the data to be monitored and controlled more effectively. By the use of the publish/subscribe model that allows edge-of-network devices to publish to a broker makes it much more expedient. Finally the Raspberry Pi hosts the network to transmit the data to Thingsboard. The data is stored as a time-series data in Thingsboard telemetry database. This thingsboard gives an user friendly experience to analyse the accumulated data. The On Board Display is also designed over Thingsboard software.

D. Visualization of data through Local hosting display

Thingsboard software gives a very user-friendly interface to display the collected values. As given in the below image for reference Fig 5. The data transmitted by the Raspberry Pi is stored in time-series form. This time-series data stored in the database is mapped to each GUI elements on the dashboard. The dashboard is designed to give a metric view of the collected data. This software provides a very unique way to just drag and drop the metric boxes and provide the certain time series inputs for that. This way the display for on-board diagnostics is designed.

IV. IMPLEMENTATION

In this chapter, complete implementation of the project is executed. Along with the results that can be observed over the end device connected to the local-host network. The design and development of CAN Bus is efficiently supported and distributed by real-time control with a very high level of reliability. The model diagram is shown in figure 6. Here, the physical layer and datalink layer specifications of controller area network are implemented. Data collected from the CAN Bus in STM32 board is then forwarded to Raspberry Pi. The data is forwarded using USB communication. The data is stored in 10 byte buffer in the following format as shown in the figure 4. This data consists information about the Speed of Left wheel, Speed of Right wheel, Differential speed between them, Angle of steering, Battery voltage and Current consumption.

In the Raspberry Pi CPU the USB exception is handled. The flow diagram is explained in the below diagram 7. This explains the exception handling in the Raspberry Pi with the USB. If the USB is not connected the exception is generated to state the USB is not communicating and COM port is not generated. Here the exception is handled in the Raspberry Pi CPU which internally generates the exception. The flow diagram explained below shows the clear execution of the model. The data transmitted in 10 byte buffer consists information about the Speed of Left wheel, Speed of Right wheel, Differential speed between them, Angle of steering, Battery voltage and Current consumption. This data is of 10 bytes and can be enhanced for future applications. Here different wheel speed (Left and Right) is calculated separately to get the differential speed. This differential speed can be used while steering the angle of the car to a particular direction. The battery voltage gives the cumulative voltage of 12 batteries connected to the system. Current consumption data can be
used to find the excess current consumption, to then find the errors in the system. This diagram explains the Raspberry Pi board to communicate with the STM32 through USB communication. Python language is being used to establish the communication. Various libraries are used to access proper data from STM32. An USB connection is established between STM32 and Raspberry Pi. Raspberry Pi is coded to check for proper USB connection and to collect data from STM32. Exception handling is also taken care in the Raspberry Pi.

This model(Figure 6) shows about the final execution of the network. The implemented network on CAN is connected to STM32, that eventually transmits the data to Raspberry Pi using the USB communication. The Raspberry Pi uses MQTT broker to send data to the Thingsboard server. Finally the Raspberry Pi hosts the network to transmit the data to Thingsboard. The data is stored as a time-series data in Thingsboard telemetry database. This thingsboard gives an user friendly experience to analyse the accumulated data. Thingsboard provides a user friendly drag and drop dashboard design studio. The time-series data present in the database is mapped to each GUI elements on the dashboard. The data on the dashboard is updated every one second. If there is any connection failure between client and server, the last received data is displayed on the dashboard.

V. RESULTS

A. On Raspberry Pi OS

The result which is shown in figure 8 is running on the Raspberry Pi OS. It can be clearly said that all the desired data is being transmitted in the 10 byte format. When the Raspberry Pi is running the program, it also hosts the server and transmits the same data on to the thingsboard server.

![Fig. 8. Dashboard displaying values on Raspberry Pi](image)

B. On Thingsboard software

As shown in figure 9 the data is also being displayed on thingsboard user interface.

![Fig. 9. Dashboard to indicate on Thingsboard software](image)
The variables are displayed as:
1. Left wheel speed.
2. Differential speed.
4. Right wheel speed.
5. Voltage that is available in the battery.
These are the outputs that have been observed. It is also observed that the local host also receives the same data in the local area network. The host has to connect to the router and access the thingsboard page that will display the same data as shown in figure 9. It also gives a very aesthetic and user friendly dashboard. Vehicle’s diagnostic get much attention from industry and researchers in recent years.

The variety and heterogeneity of vehicle diagnostics implementation has been the major reason which makes it interesting. Here, This paper started with building the CAN bus and integrating STM32 MCU in the bus. So, that the MCU can sniff the data that has been published on the CAN bus by various connected devices. Later the same data is further communicated to Raspberry Pi through USB type of communication. The OS that has been running on the Raspberry Pi forwards these data to Thingsboard software and parallely hosts a local network. The hosted network transmits the data to all the local devices that are connected to it.

The results were observed in which the data was generated by various connected devices and transmitted to CAN bus. This data was being sniffed by the STM32 MCU and process it further to Raspberry Pi. Moving forward the Raspberry Pi was realized to run the server on its machine and parallely transmit the data to router. So, that the locally connected devices can access the information. As per the results acquired, we can reasonably infer that the data can be transmitted to and collected from the CAN Bus. This data can later be analysed for further transmission through other communication schemes. It is to be noted that the wireless communication can further be enhanced by implementing latest technologies in terms of enhancing the router capabilities, using better MCU’s for capturing data and using some of the IoT gateways to intelligently manage the routing to control large number of users.

VI. CONCLUSIONS

This paper integrated Bosch CAN spec 2.0, STM32 MCU, Raspberry Pi CPU with Adafruit OS and Thingsboard software to develop a CAN Bus that collects data from the bus and transmits further to make it wireless On-Board Diagnostics. This proposed paper make use of STM32 and Raspberry Pi for data collection and transmission. Here it can also be concluded that the data collection and acquisition model can be used for analysis of the accumulated data. The real-time vehicle data is being transmitted to Raspberry Pi CPU to make it display on to Thingsboard software. The display is also integrated on to the dashboard for real time view of the vehicle. This paper uses electric car for all the data collection and monitoring, since the future of the automobile industry is switching towards electric vehicles. Using electric cars make it more futuristic and makes it worth

in the present world scenario. The proposed system is very reliable on basis of practical grounds and it gives the best performance without any major glitches. For local hosting the server does not depend on internet and the process transitions is very smooth. This model can be very beneficial and useful on real-time applications.

REFERENCES