Proceedings of the 8th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'22)
Prague, Czech Republic - July 28- 30, 2022
Paper No. EEE 117
DOI: 10.11159/eee22.117

Remote Monitoring of Heavy-duty Equipment for Predictive Control

Mahdis Salehpoor[1], Mohammad Elsayyed[1], Witold Kinsner[1], Rhyse Maryniuk[3], Connor Fry Sykora[3], Kris Egilson[4], Leslie Funk[4] and Nariman Sepehri[2]
[1]Department of Electrical & Computer Engineering
[2] Department of Mechanical Engineering
University of Manitoba Winnipeg, MB, Canada
Salehpom@myumanitoba.ca; Elsayyem@myumanitoba.ca; Witold.Kinsner@umanitoba.ca; Nariman.Sepehri@umanitoba.ca;
[3]Audesse Inc.
151 Charles St, Kitchener, ON, Canada
Rhyse@audesseinc.com; connor@audesseinc.com
[4]Airport Technologies Inc.
110 Anson Street, Southport, MB, Canada
funk@atifirst.com; egilson@atifirst.com

Abstract – The *Controller Area Network* (CAN) bus is considered the backbone of many mobile machines. It is connected to and communicates with every single *Electronic Controller Unit* (ECU) in the vehicle via CAN data frames. With modern technology, the demand for effectively utilizing this vital system can heavily reduce costs for all vehicle industries, while improving safety. Tracking each vehicle's ECU status such as motor speed, fuel consumption, and engine temperature, while logging and analyzing the data could help in monitoring and ultimately predicting vehicle failures before they happen. Building upon the CAN bus architecture, this paper presents a systematic approach that employs the J1939 protocol, to form a high layer protocol to log and parse data from the bus using the industry-standard *Media Descriptor File* (MDF) file format and utilizing an *Internet of Things* (IoT) device in addition to cloud computing. The proposed system was implemented and tested on a heavy-duty industrial snowblower as used in airports. This new system is being used in two industrial companies because of its early promising results.

***Keywords*:** Controller Area Network (CAN), J1939 protocol, Cloud Computing, Media Descriptor File (MDF), Internet of things (IoT)

1. Introduction

The CAN bus is a message-based protocol designed to allow the vehicle's *Electronic Controller Units* (ECUs), as well as other devices to communicate with each other in a reliable, priority-driven fashion. The communications over the CAN bus are done via CAN frames [1].

The standard CAN frames have an 11-bit identifier as used in most cars. The extended 29-bit identifier frame is used in the J1939 protocol for heavy-duty vehicles. A J1939 CAN data logger provides data for diagnostics and disputes. Other applications of the J1939 CAN data logging include heavy-duty fleet telematics and predictive maintenance. Specifically, J1939 data from such vehicles can be used in fleet management to reduce costs and improve safety. In addition, monitoring vehicles and machinery via IoT CAN loggers could be used to predict and avoid their breakdowns [1].

Currently, several industrial companies have designed CAN data logger hardware and software systems. However, since only a few are IoT-oriented CAN loggers, this opens the floor to integrate a substantial number of features within the logger. In this project, the use of cloud computing enhances the system when working with vehicles with no proper power shutdown, by replacing on-edge computing with cloud computing. Furthermore, adding the capability of viewing the important vehicle information (e.g., engine temperature, speed, and oil levels) through an easy to interpret visual dashboard provides a better machine-human interface. This feature would not be possible without an IoT CAN data logger that uploads and processes data in frequent time intervals. This paper describes how such a system has been designed and used in our study.

Based on the methodologies used in our implementation, the system can be further improved to use the data gathered for predictive machine learning models. This would help maintain the machinery for a longer period which saves customer downtime and costs of upgrading the hardware frequently. Figure 1 shows an example of an industry dashboard based on data gathered from our designed system.

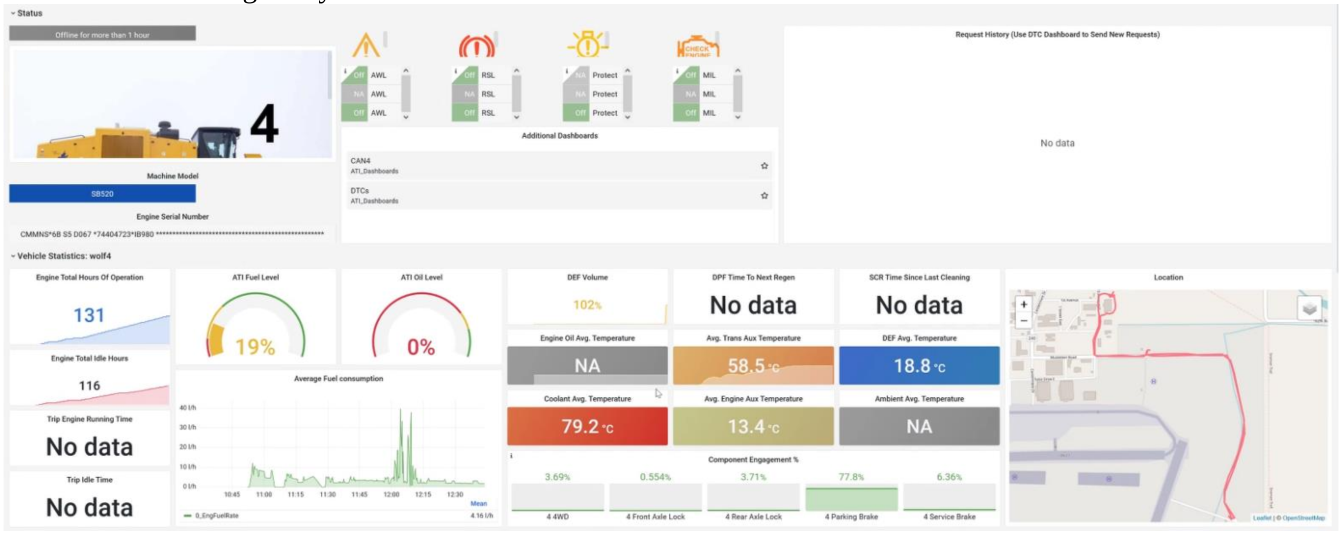


Fig. 1: ATI's Wolf SB-520 snowblower dashboard.

This paper describes the design, implementation, and testing phases completed during a research project, as defined for the collaborative effort with two industrial companies, *Airport Technologies Inc* (ATI) in Southport, Manitoba, and Audesse in Kitchener-Waterloo, Ontario.

2. System Architecture and Specifications

The system described in this paper is capable of logging CAN (J1939) data from the network and storing them in an online cloud for future analysis. Using an automated script on a server, the files stored can be downloaded and analyzed to detect any active diagnostic trouble codes (DTC). New files of the active DTCs can be created in 10-minute, or user-specified intervals.

The project has been divided into two phases based on its two main objectives. Phase one was the collection and parsing of J1939 CAN data using the industry-standard *Measurement Data Format* (MDF) to be able to view it on open source and free tools. Phase two was dedicated to the integration of the IoT J1939 *Diagnostic Trouble Codes* (DTCs) and cloud computing.

The overall system's hardware and software components included Python, Open Source ASAMMDF API, Audesse's FlexCase, and *Amazon Web Services* (AWS). Figure 2 shows the Wolf SB-520 snowblower, which is the vehicle used for the testing phase.

Fig. 2: Wolf SB-520 snowblower [2].

Figure 3 below shows the overall functionality of the system developed during this project. The system collects CAN data from the network and store them in an online cloud (i.e., an AWS S3 bucket). The data collection is done using the Audesse FlexCase as the hardware component and ASAMMDF API for data visualization. The device generates specific MDF files, specifically version 4 known as the MF4. Each time the vehicle (Wolf SB-520 snow blower) is turned on, the FlexCase starts by performing start-up routines (such as checking for local none uploaded files) and uploading them to the AWS server (i.e., S3 bucket), as well as deleting them to free up memory. The AWS LightSail server downloads new files from the bucket periodically for further analysis and data visualization.

The downloaded files from the S3 bucket could then be analyzed to detect any active DTC fault codes. For each active active DTC, a new file containing a +/– 10-minute interval of the active DTC can be generated. This step is an automated Python script that runs periodically (every 30 minutes) on the AWS LightSail server. Visualization and modeling of different 20-minute windows help understand the source of the fault code better. This also improves the efficiency of fixing the faulty system and predicting potential problems in the machinery.

The designed dynamic system helped to reduce customer downtime and minimize service calls by monitoring vehicle parameters and providing remote software updates when needed.

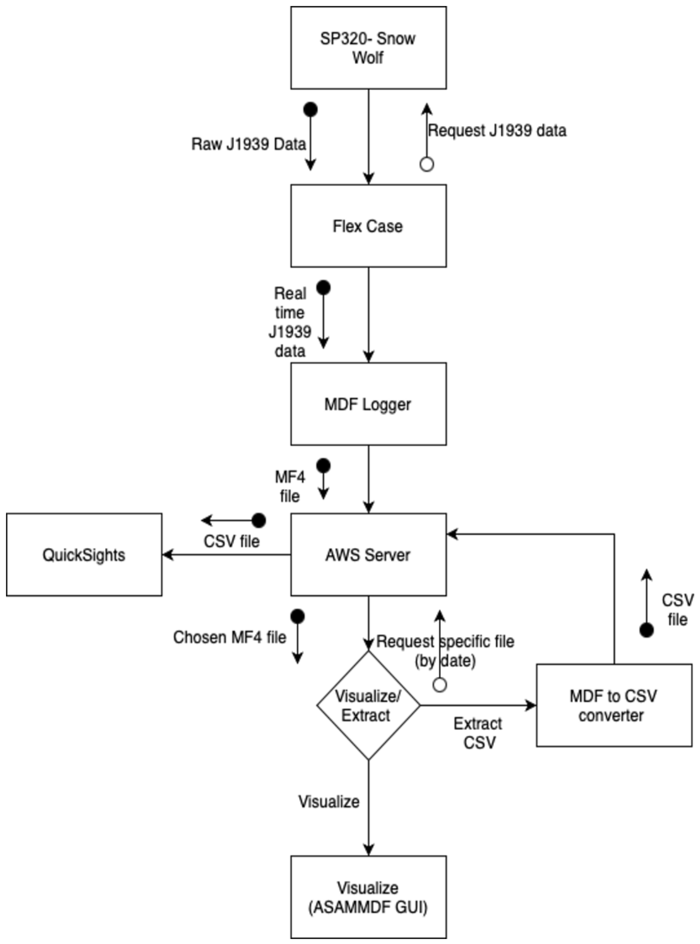


Fig. 3: System functions and interactions.

2. Phase One: Data Collection and Preprocessing

This phase was dedicated to the collection and parsing of J1939 CAN data. The collected data would then be plotted to provide visualization and a better understanding of the events that occurred during each power cycle. The ASAMMDF Graphical User Interface (GUI), also provided functionality to get data general statistics. This stage of the project was implemented and tested on one of ATI’s heavy-duty vehicles.

Figure 4, shows data statistics for the engine performance, vehicle speed, and gear histogram values during a data acquisition session on ATI's Wolf SB-520.

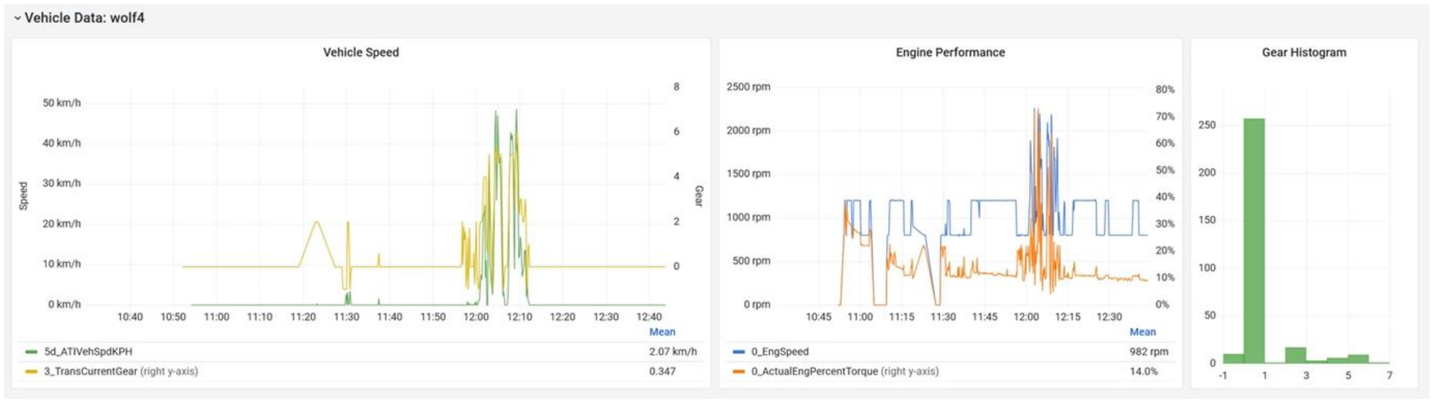


Fig. 4: Vehicle speed, engine performance and gear histogram for Wolf SB-520.

The data logging device used is the Audesse FlexCase. Based on Raspberry Pi 3 microprocessor with Ubuntu Linux distribution, which makes it highly versatile for different types of development. We wanted to have easy access J1939 data port to be directly connected to the vehicle, as well as a modem which is used in development stages and LTE capability which is used on-site.

2.1. Scope

Actively collect all required CAN data when needed and store it in a database in a time-stamped manner. This data will be used to troubleshoot/diagnose issues within the vehicle. The first step was to store the data in a *Comma-separated values* (CSV) file format. However, the CSV file format is not efficient for our design because (i) it requires much storage memory, and (ii) reading and writing into CSV files is slow. Therefore, the decision of adhering by the *Association for Standardization of Automation and Measuring Systems* (ASAM) to log in to the *Measurement Data Format* (MDF) files was taken.

The MF4 files have the five key benefits [3] including (i) interoperability since MDF is open, standardized & widely supported for CAN bus logging, ensuring interoperability across many tools, software, and APIs, (ii) fast reading and writing, (iii) highly memory efficient, (iv) Support both raw bus logging and physical values - hence you can often use the same software/APIs, and (v) The format is supported by free powerful software & APIs. Hence, easy for a *CAN Database* (DBC) to convert MF4 data to engineered values and export it to CSV, and MATLAB.

The main Python API used in this phase for data logging is the ASAMMDF API. Using this library CAN data was collected from the network. For each power cycle, MF4 files are generated in chunks based on a specific time interval. The files created are then uploaded to an online cloud for on-demand viewing. The MF4 files could be viewed as a table or plot in the ASAMMDF graphical user interface. Moreover, the ASAMMDF library is capable of extracting engineering values from the raw CAN data in the MF4 file when provided with a valid *Database CAN* DBC file.

2.2. Methodology

The FlexCase will utilize the following data collection system:

1. Record all time-stamped CAN messages in MF4 files on one or both available CAN buses. Each session (ignition on to ignition off) will be a single file on the server.

2. The code produced a separate file for a chosen time interval (e.g., every 5 minutes). Knowing that because there was no safe shutdown for the vehicle, we were operating on, so to minimize potential data loss and do more processing on the cloud side instead of the on-edge processing of the collected data frames.
3. At the end of each session, the MF4 will be stored on the edge machine.
4. The FlexCase will upload all available logs to the chosen cloud storage provider (e.g., AWS S3 bucket) on two occasions i) The next boot and ii) every 15 minutes or user-defined interval.
5. Each of the uploaded files is implicitly labeled as head and tails to ensure all separate files belonging to the same power cycle are compressed and concatenated together in one file, which is processed periodically on the cloud.

3. Phase Two

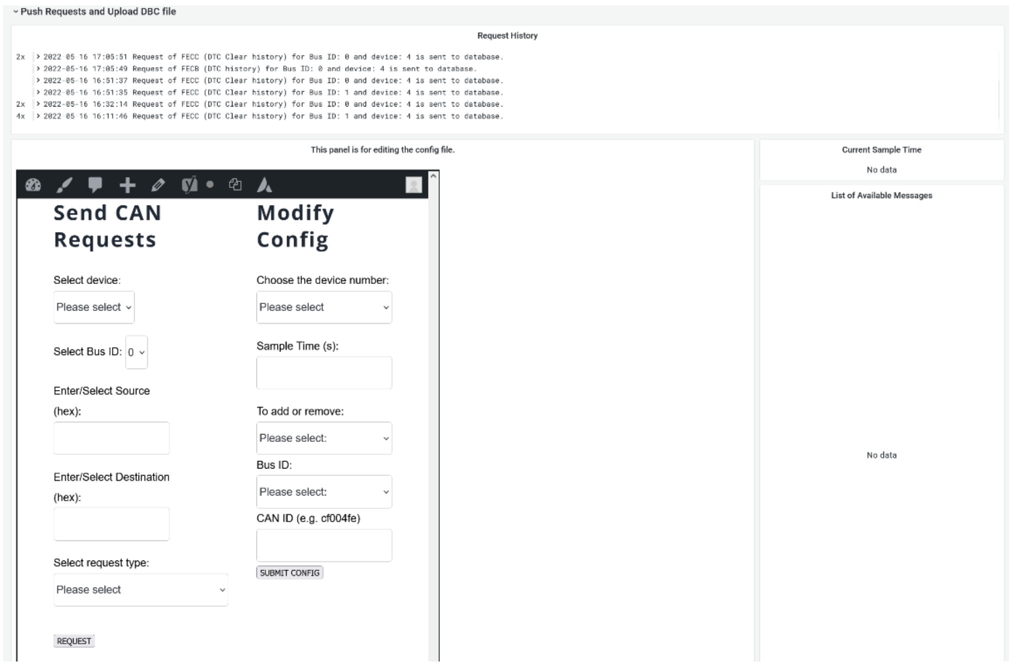
This phase was built on the completion of phase 1, which has already uploaded all available CAN data to the cloud. One of the first steps was to record various J1939 diagnostic trouble codes (DTC) faults that are broadcast on the J1939 network.

The designed implementation was to automate python scripts that would run on a cloud server (i.e AWS LightSail server) periodically and analyze the newly uploaded data to be able to detect active DTCs. Targeting a period of +/- 10 minutes from the instance the DTC detection aids in identifying the source of the problem as well as common failure patterns to potentially predict and omit similar failures.

3.1. Scope

An algorithm processes logged CAN data to identify the J1939 trouble codes and acts appropriately based on the DTC received. First, trigger the DTC flag bit to HIGH and highlight this as a trouble message and as an active DTC log. Then if the DTC corresponds to a Red Stop Lamp Status or any other DTC with a severe *Failure Mode Identifier* (FMI) (e.g., FMI = 0 or 1) an automatic email would be sent to the chosen email address as an urgent warning.

Moreover, Fig. 5 shows one of the unique features developed in the system which is a 2-way communication channel between the vehicle and Audesse’s FlexCase. This allows sending/receiving DTC commands to/from the vehicle concurrently. In addition to modifying data collection configuration (e.g. Sampling Rate, CAN bus ID,..etc) and defining vehicle-specific DTC custom messages.


Push Requests and Upload DBC file

Request History

```
2x  > 2022-05-16 17:05:51 Request of FECC (DTC Clear history) for Bus ID: 0 and device: 4 is sent to database.
    > 2022-05-16 17:05:49 Request of FECB (DTC history) for Bus ID: 0 and device: 4 is sent to database.
    > 2022-05-16 16:51:37 Request of FECC (DTC Clear history) for Bus ID: 0 and device: 4 is sent to database.
    > 2022-05-16 16:51:35 Request of FECC (DTC Clear history) for Bus ID: 1 and device: 4 is sent to database.
2x  > 2022-05-16 16:32:14 Request of FECC (DTC Clear history) for Bus ID: 0 and device: 4 is sent to database.
4x  > 2022-05-16 16:11:46 Request of FECC (DTC Clear history) for Bus ID: 1 and device: 4 is sent to database.
```

This panel is for editing the config file.

Fig. 5: Two-way communication channel Graphical User Interface (GUI).

3.2. Methodology

Multiple automated Python scripts on the cloud server will parse the uploaded CAN log and extract the active DTC messages from the CAN logs. These python scripts would run automatically every 30 minutes (or any chosen time interval). The script on the cloud server (e.g., AWS LightSail instance) fetches the uploaded files from the online storage (e.g., S3 bucket) and analyses them to produce DTC failure logs and upload them to the cloud. Specifically, it would upload the files including the +/- 10-minute interval of time-stamped active DTCs into the cloud storage.

In summary, the following conditions are applied:

1. If an active DTC is found in the logs:
   a. Identify the specific Suspect Parameter Number (SPN), FMI, and time stamp.
   b. Create a human-readable (engineering values) file that contains the requested CAN signal for 10 minutes on either side of DTC.
   c. Upload the newly generated files to the S3 bucket.
2. If no DTCs are found, do nothing.

4. Discussion of Findings and Future Work

Based on the experience gained from the project, several possible improvements to the currently existing system have been identified. One of the simplest would be to use the unsorted MF4 format for the MDF files, with variable CAN data frame lengths. We used the sorted MF4 files because the I/O library did not support the unsorted format [4]. The major advantage of the proposed modification is that writing to an unsorted MF4 file is faster and more memory efficient as compared to the sorted file.

Another challenge in this project is the sudden power loss during data acquisition because the vehicles in this project were not equipped with a safe shutdown system to warn the FlexCase, and it was not possible to add a backup power source. This led to an inevitable data loss when a power supply was lost. That is why we decided to log small-sized files for each power cycle, and utilize cloud computing to merge all files belonging to a single power cycle into a single file. This approach minimizes the data loss to the tolerable level (e.g., 5 minutes).

Several future improvements could include data hygiene, followed by several data processing steps such as (i) descriptive statistics, (ii) mono-scale time-domain and frequency-domain analyses, (iii) multiscale analysis, including wavelets, and (iv) polyscale analyses on correlated data with a short-term and long-term memory [5]. The conventional conventional statistical analysis of the data collected from the CAN network could include (a) moving zero crossing, (b) (b) moving turn count, (c) moving power, (d) moving average, (e) moving variance, (f) moving skewness, and (g) moving moving kurtosis, all done on the data frames within which the data can be considered stationary. The main goal of the analyses would be to design a smarter system that could predict upcoming operational problems, and design either manual or automated preventive maintenance of the machines.

5. Conclusions

This paper presents a novel approach to CAN-based data monitoring system. The main practical objective of the project was to improve the operation of a class of heavy equipment currently used by the industry for removing snow in critical environments. The combined data acquisition and feedback display system has been deployed in the field and is operational. The proposed design has been implemented and tested on the Wolf SB-520 snowblower often used at airports. The design considers various criteria to create a robust, easy-to-implement yet effective, and dynamic data logging and visualization system. The implementation makes use of both edge programming and cloud computing such as *Amazon Web Services* (AWS) to minimize potential data loss.

The system also provides a baseline design for future improvements, including data analysis using machine learning algorithms for effective prediction of machine failures and preventive maintenance. The system is also a candidate for adding new smart sensors with artificial intelligence at the edge.

References

[1] CSS Electronics, "Can bus explained - a simple intro [2022: The #1 tutorial]," CSS Electronics, 01-Nov-2021. [Online]. Available: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial. [Accessed: 19-May-2022].

[2] ATI, "Snow Wolf - 520," Airport Technologies Inc. [Online]. Available: https://atifirst.com/main/snow-wolf/. [Accessed: 19-May-2022].

[3] CSS Electronics, "Mf4 (MDF4) explained - a simple intro [+J1939/OBD2 data]," CSS Electronics, 01-Nov-2021. [Online]. Available: https://www.csselectronics.com/pages/mf4-mdf4-measurement-data-format. [Accessed: 19-May-2022].

[4] ASAM Wiki, "MDF details," [Online]. Available: https://www.asam.net/standards/detail/mdf/wiki. [Accessed: 19-May-2022].

[5] Witold Kinsner, *Fractal and Chaos Engineering: Monoscale, Multiscale and Polyscale Analyses*. Winnipeg, MB: OCO Research, Feb 2020, 1106 pages. {ISBN: 978-0-9939347-2-8, eBook}