DOI. 10.1109/ICCP63557.2024.10793030

Sending CAN Flexible Data-Rate Frames in an Automatic Manner to Improve Vehicle Diagnose Processes

Florian-Aurelian Krech, Cristina-Sorina Stângaciu, and Mihai V. Micea
Department of Computer and Information Technology,
Politehnica University of Timisoara,
2, V. Parvan Bvd., Timisoara, Romania
florian.krech@cs.upt.ro, cristina.stangaciu@cs.upt.ro, mihai.micea@cs.upt.ro

Abstract—**This paper provides an algorithm for the Controller Area Network (CAN) protocol, which enables transmission of CAN single frames and CAN flexible data-rate frames in an automatic manner. The automotive industry is of key importance nowadays and, in this context, our research focuses on autonomous/ automatic diagnosis and predictive maintenance techniques, which can also be easily extended to other important areas such as mechanical engineering, robotics, avionics, healthcare and so on. Our study shows improvements of up to 14% for validating and interpreting the data with respect to the response time of the ECU, which is not negligible for real-life and industrial applications, where time efficiency in detecting malfunctions is a sensitive issue.**

Keywords—CAN FD, automatic diagnosis, predictive maintenance, fault-tolerant systems, quality control.

I. Introduction

The driving force behind many of today vehicle innovations lies in the Electronic Control Units (ECUs). However, their increased complexity raises the potential for failures, underscoring the critical need for thorough functional testing to mitigate risks and uphold uncompromised quality. As the scope of tasks for test engineers expands with each innovation, the time, and resources available for completing these tasks are diminishing [1]. The only viable solution to this challenge is to streamline and accelerate the testing process.

A contemporary approach to dynamic testing and analysis encompasses the entire software development life cycle, particularly for high-quality software, such as that used in the automotive sector. Model-based testing emerges as a solution to address this complexity, offering a method to test software both during development and post-production to ensure its quality. Model-Based Testing (MBT) stands as a cornerstone in software testing, increasingly becoming essential in the automotive industry for future vehicle designs. This approach conducts software testing on a model, ensuring comprehensive, efficient, effective, and reliable testing of the product [2].

Intelligent manufacturing encompasses various application areas, one of which is fault diagnosis. A "fault" refers to any deviation from the expected, acceptable, or standard behavior within a system. It represents an undesired alteration or a tolerable malfunction that can impair overall system performance and potentially result in system failure. Fault diagnosis comprises fault detection, isolation (FDI), and occasionally identification processes. It serves as a valuable tool for monitoring the status of production lines, aiming to prevent or minimize financial losses associated with downtime [3]. Common faults encountered in production lines include incorrect sensor input/output port connections and component failures, such as motor malfunctions.

Predictive maintenance plays a crucial role in the automotive industry and holds significant importance. It has the potential to enhance both comfort and safety by enabling early detection, isolation, and prediction of potential failures [4]. Given the impressive progress in artificial intelligence technology, there is growing interest in a method for monitoring and predictive maintenance (PdM) of industrial equipment performance using production data. PdM is increasingly recognized as the most efficient solution for diagnosing equipment defects and assessing remaining lifespan within smart manufacturing and industrial big data platforms [5].

The motivation for this work started with the fact that nowadays CAN flexible data-rates are increasingly used in predictive maintenance and fault detection in automotive systems. We propose an algorithm which provides a solution to improving these processes in an automatic manner, by increasing the time efficiency with up to 14% in what concerns the response time of the ECU. Also, for the part of predictive maintenance, we will show that the malfunction of the system can be detected in at most 1 second after the initial request.

II. Related Work

Contemporary vehicles come furnished with approximately 100 ECUs, tasked with overseeing electrical systems to enhance both driving comfort and safety. For safe driving, it is imperative that ECUs are supported by a trustworthy communication network. Controller Area Network (CAN) emerged as a leading in-vehicle protocol, acclaimed for its array of benefits. These encompass superior immunity to electrical disturbances, streamlined wiring, self-diagnostic features, and swift error correction capabilities [6]. These qualities position the CAN bus as an ideal choice for the automotive sector. Employing the CAN bus showcases resilience in network setups, swiftly identifying faults while preserving system integrity [7]. Moreover, CAN functions as a message-centric protocol, ensuring efficient communication.

The CAN communication protocol adopts a CSMA/BA approach, with CSMA representing Carrier Sense Multiple Access. This means that every node within the network must first check for bus inactivity before attempting to transmit a message (Carrier Sense). CD, or Collision Detection, comes into play if two nodes initiate transmission simultaneously [8]. In such cases, the nodes identify the collision and take appropriate measures to manage it, thereby facilitating effective communication.

The conventional CAN protocol is constrained by its limited communication bandwidth, reaching up to 1 Mbps, and its payload size capped at 8 Bytes. This restricts its suitability for today's intricate automotive electrical/electronic systems. However, the emergence of CAN with flexible data rate (CAN-FD) marks a significant improvement. CAN-FD offers a higher communication bandwidth, scaling up to 8 Mbps for the payload, and boasts an increased payload size of up to 64 Bytes [9]. These enhancements broaden its applicability and accommodate the demands of modern automotive systems.

Some advantages and disadvantages of the CAN protocol are as follows [10]:

Advantages:

a. Reduces the number of wires, the potential for errors, and the overall weight
b. Centralized network: simplifies communication management
c. Flexible transmission: a message can be sent when the bus is free
d. Robustness: features reliable operation in challenging environments

Disadvantages:

a. Limited number of nodes: maximum limit is typically set at 64 nodes
b. Integrity issues: CAN systems may face integrity issues, leading to concerns regarding data accuracy and reliability
c. Limited message length and data amount: CAN imposes constraints on message length and the amount of data per message

Both the ECU and the tester need to communicate with each other on the CAN bus. The ECU should adhere to the following standards [11]:

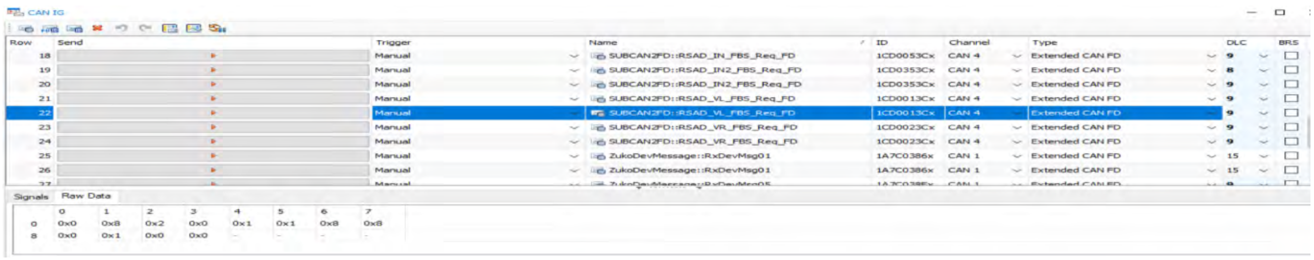
- Road vehicles – Unified Diagnostic Services (UDS): provides a comprehensive specification of diagnostic services and their parameters, ensuring compatibility and interoperability between different automotive diagnostic systems
- Road vehicles – Diagnostic on Controller Area Network (CAN): specifies the implementation of UDS on CAN, outlining the adaptation process to ensure seamless integration of diagnostic functionalities within the CAN communication protocol

The CAN protocol operates as a broadcast network, enabling all ECUs connected to the bus to receive the signals/messages transmitted. CAN messages come in two formats: the standard format and the extended format [12]. The extended CAN message format slightly differs from the standard CAN format due to the inclusion of an additional 18 bits in the arbitration field.

CANoe (Controller Area Network Open Environment), developed by Vector Private Limited, is a versatile development and testing tool primarily designed for the automotive industry. It facilitates the development, analysis, simulation, testing, and diagnosis of control units and the networks they form. With its extensive support for various communication protocols, including CAN, LIN, FlexRay, and Ethernet. CAPL (Communication Access Programming Language) is another dedicated software utilized in testing and environment simulation [13]. As an event-based program, CAPL offers functionalities such as timers, handling incoming/outgoing events, and events specific to CAN.

As automotive fault diagnosis technology advances, the associated criteria have evolved and become increasingly standardized. Presently, the diagnosis standards reach beyond identifying faults alone. They consider the entire vehicle life cycle, including development, testing, production, and post-sale maintenance phases. Furthermore, modern diagnosis systems have expanded their functionalities to include calibration, testing of ECUs, parameter measurement, and code upgrading. This comprehensive approach ensures efficient and effective management of vehicle systems throughout their lifecycle, enhancing overall performance and reliability [14] with the use of Vector Tool [15].

The existing solution for sending CAN Flexible Data-Rate Frames is to use the Iterative Generator provided by CANoe [16]. By using Iterative Generator each time, the user will need to do all the necessary steps manually to send data on a specific ID. For sending the data, the user should perform the following steps: first, the user should establish on which CAN bus and on which ID, the data should be sent. After establishing the CAN bus and the ID, the user should also set the data he wants to transmit according to the requirement. As a final step, the user should check on CAN trace if the data was sent accordingly and when the response

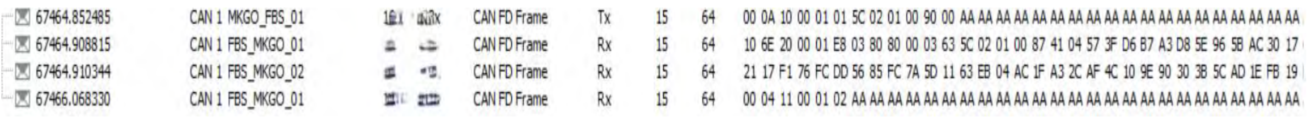


Row	Send	Trigger	Name	ID	Channel	Type	DLC	BRS
18		Manual	SUBCAN2FD::RSAD_IN_FBS_Req_FD	1CD0053Cx	CAN 4	Extended CAN FD	9	☐
19		Manual	SUBCAN2FD::RSAD_IN2_FBS_Req_FD	1CD0353Cx	CAN 4	Extended CAN FD	8	☐
20		Manual	SUBCAN2FD::RSAD_IN2_FBS_Req_FD	1CD0353Cx	CAN 4	Extended CAN FD	9	☐
21		Manual	SUBCAN2FD::RSAD_VL_FBS_Req_FD	1CD0013Cx	CAN 4	Extended CAN FD	9	☐
22		Manual	SUBCAN2FD::RSAD_VL_FBS_Req_FD	1CD0013Cx	CAN 4	Extended CAN FD	9	☐
23		Manual	SUBCAN2FD::RSAD_VR_FBS_Req_FD	1CD0023Cx	CAN 4	Extended CAN FD	9	☐
24		Manual	SUBCAN2FD::RSAD_VR_FBS_Req_FD	1CD0023Cx	CAN 4	Extended CAN FD	9	☐
25		Manual	ZukoDevMessage::RxDevMsg01	1A7C0386x	CAN 1	Extended CAN FD	15	☐
26		Manual	ZukoDevMessage::RxDevMsg01	1A7C0386x	CAN 1	Extended CAN FD	15	☐
27		Manual	ZukoDevMessage::RxDevMsg05	1A7C038Ex	CAN 1	Extended CAN FD	9	☐

Signals | Raw Data

	0	1	2	3	4	5	6	7
0	0x0	0x8	0x2	0x0	0x1	0x1	0x8	0x8
8	0x0	0x1	0x0	0x0	-	-	-	-

Figure 1. Sending a message through the Iterative Generator



```
67464.852485   CAN 1 MKGO_FBS_01   CAN FD Frame   Tx   15   64   00 0A 10 00 01 01 5C 02 01 00 90 00 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
67464.908815   CAN 1 FBS_MKGO_01   CAN FD Frame   Rx   15   64   10 6E 20 00 01 E8 03 80 80 00 03 63 5C 02 01 00 87 41 04 57 3F D6 B7 A3 D8 5E 96 5B AC 30 17
67464.910344   CAN 1 FBS_MKGO_02   CAN FD Frame   Rx   15   64   21 17 F1 76 FC DD 56 85 FC 7A 5D 11 63 EB 04 AC 1F A3 2C AF 4C 10 9E 90 30 3B 5C AD 1E FB 19
67466.068330   CAN 1 FBS_MKGO_01   CAN FD Frame   Rx   15   64   00 04 11 00 01 02 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
```

Figure 2. Overview of a CAN trace

is received, he should validate if the response received is according to the requirement. An overview of the existing solution, where the Iterative Generator is used, can be seen in Figure 1. For a data length code (DLC) smaller than 8, CANoe provides a data link library used for sending data, especially ISOTP frames.

Currently, there are some automation tools for sending CAN flexible data rates frames like DUT (Device Under Test) [17], or the Time Partition Testing tool, but some time efficiency and predictive maintenance issues can be seen in the literature [18], since when we are working with CAN flexible data-rates, the order of the messages received is trivial.

As seen in Figure 1, a message with the DLC equal with 9 is sent on Channel 4. The message, which has a total length of 12 bytes, can be seen on the raw data section of the capture.

In Figure 2, we can see an overview of the trace when sending a message via Iterative Generator. As can be seen, we sent a message with a data length code of 15, meaning 64 bytes message length, and we received 3 other messages from the ECU as a response to the message we sent, all of them having message length of 64 bytes.

III. Algorithm implementation

The algorithm we are proposing in this paper can be used in CAPL browser provided by CANoe.

To create the algorithm, we employed an iterative model research methodology. As a first iteration step, we started initially only by sending the frames in an automatic manner, without being necessary to use the Iterative Generator provided by CANoe and afterwards we performed some analysis to identify some advantages and disadvantages of the initial algorithm. After this step, we adjusted the algorithm to check for the response received from the ECU without needing a human intervention for validation. After this step, we have verified the robustness checks and assessed the algorithm performance across various conditions, in the end verifying if the algorithm achieves strong performance on unseen data and if it aligns with the objectives established from the very beginning.

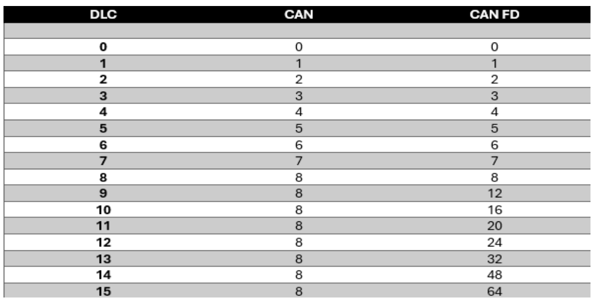
The proposed algorithm takes as input the following parameters:

- the ID we want to send the data to,
- the data we want to send on that specific ID,
- the ID we know the response should come,
- the data we expect to be received,
- a mask of the data that we expect for taking into account only the bytes we are interested in,
- the number of frames we are expecting since the maximum amount of data a frame contains is 64 bytes according to specialized literature
- and. finally, a timeout which will be used for waiting only a specific amount of time for a message.

The algorithm starts by constructing the frames we are waiting for, starting from the frame ID we give as a parameter for the response which represents the first frame we expect. The multi-frame construction starts by considering the number of frames we are giving as parameter, since it might be possible that our response might come on multiple frames, which will cover the case of multi-consecutive frame response. An example of multi consecutive frame response can be seen in Figure 2, where we are sending a single frame request and the response consists of 128 bytes, being the case where the frames are split since the maximum number of bytes a frame could contain is 64 bytes.

As a second step, we are transmitting the data via the specified ID. To transmit the data, we are considering the ID of the frame we want to send the data, the data length code, and the proper data. To do this, firstly we are establishing the protocol data unit of ID, the protocol data unit of the bit rate switch and the protocol data unit of the extended data length. If the data length code is smaller than 8, then we set the protocol data unit equal to the actual value of the data length code, otherwise we are setting the protocol data unit based on the following rule Vector provides [19].

An overview of the data length code, with respect to message length, can be seen in Figure 3.



DLC	CAN	CAN FD
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	8	12
10	8	16
11	8	20
12	8	24
13	8	32
14	8	48
15	8	64

Figure 3. DLC with respect to data field bytes.

When we want to transmit the data, we are also validating the input. For example, if we are setting the data length code (DLC) to 9 and the message length is different from 12, then we are raising the error of a wrong input DLC used. The input validation is computed based on the data that can be seen in Figure 3.

We also considered the case when we need to send some consecutive frames. In that case, if we are sending a message with data length code greater than 64, we are splitting the message. If we want to send a message with a length of 128, we are splitting the message into 2 parts, the first part containing a 64 bytes message length, followed by a frame increased by 1, with the rest of the bytes we want to send the message.

For the part of response, initially, we are using the predefined function TestWaitForAllJoinedEvents which waits for the current set of joined events.

Since there is a chance of working with multiple frames, we are considering the number of frames the response should contain and we are performing the same steps for each frame.

As a first step, we have used the predefined function testGetWaitEventMsgData, which calls up the message content.

Once a message is received, we start decrementing the number of frames we are expecting each time, as can be seen in Figure 4.

If a response to the initial request was received, we start to analyze the response considering the ID via which the response was received, the data length code and the proper data received. If the data length code is also greater than 8, then we apply the same procedure as in the previously mentioned case, where we send the data, accordingly, based on the correspondence between the data length code and the data filled bytes.

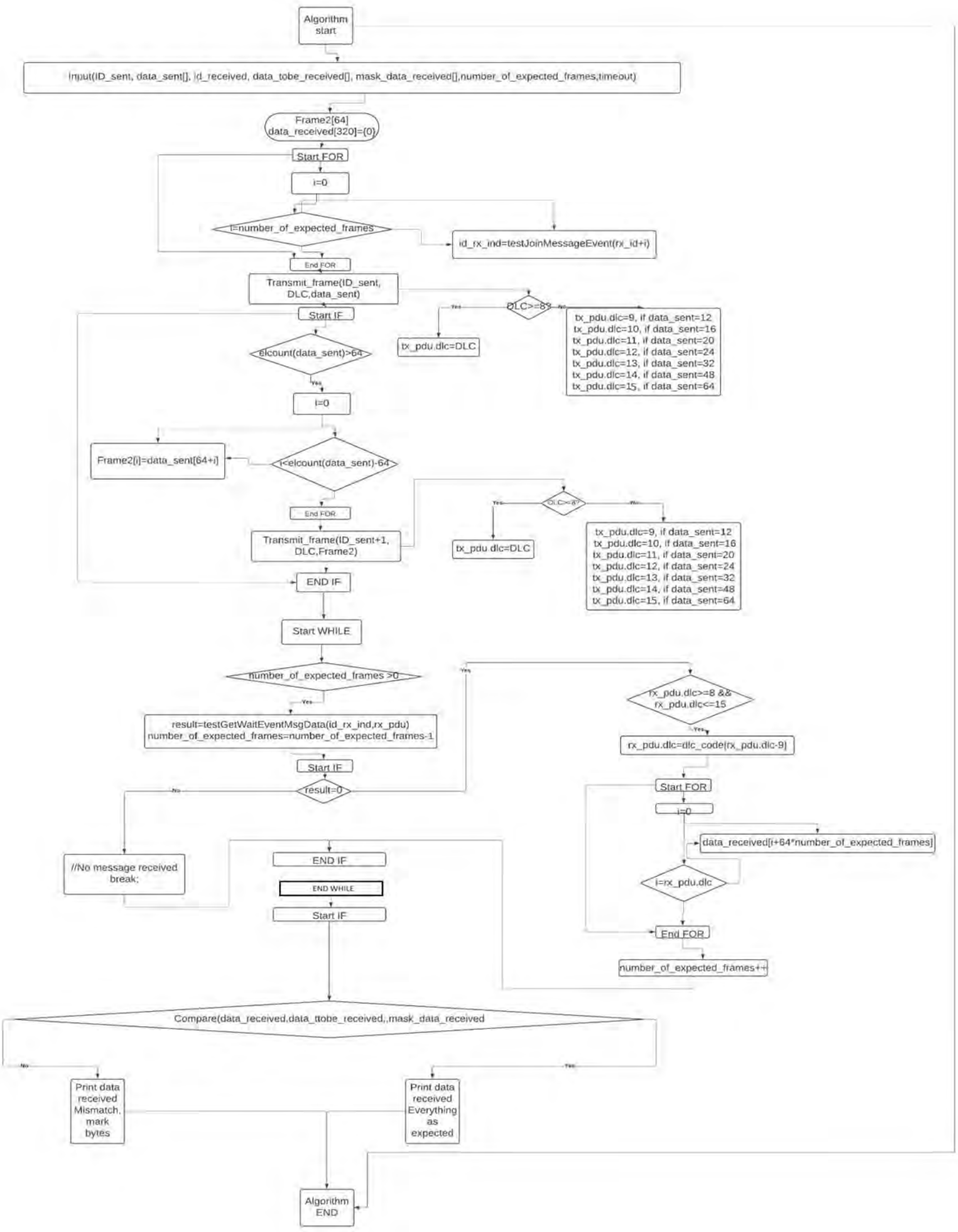

Algorithm start
Input(ID_sent, data_sent[], id_received, data_tobe_received[], mask_data_received[],number_of_expected_frames,timeout)
Frame2[64]
data_received[320]={0}
Start FOR
i=0
i=number_of_expected_frames
id_rx_ind=testJoinMessageEvent(rx_id+i)
End FOR
Transmit_frame(ID_sent, DLC,data_sent)
DLC>=8?
Yes
No
tx_pdu.dlc=DLC
tx_pdu.dlc=9, if data_sent=12
tx_pdu.dlc=10, if data_sent=16
tx_pdu.dlc=11, if data_sent=20
tx_pdu.dlc=12, if data_sent=24
tx_pdu.dlc=13, if data_sent=32
tx_pdu.dlc=14, if data_sent=48
tx_pdu.dlc=15, if data_sent=64
Start IF
elcount(data_sent)>64
Yes
i=0
Frame2[i]=data_sent[64+i]
i<elcount(data_sent)-64
End FOR
Transmit_frame(ID_sent+1, DLC,Frame2)
DLC>=8?
Yes
No
tx_pdu.dlc=DLC
tx_pdu.dlc=9, if data_sent=12
tx_pdu.dlc=10, if data_sent=16
tx_pdu.dlc=11, if data_sent=20
tx_pdu.dlc=12, if data_sent=24
tx_pdu.dlc=13, if data_sent=32
tx_pdu.dlc=14, if data_sent=48
tx_pdu.dlc=15, if data_sent=64
END IF
Start WHILE
number_of_expected_frames >0
Yes
result=testGetWaitEventMsgData(id_rx_ind,rx_pdu)
number_of_expected_frames=number_of_expected_frames-1
Start IF
result=0
No
Yes
rx_pdu.dlc>=8 &&
rx_pdu.dlc<=15
Yes
rx_pdu.dlc=dlc_code[rx_pdu.dlc-9]
Start FOR
i=0
data_received[i+64*number_of_expected_frames]
i=rx_pdu.dlc
End FOR
number_of_expected_frames++
//No message received
break;
END IF
END WHILE
Start IF
Compare(data_received,data_ttobe_received,,mask_data_received
No
Yes
Print data received Mismatch, mark bytes
Print data received Everything as expected
Algorithm END

Figure 4. Overview of the algorithm

The response received is stored and compared to the bytes we are expecting, considering the byte array with the expected response we are giving as input and the mask with the corresponding bytes that should be the same. If the message is the same as the one, we are expecting, then there will be a passed test case, otherwise the test will fail, by mentioning that the response is not received as expected, also given as input. An overview of the entire algorithm can be seen in Figure 4.

IV. MAIN RESULTS OBTAINED

Compared to the method provided by Vector, our algorithm is sending and receiving the response with an improvement of around 14% for multi-frame messages, in what concerns the interpretation and validation of the data with respect to the responding time of the ECU. Another important factor we considered, is the order of the messages that are received since we are solving the issue of multi-frames messages, and based on this factor, an improvement of at least 50% in terms of accuracy and efficiency is remarked as depicted in Figure 6, where it can be seen that if an unexpected message is received, the algorithm will detect the deviations from the specifications and will show the malfunction of the system, compared to other tools used during research, where an issue encountered by using Device Under Test tool was that an unexpected flow control from the ECU was corrupting the test results. The response is evaluated as can be seen in Figure 5 and Figure 6.

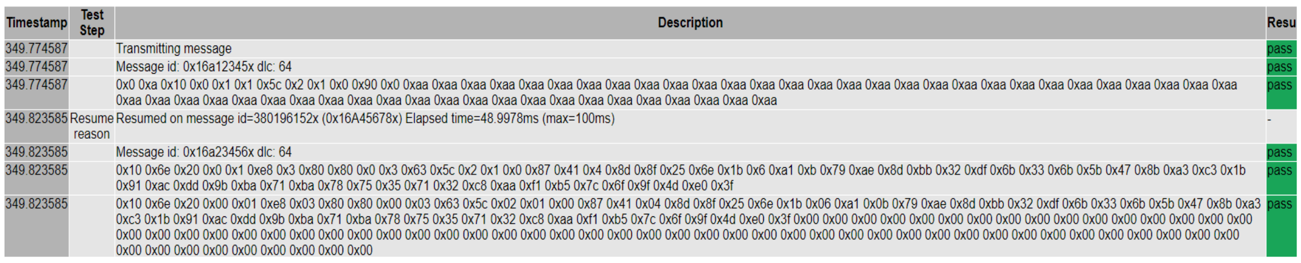
As can be seen in Figure 5, the test report generated contains all the data we are interested in, like the ID used for sending the data, the data length code and the data we want to send according to the requirement. For the response message, we can see that is automatically retrieved and stored in the test report, taking into account also the ID on which the frame was received, the data length code, the data received and also the byte array we are expected and the mask of the interested bytes, since, as we can see in Figure 5, the test report is passed. The data length code used in Figure 5 for testing the algorithm is 15.

Figure 6 illustrates a failed test report due to a timeout event. To accommodate such cases, our algorithm incorporates an additional 1-second buffer extension to ensure thoroughness. This extension allows for a comprehensive check to confirm if a message is received and if so, whether it aligns with the anticipated transmission. However, if there occurs a system malfunctions, it will be flagged and indicated in the report.

The algorithm was tested on hundreds of datasets, and each time our algorithm showed an improvement of at least 12% in what concerns the responding time of the ECU to validate the expected response. The tests performed contains only multi-consecutive frames, where Table 1 shows the necessary time to send and validate the expected response. The timing difference in Table 1 consists of the task involved in generating the answer.

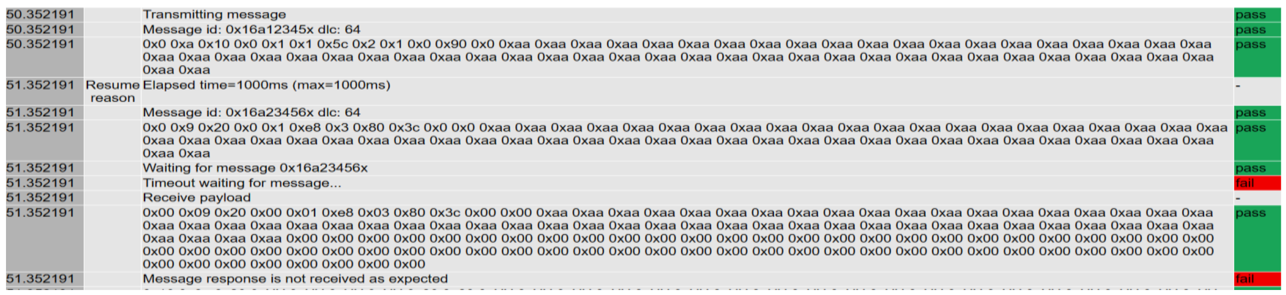
Initial procedure (ms)	**Using this algorithm (ms)**	**Time difference (%)**
0.056357	**0.048998**	**13.05%**
0.071958	**0.060531**	**15.88%**
0.068742	**0.060312**	**12.25%**
0.097856	**0.081372**	**16.84%**
0.082354	**0.071523**	**13.15%**
0.078936	**0.067639**	**14.31%**
0.099546	**0.087142**	**12.45%**
0.107146	**0.090675**	**15.37%**

Table 1. Time measurement for different scenarios



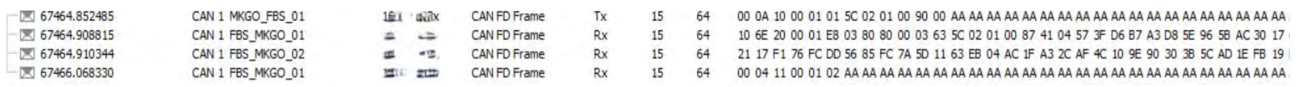
Timestamp	Test Step	Description	Resu
349.774587		Transmitting message	pass
349.774587		Message id: 0x16a12345x dlc: 64	pass
349.774587		0x0 0xa 0x10 0x0 0x1 0x1 0x5c 0x2 0x1 0x0 0x90 0x0 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa	pass
349.823585	Resume reason	Resumed on message id=380196152x (0x16A45678x) Elapsed time=48.9978ms (max=100ms)	-
349.823585		Message id: 0x16a23456x dlc: 64	pass
349.823585		0x10 0x6e 0x20 0x0 0x1 0xe8 0x3 0x80 0x80 0x0 0x3 0x63 0x5c 0x2 0x1 0x0 0x87 0x41 0x4 0x8d 0x8f 0x25 0x6e 0x1b 0x6 0xa1 0xb 0x79 0xae 0x8d 0xbb 0x32 0xdf 0x6b 0x33 0x6b 0x5b 0x47 0x8b 0xa3 0xc3 0x1b 0x91 0xac 0xdd 0x9b 0xba 0x71 0xba 0x78 0x75 0x35 0x71 0x32 0xc8 0xaa 0xf1 0xb5 0x7c 0x6f 0x9f 0x4d 0xe0 0x3f	pass
349.823585		0x10 0x6e 0x20 0x00 0x01 0xe8 0x03 0x80 0x80 0x00 0x03 0x63 0x5c 0x02 0x01 0x00 0x87 0x41 0x04 0x8d 0x8f 0x25 0x6e 0x1b 0x06 0xa1 0x0b 0x79 0xae 0x8d 0xbb 0x32 0xdf 0x6b 0x33 0x6b 0x5b 0x47 0x8b 0xa3 0xc3 0x1b 0x91 0xac 0xdd 0x9b 0xba 0x71 0xba 0x78 0x75 0x35 0x71 0x32 0xc8 0xaa 0xf1 0xb5 0x7c 0x6f 0x9f 0x4d 0xe0 0x3f 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	pass

Figure 5. Overview of a passed test report.



50.352191		Transmitting message	pass
50.352191		Message id: 0x16a12345x dlc: 64	pass
50.352191		0x0 0xa 0x10 0x0 0x1 0x1 0x5c 0x2 0x1 0x0 0x90 0x0 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa	pass
51.352191	Resume reason	Elapsed time=1000ms (max=1000ms)	-
51.352191		Message id: 0x16a23456x dlc: 64	pass
51.352191		0x0 0x9 0x20 0x0 0x1 0xe8 0x3 0x80 0x3c 0x0 0x0 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa	pass
51.352191		Waiting for message 0x16a23456x	pass
51.352191		Timeout waiting for message...	fail
51.352191		Receive payload	-
51.352191		0x00 0x09 0x20 0x00 0x01 0xe8 0x03 0x80 0x3c 0x00 0x00 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0xaa 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00	pass
51.352191		Message response is not received as expected	fail

Figure 6. Overview of a failed test report



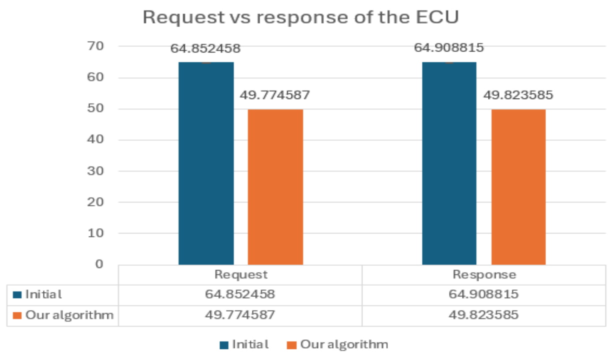
```
67464.852485   CAN 1  MKGO_FBS_01   CAN FD Frame   Tx   15   64   00 0A 10 00 01 01 5C 02 01 00 90 00 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
67464.908815   CAN 1  FBS_MKGO_01   CAN FD Frame   Rx   15   64   10 6E 20 00 01 E8 03 80 80 00 03 63 5C 02 01 00 87 41 04 57 3F D6 B7 A3 D8 5E 96 5B AC 30 17
67464.910344   CAN 1  FBS_MKGO_02   CAN FD Frame   Rx   15   64   21 17 F1 76 FC DD 56 85 FC 7A 5D 11 63 EB 04 AC 1F A3 2C AF 4C 10 9E 90 30 3B 5C AD 1E FB 19
67466.068330   CAN 1  FBS_MKGO_01   CAN FD Frame   Rx   15   64   00 04 11 00 01 02 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
```

Figure 7. Overview of the CAN trace

As another advantage of the algorithm is the fact that the message is also available on the trace, like in the case of Iterative Generator, to have a traceability of the CAN messages. An overview of the message presented in Figure 5, can be seen on trace in Figure 7.

In terms of automation, the response will be automatically generated in the test report and if the response received is different of the response we are waiting for from the very beginning, then we can deduce automatically that something is wrong without being necessarily to check the response by a human, like in the case of initial algorithm.

As can be seen in Figure 8, the time difference between the moment when we are transmitting the request versus the time, we are receiving the response is 0.05633 seconds in the case we are using the existing method, while, by using our algorithm, the time difference between the moment we are sending versus the time difference between the moment we are receiving is 0.048998 seconds (see Figure 5). A graphical representation of the data measured is depicted in Figure 8.



	Request	Response
■ Initial	64.852458	64.908815
■ Our algorithm	49.774587	49.823585

Figure 8. Request vs response of the ECU in both cases

Figure 8 shows that the algorithm we propose does not affect the time performance of the ECU, since we received the response 0.007332 seconds faster than with the initial method. In Figure 8, we represented just a positive case when we are validating our algorithm for a specific scenario. Other tests were performed, and we found that each time, the proposed algorithm was better in terms of time efficiency compared to Device Under Test tool or Iterative Generator, as seen in Figure 9. The tests performed were positive cases, where we received the response from the ECU as expected, in the amount of time according to the requirement.

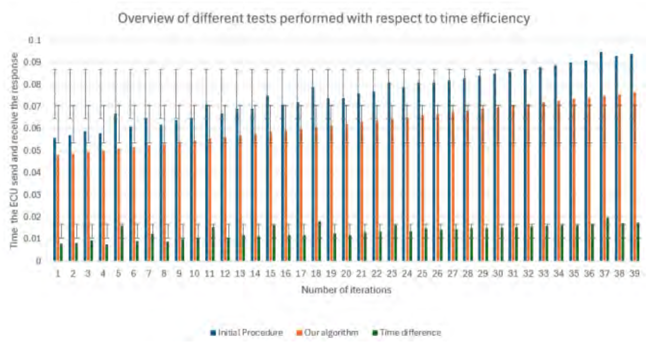
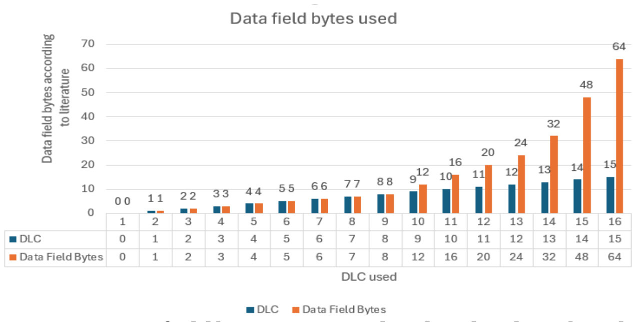


Figure 9. Overview of different tests performed.

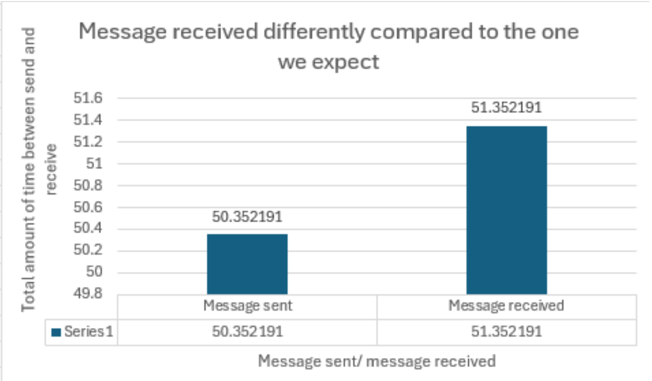
Since this algorithm works with extended CAN frames, an overview of the data length code that we considered with respect to data field bytes can be seen in Figure 10. The abscissa axis represents the data length code used for CAN protocol, while the ordinate axis, represents the number of bytes a message should have with respect to the data length code.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
■ DLC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■ Data Field Bytes	0	1	2	3	4	5	6	7	8	12	16	20	24	32	48	64

Figure 10. Data field bytes we considered vs data length code.

There are some cases when we are receiving a different message compared to the one, we are expecting. In this case, the malfunction of the system is also detected, with a time difference of 1 second, as can be seen in Figure 11. The time we are waiting for the response to be received is as mentioned in the requirement. An example of such a case, when a different response compared to the one, we expect, can be seen in Figure 6.



	Message sent	Message received
■ Series1	50.352191	51.352191

Figure 11. Message not received as expected.

V. Conclusions

In this paper, an algorithm for automatically transmission of single frames vs multi frames was developed, for diagnosing automotive systems over the CAN busses, using the Vector CANoe environment.

Our algorithm improves the automatic diagnose efficiency by streamlining the process, reducing the costs, and optimizing the resources to achieve better results.

Since we also considered the important factor of dealing with multi-frame cases, more exactly the order of the messages received from the ECU, the improvements brought are quality control, for detecting deviations from specifications, and predictive maintenance, since we are minimizing the downtimes.

As another, improvement, our study shows differences of about 14% for interpretation and validation of data considering the responding time of the ECU, which is not negligible for the automotive industry, where time efficiency in detecting malfunctions is a sensitive issue.

The algorithm scales efficiently, maintaining consistent performance as the dataset size grows. Test with hundreds of datasets exhibits only a slight increase in the processing time, validating the data, and indicating strong scalability. The algorithm complexity is O(n) since it grows linearly with the size of the input.

REFERENCES

[1] Adrian Bogorin-Predescu, Aurel Mihail Țîțu, Nicoleta–Mădălina Niță, Victor-Leonard Domnariu, "MODELING OF THE AUTOMATIC TESTING PROCESS OF ELECTRONIC CONTROL UNITS IN THE AUTOMOTIVE INDUSTRY", International Journal of Mechatronics and Applied Mechanics, 2022

[2] M. A. Khan, A. Jadoon, K. M. S. Haq, S. Mumtaz and J. Rodrigues, "An Overview of Resilient and Automatic Model-Based Testing Approaches for Automotive Industry," *2019 IEEE International Conference on Communications Workshops*

[3] Liyu Wang, Jack Hodges, Dan Yu, Ronald S. Fearing, "Automatic modelling and fault diagnosis of car production lines based on first-principle qualitative mechanics and semantic web technology", Advanced Engineering Informatics 2021

[4] N. Soltanipour, S. Rahrovani, J. Martinsson and R. Westlund, "Chassis Hardware Fault Diagnostics with Hidden Markov Model Based Clustering," *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*

[5] R. H. Kim, S. H. Oh and J. G. Kim, "Development of DNN-based diagnostic platform for self-diagnosis of electric vehicle automatic transmission controller production equipment," *2022 IEEE Ninth International Conference on Communications and Electronics (ICCE)*

[6] M. Bozdal, M. Samie and I. Jennions, "A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions," *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, Southend, UK

[7] S. G. Patil and V. R. Ratnaparkhe, "CAN Protocol–Application in Automation Electronics," *2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC)*, Aurangabad, India, 2020

[8] R. Khamamkar, A. Jadhav, S. Thoke and G. Chaple, "Smart Vehicle Safety Monitoring System Using CAN Protocol," *2018 IEEE Punecon*, Pune, India, 2018

[9] R. De Andrade, K. N. Hodel, J. F. Justo, A. M. Laganá, M. M. Santos and Z. Gu, "Analytical and Experimental Performance Evaluations of CAN-FD Bus," in *IEEE Access*, vol. 6, pp. 21287-21295, 2018

[10] Spencer G, Mateus F, Torres P, Dionísio R, Martins R. Design of CAN Bus Communication Interfaces for Forestry Machines. *Computers*. 2021; 10(11):144. https://doi.org/10.3390/computers10110144

[11] M. Wajape and N. B. Elamana, "Study of ISO 14229-1 and ISO 15765-3 and implementation in EMS ECU for EEPROM for UDS application," *2014 IEEE International Conference on Vehicular Electronics and Safety*, Hyderabad, India, 2014

[12] Oladimeji D, Rasheed A, Varol C, Baza M, Alshahrani H, Baz A. CANAttack: Assessing Vulnerabilities within Controller Area Network. *Sensors*. 2023; 23(19):8223. https://doi.org/10.3390/s23198223

[13] D. Georgescu and L. Stanciu, "Designing and Implementing a Solution to Manipulate Signals in Automated Testing Using CANoe," *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2018

[14] R. G. Lazar and C. F. Caruntu, "Simulator for the Automotive Diagnosis System on CAN using Vector CANoe Environment," *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, Sinaia, Romania, 2020

[15] Vector Informatik GmbH, CANoe - User Manual, 7th ed., Vector, Stuttgart, Germany

[16] Vector Informatik GmbH, CANoe - CANoe Product Information

[17] V. Gajul, J. K. D. Mishra and S. Tavhare, "Automation solution for Software Testing of CAN based ECUs," *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Erode, India, 2021

[18] D. E. McFeely, "The process and challenges of a high-speed DUT board project," *Proceedings. International Test Conference*, Baltimore, MD, USA, 2002

[19] Vector Informatik GmbH, CANoe – Introduction to CAN