

FEDERAL UNIVERSITY OF TECHNOLOGY – PARANÁ
DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN COMPUTER SCIENCE

EDUARDO MICHAILU MENDES

**CONTRIBUTIONS IN CAN-BASED SYSTEMS:
PROTOTYPES AND TESTS**

PONTA GROSSA

2019

EDUARDO MICHAILU MENDES

**CONTRIBUTIONS IN CAN-BASED SYSTEMS:
PROTOTYPES AND TESTS**

Thesis presented as a partial requirement to obtain the title of Master in Computer Science from the Graduate Program in Computer Science of the Federal University of Technology – Paraná – Campus Ponta Grossa.

Concentration Area: Computing Systems and Methods

Advisor: Prof. Dr. Max Mauro Dias Santos

PONTA GROSSA

2019

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa
n.10/20

M538 Mendes, Eduardo Michailu

Contributions in can-based systems: prototypes and tests. / Eduardo Michailu
Mendes, 2019.

88 f. : il. ; 30 cm.

Orientador: Prof. Dr. Max Mauro Dias Santos

Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação
em Ciência da Computação. Universidade Tecnológica Federal do Paraná, Ponta
Grossa, 2019.

1. Microcontroladores. 2. Rede CAN (Rede de computador). 3. Sinais e símbolos.
4. Arduino (Controlador programável). 5. Sistemas de transmissão de dados. I. Santos,
Max Mauro Dias. II. Universidade Tecnológica Federaldo Paraná. III. Título.

CDD 004



Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ CÂMPUS PONTA GROSSA
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Ciência da Computação



FOLHA DE APROVAÇÃO

Título de Dissertação Nº 16/2019

CONTRIBUTIONS IN CAN-BASED SYSTEMS: PROTOTYPES AND TESTS

Por

Eduardo Michailu Mendes

Esta dissertação foi apresentada às **8 horas de 31 de outubro de 2019**, na **V1-001**, como requisito parcial para a obtenção do título de MESTRE EM CIÊNCIA DA COMPUTAÇÃO, Programa de Pós-Graduação em Ciência da Computação. O candidato foi arguido pela Banca Examinadora, composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Prof. Dr. Divanilson Rodrigo de Sousa Campelo (UFPE)

Prof. Dr. Maurício Zadra Pacheco (UEPG)

Prof. Dr. Augusto Foronda (UTFPR)

Prof. Dr. Lourival Aparecido de Gois (UTFPR)

Prof. Dr. Nome (UTFPR)
Orientador e presidente da banca



Visto do Coordenador:

Prof. Dr. André Koscianski
Coordenador do PPGCC
UTFPR – Câmpus Ponta Grossa

ABSTRACT

MENDES, Eduardo Michailu. **Contributions in CAN-based systems: prototypes and tests**. 2019. Total pages: 93. Thesis (Master in Computer Science) – Federal University of Technology - Paraná. Ponta Grossa, 2019.

The CAN network is used to connect microcontrollers exclusively through a metal bus and broadcast messages. In versions 2.0A and 2.0B the CAN frame payload carries up to 8 bytes of data. To send larger data it is necessary to use more than one frame. This paper presents a CAN network prototype using six Arduino cards that generate and consume different types of signals connected through the MCP2515 CAN module. The library used to enable the MCP2515 CAN module organizes the 8 bytes of CAN frame payload into an 8-position array, each with 1 byte. Taking advantage of this feature, besides demonstrating how to use this device, some proposals for improvement in the CAN network will also be presented. The first of these is a form of treatment of periodic and aperiodic signs. CAN frame and communication bus utilization has been optimized by sending multiple signals within a single frame, carrying one signal at each position of the vector. Mapping values to intervals compatible with the available space at each vector position allowed the transport of values larger than 8 bits within the vector positions. Transmission latency was calculated using the `mills()` method and a CAN gateway was implemented to reduce the broadcast domain of messages.

Keywords: Microcontroller. CAN Network. Periodic and aperiodic signs. Signal mapping. Bus optimization.

RESUMO

MENDES, Eduardo Michailu. **Contribuições em sistemas baseados em CAN: protótipos e testes**. 2019. Total de páginas: 93. Dissertação (Mestrado em Ciência da Computação) – Universidade Tecnológica Federal - Paraná. Ponta Grossa, 2019.

A rede CAN é usada para conectar exclusivamente microcontroladores através de um barramento metálico e mensagens em *broadcast*. Nas versões 2.0A e 2.0B, a carga útil do quadro CAN carrega até 8 bytes de dados. Para enviar dados maiores, é necessário usar mais de um quadro. Este artigo apresenta um protótipo de rede CAN usando seis placas Arduino que geram e consomem diferentes tipos de sinais conectados através do módulo CAN MCP2515. A biblioteca usada para ativar o módulo CAN MCP2515 organiza os 8 bytes de carga útil do quadro CAN em uma matriz de 8 posições, cada uma com 1 byte. Aproveitando esse recurso, além de demonstrar como usar este dispositivo, também serão apresentadas algumas propostas de melhoria na rede CAN. O primeiro deles é uma forma de tratamento de sinais periódicos e aperiódicos. A utilização do quadro CAN e do barramento de comunicação foi otimizada enviando vários sinais em um único quadro, transportando um sinal em cada posição do vetor. O mapeamento de valores para intervalos compatíveis com o espaço disponível em cada posição do vetor permitiu o transporte de valores maiores que 8 bits dentro das posições do vetor. A latência de transmissão foi calculada usando o método *mills()* e um *gateway* CAN foi implementado para reduzir o domínio de *broadcast* para a transmissão das mensagens.

Keywords: Microcontroladores. Rede CAN. Sinais periódicos e aperiódicos. Mapeamento de sinal. Otimização de barramento.

LIST OF FIGURES

Figure 1 - FlexRay passive bus topology.....	9
Figure 2 – FlexRay dual channel single star configuration	9
Figure 3 – FlexRay single channel cascaded star configuration.....	10
Figure 4 - FlexRay single channel hybrid topology.....	10
Figure 5 - Embedded System.....	18
Figure 6 - OSI Reference Model and TCP / IP Fact Model.....	24
Figure 7 - OSI layers x Classical CAN.....	28
Figure 8 - Frame CAN 2.0A.....	30
Figure 9 - Frame CAN 2.0B.....	31
Figure 10 - Frame CAN 2.0B vs CAN-FD.....	31
Figure 11 - Frame CAN 2.0B vs CAN-FD.....	34
Figure 12 - CAN-FD Frame	35
Figure 13 - Block Diagram MCP2515 CAN Controller.....	38
Figure 14 - Example system implementation.....	38
Figure 15 – Module MCP2515.....	39
Figure 16 - Scenario 1	41
Figure 17 - Frame transmission.....	43
Figure 18 - Frame reception.....	44
Figure 19 - Scenario 2.....	49
Figure 20 - OSI Model x implemented scenario	50
Figure 21 - Experiment picture	50
Figure 22 - Node 1	52
Figure 23 - Node 2	53
Figure 24 - Frame data field mount flowchart.....	55
Figure 25 - Frame 0x43 with multiple signals	56
Figure 26 - Node 3	57
Figure 27 - Node 4	58
Figure 28 - Node 5	59
Figure 29 - Graphic AD conversion	62
Figure 30 - Graphic Conversion to PWM signal	63
Figure 31 - Graphic Engine speed relative to PWM	64
Figure 32 - Graphic Comparation between temperature obtained and calculated.....	66
Figure 33 - Graphic Comparation between temperature and value resulting from ad conversion.....	67
Figure 34 - Graphic Mapped signal	70
Figure 35 - Scenario 3.....	74
Figure 36 - Scenario 3.....	75
Figure 37 - Gateway CAN	76
Figure 38 - Controllers and signals.....	77
Figure 39 - Temporal analysis of the transmission of signals 0x47 and 0x48.....	78
Figure 40 - Scenario 4.....	80

SUMMARY

1 INTRODUCTION	4
1.1 OBJECTIVES.....	6
1.1.1 General Objective.....	6
1.1.2 Specific Objectives.....	6
1.2 JUSTIFICATION.....	7
1.3 RESEARCH DELIMITATION.....	7
1.4 STATE OF ART.....	7
1.4.1 Flex Ray.....	8
1.4.2 Automotive Ethernet.....	11
1.4.3 LIN (Local Interconnect Network).....	11
1.4.4 MOST.....	12
1.5 THESIS STRUCTURE.....	13
2 THEORETICAL REFERENCE	14
2.1 SYSTEMATIC LITERATURE REVIEW.....	14
2.2 EMBEDDED SYSTEMS.....	17
2.4 SAFETY CRITICAL EMBEDDED COMPUTING SYSTEMS.....	20
2.5 COMMUNICATION NETWORKS BETWEEN COMPUTER SYSTEMS.....	22
2.5.1 Communication Network Paradigms – Event Triggered and Time Triggered...	24
2.5.1.1 Event triggered architecture.....	25
2.5.1.2 Time triggered architecture.....	26
2.6 FUNDAMENTALS OF CAN.....	28
2.6.1 Controller Area Network With Flexible Data-Rate (CAN-FD).....	33
3 METHODOLOGY	36
3.1 MCP2515 CAN Module.....	37
4 EXPERIMENTAL RESULTS	39
4.2 EXPERIMENTS AND SCENARIOS.....	39
4.2.1 Scenario 1 - Single Signal.....	40
4.2.1.1 Scenario 1 - Experiment 1 - Single Sign.....	45
4.2.1.2 Scenario 1 - Experiment 2 - Single Sign.....	46
4.2.2 Scenario 2 - Multiple Signals.....	47
4.2.2.1 Node1.....	51
4.2.2.2 Node2.....	52
4.2.2.3 Node3.....	53

4.2.2.4 Node4.....	57
4.2.2.5 Node 5.....	58
4.2.2.6 Description and demonstration of transmitted signals.....	59
4.2.2.6.1 Dc motor speed control	60
4.2.2.6.2 Ambient temperature measurement.....	64
4.2.2.6.3 Light sensor signal.....	67
4.2.2.6.4 Scenario 2 - Experiment 1 - Aperiodic Signal Transmission.....	70
4.2.2.7 Latency.....	71
4.2.3 Scenario 3	72
4.2.3.1 Gateway CAN.....	75
4.2.3.2 Simulated temperature signal.....	78
4.2.3.3 Simulated oil level signal	79
4.2.4 Scenario 4	80
5 CONCLUSION	82
5.1 FUTURE WORKS	83
REFERENCES.....	84

1 INTRODUCTION

We live in a time where data streaming processes are becoming more common in many different segments. This process seeks to share data and increasingly optimize the use of physical transmission media and other devices.

The growing technology embedded in the automotive industry is made possible by microcontroller-type computer systems. Typically, a microcontroller is connected via point-to-point connections with a set of sensors and actuators for managing a vehicle segment. Through the CAN bus, data managed by a microcontroller can be shared with other microcontrollers, allowing the optimization of physical media and the integration between these computer systems.

Data generated by the most diverse device types is shared with a multitude of device types, characterizing the data entry process that will be consumed by computer systems. This sharing is done through also shared physical means. By consuming this data, computer systems generate output signals to devices on the same or other networks. Network protocols in turn are responsible for generating and interpreting signals carried over a shared physical medium.

This mechanism of data exchange between computer systems, standardized by protocols, began to be used by the Brazilian automotive industry in the 90's. Until then, the signals generated by sensors, and actuator states were brought to the control devices through point systems. to the point. Therefore, from the mentioned period, the concept of data sharing between automotive computer systems began to be used.

Another factor that led to the use of communication networks in automotive systems, besides the sharing of resources, was the economy and optimization of physical means of communication, which led to a consequent reduction in the weight generated by the larger number of cables needed for point systems. to the point, that as mentioned below can reach 90kg. Among other advantages, weight reduction assists in fuel economy.

The harness weighting reaches more than 91kg depending on the vehicle, is several kilometers long and the third expensive and heaviest component of a car (VAN RENSBURG and FERREIRA, 2003)

As a solution to reduce the large number of cables used in automotive communications, we used the Controller Area Network (CAN), developed in the late

1980s by Bosch. Different parts of vehicles have computer systems to control and monitor their variables, and signals from this segment are available to other segments through the CAN bus.

All the electrical system nodes inside the bus vehicle are attached to the CAN bus through their own respective CAN transceiver, CAN controller and SBC (SALUNKHE, KAMBLE and JADHAV, 2016).

For example, the motor run controller provides frames containing variable data on the CAN shared bus. These frames are propagated in broadcast. The other controllers can decide whether or not to use the data contained in that frame through a data identification field that it loads. Thus, for example, data generated by the engine controller can be used by the dashboard controller to display variable values such as engine temperature, lubricant pressure, fuel level, diagnostic messages, etc.

With the evolution of automotive and industrial technologies and the need for interaction not only between computer systems, but with the external environment and people, the current CAN network needs upgrades to satisfactorily support growing data traffic on the network. State of the Art E/E-Architectures tend to integrate additional Electronic Control Units (ECUs) with a direct connection to the necessary sensors and actuators for optional functionality (BRUNNER et al, 2017).

Characteristics of current CAN networks make the problem of increasing data volume more evident. Some of these features include broadcast messages. This often causes a controller to receive a message from which it is not a recipient and needs to delete it. Computation time is consumed before the message is parsed before being deleted.

There are some solutions that propose major improvements in the CAN network to address the considerable increase in data generated by the different segments of the automotive network is the integration between new services and features and the CAN protocol. Firstly, we can cite a new approach to the CAN network, which will come into use from 2019 on some vehicles, called CAN-FD. In addition, another approach under development provides redundant mechanisms for the CAN network. This approach is called FlexCAN.

These new implementations should deliver signals that carry important data on the functioning of automotive devices (Vehicle System), as well as data that will generate actions on Vehicle System regarding the environment where vehicles will be inserted (Automotive Perception System) at intervals. deterministic and acceptable

time limits for the correct functioning of the system. These questions are related to autonomous vehicles, which will be the area of application of this research. In relation to this work, these new approaches will be treated as future work.

Especially autonomous driving requires timing and data rate to be strictly guaranteed by the in-car network (STEINBACH et al, 2015).

1.1 OBJECTIVES

The work in question, which documents the research to be carried out and addresses the concepts involved and necessary for the development of the research. The general and specific objectives in accordance with the topics covered are described below.

1.1.1 General Objective

The work in question seeks as a general objective:

- Present contributions and improvements to current CAN networks.

1.1.2 Specific Objectives

Specifically the overall goal will be achieved as follows:

- Understand the characteristics and properties of CAN;
- Create a prototype for experiments and to understand how the Arduino controller works along with the CAN MCP 2515 interface;
- Implement data transmission larger than the CAN frame data field;
- Implement multiple signal transmission within a single CAN frame;
- Implement a CAN gateway to reduce the broadcast domain of messages on the network.

1.2 JUSTIFICATION

Given the current scenario of system integration and the increasing development of technology for the automotive sector, new solutions, as long as viable are always necessary. In automotive embedded electronic systems, when it comes to feasibility, properties such as determinism, efficiency and cost are involved.

Thus, it would need only a possible update to the standards of automotive networks already existing and widely used without the great work of implementing a new technology, such as FlexRay for example, from the beginning. The feasibility of the research in question is the fact that there will be an update on a technology already established in the industry and low cost, to meet the new demands on communication in the automotive sector and may also be applied to the industrial sector.

1.3 RESEARCH DELIMITATION

In the work in question will be addressed themes that have relationship with what is described in the general objective. The research will be limited to the development of a method to implement improvements in the CAN network that aim to optimize the use of the shared bus. A prototype of a network formed by microcontrollers responsible for the generation and consumption of signals similar to those used in automotive networks will be built, where the characteristics described in the specific objectives will be implemented.

1.4 STATE OF ART

There are currently several automotive communication network technologies available that can be used according to design requirements such as data rate, traffic prioritization, meeting time demands, size of data transported, resource availability, etc. The CAN protocol itself has some variants associated with the factors mentioned. The differences between these variants are mainly in the network data rate.

Low Speed CAN, known as LS-CAN, currently standardized to ISO 11898-3: 2006 has a data rate of 40 to 125 Kbits / sec. This is a low value today, but this version

of the protocol has the advantage of being fault tolerant. In the event of a bus failure, connected devices continue to communicate, as in this pattern each device on the network has its own termination.

This simple, low standard meets communication systems where speed is not important, such as lighting systems, window actuation, lock actuation, air conditioning systems, etc.

High Speed CAN, known as HS-CAN, has a higher data rate from 40 Kbit / s to 1 Mbit / sec. It is the standard most used today in automotive networks due to its low cost, robustness and high speed. Due to this high speed you need 120 ohm resistors at each end of the network. Standardized by ISO 11898, it is the basis for SAE J1939.

In 2012, Bosch created a new version of the CAN protocol called CAN-FD (Controller Area Network - Flexible Data Rate). This new version brought important changes as the larger 64-byte frame data field counts 8 bytes in previous versions and higher data rate, now reaching 8 Mbit / sec (in previous versions it was 1 Mbit / sec). This higher data rate is used in the data transmission phase. The arbitration phase is still running at 1 Mbit / sec.

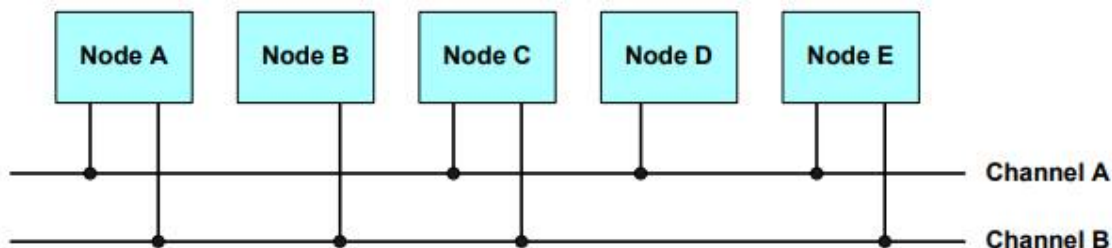
1.4.1 Flex Ray

FlexRay is characterized by its high level of determinism and security in the transmission of information, as well as high data rate (10 mbps). It also allows members to be organized into several different topologies and may also use hybrid topologies. By contrast, it is too expensive and complex an architecture.

FlexRay can use one or two wires as a physical transmission medium. Using two wires is a way of creating redundancy for fault-tolerance transmission media. The use of one or two wires can be applied to various topologies. There are several ways to design the FlexRay cluster. It can be configured as a single-channel or dual-channel bus network, a single-channel or dual-channel star network, or in various hybrid combinations of bus and star topologies (MAKOWITZ; TEMPLE, 2006).

FlexRay also allows several different topologies, which can be related to using one or two wires for transmission. Figure 1 shows the passive topology, similar to the bus topology, where a network member can be connected to one or both communication channels.

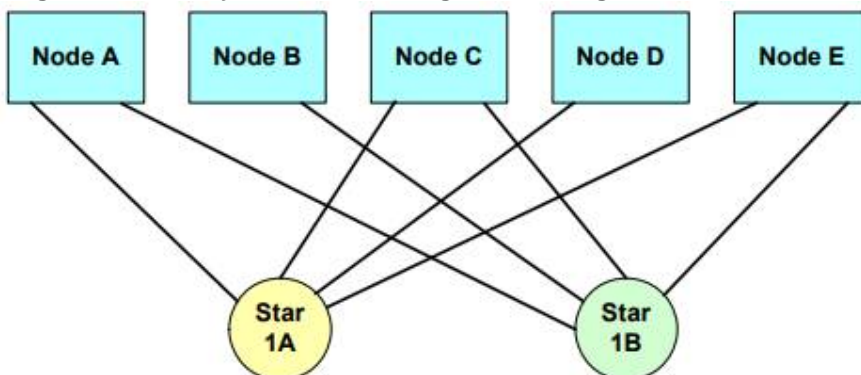
Figure 1 - FlexRay passive bus topology



Source: MAKOWITZ; TEMPLE (2006)

Another possible topology is shown in figure 2, called active star topology, where a network member acts as a concentrating device, receiving the frames and redirecting them to their destination. This topology can use single or dual channels of communication and can be configured in many different ways.

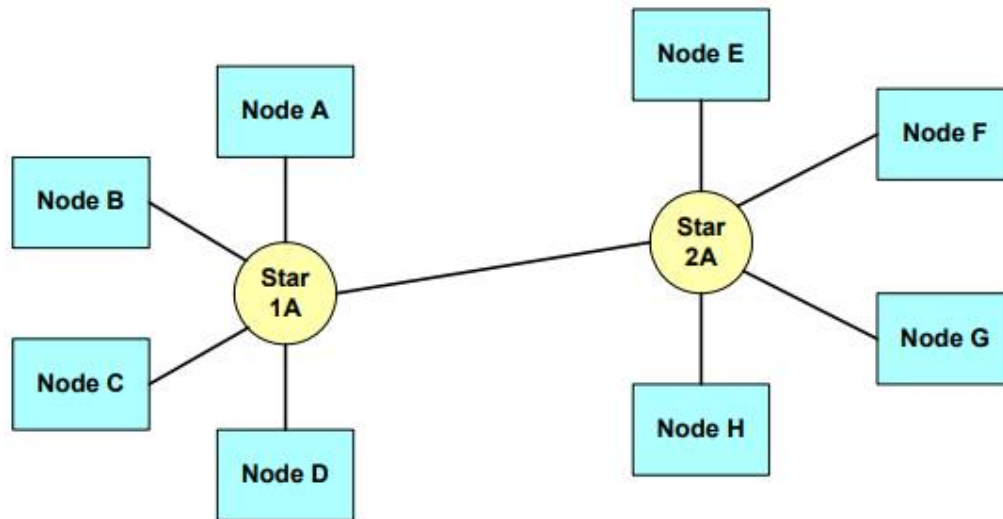
Figure 2 – FlexRay dual channel single star configuration



Source: MAKOWITZ; TEMPLE (2006)

Figure 3 below illustrates a composite star topology. This topology aims to increase the physical distance between network members and optimize physical means of communication.

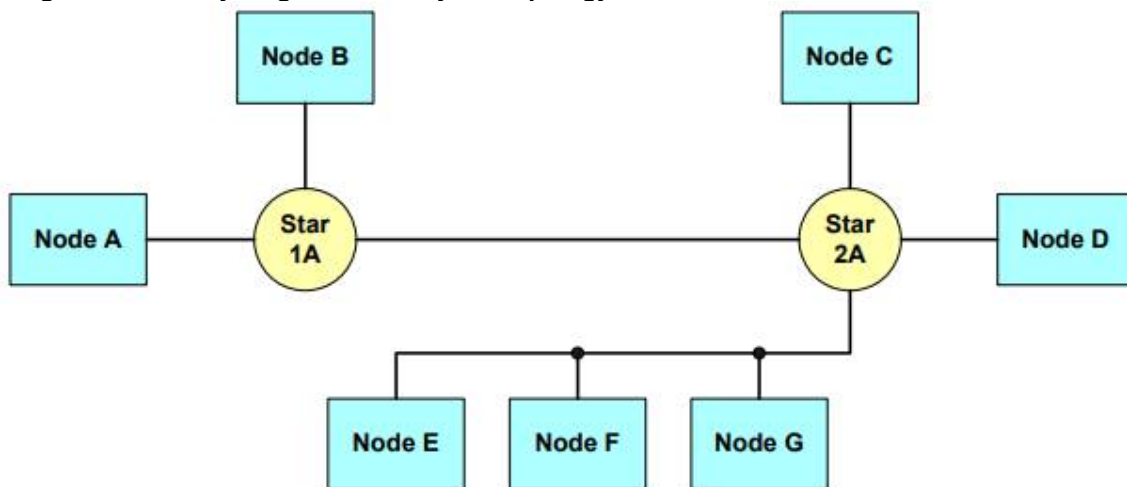
Figure 3 – FlexRay single channel cascaded star configuration



Source: MAKOWITZ; TEMPLE (2006)

The topology used by FlexRay can also be hybrid, as shown below:

Figure 4 - FlexRay single channel hybrid topology



Source: MAKOWITZ; TEMPLE (2006)

For managing multiple bus accesses, FlexRay uses the well-known TDMA, where each member of the network is synchronized with each other and waits for their time period to perform a transmission. In addition, the architecture still reserves a space in time for dynamic frames to be transmitted, thus meeting the two paradigms of access to the physical transmission medium, making the design of a protocol-based network somewhat complex.

1.4.2 Automotive Ethernet

Ethernet is a widely used architecture in corporate computer networks. It is a communication technology that has caught the attention of automakers due to the fact that it achieves high data transmission rates through twisted pair metal cables and in some fiber optic versions. Another advantage is that it uses multiple physical media for transmission and reception, allowing network nodes to operate in full duplex mode.

One of the challenges of using Ethernet in automotive networks is the fact that it is a network sensitive to various types of external interference on the transmission medium, in which case failures in transmission and reception of data. Because there is no synchronization between nodes, there are multiple devices transmitting at the same time. This leads to a scenario of lack of temporal determinism and latency that may be excessive when transmitting a critical signal from a sensor to a controller.

There are working groups that are addressing these issues to enable the use of Ethernet architecture in automotive systems. One is the IEEE 802.1 group (TSN task group), which features several implementations in the Ethernet architecture to meet certain time demands required in automotive communication systems.

Another working group seeks to address the issue of noise that may impair transmission over the Ethernet network that originates in automotive systems such as ignition coils and alternators. These are changes made to the physical layer, initially designed by Broadcom, initially called BroadR-Reach, and then standardized by IEEE, group 802.3bw, and then called 100BASE-T1. It operates with 100 Mbps data rate that uses a single pair for transmission and reception, being made by overlapping signals.

1.4.3 LIN (Local Interconnect Network)

LIN is a serial communication protocol that uses bus and broadcast topology. It is a communication protocol of the master / slave type. For this reason, it does not need mechanisms to detect bus collisions. The bus consists of a single wire, can be up to 40 meters long and allows speeds of up to 20 kbps. It is considered a low speed network, but also in extremely low-cost compensation. The frame data field has variable sizes, which can be 2, 4 or 8 bytes.

It emerged in 1999, in its version 1.0 through a consortium between several car makers and Motorola. It went through updates until it reached its 2.2A version in 2010 and was standardized by SAE as J2602. In 2016, the CiA (CAN in Automation) group standardized it as ISO 17897.

The LIN network can be used as a complement to the CAN network and can be used as well as CAN-LS in systems where transmission speed is not important. A LIN Master can operate as a gateway between the LIN network and the CAN network.

Its operation is relatively simple. The master node requests information from slaves, who send a response with their data when requested. This is done by sending the master a frame called a header that contains a 14-bit field that marks the beginning of the frame, an 8-bit field used for synchronization, and an identification field that has 6 bits. The identification field is used to identify the slave that must respond and the requested data. The requested slave responds with a frame called response, which contains only the 16 to 64-bit data field and a checksum field.

1.4.4 MOST

MOST stands for Media Oriented Systems Transport. It emerged in 1997. It is a synchronous automotive network where a master provides the clock for all other devices on the network. It is designed to synchronously carry video, voice, audio and control signals through optical fibers or metallic means. Because it operates synchronously for data streaming and uses fiber optics, it is a highly deterministic network. Network specifications are maintained by a consortium of some of the world's largest automakers.

The network topology can be daisy-chain or ring (more frequent). Regarding the OSI model, the MOST network implements services at all layers. As a result it offers various implementation and maintenance facilities. These features include plug & play functionality, which facilitates the process of inserting or removing devices on the network. It also has network health testing and diagnostic tools.

There are three generations of the MOST network, the first being called MOST25, which uses optical fiber as the physical medium and operates at up to 25 Mbps. The second generation, called MOST50 uses optical fiber or UTP metallic cable.

Operates at data rates of 25 to 50 Mbps. MOST150 is the third generation and operates data rates up to 150 Mbps using fiber optic or UTP wire.

MOST network is a strong competitor of automotive Ethernet for use in perception systems. Their use for data transmission in automotive systems such as Power Train or Engine Control bumps into issues such as CAN, Flex-Ray or LIN being major paradigms of the automotive industry. Another fact is that there is a huge technological difference between the current sensors and actuators used in automotive networks and the MOST network. Transporting data from these sensors and actuators to the MOST network would require the use of transceivers and gateways, and today's automotive networks are now closer to these devices.

1.5 THESIS STRUCTURE

This paper is organized in 5 chapters. Chapter 1 presents a general introduction to what will be worked on, objectives, justifications and delimitation of the area to be researched.

Chapter 2 presents the bibliographical theoretical framework used in the work, as well as a description of the steps performed and the systematic literature review methodology used to select the portfolio to support the research. It also presents all the concepts that are part of the area that will be researched, such as computer systems, communication networks, communication network paradigms, devices used in assembling the prototype, etc.

Chapter 3 presents the methodology used in the research development, as well as describes the steps taken to develop the integration between different communication network technologies.

Chapter 4 presents a description of the scenarios and experiments used and an analysis of the results obtained through the tests performed on the created prototype.

Chapter 5 presents the conclusions, presenting the contributions of the work to the area of study and suggestions for future work.

2 THEORETICAL REFERENCE

This chapter gives an overview of the elements that are related to the research performed.

Section 2.1 presents the methodological steps of the systematic literature review used. Section 2.2 presents concepts on autonomous driving. Section 2.3 presents concepts related to equipment and protocols that are applied to the project, and finally Section 2.4 presents concepts about embedded systems used in critical and safety applications. Section 2.5 presents concepts about communication networks and subsection 2.5.1 presents communication paradigms related to the researched area. Section 2.6 and subsection 2.6.1 present fundamentals about CAN networks and their variants.

2.1 SYSTEMATIC LITERATURE REVIEW

Quality scientific research demands after choosing the theme of a solid theoretical basis, which aims to provide the researcher with necessary knowledge in the area investigated, state of the art construction and identification of gaps in the area to be investigated.

This material is made up of articles from journals, congresses, book chapters, etc., that are related to the area investigated, and should be selected according to the research theme and subthemes, keywords, etc. One of the difficulties encountered by researchers in this step is to correctly select the materials, because when performing searches in databases, usually many papers are returned.

This initial task can be complex and exhausting, requiring a large amount of time from researchers (BARHAM *et al*, 2014).

Thus, it is recommended to use some systematic literature review tool, which will help the researcher to organize his theoretical basis for research development and the creation of the state of the art of the subject to be investigated. The amount of material available and current communication technologies contribute to a considerable amount of work available to the researcher.

This results in an increase in the global scientific literature as a whole, with materials found in various databases available to researchers (BHUPATIRAJU *et al*, 2012).

The use of a tool that organizes these works in a qualitative manner is essential. The number of scientific publications and the number of journals has increased considerably in recent years (PAGANI, KOVALESKI and RESENDE, 2015).

This wide range of jobs requires the selection of those that are most significant to make up the portfolio (SMALL *et a*, 2014).

To assemble the portfolio of articles for theoretical basis and state-of-the-art construction regarding the theme and objectives of the work, the methodology developed at UTFPR, called *Methodi Ordinatio*, was chosen, although with some adaptations. For example, to search the IEEE Xplore database were considered papers presented in congresses, because in this modality there are relevant works related to the area of study.

Methodi Ordinatio is a Multi-Criteria Decision Aid (MCDA) methodology in the selection of scientific articles for the composition of a bibliographic portfolio (PAGANI, KOVALESKI and RESENDE, 2015).

The methodology will order these works by their relevance in the area of study. The methodology consists of nine phases, which for the research work in question will be described below.

After defining the research theme and tapering it to a certain area, the next and natural step for the development of the work is the construction of a portfolio of related works to assemble the state of the art of the chosen theme.

The work in question will discuss analysis, implementation improvements, prototyping and testing on a CAN-based automotive communication network. As theoretical basis will be addressed concepts about Embedded Systems, Communication Networks and CAN Networks.

The choice of this theme corresponds to phase 1 of the methodology chosen to create the work portfolio for theoretical basis of the research. The search to assemble the work portfolio will be carried out on digital and technical bases. In general, for all axes, works published between 2010 and 2018 will be selected, with some exceptions from older relevant works.

This is a reasonable time for good groundwork not to be discarded, and works that point to new advances in research will be selected. The basis used for drafting the state is IEEE Xplore.

Searches will be performed in the base above using their keywords or combinations so that the search returns the works quantitatively and qualitatively.

In phase 2, a preliminary exploratory search was performed, which allows a relative evaluation of the keywords and the quantity and quality of works returned within the defined time interval.

In this phase searches will be performed in the selected base (IEEE Xplore), using the keywords and combinations between them through Boolean operators.

The terms searched in the databases were combinations between the following items:

- Communications networks;
- Automotive communication networks;
- Periodic and aperiodic signs;
- CAN (Controller Area Network).

Stage 3 of the methodology is to determine which keywords are used and their combinations to perform the database searches.

In stage 4, searches are performed and their results exported to a reference manager application. For this we used the Mendeley Desktop reference manager.

In stage 5, the articles were organized by area and research base (keywords) within Mendeley Desktop. A manual filtering procedure was also performed, based on the keywords, relating them to the titles and abstracts of the articles, leaving 33 papers.

Some duplicate works were eliminated and at this point there was a change in the review protocol defined by the chosen methodology. Papers considered relevant to the research, something detected through title and summary analysis were included in the portfolio.

As this type of work has no impact factor, they were classified differently from the one proposed by the methodology.

In stage 6 were identified the impact factors, year of publication and number of citations of the works through the Google Scholar platform.

In stage 7, the formula suggested by the methodology for ranking works within a spreadsheet, already created with all references obtained in the previous steps, was applied.

Although they do not have an impact factor, which is considered for ranking papers published in journals, the conference papers were also applied the formula, based on the number of citations of the papers, information obtained from the Google Scholar tool.

Some articles have been added to the portfolio manually because they have been published in very recent congresses and have titles directly linked to the theme and objectives of the paper or papers published at conferences.

From the spreadsheet generated with all the information cited above, it is easily possible to apply several types of filter to classify the works published in journals or conferences to read, based on the order in which they were classified by the methodology formula.

The spreadsheet was used to filter papers by the areas described above and sort them for reading.

2.2 EMBEDDED SYSTEMS

An Embedded System is a microcontrolled electronic computer system that is dedicated to controlling a single system. Being a computer system, it is made up of hardware and software. All hardware is encapsulated in a silicon wafer containing processing core, volatile and nonvolatile memories, internal communication buses, communication ports, and I / O devices. It differs from common electronic systems because it has computational intelligence, discussed below:

The software, or Operating System, which in this case is known as firmware, in turn is a set of instructions that control the operation of hardware and are responsible for the computational intelligence of this type of system. Basically a firmware can be divided into two parts.

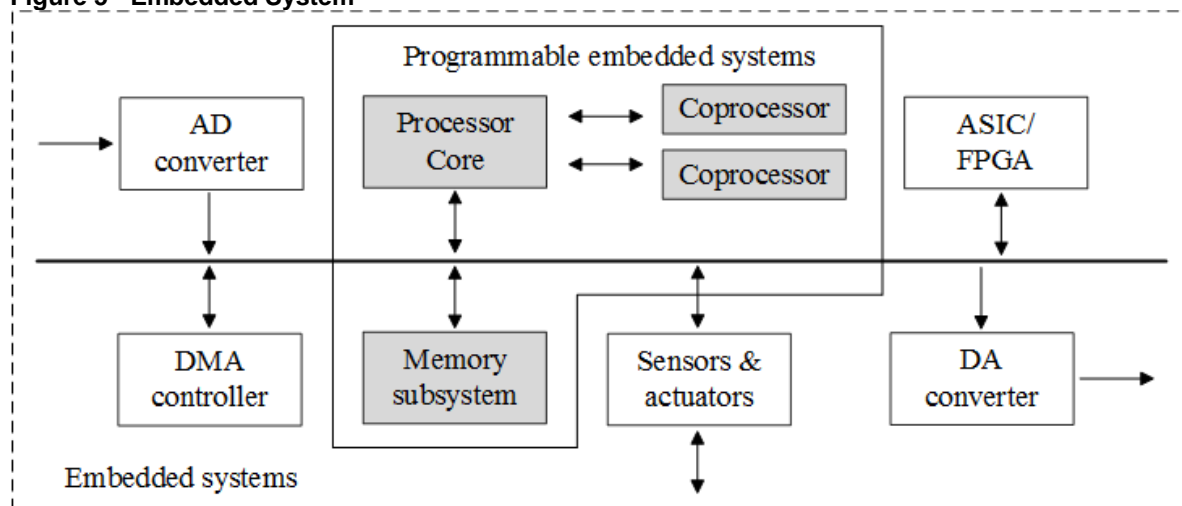
One that configures computer system hardware parameters and others that run cyclically on input data, generating output data, all for an external system to function properly, such as a traffic light, an industrial plant, automotive systems, among others.

These are usually computer systems dedicated to coordinating the actions performed by a very specific system, unlike microprocessor systems that are generic computer systems. In short, they are small-scale computers, dedicated to controlling some kind of process, such as traffic controllers, industrial machines, automotive devices, appliances, etc.

Even very small programs may contain highly sophisticated algorithms, requiring a deep understanding of the domain and of supporting technologies, such as signal processing (LEE, 2000).

Figure 6 illustrates in simplified form an embedded computer system.

Figure 5 - Embedded System



Source: Adapted from Al-Hashimi (2006)

An embedded system is a type of small-scale computational system, so it is made up of several parts as illustrated in the previous figure. The central item, called Programmable embedded systems has the processor and coprocessor, responsible for executing the instructions contained in the firmware, which in turn is stored in a nonvolatile memory area in the memory subsystem.

An embedded system also has interfaces that use data generated by events external to the Programmable embedded system, which are the inputs. These inputs can use both analog and digital signals from external sensors that monitor process variables and take this data to be consumed by the computer system.

Consuming data means executing on them the instructions contained in the firmware used, generating output signals to digital or analog interfaces to generate external events resulting from the processing performed on input data.

Embedded computer systems have different types of constraints, which depend on the purpose of the system. For example, there are systems that must be low cost, systems with high security, systems with high availability, systems with deterministic response times, etc.

Regarding the computer system with time constraints, they are divided into hard real time system and soft real time system. Hard real time systems have strict time constraints, while soft real time systems have some flexibility in their response times.

This operation is possible thanks to the operating system used in these systems, which are called Real time operating system (RTOS). These systems are designed to service all deadlines through task schedulers that ensure that tasks perform concurrently and meet their time constraints.

In other words, it is a system that generates time-deterministic outputs, regardless of disturbances, computational load, resource sharing or other factors that may cause delays in output signal generation. These delays can have serious consequences if the system is in control and interacting with critical processes such as chemical plants, boilers, medical equipment, safety systems, etc.

Embedded systems are used in a dedicated way to control real-world processes, capturing environmental information through sensors or receiving signals through devices deliberately triggered by a user (thus characterizing computer system input processes). A microcontrolled computer system consumes the generated data, processing it and generating output signals for actuators.

As the microcontrollers used in this type of system are systems dedicated to a certain function, they are low cost systems with few computational resources. Slow computers, such as microcontrollers, and timing precision, predictability, and repeatability may be far more important than speed (LEE, 2017).

In obtaining data from the variables involved in the controlled process, the embedded system processor executes software-based instructions to generate outputs by performing calculations, comparisons, selections, repetitions, and other computational operations, with the purpose of generating data and output or response

actions, which they can make information appear to an operator, trigger from some kind of actuator or even generate a PWM signal, for example.

This type of computer system, due to its low cost and possibility of operation based on several types of restrictions, is widely used in the automotive industry to control vehicle subsystems such as engine, brakes, airbags, comfort and entertainment devices, etc.

The systems that control the processes mentioned do not work in isolation. They are connected to a communication network to share resources and exchange information with each other, in order to optimize physical means of transmission and integration between the different processes they control, and the possibility of interaction with larger networks such as the Internet, allowing that are controlled and monitored remotely.

2.4 SAFETY CRITICAL EMBEDDED COMPUTING SYSTEMS

Safety Critical Embedded Computing Systems are computer systems that control, monitor and manage systems where safety and functionality are absolutely essential factors. In fail-safe systems, hardware, software, or an operator detects a failure and modifies effector output so that the system enters a safe, generally nonoperating state (DUNN, 2003).

They are systems in which if there is any kind of failure in its correct functioning or even a malfunction or degraded functioning, the result will be catastrophic. The consequences of a possible failure of such a system include loss of human life, damage to human and animal health, damage to external and remote environments or the environment in which the system is inserted, environmental damage, damage to the system itself. system, etc.

In Safety Critical Embedded Computing Systems, in addition to considering the hardware, which must be robust and functional, the issue of the software used is also essential, especially regarding cyber security. Software failures can also lead to disastrous consequences. In this regard, there are many techniques available for developing efficient applications, such as modeling tools, formalization, software engineering techniques, and algorithm design and analysis techniques.

In this last item, algorithms will define which actions should be performed and can be mathematically modeled and analyzed, in order to develop increasingly efficient solutions in terms of computational resource consumption and response time.

The common approach among all functional safety standards is to identify possible hazards that are caused by malfunctioning of the embedded system and classify the criticality of such a malfunctioning (BAUMGART *et al*, 2014).

The general view of safety assurance is to minimize the risk that may lead to accidents. In this view, not only the computer and software products have to be evaluated for safety, but also the tools used to develop this hardware and software (KORNECKI and ZALEWSKI, 2008).

Generally speaking, if a failure occurs that leads to serious consequences such as those mentioned above, the system can be considered as Safety Critical Embedded Computing Systems. We can cite some examples of various levels of Safety Critical Embedded Computing Systems. A passenger plane has a Safety Critical Embedded Computing Systems, as a crash could result in a crash and the consequences of such an event would be disastrous.

An example of a system used in aviation is the fly-by-wire, where a set of three computers called the primary fly controller (PFC) receives pilot commands through buttons and generates signals that will trigger electric or hydraulic actuators that will move flaps, for example, and these movements are still limited by design.

As a matter of fact, the DbW systems enhance the safety of the vehicle occupants. This is done by making the conditioning of the driving commands easible, by allowing more accurate maneuvers thanks to the use of closed-loop controlled electric drives, and by increasing the car's efficiency, since they utilize electrical equipment with much less losses (PIMENTEL, 2003).

Some automotive systems may also be classified as safety critical systems. As an example, we can mention the ABS brake system or airbag systems. The concept of autonomous vehicles will also need systems with these restrictions, as they will serve for monitoring of traffic lanes and for safe distance control in relation to people, objects and other vehicles.

There are still systems that, if operated improperly or erroneously, can have serious consequences, such as engineering applications for design drawings, structural analysis applications, etc. These systems must be correctly matched to the purpose they serve. Safety-critical systems exist in a certain application context.

Certainly, the details of safety critical aerospace systems are different from those of the space shuttle, process control, or automotive (PIMENTEL, 2003).

Another relevant issue is the security of these systems. Because they have the characteristic of safety critical systems, if the repositories where they are stored are invaded and controlled by malicious people, the consequences would be disastrous. Security is an essential component in safety-critical embedded systems to ensure the required level of safety given the ever-increasing level of connectivity of these systems (MORENO and FISCHMEISTER, 2017).

There are formal methods for identifying, developing and operating this type of system. Functional safety standards like IEC 61508 or the automotive standard ISO 26262 provide a set of requirements and process steps to identify critical functions and classify their criticality (BAUMGART, 2014).

2.5 COMMUNICATION NETWORKS BETWEEN COMPUTER SYSTEMS

With the increasing use of computer systems of various types for the management of various types of processes, the need arose to share physical and logical resources between these computer systems. Due to the increasing number of variables and processes to be controlled, peer-to-peer systems became unviable.

Sharing data, computing resources and transmission media has generated what we know today through communication networks. The integration between computer systems through communication networks has made control systems more interactive, intelligent, versatile and has made better use of both the physical media and data generated by these systems.

Data communication is data exchanges between two devices through some kind of transmission medium, such as a wire cable. The communication system must be made up of the media and protocols (WETHERALL and TANENBAUM, 2011).

By replacing the old peer-to-peer systems with digital communication networks, there has been savings and optimization of physical means of transmission, formerly dedicated only to carrying signals that carry data generated by a single system and consumed by a single control system.

Sharing a physical transmission medium also allows data to be shared and used by computer systems that are different or distant from the systems that generated

these signals, thereby optimizing the use of the physical medium and reducing the amount of material needed to construct that medium physicist.

A network enables the concept of data streaming, where data used by one system is not only isolated on the system itself, but is available so that other computer systems can use that data. When data is exposed to different computer systems through a communication network, different approaches can be made about them through different applications, such as exposure to artificial intelligence algorithms in order to mine the data, for example.

For two or more computer systems to exchange information, there must be a network protocol in addition to some physical medium. A protocol is a formal language for creating messages that will be encapsulated within frames. To be able to communicate directly, machines on the network must know the same protocol.

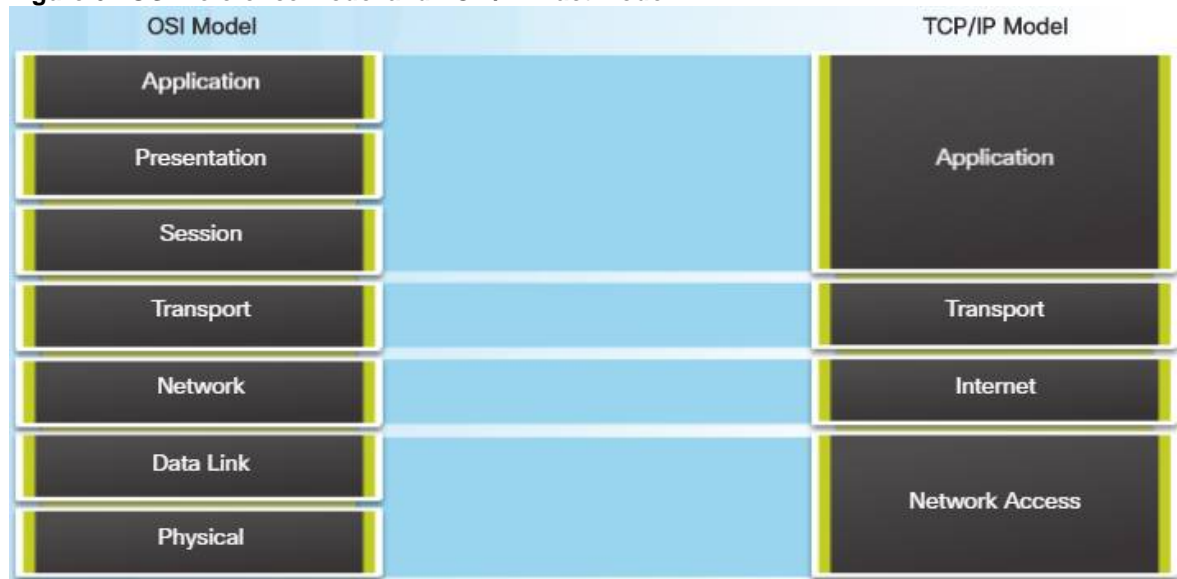
It is still possible for two machines using different protocols to communicate. In this case it is necessary to use another system, called a gateway. A gateway must know the different protocols involved in the network and translate the signal received from one protocol to another protocol.

In a digital network between computer systems, many different types of constraints are involved, such as cost, synchronization, data rate, security, determinism, etc., providing a very large environment for research and development of solutions best suited to today's demands.

The Open Systems Interconnection (OSI) Model serves as a reference model for communication network design. It was created by ISO in 1971 and formalized in 1983. As a reference model only, there is no architecture or network protocol assigned to the model.

The OSI Model defines only tasks performed by network devices to communicate between two stations. These tasks are called layers, and according to the OSI Model to have end-to-end communication between two stations seven steps (layers) are required, which are shown in the following figure:

Figure 6 - OSI Reference Model and TCP / IP Fact Model



Source: Adapted from cisco.ct.utfpr.edu.br (2019)

The figure shows the reference model and a de facto model in use, which is TCP / IP. The TCP / IP model is similar to the OSI model, with some layers less, however. The model has capabilities to encapsulate the data to be transmitted and also the control information for forwarding frames between intermediate devices until message delivery to the device. of final destination.

The CAN network, dealt with in this work and other field networks, usually only use layers 1, 2 and 7. At layer 7 is where data is generated, layer 2 defines frame format according to the protocol used and layer 1 transmits the encoded frame in a format compatible with the medium used (metallic, wireless or fiber optic). In this type of network the services of the other layers are not used.

2.5.1 Communication Network Paradigms – Event Triggered and Time Triggered

In a real-time systems communication network with limited shared resources (there are fewer message paths than there are messages to be transmitted), as a physical means of transmission, there must be some way to ensure that all devices can transmit your messages to other devices on the network.

In other words, network architectures and protocols need to ensure that there is no monopoly on the use of network resources by any device or service. This is done

through scheduling services, which are responsible for giving flow to data generated at end points across the network. In this sense, there are two architectures that provide this type of service, called Event Triggered architecture and Time Triggered architecture.

Event-oriented architecture consists of having a frame transmitted by the bus as soon as possible, depending on the conditions of use of this feature.

The event triggered architecture consists of program modules that respond external inputs or communication messages immediately. The time-triggered architecture consists of program modules that respond them periodically (ITAMI, ISHIGOOKA and YOKOYAMA, 2008).

In time-oriented architecture, there are predetermined time slots for participants to transmit their frames within those slots.

To provide organized bus access in distributed real-time networks, time-triggered communication is mainly used. Thereby, each bus participant is granted bus access during defined time slots (EINSPIELER and STEINWENDER, 2018).

2.5.1.1 Event triggered architecture

In Event Triggered architecture the transmission of the generated messages is done immediately, whenever possible. Whenever possible, it means that resources are available to transmit the frame, such as backbone, for example. If this does not occur, the message is buffered and awaits the release of the transmission channel.

In this approach as members do not have predetermined transmission times, collisions may occur and protocols must be in place to treat possible collisions.

The event triggered architecture consists of program modules that respond external inputs or communication messages immediately (ITAMI, ISHIGOOKA and YOKOYAMA, 2018).

This architecture, despite having a more dynamic behavior regarding the use of shared resources, ends up creating a scenario of non-determinism response time for requests. This means that the device that has made a request and is waiting for a response is unsure how soon it will respond.

Event-triggered systems have the advantage of being much more Flexible and, thus, are much more handy when dealing with changing requirements or uncertain

knowledge (aperiodic events for instance as their period is not known) (SCHELER and SCHRÖDER-PREIKSCHAT, 2010).

This is because as messages are being transmitted over any given time interval, and as they are generated, frames can be formed on either backbones, buffers, transmission media or concentrating devices, causing bottlenecks and non-zero sources. determinism. Although there are services that deal with frames that are primarily in layers above the link, the computational time spent for this analysis of which frame is a priority also becomes a source of non-determinism.

This architecture is used in corporate networks without time restriction and in some cases in automotive networks for the transmission of aperiodic signals or in case of frames without temporal restriction, as a signal generated by the processor that controls the ABS system of an automotive network, for example. In this case, priority issues in transmissions are observed.

It is not recommended for communication between safety critical systems due to non-temporal determinism and the possibility of very high jitters. As a disadvantage of event-based networks, one can cite the fact that they are non-deterministic networks temporally, since as stations have autonomy to start transmissions, they first need to check the availability of the physical medium, and will only transmit if there are resource availability. In case of more than one station trying to transmit, there may be a dispute for access to the backbone, which will generate temporal non-determinism. This queue can make the transmission time of a message unpredictable.

Nevertheless, this approach has some advantages. One of them is its versatility and dynamism in relation to the autonomy that participants have to start transmitting their frames. Another advantage is the ease of adding and removing devices on the network. Since participants have autonomy to perform transmissions, no media access settings need to be made.

2.5.1.2 Time triggered architecture

In Time Triggered Architecture, hub devices and physical media are shared temporarily through a technique called Time Division Multiple Access (TDMA), where devices transmit their frames across the backbone at predetermined and distinct time

intervals. As a result, the possibility of conflicts or collisions because more than one member of the network emits a signal in the physical environment is very small.

In time-triggered networks, the possibility of a collision on the shared medium is minimized by granting every access member only for distinct time slots (EINSPIELER, STEINWENDER and ELMENREICH, 2018).

This environment is more suitable for real-time systems, since due to the fact that members use distinct and periodic time intervals, it defines what is known as a deterministic network. In this scenario, members know what time intervals to transmit and know what time intervals they will receive data.

The time-triggered architecture is more suitable than the event-triggered architecture for hard real-time systems because of its predictable behavior (ITAMI, ISHIGOOKA and YOKOYAMA, 2018).

With the increasing use of computer systems and the consequent growth of these communication networks, some challenges have emerged. For this architecture one of the challenges is how to handle messages that carry aperiodic signals, that is, sporadically generated signals that are outside the context of network time synchronization.

The disadvantage of this approach is its own lack of dynamism and flexibility in network configuration. Because all clocks must be in sync, adding or removing a device from the network will re-sync all participants. This time-based transmission also generates low utilization of the physical transmission medium.

The advantages of using this type of network are the temporal guarantees of frame delivery (determinism), low jitter and predictable behavior. Another important factor is the ability to handle frames as a priority, since access to the backbone for transmission is by time intervals and this access rule can be applied.

In this architecture there is a need for synchronization of the transmitter member with the receivers. This can be done from a global network clock, by distributing transmission times and periods or from a central network member. This technique is known as synchronous transmission.

2.6 FUNDAMENTALS OF CAN

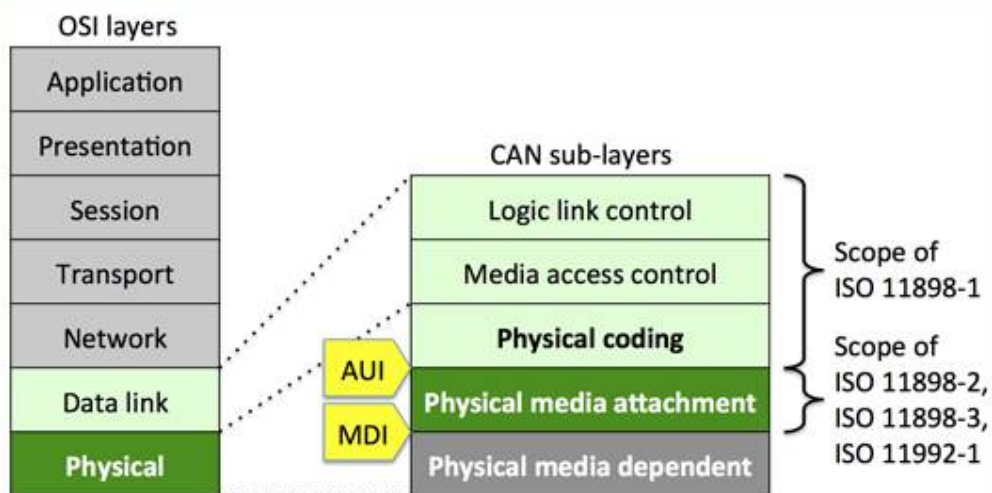
The increasing number of functionalities in modern vehicles with real-time communication constraints demands higher data transmission rates than those currently available from the dominant Controller Area Network (CAN) standard.

The Controller Area Network (CAN) is a serial field bus system with a message-based protocol. The protocol is multi-master based. CAN was developed by the Robert Bosch GmbH and released in 1986 (SALUNKHE, KAMBLE and JADHAV, 2016).

Regarding the OSI communication model, the classic CAN and CAN-FD network services are related to Layers 2 (Data Link) and 1 (physical)

Regarding the OSI communication model, the CAN and FD CAN services are related to layers 2 (Data Link) and 1 (physical). Figure 8 shows the relationship of OSI/ISO Reference Model and layer structure of CAN architecture.

Figure 7 - OSI layers x Classical CAN



Source: CiA - CAN in Automation (2014)

The upper layers are specified by the network designer, that is, used by protocols for data generation and reading which CAN-based as communication systems to perform the transmission.

The data link layer is the one that uses information from the upper software layer and frames the frame accordingly. This layer is subdivided into two sublayers, as follows:

The LLC (Logic Link Control) sublayer is responsible for ensuring the interoperability of any upper layer protocol with the physical medium to be used for transmission.

The Media Access Control (MAC) sublayer is responsible for ensuring media access (CSMA / CA) operations, defining non-destructive arbitration, detecting errors, and creating the frame to be transmitted to the lower layer.

This layer is also where it is treated about the priority of shared media access. The arbitrage field of a frame carries the message's priority along with its identification. The lower the value, the higher the priority, and the station gains bus access to transmit its frame.

CAN is a network that uniquely connects microcontrollers, as its name suggests. This connection is made through twisted pair type metal cable, which can be shielded or not. Importantly, cable quality influences the quality of the transmitted signal. Microcontrollers in turn are connected to sensors and actuators via point-to-point connections with their digital or analog communication pins. CAN also has the characteristic of operating with twisted pair passive metal bus and differential signal, which makes the signal transmission has a good immunity against external interference, something very relevant in automotive networks, where mainly in the engine there are many sources of interference. In this type of transmission, the data is represented by the recessive bit 1 and the dominant bit 0. To represent bit 0 the voltage on both channels is 2.5V, the difference being 0. For bit 1 the voltage on CAN_H is 3.5V and 1.5V for CAN_L, generating a potential difference of 2V.

Regarding message identification, the CAN network has a field in the arbitrariness phase that identifies its content. There is no explicit address in the messages. Instead, each message carries an identifier that controls its bus priority and can serve as an identification of its content (LUGLI and SANTOS, 2009).

The CAN network was initially developed for use in the automotive industry to connect different embedded systems with electronic control (power train, chassis, body, dashboard, etc.). Currently it is also widely used in other industries such as industry, hail-vehicle, aircraft, marine, agricultural equipment, etc., due to its reliability and safety.

Controller Area Network (CAN) is a serial communications bus designed to provide simple, efficient and robust communications for in-vehicle networks (DAVIS, ROBERT et al, 2007).

As with all digital communication networks, CAN's main objective is data sharing and physical means of transmission. Data sharing results in greater and better interaction between automotive computer systems with their users and even with other computer systems that are part of the same vehicle. The concept of data and resource sharing is quite explicit in a CAN network. Connecting to other microcontrollers (CAN) is via a bus shared with other microcontrollers that also share their data.

Thus, a microcontroller has access to a much larger number of signals than its number of input and / or output pins. This distributed system allows signals to be harnessed by different systems, enabling new control and monitoring features to be implemented.

With a CAN network a network can be divided into smaller subsystems, thus not having a single point of failure (gateway). Even if the gateway fails, even in a degraded manner the subsystems will continue to operate interacting with each other.

The following shows the operation of three variants of the CAN network. The differences are in baud rates and data frame size as described below:

The CAN 2.0A version (ISO 11519), also known as Standard CAN, has an 11-bit identifier, 8-byte payload and a maximum bit rate of 250Kbps.

Figure 9 below illustrates the CAN 2.0A standard frame:

Figure 8 - Frame CAN 2.0A

S O F	11 bit identifier	R T R	I D E	r0	DLC	0...8 bytes data	CRC	ACK	E O F	I F S
-------------	----------------------	-------------	-------------	----	-----	------------------	-----	-----	-------------	-------------

Source: Adapted from CORRIGAN (2002)

The CAN 2.0B version (ISO 11898), also known as full CAN, has a 29-bit identifier field, 8-byte payload, and a maximum bit rate of 1Mbps.

Figure 10 below illustrates the 2.0B standard chart:

Figure 9 - Frame CAN 2.0B

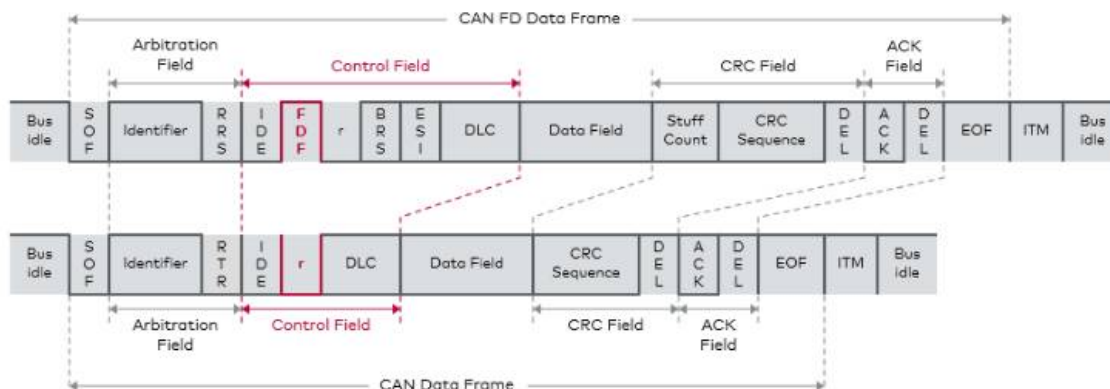
S O F	11 bit identifier	S R R	I D E	18 bit identifier	R T R	r1	r0	DLC	0...8 bytes data	CRC	ACK	E O F	I F S
-------------	----------------------	-------------	-------------	----------------------	-------------	----	----	-----	------------------	-----	-----	-------------	-------------

Source: Adapted from CORRIGAN (2002)

The CAN-FD (CAN with Flexible Data Rate) version also has a 29-bit identifier field, a 0 to 64 byte payload, and a data rate of up to 8Mbps (depending on the quality of the transmission medium).

Figure 11 below illustrates the CAN-FD standard frame and the CAN standard frame for comparison:

Figure 10 - Frame CAN 2.0B vs CAN-FD



Source: Vector E-Learning (2018)

The differences between CAN 2.0 and CAN-FD networks are in data rate and frame payload size. The following is a comparative table between CAN 2.0 and CAN-FD networks:

Board 1 - Comparative board - CAN x CANFD

Property	LS-CAN	HS-CAN	CAN2.0A / 2.0B	CAN-FD
Data-rate	40Kbps to 125Kbps	40Kbps to 1Mbps	20Kbps to 1Mbps	Up to 8Mbps (data field)
Payload	Max 8b	Max 8b	Max 8b	12, 16, 20, 24, 32, 48 or 64b
Identifier field size	11 bits	11 bits	11 bits 2.0A 29 bits 2.0B	29 bits

Property	LS-CAN	HS-CAN	CAN2.0A / 2.0B	CAN-FD
Messages on the network	2048	2048	2048	>5x10 ⁸
Messages on the network	2048	2048	2048	>5x10 ⁸
Media access control	Event-triggered	Event-triggered	Event-triggered	Event-triggered
Signals by payload	One	One	One	Several

Source: Self Authorship

Below are some of the reasons that make the CAN network so popular and widely used in many sectors:

- Low cost: ECUs can communicate via a single CAN interface, i.e. not direct analogue signal lines, reducing errors, weight and costs;
- Centralized: The CAN bus system allows for central error diagnosis and configuration across all ECUs;
- Robust: The system is robust towards failure of subsystems and electromagnetic interference, making it ideal for e.g. vehicles;
- Efficient: CAN frames are prioritized by IDs - the top priority gets bus access, yet frames are not interrupted;
- Flexible: Each ECU connected to CAN bus can receive all transmitted messages. It decides relevance and acts accordingly - this allows easy modification and inclusion of additional nodes (e.g. CAN bus data loggers).

Further Advantages of CAN Bus

- Fast & deterministic;
- Suitable for hard real-time systems;
- In contrast to the Ethernet protocol, with non-deterministic collision detection and backoffs;
- Highest priority message gets immediate access once the bus is free;
- Ethernet packets often need to wait even if the bus is free.

2.6.1 Controller Area Network With Flexible Data-Rate (CAN-FD)

The Controller Area Network with Flexible Data Rate (CAN-FD) is an improvement on the CAN 2.0 protocol as defined in the ISO 11898-1 standard (HARTWICH, 2012).

It presents new features that enable higher speeds than traditional CAN when transmitting the data part, but is designed to be backward compatible.

The CAN FD maintains the major features of the CAN for overriding the existing CAN network bus into CAN FD preparing the next generation in-vehicle network system (OH, WI and LEE, 2017).

The physical and data link layers can be shared by the message formats of both protocols in such a way that any CAN-FD node can receive messages sent from a CAN node. This property allows for a smooth transition from CAN to CAN-FD networks.

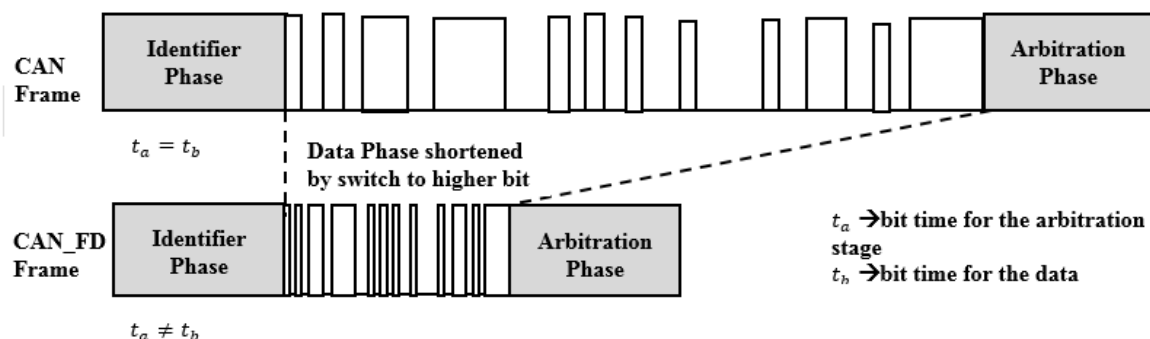
The CAN-FD frame allows the transmission of payloads up to 64 bytes, as opposed to the maximum 8 bytes of CAN and an increased baud rate of up to 8 Mbit/s for the bits following the arbitration stage (the message identifier).

To use large payloads in CAN-FD efficiently, it is therefore needed to pack multiple messages and signals in one single frame (BORDOLOI and SAMII, 2014).

Figure 2.5 shows a comparison of transmission times for the old CAN frame and the new CAN-FD frame, in which the data transmission is shorter thanks to a higher bit rate. The example shows the identifier field in the arbitration phase transmitted at 1Mbps, and data phase transmitted at 4Mbps in CAN-FD.

However, the opportunity of a larger payload can be exploited Figure 12. In CAN FD, the data field phase is shortened by switching from a lower to a higher bit rate. The arbitration phase remains the same. only by redesigning (entirely or in part) the message set.

Figure 11 - Frame CAN 2.0B vs CAN-FD



Source: DE ANDRADE et al (2018)

Finding a new packing of signals into CAN-FD frames is not a trivial problem, indeed, it is an instance of a variable size bin-packing problem (BORDOLOI, SAMII, 2014).

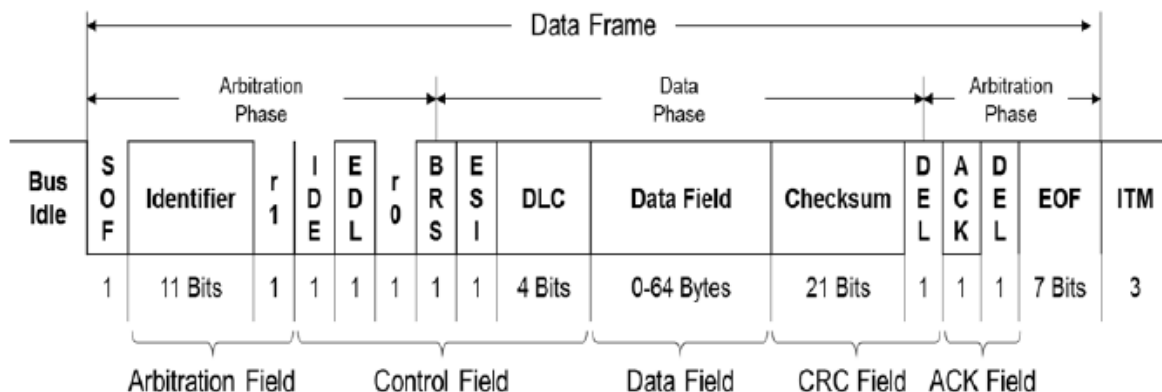
The Controller Area Network with Flexible Data Rate works with two bit rates: one for the arbitration field and the other for the data field. The arbitration bit rate, also called normal bit rate, is applied until the BRS bit (included) and after the CRC delimiter bit (excluded, the frame format is shown in figure 2). The data bit rate starts from the ESI field up to the CRC delimiter and allows up to 8 Mbits/s, instead of the 1 Mbit/s of the standard CAN.

Three new bits are added in the arbitration field: the EDL (Extended Data Length), the BRS (Bit-Rate Switch), and the ESI (Error State Indicator). The EDL field defines the data length with the highest bit at zero if the message size is 0 to 8 bytes (0000 to 1000) and using the remaining 7 combinations with the highest bit at 1 to encode the lengths 8, 12, 16, 20, 24, 32, 48, and 64 (1000 to 1111). The BRS bit determines whether the message is transmitted with the normal rate (up to 1 Mbps) or with the increased one (up to 8 Mbaud).

The ESI bit is transmitted dominant by error active nodes, recessive by error passive nodes (HARTWICH, 2012). Figure 2.6 shows the frame structure of CAN FD, with the standard and extended frames. In terms of format, the new frame does not differ from CAN 2.0 with Start of Frame (SOF), Arbitration, Control, Data, CRC, Acknowledgment (ACK), and End of Frame (EOF) fields (HARTWICH, 2012).

Figure 12 below illustrates the CAN-FD frame format.

Figure 12 - CAN-FD Frame



Source: AN, JEON (2017)

As in CAN, the arbitration field defines the message that wins the contention (lowest identifiers have highest priority).

Similar to CAN, in CAN-FD, an opposite bit is added to the frame every time there is a sequence of 5 bits of the same value (bit stuffing rule). However, stuff-bits are only inserted in CAN-FD from the SOF until the end of the data field, leaving the ACK and EOF fields without stuff bits. Furthermore, the CRC is stuffed (MUTTER, 2015).

In addition, as CAN FD allows larger payload sizes, a new CRC formulation is required to retain the level of reliability of CAN 2.0 (MUTTER, 2015).

We shall consider that the CAN_FD standard adopts different stuff rules for the CRC field considering it not fixed such in traditional CAN.

For the next generation of automotive systems in application level consider that the traditional CAN specification has limitations with bit rate of CAN limited to 1 Mbps and data field limited to 64 bits (8 bytes).

Then, that has taken in motivation for CAN_FD in order to increased demand for bandwidth in automotive communication systems, large gap between the CAN (max 1 Mbps) and FlexRay (10 Mbps), time-triggered communication is not Flexible enough, great effort to migrate to FlexRay/Ethernet, hardware costs and software changes.

Therefore, the use motivating cases, not limited such as accelerate flash from SW in ECUs to production lines and services, increased communication demand between ECUs, accelerate communication in long lines and avoidance of large spacing in long messages.

First of all, we shall demonstrate the motivation and benefits which taken to development of CAN-FD in order to attend some of the applications that was demanding for new technology of communication of networks in level of increasing the payload and data rate. The first three examples are related to automotive application and the fourth is for the automation application.

The capacity of payload for CAN-FD is increased so that it can convey more data and support more the transport protocols. The main application example of the enhancement of CAN-FD in relation to CAN is for the ECU flash programming. For the same SW download, with CAN-FD enables to reduce the time in 50% in order to reduce the time which the vehicle stands in station of flash programming in production line or services.

3 METHODOLOGY

The work to be developed will contribute to computing, in order to implement improvements in the CAN network. These improvements will be demonstrated by prototyping using Arduino boards connected through peer-to-peer connections to diverse devices, generating and consuming signals similar to those used in automotive systems. For sharing the generated data, these Arduino cards will be connected to each other using the CAN MCP2515 module.

In the prototype created the controllers that generate signals will make these signals available on the CAN bus so that they can be consumed by other controllers also connected to the bus. Periodic signals will be used, which will be transmitted on the bus at each execution of the controller source code infinite loop and aperiodic signals, transmitted whenever an event occurs during the controller source code execution.

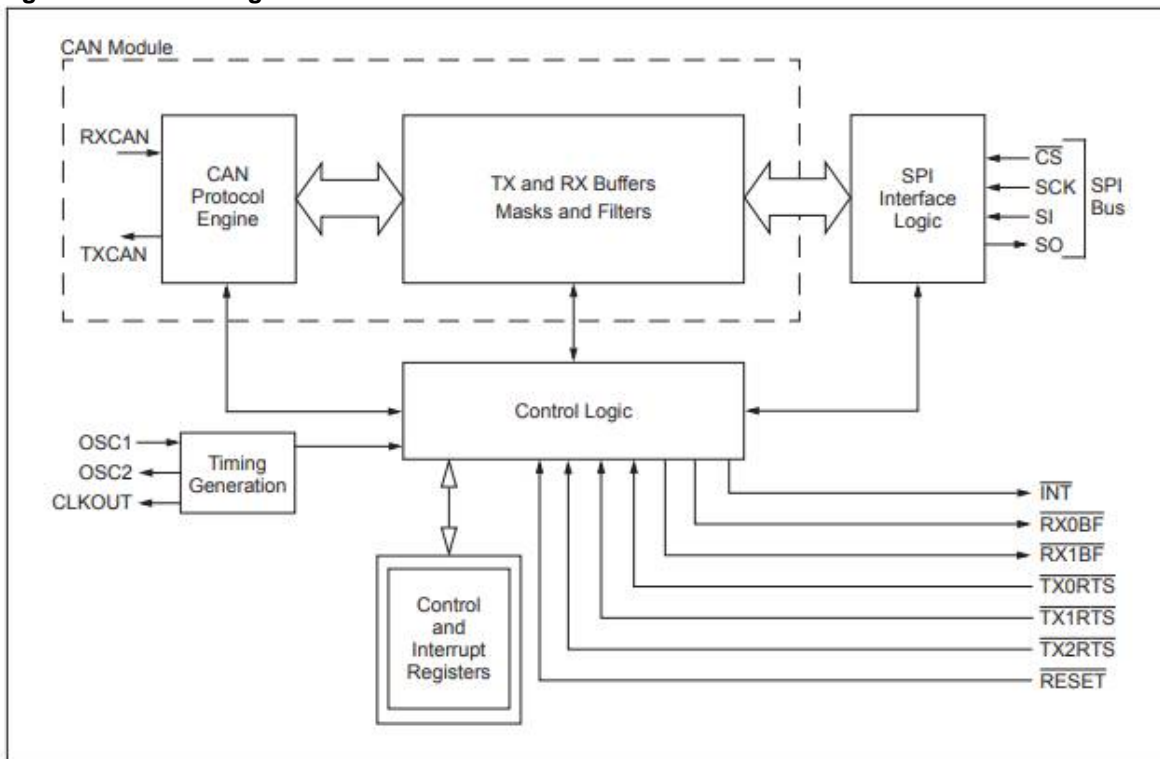
As will be described below, the different signals used will receive an identifier. This identifier will be used by the MCP2515 module in the CAN network arbitration process performed automatically by the protocol implemented in the module to determine which signals have bus transmission priority and also to create the filters used for the controller to decide whether or not to use it. frame received.

3.1 MCP2515 CAN Module

MCP2515 is a controller that is used to give MCUs access to the CAN network in version 2.0B at a speed of 1 Mb / s. Communication between the MCU and the MCP2515 controller is via the Serial Peripheral Interface (SPI) protocol in modes 0,0 and 1,1. This controller has an unused MCU message filtering mechanism that is based on filters that are configured according to the identification of CAN network frames. It has two receive buffers and three transmit buffers with message prioritization capabilities based on their identifier.

It includes a CAN module, which has all protocol mechanisms for message transmission and reception, such as error checking and message identifier analysis and comparison with user-defined filters. It also has a set of registers that is used to configure the operations performed by the module and the block responsible for the SPI protocol. Figure 13 below shows the internal structure of the MCP2515 module. The CS, SCK, SI and SO pins connect the module to the Arduino board through the SPI protocol. The SPI controller connects to the transmit and receive buffers, which in turn are connected to the module that implements the CAN protocol, which has the RX and TX communication interfaces connected to the TJA1050 transceiver, which converts the data into the electrical signals that will be transmitted. CAN bus (CANH and CANL).

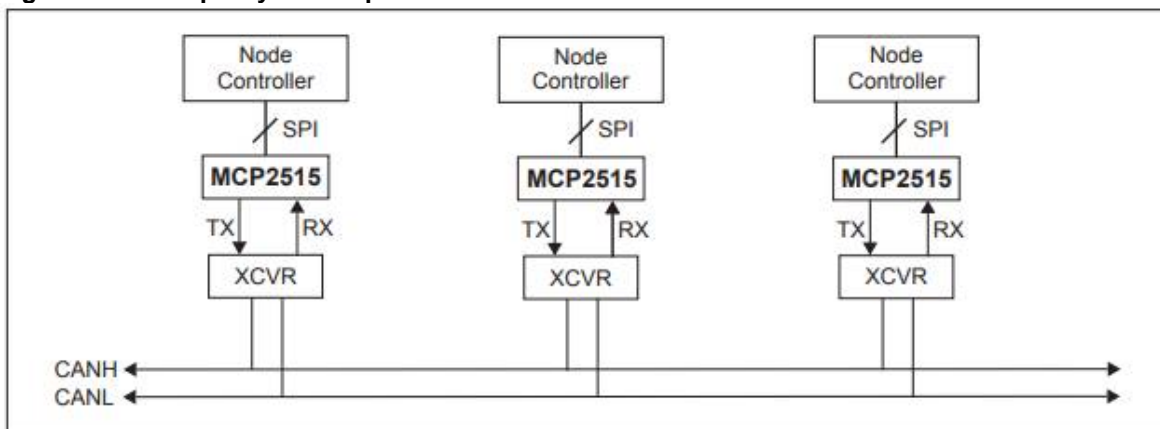
Figure 13 - Block Diagram MCP2515 CAN Controller



Source: microchip.com (2018)

Figure 14 below gives a simplified example of a CAN network assembly between MCUs using the MCP2515 controller. The image shows the controller that is connected via point to point links with sensors and actuators and the MCP2515 module connected via the SPI protocol. The MCP2515 module is connected to the bus via the TJA1050 transceiver.

Figure 14 - Example system implementation

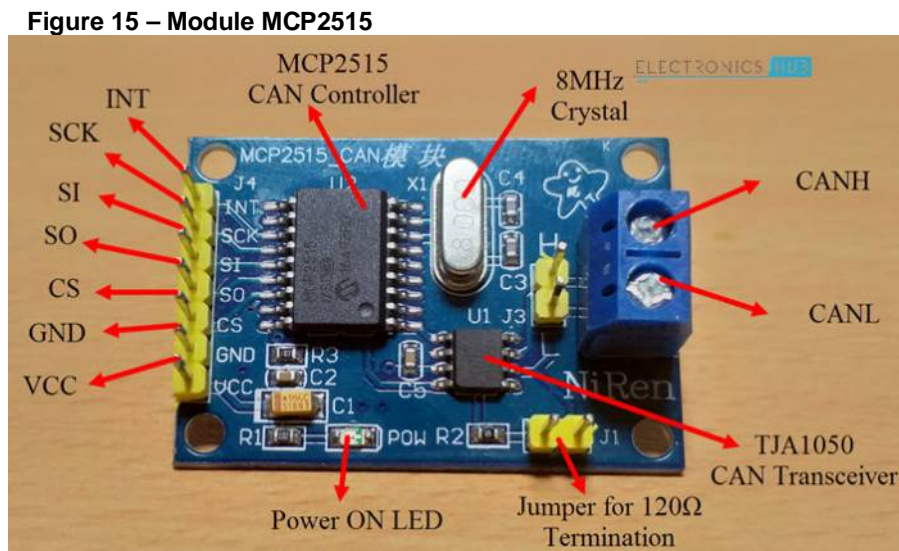


Source: microchip.com (2018)

4 EXPERIMENTAL RESULTS

The validation of the proposals present in this work will be through the implementation of prototypes, initially using the Arduino platform. Arduino is an open source platform, widely used in academia for conducting experiments and developing research. It is a platform that has a microcontroller to perform the actions contained in the software created and also has additional peripheral devices, which allow its integration with external systems. Its more detailed description can be obtained at arduino.cc.

In conjunction with the microcontroller mentioned above, a CAN BUS MCP2515 module was used. This module allows you to connect the microcontroller used or others to a CAN network and make the transmission or reception of data available on the bus. For integration between the communication module and Arduino the library “mcp_can.h” was obtained from GitHub. Figure 15 below shows the MCP2515 module in detail.:



Source: electronicshub.org (2018)

4.2 EXPERIMENTS AND SCENARIOS

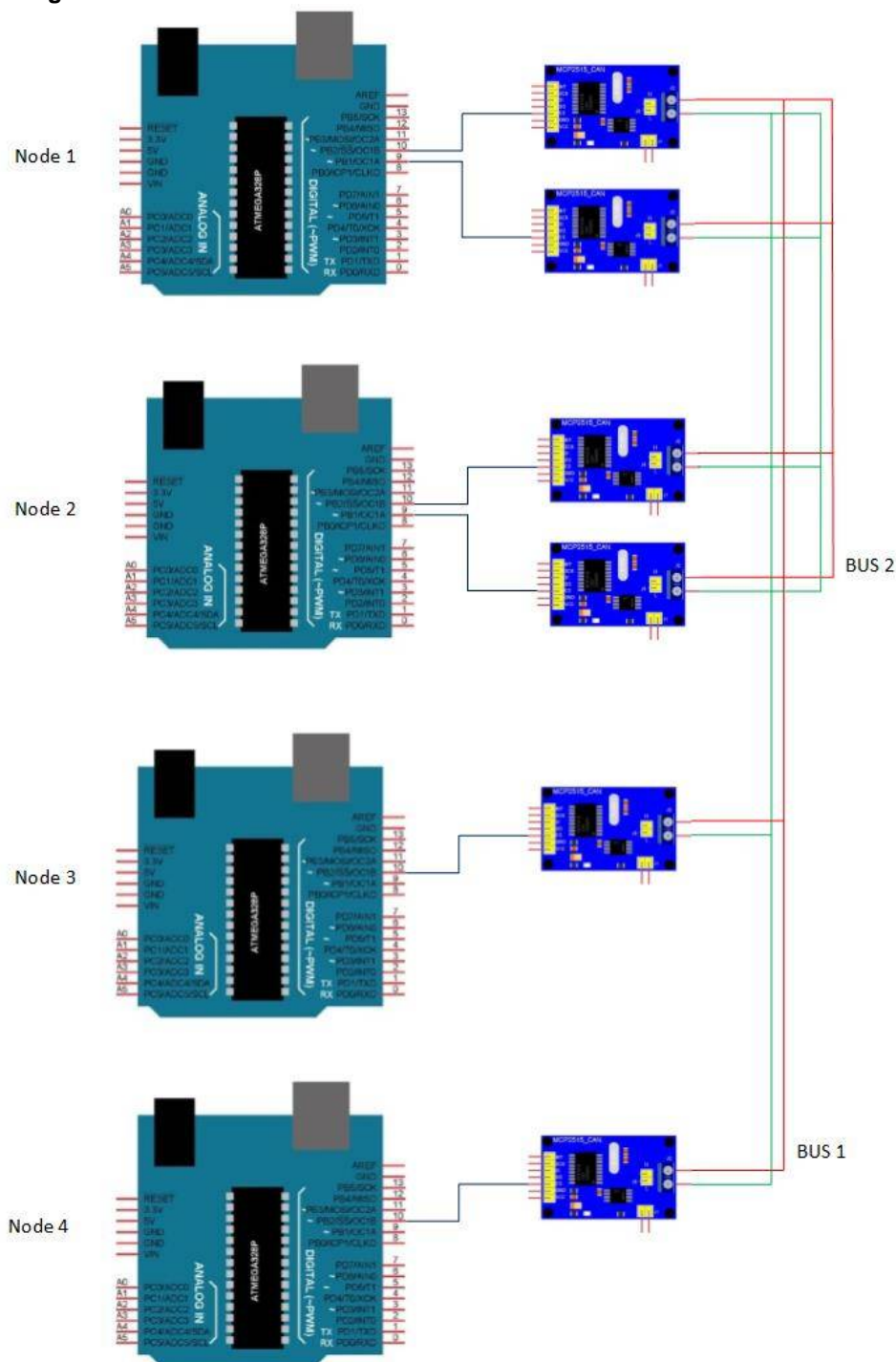
The first sets of experiments are based on a publication from electronicshub.org. The scenario created in the publication has been enhanced with

redundant controller interfaces and a also redundant CAN bus to meet safety critical systems requirements through load balancing between transmitters and buses and redundant interface that comes into operation when problems occur in the interface or bus used.

4.2.1 Scenario 1 - Single Signal

To perform the experiment, the topology shown below was set up to meet reliability and availability, and the use of redundant paths to avoid congestion in frame transmission. In addition, as a result, a more robust network is obtained regarding CAN bus availability. This will be done by using redundant CAN MCP2515 modules on the controllers and consequently redundant communication buses between network members. Figure 16 below demonstrates the scenario created:

Figure 16 - Scenario 1



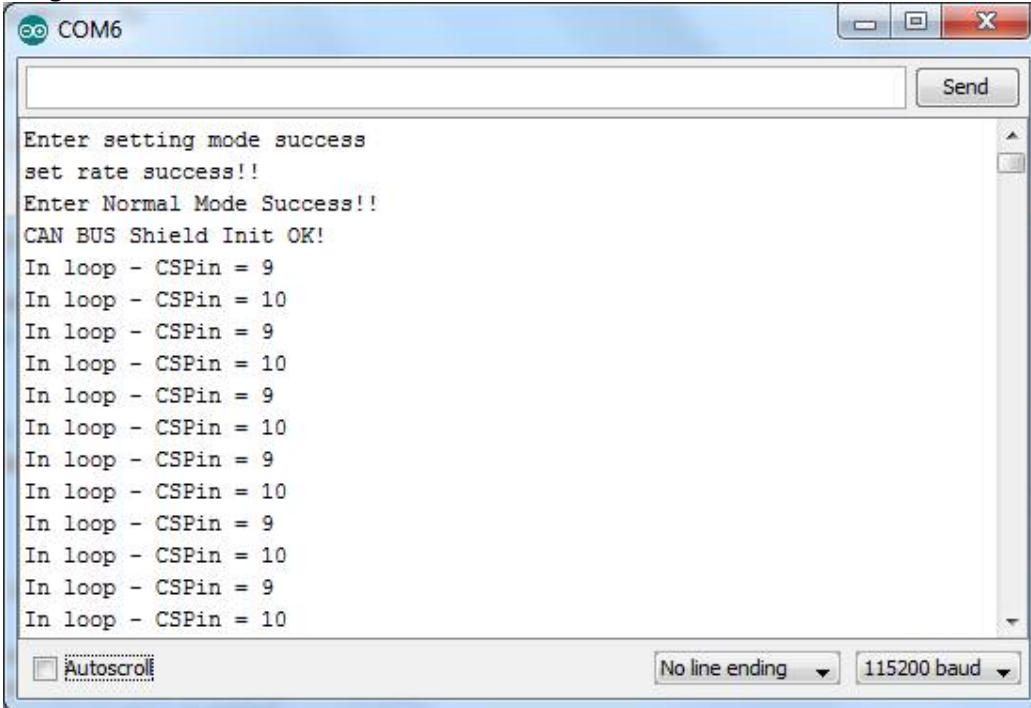
Source: Self authorship

In scenario 1 shown above, Node 1 and Node 2 have redundant CAN bus interfaces. These interfaces can be used in different ways to meet temporal transmission requirements of important data and make the network fault tolerant, and

these network members can be used for critical system control, for example. The frames generated by these devices can be sent across different interfaces, creating a transmission load balanced communication system and still being tolerant to failures that may occur at the communication interface or bus.

The CAN bus is known to consist of two communication channels, called CAN-H and CAN-L, where a bit is interpreted as 0 or 1 through the potential difference between the two channels. Due to this it is necessary to use resistors at the end of the bus. If this resistor is not used, when the signal reaches the end of the bus it will be reflected back to the channel causing collisions and impairing communication. In the experiment this was proven, because without the use of resistors at the end of the bus the reception of transmitted frames would not occur. The function of the resistor is to dissipate the signal at the end of the bus so that it is not reflected. Although most literatures recommend 120 resist resistors, in the experiment 300 Ω resistors worked properly.

To perform the experiment, an 8-byte frame was transmitted repeatedly from Node 1. Initially, the Arduino serial interface was used to analyze the transmitted data. With the implementation made, the transmission of an average 584 frames per minute was calculated. The following figure illustrates the frame transmission, always happening alternately between the two interfaces connected to the host, as shown in the figure above.

Figure 17 - Frame transmission

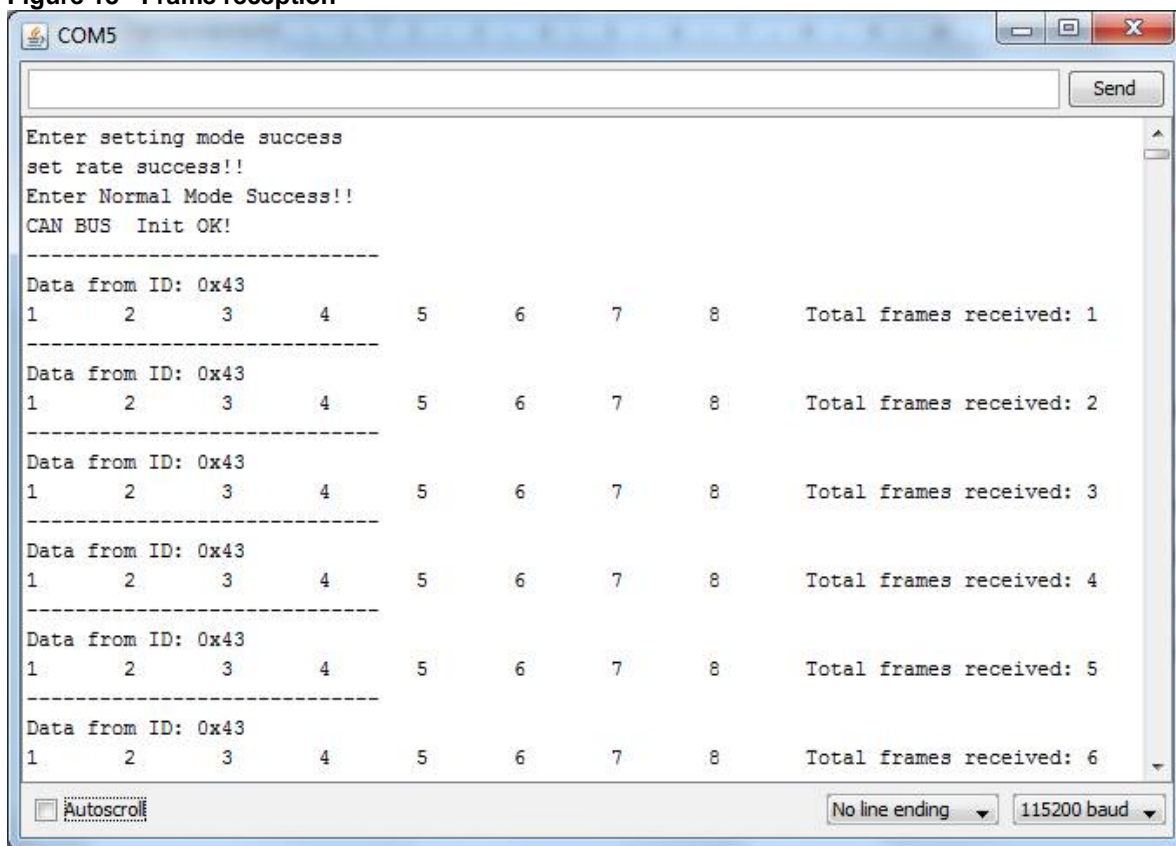
```
COM6
Enter setting mode success
set rate success!!
Enter Normal Mode Success!!
CAN BUS Shield Init OK!
In loop - CSPin = 9
In loop - CSPin = 10
In loop - CSPin = 9
In loop - CSPin = 10
In loop - CSPin = 9
In loop - CSPin = 10
In loop - CSPin = 9
In loop - CSPin = 10
In loop - CSPin = 9
In loop - CSPin = 10
In loop - CSPin = 9
In loop - CSPin = 10
In loop - CSPin = 9
In loop - CSPin = 10
Autoscroll No line ending 115200 baud
```

Source: Self authorship

The image above shows the contents of the CSPin variable that stores the communication pin number used for Arduino to transmit the frame to the CAN module used. Each “In loop - CSPin = x” line corresponds to the transmission of a frame through the buses used in this created network segment.

The following figure shows the receiving device receiving the frame transmitted by the sender, as shown above

Figure 18 - Frame reception



Source: Self authorship

The image shows the 8-byte frame repeatedly received from the transmitter, as well as the received frame number, which has the id 0x43.

The following sections describe two experiments performed to prove the robustness and availability of the proposed network above in the event of communication bus disruption or defects in the network node's CAN communication interface. The test basically consists of transmitting frames over a 30 second period. Within this time window, the communication interface will be disconnected for 5 seconds, simulating a fault situation. The following tests show the improvement in network data availability and transmission in the proposed redundant bus topology over a common topology.

4.2.1.1 Scenario 1 - Experiment 1 - Single Sign

In this experiment a classic topology was used, using only one communication bus between the network nodes. The test basically consists of disconnecting the transmission interface from the network for 5 seconds and counting the number of frames received at the destination. To prove the results, three counts of frames received at the destination were performed and at the end the arithmetic mean between the three data collections was extracted.

The experiment described in this section is to add a CAN interface to the network member connected to a redundant bus connected to the main bus, thus creating a multipath environment for frame transmission. In this scenario, the controller switches the communication interface used for frame transmission through redundant bus, balancing data traffic between available interfaces and buses, providing multiple frame transmission paths.

The tables below show data collected in a classic environment and in an environment where the proposed redundant interfaces and load balancing for the CAN network has been applied. The same frame transmission procedure was used, but this time a fault simulation was made, disconnecting the transmitter interface from the CAN network for 5 seconds. The table below illustrates the results obtained. It can be observed that there was an improvement of about 6% in the reception of frames, compared to the scenario without redundancy. The following tables illustrate the result obtained:

Table 1 - Analysis of results obtained with single interface

Transmitted frames	Received Frames	Lost Frames	Percent loss
286	244	42	15%
289	222	67	23%
287	232	55	19%

Source: Self authorship

The following table 2 illustrates the behavior of the above scenario, now with redundant communication interface on the transmitting device. Data were obtained with the communication interface disconnected for 5 seconds within a 30 second transmission period.

Table 2 - Analysis of results obtained with dual interface

Transmitted frames	Received Frames	Lost Frames	Percent loss
296	268	31	10%
300	261	39	13%
294	248	46	15%

Source: Self authorship

The above tables clearly illustrate the advantage of using architecture with systems containing redundant resources, where it can be seen that the percentage of frame loss is lower. This result demonstrates that in the case of load balancing application, communication channels, even being redundant, cannot fail, because in this technique redundant channels are used to decrease the latency in the transmission of critical data frames. The experiment only showed a drop in the frame loss percentage in case of bus or interface failures.

For the next experiments, warning messages should be implemented regarding the interruption in the operation of interfaces or buses, the acquisition of frames will be implemented so that analysis can be made about transmission times and network behavior with larger traffic volumes, as well as other improvements applied the CAN network.

4.2.1.2 Scenario 1 - Experiment 2 - Single Sign

In this experiment, redundant interfaces and buses were used, but with a different behavior from the previous scenario. In this scenario the redundant interface and bus are idle until the controller decides to use them. This decision is based on the interface health check, a condition that is evaluated immediately after the source code infinite loop begins. Upon detecting any abnormalities in the transmitting device, which may be the interface or bus experiencing problems, the controller will use the second communication interface, ensuring that frames continue to be delivered after a brief period of switching between interfaces. This experiment only brings robustness to the network.

The methodology for the experiment is the same as described above. The device used as a transmitter transmits for 30 seconds and the interface is disconnected for 5 seconds, simulating a problem. This result will be compared to an earlier result

described above, which had its data collected on a classic CAN network without redundancies.

The following table shows the results obtained through this experiment:

Table 3 - Analysis of results obtained with dual interface

Transmitted frames	Received frames	Lost Frames	Loss percentage
274	252	22	8%
289	273	26	6%
284	265	19	7%

Source: Self authorship

This result demonstrates that the loss percentage is lower in the idle redundant interface scenario compared to the load balanced scenario. This is natural because in the balanced scenario the microcontroller tries to transmit across the idle bus, which is not the case with the redundant bus scenario, where the faulty transmission interface is disabled and the redundant interface goes live.

4.2.2 Scenario 2 - Multiple Signals

For this experiment a prototype with five microcontrollers connected through a CAN bus was assembled. Multiple signals were used, simulating an automotive communication system. As with this type of system, signals generated from one network member are used by other members for real-time information display or device triggering. The arbitration process performed on the CAN controller used handles the priorities. Each network member has the ability to decide whether or not to process a received frame. Initially a prototype was created with a single broadcast domain. In the following sections a prototype will be presented that divides the network into two segments. The signals used and their respective identifiers are described in the following board:

Board 5 - Signals and Identifiers

Signal	Identifier
Response Frame - Latency Calculation	0x42
Temperature signal	0x43
Discrete signal for relay drive	0x43
Potentiometer	0x45

Signal	Identifier
Engine speed	0x46
Lightness – LDR sensor	0x47
Temperature - simulated by potentiometer	0x48
Oil level - simulated by potentiometer	0x49
Temperature sensor error	0x50
Oil level sensor error	0x51

Source: Self authorship

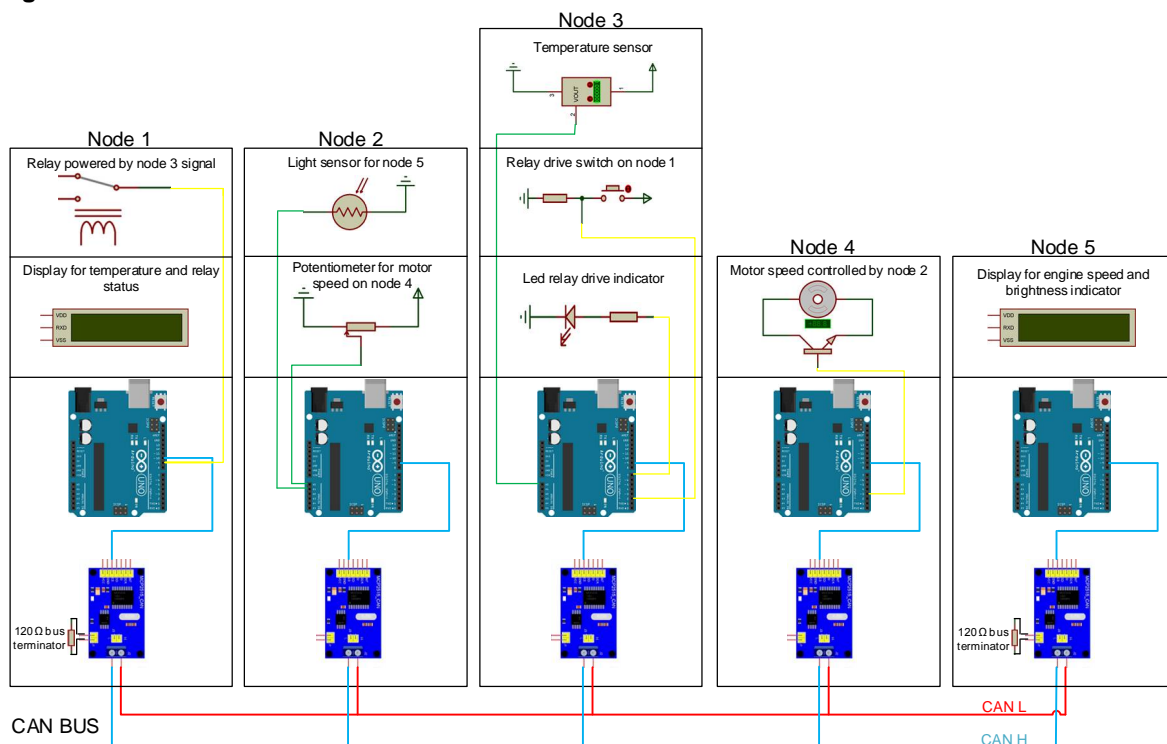
The board above shows that two signals have the same identification within the CAN network, 0x43. This was purposely created as a way of proposing a solution to a very common problem in temporally deterministic communication networks which is the issue of treating aperiodic signals within periods where cyclic periodic signals are transmitted. The above signals will be carried in standard CAN frames, which have 11-bit identifiers and are used in the arbitration process that determines the priorities for bus utilization. All signals are up to 8 bits long.

The signals described above are not used by all network members. A feature of the CAN network is the ability of network members to decide to process an incoming or outgoing frame. This decision is made using the frame field that identifies what information is being carried by that frame. For this in the source code of each member of the network a mask has been configured, which determines which identifier bits will be analyzed and a filter, which is created to be compared with the identifier bits. The mask for each receive buffer has been set to 0x111FF on all network members. This means that all identifier bits will be matched against the filter, individually configured for each member, according to the message to be received.

In the implementation in question, we exploited a feature of the library used in the project (`mcp_can.h`) that organizes the 8 bytes of CAN frame payload into an 8 position array, where each position stores 8 bits (1 byte). The proposed solution is to use one position of the vector to carry a periodic signal and another position to carry the aperiodic signal. This was implemented by using a switch connected to a microcontroller digital input called Node 3. When changing the state of the digital input used, a routine that stores this state at a vector position is executed. Thus the aperiodic signal is carried in the same frame as the periodic signal. This signal is used to drive a relay on the Node 1 microcontroller.

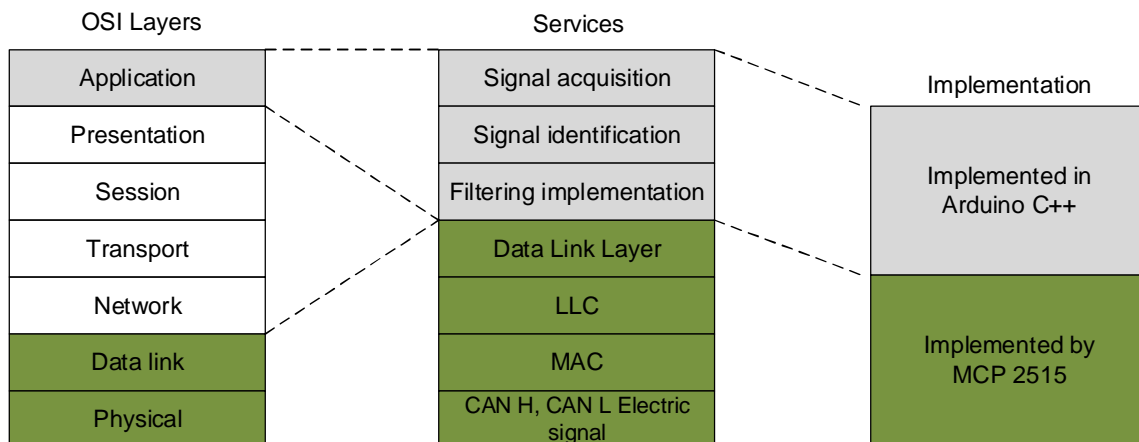
Figure 19 below illustrates the scenario created with the drivers and devices connected in peer-to-peer mode with each other. The following sections describe in detail how it works.

Figure 19 - Scenario 2



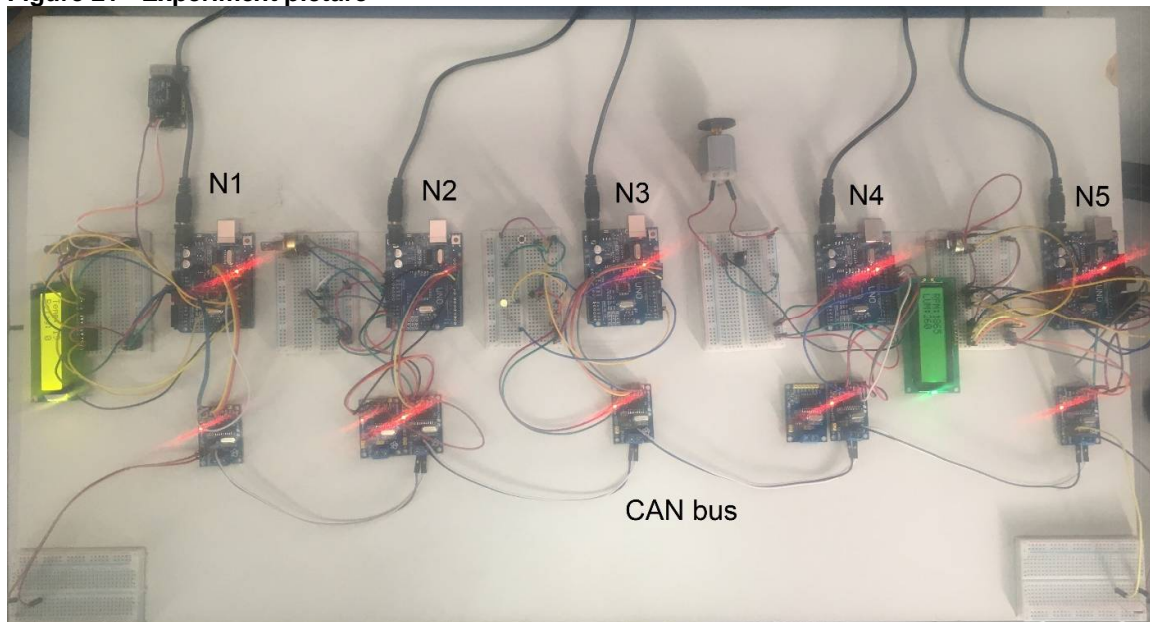
Source: Self Authorship

The driver firmware shown above was developed in Arduino's own C ++ language. Acquisitions and treatment of signals sent and signals received through the implemented CAN network were made. Issues related to the lower layers were also implemented, such as transmission vector size, identification definition and frame filtering. Figure 25 below illustrates the created model compared to the OSI reference model.

Figure 20 - OSI Model x implemented scenario

Source: Self Authorship

The following figure 21 is a real photo of the created scenario where it is possible to identify the controllers connected through the CAN bus.

Figure 21 - Experiment picture

Source: Self Authorship

In the following sections follows a detailed description of each network member created, as well as the signals that are generated and consumed.

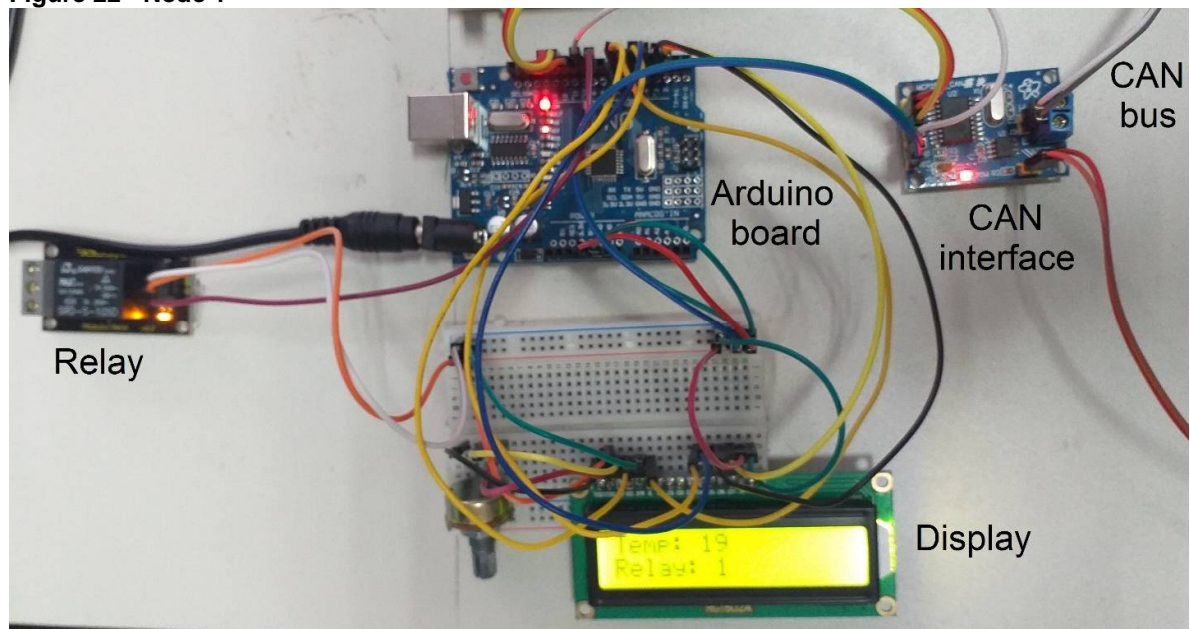
4.2.2.1 Node1

This device only consumes signals generated by other devices and made available on the CAN bus. These signals are generated by N3 and when received by N1 are displayed on an LCD display connected to it. The signals used are of ambient temperature and a discrete change signal of logic change of a digital pin, transported in the same frame, but in different positions of the data vector. As with other network members, a mask and filter have been configured so that only the desired signals are received by the controller.

The received temperature signal is equivalent to the actual ambient temperature value, measured in degrees Celsius, and is read at the first position of the vector sent within the frame identified by 0x43. Upon receipt of this signal the value is used to display the ambient temperature on an lcd display. As mentioned above, the mask for both receive buffers has been set to 0x111FF, causing all handle bits to be matched against the filter to determine whether or not the frame will be received. The filter has been set to 0x43. This will cause only the frame that has this identification to be received, and the others are discarded.

The received discrete signal is used for logic level change of digital pin 8, where a relay type device is connected. This signal is recorded at the second position of the vector sent also within the CAN frame also identified by 0x43. Because this is a discrete signal, this position stores the values 0 or 1. The controller writes this digital value to said digital pin and automatically the relay connected to that pin is switched on or off. The figure below shows Node 1.

Figure 22 - Node 1



Source: Self Authorship

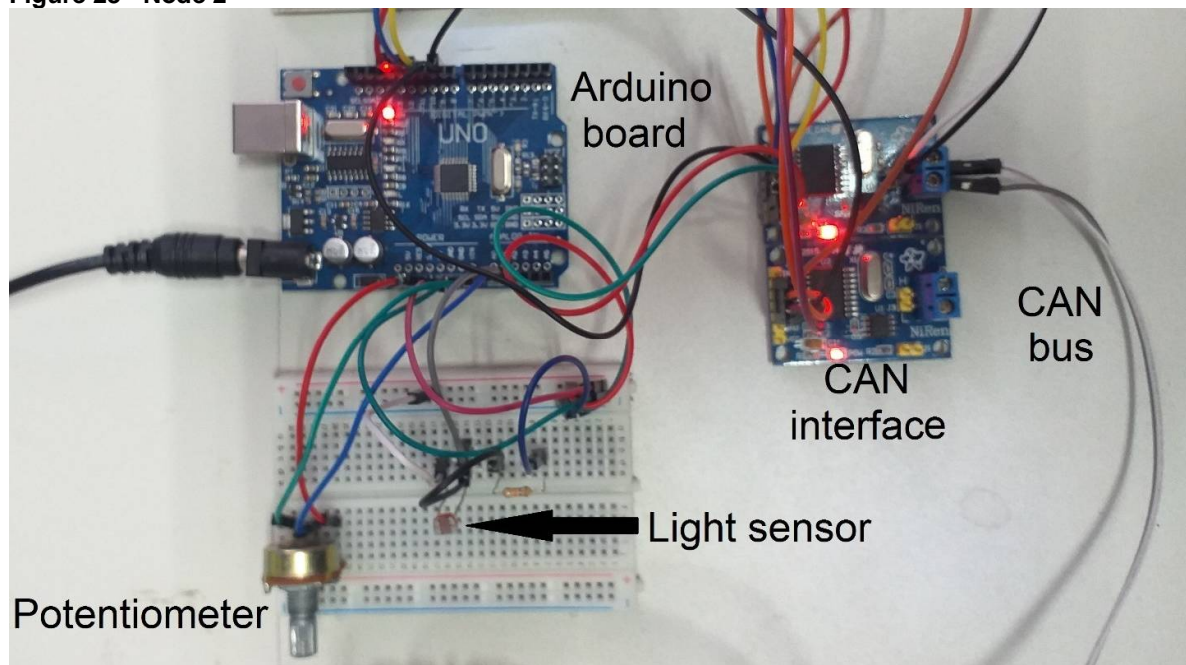
4.2.2.2 Node2

This controller has a 10k potentiometer connected to its analog pin 0. This potentiometer causes the voltage on this analog pin to vary between 0 and 5v. Having an 8-bit resolution, the controller's AD converter converts the received value to a number between 0 and 1023, where 0 equals 0v and 1023 equals 5v voltage at said analog pin. The generation of this signal aims to control the speed of a DC motor connected to Node 4, simulating the operation of an acceleration pedal.

This speed control will be done by generating a PWM signal. Due to the fact that the limit of each position of the vector transmitted through the CAN bus is 8 bits, it would not be possible to transmit the number 1023 for example, if the potentiometer was in the position with the minimum resistance. Because of this, we chose to use the Arduino C ++ language map method, which allows you to perform a mapping between two value ranges. In this case, the range from 0 to 1023 was mapped to 0 to 255 (the resolution of the Arduino digital pin that has the PWM function is 8 bits) before transmitting. This value is written to the first position of the vector and sent in a frame on the CAN bus with the id 0x45.

This controller also has an ldr-type light sensor connected to its analog input 1. The signal received at this input varies from 0 to 1023 according to the light level on the sensor. The higher the brightness the higher the value. For scanning the multiple transmit and receive buffers of the MCP2515 controller, this signal is sent on the CAN bus labeled 0x47 and is used for display on the Node 5 lcd display. Due to the data field limit this signal is also mapped before be transmitted. Values from the range 0 to 1023 are mapped to 0 to 255. The image below shows Node 2.

Figure 23 - Node 2



Source: Self Authorship

4.2.2.3 Node3

This controller is responsible for generating two signals: The first is an analog signal obtained through its analog pin 0 where an LM35 type ambient temperature sensor is connected. This sensor outputs a voltage signal that varies 0.1 volts with each one degree (C) change in ambient temperature. Its operating range is from -50°C to 150°C and the voltage signal emitted on the analog pin 0 of the controller will vary between 0 and 5v according to the measured temperature.

As with Node 2, this signal will be converted by the Arduino AD converter to a number between 0 and 1023. Since this number cannot be transmitted within one of

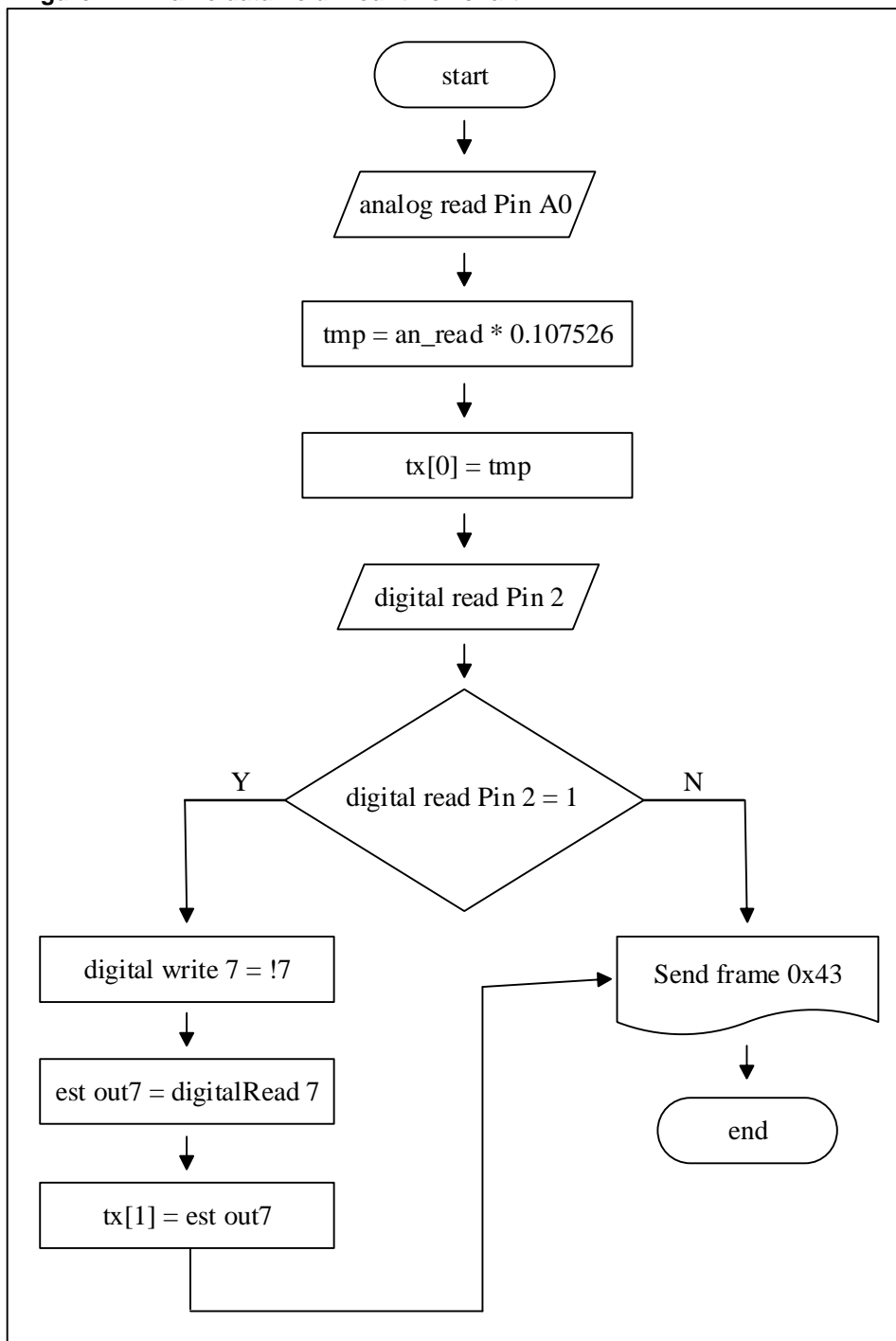
the positions of the transmission vector, the value has been converted by equation to the quantity. temperature and transmitted in position 1 of the transmission vector through the CAN bus with the id 0x43.

In addition to this analog signal, this controller receives a digital signal on its pin 7, which has its logic state changed to high when a switch is actuated. When this logic level change occurs, a routine is executed that records the digital state of said pin and transmits it in the second position of the transmission vector within the CAN frame containing the same identification. This optimizes the use of the CAN frame by transmitting two signals in a single frame.

Node 3 cyclically acquires and transmits a temperature signal through the CAN bus. This node is also responsible for acquiring and transmitting a discrete signal across the bus. Because it is transmitted cyclically, the temperature signal is considered a periodic signal as it is transmitted on the bus every 100 ms approximately. The discrete logic change signal from digital pin 7 is an aperiodic signal as it is transmitted sporadically whenever an event occurs, in which case the switch is triggered. This is one of the proposals of the work in question.

Acquisition and transmission of the generated aperiodic signal are made from a relatively simple algorithm, where the assembly of the frame with the periodic signal is not interrupted, only after the discrete signal has been loaded in the second position of the CAN frame. For better understanding and clarity, this process is described through the flowchart shown below:

Figure 24 - Frame data field mount flowchart



Source: Self Authorship.

This flowchart illustrates the assembly of frame 0x43 carrying one or two signals. Every time the logic state of pin 2 is high, the logical state of pin 7 is reversed. This function only illuminates a circuit led indicating that the relay on the remote controller has tripped. When this happens the logic state of pin 7 (0 or 1) is sent in CAN

frame 0x43 at position 1 so that it is used to change the logic level of the pin where the relay is connected to the remote controller. If the logic state of pin 2 is 0 then CAN frame 0x43 will be sent carrying only the temperature signal at its 0 position. Figure 25 below illustrates the frame data field sent through the CAN bus.

Figure 25 - Frame 0x43 with multiple signals

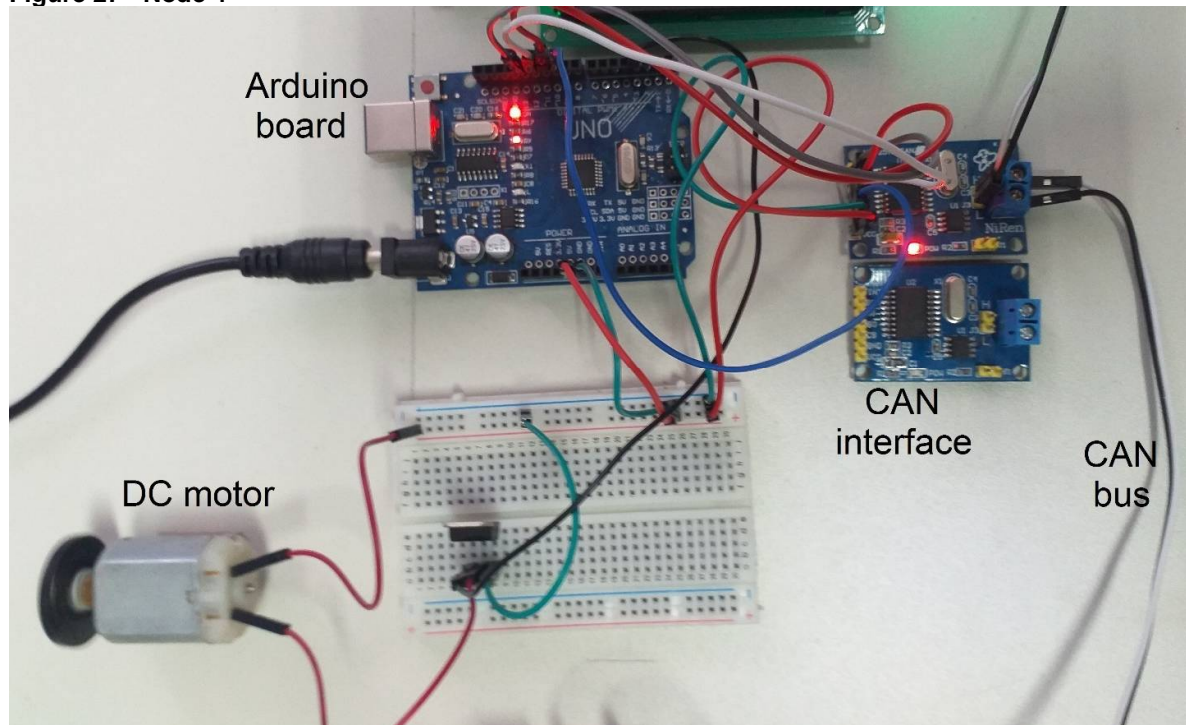
Position 0	Position 1	...	Position 7
Periodic signal. Temp Up to 8 bits	Aperiodic signal. Logic level Up to 1 bit	Not used	Not used

Source: Self Authorship

The figure below is a real photo of Node 3, where you can see the devices that make up this node, such as the relay trigger switch on node 1, the LM35 sensor that measures the ambient temperature shown on node 1 and the led which is triggered when the relay on node 1 is powered on.

id 0x46. Due to the size limit in the CAN frame data field, which is 8 bits, the maximum value to be transmitted is 255. Since the maximum motor speed is 4100 rpm, this value must be mapped before it can be transmitted in an 8 bit space. The following figure 27 is a real image of node 4.

Figure 27 - Node 4

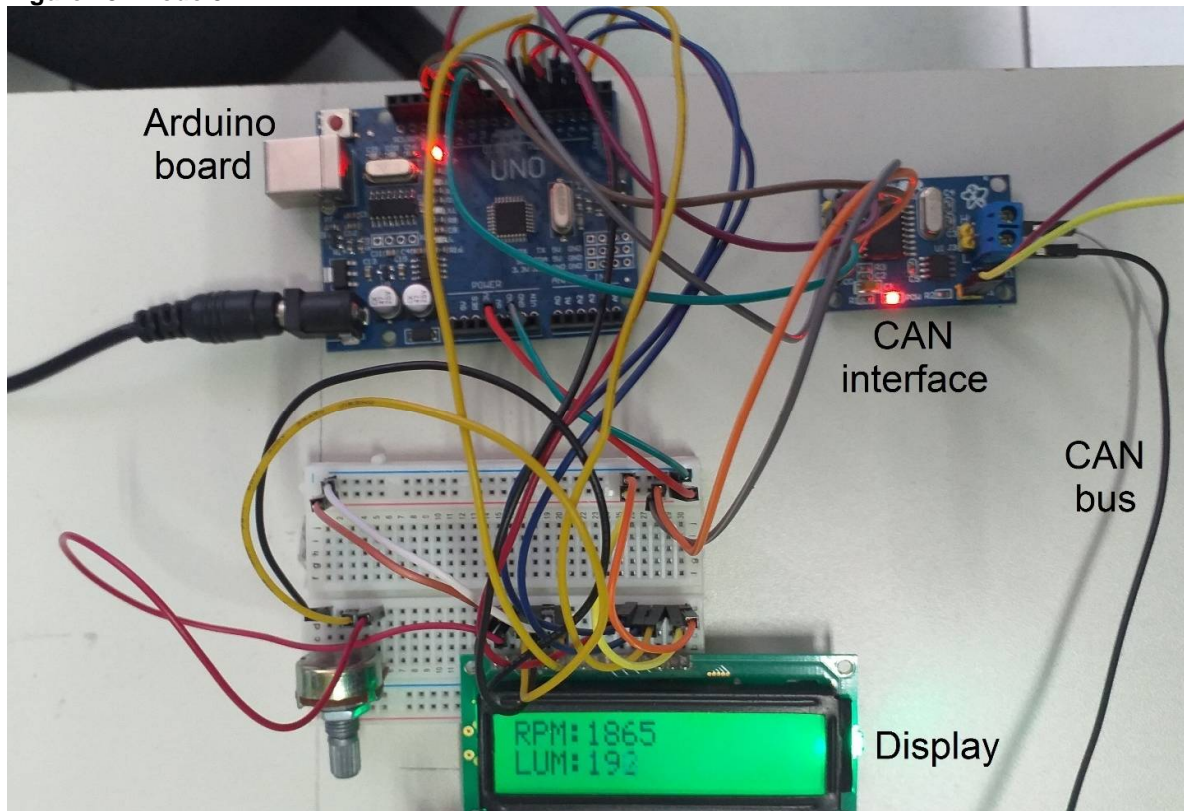


Source: Self Authorship

4.2.2.5 Node 5

This controller uses the signal identified by 0x46 to display on a lcd display the rpm speed of the DC motor connected to Node 4. This controller receives the value written on the Node 4 PWM pin and converts it through the mapping performed by the map method in a value corresponding to the engine speed. This value is then shown on said lcd display. In order to receive only the desired signal, it also had its mask set to 0x11FF and the filter set to 0x46. The following figure 28 is a real image of Node 5, where you can see the display showing the rotation of the motor connected to node 4 and the light signal transmitted by node 2.

Figure 28 - Node 5



Source: Self Authorship

To exploit the multiple receive buffers of the MCP2515 module, this controller also uses a signal identified by 0x47. This is a signal generated by a light sensor connected to Node 2. The signal is mapped to the range 0 to 255 so that it can be transmitted within an 8-bit space because the original signal is up to 10 bits in size. When the frame with this information arrives at Node 5 this mapping is undone and the signal returns to its original format, as was generated in Node 2.

4.2.2.6 Description and demonstration of transmitted signals

This section will show how the signals were generated and the result of their use by the target network member. The signals generated and consumed, in short, are an analog signal generated by a potentiometer connected to Node 2 and used by Node 4 for speed control of a DC motor, an analog signal generated by a temperature sensor LM35 at Node 3 and used to display the room temperature on a Node 1 display, and finally a proposed solution to the problem of aperiodic frame transmission next to

frames that carry periodic signals, where the logic level of an Node 3 digital pin will be changed through an external action, and when this occurs this logic level will be encapsulated in the same frame that transmits the temperature signal.

These signals were acquired using a VBS (Visual Basic Script) script that monitored the serial communication port between the computer and Arduino during the test period. The script generates a txt file with the values transmitted through the serial port of the Arduino board. After that this data is entered into a graphing application. The values received after performing conversions will be compared with the mathematical equations that define these conversions to prove the accuracy of the values used by the controllers.

4.2.2.6.1 Dc motor speed control

The signal for speed control of this motor connected to Node 4 is generated by a potentiometer connected to an analog input at N2. This potentiometer makes the voltage at this input range between 0v and 5v. The end result is that when the input is 0v the motor will be stopped at N4, for the 2.5v input the motor will be at half its speed, for the 5v input the motor will be at full speed, and so on. .

The AD converter of the controller used has a 10-bit resolution, so it will convert this input voltage to a digital value between 0 and 1023 ($2^{10} = 1024$). For example, when the voltage is 0v, the converted value will be 0, when the voltage is 2.5v, the converted value will be 511, and so on. The mathematical equation representing this conversion will be shown below, where v is the voltage present at the analog input of the controller:

$$cnv = v \times 1023 \div 5 \quad (1)$$

For an A/D conversion, if we have 2.5v of voltage at the analog input, replacing variable v in the formula above we have:

$$\begin{aligned} cnv &= 2,5 \times 1023 \div 5 \\ cnv &= 511,5 \end{aligned}$$

Due to the size available at each position of the vector transmitted within the frame on the CAN bus, which is 8 bits, the result of converting the signal received at the controller's analog input, which is from 0 to 1023, had to be turned into a value within from 0 to 255. This is appropriate as the motor speed will be controlled by pwm, and the resolution of the pin that has this function in the controller is 8 bits. The mathematical equations that represent this conversion, initially put in the patterns of a rule of 3, will be shown below:

$$\begin{array}{ccc} 255 & 1023 & \\ \text{pwm} & x & \end{array} \quad (2)$$

When calculating this rule of three we will have the equation shown below, where x is the value resulting from the AD conversion:

$$\text{pwm} = 0,249266862170088 \times x \quad (3)$$

Applying the resulting A/D conversion value found above, which is 511,5 in the formula to determine pwm we have:

$$\text{pwm} = 0,249266862170088 \times 511,5$$

$$\text{pwm} = 127,5$$

This is the mathematical representation of the conversion of a signal whose values range from 0 to 1023 to values ranging from 0 to 255. The C ++ programming language used in the Arduino IDE has the map method, which automatically converts as following example:

$$x = \text{map}(y, 0, 1023, 0, 255) \quad (4)$$

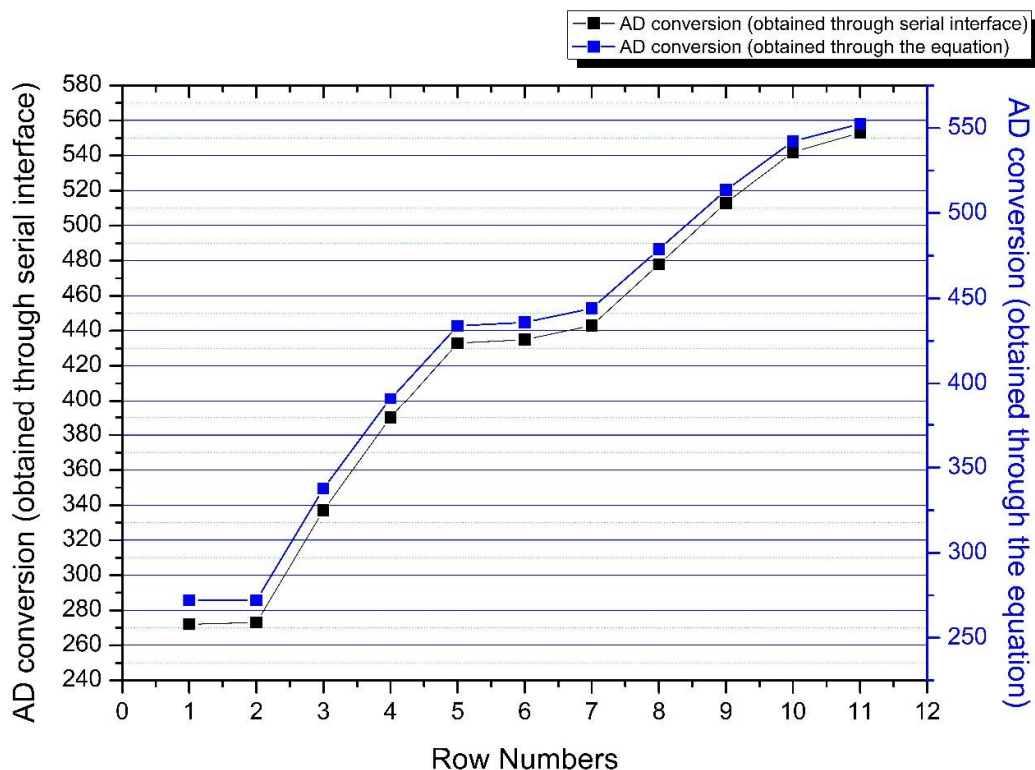
In this statement, x is the variable that will receive the value converted through the map function, which has as parameters the variable y, whose value ranges from 0 to 1023, in a number within the specified range, in this case between 0 and 255. Thus, y = 0 will have x = 0, y = 511 will have x = 127, and so on. In this case, the variable x has the duty cycle that will be used for motor speed control through pwm. This variable

is inserted at position 0 of the vector transmitted within the frame through the CAN bus with the id 0x45.

Motor speed will be measured by the relationship between the PWM signal and the maximum motor speed when the average output voltage is 5V. The maximum speed for this engine using these parameters is 4100 rpm. This value was found with a rule calculation of three, where the maximum speed with 12V, according to the engine manufacturer's manual is 10000rpm.

Figure 29 below shows the comparison chart between the value of the AD conversion obtained using a VBS script that captures data from the serial port and the value obtained through the calculation performed with the equation described above. It can be observed that the values are very close, and the difference is the fractional part of the obtained number.

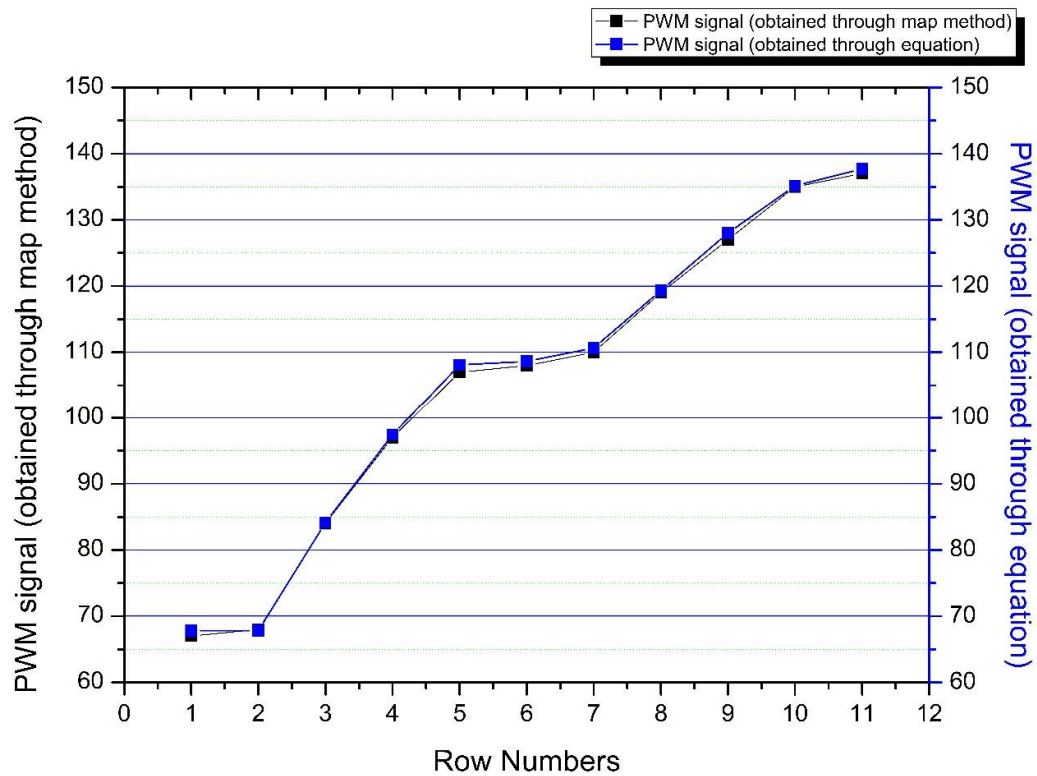
Figure 29 - Graphic AD conversion



Source: Self authorship.

Figure 30 below shows the value that will be used as a PWM signal for motor speed control, comparing the values obtained through the equation with the value obtained through the map method described above. Also in this case it can be observed that the values are very close.

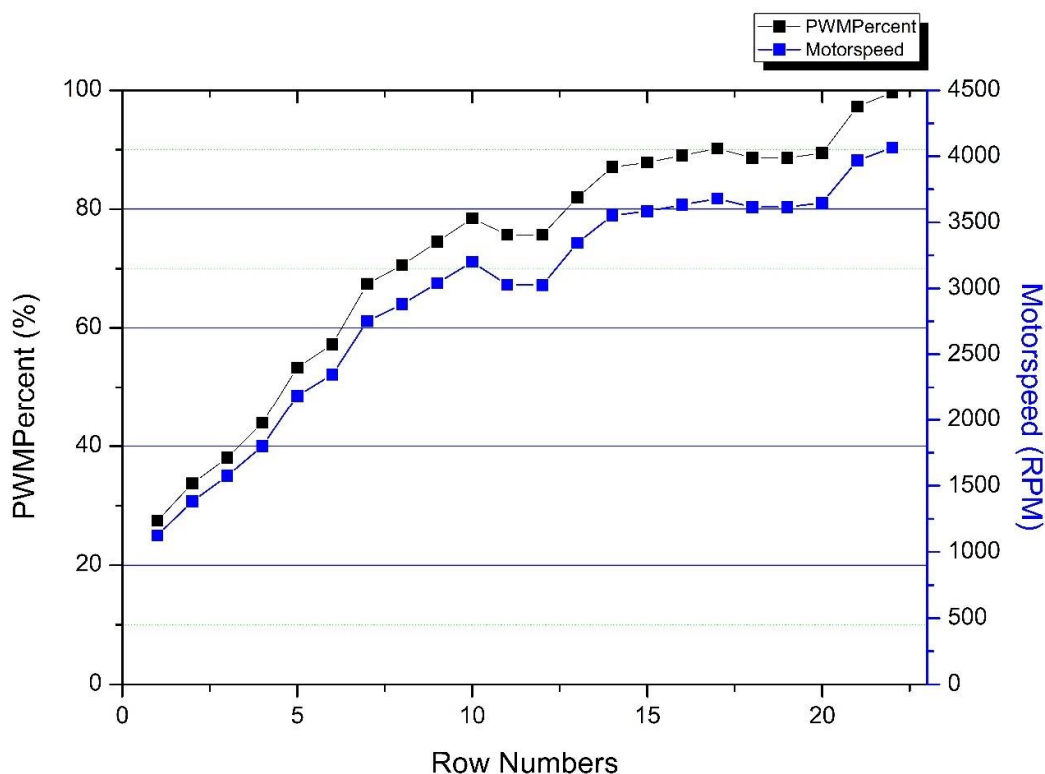
Figure 30 - Graphic Conversion to PWM signal



Source: Author (2019).

Figure 31 below shows the motor speed relative to the pwm signal.

Figure 31 - Graphic Engine speed relative to PWM



Source: Self authorship

4.2.2.6.2 Ambient temperature measurement

Node 3 has a temperature sensor type LM35, which works at temperatures between -55°C and 110°C connected to one of its analog inputs. This sensor reads the ambient temperature and outputs an analog voltage signal on one of its pins, which varies linearly with the measured temperature. The sensor changes the input voltage by 10mV for each grade C read. As with the signal described above, the Arduino AD converter will convert this voltage to a number between 0 and 1023, due to the 10-bit resolution of the converter. This value is not interesting as it does not show the temperature clearly, so you should convert it to the measured temperature. This is easily done by using a simple 3 rule as follows:

$$\frac{110 - 1023}{tmp - x} \quad (5)$$

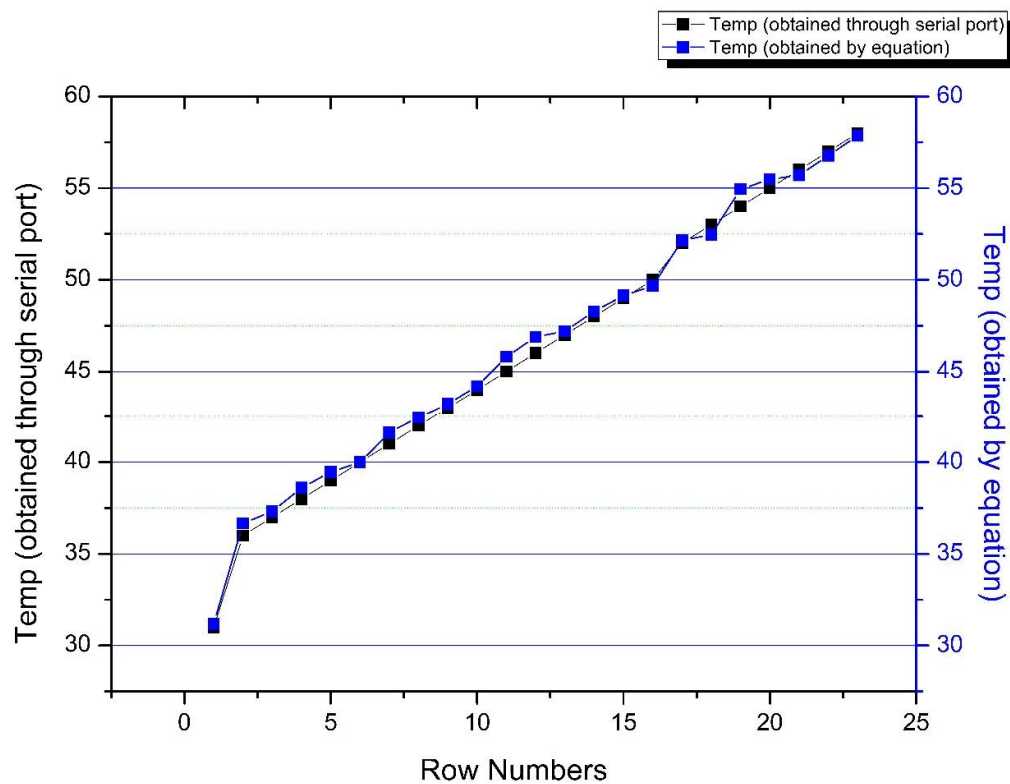
When calculating this rule of three we will have the equation shown below, where x is the value resulting from the AD conversion:

$$tmp = 0,10752688172043 \times x \quad (6)$$

For a value resulting from A/D conversion of 420, applying in the formula above we have:

$$\begin{aligned} tmp &= 0,10752688172043 \times 420 \\ tmp &= 45,1612 \end{aligned}$$

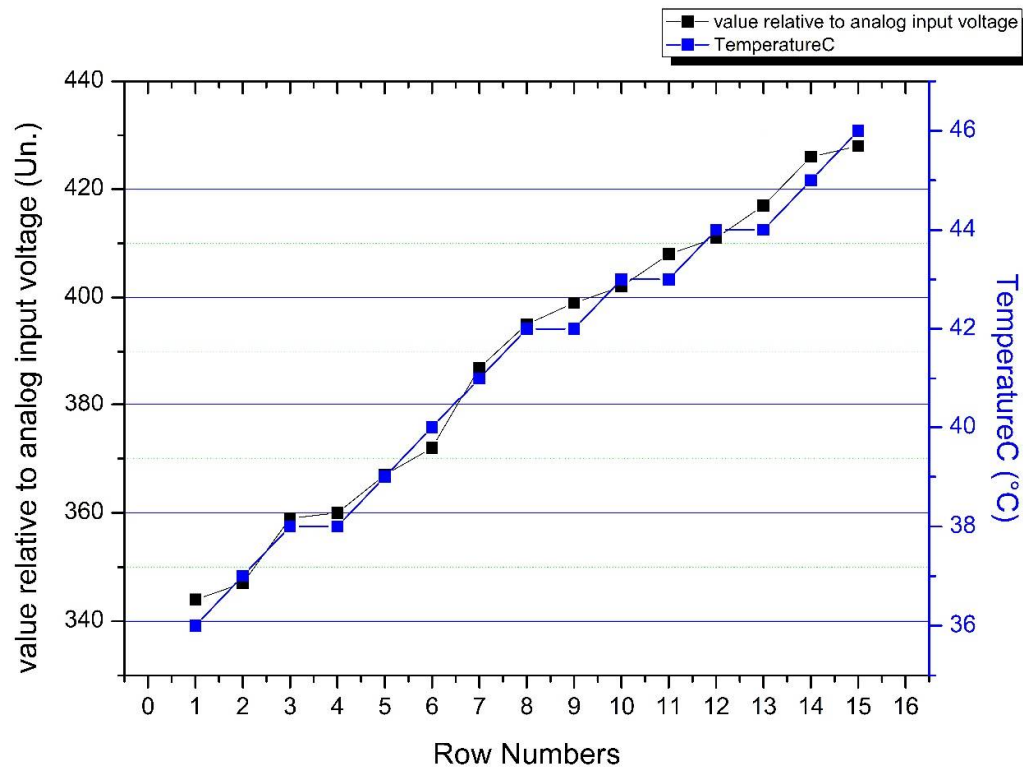
This temperature value will be sent over the CAN bus for Node 1 display. This frame will be labeled 0x43. Figure 32 below shows the comparison between the temperature value obtained by using the VBS script that captures data from the serial port and the value obtained by the calculation performed with the equation described above. It can be observed that the values are very close, and the difference is the fractional part of the obtained number.

Figure 32 - Graphic Comparison between temperature obtained and calculated

Source: Self authorship

Figure 33 below shows the measured temperature relative to the analog signal converted by the controller AD converter.

Figure 33 - Graphic Comparison between temperature and value resulting from ad conversion



Source: Self authorship

4.2.2.6.3 Light sensor signal

This signal is detected by a Light Dependent Resistor (LDR) sensor. It is a device that varies its resistance according to the luminosity that affects it. This sensor was connected to Node 2 analog input 1. According to the brightness detected in the environment, the Arduino board's AD converter generates a value between 0 and 1023 (10-bit AD converter), the higher the brightness on the sensor, the higher the value read.

This experiment was used to present a solution for a specific objective mentioned at the beginning of the work. As already mentioned, the library used to activate the MCP2515 CAN interface sends an 8-position vector with 8 bits at each position. This solution consists of carrying a signal larger than 8 bits within a vector position. When a value larger than the CAN frame data field needs to be transported, the solution implemented is to split the data into two or more frames. In the proposed

solution, only one frame is used. This was implemented using the Arduino language map method. This method maps values to larger or smaller ranges.

In the experiment, the signal generated by the LDR sensor ranges from 0 to 1023, because the Arduino AD converter is 10 bits. In order to be transmitted, this signal has been mapped to a range between 0 and 255 (8 bits), recorded at one of the transmission vector positions and transmitted over the CAN bus. Upon arriving at the destination, the opposite mapping was done. The signal within the range 0 to 255 has been mapped to the range 0 to 1023.

The equation below, obtained from a rule of three, mathematically describes how this mapping is calculated.

$$\text{mapped value} = 0,249266862170088 \times \text{an_read} \quad (7)$$

For the value resulting from the A / D conversion of 511, applying the formula above we have:

$$\begin{aligned} \text{mapped value} &= 0,249266862170088 \times 511 \\ \text{mapped value} &= 127 \end{aligned}$$

In this case the received signal value of 511 will be converted to 127 to be carried within the 8 bit field.

Upon arrival at the destination, the value mapped within the 8-bit space returns to its original format through the equation (obtained by a rule of three):

$$\text{mapped value} = 4,011 \times \text{an_read} \quad (8)$$

Reversing the mapping we have:

$$\begin{aligned} \text{mapped value} &= 4,011 \times 127 \\ \text{mapped value} &= 509 \end{aligned}$$

Because of the fractional parts in the equations, the destination values vary slightly. In this case the original value of 511 after mappings resulted in 509, an irrelevant difference of 0.391%.

The table below shows some values acquired through the Arduino serial interface on the transmitter and receiver. The values were placed side by side for easy comparison between the generated signal and the received signal after mapping and mapping reversal. The difference in each case was calculated and the average difference for the collected data.

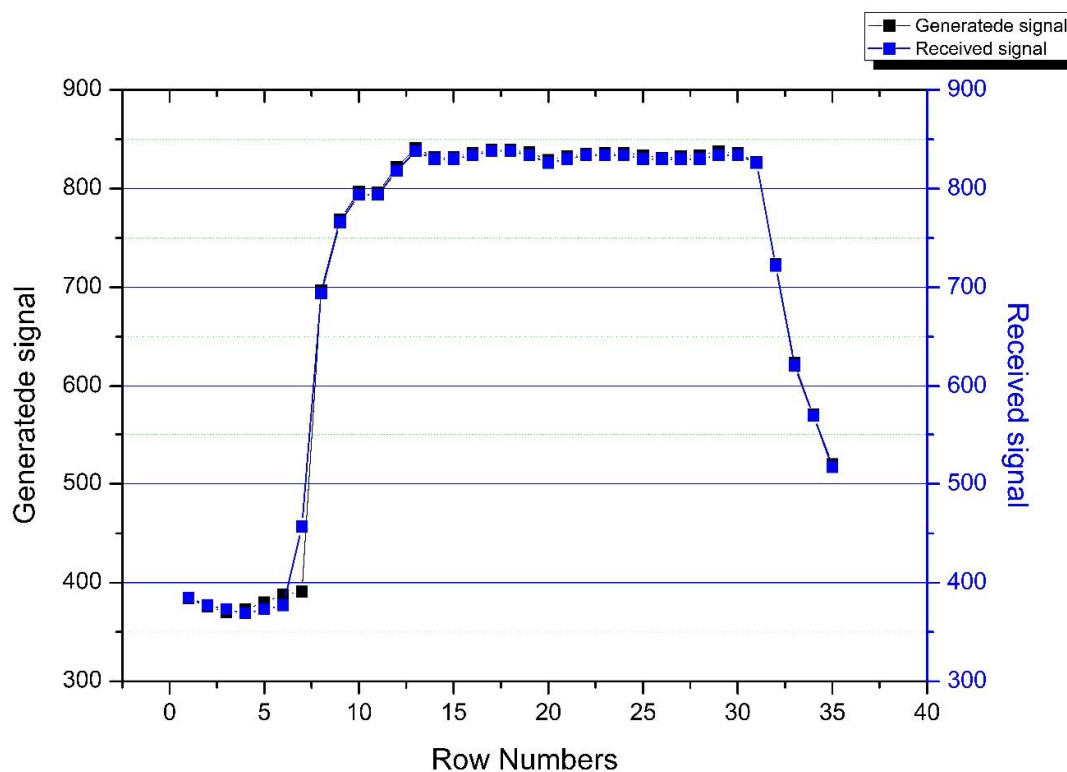
Table 4 – Reverse luminance signal mapping analysis

Signal at source	Signal at destination	% difference
697	694	0,430%
769	766	0,390%
797	794	0,376%
796	794	0,251%
822	818	0,487%
841	838	0,357%
832	830	0,240%
832	830	0,240%
836	834	0,239%
839	838	0,119%

Fonte: Self authorship

The figure 34 below shows the comparison between the brightness signal acquired on Node 2 and the signal received on Node 5 after the mapping reversal. As mentioned above, this mapping allows a signal to be transmitted within a frame smaller than its size. As in other cases this signal was acquired by running a script in VBS that monitors the serial port where the Arduino card is connected.

Figure 34 - Graphic Mapped signal



Source: Self authorship

4.2.2.6.4 Scenario 2 - Experiment 1 - Aperiodic Signal Transmission

All signals described above that are being handled in the prototype are periodic signals, transmitted at each run of the infinite loop running on the controllers. This section presents a proposal for the transmission of aperiodic signals within the tasks performed by microcontrollers during their operation, where periodic signals are being generated and where they are being transmitted.

For the demonstration, a discrete signal generated by a push button connected to a digital pin configured as input on Node 3 is being used. This push button changes the logical level of that pin. Then this new logic level is stored in a variable and then transmitted over the CAN network. Node 3 periodically transmits a temperature signal that is used by Node 1 for display. This temperature signal is stored at position 1 of the char vector transmitted over the CAN network. Said discrete signal is stored in position 2 of the vector whenever the logical level of the pin to which the push button is connected is changed.

Thus, a frame is transmitted over the CAN network with identifier 0x43, which carries two signals: At position 1 the temperature signal read by the LM35 sensor and at position 2 the discrete signal (0 or 1) that is used by Node 1 to activation of a relay connected to one of its digital pins.

4.2.2.7 Latency

The signal transmission latency of the described scenario was calculated by implementing a return message from the destination controller of a transmission. To find the transmission time of a message we used the `micros()` method, which is part of the Arduino C++ language object set. This method returns the amount of microseconds passed since the Arduino board was powered on. The methodology used to measure network latency with this method is relatively simple.

The transmitter Arduino board's operating time was written to an unsigned long variable called `time1` just before frame transmission occurred. The transmission of a response signal, an 8-bit frame, was implemented at the signal receiver. To avoid compromising latency measurement with CAN network arbitration this signal has the id 0x42, thus being the smallest identifier of the scenario. This response frame was transmitted on the CAN bus immediately upon receipt of the signal by the destination controller.

Immediately upon receipt of this response frame by the signal source controller, the Arduino board's runtime was written to an unsigned long variable called `time2` using the `micros()` method. Thus, there is the time before the signal transmission and the time after the reception of the response signal, being roughly considered as network latency the difference between `time2` and `time1`. The latency measurement test was performed between Node 3 (discrete signal and temperature transmitter) and Node 1 (discrete signal and temperature receiver) and Node 2 (potentiometer signal transmitter) and Node 4 (motor speed controller).

For purposes of analysis and verification of the results of the experiment, initially the controller operating times were recorded in `time1` and `time2` without receiving the response frame on the transmitting node. In sequence, the reception of the response frame on the transmitting node was activated and the operating times were recorded in `time1` and `time2`. Test results with latency measured in microseconds

will be shown below. One notices a difference in the latency of the first test compared to the second, obviously resulting from the reception time of the response frame.

The first test was performed with Nodes 3 and 1. Node 3 was assigned to the unsigned long time1 variable called the micros method that returns the amount of microseconds the Arduino board has run immediately before the sendMsgBuf method call that transmits the frame with ID 0x43 (temperature signal and discrete signal). On Node 1 immediately after receiving the frame through the readMsgBuf method, the response frame was transmitted to Node 3 with the id 0x42. Immediately after receiving this frame in Node 3 (confirmed by printing the value received on the serial interface of the Arduino board), the unsigned long time 2 variable was called the micros method, resulting in the difference between time2 and time1 a. approximate latency of transmission.

The average latency obtained between Node 1 and Node 3 transmissions after approximately 500 frames were transmitted between the two devices was 413 μ s and a 6.5% difference from the time taken without response frame reception. The latency obtained between Nodes 2 and 4 was approximately 395 μ s for the same number of frames, being the time difference between the test without receiving the response frame and receiving this frame of 16%.

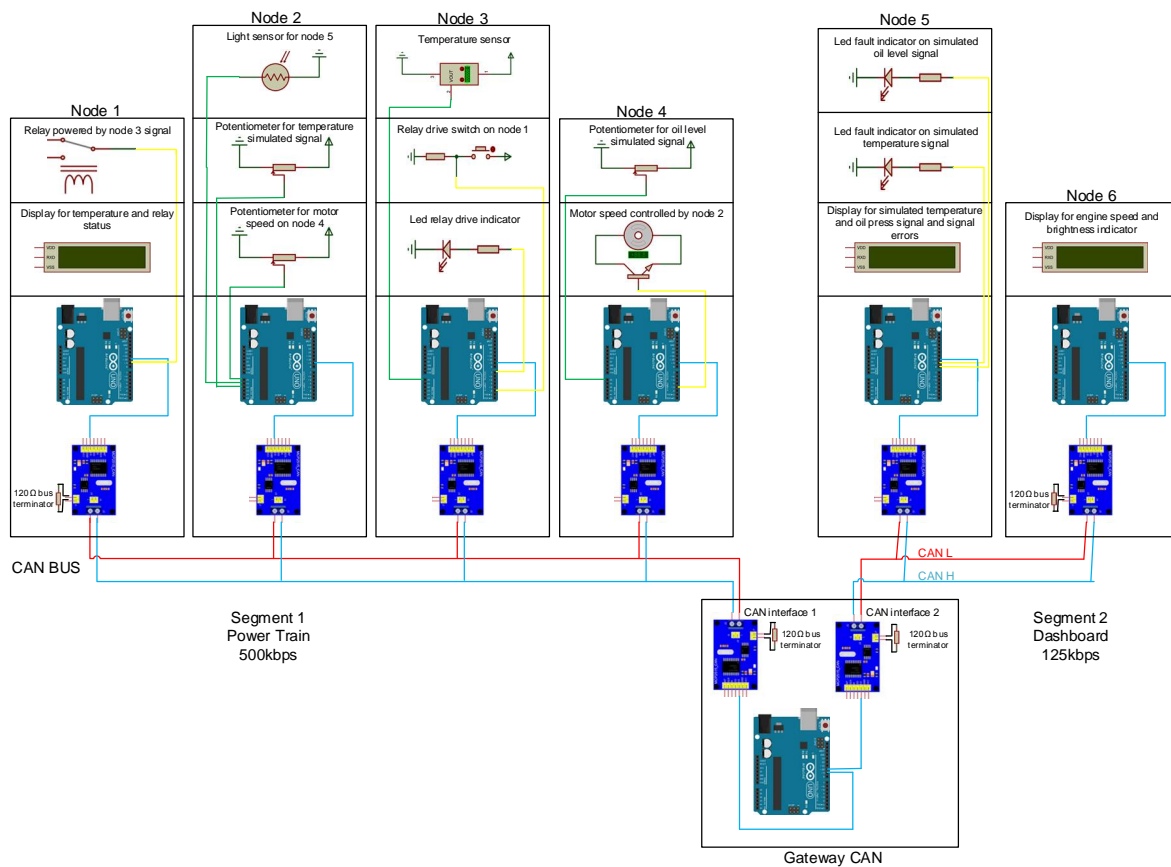
4.2.3 Scenario 3

This test scenario is an update of scenario 2. In addition to the signals used in scenario 2, two more were implemented. A simulated temperature signal from a potentiometer connected to Node 2 and an oil level signal simulated from a potentiometer connected to Node 4. These signals have been implemented to simulate sensor failure situations, as will be shown below.

Another implementation made in the scenario was the use of a new Arduino board with two CAN interfaces, dividing the scenario into two domains, one called Power Train and the other Dashboard, simulating real automotive communication systems. The purpose of this mechanism in the scenario is to reduce the size of the CAN network broadcast domain by creating two network segments. The new node added will act as a bridge, allowing only the frames used on the other bus, thus reducing unnecessary network traffic and the consumption of computational resources on CAN interfaces to decide whether or not to process a received frame. Frame forwarding from one segment to another is based on its identification.

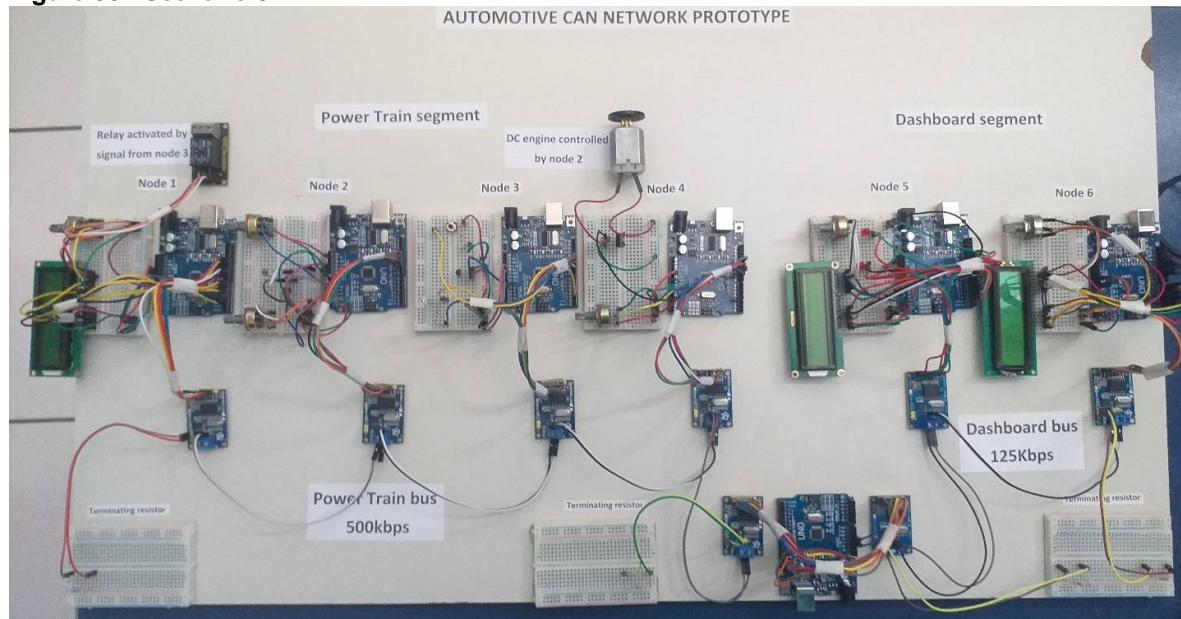
When a frame is received by interface 1, connected to the Power Train segment, it will only be forwarded to the Dashboard segment through interface 2 if the signal carried on the frame is required for any microcontroller in that segment. Otherwise this frame will be discarded. The same process happens if a frame is received by interface 2. In the example in question, the temperature signal and the discrete signal generated on node 3 will not be forwarded to the Dashboard segment, since in this segment no controller uses this signal. Figure 35 below illustrates this scenario.

Figure 35 - Scenario 3



Source: Self authorship

Image 36 below is a real photo of the created scenario:

Figure 36 - Scenario 3

Source: Self authorship

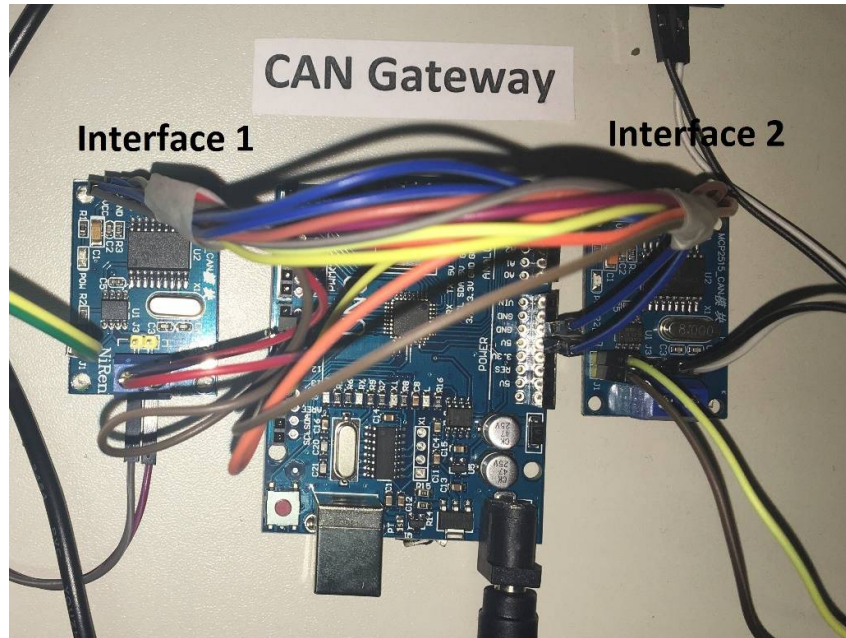
4.2.3.1 Gateway CAN

The CAN gateway was implemented using two MCP2515 interfaces connected to an Arduino board. For operation of both interfaces, in the source code of the gateway were created instances CAN0 and CAN1, associated with digital pins 9 and 10 of the Arduino board, respectively. The CAN0 instance was used by the segment named Power Train and the CAN1 instance by the segment named Dashboard. The configured filters followed the same pattern as the other experiments. Mask set to 0x111FF and filters to receive only frames that will be sent by the second interface to the other bus.

When receiving frames with the identifications 0x47, 0x48 and 0x49 (light signal, temperature simulated signal and oil level simulated signal, respectively) on the CAN0 interface (digital pin 9) the controller forwards the same ID to the CAN1 interface (digital pin 10), connected to the dashboard. The luminance values and the simulated temperature and oil level values are displayed on the displays at nodes 5 and 6. In the work in question no frame of the dashboard segment is used in the power train segment. Frame transmission from CAN1 interface to CAN0 interface was implemented only in the latency test, where a frame with id 0x42 is transmitted from

the frame's target node. The image below is a real photo of the CAN gateway implemented in the prototype.

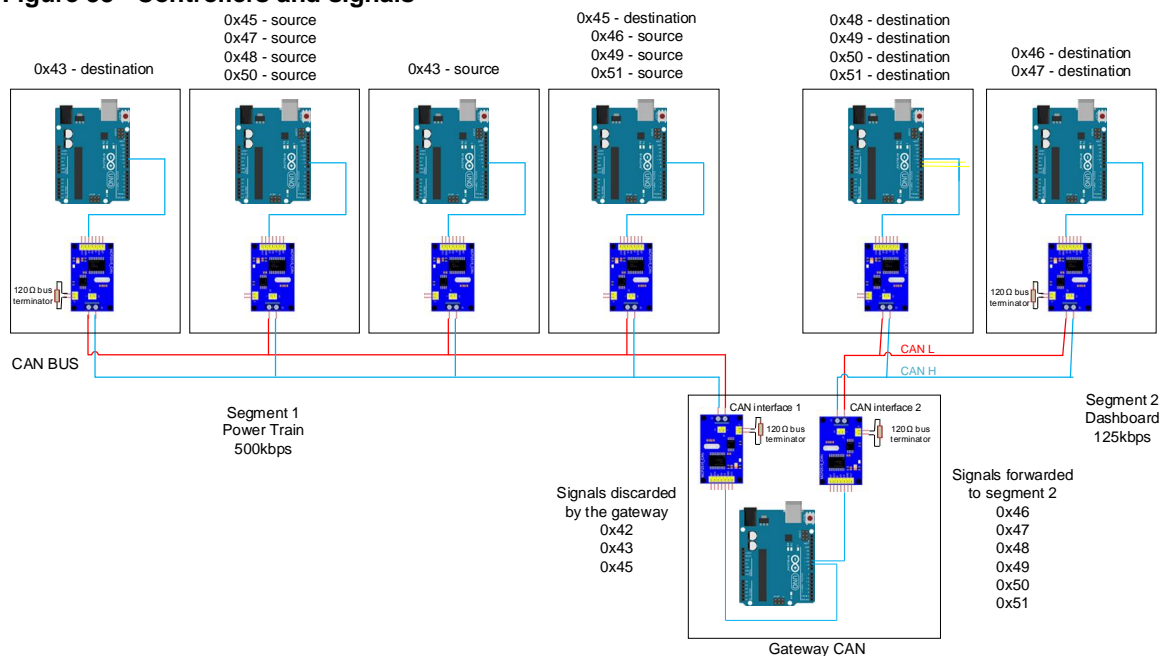
Figure 37 - Gateway CAN



Source: Self authorship

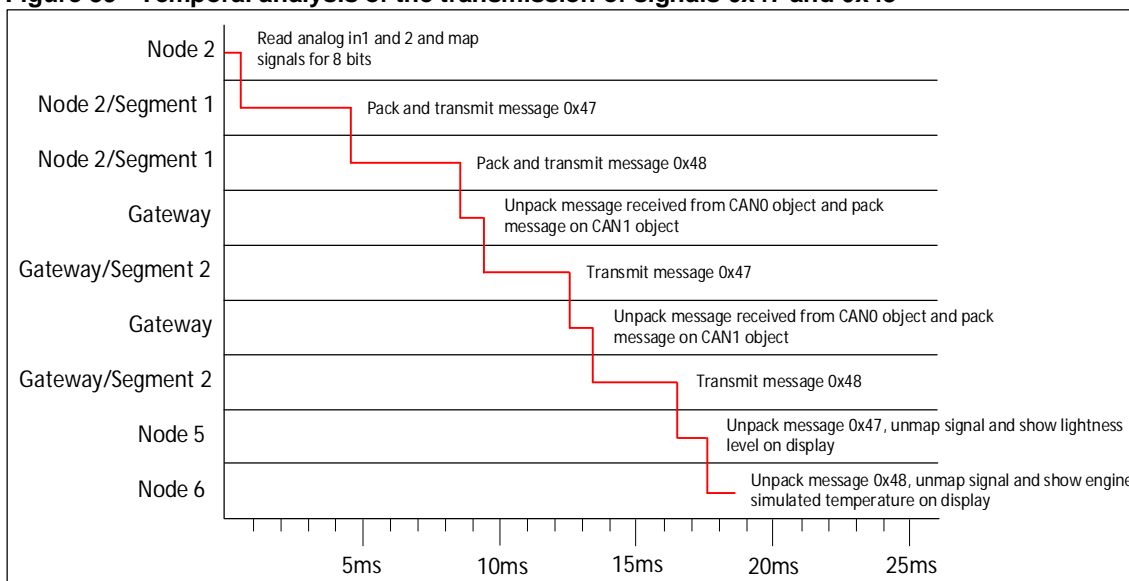
The following image shows all the signals used in the prototype in their respective controllers. It also describes which signals are discarded by the gateway and which are forwarded for use in segment 2 controllers, called a dashboard.

Figure 38 - Controllers and signals



Source: Self authorship

The following figure illustrates a graph of temporal analysis of the transmission of the 0x47 signal (luminance signal) and the 0x48 signal (simulated temperature signal), both from node 2, since the acquisition. The signals arrive at the gateway, which forwards them to segment 2 and is destined for nodes 6 and 5, respectively, which displays them on a liquid crystal display, simulating a vehicle's instrument panel. The times were obtained using the `mills()` method of the C++ language of the Arduino platform.

Figure 39 - Temporal analysis of the transmission of signals 0x47 and 0x48

Source: Self authorship

4.2.3.2 Simulated temperature signal

This signal is generated by a potentiometer connected to analog pin 2 of node 2. The purpose of this mechanism is by adjusting the potentiometer signal to simulate the operation of linear temperature signal sensor to the input voltage on the pin. The simulated temperature signal ranges from 0 to 140°, linearly to the voltage of 1 to 4v injected in the pin 2 of the controller by adjusting the potentiometer. Voltages less than 1v or greater than 4v are interpreted as error in reading the signal. If the value resulting from the pin 2 read conversion d is less than 205 or greater than 819 (less than 1v or greater than 4v, respectively), a faulty reading frame of the temperature reading 0x50 is transmitted on the network. This signal is relayed by the CAN gateway to the dashboard segment, where node 5 displays a read error message and triggers a digital output that has a red led attached to it, generating a visual alert of readout on the temperature sensor.

If the voltage value read on pin 2 is within the values considered in the simulation to be normal according to the read temperature, the value resulting from the ad conversion is mapped to a range between 0 and 255 so that it can be transmitted within the range. 8 bits of CAN frame. This signal is transmitted on the network with the ID 0x48 for display on node 5. As node 5 is in the dashboard segment, this signal

is received by the gateway through the interface connected to the power train segment and relayed to the dashboard segment for display. When received on node 5 the 8-bit value is converted by equation to the simulated temperature value within the range between 0 and 140°C and displayed on that node's display.

4.2.3.3 Simulated oil level signal

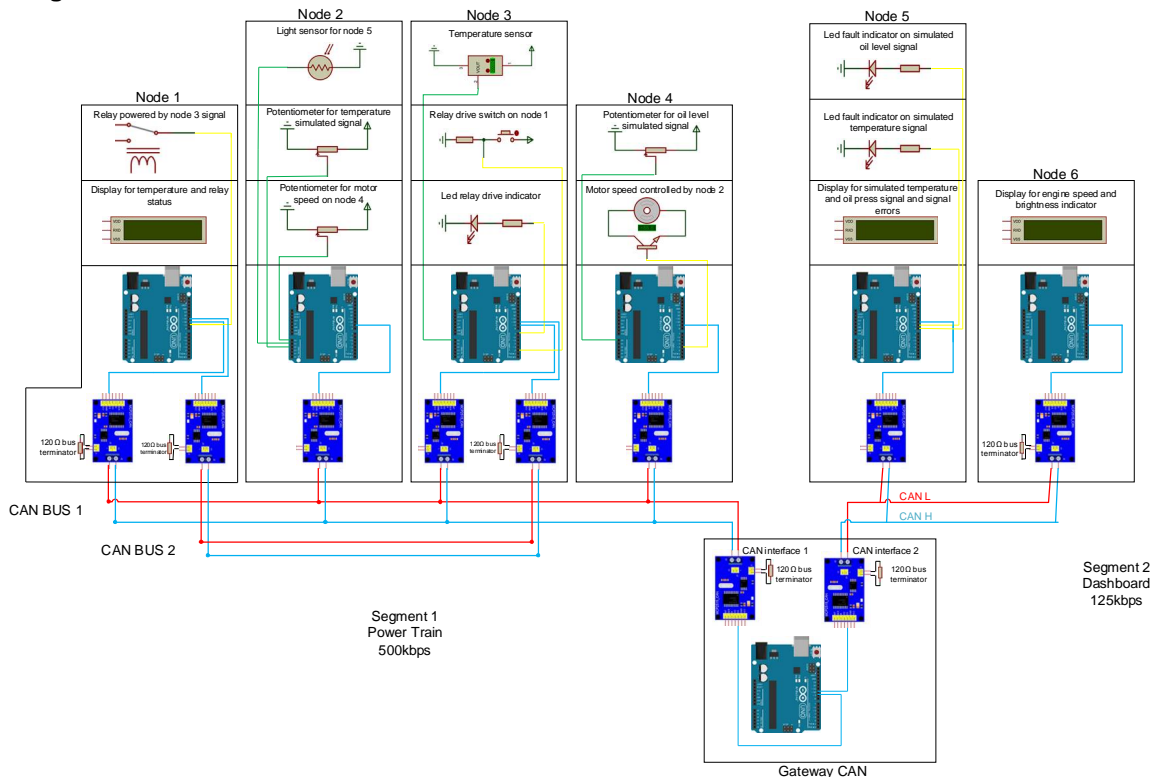
This signal is generated by a potentiometer connected to analog pin 0 of node 4. The purpose of this mechanism is by adjusting the potentiometer signal to simulate the sensor operation of a linear oil level signal to the input voltage on the pin. The simulated oil level signal ranges from 0 to 100%, linearly to the 1 to 4v voltage injected into pin 0 of the controller by adjusting the potentiometer. Voltages less than 1v or greater than 4v are interpreted as error in reading the signal. If the value resulting from the pin 2 read conversion ad is less than 205 or greater than 819 (less than 1v or greater than 4v, respectively), a temperature reading failure frame with the identification 0x51 is transmitted on the network. This signal is relayed by the CAN gateway to the dashboard segment, where node 5 displays a read error message and triggers a digital output that has a red led connected to it, generating a visual readout alert on the level sensor oil.

If the voltage value read from pin 0 is within the values considered in the simulation to be normal according to the oil level, the value resulting from the ad conversion is mapped to a range between 0 and 255 so that it can be transmitted within space. 8-bit CAN frame. This signal is transmitted on the network with the id 0x49 for display on node 5. As node 5 is in the dashboard segment, this signal is received by the gateway through the interface connected to the power train segment and relayed to the dashboard segment for display. When received on node 5 the 8-bit value is converted by equation to the simulated oil level value, which varies from 0 to 100% and displayed on that node's display.

4.2.4 Scenario 4

This scenario is an update from scenario 3. This update proposes to use a second CAN bus in the Power Train segment. This second bus aims to eliminate a known problem in starvation computing. In CAN networks this problem causes one node on the network to monopolize bus usage and prevent other nodes from transmitting their frames. This is a likely problem in the created scenario, where the relay remote trigger aperiodic signal is triggered by a sporadic event. Precisely because it has this characteristic, if this event occurs repeatedly, the frame containing this signal will be transmitted repeatedly on the network, preventing lower priority signals from using the bus. Figure 38 below illustrates this scenario.

Figure 40 - Scenario 4



Source: Self authorship

The proposed solution to the problem of monopolizing bus use by an aperiodic signal was to implement a second CAN interface on nodes 1 and 3. This second interface is connected to a second bus, called CAN BUS 2. In the source code of nodes involved in this process, CAN0 and CAN1 instances associated with digital pins 9 and

10 and connected to bus CAN 1 and CAN 2, respectively, were implemented. On node 3, when transmitting the aperiodic signal, within the signal generation routine, the CAN1 instance selection was implemented to use the CAN 2 bus. On node 1, inside the infinite loop the secondary interface buffer is read for verification. of the arrival of the frame with the aperiodic sign.

5 CONCLUSION

It has been shown that with the suggested implementations, the optimization of the CAN frame data field as well as the optimization of the CAN bus has been achieved, as the work in question demonstrates a way to transport several signals within the same frame at a cost computationally low, since the routine that stores the aperiodic signal in the vector to be transmitted consumes only one assignment and one comparison at best (when the logical level is not changed), and five basic operations (assignments and comparisons) at worst, which this is when an event occurs and the signal needs to be stored in the vector that will be transmitted over the CAN network.

This way, a better use can be made in using the frame transported through the bus. In the prototype in question only two bytes were used, one for each signal, but the concept shows that all positions of the vector can be filled and the frame sent with up to eight signals. This is one of the characteristics of the new CAN approach, called CAN-FD, but in the present work a similar result was obtained with the use of the classic CAN network.

In terms of numbers two factors can be cited. Considering the 1-byte signal transmission and the maximum capacity of the CAN frame, which is 8 bytes, to carry both signals, 25% of the total payload is used, compared to 12.5% of use for transporting one signal only. 100% utilization of the frame data field is achieved if the vector to be transmitted is created with only 2 bytes, which is the one implemented in this case. In this case there is still the fact that smaller frames can be transmitted. Smaller frames will be transmitted in a shorter time and will consume less bus.

Considering this with respect to the total size of a CAN frame, which is 128 bits for CAN 2.0A, for transmitting a 1 byte signal, only 6% of the frame is payload. When carrying multiple signals this number can improve to 50% payload in a frame. It can also be mentioned the fact of the ratio of 1 signal to 128 bits, with a common implementation for 8 signals to 128 bits using the proposal.

The proposed signal mapping, as shown in the graphs, also proved to be efficient for the case of the need to transmit signals that are more than 8 bits long, in this case the signal used for dc motor speed control, the speed signal. of the dc motor

and the luminance signal, all larger than 8 bits were adequately adapted to be transmitted and after transmission were very close to the original analog value.

The implementation of the CAN gateway has also proved to be an efficient solution for reducing the CAN bus and reducing the consumption of computational resources used by the controllers to make a decision whether or not to process a frame. The data shown above also shows that the increase in latency as a result of frames passing through the gateway was only 4%.

5.1 FUTURE WORKS

Future work can be developed using CAN FD-compliant interfaces, because in this new version of the CAN network up to 64 bytes of data can be transmitted and the way of organization of this data field proposed in the work proved to be very efficient and low computational cost.

Dynamic identification of frames transmitted over the CAN network is also an interesting research option. In this case it will be necessary to develop an algorithm that identifies when information is required within the network. The controller that generates this signal may temporarily change its identification and send it over the network with an identification that makes it a priority over other signals used.

Flex CAN is also an interesting proposition for proposing the use of redundant controllers and buses, as well as meeting temporal requirements at the application layer.

The MCP2515 controller, according to its data sheet has the clock source function for other devices. In the future this controller property will be exploited to associate the ability to send multiple signals in a single frame to a temporally deterministic network.

REFERENCES

AL-HASHIMI, B. M. **System-on-Chip: Next Generation Electronics: Materials, circuits and devices**. 2. ed. London: The Institution of Electrical Engineers, 2006.

AN, H. S.; JEON, J. W. Analysis of CAN FD to CAN message routing method for CAN FD and CAN gateway. In: CONTROL, AUTOMATION AND SYSTEMS (ICCAS), 17., 2017, Jeju. **Proceedings...** Jeju: IEEE, 2017. p. 528 - 533.

Arduino Home. Arduino - Introduction. **What is Arduino**. 1 jan. 2007. Available in <<https://www.arduino.cc/en/Guide/Introduction>>. Access in: 20 jan. 2019.

BARHAM, B. L., Foltz, J. D., & Prager, D. L. Making time for science. **Research Policy**, Eindhoven, v. 43, n. 1, p. 21–31, feb. 2014.

BAUMGART, S. et al. Variability management in product lines of safety critical embedded systems. In: Embedded Systems (ICES). 10., 2014, Coimbatore. **Proceedings...** Coimbatore: IEEE, 2014. p. 98-103.

BHUPATIRAJU, S. et al. Knowledge flows: analyzing the core literature of innovation, entrepreneurship and science and technology studies. **Research Policy**, Eindhoven, v. 41, n. 7, p. 1121-1282, set. 2012.

BORDOLOI, U. D.; SAMII, S. The frame packing problem for CAN-FD. In: REAL-TIME SYSTEMS SYMPOSIUM (RTSS). 12., 2014, Rome. **Proceedings...** Rome. IEEE, 2014. p. 284-293.

BRUNNER, S. et al. Automotive E/E-architecture enhancements by usage of ethernet TSN. In: Intelligent Solutions. In: EMBEDDED SYSTEMS (WISES). 13., 2017, Hamburg. **Proceedings...** Hamburg. IEEE, 2017. p. 9-13.

Can Newsletter. Can Newsletter Online. **CAN-FD and the CRC issue**. 01 mar. 2015. Available in <https://can-newsletter.org/uploads/media/raw/e6cbd155b5f394b2d9edefed962ce0eb.pdf>, Access in 10 out. 2018.

CiA – CAN in Automation. CAN physical layer. **CAN knowledge**. Available in <<https://canopen.org/can-knowledge/can/systemdesign-can-physicallayer/>>, Access in: 01 out. 2018.

DAVIS, R. I. et al. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. **Real-Time Systems**, Zurich, v. 35, n. 3, p. 239-272, 2007.

DE ANDRADE, R. et al. Analytical and Experimental Performance Evaluations of CAN-FD Bus. **IEEE Access**, Vancouver, v. 6, p. 21287-21295, 2018.

DUNN, W. R. Designing safety-critical computer systems. **Computer**, Vancouver, v. 36, n. 11, p. 40-46, 2003.

EINSPIELER, S.; STEINWENDER, B.; ELMENREICH, W. Integrating time-triggered and event-triggered traffic in a hard real-time system. In: 2018 IEEE INDUSTRIAL CYBER-PHYSICAL SYSTEMS (ICPS). 1., 2018, St. Petersburg. **Proceedings...** St. Petersburg. IEEE, 2018. p. 122-128.

Electronics Hub. Arduino MCP2515 CAN Bus Interface. **CAN Protocol**. 22 aug. 2018. Available in <<https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/>>, Access in: 20 jan. 2019.

HARTWICH, F. et al. CAN with Flexible data-rate. In: IEEE INTERNATIONAL CONFERENCE OF COMMUNICATIONS, 1., 2012, Ottawa. **Proceedings...** Ottawa: IEEE, 2012. p. 1-9.

ITAMI, Y.; ISHIGOOKA, T.; YOKOYAMA, T. A Distributed Computing Environment for Embedded Control Systems with Time-Triggered and Event-Triggered Processing. In: EMBEDDED AND REAL-TIME COMPUTING SYSTEMS AND APPLICATIONS, 14., 2008, Kaohsiung. **Proceedings...** Kaohsiung: IEEE, 2008. p. 45-54.

KORNECKI, A.; ZALEWSKI, J. Safety assurance for safety-critical embedded systems: Qualification of tools for complex electronic hardware. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY, 1., 2008, Gdansk. **Proceedings...** Gdansk: IEEE, 2008. p. 1-4.

LEE, E. A. What is Real-Time Computing? A Personal View. **IEEE Design & Test**, Karlsruhe, v. 24, n. 1, p. 64-72, 2017.

LEE, E. A. What's ahead for embedded software? **Computer**, Vancouver, v. 33, n. 9, p. 18-26, 2000.

LUGLI, A. B.; SANTOS, M. M. D. **Sistemas Fieldbus para Automação Industrial–DeviceNET, CANopen, SDS e Ethernet**. 1. Ed. São Paulo: Editora Érica, 2009.

MAKOWITZ, R.; TEMPLE, C. FlexRay-a communication network for automotive control systems. In: INTERNATIONAL WORKSHOP ON FACTORY COMMUNICATION SYSTEMS, 2., 2006, Torino, **Proceedings...** Torino: IEEE, 2006. p. 207-212.

MORENO, C.; FISCHMEISTER, S. On the Security of Safety-critical Embedded Systems: Who Watches the Watchers? Who Reprograms the Watchers? In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS SECURITY AND PRIVACY (ICISSP). 3., 2017, Porto, **Proceedings...** Porto: University of Waterloo, 2017. p. 493-498.

National Instruments. Controller Area Network (CAN) Overview. **Understanding CAN with Flexible Data-Rate (CAN FD)**. 05 mar. 2019. Available in <<http://www.ni.com/white-paper/52288/en/>>, Access in: 28 set. 2019.

National Instruments. Flex Ray. **FlexRay Automotive Communication Bus Overview**. 28 may 2019. Available in <<http://www.ni.com/white-paper/3352/en/>>, Access in: 28 set. 2019.

OH, J. H.; WI, J. U.; LEE, S. E. Design of CAN-CAN FD bridge for in-vehicle network. In: SOC DESIGN CONFERENCE (ISOCC), 10., 2017, Seoul. **Proceedings...** Seoul, 2017: IEEE, 2017. p. 312-313.

PIMENTEL, J. R. Designing safety-critical systems: A Convergence of Technologies. In: SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 22., 2003, Florence. **Proceedings...** Florence, 2003: IEEE, 2003. p. 1-3.

PAGANI, R. N.; KOVALESKI, J. L.; RESENDE, L. M. *Methodi Ordinatio*: a proposed methodology to select and rank relevant scientific papers encompassing the impact factor, number of citation, and year of publication. **Scientometrics**, Budapest, v. 105, n. 3, p. 2109-2135, 2015.

SALUNKHE, A. A.; KAMBLE, P. P.; JADHAV, R. Design and implementation of CAN bus protocol for monitoring vehicle parameters. In: RECENT TRENDS IN ELECTRONICS, INFORMATION & COMMUNICATION TECHNOLOGY, 1., 2016, Bangalore. **Proceedings...** Bangalore, 2016: IEEE, 2016. p. 301-304.

SCHULER, F.; SCHRÖDER-PREIKSCHAT, W. The RTSC: Leveraging the migration from event-triggered to time-triggered systems. In: OBJECT/COMPONENT/SERVICE-ORIENTED REAL-TIME DISTRIBUTED COMPUTING, 13., 2010, Carmona. **Proceedings...** Carmona, 2010: IEEE, 2010. p. 34-41.

SMALL, H.; BOYACK, K. W.; KLAVANS, R. (2014). Identifying emerging topics in science and technology. **Research Policy**, Eindhoven, v. 43, n. 8, p. 1450–1467, 2014.

Texas Instruments. Introduction to the Controller Area Network (CAN). **Application Report**. 1 Aug. 2008. Available in <<http://www.ti.com/lit/an/sloa101b/sloa101b.pdf>>, Access in: 05 may. 2019.

VAN R.; PA J.; FERREIRA, H. C. Automotive power-line communications: Favourable topology for future automotive electronic trends. In: INTERNATIONAL SYMPOSIUM ON POWER-LINE COMMUNICATIONS AND ITS APPLICATIONS, 7., 2003, Kyoto. **Proceedings...** Kyoto, 2003: IEEE. p. 103-108.

Vector E-Learning. Vector Informatik GMBH. **Introduction to CAN | Distinguishing CAN from CAN FD Frames**. 27 apr. 2018. Available in <<https://elearning.vector.com/mod/page/view.php?id=364/>>, Access in: 09 out. 2018.

WETHERALL, D. J.; TANEMBAUM, A. **Redes de Computadores**. 5. Ed. Rio de Janeiro: Pearson Education, 2011.

Woodside Capital Partners. WCP Publishes. **ADAS/Autonomous Sensing Industry: Beyond the Headlights Report**. 27 sep. 2016. Available in <<http://www.woodsidecap.com/wcp-publishes-adasautonomous-sensing-industry-beyond-headlights-report/>>, Access in: 03 nov. 2018.