BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

SYSTEM FOR RACE CAR PARAMETERIZATION

SYSTÉM PRO PARAMETRIZACI ZÁVODNÍHO AUTA

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **PATRIK TISZAI**
AUTOR PRÁCE

SUPERVISOR **Mgr. KAMIL MALINKA, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2021/2022

Bachelor's Thesis Specification

Student: **Tiszai Patrik**
Programme: Information Technology
Title: **System for Race Car Parameterization**
Category: Information Systems
Assignment:

1. Learn about the relevant technologies for CAN bus messaging.
2. Analyse the current CAN bus messaging solution of the TU Brno Racing team.
3. Based on user needs, propose a new solution for sending messages to devices connected to the CAN bus (pedal, VCU, etc.). The solution will support enhanced functionality, in particular: automated configuration retrieval (incl. parsing and data processing), unique message and device identification system, support for configurable message filters incl. analytical tools for optimizing filter usage, access control for multiple users, change control and history tracking.
4. Implement proposed solution and conduct testing with focus on performance parameters.
5. Design ways to integrate with an existing message tracking solution (PCAN-View).
6. Discuss possible future extensions.

Recommended literature:

- Do, Duc Huy. Automated Tool for CAN Bus Message Mapping. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.
- relevant documentation

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Malinka Kamil, Mgr., Ph.D.**
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 11, 2022
Approval date: November 3, 2021

Abstract

The work aims to design and implement an application for creating the configuration of messages and device filters in the CAN-bus for the school development team TU Brno Racing. Above all, the application should streamline and reduce the error rate when creating new configurations. For successful development, a requirements analysis was first performed in the team and subsequent design and implementation. The result is a functional web application that is tested for the usability of the configuration and user interface.

Abstrakt

Cieľom práce je navrhnúť a implementovať aplikáciu na vytváranie konfigurácie správ a filtrov zariadení v CAN-bus zbernici pre školský vývjový tím TU Brno Racing. Aplikácia by mala predovšetkým zefektíniť a znížit chybovosť pri vytváraní nových konfigurácií. Pre úspešný vývoj, bola najskôr vykonaná analýza požiadaviek v spomínanom tíme a následný návrh a implementácia. Výsledkom je funkčná webová aplikácia, ktorá je otestovaná s ohľadom na použiteľnosť vytváranej konfigurácie a užívateľského rozhrania.

Keywords

CAN-bus, STM, message filtering, identifier assignment, user interface, Flask, library creation

Kľúčové slová

CAN-bus, STM, filtrovanie správ, pridelovanie identifikátorov, užívateľské rozhranie, Flask, vytváranie knižnice

Reference

TISZAI, Patrik. *System for Race Car Parameterization.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Kamil Malinka, Ph.D.

System for Race Car Parameterization

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Kamlila Malinku, Ph.D..Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Patrik Tiszai
May 11, 2022

Acknowledgements

First and foremost, I would like to thank my supervisor Mgr. Kamil Malinka, Ph.D. from the Department of Intelligent Systems, for his patient guidance and lots of insightful and sharp comments. I am also grateful to all team members from TU Brno Racing, who helped me find helpful research material and other relevant resources. Finally, a huge thank you goes to my family and friends for their support and encouragement.

Contents

Chapter 1

Introduction

I am a member of our school research team situated at the Faculty of mechanical engineering, where we are designing a formula student race car. These race cars have to be designed and constructed wisely because before entering the race track, every car must undergo a scrutineering to justify its design and manufacturing.

To provide flawless communication between many devices in the car, a CAN bus network protocol is being used for many beneficial reasons. The apparent reason was the worldwide usage of the Can-bus protocol in the automotive industry. This solution of communication is considered a straightforward and low cost.

The connectivity is provided only via a single wire instead of direct complex analog signal lines. This reduces errors, weight, and cost, which are fundamental aspects of building a competitive racing car.

The thesis is divided into seven chapters. The chapter 2 deals with the laying of the theoretical basis for the correct design. The reader can also learn basic information about the communication protocol used in the race car and its various types. The end of the chapter deals with comparisons with other communication protocols. In the next chapter, I am focusing on describing the usage of this protocol in the team. So this chapter is about calculating all the necessary parameters of the protocol to provide respectful functionality. The chapter 4 describes the analysis of the solution, including the definition of the problem with the output of the definition of user needs. Next, a mapping algorithm alongside filter assignment is described here. In the end, the expected data structure for the application is described. The next chapter is dedicated to designing the graphical user interface and all the necessary parts for developing a web application. The last chapter 6 is devoted to the implementation of key elements of the application. The end of this chapter consists of various user tests and their evaluation.

Chapter 2

Theoretical introduction

2.1 Formula Student

Formula Student is the European version of the original US Formula SAE (Society of Automotive Engineers) competition. It is a prestigious competition of university teams composed of bachelor's and master's studies students. The beginnings of the competition date back to 1981, when the idea of university races was born in the USA. It came to Europe 17 years later. Moreover, races are not taking place only on these continents. In addition to America and Europe, the competition is also held in Brazil, Japan, India, and Australia. Overall, over 800 teams from all over the world take part in these races. There is, therefore, no need to emphasize the importance of participation as it plays an important role in the rating and prestige of the whole university.

The aim of the competition remains unchanged every year. In addition to gaining valuable experience for participants and making friends with colleagues from abroad, the main goal is to show the world the ability to build a single-seat racing car, which must be maneuverable, powerful, reliable, and safe at the same time. The task of each team is to find all the resources needed for such a challenging task as the construction of a racing car undoubtedly is.

The winning team of each race is determined based on scoring from static and dynamic disciplines. Without a doubt, not only the quality of the car but also the quality and skill of the driver himself plays a big role in winning races.[4]

TU Brno Racing

Our team, TU Brno Racing, consists of students of faculties of the Brno University of Technology. The core of the team is composed of students of mechanical engineering. However, in recent years the formula student is becoming more popular among students from the faculty of electrical engineering and the faculty of information technologies. The main reason is the transition from an internal combustion engine powertrain to an utterly electric powertrain with the integration of driverless elements. Throughout the development, we receive professional support and consultation from both Ph.D. students, senior management, and, what is more, from external experts, not only from the automotive industry.

Our main goal is to design, construct and test a one-seat racing vehicle suitable for participation in international competitions held in different parts of the world. Although it is an important motivation, there is another one that drives our racing team - the desire for knowledge, experience in the field of the automotive industry, and the ability to work

in a good team, which is our driving force to achieve the goal of being the best team in the world. Nevertheless, all this hard work would be worthless without our sponsors, who provide us with financial, material, and consulting support.

Figure 2.1: Team TU Brno Racing

Team achievements

In 2021, our first electric formula racing car, ED1, was introduced, and even in the first race that we attended in Most, Czech Republic, we could achieve a promising result. Overall, our race car has ended in solid second place, winning in business plan presentation and efficiency. The same year, our team also attended races with the other car, DX, which was the team's last internal combustion engine formula. With our ED1 race car, we are 36th in the world ranking, and the DX formula ended 24th in the world ranking.

Formula ED1 concept

Every formula student race car must follow some rules, which are updated every year and released by Formula student Germany. These rules are essential for teams, and ignoring them results in disqualification. Our formula is made of a combination of carbon fiber and steel tubes used at the rear of the car. It is powered by two electric motors driving the rear wheels. A fast and reliable race car also requires stable communication between devices; for this purpose, a CAN bus network is used, and this is also the main topic of this bachelor thesis.

2.2 Can Bus introduction

Control Area Network (CAN) is a network communication protocol used for communication between microcontrollers and other devices. CAN was introduced in February 1986 at the Society of Automotive Engineers congress by Bosch GmbH. In CAN, a broadcast type of communication is being used, so the message is transmitted to all devices connected to the CAN bus, and the devices using filters determine which message is relevant reading for them.

Compared to other solutions, where data is sent directly to every endpoint, CAN provides a simple, cheap, efficient, and flexible implementation of communication between devices. CAN bus is mainly used in the automotive industry. [10]

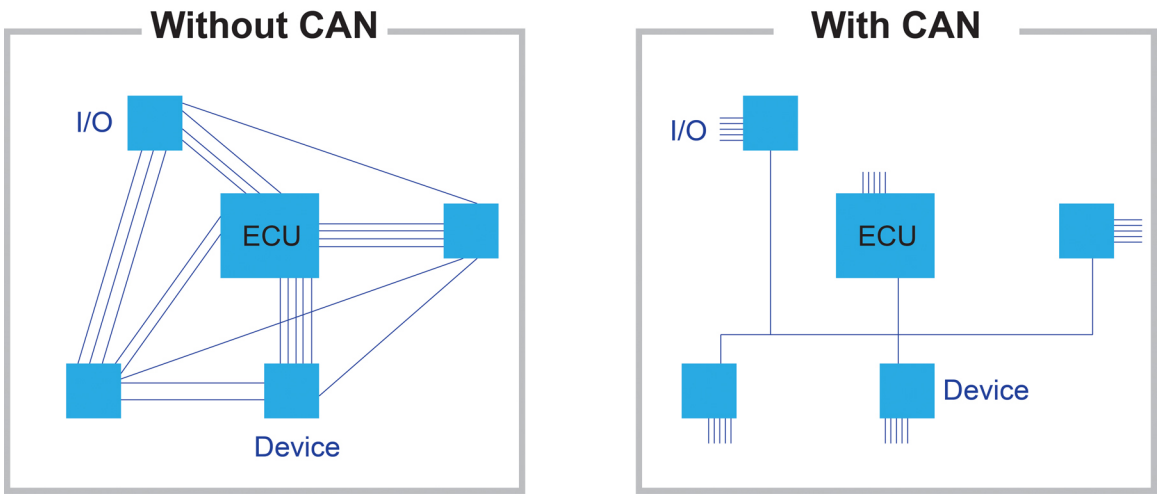


Figure 2.2: CAN bus compared to different network solution [12]

2.3 History

The roots of the CAN bus reach out to the early 1980s when an evaluation of serial bus systems was ongoing by engineers at Bosch, whose aim was to adopt some of these systems into the passenger car industry. But none of the available network protocols satisfied their requirements, so in 1983 the development of a new serial bus system was started by Uwe Kienche. The main purpose of this new protocol was to add new functionality to existing solutions, but as a by-product, the reduction of wiring harness was also achieved.

During development also, some big companies were involved in this process. In the very first stage of specifying the requirements of the new protocol, Mercedes-Benz got involved, and so did Intel, mainly because of potentially becoming the main semiconductor vendor. Professor Dr. Wolfhard Lawrenz from the University of Applied Science Braunschweig-Wolfenbutte, Germany, played a big role in this process as a consultant who gave the name "Controller Area Network" to the newly created network protocol. Academic assistance was provided by Professor Dr. Horst Wettstein from the University of Karlsruhe.

The introduction of a new network serial bus protocol was announced in February of 1986 at the Society of Automotive Engineers Congress in Detroit. The first controller chip, the 82526 was introduced by Intel after several months of presentation and publica-

tion describing the new serial bus protocol. The 82526 was the first chip containing the implementation of CAN.

Although CAN protocol was mostly meant to be used in passenger vehicles, the first implementation found its place in a completely different market compartment, especially in northern Europe. The first use of CAN in passenger cars was in 1991 when Mercedes-Benz started using it in their upper-class car line-up. In 1995 CAN was also adopted by BMW in its series 7 cars. This invention is considered one of the most successful network protocols ever introduced thus of its robustness.[2]

2.4 CAN protocol standards

International Standardization Organization (ISO) defined CAN as a serial bus originally developed for the automotive industry to replace the complex wiring system with a two-wire bus. CAN protocol offers two versions. Standard CAN provides an 11-bit identifier, that is enough for creating 2^{11}, or 2048 unique message identifiers. For addressing more messages, extended CAN was introduced with a bit field space increased up to 29-bit, thus 2^{29}, or 537 million unique identifiers can be created in case the standard 11-bit CAN cannot provide enough identifiers. [6]

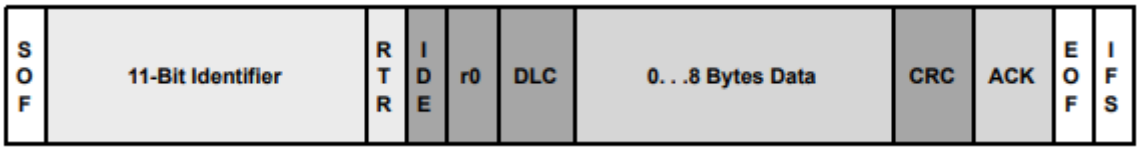


Figure 2.3: Standard CAN

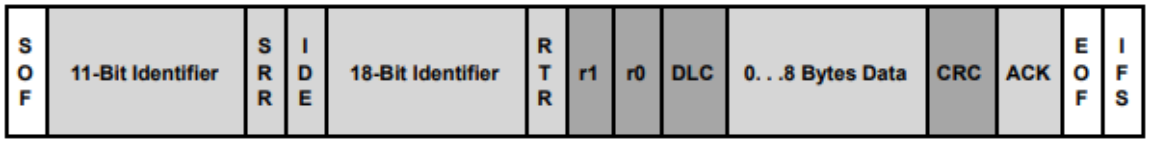


Figure 2.4: Extended CAN

- SOF — The Start Of Frame (SOF) bit marks the start of a message, and is used to synchronize the nodes on a bus.
- Identifier — Determines the priority of the message. Lower value means higher priority.
- RTR, SRE — The single remote transmission request (RTR) bit is set for remote frames
- IDE — The single identifier extension (IDE) bit shows whether the standard or extended CAN message identification has been used.
- r0 – r1 — Reserved bit.
- DLC — The 4-bit data length code (DLC) contains the number of bytes of data being transmitted.

- Data — Application data up to 64 bits.
- CRC — The 16-bit cyclic redundancy check (number of bits transmitted).
- ACK — When a valid message is accepted by receiver, then an acknowledgment bit is sent.
- EOF — 7-bit field marks the end of a CAN frame.
- IFS — The amount of time required by the controller to move a correctly received frame to its proper position in a message buffer area.

2.5 Bus access and arbitration

Bus access is event-driven and occurs randomly. Simultaneous occupation of the bus is implemented with a bit-wise arbitration. The identifier in the message frame determines the priority. The lower the binary identifier number, the higher priority is represented. Therefore the node with higher priority gets access to the bus. Signals consisting only of zeroes hold the bus dominant and can occupy the bus for the longest time. A recessive bit (1) is always overwritten by a dominant bit (0) on the CAN bus. Therefore, CAN finds its usage in real-time systems because this approach with a combination of an error handling mechanism provides very good data consistency.[8]

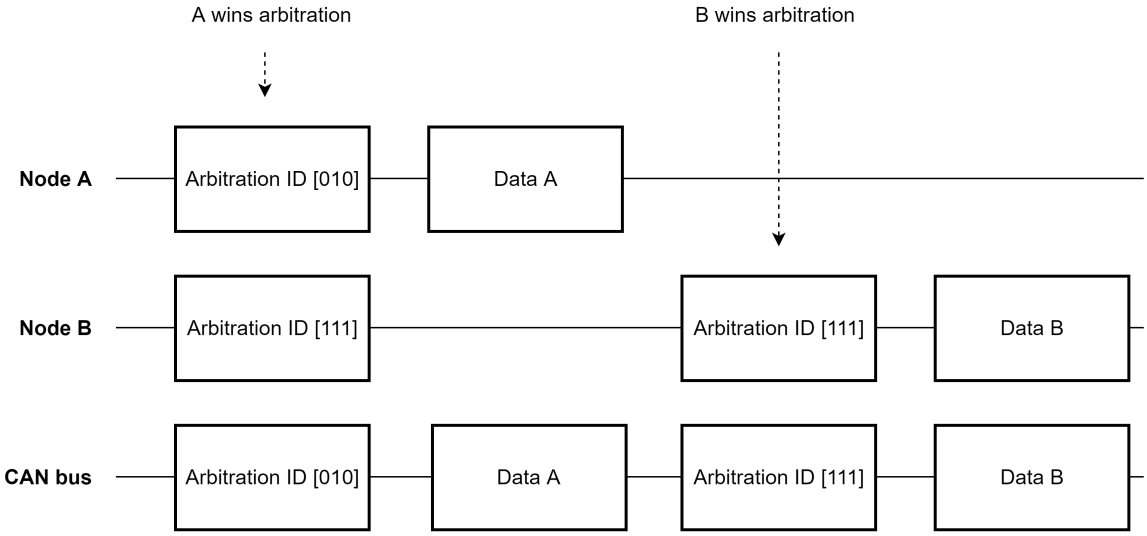


Figure 2.5: Example of bus arbitration

2.6 CAN bus applications

Can bus network was mainly created to service in the automotive industry. Its initial focus remained untouched, this is the most common network used in vehicles nowadays. Throughout the years, the CAN bus was also adopted by other industries, mainly because of its complexity and easy-to-use mechanism. Such as trams, undergrounds, light railways, and long-distance trains use CAN on different levels of the network throughout these vehicles. CAN is also present in aerospace applications, such as flight-state sensors, navigation systems, and many others. The field of medical assessment is also incorporated with CAN as an embedded network inside medical devices.[1]

2.7 CAN bus benefits over other solutions

Can bus network offers a very complex networking solution for communication. It provides a low-cost solution for establishing communication between separate devices by a significant reduction in wiring. Also, a big advantage is the electronic control unit does not necessarily need to have an analog to the digital input to every device, but all the devices communicate on a two wire CAN interface. Thus, the cost of the whole system is decreased as well, compared to other systems. For reliable communications an error detection is also built into the protocol [11]

The communication data is broadcasted over all devices inside the CAN interface, so every single device can choose whether to accept the message or not. It's done by using filters on each device to accept the relevant messages. Every message has its distinguished unique identification (ID). This ID defines the message priority when sending data over CAN. The priority has a significant impact when more nodes want to send more messages simultaneously, so the node with higher priority gets transmitted over the lower priority.

Chapter 3

CAN-bus usage at TU Brno Racing

3.1 Communication structure in the vehicle

All devices required for running the car are connected to the CAN-bus network, which is separated into three buses, CAN-generic, CAN-critical, and CAN-ams. Can-critical consists of essential messages required for running the car. CAN-generic is mainly used as a reserved bus in case there is a problem on CAN-critical. The CAN-ams is used for the messages monitoring the battery pack. This communication network protocol was chosen because of its flexibility when adding new devices to an existing network, the implementation of this solution was also considered very easy compared to other protocols, but the main reason is significant wire reduction; therefore, the overall weight of the vehicle is kept minimal.

Every single CAN bus network consists of the vehicle control unit (VCU), where all the logic behind controlling all the systems in the vehicle is stored. Other devices connected are the Steering unit, Dataloger unit, High-voltage box unit, Pump unit, and Telemetry unit.

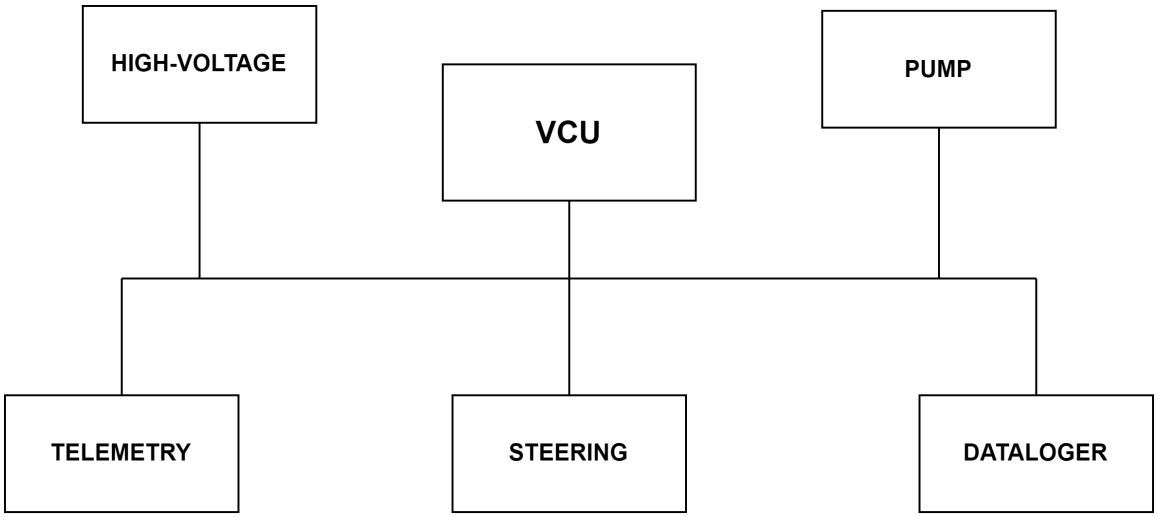


Figure 3.1: Communication structure inside the vehicle

3.2 Message identification and filters

Currently, team TU Brno Racing is using a custom-developed library for securing proper communication via CAN bus. The source code of this library is completely written in C code. In this library all the required identifications and filters are defined to enable flawless functioning of CAN bus network. The Can bus library is a separate module that is included in each device's program, so all the devices can see the configuration between each other if necessary.

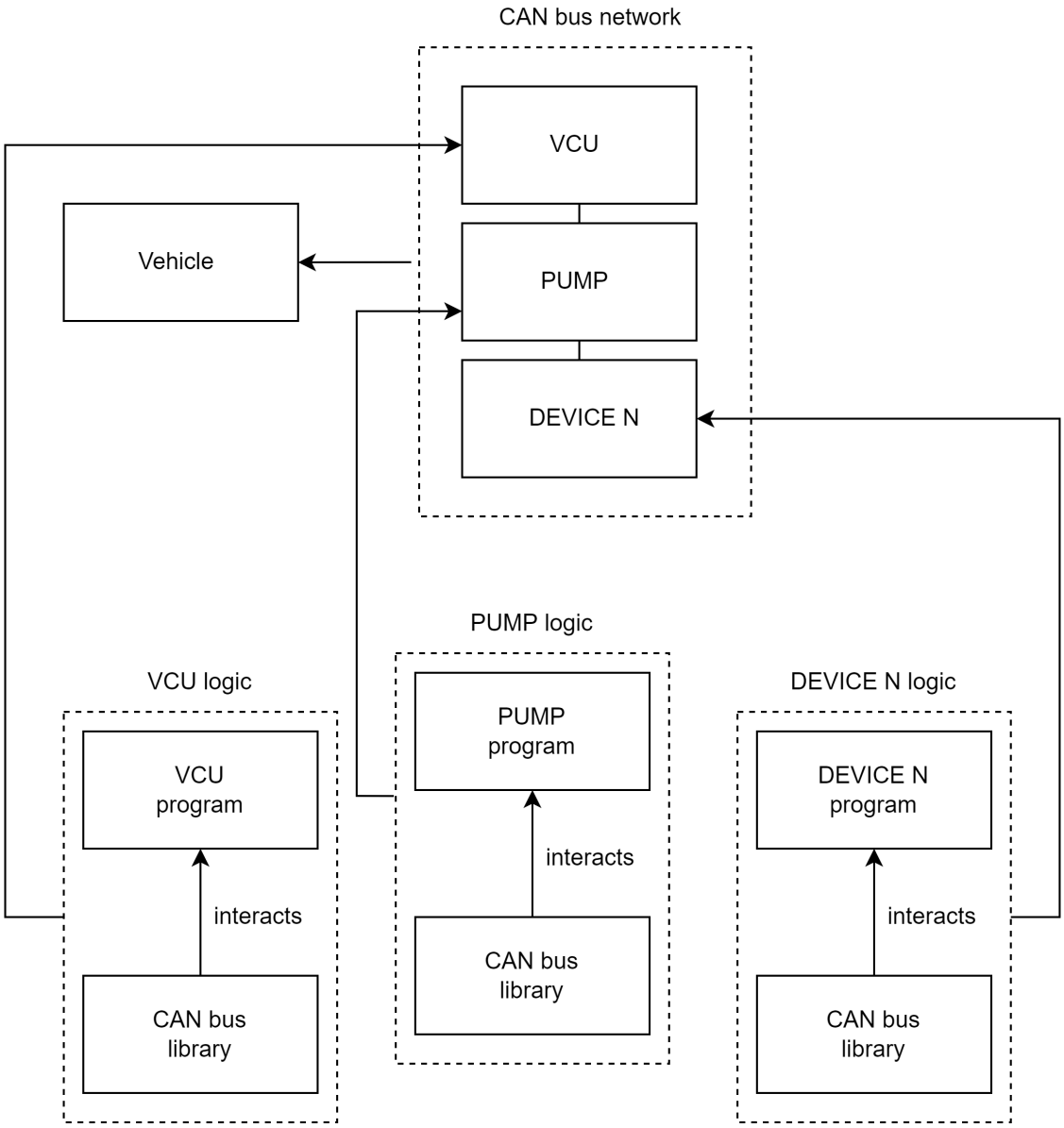


Figure 3.2: Structural demonstration of vehicle management

3.3 Block diagram of CAN on used microcontrollers

The team uses microcontrollers from the company STMicroelectronics (STM), so the CAN bus will be described on these. These microcontrollers have been chosen for their ease of programming and various materials available on the internet regarding some programming issues.

For setting up the messages, three transmit mailboxes are provided to the software. For deciding the message priority, a Transmission scheduler is used. The Can implementation on STM microcontrollers provides 14 configurable identifier filters for selecting only relevant messages for the current device; also two receive FIFO (first in first out) fronts are implemented on the microcontroller. These buffers are entirely managed by hardware, and each FIFO can store 3 complete messages.[3]

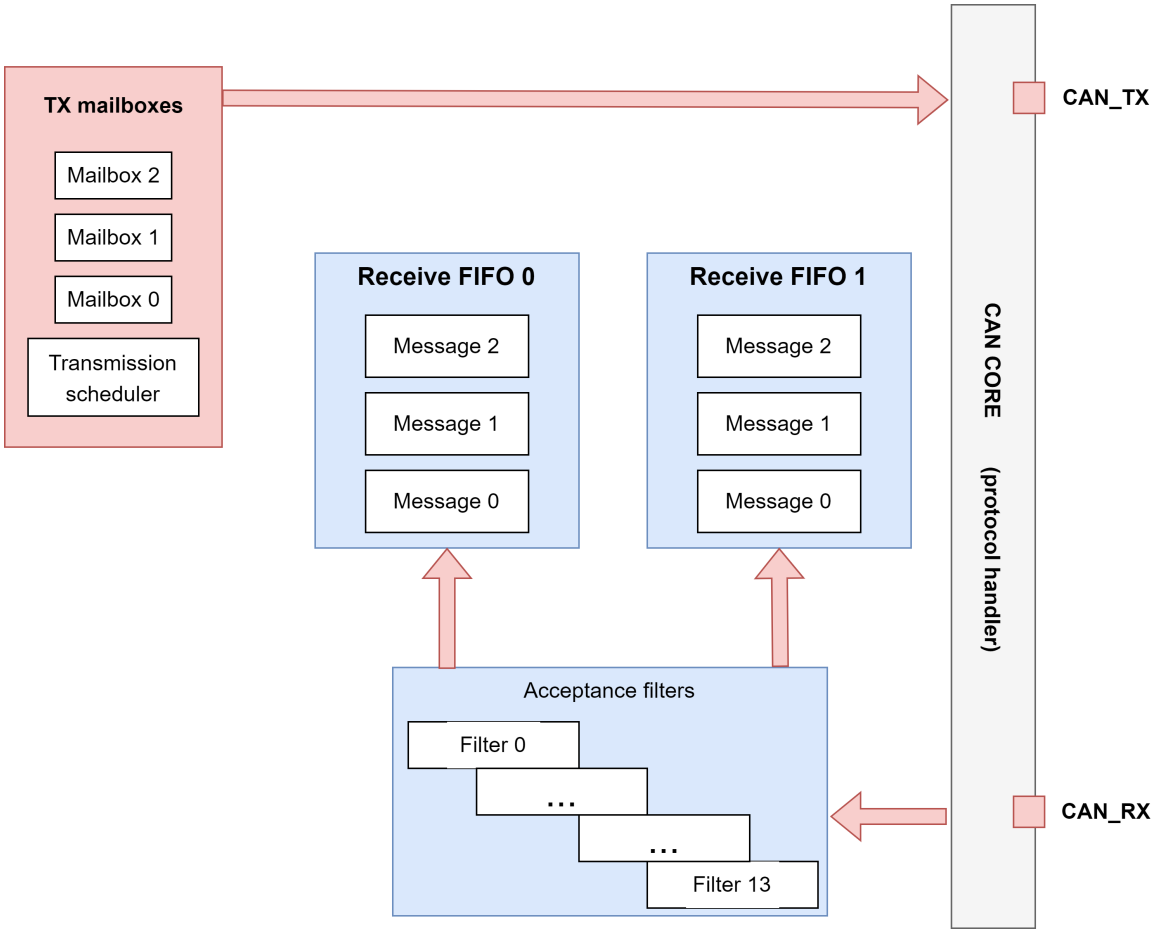


Figure 3.3: Block diagram of CAN implemented on STM32 microcontrollers

3.4 Library definition

The current solution for creating the library is managed without using any automated software. To satisfy all needs of other programs using this library, a certain structure must be followed.

Structure of the library:

- Dragon_CANbus.c
- Dragon_CANbus.h
- Dragon_CANbus-deviceXXX.h
- Dragon_CANbus-filter.h
- Dragon_CANbus-filter.c

Files **Dragon_CANbus.c** and **Dragon_CANbus.h** provide the overall configuration of CAN bus, **Dragon-CANbus_filter.h**, and **Dragon-CANbus_filter.c** consists of all call-back functions to every message, all unique identifications for every message, and all the filter banks for different devices are defined here. **Dragon-CANbus_deviceXXX.h** files are used for activating call-back functions (different for each device).

3.5 Current solution for creating identifiers and filters

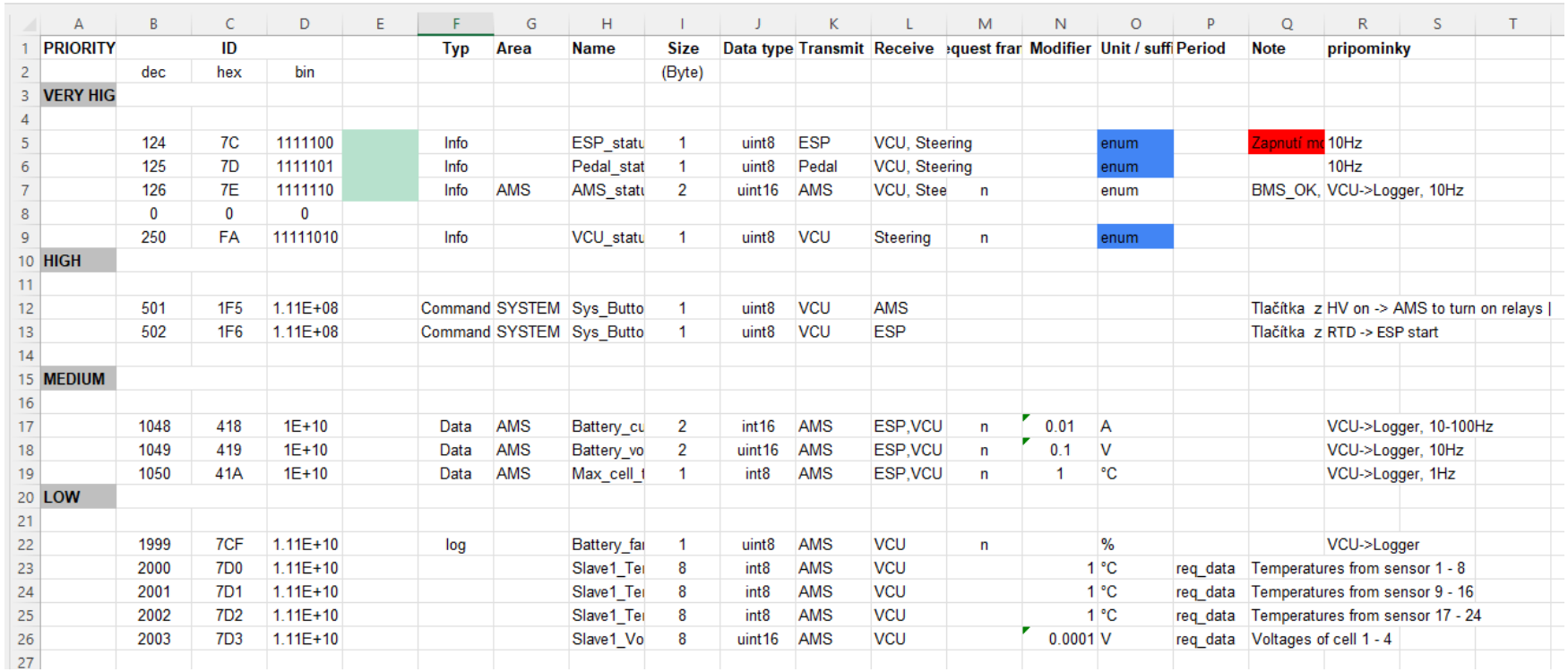
The first step for creating a functioning library that is used by all the devices connected to the CAN-bus network starts with filing in an excel document shown in picture 3.4. In this file, all the necessary information could be found, such as the name of the message, the priority of the message, and more. The priority plays a significant role in the configuration. It has four stages very high, high, medium, and low, the team defined these stages, and each section is relative to a specific message ID range. These ranges come out from analyzing the number of messages within a single priority section. Currently, these ranges are used:

- **Very high**: 1 – 500
- **High**: 501 - 1000
- **Low**: 1501 – 2048

After getting familiar with the parameters of every priority section, the messages can be added to the list, and the user must choose the correct section where to input the message regarding its importance.

The term “input the message” means that the user fills in the name, data type of the message, the area, specific for the message, size, transmitter, receiver, modifier, periodicity, and notes. The only required information for further processing is the name of the message and the receiving device name; other information is only for deeper specification information of the message. It is beneficial for developers when working with the messages.

After having all the messages entered in the next step, it is necessary to manually calculate an identification number for every message corresponding to the priority section. Based on the receiver device of the message, a filter is made. This filter is also manually calculated; thus, there is a significant chance of making an error. The calculation will be described in the section 3.6.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	**PRIORITY**		**ID**			**Typ**	**Area**	**Name**	**Size**	**Data type**	**Transmit**	**Receive**	**equest frar**	**Modifier**	**Unit / suffi**	**Period**	**Note**	**pripominky**		
2		dec	hex	bin					(Byte)											
3	**VERY HIG**																			
4																				
5		124	7C	1111100		Info		ESP_statu	1	uint8	ESP	VCU, Steering			enum		Zapnutí m	10Hz		
6		125	7D	1111101		Info		Pedal_stat	1	uint8	Pedal	VCU, Steering			enum			10Hz		
7		126	7E	1111110		Info	AMS	AMS_statu	2	uint16	AMS	VCU, Stee	n		enum		BMS_OK,	VCU->Logger, 10Hz		
8		0	0	0																
9		250	FA	11111010		Info		VCU_statu	1	uint8	VCU	Steering	n		enum					
10	**HIGH**																			
11																				
12		501	1F5	1.11E+08		Command	SYSTEM	Sys_Butto	1	uint8	VCU	AMS					Tlačítka z	HV on -> AMS to turn on relays |		
13		502	1F6	1.11E+08		Command	SYSTEM	Sys_Butto	1	uint8	VCU	ESP					Tlačítka z	RTD -> ESP start		
14																				
15	**MEDIUM**																			
16																				
17		1048	418	1E+10		Data	AMS	Battery_cu	2	int16	AMS	ESP,VCU	n	0.01	A			VCU->Logger, 10-100Hz		
18		1049	419	1E+10		Data	AMS	Battery_vo	2	uint16	AMS	ESP,VCU	n	0.1	V			VCU->Logger, 10Hz		
19		1050	41A	1E+10		Data	AMS	Max_cell_t	1	int8	AMS	ESP,VCU	n	1	°C			VCU->Logger, 1Hz		
20	**LOW**																			
21																				
22		1999	7CF	1.11E+10		log		Battery_fa	1	uint8	AMS	VCU	n		%			VCU->Logger		
23		2000	7D0	1.11E+10				Slave1_Te	8	int8	AMS	VCU		1	°C	req_data	Temperatures from sensor 1 - 8			
24		2001	7D1	1.11E+10				Slave1_Te	8	int8	AMS	VCU		1	°C	req_data	Temperatures from sensor 9 - 16			
25		2002	7D2	1.11E+10				Slave1_Te	8	int8	AMS	VCU		1	°C	req_data	Temperatures from sensor 17 - 24			
26		2003	7D3	1.11E+10				Slave1_Vo	8	uint16	AMS	VCU		0.0001	V	req_data	Voltages of cell 1 - 4			
27																				

Figure 3.4: Table for inputting CAN message data

3.6 Calculation of filters

As described in the section 3.3, the microcontroller offers two hardware FIFO buffers accompanied by 14 configurable filters, so in total, there are 28 available filters to configure. To avoid missing reading any received messages, the team decided to set each filter to admit only 3 messages, but this is only possible with the two most prioritized message categories because other categories consist of much more messages than the filters can take, so this rule is not applied there. Therefore, filters for medium and low priority messages take more; this number differentiates on total message quantity.

The filter is defined with two parameters:

1. CAN ID allowed to be received
2. Mask register for masking the CAN ID

Example of filter calculation accepting only one message with ID = `0x245`.

```
Acceptance Filter =0x245
  0 1 0 0 1 0 0 0 1 0 1  (RecieveID)
  &
  1 1 1 1 1 1 1 1 1 1 1   (Mask Register)
-----------------------------------------------------
  0 1 0 0 1 0 0 0 1 0 1  -->Accepted ID
```

Figure 3.5: Filter calculation example [14]

If more CAN ID numbers needs to be accepted with one filter, then the mask register is changed, e. g. if the mask register is changed to `0xFE`, messaged with IDs `x245` and `0x244` are accepted.

Chapter 4

Mapping analysis and design

This chapter describes the whole analysis, which started with the definition of the problem, the survey target group, and then the definition of user needs. Next, a mapping system based on this analysis is described.

4.1 Problem definition

The main reason is the growing amount of messages and complicated recalculation. When a single message with a predefined ID needs to be added to the list, it is sometimes easier to recalculate the whole project to avoid any inconsistency.

Another factor is variability between team members, not everyone is fully dedicated to this project and leaves after only a year, and it is not beneficial for them to learn the complicated system just for one year, so we sometimes have to bother the older team members who developed this solution to help the team.

The last big factor is the "human factor" all humans make mistakes at some point, and this is not accepted in racing because it can cause a more serious outcome than just missing one reading from a sensor that someone has not configured correctly.

4.2 System requirements from TU Brno Racing

The new system for creating unique CAN bus messages and the following assignment of them to device filters for team TU Brno Racing has some requirements from the users.

Specified requirements for the application:

- It should be encapsulated into a web application to provide easy access for team members and not cause incompatibilities between versions of the operating system.
- The system must be completely automatic without any further classification needed, i.e., the user inputs only necessary parameters of a CAN message, like name, priority section, and the receiver name, optionally some notes, the system will automatically calculate everything required for creating the library.
- The final product of the system will consist of a few essential documents required to have a beneficial impact on our team members.

- One output file will provide overall information about the configuration, meaning there will be a complete list of all CAN messages sent over the network with some additional information for a better understanding of the flow of the messages, and the other documents will create the library used in other projects across team members.
- Possibility to show message properties.
- Ability to download the crated library.
- The application should provide history tracking, meaning that it would be possible to select an older configuration and download it.
- Simple statistics tools, regarding filter usage for each device.

Mapping process analysis

Currently, there isn't any capable system to satisfy team requirements because the microcontrollers that we use are mostly popular among individuals or small companies like our team; therefore, it is not beneficial for software companies to develop a variable mapping system for these microcontrollers. Another big factor is that the microcontrollers we are using don't explicitly define the exact way how to use the CAN bus implemented, so a few years prior, our team has decided to create self-made modules to interact with CAN implementation of the microcontroller supplier. Therefore the whole CAN message and filter library was created to fill our needs, but with a big weakness, that it would be almost impossible to find any automated program for configuring the CAN.

Considering all these factors, not being able to find any program reaching the requirements for creating the filters, a custom-developed system has to be developed for this purpose. The lack of possible algorithms suitable for this system is mainly caused by very specific requirements to get the most out of the currently used microcontrollers and not necessarily change the whole architecture, which has been proved very reliable and effective.

4.3 Mapping algorithm

For the usage of the strength of CAN bus on our microcontrollers, we are using hardware filters provided by the chip. These filters work faster than using any kind of software filter based on if-else clauses. But there are some limitations. Two big limitations that need to be counted in this mapping algorithm are the maximum number of hardware filters for a single chip and also the maximal effective count of messages assigned to a single filter.

The number of filters for each device is 28, and 3 messages per filter; these add up to 84 possible messages to be configured for each device. In our system configuration, is this not always enough, so in some cases, we are forced to assign more messages to a single filter. In this case, the division into priority sections comes very handily because we can decide which messages are not that prioritized and essential to run the car properly, so we can afford not to get the message all the time because the messages are sent with some periodicity specific for each one of them and in the next iteration it is possible to get it right, but this is unacceptable behavior in important messages.

Mapping logic

The calculation of unique identification is based on creating identification clusters of two, three, four, and eight. Also, bigger clusters could be implemented, but for the purposes of this project, this is more than enough because in the worst-case scenario, when using clusters of size 8, the maximum amount of messages assigned to one device is increased from 84 to 224.

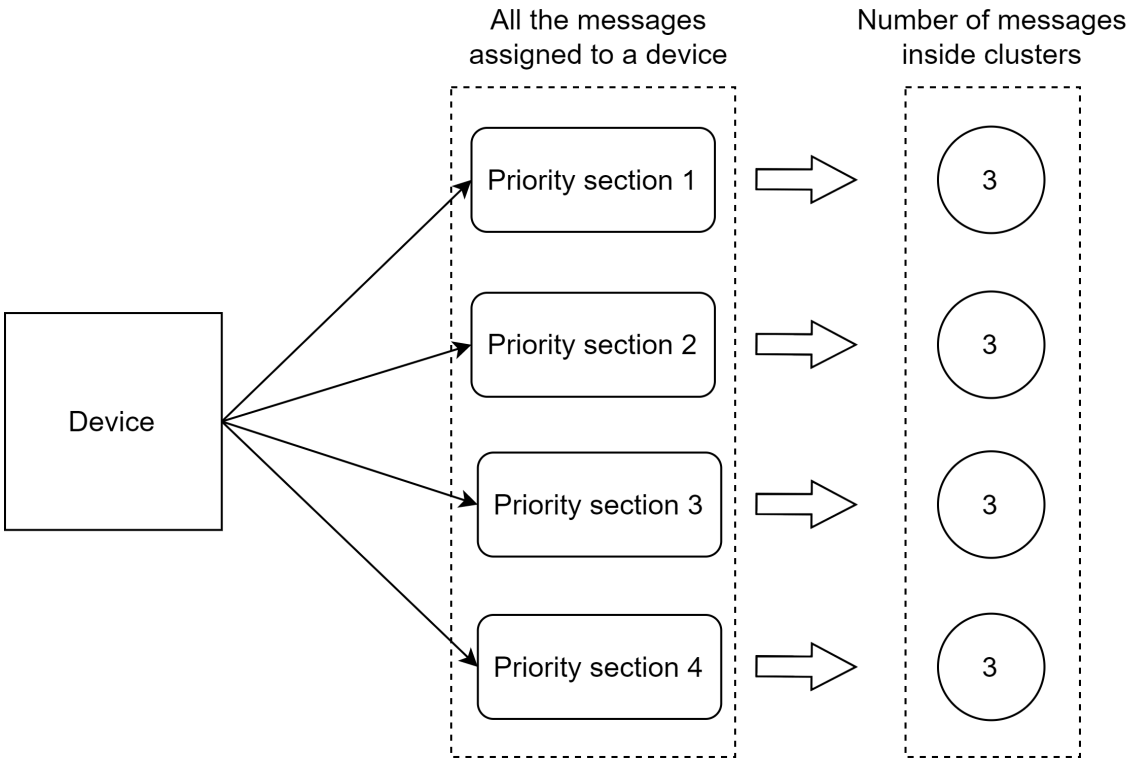


Figure 4.1: Ideal usage of clusters

Mapping restrictions

Taking into consideration the fact that the system works the best when only 3 messages are assigned to a single filter, a restriction has been created, so the cluster of 3 is primarily used for messages in the first two highest priority sections, and clusters of size 4 and 8 are only used in two lowest priority sections of messages, and still provide enough space. Respectively, when there are less than 84 messages assigned to one device, then only clusters of size 3 are applied to all priority sections. This decision was also reviewed and approved by the team.

Clusters of size 2 and a single message inside the cluster are only used in edge cases when it is impossible to fit them inside main groups of 3, 4, and 8, without utilizing the full free space of a cluster.

4.4 Cluster sizes explanation

The chosen sizes imply filter masks that also have to be used. These filter masks, in combination with a starting identification number, create one whole filter.

Cluster of size 2

For creating a group of 2 numbers a set of masks is used.

Values of the masks

1. `0x7FE` - (binary 011111111110)
2. `0x7FD` - (binary 011111111101)

The mask `0x7FE` is used when the starting IDs least significant bit is 0 and the mask `0x7FD` when the second least significant bit is 0.The image 4.2 shows the possible division.

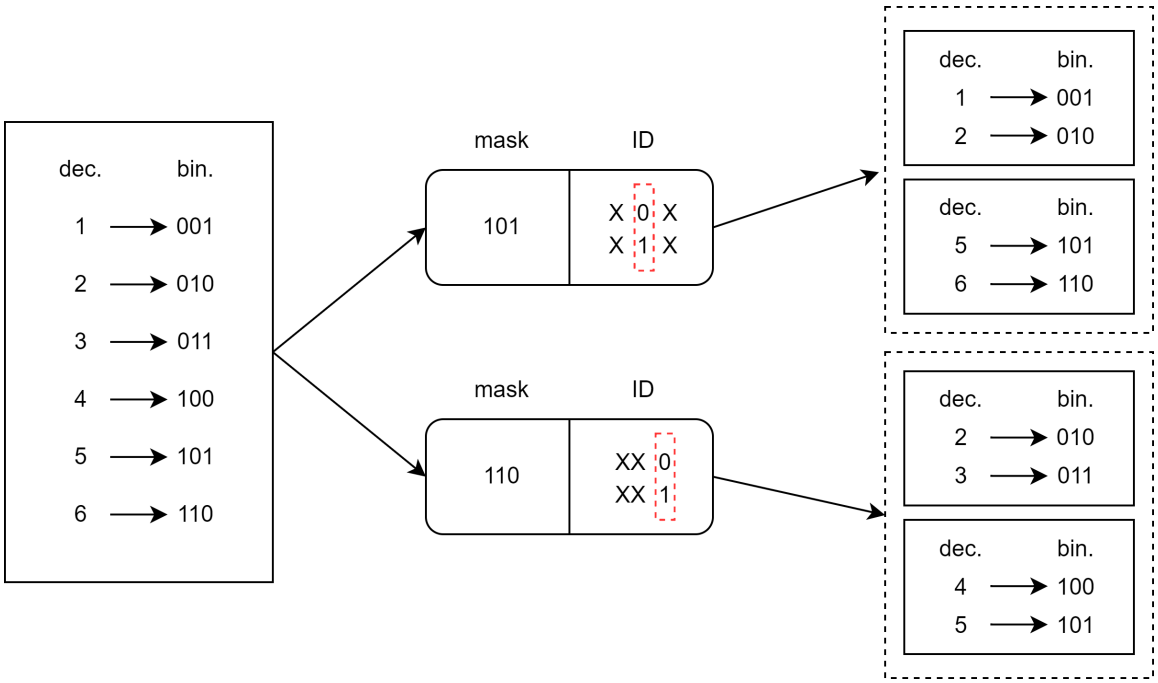


Figure 4.2: Cluster of size 2

Cluster of size 3 and 4

For this cluster, only a single mask is used with value `0x7FC`) - (binary 011111111100). So the two least significant bits determine the group, but for not causing any collisions all the IDs divisible by 4 are excluded because if this mask is used, then it also possible to create a group of four elements and not assign these values to the messages means that the filter in one hand accepts more than 3 messages, but regarding no real ID with the fourth number inside network is assigned the filter will accept only 3 messages in reality. For cluster size 4 the mask is the same, and the ID numbers divided by 4 are not excluded.

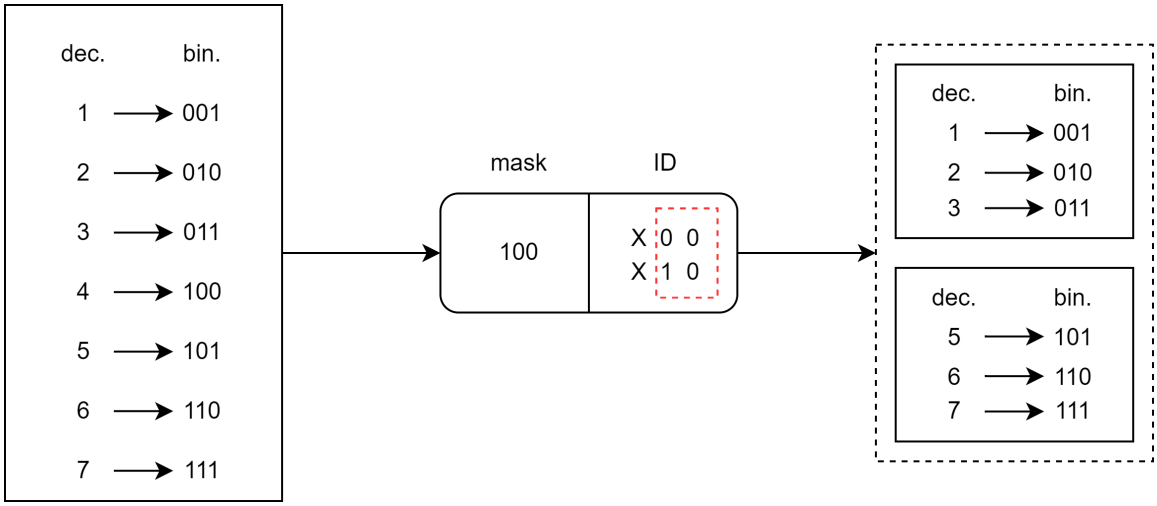


Figure 4.3: Cluster of size 3

Cluster of size 8

This cluster is only used in less prioritised messages with a single mask valued `0x7F8` - (binary 011111111000), meaning that the 3 least significant bits are defining the group of 8 elements.

Concerns about not enough clusters

After evaluation of all factors within the team influencing the number of messages and the distribution between priority sections, a final statement was set, showing that this proposed solution for creating only these 4 types of clusters would completely satisfy the needs and still have enough free filters for use in the future. The two least prioritized sections will also use the combination of smaller size clusters to exclude some unnecessary messages. Another factor in the evaluation was the current design of the communication inside the car, which is not going to be changed unless some new rules are released by Formula student Germany that our teams have to follow, but this is nothing that cannot be influenced by the team. If some rules are changed in the future, the design of the algorithm will be changed to be rules compliant.

4.5 Filter assignment

Each device connected to a CAN bus network has the capability of specifying 28 filter banks. These filters have an exact structure, being a 2D array of size [14][6].

mask0	id0	mask1	id1	fifo	en
{0x07FC,	0x007C,	0x07FE,	0x012C	0x0001,	0x0001},
{0x07FE,	0x0190,	0x07FF,	0x01F4,	0x0001,	0x0001},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FE,	0x041A,	0x07FC,	0x041C	0x0000,	0x0001},
{0x07FF,	0x07DA,	0x07FE,	0x07EE,	0x0000,	0x0001},
{0x07F8,	0x07F0,	0x07FF,	0x0000	0x0000,	0x0001},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000},
{0x07FF,	0x0000,	0x07FF,	0x0000	0x0000,	0x0000}

Figure 4.4: Demonstration of filter structure

For creating a working filter, both `mask0` and `id0`, respectively, `mask1` and `id1`, have to be defined in correspondence to the FIFO buffer (this program uses mainly the first buffer unless it is already fully occupied), and the last element `en` is set to 1, representing that the filter is configured correctly and could be used.

4.6 Input data for the configuration

The program takes only a `.csv` file with all the necessary information for creating the configuration. The file has to use semicolons as separators and has to have a header for defining column names.

Header keywords

1. **Type** - basic information about the message
2. **Area** - part of the network where the message is sent
3. **Name** - name of the message
4. **Size** - size of the carried data inside message
5. **Data type** - data type of carried data inside message
6. **Transmit** - name of the device that transmits the message

7. **Receive** - name of the receiver device, support multiple devices, separated by a comma
8. **Request frame** - additional information to data
9. **idNum** - specification of custom ID

All the columns, except Area, Name and Receive doesn't have to contain any data, because it is only used for information purpose for the programmer and not influencing the overall configuration.

```
Type;Area;Name;Size;Data type;Transmit;Receive;Request frame;idNum;
;;;;;;;;
FW;;FWU_Downstream;8;uint8;Datalog;VCU,HV_Box,Steering,Pump,ESP;;
FW;;FWU_Upstream;?;uint8;Datalog;VCU,HV_Box,Steering,Pump,ESP;;
Command;;Reset;1;uint8;Datalog;VCU,HV_Box,Steering,Pump,ESP;nw;
;;;;;;;;
Info;;VCU_status;1;uint8;VCU;Datalog;n;
Info;;ESP_status;1;uint8;VCU;Datalog;n;
Info;;Pedal_status;1;uint8;VCU;Datalog;n;;
Info;;AMS_status;1;uint8;VCU;Datalog;n;
Info;;Pump_status;1;uint8;PUMP;Datalog;;
;;;;;;;;
Error;AMS;NonCrit_error;2;uint16;VCU;Datalog;n;
Info;;TS_on;1;uint8;VCU;Datalog;y;
Command;;Discharge;1;bool;VCU;HV_Box,Datalog;n;
Data;;Brake_pres;2;uint8;VCU;Datalog;;
;;;;;;;;
Data;AMS;Battery_current;2;int16;VCU;Datalog;n;
Data;AMS;Battery_voltage;2;uint16;VCU;Datalog;n;
Data;AMS;Max_cell_temp;1;int8;VCU;Datalog;n;
Data;AMS;Avg_cell_temp;1;int8;VCU;Datalog;n;
Data;AMS;Min_cell_temp;1;int8;VCU;Datalog;n;
```

Figure 4.5: draft of `.csv` file

Chapter 5

GUI design

In addition to functionality, the application should also provide a sufficiently intuitive, simple, and usable interface. The result of the analysis of the main goals of the application and user needs is the following design of the user interface. Each designed screen and its elements are based on user requirements and solve some predefined issues.

5.1 Use case

In the picture 5.1, the use case diagram is present, designed in UML language. This diagram shows basic operations that the application should support on behalf of the user requirements. These operations should be projected into the functionality of each element in the proposed user interface.

The structure of the application is divided into four essential screens, where the user is capable of switching between them using a navigation bar placed at the top of the screen. This navigation bar is designed to be consistent throughout the whole application. There is also a logout button and the name of the user currently logged in on the right-hand side of the navigation bar, only when the user is logged in.

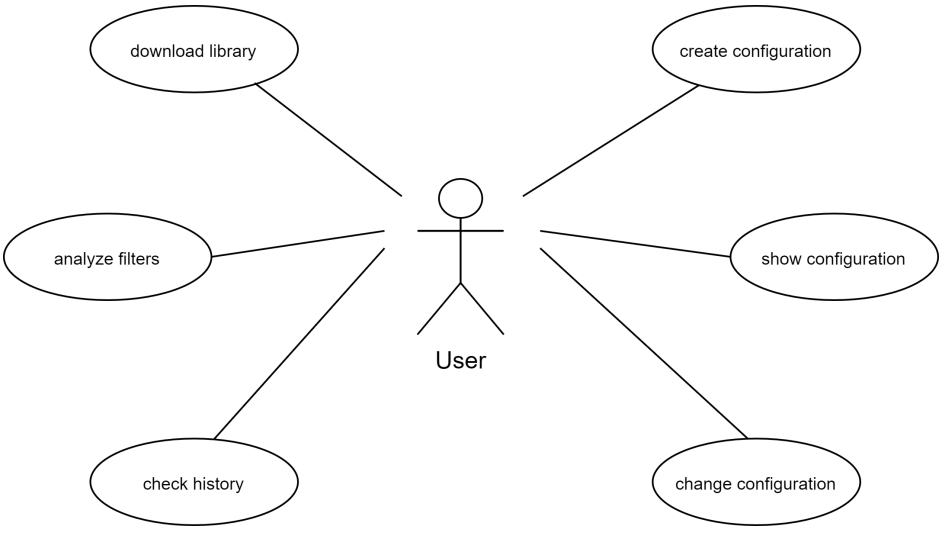


Figure 5.1: Use case diagram

The navigation bar consists of these screens:

- Create configuration
- Show configuration
- History
- Analyze

Login and register page

Before accessing all the functionality of the application, authentification would be required. For this purpose, two separate pages were designed; one is the register page, where the user inputs:

- User name
- Email
- Password

The second page is for already registered users, where it is necessary to log in to the application to gain full access to the functionality. The login page also would be the first screen after opening the application, so there will also be a button for registering a new user. The form for logging in on this page consists of only:

- User name
- Password

Create configuration

Following the user requirements, it was important to create an intuitive and easy-to-use screen for defining the configuration of whole CAN bus messages and filters. This page is also set as the homepage of the whole application after logging in, it is due to the frequent creation of configuration, and it would facilitate the work with the application. A simple form is present on this page, consisting of these parameters to be inputted:

- Name of the configuration
- Comment
- Input file (.csv only)

Configuration

This screen shows the current configuration of Can-bus messages. The messages will be displayed as a table where each row of the table shows a single message. The table should have a couple of columns for better orientation in the messages by the user. These columns would consist of:

- ID of the message

- Name of the message
- Name of the receiver device, support multiple devices, separated by a comma
- Size of the carried data inside message
- Data type of carried data inside message
- Name of the device that transmits the message
- Request frame - additional information to data

Above the list of messages is placed a download button for saving the CAN bus library created by the backend to the computer.

Show history

In the user specification was also mentioned that the application should provide a history of recent configurations. For this purpose, a separate page is created, where the user could look at the complete list of configurations with some additional information about the configuration done in the application. When clicking on one of these, the user would be redirected to the page `Configuration`, where the selected configuration will be displayed.

Analyze

The last page will be used to show analytics about the configuration. Each device in the configuration should have its own card on the page. In the header of the card, there is the name of the device, under the title, the total number of messages assigned to this specific device finds its place, and the last important parameter displayed in the card is the number of remaining free filter banks inside the device configuration.

5.2 Web framework

It is a software framework that was developed in order to simplify the web development process, so it allows web developers to write Web applications without knowing any low-level details about sockets, protocols, or process management. In general, frameworks provide support for interpreting requests, such as handling cookies and sessions. It is also helpful in producing responses and storing data persistently. Nowadays, numerous frameworks provide some sort of customization in their support for utilizing components and activities already mentioned above, so in the real world, it is possible to create a full-stack framework from scratch only using existing components.[7]

Flask

Flask is a lightweight WSGI web application framework written in python. It is free and open-source and lets developers develop web applications easily. Also, getting started with flask is very quick and easy, providing the ability to scale up to a complex application. Flask is based on the Werkzeg WSGI toolkit and the Jinja2 template engine, developed by Armin Ronacher, who led a team of international Python enthusiasts called Poocco. Flask was originally an April's fool joke that escalated into a serious useful application.[5]

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

if __name__ == '__main__':
    app.run()
```

Listing 1: Simple example in Flask

Bootstrap

Bootstrap is a free and open-source front-end development framework for the creation of websites and web apps. The Bootstrap framework is built on HTML, CSS, and JavaScript to facilitate the development of responsive, mobile-first sites and apps. Bootstrap was originally named Twitter Blueprint and developed by Mark Otto and Jacob Thornton at Twitter. The intention was to achieve consistency across different tools. Nowadays, it offers user interface components, layouts, and JavaScript tools along with the framework for implementation. This application would be accompanied by Bootstrap Version 5.[9]

5.3 Mongo DB

MongoDB is a document-oriented NoSQL database used for high-volume data storage. Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents. Documents consist of key-value pairs, which are the basic unit of data in MongoDB. Collections contain sets of documents and functions, which is the equivalent of relational database tables.[13]

Key features of MongoDB architecture

- Database - a container for collection, each database consists of files on file system, a MongoDB can accompany numerous databases
- Collection - it is similar to table in SQL database, collection can be created inside database and doesn't have to follow any structure
- Document - a record in a MongoDB collection, consists of field name and value
- Field - a name-value pair
- Id - required in every MongoDB document and represents a unique value, it is like primary key in SQL based database

Database structure

One of the main parts of developing an application is choosing the correct database type and the specific data model that will fulfill all the requirements from the user. For this

project, a MongoDB database is selected because the key feature of the application is to calculate a configuration that would be accessible by all the users no matter where they are. So the database should offer the possibility of storing documents, and MongoDB does this all. Also, relations aren't required. The picture 5.2 shows the database structure used with MongoDB.

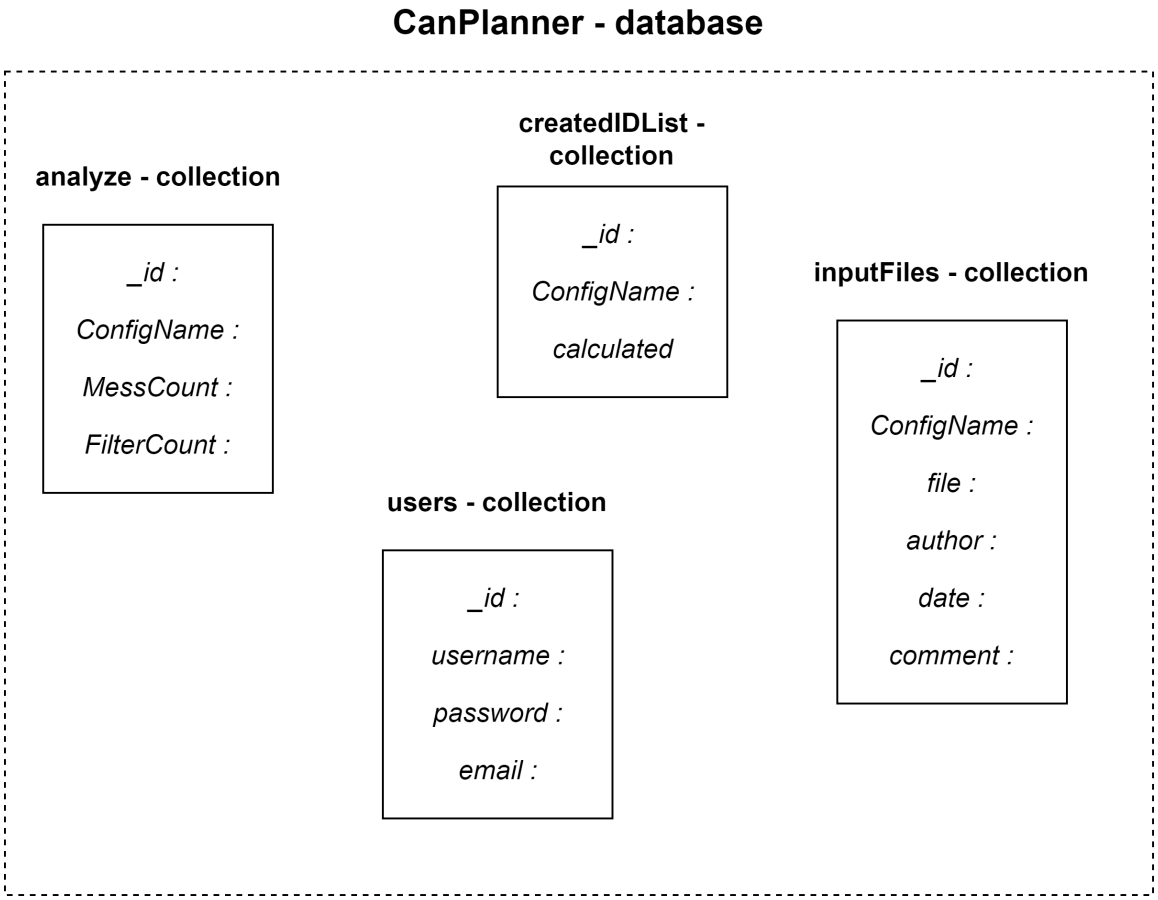


Figure 5.2: Database structure used in application

Chapter 6

Implementation and testing

For the implementation of the application, I have chosen a popular framework flask that is described in section 5.2, and Visual studio code was used as a development environment. In this section, I will be describing essential parts of the development of a mapping algorithm, including the implementation of the user interface. The last part of this section will deal with testing the application in team TU Brno Racing.

6.1 Mapping and filtering

This part of the application required much information to be stored throughout the whole process of calculation, so I decided to create two python classes, `class message()` and `class filter()`. The `class message()` is used for storing all kinds of information related to the CAN message, and the `class filter()` represents with its data a complete block of filters for one device.
The calculation of unique identification numbers and filters is implemented in these files:

- parse_input.py
- create_message.py
- create_filters.py

Parsing input data

The application is expecting a specific format for loading all the necessary information, as described in section 4.6. After parsing, the values from the document are stored inside the class message. This class only represents one CAN message, so these objects are assigned to a list(this list will be represented as a message list in upcoming parts of this document) that is then used as input for other functions.

Creating the message

Before calculating the identification, the message list is sorted by ascending message priority and also by receiver device names. This kind of sorting helps to reduce filter usage because messages that have to be received by the same device have identification next to each other.

Creating filters

An important part of the application is the creation of filters; these follow the proposal mentioned in section 4.5, with one change to the original idea. The last few messages that don't have the exact size of a cluster in priority sections 3 and 4 are not separated into smaller clusters but remain with the same mask. This decision was made after testing different configurations on a real CAN-bus network, and it has no noticeable effect on overall message acceptance. It also provides more free filters for further messages.

Library

The most important part of the application was reducing errors created by humans when not paying 100% attention, and the part when the library had to be assembled was the most critical and had caused many hours of debugging when a mistake was made. The library consists of two main files `Dragon_CANbus_filter.c` and `Dragon_CANbus_filter.c`, and files for each device `Dragon_CANbus_DEVICE.h`. Each of these files has its own header, telling some information to the user about the particular file because each of them has a different role in the library.

6.2 GUI implementation

The graphical user interface of this application is an important part for users to make their life easier when making the application more interactive. The most important parts of this GUI will be described in this section. The whole GUI appearance can be seen in appendices B.

Navigation

The most important part of a GUI is a navigation bar for orientation between pages. It is implemented as a part of the template for other pages in file `layouts.html`. The navigation bar used in the application is fully created using Bootstrap as all the other elements, such as forms, buttons, and tables.

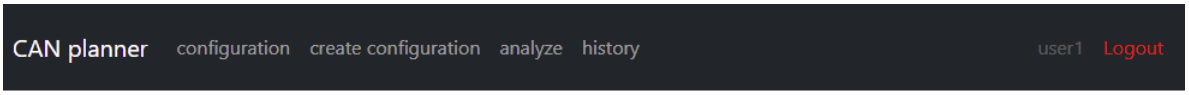


Figure 6.1: Navbar used in the application

Screens and elements

All the screens are located in folder `templates`. The screen is named relative to the content they are showing. Each of the files uses jinja2 as a template engine. A simple usage of jinja2 is displayed below.

```
      <tbody>
        {% for mess in messList %}
        <tr>
          {% for elem in mess %}
          <td>
            {{ elem }}
          </td>
          {% endfor %}
        </tr>
        {% endfor %}
      </tbody>
```

Listing 2: Usage of jinja2 from the application

Work with the database

Every user's input configuration file and calculated message IDs are stored inside the database. The usage is very simple because the chosen database completely fits all requirements of the application. The database is capable of storing different types of data without declaring it in advance, which is used for storing configuration files. For inserting and querying data inside the database, two methods are used:

```
configName = request.form['configName']
confTextArea = request.form['configTextArea']
file = request.files['file']

#inserting the file to the database
inputFiles.insert_one({'configName' : configName,'file':file.read(), 'comment' : confTextArea})
```

Listing 3: Inserting data retieved from form to the database

```
record = analyzeDB.find_one({'configName' : session['configName']})
messCount = json.loads(record['messCount'])
filterCount = json.loads(record['filterCount'])
```

Listing 4: Retrieving data from database

6.3 Testing

The testing of any sort of application is very crucial in development. The process of testing was ongoing throughout the development of the application from the start. The first thing that was observed was the faultiness in creating the library. In the early stages of the development, this part was the most important because not every approach invented was understood by other colleagues in the team, but after a few versions of the library, both ways have crossed, and the functioning library was born. This library created by this program was also part of the team TU Brno Racing formula ED1 on race in the Czech Republic in 2021, but at this stage, the application was working without the graphical user interface that was created later.

The GUI was then also tested by team members. Five team members were randomly selected from the electric powertrain section and were given several tasks to accomplish.
Tasks for testing:

- Create an account and login into the application
- Create a new configuration from the given file
- Download the created library
- Check how many filters are used in this configuration
- Create new configuration from other given file
- Download the library from the previous configuration

During these tasks, the user's work with the application was observed, focusing on the clarity of the graphical interface design. Task speeds were also observed, and whether the users got stuck on any of the tasks. At the end of the testing, users received a short questionnaire regarding the user interface.

Results of the testing

After completing the testing, it was determined which tasks were the biggest problem and whether the users were able to quickly orient themselves in the application. Regarding the proper communication with the team, while developing the graphical user interface, no testing attendant showed any sights of hesitation following the tasks.

Chapter 7

Conclusion

This work aimed to design and implement an application for creating the configuration of messages and device filters in the CAN-bus for school research team.

At the beginning, I studied the current configuration creation solution in the team. I then started looking for various commercial solutions for the problem. However, due to the very detailed specification of the system used, it was impossible to find a system that would meet user requirements. For successful implementation, I also had to choose several of the many technologies for web application development and study them.

The application's design is based on the definition of the problem that needs to be performed to define user needs. Based on this information, it was possible to design an algorithm to map and filter CAN-bus messages with a user interface that meets these needs.

The application was subsequently created using the Flask framework in the Python programming language and the Visual Studio Code development environment. I gained the basis of the correct design using various architectures that can be easily and clearly expanded from the study of technology in the design part of the thesis.

During the creation of this work, I encountered several problems, especially while obtaining requests from different users of the proposed application. The requirements were very fragmented and difficult to meet all. However, after a joint consultation, we found a consensus that the application could undergo the next phase, implementation. I did not encounter any unsolvable problem during the implementation, so the development could be shifted to the last stage. The last phase was testing, which worked very well due to well-defined user requirements.

From the point of view of the future of the application, it is possible to extend the creation of various independent configurations in the case of using several CAN-bus buses together with the addition of several analytical tools.

Another possible extension would be the integration of PCAN-View into the application. But at this time, it would be redundant because it has already been implemented as a part of another bachelor thesis by a team member.

The result of this bachelor thesis is a functional application that is tested and run on a real device.

Bibliography

[1] *Controller Area Network (CAN) Overview* [online]. [cit. 2021-01-15]. Available at: https://www.ni.com/cs-cz/innovations/white-papers/06/controller-area-network--can--overview.html.

[2] *History of CAN technology* [online]. [cit. 2022-01-15]. Available at: https://www.can-cia.org/can-knowledge/can/can-history/.

[3] *STM32F7 - BxCAN*. https://www.st.com/content/ccc/resource/training/technical/product_training/group0/49/99/d2/9b/67/7a/48/c0/STM32F7_Peripheral_bxCAN/files/STM32F7_Peripheral_bxCAN.pdf/jcr:content/translations/en.STM32F7_Peripheral_bxCAN.pdf.

[4] *What is the Formula Student Germany competition?* [online]. [cit. 2022-05-05]. Available at: https://www.formulastudent.de/about/concept/.

[5] Aggarwal, S. *Flask Framework Cookbook - Second Edition*. July 2019. ISBN 1789951291.

[6] Corrigan, S. *Introduction to the Controller Area Network (CAN)*. 2016.

[7] Curie, D., Jaison, J., Yadav, J. and Fiona, J. Analysis on Web Frameworks. *Journal of Physics: Conference Series*. november 2019, vol. 1362, p. 012114. DOI: 10.1088/1742-6596/1362/1/012114.

[8] Do, D. H. *Automated tool for CAN bus message mapping* [online]. Prague, 2019. Bachelor's thesis. Faculty of information technology CTU in Prague. Available at: https://dspace.cvut.cz/bitstream/handle/10467/88148/F8-BP-2020-Do-Duc%20Huy-thesis.pdf.

[9] Elrom, E. CSS, Bootstrap, Responsive Design. In:. December 2016, p. 131–164. DOI: 10.1007/978-1-4842-2044-3_6. ISBN 978-1-4842-2043-6.

[10] Gay, W. CAN Bus. In:. June 2018, p. 317–331. DOI: 10.1007/978-1-4842-3624-6_18. ISBN 978-1-4842-3623-9.

[11] Inc, R. E. A. *Using CAN Bus Serial Communications in Space Flight Applications*. white paper. 2018.

[12] Robinson, A. CAN Bus Cleared for Space Flight. *Design World*, 13. 2016 [cit. 2022-01-10]. Available at: https://www.designworldonline.com/can-bus-cleared-for-space-flight/.

[13] Taylor, D. What is MongoDB? Introduction, Architecture, Features Example. *Guru99*. [cit. 2022-01-15]. Available at: https://www.guru99.com/what-is-mongodb.html.

[14] Yadav, A. Bus Arbitration and Message Identification. *EmbedClogic*. [cit. 2022-01-15]. Available at: https://embedclogic.com/can-protocol-protocol-to-broadcast-message-on-a-network/can-bus-arbitration-and-message-identification/.

Appendix A

Contents of the flash drive

The flash drive contents:

- source codes of the application with documentation
- text of the bachelor thesis in `PDF` format
- source codes of bachelor thesis texts in LaTeX format
- `README.txt` file, with manual to the application

Appendix B

GUI screens

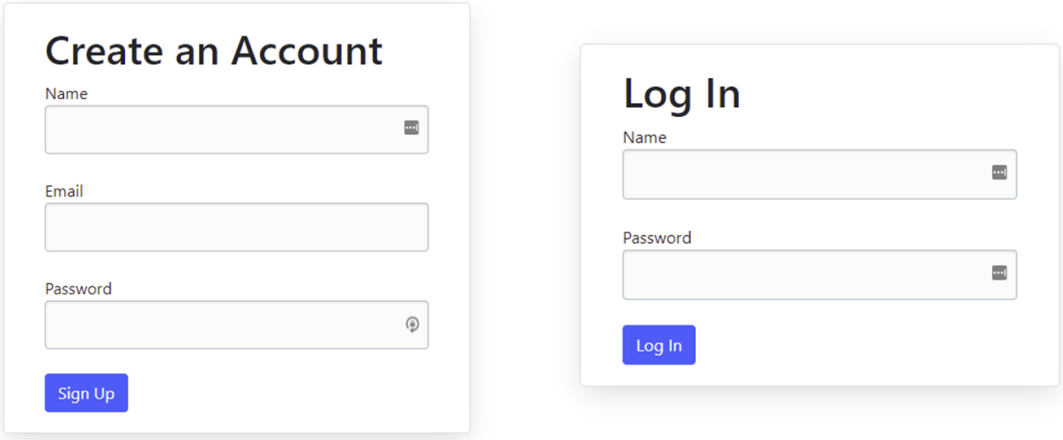


Figure B.1: Register and login

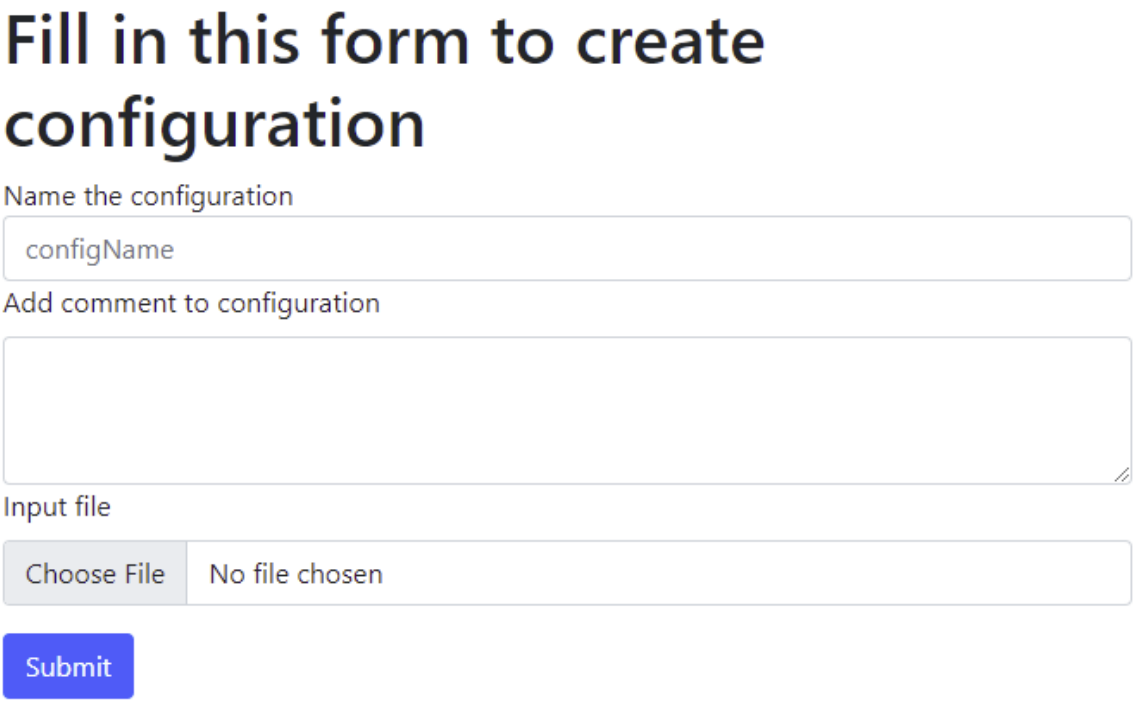


Figure B.2: Form for creating the configuration

Configuration name : test2-new

Download library

Name	ID number	priority	area	data type	size	transmit	receive
Star_button	1	1		uint8	1	STEERING	VCU
Reset	2	1		uint8	1	Datalog	ESP
Reset	2	1		uint8	1	Datalog	HV_Box
Reset	2	1		uint8	1	Datalog	Pump
Reset	2	1		uint8	1	Datalog	Steering
Reset	2	1		uint8	1	Datalog	VCU
VCU_status	501	2		uint8	1	VCU	Datalog
Pedal_status	502	2		uint8	1	VCU	Datalog
NonCrit_error	1000	3	AMS	uint16	2	VCU	Datalog
TS_on	1001	3		uint8	1	VCU	Datalog
Battery_current	1500	4	AMS	int16	2	VCU	Datalog
Battery_voltage	1501	4	AMS	uint16	2	VCU	Datalog

Figure B.3: Table of all messages from the configuration

Configuration Name	Author	Date	Comment	
test	user1	2022-05-09	Configuration for only testing	Load
test2	user1	2022-05-09	new testing configuration	Load
test3	user2	2022-05-09	new testing configuration	Load

Figure B.4: Recent configurations

Device name	Total messages	Filter usage
VCU	14	5 / 24
HV_Box	4	2 / 24
Steering	5	2 / 24
Pump	3	1 / 24
ESP	5	2 / 24
Datalog	21	6 / 24

Figure B.5: Analysis of the filters