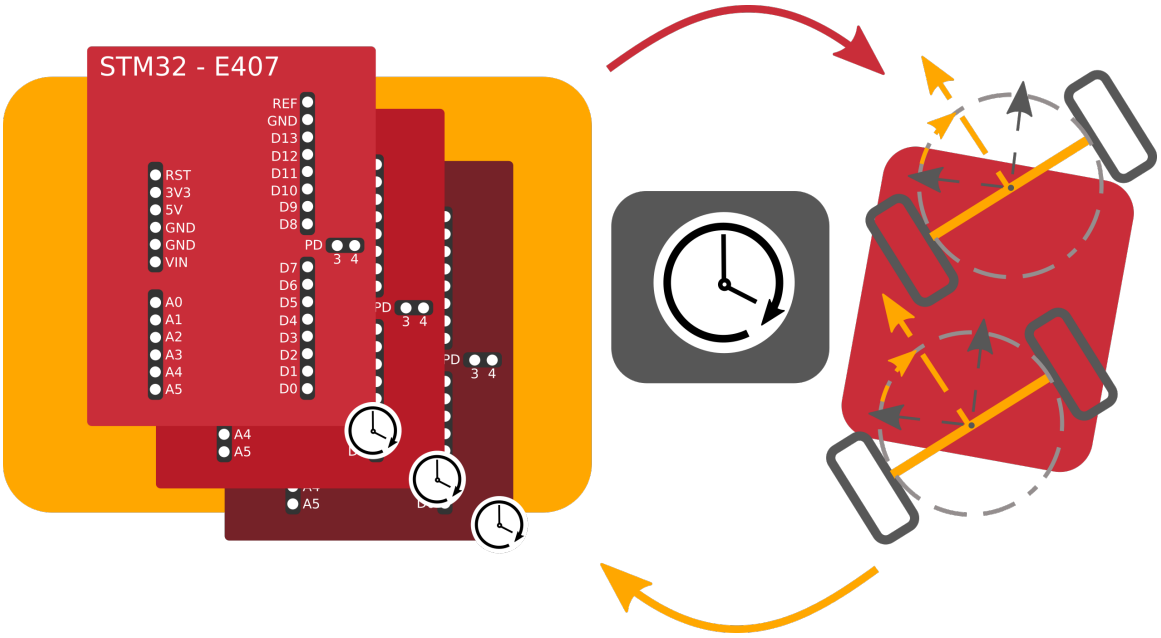
HAN_UNIVERSITY OF APPLIED SCIENCES

HAN MASTER MAJOR PROJECT

Deployment of a Time-Triggered Architecture schedule for a fully redundant Real-Time Embedded System with MATLAB-Simulink



Diego Martín López 659292

Company supervisor: Aart-Jan de Graaf – lector Meet- en Regeltechniek
HAN supervisor: Sanket Dutta – researcher at HAN

Arnhem, 2nd of February 2022

Summary

This project aims to prove that it is possible to develop and deploy a Real-Time distributed controller using HANcoder, an extension of Matlab-Simulink developed by the HAN. This is an interesting topic for the HAN's embedded system department as they can use the findings of this thesis as educational material for the Master in Engineering Systems Embedded Systems (MES-ES) module. It is also interesting to develop a tool that allows building Real-Time embedded applications within the HANcoder environment as this will open up a new field of projects to develop.

The test case chosen to be developed as a template model focuses on a Time-Triggered architecture schedule for a fully redundant embedded controller for a two-axle vehicle. The embedded systems department already had research done regarding this topic and the Minor Project from the course 2021 in the MES-ES module set the foundation on generating global time in an ensemble of boards using the CAN communication protocol. This project follows up from both pieces of research and combines them developing the controller for a three boards ensemble, so triple modular redundancy can be performed over the output signals to the vehicle.

The Time-Triggered schedule is divided into two basic cycles. The first basic cycle's purpose is to ensure that the boards are synchronized and to set the time master of the ensemble. In an ensemble of boards it is important to set a global time so all the components in the system follow the same time frame. During the first basic cycle the three boards vote which of them should be the time master of the ensemble, i.e. in charge of setting the global time. To improve redundancy the communication between the boards is done using two CAN channels sending the same information. Every message is also transmitted several times during the same communication task to improve the chances of being received.

The second basic cycle of the Time-Triggered schedule is devoted to perform the controller calculations. Apart from the controller boards, the test case system is compounded of two more boards: one to generate the input signal and the other as a vehicle emulator. The controller receives the steering and speed signals from the input generator and the wheel displacement from the vehicle emulator. Then the three controller boards make the same calculations to generate the new torque for the vehicle wheels according to the input signals and their current state. The two controller boards that are not acting as time master send their respective calculations in two separate communication tasks so that the master can assess their validity. With the triple modular redundancy the master checks whether the three calculations agree or not. As long as two of them agree, the output is considered valid and sent to the vehicle emulator.

The Real-Time embedded controller has been successfully developed and deployed. The vehicle emulator has been observed to follow the input generator signal with a PID controller of 50 ms cycle time. This proves that the concept is feasible and functional. However, there are three main issues regarding time constraints and communication that limit the prototype's potential. Firstly, there are missing messages during the boards' communication exchange. While the boards are synchronized, it has been observed that some messages are not received by all other boards. Secondly, there is a high computational load during message transmission, so the system cannot run with a faster execution time. Lastly, the current time master election policy requires a lot of time thus it is advised to optimize this procedure. Saving up time in the presented controller operations is key to allow for more complex projects to use the research of this thesis.

Abbreviations

AVG	average
COMM	communication
COMP	computation
MAX	maximum
MIN	minimum
Rx	reception
sync	synchronization
Tx	transmission

Acronyms

CAN	Controller Area Network
FreeRTOS	Free Real-Time Operating System
HAN	Hogeschool van Arnhem en Nijmegen
IRQ	Interruption Request
LT	Local Time
MES	Master in Engineering Systems
MES-ES	Master in Engineering Systems - Embedded Systems
OS	Operating System
PID	Proportional - Integral - Derivative
RTES	Real-Time Embedded System
TMR	Triple Modular Redundancy
TO	Time Out
TTA	Time-Triggered Architechture
TTCAN	Time-Triggered CAN
TW	Time Window
WCET	Worst Case Execution Time

Contents

1 Introduction

This project aims to develop and deploy an embedded controller for a two-axle vehicle within the MATLAB-Simulink environment. The focus of the project lays on the temporal aspects of the controller, mainly cycle time, ensemble precision and temporal control. This is done to prove that it is possible to control a vehicle using this methodology and to finish a software ready for further study, for both education and research. The temporal part of the software is designed in such a way that it is possible to build and design different Real-Time applications upon it using MATLAB-Simulink. In the next sections, this topic is introduced step by step.

1.1 Background

Distributed embedded systems popularity is rising as computer power becomes stronger and cheaper. When designing an application for any field that involves some risk, either on users or products, safety is a very important requirement. One way of achieving safety is by obtaining robustness with redundancy, in other words, having more than one point where the system can fail, so another part of the system can take over and resume operations. Embedded controllers rely on this idea, that is why several processors are temporally synchronized and merged in an ensemble with several failure points. The HAN embedded systems department is particularly interested in developing this technology using MATLAB-Simulink, concretely HANcoder, which is an extension of Simulink developed by the HAN. This is because this platform is being used for both educational purposes and research.

Embedded system projects usually require an interdisciplinary background. A software design methodology that helps with department communication and system integration is model-based design (Gérard et al., 2010). Embedding software has traditionally required specific text programming (usually using C) based on some previously designed specification. However, model-based design is becoming more popular as an alternative. In this other method programming blocks are joined together in a model directly resembling the specification. HAN University of Applied Sciences has developed extra layers of functionality over MATLAB-Simulink with HANcoder and HANTune, which provide blocks for, among others, Controller Area Network (CAN) communication, and allow for hardware in the loop testing, respectively. This helps in speeding up the development process flashing the embedded software onto the hardware and allowing for fast prototyping.

After rigorous research, it has been found that Real-Time Embedded Systems (RTES) development with MATLAB-Simulink is either based on simulation or lacks some key features we consider important, such as external clock activation or global time establishment over a distributed system. When dealing with an ensemble of boards that have to be synchronized, it is important that all the boards run their programs with the same *time* or rhythm, which is called granularity. There is a limit of 10 kHz on MATLAB software granularity and no built-in option for execution under a hardware clock interruption basis has been found. This latter would allow for the granularity to be set by an external clock. The STM32-E407 boards employed in this project are also not part of the supported hardware by the official source (MathWorks, 2021). There is some work done regarding scheduling with a MATLAB toolbox described in (Sivakumar et al., 2015) or (Vora et al., 2009) and for Linux target RTES, as can be seen in (Quaranta & Mantegazza, 2002a), but there is no deployment on any STM32 boards (apart from the discovery series). There are also no flexible proposals to develop Real-Time distributed applications with a hardware clock synchronization. The work proposed in this project plan will develop some tools to cover these functionalities to solve a technological problem and thus allow for further development works with HANcoder on Real-Time distributed applications.

This Major Project does not start from scratch, as it is based upon the Minor Project of the embedded systems module of the Master of Engineering Systems at the HAN. The Minor Project for the embedded systems module proposed the development of a two-axle vehicle controller within a fully redundant context using MATLAB-Simulink and HANcoder. More information about the two-axle vehicle is presented later in chapter 2. The controller, including the motion controller and the motion estimator, processes the signals from the input generator towards the vehicle emulator, as presented in figure (1). However, there seems to be no evidence yet that a Real-Time distributed controller can be actually programmed within the MATLAB-Simulink and HANcoder environments. The initial version of the software allowed for Time-Triggered message communication among three STM32-E407 boards using CAN with time master voting logic in the ensemble. This Major Project focuses on the latest phases of the Minor Project proposal, building upon the initial version of the software and including deployment of the controller.

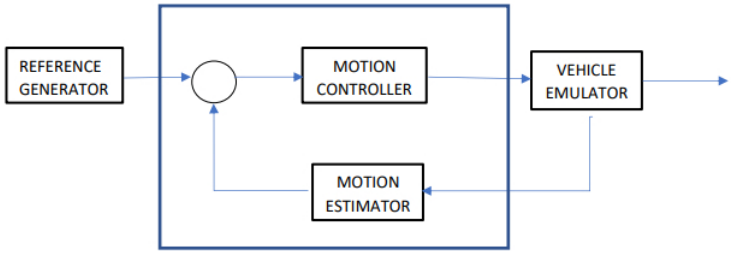


Figure 1: High-Level overview of the control structure. The blue box encircles the controller functionality. This figure was taken from the Minor Project description in the MES-ES.

There are two important aspects already implemented in the Minor Project that help in the robustness of the Major Project development. Using a Time-Triggered schedule requires thinking prior to software execution when every different task of the program shall be executed. This makes the design more complex but allows the controller to have a deterministic behaviour. The choice of using three boards allows to use triple modular redundancy in later phases of development. This is used during this project for the value domain (the controller calculations) to ensure that, even if one of the three calculations from the boards (one calculation per board) is incorrect, there are two calculations that are consistent with each other. These two ideas are further explored and explained in chapter 4.

The controller will not be deployed in a real car but it will just be tested under a computer simulation, with the controller fully deployed within several STM32-E407 boards. The signals input to the controller and processed by it will be managed by computer emulators. The different components of the system already exist in a MATLAB-Simulink program without temporal elements taken into account. The project focuses on (1) the development of a time synchronization software that can be used as a template in the future for further development of RTES, (2) the adaptation of the already existing software of the signal generator, controller and vehicle emulator to fit within the time synchronization software framework and (3) the deployment of the whole system in an ensemble of boards, using a test case to demonstrate that the software is able to control a two-axle vehicle. The test case is studied and analyzed exploring the different scenarios where failures could happen (using fault injection) and either visualizing or recording the states of the different variables (monitoring).

1.2 Problem definition

As already stated in the previous chapter, no evidence has been found that a Real-Time Time-Triggered fully redundant controller can be deployed in STM32-E407 boards using HANcoder. This Major Project is devoted to develop and deploy a two-axle vehicle controller using HANcoder and make further time analysis on it. Once the deployed controller is appropriately working, it is of major relevance to check the system granularity and ensemble precision in order to assess the controller validity for future real life applications.

1.3 Research questions

This project is born from the following research question:

- *Is it possible to deploy the controller in the STM32-E407 boards with Real-Time communication using HANcoder?*

Some sub-questions that will help on the resolution of the main question are the following:

- *What is the minimal granularity of the software?*
- *What is the communication delay?*
- *How many nodes does the system need in order to be fully redundant?*
- *What is an appropriate frequency for the input signal to the controller?*
- *Is there any overhead in the controller because of the signal processing that may delay the Time-Triggered communication?*

This project aims to answer affirmatively to the main question and also go beyond and ask:

- *Is the time response of the emulator reasonable with the deployed controller?*

This is especially important to check, as it will influence further development of RTES using HANcoder in the future.

1.4 Project objective

The main objective of this project is to develop and deploy a two-axle vehicle fully redundant distributed controller with Time-Triggered communication.

1.5 Project scope

It is important to set expectations on the scope of the project as well as to define the difference between the company request and the university's Major Project scope.

The scope of the project is set with the controller being deployed and tested on the boards using both HANcoder and HANTune for the embedded systems department and future projects designing Real-Time Systems. It would be optimal to build and try out its performance on a real vehicle, but the time constraints of the project limit the development to just testing and observing the vehicle emulator signals.

Regarding the master level versus project goal, software documentation is important for the final product, but it is not required for showing the master level of the project. Furthermore, in order to solve the embedded systems department problem, it might not be necessary to develop a controller using a Linear Quadratic Regulator (as originally stated in the project plan), but something less sophisticated may work. Designing a more complex controller could have helped towards showing master level, but after discussing with the stakeholders, it has been decided to invest all the attention on deploying the controller. Furthermore, developing the software required to make the distributed controller work, dive into the different Real-Time aspects of the system and investigate the deployed controller behaviour is sufficient to show master level, as in-depth knowledge from the embedded systems module is required. A PID controller is sufficient to test the success of the software, as it is only required to verify that the vehicle emulator follows the signals from the input generator. It was advised to try development of other controllers only if there was spare time at the end of the project. Further investigation regarding communication problems was required at the last stage of the project instead of developing other controllers.

1.6 Project outline

It is important to start defining the two-axle vehicle specification, as it is not a common vehicle and yet it is the target for the controller. Both the vehicle description and its control requirements are discussed in chapter 2.

The literature survey in chapter 3 focuses primarily on the security of embedded systems, vehicle distributed controllers and Time-Triggered communication. The controller application aimed to be designed in this project has to be redundant and fail-proof. This is why the main topics follow a connection from distributed applications to Time-Triggered communication covering subjects such as triple modular redundancy and task models.

The methods from the literature survey form a base for the project methodology in chapter 4, which first describes the different components of the system, along with the software and hardware limitations. Then it analyzes how the thesis problem is solved from a chronological perspective: first, the time synchronization fundamentals of the boards are developed, then the distributed controller is integrated into the time model and lastly, the controller, input generator and vehicle emulator are deployed. At the end of the chapter the different monitor strategies and fault injection of the test case are discussed.

A brief software overview is presented in chapter 5. The main software structure and modules are presented along with the software activation hierarchy. This gives a better understanding of how the time is handled in the program, the differences between matrix cycles, basic cycles and tasks and the relationship between them and the CAN network interface.

The results in chapter 6 present all the measurements of the time analysis and the test case. The time analysis is compounded of the following points:

- System granularity: assessing how frequent the hardware interrupts of the system are.
- Ensemble precision: that looks into the synchronicity of the boards.
- Communication delay: to see how long does it take for a message to travel from one board to another.
- Tasks' computational time: so the execution time can be studied.
- Basic cycle duration: optimized based on the previous points.

The test case results help diving into the performance of the controller on the vehicle response. In the end, fault injection results are also presented.

The discussion in chapter 7 goes through all the results information and dives into details analysing each of them, so the performance of the system as a whole can be assessed. In this chapter the original research questions are also reviewed, hence it is possible to see to what extent the main question has been answered. Finally, the conclusion in chapter 8 gives a summary of the discussion and presents some future recommendations.

2 Two-axle vehicle specification

As its name already states, the two-axle vehicle counts with two axles, the front (F) and the aft (A). They can move independently from each other and rotate 360°. The angular displacement from their rest position is denoted with the letter δ. A diagram showing the vehicle scheme is presented in figure (2). The diagram is presented from an observer's point of view O, from which the vehicle V is located s units apart at an angle φ, moving at a rate $\dot{\boldsymbol{s}}$.

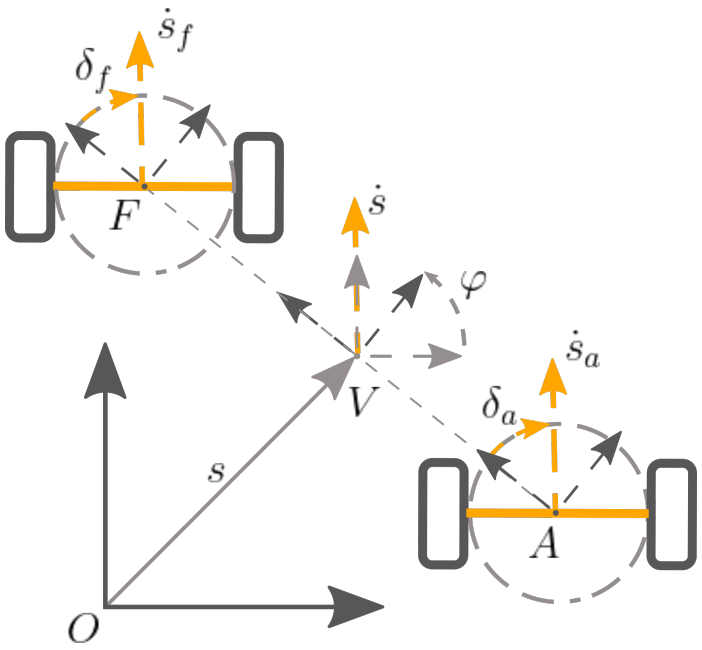


Figure 2: Two-axle vehicle system diagram. The vehicle main frame V moves at velocity $\dot{\boldsymbol{s}}$, the front part F at $\dot{\boldsymbol{s}}_{\boldsymbol{f}}$ and the aft A at $\dot{\boldsymbol{s}}_{\boldsymbol{a}}$.

The controller aims to control the vehicle sending torque signals to each of the wheels. For this, a reference generator will send the user's desired values for speed and steering, and a vehicle emulator will send the sensor values of the current displacement in each wheel. The controller must process these signals and convert them into torques. Further explanation on this subject is treated later in chapter 4.4.

It is important to decide the frequency at which the controller is going to send new information to the actuators in the vehicle emulator. According to Aart-Jan de Graaf, company supervisor and main stakeholder regarding the vehicle control, the frequency control for propulsion is advised to be 10 samples per second, and for steering 20 samples per second. Because the controller does not discern speed from steering when sending the output signal, the control loop's frequency must be the fastest of the two, 20 samples per second. The feasibility of this is presented in chapter 6 and discussed later in chapter 7.

3 Literature survey

This project system is compounded of several electronic boards that may run at the same time to send a consistent output to the vehicle actuators. This is a distributed vehicle controller that requires deterministic behaviour and synchronous communication.

There are many advantages when choosing a distributed control for a vehicle, such as optimization, robustness, simplified maintenance, standardization, economy or comprehensiveness, as presented in (Callen, 1998). This thesis focuses on the robustness aspects of the controller. There are different strategies to ensure robustness in a system such as fault containment units and triple modular redundancy (Kopetz, 2011a), along with Time-Triggered communication (Kopetz, 2011b). The main controller is deployed onto three independent boards that can reset when a fault is detected. These boards make the same calculations to make sure the output value is correct even if one of the boards has failed. These mentioned strategies together with double channel communication are employed in this project to make the system fully-redundant.

Vehicle controllers are usually decoupled into a velocity (longitudinal) controller and a steering (lateral) controller, as it can be seen in the proposals from (Samak, Samak & Kandhasamy, 2020) and (Shahian Jahromi et al., 2017). These articles also pay special attention to the vehicle physical limitations and define the output limits of the actuators. The controller presented in this thesis is also internally divided into steering and speed but the physical analysis is different from common vehicles, as the control is performed over a two-axle vehicle. The input and output ranges and units are already defined by design. The controller cycle frequency for autonomous vehicles is usually recommended to be 100 Hz, as can be seen in (Göhring, 2012), but Aart-Jan de Graaf, the company supervisor of this thesis, has personally recommended that 20 Hz is enough for this project's application.

An example of a distributed controller for an autonomous ground vehicle can be found in (Baek et al., 2008). The vehicle tasks in this article are very complex, as it is an all-terrain vehicle and requires communication among different sensors and processors. Communication is held via CAN-network. When looking at other automotive protocols for communication, (Mathur, Saraswat & Mathur, 2014) presents other choices apart from CAN, such as Local Interconnect Network, FlexRay or media-oriented system transport.

The system of this thesis uses Time-Triggered communication over CAN, which is very similar to the Time-Triggered CAN (TTCAN) protocol presented in (Leen & Heffernan, 2001). The communication protocol employed using HANcoder cannot follow all the specifications of TTCAN, as later explained in chapter 4.3.2. The main feature both protocols have in common is that they set an extra layer over CAN with a Time-Triggered offline schedule.

To find an appropriate schedule it is important to define how the communication tasks are going to be handled in the system. The model employed in this project is based upon some ideas proposed by (Freier & Chen, 2014). It defines a phase parameter ϕ to decouple the computation from the communication of a task. This project uses this parameter as a delay to separate when the communication task starts and when the message is actually sent. More information about the task model is discussed later in chapter 4.3.3.

The Time-Triggered schedule for this project requires taking into account the computational and communicational aspects of the tasks. The article by (Craciunas, Oliver & Ecker, 2014) dives into the mathematical analysis of a schedule with computational tasks that depend on the network message transmission, either receiving messages, transmitting them or both. It also proposes an optimal scheduling strategy considering the message-computation relationships and tries to find a solution with a minimum matrix period cycle.

The schedule of the project is simple enough not to require an algorithm to set it but it takes ideas from (Craciunas, Oliver & Ecker, 2014) to set the correct order of tasks and dependencies among calculations and messages. This article also discusses the challenges they encountered when deploying a real-time system, such as the increase in the time of the computational tasks due to the overhead in the deployed Free Real-Time Operating System (FreeRTOS). Even though they calculated the Worst-Case Execution Time (WCET) with analytical techniques, the resulting measured time of the tasks period when the system was deployed was higher. The system for this thesis is also deployed over a FreeRTOS and, as described in (Craciunas, Oliver & Ecker, 2014), making analytical calculations of the WCET is very hard. As proposed in (Rapita Systems, 2021), the execution time of the tasks will be measured and a 20% safety margin will be added to the result to estimate the WCET of each task in the matrix cycle.

There is also work made around the creation of real-time schedules for distributed embedded applications using MATLAB. As described in (Quaranta & Mantegazza, 2002b) and (Palaniswamy et al., 2015), there are specialized toolboxes to work with schedules and resource sharing. However, this project focuses on the development of the software just using basic Simulink and HANcoder blocks.

4 Methodology

The objective of this project is to deploy a distributed controller with Time-Triggered communication for the two-axle vehicle described in (de Graaf, 2017). In order to achieve this the project development is divided into different phases that shape the parts of the system:

1. Time-synchronous system.
2. Time-Triggered controller.
3. System deployment and testing.

These phases are presented and explored in this chapter. The different reasoning behind each development choice, deployment procedure and the testing option is explained in each section. As an initial overview, a scheme of the whole system and its components is shown in figure (3). This scheme shows, from top to bottom, the software, hardware and monitor components, respectively.

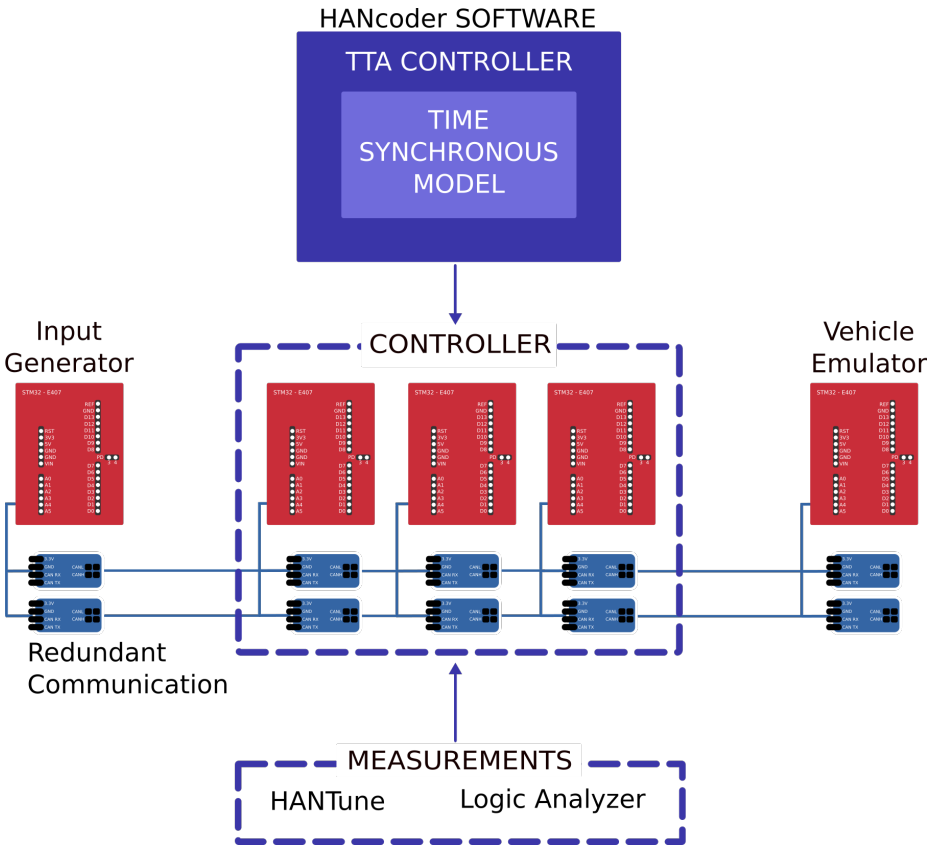


Figure 3: Overview of the whole system. The red components are STM32-E407 boards that contain the system's software. The blue components are CAN transceivers that set a double CAN communication network and allow the boards to communicate with each other.

The system software is divided into the time-synchronous model, which is in charge of keeping the controller boards synchronized, and the Time-Triggered Architecture (TTA) controller software, which processes the inputs from the two-axis vehicle and converts them into the signals output to the vehicle emulator. The hardware components consist of five boards, one for the input generator, three for the controller and one for the vehicle emulator, and ten CAN transceivers, for double CAN communication. There are three controller boards to have triple modular redundancy on the controller decisions and double CAN communication to have redundancy on the message exchange between the boards. The monitor components employed in this project are HANTune, with which it is possible to see the software signals in real-time, and a logic analyzer with eight channels, which allows for monitoring digital signals on the boards. HANTune is very helpful to see how the software signals change while the code is running, but its sample time is limited to 100 Hz and it increases the system overhead. The logic analyzer is used when it is necessary to check the behaviour of the system at a task activation level.

Before diving into the different project phases, it is important to declare a set of hardware and software features that determine the boundaries of the possible performance of the current version of the prototype. After these limitations are presented, an overview of the core ideas behind the Time-Triggered schedule is described. With these ideas in mind, the synchronization basic cycle, the controller basic cycle and the input generator and vehicle emulator matrix cycles are explained in order. Finally, the methodology is finished with a description of the system deployment and testing strategies.

4.1 Limitations

The main hardware choice is the STM32-E407 board, as it is the only board supported by HANcoder to use CAN communication with two CAN channels. Both the hardware and software choices by the client impose some limitations on the project. For instance, HANcoder and HANTune are the sources of the following ones:

1. HANcoder deploys FreeRTOS with its software. The operations of this Operating System (OS) create a big overhead on the processor when the hardware Interrupts ReQuests (IRQs) of the software are too frequent. It has been observed that the conflicts between OS and IRQs happen with 100 kHz granularity on the interrupts.

2. HANTune requires connecting the development computer with the board to monitor the software signals. These connections employ some of the FreeRTOS resources for communication, so using HANTune while the IRQs are frequent increases the overhead even more.

3. Software development with HANcoder is limited by the blocks offered by the program. One could develop more blocks by coding with MATLAB or C, but the client suggests avoiding this when possible to prioritize software model development. The software proposed is based on the creation of new functionalities by using the current blocks, but some high-level blocks allow for little extra configuration, such as the blocks regarding CAN communication.

4. The software governing the two CAN channels in HANcoder does not allow them to be independent, so redundancy using these two channels does not improve the robustness of the system. This is because both channels share a buffer at a low C programming level, thus if one of them fails, the other stops working too. This is ignored during the rest of the project as the client has already contacted the HANcoder development team to correct this problem. It is expected that newer versions of HANcoder will allow for using both CAN channels for robustness, but it is not clear when this will happen. A prototype has been developed for this project soldering all the CAN communication connections to reduce the chances that a communication problem arises because of loose cables.

The main hardware limitation is imposed by the STM32 boards. The number of connections of the board limits the number of elements that can be connected to them. At the moment the required connections are four pins for communication and three pins for unique board identification. This hardware limitation does not have a heavy impact on the project. This is because there are enough connections in the boards for the required functionalities. However, this may change in case this project had to be extended in the future.

4.2 Time-Triggered schedule overview

A Time-Triggered schedule is compounded of different tasks, either for computation or communication, that must happen in predefined time windows (TW). The moment in time, when each TW starts, is called time mark and they are numbered along a basic cycle. The project has a task schedule (or matrix cycle) that counts with two basic cycles, the first for time synchronization and role selection (presented in section 4.3.2) and the second for the controller calculations (presented later in section 4.4). An example of a standard matrix cycle is presented in figure 4.

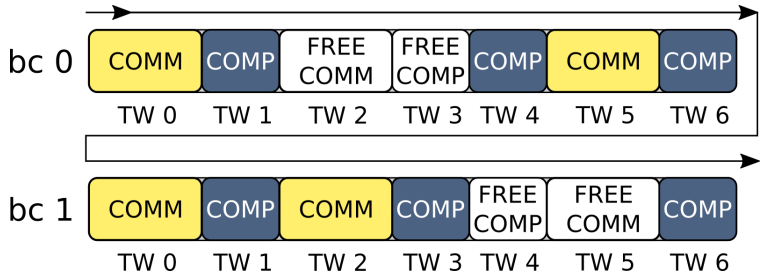


Figure 4: Example of a matrix cycle. It is compounded of two basic cycles (bc0 and bc1) with a series of time windows (TW). The different tasks presented are either communication tasks (COMM) or computational tasks (COMP). Free windows for computation (FREE COMP) and communication (FREE COMP) are also included.

The matrix cycle example from figure (4) is read from left to right and from top to bottom, as the arrow path shows. There are two different kinds of tasks: communication and computation tasks. During a communication task, one of the boards transmits a message and the others wait for the message to be received. Every message is transmitted several times during the same communication task with the same information, to raise the chances that the message reaches its destination. In this project proposal every communication task is followed by a computational task that processes the contents of the message (if any was received). A computational task handles any kind of operation that does not involve communication. There can be free time windows, where there is no task scheduled for that period.

The basic cycles of a matrix cycle usually have the same temporal length. This is because the length of a basic cycle is in most cases determined by the time required to synchronize the boards again. Since each board has its own local time governed by its own clock, the frequency of the clocks of the different boards does not have to be exactly the same. This can cause a drift between the boards. This is why the boards are not synchronized just once, but regularly, at the beginning of every basic cycle with what is called the reference message. This reference (or sync.) message contains the global time information from the time master board to ensure all boards remain synchronous. This project does not follow the idea that both basic cycles shall have the same period though. The limitations found in communicational and computational time (presented later in chapter 6) combined with the recommendation of making the complete matrix cycle frequency 20 Hz do not leave much spare time for free windows during the controller matrix cycle. Also, the drift of the boards is not as important as to desynchronize them before a new reference message arrive. This is why it was preferred to divide the matrix cycle in its basic cycles regarding task objectives instead of precision time for scalability and system comprehension.

4.3 Time synchronous model

The system boards have to send and receive messages independently at some precise moments in time. This requires them to share a consistent global time. The global time is set by one of the controller boards, the master, and followed by the other boards in the ensemble, the slaves. The basic cycle that ensures there is an agreed master is presented in section 4.3.1. It is essential that in case the master fails, one of the slaves takes its place as master so that the system does not stop working synchronously. The reasons behind choosing the strategy followed in section 4.3.1 are discussed in the Master-Slave decision strategy section 4.3.2. Every message sent and received by each node is planned to happen within the same framework, called the communication task model. This model is presented in the Time-Triggered communication task model in section 4.3.3.

4.3.1 Synchronization basic cycle

When the system boards are turned on, they wait to receive a reference message from a master that has awakened before them. If a board from the controller does not receive any reference message during a matrix cycle time, it will proclaim itself as the master of the ensemble and will start the first basic cycle. If some other board was already listening by this moment they synchronize. From then on the system will execute the tasks from the matrix cycle consecutively. In case the time master of the ensemble fails to share the reference message, one of the other boards should take the master role instead. The strategy followed to ensure that there is a time master board sharing a correct time reference at the beginning of the controller calculations is based upon an election held among the three controller boards. This poll is performed during a whole basic cycle, as represented in figure (5).

The first task activating at the beginning of the basic cycle is the communication task *Sync bc0* for board synchronization. During this communication task, the master shares its local time with the so-called reference message, thus the slaves can synchronize with it. Every communication task is followed by a computational task, *Check COMM*, to analyze the messages received by both CAN channels (if any). In case any message has been received, or none of the messages received is considered coherent, a failure will be acknowledged and the message will be considered as missed. If both messages are received and are coherent, the message from CAN channel 1 is processed.

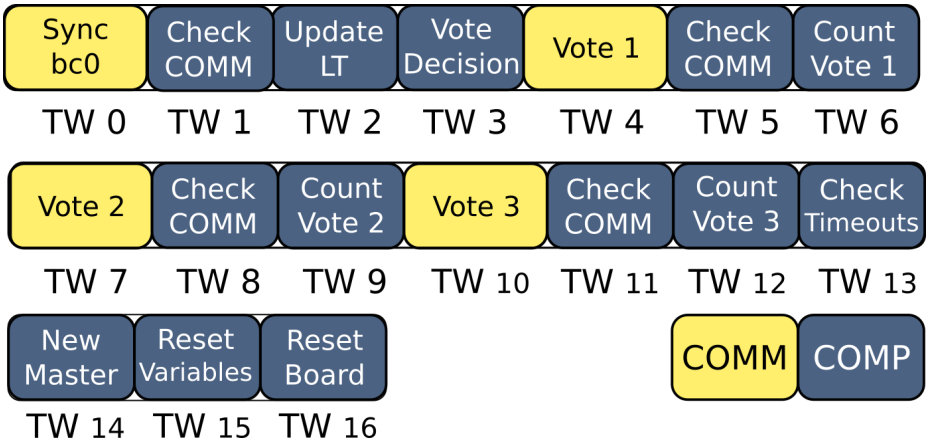


Figure 5: Basic cycle scheme for the election process when choosing a time master. Communication (COMM) tasks are painted in yellow and computational (COMP) tasks are painted in blue.

The slaves update their own local time during the *Update LT* computational task. The local time update takes into account the difference between when the reference message is expected to be received and when it is actually received. This difference is called the desynchronization of the board. If the desynchronization is negative the local time is reduced and none of the already performed tasks is repeated. If the desynchronization is positive the local time is increased in small steps to ensure that no task is skipped. However, if the reference message is either regarded as incorrect or has not been received at all, the slave board considers the master at fault and decides to vote for another board to be the new master. The vote decision happens during the *Vote Decision* computational task. When the time reference is received and it is consistent with the local time, then the vote is cast for the current master. Otherwise, the board with the lowest error count apart from the current time master is chosen. The error counter increases for each board every time a message from that board is not received. The criteria on how to choose which board to vote are depicted in a flow chart in appendix A.

The communication tasks that are used to exchange the voting information are *Vote 1*, for the vote of board 1, *Vote 2*, for the vote of board 2 and *Vote 3* for the vote of board 3. The board transmitting the board information (for example, board 1 during *Vote 1*) broadcasts the board ID of the board that it believes should be the master board for this matrix cycle to the other boards. The boards receiving information activate the corresponding *Count Vote* computational task to take that vote into account for the election.

In the *Check Timeouts* computational task it is considered which messages were not received during the basic cycle so the error counter for the board whose message was not received is increased by one (fault detection). If no messages have been received by a board during this basic cycle a flag is set to reset the board at the end of the basic cycle. In the *New Master* computational task the master is chosen regarding the votes received from the election. During the *Reset Variables* task different variables are reset to make the board ready for the next basic cycle. If the reset flag is set, in the *Reset Board* computational task the variables in charge of initialization are reset and the board returns to a state in which no role is yet assigned, and waits until a reference message is received.

4.3.2 Master - Slave decision strategy

The TTCAN article (Leen & Heffernan, 2001) presents a simpler strategy to choose a new master when the current master fails. This procedure assumes every message sent to the CAN network is broadcast and has a certain priority. However, HANcoder CAN blocks cannot fulfil these two requirements at the same time.

A message can be sent with a specific ID, which sets the message priority, but another board that wants to receive this message has to check for messages with that specific ID in the network. This means that messages are not effectively broadcast. For the broadcast to be effective, every time a message is sent an interrupt should happen in the receiver end of all other boards. It is also important that checking if a message has arrived happens as soon as possible after the message has actually arrived, as otherwise the board receiving the message cannot compare its own time with the reference time received.

It could also be possible to send all messages with the same ID. This way the boards do not have to pay attention to different IDs, but this would mean that all messages have the same priority, so in this case the strategy from (Leen & Heffernan, 2001) could not be performed either. The strategy proposed in section 4.3.1 is complex and time-consuming, but it ensures that in case of time master failure one of the other boards will take its place in a democratic and organized manner. It is also important to remark that for the message exchange to work every board must be synchronized, otherwise, they could miss the messages sent. The moment a board does not receive messages from any other board, it performs an auto-reset (*Reset Board* computational task), so it can synchronize again with the next reference message.

4.3.3 Time-Triggered communication task model

Every communication task in the program is defined by the same model, either if the board executing the task is transmitting or receiving. A board that should transmit a message during a communication task will wait until the appropriate moment to send it. If more than one message is programmed to be sent during the same task, these are sent in the appropriate moments through the correct channels. A board that is supposed to receive a message during the communication task will ask the CAN interface at every IRQ to check if a new message has arrived.

A communication task has a time limit τ beginning from the moment it starts, called period. The message is sent after a known time ϕ since the task starts, called phase. The time δ between transmission and reception is the communication delay. If the board is transmitting it sends the message through the CAN send block ϕ ticks after the task starts. If the board is listening it checks the CAN receive block every cycle from start until the message is received or a maximum time of τ ticks passes. If the message is not received at this time, it is considered lost and the matrix cycle keeps going on. If the message received is the master's reference time, analysis on when the message was received is held in order to make any changes on the current local time if necessary. This requires the communication delay to be known, a value presented later in chapter 6. An example of how the communication is held with three boards with just one message being sent through one channel is presented in figure (6).

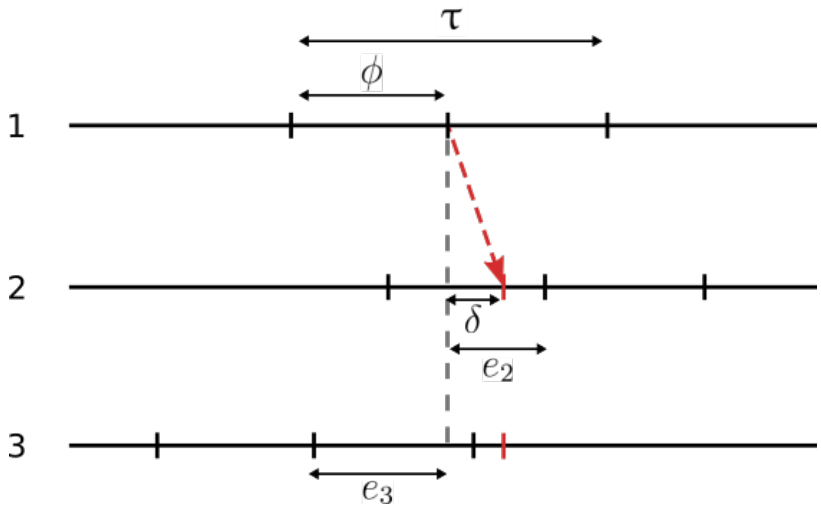


Figure 6: Communication task example between three nodes.

In the example from figure (6), board number (1) works as the master while boards (2) and (3) are slaves. Boards (1) and (2) are synchronized well enough (the time error e_2 is smaller than $\phi + \delta$). This means board (2) receives the message and acknowledges the time error e_2, so it can correct it for the next cycle. Board (3) is not synchronized enough with board (1) (time error e_3 is bigger than $\phi - \delta$), so it misses the message. If board (3) does not receive any other message from the ensemble it will auto-reset at the end of the cycle. If it does receive messages from board (2) because it is better synchronized with it, board (2) will receive more votes in the next election, it will become the new master and all other boards will synchronize with it.

The example from figure (6) shows just one message being sent from board (1) to the others. With two CAN communication channels, this corresponds to the scenario when messages from both channels have the same phase ($\phi_1 = \phi_2$). However, in this project, these phases ϕ_1 and ϕ_2 are not the same, as this way the messages are more likely to be received by other boards. Moreover, more than one message is sent through each channel during a communication task, to further increase the chances of reception. The maximum amount of messages that can be sent through a channel has been set to seven. This is because only the first byte of every message sent has been allocated to temporal information. This temporal information includes the ID of the sender with 4 bits, the basic cycle in which the message was sent with 1 bit and the message count of the transmitted message with the 3 remaining bits. The other seven bytes in every message contain each an unsigned eight (uint8) data value. If the value that is transmitted was originally a float variable it has to be coded into an uint8 data value before the transmission and decoded back to float at the receiving end. The coding strategy chosen is explained in appendix F.

When communication is held, because of how the CAN blocks in HANcoder work, the receiving board should be waiting for the message in advance with the message buffer clean in order to receive both the value and the time when the appropriate message arrives. A state machine in three stages has been developed to get the correct message in time. First the CAN receive block is called to empty the message buffer, then the block is called repeatedly until a message arrives and, lastly, the message content is forwarded to the communication check task.

4.4 The controller in the Time-Triggered model

The distributed controller employed for this project is based upon a basic PID CAN distributed controller designed by (de Graaf, Dutta & Manani, 2020), in which both supervisors of this Master thesis contributed. This controller is internally divided into steering and speed controllers, using some basic calculations to convert the user set points and the sensor signals into the actuator signals for each wheel. A scheme representing the message exchange with the variable values among the reference generator, controller and vehicle emulator is presented in figure (7). Detailed information about the controller operations is presented in appendix C.

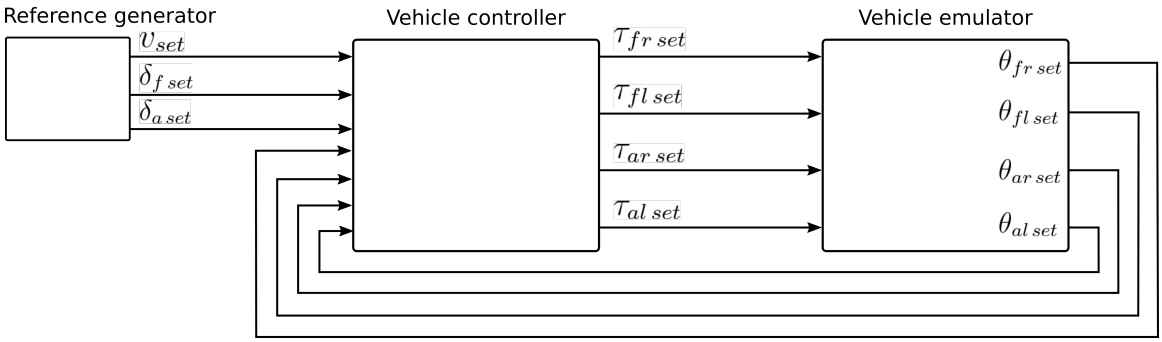


Figure 7: Signal exchange between reference generator, vehicle controller and vehicle emulator. The reference generator sends the set velocity v_{set} and the steering angles for the front $\delta_{f\,set}$ and for the aft $\delta_{a\,set}$. The vehicle controller receives the signals from the reference generator and the sensor values from the vehicle emulator, which correspond to the displacement θ_{set} on each of the wheels. The vehicle controller also outputs the torques τ_{set} of each of the wheels to the vehicle emulator.

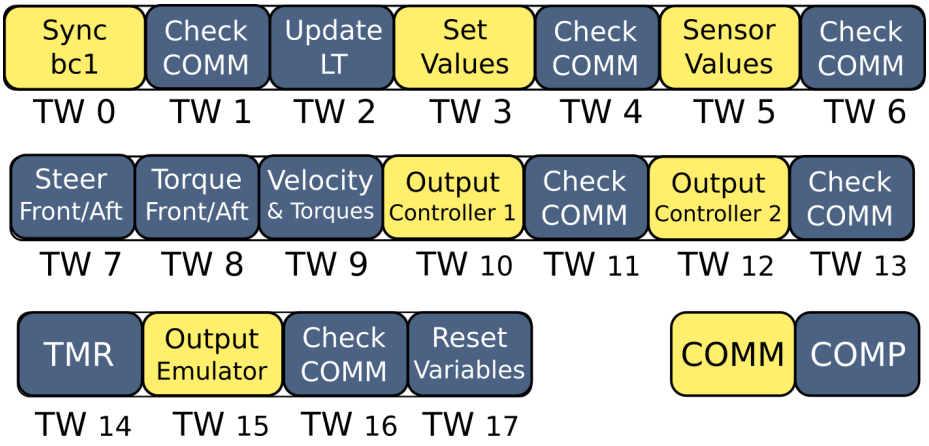


Figure 8: Basic cycle for the controller operations. Communication (COMM) tasks are painted in yellow and computational (COMP) tasks are painted in blue.

The controller operations basic cycle can be found in figure (8).

The beginning of the controller basic cycle is identical to the time synchronization basic cycle in figure (5). The master board sends the reference message with the global time and the slave boards update their own local time depending on the desynchronization. The next two communication tasks are the *Set Values*, to receive the information from the input generator, and the *Sensor Values*, with the information of the vehicle emulator.

From then on, the different operations to calculate the output signal are performed in consecutive tasks. These calculations are performed by all the controller boards, presumably getting (almost) identical results. When the calculations are done, the slave boards broadcast their output values during the communication tasks *Output Controller 1* and *Output Controller 2*. These values are received and checked by the master board to perform the Triple Modular Redundancy operation (*TMR* computational task). If at least two of the values are consistent with each other, the master will output the one received first at the *Output Emulator* communication task. In case that there were several faults in the calculation of the next output value for the vehicle emulator, the master board outputs an empty message. Apart from the output information for the vehicle actuators, the controller also sends information about the communication and TMR errors with an error log.

The complete system controller matrix cycle is compounded by the two already presented basic cycles: the master election from figure (5) and the controller operations from figure (8). At the beginning of each basic cycle, the global time is shared by the master to remove the time drift from the other boards. These basic cycles are executed one after the other in succession while the system is operating, starting from the moment the initialization finishes. A representation of both cycles together is presented in figure (9).

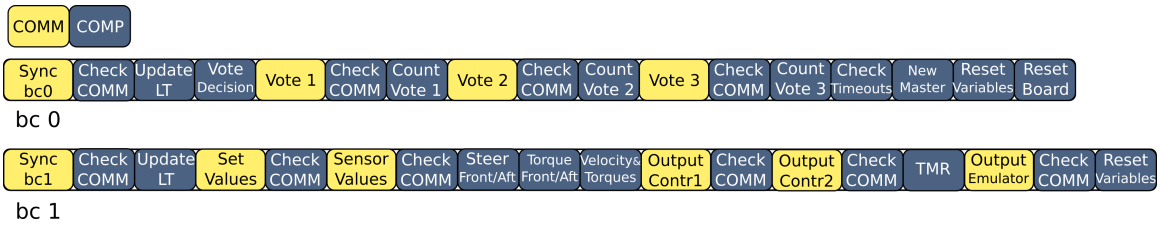


Figure 9: Matrix cycle of the vehicle controller. Communication (COMM) tasks are painted in yellow and computational (COMP) tasks are painted in blue.

Regarding software merging of the Time-Triggered controller into the time synchronous model, information on how the different basic cycles are managed can be found in chapter 5.

There is some common information required at the controller calculations for all the controller boards to get results as close as possible to each other: the integral sums. If a board stops its operation for some time and then gets auto-reset (or even hard-reset) it requires information from the current integral state of the system. The idea of the ground state is introduced to solve this problem. With every reference message, along with temporal information, value domain information with the integral calculations are shared from the master to the other boards. This way, a freshly synchronized board will have the right information to make the same operations as the other boards in the ensemble so the TMR operation regards those calculations as valid too. The current prototype proposal does not require much ground state information to be shared at every reference message, so a single reference message suffices to share all the required data.

4.5 The input generator and vehicle emulator

Both input generator and vehicle emulator were already designed in a Simulink model structured to follow Simulink time instead of hardware interrupts. For this project, it is important to redesign both for the communication between them and the controller to remain synchronous. Their operations were divided into tasks like it was done for the controller. However, they must do fewer calculations over the matrix cycle process, as they are not in charge of the time master decision and have less communication exchange with the ensemble. They synchronize their own local time every time they receive the reference signal, regardless of which board sent the message, and reset after doing their calculations if no reference message was received. It is important for them to expect the reference message at a specific time window thus they are required to follow a similar matrix cycle structure. Resetting as soon as possible when no reference signal has been received is important to focus on resynchronization. The input generator uses the set signal values defined in HANTune. These signals are sent to the controller in the *Set Values* communication task of the controller operations basic cycle, as it can be seen in figure (8). The input generator matrix cycle is presented in figure (10).

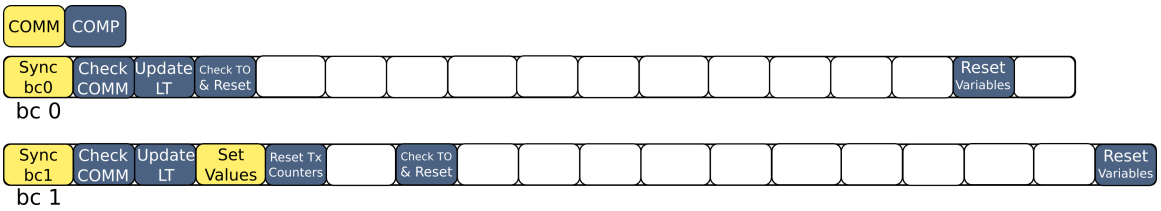


Figure 10: Input generator matrix cycle. Communication (COMM) tasks are painted in yellow and computational (COMP) tasks are painted in blue. White tasks are free windows to keep this matrix cycle synchronous with the controller's.

At the beginning of each basic cycle, the input generator receives the reference time and updates its local time, as all the other boards do. In the first basic cycle, after processing the reference message the time window *Check TO & Reset* activates. First, the Time Out (TO) for the reference message is checked and if no message was received the board is reset. At the end of the first basic cycle (bc 0), if the board was not reset the variables to prepare for the next basic cycle are reset, at the same time window as in the controller's basic cycle 0. In the second basic cycle (bc 1) of the input generator, after processing the reference message, the set values from HANTune are transmitted to the controller. The input generator does not have to process any message afterwards, because it was the board sending the message but it must reset its transmission counters at *Reset Tx Counters*. One more time, the time out for the reference message is assessed and if no message was received the board resets to pay attention to a new reference message as soon as possible. Again, if the board was not reset, the cycle variables are reset at the end, matching the controller's second basic cycle *Reset Variable* time window.

The vehicle emulator sends the sensor signals to the controller and receives the actuator values at the end of the matrix cycle. During the first basic cycle, it does all the calculations for the sensor values. These calculations include the steer angle rates for the front $\dot{\delta}_f$ and aft $\dot{\delta}_a$ parts of the vehicle and the velocity of the actuators v_{act}, which are needed for the calculations of the wheel angle displacement θ of each wheel. The equations employed to make the calculations in the vehicle emulator can be found in appendix E.

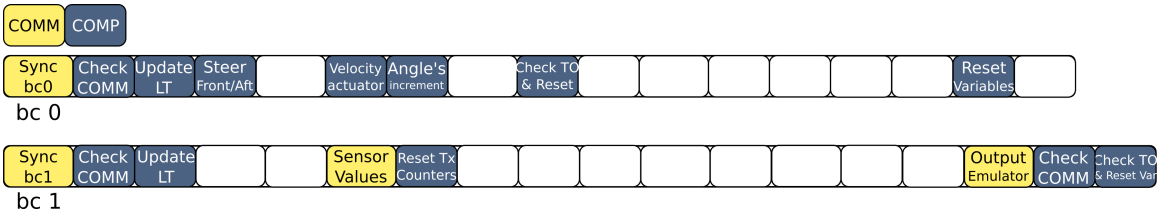


Figure 11: Vehicle emulator matrix cycle. Communication (COMM) tasks are painted in yellow and computational (COMP) tasks are painted in blue. White tasks are free windows to keep this matrix cycle synchronous with the controller's.

The matrix cycle for the vehicle emulator is presented in figure (11). The different operations to calculate the sensor values are performed over the first basic cycle. Empty time slots are left in between operations for every communication task happening in the controller. This way both matrix cycles remain synchronous, considering communication and computational tasks do not necessarily have the same duration. The sensor values are sent during the *Sensor Values* communication task. As it happened in the input generator matrix cycle, after sending the message a *Reset Tx Counters* task activates. The values for the actuators are received with the *Output Emulator* communication task. The last task in the matrix cycle assesses the time out of the reference message and resets the cycle variables before starting over.

4.6 System deployment and testing

When working with a simulation, there are factors that are sometimes disregarded that impact the performance of the system when taking it to the real world. This is why deploying the system into a hardware environment to study its behaviour is an important and challenging step. The challenge starts when the simulation falls apart under physical limitations and unexpected problems. In this project, there are already some issues forecasted at the design stage, such as the software clashing with the underlying FreeRTOS at high granularity frequencies or the minimum communication delay. The most unexpected limiting event found in this project can be found at the end of chapter 6.

This last section of Methodology shares a brief overview of how the system was deployed, presenting the main tools employed. It continues by defining the test case developed to showcase the findings of the project and finishes by listing the measurements and fault injections presented in chapter 6.

4.6.1 Deployment

From the beginning of the Minor Project where this thesis' code started to take shape, the development focused on fast prototyping. Deployment has been done taking small steps with new features and testing that the behaviour was (at least to some extent) as expected. The system was deployed step by step, from a local tick generator to a global time synchronized ensemble of boards. The controller behaviour was tested in a simulation before deploying it to the boards, to ensure that the calculations performed, the communication packaging to send through CAN and vehicle response was coherent with the original Simulink source. The new simulation had the main difference of being executed under a signal interrupt basis instead of using Simulink's software activation. The PID controller gains were copied from the original controller, as it had already been tested for the speed loop.

The code was developed using MATLAB-Simulink model-based design environment, along with HANcoder-1.0 extension. When the code is compiled a '.srec' file is generated, which can be flashed onto the STM32-E407 boards using the program Microboot. A detailed electric scheme with the connections between the different boards is presented in appendix B. It is possible to see the system's signal evolution while the boards are running using the HANTune_build_68 software. HANTune is really useful to check the state of the internal signals of the program but it has a maximum sample frequency of 100 Hz. This means that it is not able to record with enough precision everything that happens at the clocks frequency rate, as their ticks are in the order of the kHz. That is why a logic analyzer with a maximum sample frequency of 24 MHz is used to check any event that happens in the software at the level of task activation. The logic analyzer employed is from the company Az-delivery and counts with 8 digital channels to monitor signals concurrently in real-time. The software used to connect the developer laptop to the logic analyzer is Logic (2.3.37) from SALEAE, as it is free, recognizes the logic analyzer automatically and has the option to export the recordings to a '.csv' file. The digital records were processed using python scripts to get some of the results from chapter 6.

4.6.2 Test case definition

The test case for this project aims to demonstrate that the vehicle controller can make the vehicle emulator behave according to the values set by the reference generator. Also, it shows that the system is robust under certain failures and the operation continues in spite of them. The operational design domain, under which the test case will perform its activities, is the following:

1. The controller only controls the speed loop of the vehicle. The controller it was based on was only tested for its speed loop and the controller in this project does not add up any new functionalities.

2. If a controller board does not receive either a set message (from the input generator) or a sensor message (from the vehicle emulator), it will not perform the calculations for the output. It will raise an alarm and send an error log to the master controller board.

3. If the vehicle emulator does not receive a message with controller outputs it will remain in the same state as it was. In this case, the sensor value transmitted to the controller will remain as zero until a new message is received.

4. The vehicle emulator will not process the error log received by the controller.

The simplified operation defined for the test case is meant to focus the attention on the redundancy and synchronicity achieved on the controller side. No complex operations were performed outside the controller design as these were considered out of scope.

4.6.3 Measurements and fault injection

There are five main measurements that assess the synchronicity and correct temporal behaviour of the boards while performing operations.

1. The **granularity** that shows that the code is activated at a steady pace at the frequency set by the software interrupts.
2. The **communication delay** from the moment a message is sent until it is received. This measurement is needed to calculate the ensemble precision.
3. The **ensemble precision**, which shows what is the time difference between the slave boards and the time master board in the controller.
4. The **tasks execution time**, to get to know how much operational load is in the different tasks of the matrix cycle and define the WCET.
5. And finally, the **controller cycle duration**, which is extrapolated from the other measurements.

When the vehicle emulator got to respond to the set values from the input generator, some fault injection was performed. The tests are mainly focused on the time synchronized model.

1. The system can keep running even if one of the controller boards is shut down.
2. While the system is running, a controller board that is switched off, must be able to synchronize and engage with the system, resuming normal operation.
3. If a controller board has just stopped being synchronized with the rest of the system it must be able to recognize that and perform an auto-reset, achieving synchronicity again later.

The fault injection is provoked by switching on and off the boards in the ensemble with the wake-up button. When the wake-up button is pressed the board freezes until the wake-up button is pressed again. All the information about the messages lost by the controller boards during the second basic cycle is reflected in the error log that is output to the vehicle emulator along with the actuator values.

5 Overview of the software architecture

This master thesis shows the main choices in the design, deployment and testing of the prototype. This prototype serves as a demonstration to answer the main research question of the project. However, a big part of the development of the prototype was invested in the project's software. The software implements the core ideas from the MES-ES applying the knowledge about Real-Time distributed systems. The Time-Triggered schedule is hard-coded with the different tasks of the matrix cycles and the local time is generated in each board using a hardware clock.

This chapter makes a brief overview of the main features of the software, presenting how the role of the boards is chosen, how the time is managed and how the different subsystems are handled and activated. It also shows the relationship between the CAN interface and the tasks for a better understanding of the message exchange between the boards.

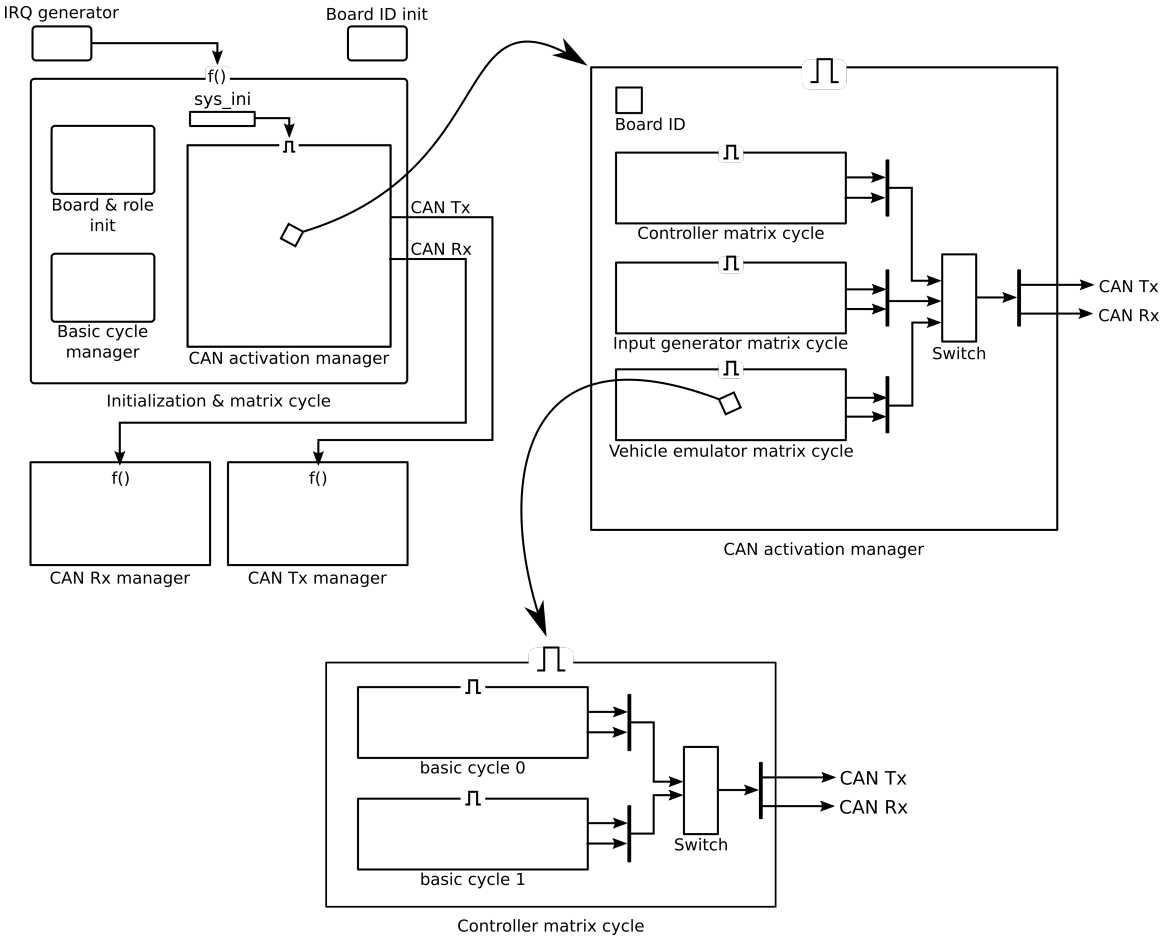


Figure 12: Overview of the software architecture.

The project's software is divided into different systems and subsystems. A main overview is pictured in figure (12). Looking at the highest overview of figure (12) (leftmost part), at the top-right part, the board ID initialization is presented. Each board performance is differentiated by this number, which is selected by powering a set of digital inputs following binary code. This is the only system presented that is not ruled by the IRQ generator from the top left corner. The IRQ generator is scheduled to generate an interrupt by the hardware clock of the board according to the granularity of the system. This interrupt activates the initialization and matrix cycle system via a function call. Every interruption executes the code inside this system once.

If the matrix cycle has not been initialized yet, the board and role initialization subsystems run first. Depending on if the board is part of the controller or not the initialization procedure is different. When the controller boards are turned on, they wait for a whole matrix cycle duration waiting for a message. If during this time a controller board has not received any, it continues operating as a master and sends the first reference time message, setting the board as initialized and starting the matrix cycle. The input generator and vehicle emulator always wait until they receive the first reference message from the master. If a board is reset during its matrix cycle, it starts this initialization procedure from the beginning. The main difference between an auto-reset from the boards' side in the matrix cycle and a hard-reset done by the user turning the board off and on again, is that with the auto-reset the board does not lose all the variables information from the matrix.

The basic cycle manager is in charge of updating the basic cycle counter every time the local ticks counter reaches the duration of one basic cycle. When the basic cycle is updated, the local time is reset, so the new cycle starts from the beginning. The CAN activation manager is in charge of activating the CAN interface subsystems for transmission (Tx) and reception (Rx) of messages. When looking inside (right part of figure (12)) the three matrix cycles can be visualized. Just the correct matrix cycle is activated depending on the Board ID decided previously. Each matrix cycle subsystem outputs booleans declaring if the CAN Tx or CAN Rx subsystems must be activated during this activation. Only the correct CAN signals are chosen to be output from the CAN activation manager, again depending on the Board ID value.

Each matrix cycle subsystem contains the information of the basic cycles of the boards (bottom picture in figure (12)). Inside the basic cycle subsystems, each task is declared in a similar fashion, using enabled subsystems. This kind of subsystem is only activated when a condition is set to true. When looking at the whole picture in figure (12), from left to right and then to bottom, the following three activations happen in succession. The CAN activation manager is only activated if the system has finished its initialization. The correct matrix cycle is activated depending on the Board ID value. The basic cycle activated inside the matrix cycle subsystem depends on the basic cycle counter value. Finally, the appropriate task of the basic cycle is activated depending on the local time counter, which at each activation will be equal to a value contained in a unique time window of the cycle. If the currently active task requires sending or receiving a message, the CAN Rx manager or the CAN Tx manager are activated with a function call. These two subsystems are responsible for handling the message buffer to either update it with new information from the CAN network or send the message buffer content to the network. Figure (12) shows only one CAN Rx and one CAN Tx manager, but the software has one of these systems per CAN channel.

One of the main ideas taken into account when designing the software of this project is hardware agnosticism. This means that the software can run on any STM32-E407 board and still perform any of the component tasks: controller, input generator or vehicle emulator. Furthermore, the controller boards can take either the slave or the master role while the program is running. There is only one version of the software and it is flashed on all the boards without fine-tuning to fit a specific role. The way a board is identified as any of the described components is by hardware connections. More in-depth information about hardware connections can be found in appendix B but there are several input pins, from *D2* to *D7*, that are chosen to designate a board ID. Each board has a unique board ID with which the software is able to give it its precise instructions. Another important idea from the software perspective is interfacing. There is a clear distinction between the program systems that are in charge of the board tasks, regardless of its role, and the program systems that manage communication between boards. Every time a message is sent or received the software must use the CAN communication interface to perform the message exchange.

The characteristic that stands out the most when comparing the software of this thesis with a more conventional MATLAB-Simulink program is how the simulation time is handled. MATLAB-Simulink lets the FreeRTOS handle the program execution time by default. The program sample time is defined in its configuration and the software blocks run at that rate. If the sampling frequency is set at 10 kHz (normally the fastest sample frequency in Simulink), the program runs its loops at this frequency. However, for this thesis it has been decided to set the sampling frequency by hardware interrupts with an internal clock of the boards. By relying on an external source for software granularity, the sampling frequency is more reliable as it is not depending on a computer calculation that could either be delayed or accelerated because of processor overhead. The software subsystem in charge of creating the software activation signals for the other subsystems is the IRQ generator. This generator sets a stable rhythm for the other parts of the software, creating the notion of the local time of the board. It is also possible to change this sampling frequency to be even faster than Simulink's limit of 10 kHz.

6 Results

The results are divided into the time analysis of the prototype behaviour and the test case results. The first section focuses on the time aspects of the clock activation, the boards' synchronization, the message exchange and the task periods. The second section presents the vehicle emulator response and the boards' behaviour with fault injection. Unless specified otherwise, the parameters used for the measurements are the following: granularity $gr = 0.1$ ms, communication period $\tau = 40$ ticks, baud rate 1000 kbit/s, CAN1 phase $\phi_1 = 5$ ticks and CAN2 phase $\phi_2 = 8$ ticks.

6.1 Time analysis

In order to be aware of the potential of the prototype and emphasize the points where further research is required, it is important to assess the time limits and performance of the boards during operation. This time analysis section presents the measurements performed over the granularity, ensemble precision, communication delay, computational task time and controller cycle duration. Here the results are merely presented and further analysis is held later in chapter 7.

6.1.1 System's granularity

Every time the hardware clock in each board activates, it generates a function call to activate the code. A digital output has been toggled every time this has happened, so the difference between two consecutive toggles results in the system granularity. A recording of the granularity over a whole matrix cycle is presented in figure (13). The average granularity recorded is $\bar{gr} = 0.10001$ ms, the maximum $gr_{max} = 0.13271$ ms and the minimum $gr_{min} = 0.06983$ ms.

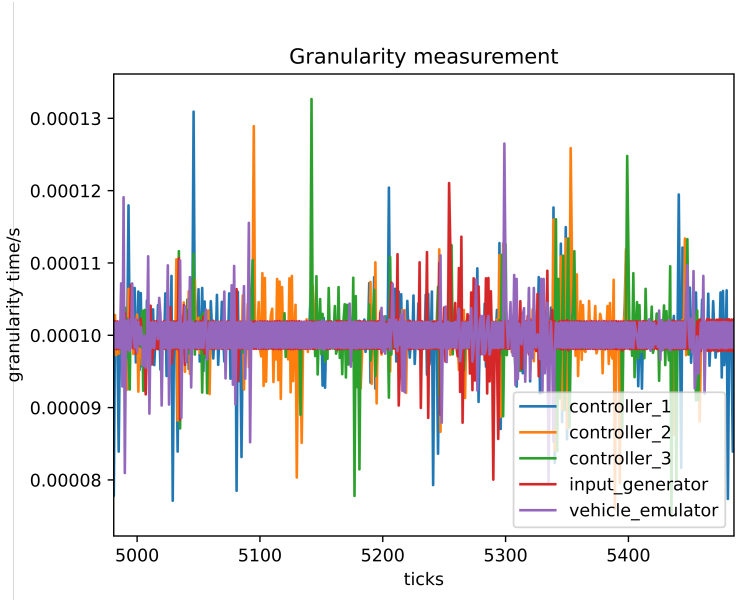


Figure 13: Granularity measurement over a matrix cycle for 0.1 ms granularity.

The granularity employed for the test case is 0.1 ms, the same as the minimum software activation that can be achieved with Simulink. However, this project's software has the option to run faster. Unfortunately, running at ≤ 0.07 ms triggers a clock activation error, as can be observed in figure (14). Instead of always activating every 70 µs, there are periods of time in which the clock activates every ~ 65.6 ms instead.

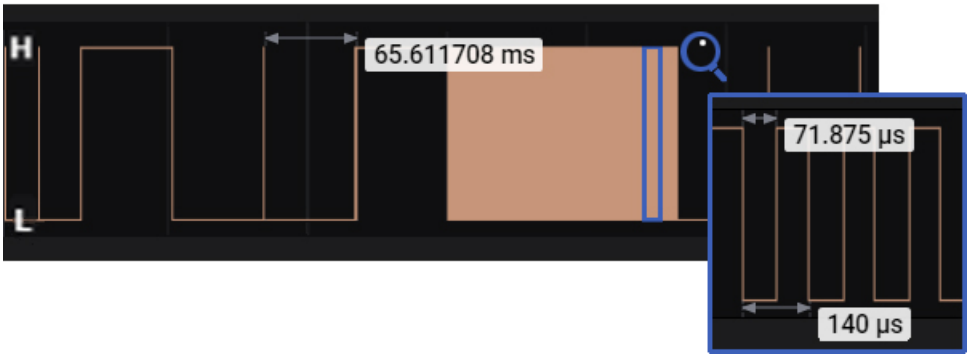


Figure 14: Measurement portion of the granularity measurement with 0.07 ms granularity with the logic analyzer. The measurement shows the highs (H) and lows (L) of the clock activation. A small portion has been magnified to clarify the frequency of the signals in the solid section.

6.1.2 Communication delay

When a message is transmitted, some time passes from the moment the message is sent from one board until it is received by the other. To measure it in an automatized fashion two digital outputs were used: one is toggled when the message is sent at the sender and the other when the message is received at the receiver board. This allows measuring the time difference between one event and the next with the logic analyzer. The result found is that the communication delay between boards is around 0.3 ± 0.1 ms for both CAN1 and CAN2 channels. This measurement was done using only one message during the transmission in each communication task.

6.1.3 Ensemble precision

To check how much every board is out of phase with respect to the global time of the master board the difference in ticks between the moment the synchronization message is expected to be received and when it is actually received is calculated, from now on called desynchronization. This relies on the knowledge of two entities, the time schedule of the board transmitting the message and the communication delay. This last temporal entity is estimated through observation and it is not a constant value. Also, this desynchronization is used to correct the local time of a board with respect to the master. It is therefore important to minimize the influence of a too-long unexpected communication delay, as it could make the board correct its local time to a wrong global time. This is why the desynchronization value has been limited to ± 15 ticks. The average, maximum and minimum values of the desynchronization ticks of the different boards are presented in table (1).

	$Controller_2$	$Controller_3$	Vehicle Em.	Input Gen.
AVG	1	2	12	14
MAX	15	15	15	15
MIN	-15	-15	-14	-15

Table 1: Mean desynchronization ticks from synchronization messages from the master ($Controller_1$) and each other board. The values presented have been extracted from a fifteen minutes long recording with HANTune.

6.1.4 Computational tasks time

The clock activates periodically after the time defined by the granularity of the system, so the whole software is run once again. Before this happens, the previous software run should have already finished. Measuring the time elapsed from the moment the code is activated until it finishes all its computations allows to assess how much spare time is there for extra calculations. Figure (15) presents the execution time at each tick during a matrix cycle of every board. The maximum execution time recorded is $et = 0.068\,\text{ms}$. Adding a 20% safety margin the estimated WCET is $0.082\,\text{ms}$.

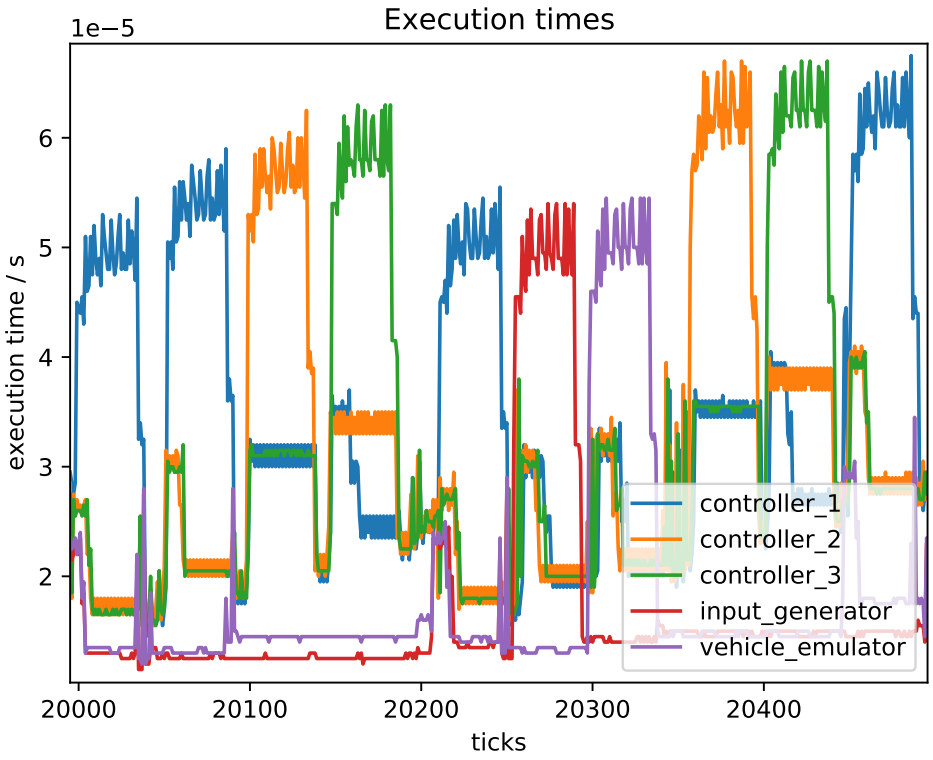


Figure 15: Execution time of the different boards over one matrix cycle.

6.1.5 Controller cycle duration

Every computational task in the code has been set for 4 ticks and every communication task for 40 ticks, so according to the matrix cycle presented in figure (9) the whole matrix cycle duration is 500 ticks. Every tick, as presented in section 6.1.1, lasts 0.1 ms, making the whole matrix cycle duration 50 ms. This is the controller cycle duration proposed by the company supervisor of this project Aart-Jan de Graaf to control the vehicle emulator.

6.2 Test case results

The main research question of this thesis is answered in this section when presenting the vehicle response of the vehicle emulator when changing the set value at the input generator. Afterwards, the behaviour of the system with the fault injection is explained and one of the major limitations of the prototype is presented: the missing messages.

6.2.1 Vehicle response

The main research question of this report asks if it is possible to deploy a controller in the STM32-E407 boards with Real-Time communication using HANcoder. As presented in figure (16), it is possible. The set values from the input generator are received and processed by the controller. After triple modular redundancy is assessed, the controller sends the output result to the vehicle emulator, which generates the estimated speed v_est_s from figure (16). Because of the vehicle emulator's physical limitations, there is a speed limit of 10 m/s and an acceleration limit of 2 m/s^2.

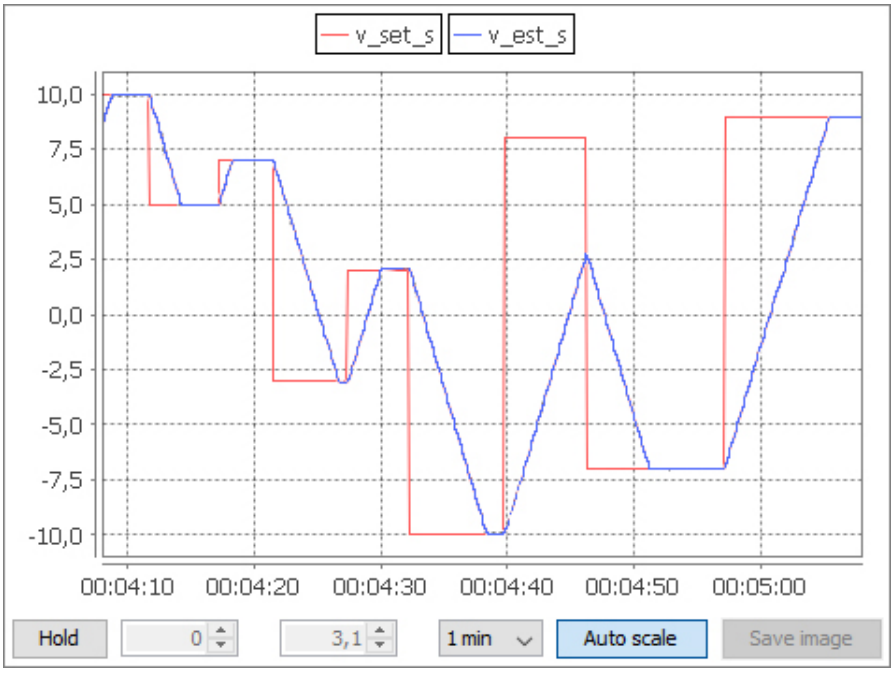


Figure 16: Superposition of the vehicle response (v_est_s in blue) of the vehicle emulator and the velocity set (v_set_s in red) of the input generator. The y-axis shows speed in m/s and the x-axis spans a duration of one minute.

6.2.2 Fault injection

As previously presented in the test case operation design description, there are different scenarios in which the prototype must behave as planned to show that the system is robust against communication and board errors. The prototype behaves as expected with the following remarks.

- When one of the controller boards is shut down, the other two boards change their roles if necessary to make sure there is one master. If there are no other failures, the two remaining boards will continue to operate and agree in the speed calculations, so the TMR outputs a coherent torque value for the wheels.
- The error log also shows which board failed. If two boards are shut down, the output from the controller remains at $\tau = 0\,\text{Nm}$ and the error log shows the disagreement between the boards and the missing messages.
- The vehicle emulator remains at the same state while the torque is zero.
- Resuming the operation of a board by just clicking the wake-up button or by reinitializing it again with a hard reset, allows it to synchronize with the master (if there is any) or to take that role.

6.2.3 Missing messages

One of the major problems currently limiting the software potential is the missing messages error. While the boards are operating, even though the vehicle emulator works fine and the vehicle response remains stable regardless of this problem, it is possible to see, after enough time waiting, that some messages are sent but not received. Closer inspection with the logic analyzer shows how sometimes messages are sent by a board and, while the receiver board is synchronized, no messages are acknowledged. This problem gets worse if instead of sending several messages only one message is sent every communication task. A fifteen minutes recording example of missed messages during operation is presented in table (2).

	Sync_0	Vote_1	Vote_2	Vote_3	Sync_1	Set	Sensor	Out_1	Out_2	$\text{Out}_{\text{Contr.}}$
Controller_1	x	x	0	0	x	194	111	0	0	x
Controller_2	0	0	x	0	0	174	140	x	x	x
Controller_3	0	0	1	x	0	154	130	x	x	x
Input Gen.	4	x	x	x	3	x	x	x	x	x
Vehicle Em.	39	x	x	x	38	x	x	x	x	44

Table 2: Number of messages not received on each board during fifteen minutes data recording. The 'x' means that the message is not meant to be received by that board. During the recording duration Controller_1 remained as the master of the ensemble.

The occurrence of this problem seems random and unstable. It has been observed that by reducing the communication tasks period (currently 4 ms) the missing messages increase. The values shown in table (2) might not be accurate enough of the occurrence frequency of this error but shows in general terms how some roles are more sensitive than others to miss certain messages. More in-depth results of this problem are presented in appendix G.

7 Discussion

The presented results lead to a series of different questions or missing relationships that are either answered or assessed in this section. These topics are divided into three groups:

1. Why is there a limit in granularity and what is causing it? What is causing the maximum execution times? Are granularity and execution time correlated?

2. Is the ensemble precision related to the communication delay? Why do input generator and vehicle emulator have bigger desynchronization values?

3. What is causing the missing messages? Why are input generator and vehicle emulator values far more impacted by this problem? Why is this limiting the software? What other limitations have been found?

After assessing the limitations of the prototype, the research questions of the project are compiled and discussed, finally leading into the conclusion and recommendations in chapter 8, where some advice is presented for further investigation and improvement of this project's application.

7.1 Granularity and execution time

When trying to reduce the granularity of the system to increase the cycle speed a limiting granularity time of $gr = 0.07\,\text{ms}$ was found. This time is close to the maximum execution time required by some of the tasks in the matrix cycle, as seen in figure (15). When the code requires more time to finish its execution than the time left for the next activation, the clock lag activation error from figure (14) triggers.

Figure (15) shows the execution time required by every tick in a matrix cycle, so it is possible to compare this to the matrix cycle in figure (9) and find a dependency between tasks and execution time. Every high execution time section from this figure corresponds to a communication task in the cycle. The order of communication tasks is: synchronization in basic cycle 0 (controller_1), vote 1 (controller_1), vote 2 (controller_2), vote 3 (controller_3), synchronization in basic cycle 1 (controller_1), set values (input generator), sensor values (vehicle emulator), output controller 1 (controller_2), output controller 2 (controller_3) and controller output (controller_1). After close inspection of this figure in order, this consecutive order of tasks is apparent.

When comparing the granularity from figure (13) and the execution time from figure (15), it is possible to see some similarities that lead to think that they might be correlated. They have been merged together in figure (17) for better visualization. With close inspection of figure (17) it is possible to see that for every peak in the granularity section (top) there seems to be a correspondence to a high execution time section (bottom). Red dashed lines help visualize the alignment between granularity and execution time. Also, by the end of the communication task, there are some low granularity peaks on every board. In the execution time part, between the sensor values task (purple) at 4800 ticks and the output control 1 task (orange) at 4850 ticks, there is also a spiky section where the controller boards make the calculations for the output to the vehicle emulator. These spikes are also present in the granularity section.

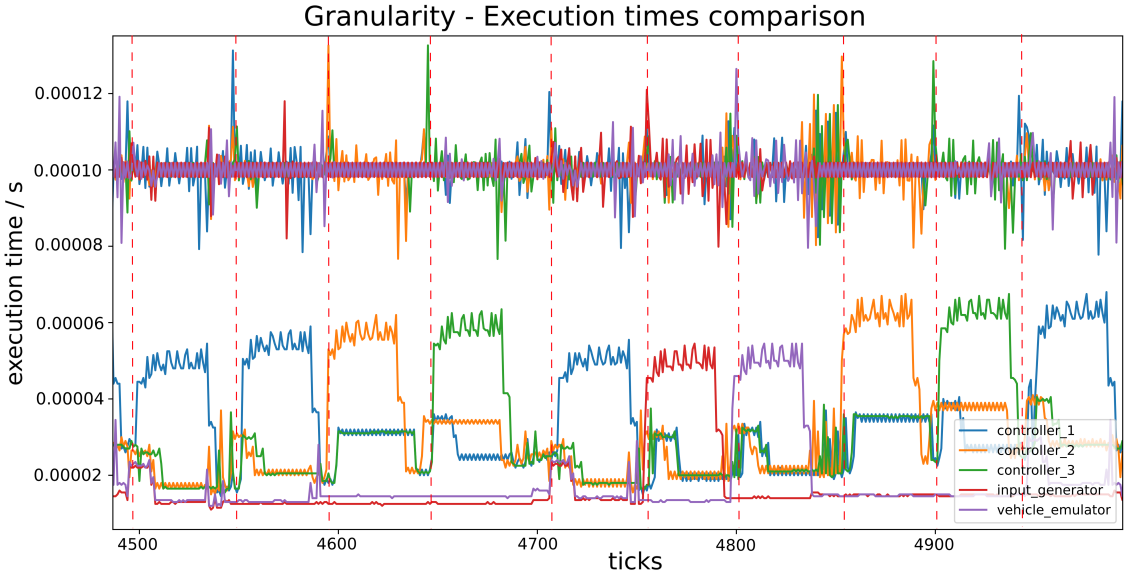


Figure 17: Superposition of granularity measurement (top) and execution time measurement (bottom). The vertical dashed red lines help visualize the apparent correlation between the two.

All these details matching together guide us closer to the idea that the clock activation is related to the time each task requires to finish its execution. This is inherently wrong, as choosing the hardware clock was mainly because it is important to be able to have an external source of ticking that will ensure that the deadlines are met regardless of computational load. Also, if the computational load gets close to the tick activation time, the code stops working because of the clock lag activation error shown in figure (14).

However, when looking at the whole picture, it does not seem like such a big problem, because the clock activation seems to be 'recovering' some of the lost time in some parts, so the average cycle duration remains at 50 ms. Moreover, keeping every software activation at a low execution time value ensures that the granularity does not deviate much from its average expected value. The code also allows the user to choose to subdivide a task further inside the computational time window. If it is possible to reduce the execution time at the transmission of messages, the highest granularity peaks would disappear, allowing for choosing a lower granularity value without triggering the clock lag activation error. It might be also wise to look for some way of activating the code and the clock so they are decoupled, making the core code more robust and true to the idea of reaching deadlines regardless of computational load.

7.2 Communication delay and ensemble precision

There are two ways with which two boards can stop being in sync while continuing their operation. One possibility is that the ticking frequency of both boards could be different enough to desynchronize before the next synchronization message. The other is that a sync message could have an unexpected longer delay provoking the receiver to correct its local time to a 'wrong' global time based on the temporal information it received. The granularity analysis points towards discarding the first option, as all the boards present the same average granularity over a longer time than a basic cycle. The second option seems to, at least, be in accordance with the values presented in table (1) for the controller. The average controller slave desynchronization with the master has been found to be 1 or 2 ticks, matching the communication delay found in the communication delay analysis. This is not true for the vehicle emulator and the input generator though.

Further research is required to find an explanation on why the desynchronization values for the vehicle emulator and input generator are higher than for the controller boards. The current hypothesis is that, as the second option suggests, there happens to be a higher amount of longer than expected communication delays in the message exchange with these boards. This does not match the measurements found for the communication delay analysis, but this analysis was done for only one message for every transmission. This leaves the question open: when transmitting several messages per communication task, does the average communication delay between controller and both vehicle emulator and input generator increase? Or is there some other phenomenon provoking the higher desynchronization values? Is this related to the missing messages?

7.3 Missing messages and limitations

At the beginning of the project, when messages were lost, it was assumed that the reason was that the boards participating in the message exchange were out of sync. However, even though this could still be the reason that explains some of the messages lost presented in table (2), it has been observed that some messages transmitted are just not received, even when the boards are synchronized.

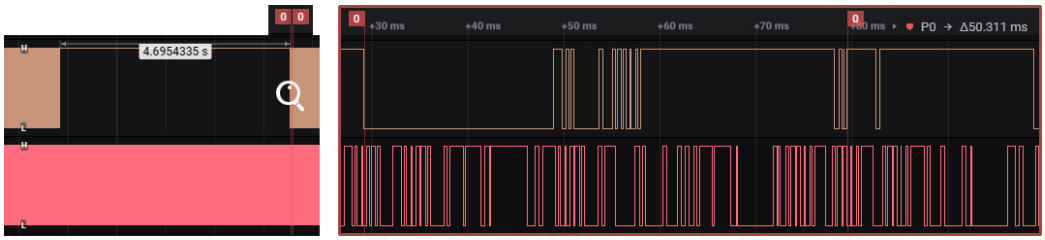


Figure 18: Measurement portion of the activation tasks of the input generator (brown) and the master board (red) with the logic analyzer. The magnified portion (right) shows the activating tasks during a matrix cycle right after the input generator resynchronizes. In the not magnified portion (left) it is shown that the input generator remained reset during more than 4.6 seconds.

If the reference message from the first basic cycle is lost at the input generator or the vehicle generator, these boards reset and wait until a new synchronization message is received. If the message lost error persists for some consecutive basic cycles, as happened in the example from figure (18), the board keeps waiting. This means that more messages are missed in the controller end, as the input generator or vehicle emulator are 'just listening' waiting for a sync message to arrive instead of sending the messages that they are supposed to send.

Interestingly, as table (2) shows, the input generator seems to suffer less from the missing message problem as a receiver than the vehicle emulator, which probably reflects that the problem is not just unique but a combination of several sources that require further research. Possible causes of this problem could range from software problems (code errors that require refinement) to hardware problems. The connections of the boards, even being soldered, have presented some communication troubles and instabilities, probably because the prototype had to go through different trips, making some of the cables get loose and requiring soldering again. Also, due to an error in the initial design of the prototype, the cables used are not twisted pairs as the CAN hardware protocol states. Some electromagnetic interference could also be impairing the message transmission. Moreover, because the CAN channels are not independent of each other, when one of them fails the other does so too. Low-level inspection of the message exchange is required, at both the hardware side, checking where the message is lost, and in the software, inspecting if the message is actually being received but not processed because of desynchronization or other issues. More insight of different parameters influencing the missed messages frequency is presented in appendix G.

The main limitation in the current version of the prototype is the cycle time. If the boards were able to perform controller operations in less time, more sophisticated operations could be built upon them and more communication tasks could be introduced in the matrix to increase the information flux between them. More frequent communication exchange is important when a higher amount and more precise data is held in the operations that have to undergo the triple modular redundancy. The cycle period is limited by different factors:

1. **Missing messages** are currently degrading the performance of the prototype for the test case proposed, as repeatedly missing messages at the vehicle emulator makes the system unresponsive. Moreover, it is not acceptable to have missing messages for no reason on a usual basis. It also has been observed that by reducing the communication task period the number of missing messages increases. Trying to reduce the current matrix cycle duration by reducing the communication tasks period would increase the missing message occurrence which, in an extreme case, could even put the synchronization of the ensemble at risk.

2. Another way of reducing the matrix cycle duration is decreasing their **granularity**. With the current prototype, it is possible to safely increase the board's speed by 20% (reduce the 0.1 ms ticks to 0.08 ms), but it would be desirable to increase the speed even further, as the software limit is far beyond. Due to initial tests performed during the Minor Project, it is suspected that it is possible to reduce granularity below 0.05 ms without having the FreeRTOS clashing influence in the software.

3. Requiring the **role election** during basic cycle 0 is another important drawback. A whole basic cycle (currently being almost half of the whole matrix cycle) is required for the controller boards to acknowledge the time master board during that cycle. It would be important to either change the way CAN communication is managed with the CAN blocks in HANcoder so it is possible to follow the TTCAN protocol, as suggested before in section (4.3.2), or think of a more efficient way of handling the role allocation in the controller.

Unravelling the source of the missing messages, finding a way of reducing the computational load during transmission tasks and improving the management of the role allocation at the controller could allow the prototype to reach a higher level of performance. If the cycle period could be reduced to its physical limits (minimum communication delay in hardware and FreeRTOS frequency limitation in software) the platform developed in this project could be employed as a basis for future research and more complex Real-Time controllers.

7.4 Research questions discussion

Finally, the research questions have to be answered to assess to what extent the research of this project was successful. Most of them have already been answered throughout the text, either when looking into literature, asking the supervisors or looking at the results. In the following paragraphs, a compilation of them is presented for clarity, including additional reflection and discussion.

- *Is it possible to deploy the controller in the STM32-E407 boards with Real-Time communication using HANcoder?*

As presented in figure (16), the main objective of the research has been fulfilled. This project major success is creating a workbench for future students to learn with and taking a first step towards a useful tool in the industry.

- *What is the minimum granularity of the software?*

As it has been shown in the execution time results in section 6.1.4, the software activation has a time resolution in the order of hundreds of microseconds. If the software load could be reduced during the transmission tasks the time resolution could probably be reduced too. Taking care of the clock lag activation error presented in figure (14) and minimizing the execution time of the tasks in every tick would allow for reducing the granularity of the system close to the limit of the FreeRTOS frequency.

- *What is the communication delay?*

Measurements presented in section 6.1.2 show a communication delay of 0.3 ± 0.1 ms when the baud rate is 1000 kbit/s. However, the results presented in 6.1.3 point towards the possibility that, at least at the input generator and vehicle emulator, the communication delay is bigger when increasing the number of copies of a message transmitted during the same communication task. Further research is required in this topic, exploring the message exchange in the CAN network with more detail and taking into account that the communication delay could also be sensible to different parameters, as it happens to the missing message problem studied in appendix G.

- *How many nodes does the system need in order to be fully redundant?*

The controller part of the system, which was the main focus of the project as presented in the scope of section 1.5, counts with three controller boards. The triple modular redundancy, along with double channel communication and multicast copies of a message in every communication task make the controller redundant. Three nodes were required in the controller to perform the triple modular redundancy.

- *What is an appropriate frequency for the input signal to the controller?*

As proposed by Aart-Jan de Graaf, the company supervisor of this project, a cycle time of 50 ms (20 Hz) is enough to control the vehicle emulator. This cycle time was proposed to control both steering and speed control loops. Regarding the company supervisor's recommendation, only 100 ms (10 Hz) were required for the speed control loop. Because the original controller that was adapted into the Real-Time software has not been tested for the steering loop, the vehicle controller of the project is only controlled in the speed loop. It is left for further research to improve the functionality of the prototype with the steering loop too, but the code already contains some of the elements required to make it work, such as the cycle time and the controller calculations.

- *Is there any overhead in the controller because of the signal processing that may delay the Time-Triggered communication?*

The Time-Triggered schedule strategy followed allows dividing the computational load into different tasks. As long as an atomic operation does not require too much computational load, it is possible to subdivide the operations accordingly to ensure that every software activation time does not surpass a granularity period. As shown in figure (15), communication tasks require more computation work than other tasks. However, as presented in figure (17), when choosing a granularity of 0.1 ms the execution time remains always below the tick activation threshold.

- *Is the time response of the emulator reasonable with the deployed controller?*

The vehicle emulator receives the information from the controller accordingly to the design. The response depends on both the vehicle emulation physical limitations and the controller properties. As presented in figure (16) the vehicle emulator responds to the input generator as expected. The main limitation of the prototype regarding vehicle response is the missing messages problem. As long as the vehicle emulator misses messages from the controller the vehicle remains unresponsive. The moment the vehicle emulator receives messages again, the prototype is prepared to recover from the faults and the system reengages in communication, allowing control over the vehicle again.

On the one hand, the communication problems found in the prototype limit its performance and make the system unsafe. On the other hand, the missing messages create a scenario in which the boards are able to recover from unexpected and random faults. They are successfully proving their redundancy by changing the master role when required or acknowledging one of the copies of a message sent from both CAN channels, even if that was not the first copy in the transmission. Of course, if more than one fault occurs during the same task, the system is not able to cope with it, sometimes resulting in the vehicle being unresponsive. Further research is required to identify the missing messages root problem (or problems), but at this stage it is possible to confirm that the main mechanisms that were designed for this prototype are working in place.

8 Conclusion and recommendations

The project's main objective has been fulfilled by showing that it is possible to deploy a distributed controller in the STM32-E407 boards with Real-Time communication using HANcoder. The deployed controller is made of three boards to perform triple modular redundancy with the calculations of the output to the actuators. The prototype counts with two more boards, one for the input generator and another one with the vehicle emulator. As proposed by the company supervisor Aart-Jan de Graaf, a controller cycle of 50 ms is enough to control the speed loop of the vehicle emulator, obtaining a reasonable response. The concept is fully-redundant, as there is no fragile point in the design where one single failure provokes the system to stop operation.

This achievement is a milestone towards Real-Time distributed systems projects using model-based design, which helps engineers with different specializations to understand each other's work. Further research is required in this field in order to popularize distributed systems development in groups and implement modern agile methods in the workflow. The current prototype suffers from communication problems but the main temporal mechanisms and redundancy features have shown a robust performance under a limited amount of failures at the same time.

The following recommendations aim to put the focus on the current most important topics that prevent the system from performing at its full capacity:

1. Renew the connections between the boards with twisted pair cables as stated in the CAN hardware protocol. Check the CAN communication between the boards with the new connections and ensure that they are safe and stable.

2. Decouple the CAN channels in the HANcoder software to make them independent of each other. The current HANcoder blocks do not offer independent channels for redundant communication.

3. If the problem of the missing messages persists, investigate its source by analyzing the message path, from the transmitter to the receiver. Identify at which point the message is lost, as the cause could be the hardware, the software or a combination of both. Recording the variable's information within the program for later processing could help with system diagnosis.

4. Minimize the computational load during the message transmission. This will allow reducing the granularity of the system.

5. Simplify the master-slave election policy, as it currently requires a whole basic cycle. It can be shortened by changing the way the CAN blocks of HANcoder handle the messages so they fit the specification from the TTCAN better.

The main objective of assessing these issues is to reduce the time required by the controller boards to perform the calculations during the controller cycle. All these time shortening proposals would save time to do other essential activities needed in more complex controller projects.

This project has shown that it is possible to build a Real-Time controller using HANcoder. The current prototype can already be used as a template for a fully-redundant distributed embedded Real-Time application but more research is advised to make it reach its utmost potential and be a useful tool to solve technological problems in the industry.

Bibliography

Baek, W., Jang, S., Song, H., Kim, S., Song, B. & Chwa, D., (2008), A CAN-based Distributed Control System for Autonomous All-Terrain Vehicle (ATV) [17th IFAC World Congress], *IFAC Proceedings Volumes*, *41*(2), 9505–9510, https://doi.org/https://doi.org/10.3182/20080706-5-KR-1001.01607

Callen, J.N., (1998), Distributed control for unmanned vehicles, 5.

CAN_CiA, (2022), Designing a CAN network [Accessed: 30-01-2022], Available at: https://www.can-cia.org/can-knowledge/can/design-can-network/

Craciunas, S.S., Oliver, R.S. & Ecker, V., (2014), Optimal static scheduling of real-time tasks on distributed time-triggered networked systems, 9.

de Graaf, A.-J., (2017), Two axle vehicle, 22.

de Graaf, A.-J., Dutta, S. & Manani, A., (2020), CAN based distributed controller.

Freier, M. & Chen, J.-J., (2014), Time Triggered Scheduling Analysis for Real-Time Applications on Multicore Platforms, 6.

Gérard, S., Espinoza, H., Terrier, F. & Selic, B., (2010), *6 Modeling Languages for Real-Time and Embedded Systems* (H. Giese, G. Karsai, E. Lee, B. Rumpe & B. Schätz, Eds.), Springer Berlin Heidelberg, https://doi.org/10.1007/978-3-642-16277-0_6

Göhring, D., (2012), Controller Architecture for the Autonomous Cars: MadeInGermany and e-Instein, 23.

Kopetz, H., (2011a), *Section 6.4.2 - Fault-Tolerant Unit* (Second), Springer, https://doi.org/https://doi.org/10.1007/978-1-4419-8237-7_6

Kopetz, H., (2011b), *Section 7.5 - Time-Triggered Communication* (Second), Springer, https://doi.org/https://doi.org/10.1007/978-1-4419-8237-7_7

Leen, G. & Heffernan, D., (2001), TTCAN: A new time-triggered controller area network, 18.

Lundin & Rai, S.b., (2019), How to calculate bus load of CAN bus? [Accessed: 30-01-2022], Available at: https://electronics.stackexchange.com/questions/422998/how-to-calculate-bus-load-of-can-bus

Mathur, R., Saraswat, R. & Mathur, G., (2014), An Analytical Study of Communication Protocols Used in Automotive Industry, 7.

MathWorks, (2021), Embedded Coder Supported Hardware [Accessed: 10-08-21], Available at: https://nl.mathworks.com/help/ecoder/supported-hardware.html

Palaniswamy, S., Vinod, B., Devi, R. & Rajkumar, E., (2015), Real-Time Task Scheduling for Distrib-

uted Embedded System using MATLAB Toolboxes, *Indian Journal of Science and Technology*, *7*, https://doi.org/10.17485/ijst/2015/v8i15/55680

Quaranta, G. & Mantegazza, P., (2002a), Using Matlab-Simulink RTW To Build Real Time Control Applications In User Space With RTAI-LXRT.

Quaranta, G. & Mantegazza, P., (2002b), Using Matlab-Simulink RTW To Build Real Time Control Applications In User Space With RTAI-LXRT.

Rapita Systems, (2021), Discover Worst-Case Execution Time [Accessed: 06-11-2021], Available at: https://www.rapitasystems.com/worst-case-execution-time

Samak, C., Samak, T. & Kandhasamy, S., (2020), Control Strategies for Autonomous Vehicles, 49.

Shahian Jahromi, B., Hussain, S., Karakas, B. & Cetin, S., (2017), Control of autonomous ground vehicles: a brief technical review, *IOP Conference Series: Materials Science and Engineering*, *224*, 012029, https://doi.org/10.1088/1757-899X/224/1/012029

Sivakumar, P., Vinod, B., Devi, R. & Rajkumar, E.R., (2015), Real-Time Task Scheduling for Distributed Embedded System using MATLAB Toolboxes, *Indian journal of science and technology*, *8*.

Vora, V., Y.N.Makwana, Kothari, A. & Suthar, A., (2009), Implementation of Real Time Scheduling in MATLAB.

A Vote criteria

The vote decision computational task in the controller matrix cycle follows the criteria from figure (19).

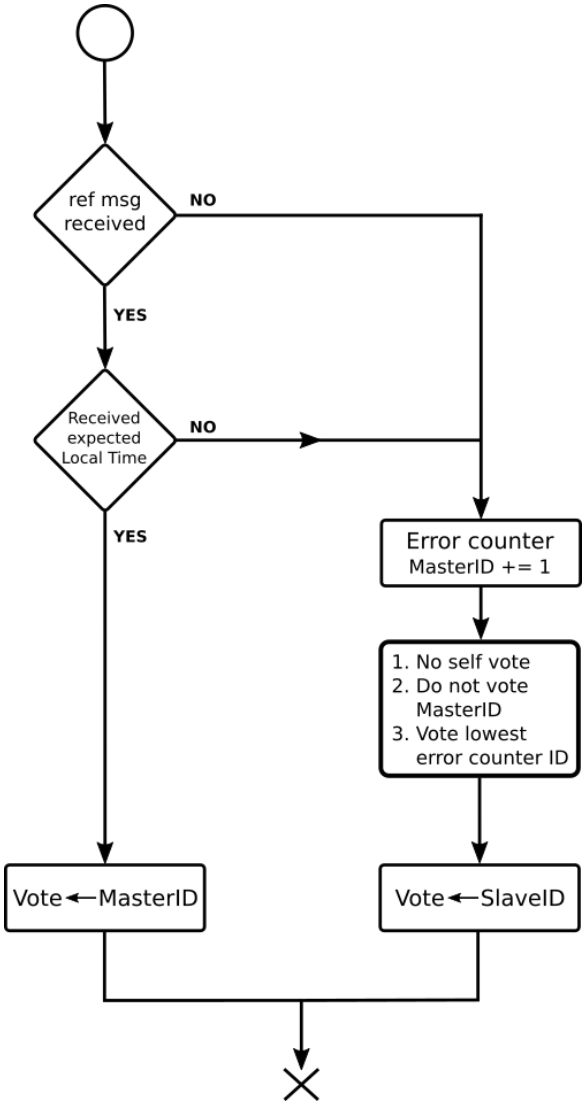


Figure 19: Flowchart of the voting decision logic when casting a vote for the time master. If the reference message received from the master corresponds with the expected Local Time, the vote is cast for the same master board. If any of the other conditions were not set the error counter for the board being the master is increased by one and another board is voted to be the new master. There are three conditions when selecting the vote for a slave board: (1) a board cannot vote itself, (2) a board cannot vote a master that has failed and (3) the vote should be for the slave board with the lowest error counter registered.

B Hardware connections

The main connections among the boards are those setting the CAN communication channel. Each board is connected to two CAN transceivers (one per CAN communication channels) using the 3 V and ground pins along with pins 3 and 4 for reception and transmission respectively for CAN communication channel 1 and pins D11 for reception and D1 for transmission for CAN communication channel 2.

Apart from CAN communication, there are also connections regarding the boards' software functionality. Applying 5 V voltage to pins D2 to D7, combining them using the binary base, it is possible to assign different ID numbers to each board. An electric scheme graphically showing all the connections is pictured in figure (20).

Monitoring and testing are done with HANTune. It is possible to connect each board with the developer's laptop using a USB connection. This way the board gets power and HANTune can establish a connection to access all the Simulink signal values. It also allows changing parameters during running time. However, connecting with HANTune requires extra processing by FreeRTOS to manage the communication, so connecting when the hardware interrupts are too frequent may make the system crash. Moreover, HANTune has a limited sample time of 100 Hz, so it is not possible to monitor properly the signals with a high rate of change. That is why HANTune is used to monitor the most steady signals in the system and change the input generator parameters.

The system hardware interrupts run faster than HANTune sample time, so to check that the tasks of the matrix cycle are being activated in the appropriate moment a logic analyzer is used. It can be connected to output pins A0 to A5 and D8 to D13 of each board to up to eight channels and it runs with a sample time of up to 24 MHz. When an interesting event in the system happens the pin is toggled, so a digital signal is read by the logic analyzer. This way it is possible to verify the time consistency of the ensemble.

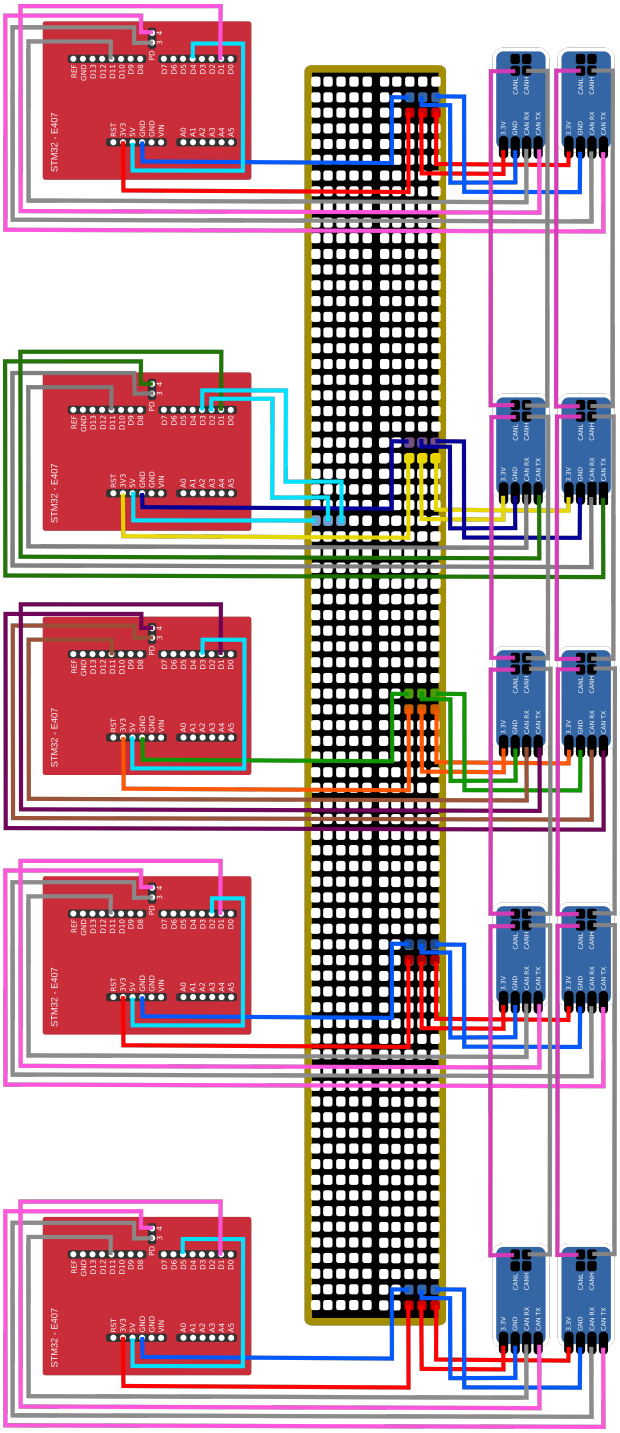


Figure 20: Basic electric scheme of the system with CAN communication and ID connections.

C Controller operations

The controller starts estimating the steer front and aft angles from the wheel angles information provided by the sensors.

$$\begin{aligned} \delta_{f\,est} &= \int \frac{(\dot{\theta}_{fr\,act} R_{fr} - \dot{\theta}_{fl\,act} R_{fl})}{T_{lf}} \mathrm{d}t \\ \delta_{a\,est} &= \int \frac{(\dot{\theta}_{ar\,act} R_{ar} - \dot{\theta}_{al\,act} R_{al})}{T_{la}} \mathrm{d}t \end{aligned} \tag{1}$$

where $\delta_{f\,est}$ and $\delta_{a\,est}$ are the steering front and aft angles, respectively. Every $\dot{\theta}_{act}$ corresponds to the angle rate of one of the wheels and the R variables are the radii from each wheel to its axis. T_{lf} and T_{la} are the torques at the front and aft of the vehicle.

With the estimation of the steering angles it is already possible to calculate the torque at each wheel with the first PID controller.

$$\begin{aligned} \tau_{fr\,out} &= k_p(\delta_{f\,set} - \delta_{f\,est}) + k_i \int (\delta_{f\,set} - \delta_{f\,est})\mathrm{d}t + k_d \frac{\mathrm{d}(\delta_{f\,set} - \delta_{f\,est})}{\mathrm{d}t} \\ \tau_{fl\,out} &= -\tau_{fr\,out} \\ \tau_{ar\,out} &= k_p(\delta_{a\,set} - \delta_{a\,est}) + k_i \int (\delta_{a\,set} - \delta_{a\,est})\mathrm{d}t + k_d \frac{\mathrm{d}(\delta_{a\,set} - \delta_{a\,est})}{\mathrm{d}t} \\ \tau_{al\,out} &= -\tau_{ar\,out} \end{aligned} \tag{2}$$

where each τ_{out} variable is the torque at each corresponding wheel, k_p, k_i and k_d are the proportional, integral and derivative gains of each controller and δ_{set} is the steering angle set by the reference generator, either at the front or at the aft.

The inputs to the vehicle controller also allow it to estimate the speed of the vehicle v_{est}.

$$v_{est} = \frac{\dot{\theta}_{fr\,act} R_{fr} + \dot{\theta}_{fl\,act} R_{fl} + \dot{\theta}_{ar\,act} R_{ar} + \dot{\theta}_{al\,act} R_{al}}{4} \tag{3}$$

Using the second PID controller, it is possible to calculate the control effort speed $v_{ctr\,eff}$.

$$v_{ctr\,eff} = k_p(v_{set} - v_{est}) + k_i \int (v_{set} - v_{est})\mathrm{d}t + k_d \frac{\mathrm{d}(v_{set} - v_{est})}{\mathrm{d}t} \tag{4}$$

Lastly, the torque sent to the actuators is each of the τ_{set} variables, one for each wheel.

$$\begin{aligned} \tau_{fr\,set} &= \tau_{fr\,out} + v_{ctr\,eff} \\ \tau_{fl\,set} &= \tau_{fl\,out} + v_{ctr\,eff} \\ \tau_{ar\,set} &= \tau_{ar\,out} + v_{ctr\,eff} \\ \tau_{al\,set} &= \tau_{al\,out} + v_{ctr\,eff} \end{aligned} \tag{5}$$

The derivatives and integrals are handled following the definitions from appendix D.

D Derivative and integral definitions

The derivative of a value received can be performed if the last value is still stored. The approximation used is,

$$\frac{\mathrm{d}y}{\mathrm{d}t} \approx \frac{\Delta y}{\Delta t} = \frac{y_2 - y_1}{t_2 - t_1}$$

where y_1 is the previous value of y received at t_1 and y_2 is the current value of y received at t_2. This approximation is better the smaller is Δt. In this project, Δt corresponds to the granularity of the system.

Following the rule that a two-dimension integral is the area under the curve, it is possible to approximate the result by using the trapezoidal rule with the received values every iteration.

$$integral += \frac{y_2 + y_1}{2}(t_2 - t_1)$$

This approximation is also better the smaller is the difference in time when the signals were received. This method requires to store the previous value of the signal y_1 and its time t_1 and also the accumulative sum of the variable $integral$.

All the operations are just basic sums and products so it is possible to group them in time windows with a specific task and similar computational load. The only limitation when designing a computational task is that it shall require less computational time than a local tick. This is checked with measurements presented in chapter 6.

E Vehicle emulator operations

The following equations are the ones employed to make the calculations in the vehicle emulator. These calculations include the steer angle rates for the front $\dot{\delta}_f$ and aft $\dot{\delta}_a$ parts of the vehicle and the velocity of the actuators v_{act}, which are needed for the calculations of the wheel angle displacement θ of each wheel. Starting with the steering rates:

$$\begin{aligned} \dot{\delta}_f &= \int \left(\frac{\tau_{fr\,set}}{R_{fr}} - \frac{\tau_{fl\,set}}{R_{fl}} \right) \frac{T_{lf}}{I_f} \mathrm{d}t \\ \dot{\delta}_a &= \int \left(\frac{\tau_{ar\,set}}{R_{ar}} - \frac{\tau_{al\,set}}{R_{al}} \right) \frac{T_{la}}{I_a} \mathrm{d}t \end{aligned} \tag{6}$$

where I_f and I_a are the moments of inertia at the front and aft parts of the vehicle.

The actuator velocity is calculated as follows,

$$v_{act} = \int \left(\frac{\frac{\tau_{fr\,set}}{R_{fr}} + \frac{\tau_{fl\,set}}{R_{fl}} + \frac{\tau_{ar\,set}}{R_{ar}} + \frac{\tau_{al\,set}}{R_{al}}}{m_v} \right) \mathrm{d}t \tag{7}$$

And the wheel angle displacements are calculated this way,

$$\begin{aligned} \theta_{fr} &= \int \left(v_{act} + \frac{\dot{\delta}_f T_{lf}}{2} \right) \frac{1}{R_{fr}} \mathrm{d}t \\ \theta_{fl} &= \int \left(v_{act} - \frac{\dot{\delta}_f T_{lf}}{2} \right) \frac{1}{R_{fl}} \mathrm{d}t \\ \theta_{ar} &= \int \left(v_{act} + \frac{\dot{\delta}_a T_{la}}{2} \right) \frac{1}{R_{ar}} \mathrm{d}t \\ \theta_{al} &= \int \left(v_{act} - \frac{\dot{\delta}_a T_{la}}{2} \right) \frac{1}{R_{al}} \mathrm{d}t \end{aligned} \tag{8}$$

Derivatives and integrals are handled the same way as in the controller operations.

F Message coding/decoding in the CAN communication

CAN communication allows for the transmission of bytes of unsigned integer data. To transmit float numbers it is necessary to code the information before the transmission and decode it at the receiving end. An example of this encoding is presented in figure (21).

Figure 21: Eight-bit data binary encoding example. The most significant bit (red) corresponds to the sign of the number (0 is positive). The grey bits correspond to the integer part of the number and the blue bits to the decimal part. This number, 67 in binary, originally represents +4.1875.

To understand how the coding works it is possible to see the example from figure (21). This could be a data value from a torque variable. Torque variables in the controller have a range of [-5, 5]. The absolute value of the integer part of the float number requires a maximum of three bits to be represented in binary ($5_{10} = 101_2$). This means that using a bit for the sign and three bits to represent the integer part, the remaining four bits can be used to represent the decimal part of the number.

Using +4.2315 as an example, the sign (+) is considered as 0 and the integer part is 4. The remaining question is how to represent the decimal part 0.2315 with a precision of four bits. The resolution of the number with four bits decimal point is $2^{-4} = 0.0625$, so dividing 0.2315 over the resolution 0.0625, the fixed result is 3 (11 in binary base). Lastly, the three numbers are summed up together taken into account their weight in the byte value: $0 \times 2^7 + 4 \times 2^4 + 3 = 67$.

Knowing the precision (bits for the decimal part) used to code the number it is easy to decode it back to its original value in the receiving end.

G Missing messages error further analysis

The missed messages have been measured for approximately fifteen minutes time using different parameters. This will hopefully give a deeper insight on different elements that can be provoking this issue.

G.1 System information exchange and total number of messages sent

The matrix cycle has a total of ten communication tasks:

1. sync0, the reference message sent by the time master at the beginning of the first basic cycle and received by all other boards. The time master has been board_1 in all the measurements presented in this appendix.
2. vote1, the message sent from board_1 to the other controller boards with the vote information.
3. vote2, the message sent from board_2 to the other controller boards with the vote information.
4. vote3, the message sent from board_3 to the other controller boards with the vote information.
5. sync1, the reference message sent by the time master at the beginning of the second basic cycle and received by all other boards.
6. set, the message with the set values from the input generator to the controller boards.
7. sensor, the message with the sensor values from the vehicle emulator to the controller boards.
8. out1, the message from board_2 to the time master with the controller calculations.
9. out2, the message from board_3 to the time master with the controller calculations.
10. outctrl, the message from the time master to the vehicle emulator with the actuator values and the error log.

The number of missed messages is the total number of times one of these communication tasks has not been acknowledged. During a task several copies of a message are sent through both channels, CAN1 and CAN2. If none of the copies sent is received, the message is counted as lost. The total number of messages sent during a matrix cycle is equal to the number of different communication tasks. A matrix cycle lasts for 50 ms, so there are 1200 messages/min of each kind. Taking into account that the measurement time was 15 ± 1 minutes, the total number of messages sent of each different kind is 18000 ± 1200 messages. When increasing or reducing the COMM period the matrix cycle duration also changes. For COMM = 30 ticks the matrix cycle duration is 40 ms and for COMM = 50 ticks the matrix cycle duration is 60 ms, so the amount of messages sent is 22500 ± 1500 messages and 15000 ± 1000 messages, respectively.

G.2 Missed messages variability

When repeating the missed messages measuring with the same parameters, we observe some intrinsic variability in the results. The parameters chosen are COMM = 40 ticks, $\phi_1 = 5$ ticks and $\phi_2 = 5$ ticks, with a baud rate of 1 Mbit/s. An analysis of five measurements with the mean, standard deviation, maximum and minimum of every different board misses for every different message is presented in figure (22).

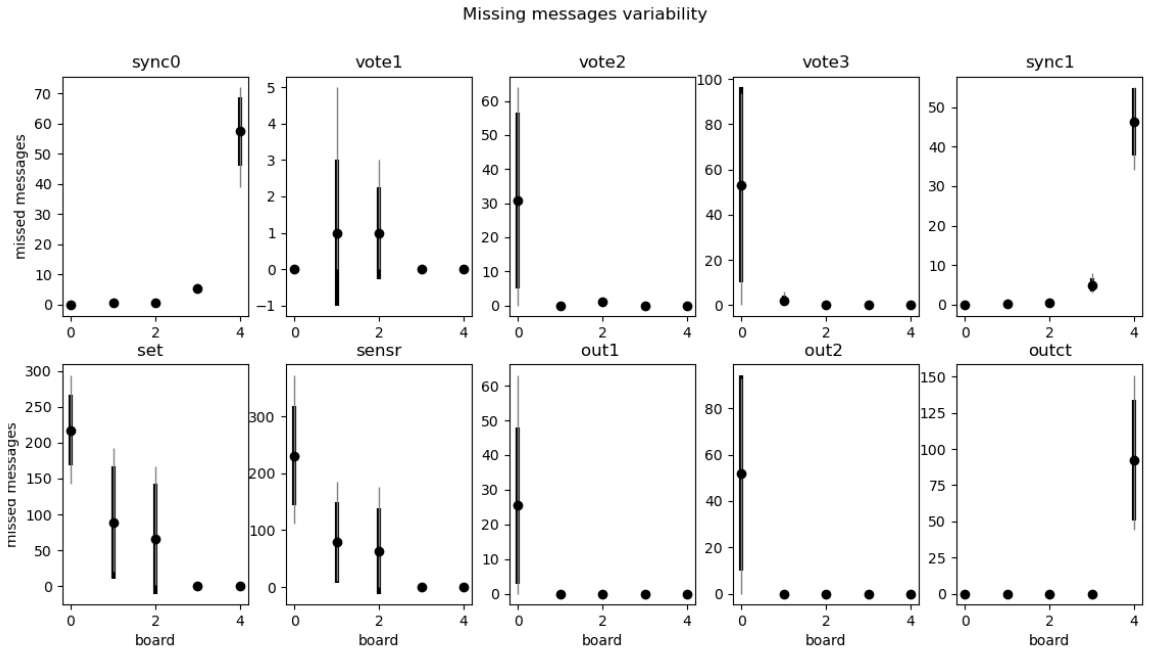
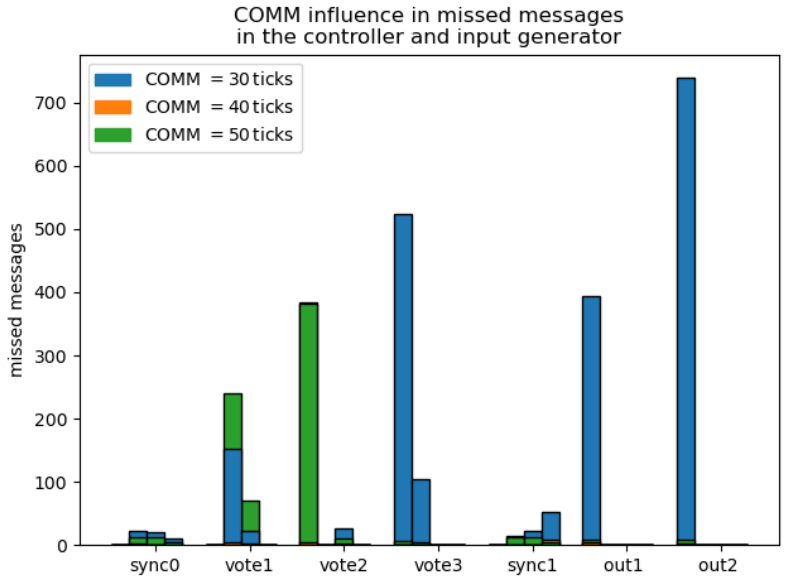


Figure 22: Mean (dot), standard deviation (darker bar) minimum and maximum found for five different measurements of the missed messages for every board at the reception of each different message. Each dot in each subfigure represents a board, from left to right, board_1, board_2, board_3, input generator and vehicle emulator.

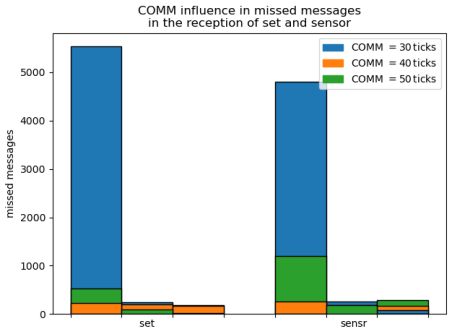
In figure (22) it is possible to see that the missing messages of different boards for some of the messages present a high variability. For example, board_1 has been observed to miss between 0 and more than 60 messages when listening to vote2 message. This variation analysis sets some background to understand that making a single recording with a particular set of parameters that shows a low amount of messages missed does not mean the missing chance is necessarily always low for that particular configuration.

G.3 Influence of different parameters

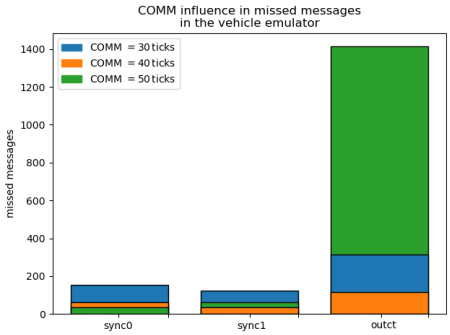
As already stated in the discussion of the report, the messages could be missed for different reasons. When thinking about parameters that could be changed in the software that could influence the missing rate the communication period could be important. Sending more copies of a message during a task with a longer communication period or reducing the communication time with a shorter period could change the miss chance. A single recording for three different COMM times is presented in figure (23).



(a)



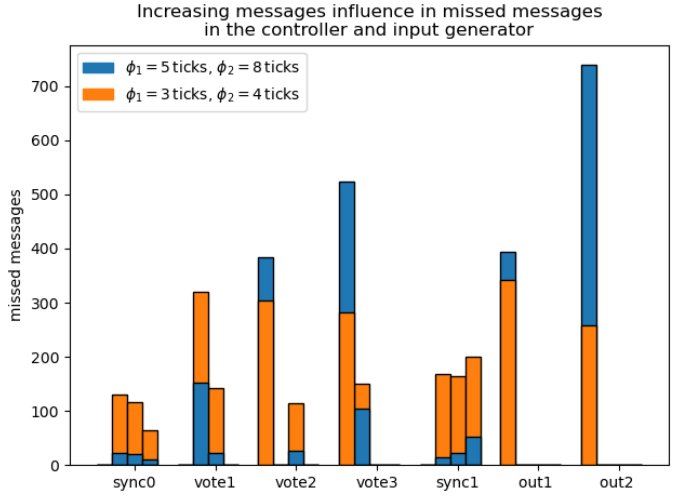
(b)



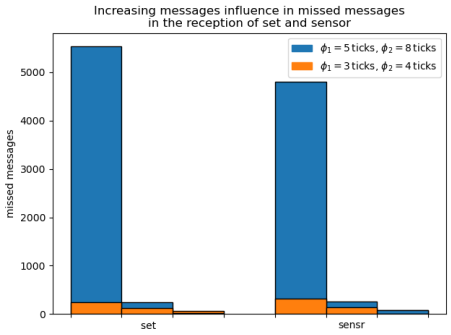
(c)

Figure 23: Missed messages for every different board listening to each different message for different COMM values. (a) The controller and input generator listen to controller messages, (b) the controller listen to set and sensor messages and (c) the vehicle emulator listens to time master messages. Every board misses are represented by a bar for each message, from left to right, board_1, board_2, board_3, input generator and vehicle emulator.

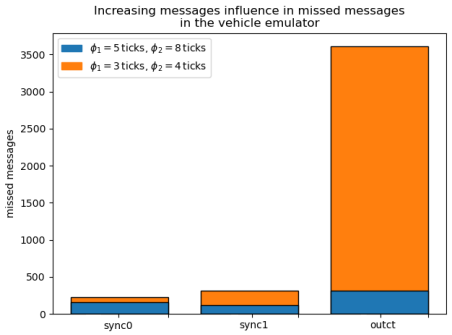
In general, taking into account the variability from figure (22), COMM = 30 ticks and COMM = 50 ticks present worse results than COMM = 40 ticks. It seems the maximum set load of seven copies per message presents a worse behaviour than sending some lower amount, as can be seen for the longest COMM period. For a shorter COMM period it is possible that the amount of copies sent per message is too low, so the chances of a missed message increase. The next analysis focuses in the amount of copies sent during the same task. Figure (24) shows the missed messages for COMM = 30 ticks for two different transmitting frequency configurations. For $\phi_1 = 5$ ticks, $\phi_2 = 8$ ticks (the frequency chosen for the COMM measurement) the amount of messages per task is eight. For $\phi_1 = 3$ ticks, $\phi_2 = 4$ ticks the amount of messages per task is fourteen (the maximum allowed in the prototype).



(a)



(b)



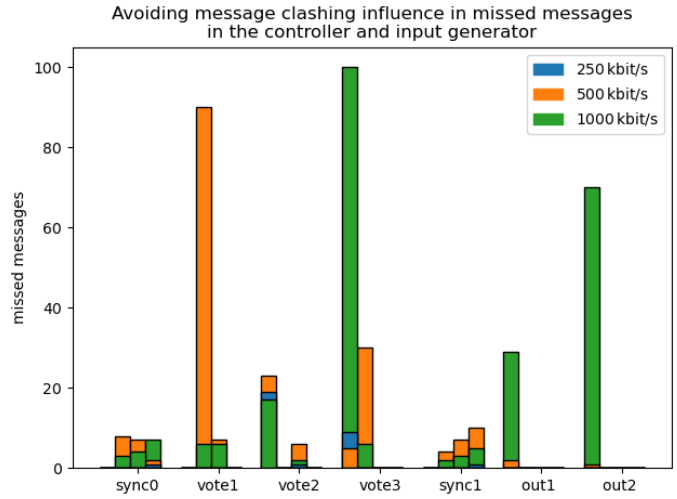
(c)

Figure 24: Missed messages for every different board listening to each different message for two different transmitting frequencies. (a) The controller and input generator listen to controller messages, (b) the controller listen to set and sensor messages and (c) the vehicle emulator listens to time master messages. Every board misses are represented by a bar for each message, from left to right, board_1, board_2, board_3, input generator and vehicle emulator.

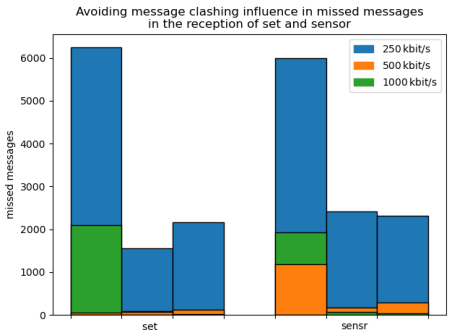
Sending more copies of the same message does not seem to improve the performance in general. However, it is interesting to see how for a lower amount of messages the reception at the controller for the set and sensor messages is better, while for a higher amount of copies the reception at the vehicle emulator for the outctrl message is better.

The last configuration explored tries to set a message frequency and baud rate couple that avoids message clashing in the network. According to (CAN_CiA, 2022), lower baud rates are more robust. Also, waiting until the last message sent has arrived at its destination could help in CAN management during a communication task, resulting in a lower missing rate. With baud rate = 1 Mbit/s a total of 4000 bits can be transmitted during a COMM = 40 ticks communication task. Baud rates of 500 kbit/s and 250 kbit/s has been used for COMM = 50 ticks, which allow for 1250 bits and 2500 bits, respectively. Assuming a worst case scenario of 130 bits per message frame in its copy transmission, as presented in (Lundin & Rai, 2019), there is no case in which the maximum of seven sent copies is over the capabilities of the network. When checking the communication delay for different baud rates, 3 ticks delay was found for 1 Mbit/s, 5 ticks for 500 kbit/s and 7 ticks for 250 kbit/s. At least, these amounts of ticks have been set from one copy to the next in the transmission for the measurements presented in figure (25). The message phases chosen are $\phi_{init2} = 5$ ticks, $\phi_1 = \phi_2 = 10$ ticks for COMM = 40 ticks and baud rate 1 Mbit/s, $\phi_{init2} = 5$ ticks, $\phi_1 = \phi_2 = 10$ ticks for COMM = 50 ticks and baud rate 500 kbit/s, $\phi_{init2} = 8$ ticks, $\phi_1 = \phi_2 = 16$ ticks for COMM = 50 ticks and baud rate 250 kbit/s.

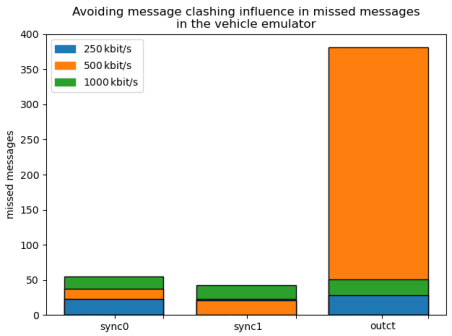
When looking at the results for avoiding message clashing in the network, there is not a noticeable difference when measuring at COMM = 40 ticks with respect to the results when sending the messages without avoiding message clashing. With lower baud rate the reception of the messages set and sensor by the controller is unstable but the reception of outctrl at the vehicle emulator is very good. The opposite happens for baud rate 500 kbit/s.



(a)



(b)



(c)

Figure 25: Missed messages for every different board listening to each different message for three avoiding message clash configurations. (a) The controller and input generator listen to controller messages, (b) the controller listen to set and sensor messages and (c) the vehicle emulator listens to time master messages. Every board misses are represented by a bar for each message, from left to right, $board_1$, $board_2$, $board_3$, input generator and vehicle emulator.

G.4 Conclusions

With the current data it is not possible to assess a perfect scenario where the missing messages is almost zero. However, it is possible to see some negative correlation between the reception of the messages set and sensor received by the controller boards and the message outctrl received by the vehicle emulator. When there is a low load of copies per message or the baud rate is 250 kbit/s the reception of the messages set and sensor is unstable at the controller but the message outctrl is received well by the vehicle emulator. The situation gets reversed when there is a high load of copies per message or the baud rate is 500 kbit/s. The best configuration found is the presented for figure (22).