



CHALMERS



Development and implementation of remote steering

Bachelor's thesis in Mechatronics Engineering

**OSSIAN BERGSTRÖM
TOBIAS WERNERSSON**

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

BACHELOR'S THESIS 2021

Development and implementation of remote steering

OSSIAN BERGSTRÖM
TOBIAS WERNERSSON



CHALMERS

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Development and implementation of remote steering
Ossian Bergström
Tobias Wernersson

© OSSIAN BERGSTRÖM, 2021.
© TOBIAS WERNERSSON, 2021.

Supervisor: Klomp Matthijs, Solution Architect, Volvo Cars
Fredrik Bruzelius, VTI, Chalmers University of Technology
Examiner: Fredrik Bruzelius, VTI, Chalmers University of Technology
Report number: 2021:02

Bachelor's Thesis 2021
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Car that were used during thesis work to apply remote steering

Typeset in L^AT_EX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Abstract

During the last couple of years, the strive for developing autonomous cars has drastically increased. With this said, new problems arise parallel to the evolution of this new technology, as it grows bigger. This thesis examines an alternative way of steering these autonomous cars, when the cars themselves malfunction or are unable to manage a certain situation. Volvo Cars AB presented a thesis concept that involves an investigation of the possibilities of steering a Volvo XC60 remotely from a central communication unit. The method applied to achieve this utilized UDP data transmission over Wi-Fi and a remote gaming steering wheel. To remotely send the input steering angle value from the remote steering wheel to be received by a VN8911 module implemented on a test rig in the first phase of the thesis and later on being implemented on a XC60.

To analyse if the implementations and the alternative way of steering the XC60 remotely, function as intended. A couple of qualitative oriented tests were performed to determine whether the alternative way of steering will work in practise. This based on the test drivers' intuition and personal judgement. Furthermore, the test driver will manoeuvre the car simultaneously as watching a live streamed video of the test course with the help of a GoPro Hero 8.

Due to this being a bachelor thesis, some limitations were applied to cut down any unnecessary time demanding steps, in order to have enough time to finish within the timespan of the thesis. The limitations of the thesis will include Wi-Fi only being implemented during the test rig tests and not during the tests performed with the implementations on the XC60. Instead, the tests will be performed over LAN and the Wi-Fi latency that would have been applied will instead be replaced by a simulated time delay implemented in the sending UDP programming script. Furthermore, the test driver will simulate the remote environment of steering the car from the backseat watching the livestreamed video.

Towards the end, the tests performed on the test rig provided an outcome enabling an implementation of the developed hardware and software setup, on a Volvo XC60. By analysing the data achieved from the tests performed on the car, an outcome indicating great drivability and manoeuvrability was achieved. Whilst operating during relatively high velocities and with different time-delays.

Keywords: Remote steering, UDP, CANoe, CAPL, VN8911, Python, Tests, Autonomous Cars.

Acknowledgements

We would like to extend our inmost gratitude to Volvo Cars for providing us such a thrilling opportunity and project to test our engineering abilities, It has been an absolute honor.

First and foremost, we would like to give special thanks and appreciation to our supervisor Matthijs Klomp for his counseling and support during the entire duration of this thesis. For showing great interest and great passion for the subject itself and for the provided guidance and improving the quality of our bachelor's thesis.

Additionally, we would like to pay our gratitude to the people working at Volvo cars who helped us during different circumstances. Especially Bill Nakos and Jeber Mandus for sharing their knowledge and expertise while providing us a helping hand regarding both hardware and software related issues.

We also would like to thank our examiner Fredrick Bruzelius for his expertise in signal processing and in many other subjects and for all conversations and discussions along the way.

Last but not least, we would like to thank Ravi Linder and Emil Johansson for providing great company and humorous conversations during days at the office.

Ossian Bergström and Tobias Wernersson, Gothenburg, June 2021

Contents

List of Figures	xiii
List of Tables	xv
Terminology / Abbreviations	xvii
1 Introduction	1
1.1 Background	1
1.2 Aim	1
1.3 Limitations	2
1.4 Specification of issue under investigation	2
2 Theory	3
2.1 Python	3
2.1.1 Pygame	3
2.2 Matlab	4
2.3 OBS Studio	4
2.4 Network Latencies	4
2.5 UDP: User Datagram Protocol	4
2.6 Transfer function	6
2.7 Bode Diagram	6
2.8 Coherence	7
2.9 CANoe	7
2.9.1 Controller Area Network	7
2.10 VN-module	8
2.10.1 VN-8911 Setup	8
2.11 Electrical Control Units and Interface	9
3 Methods	11
3.1 Hardware implementations	12
3.1.1 Rig	12
3.1.2 Breakout Cabling Kit	13
3.1.3 Remote Steering Wheel	15
3.1.4 GoPro	15
3.1.5 Additional Hardware	15
3.1.6 Car Setup	16
3.2 Software implementations	17

3.2.1	Reading the steering angle	17
3.2.2	Sending the value over UDP	19
3.2.2.1	UDP - Python	20
3.2.2.2	UDP - CAPL	21
3.2.3	The Rig and the whole setup	22
3.2.4	LAN and Wi-Fi tests	24
3.2.5	Test page explanation	24
3.2.6	Simulation setup for car implementation	25
3.2.7	Matlab code	27
3.3	Tests and Simulations	28
3.3.1	Sending and Receiving	28
3.3.2	Rig And Car Test	29
3.3.2.1	Rig test	29
3.3.2.2	Car Test	29
3.3.2.2.1	Maximum Delay	30
3.3.2.2.2	Low Intensity	31
3.3.2.2.3	High Intensity	32
3.3.3	Evaluation	32
4	Results	35
4.1	Latency tests of LAN and Wi-Fi	35
4.2	Vehicle test results	36
4.2.1	Low intensity test results	36
4.2.2	High intensity test results	42
4.3	Test Sheet and Inaccuracies.	48
4.3.1	Inaccuracies noticed	48
4.3.2	General thoughts	48
4.3.3	Evaluation of Maximum Delay	48
4.3.4	Evaluation sheet	49
4.3.5	Further Thoughts Regarding Tests	50
5	Discussion	53
5.1	Overall conclusion	53
5.2	Hardware implementations	53
5.3	Software implementations	54
5.4	Further developments	55
	Bibliography	57
A	Appendix A	I
B	Appendix B	III
C	Appendix C	VII
D	Appendix D	XI
E	Appendix E	XV

F	Appendix F	XIX
----------	-------------------	------------

List of Figures

2.1	Typical architecture of a User Datagram protocol message.	5
2.2	Visualizing the sending structure of data transportation over UDP . .	6
2.3	Typical architecture of a CAN-based system	8
2.4	Vn-module and user desktop connection.	9
2.5	IP-address configuration of VN8911	9
3.1	Network setup without break out cable.	12
3.2	Connection between rig, VN8911 and computer	13
3.3	Breakout Cabling Kit	13
3.4	CANoe configuration network setup with the breakout cable.	14
3.5	Connection between rig, VN8911 and computer with breakout cabling kit	15
3.6	Displays the setup for steering a car	16
3.7	Code describing the initialization of the Pygame library and the joy- stick module.	17
3.8	Code describing the counting and initialization of the connected joy- stick devices.	18
3.9	Code describing the method of how the program attains the input value of the remote steering wheel.	19
3.10	Code describing the manipulation of the steering angle value in Python, to a value susceptible for CAPL.	19
3.11	The receiving node from the simulation setup in CANoe.	20
3.12	Code describing the declaration of datagram socket following the UDP structure.	20
3.13	Code describing the implementation of <i>OPEN</i> and <i>CLOSE</i> button on the test page.	21
3.14	Code describing the final modification before sending the steering angle value to the CAN bus.	22
3.15	Visual representation of the created simulation setup utilized on the tests performed on the test rig.	23
3.16	Visual GUI representation of the test page in CANoe, used for direct interaction between the car and the GUI.	25
3.17	Visual representation of the created simulation setup utilized on the tests performed on the XC60.	26
3.18	Visual representation of CAN2_Networks and ChassiCAN_VDDM CAN busses in the simulation setup.	26

3.19	Code describing an example of sending Frames and signal between CAN networks programmed in the Gateway node.	27
3.20	Code describing the functions generating the data analysis from the different tests performed.	28
3.21	Remote steering wheel and computer that display the view placement	30
3.22	Figure Demonstrating test track at PV-north.	30
3.23	Caption	31
3.24	Illustrating concourse for high intensity test	32
4.1	Bar graph visualizing the difference in latency between Wi-Fi and LAN performed 5 times.	36
4.2	Timespan plot for 0 ms simulated time-delay during the low intensity test.	37
4.3	Timespan plot for 50 ms simulated time-delay during the low intensity test.	38
4.4	Angular difference in the timespan plot during 50 ms low intensity test, between input and output steering angle	38
4.5	Bode plot and coherence plot with the time-delay 0 ms during the low intensity test.	41
4.6	Bode plot and coherence plot with the time-delay 50 ms during the low intensity test.	42
4.7	Timespan plot for 0 ms simulated time-delay during the high intensity test.	43
4.8	Timespan plot for 50 ms simulated time-delay during the high intensity test.	44
4.9	Timespan plot for 150 ms simulated time-delay during the high intensity test.	44
4.10	Bode plot and coherence plot with the time-delay 0 ms during the high intensity test.	46
4.11	Bode plot and coherence plot with the time-delay 50 ms during the high intensity test.	47
4.12	Bode plot and coherence plot with the time-delay 150 ms during the high intensity test.	47
4.13	Illustrations of how the car was steered through cone track with different delays	51
A.1	Maximum delay table	II
A.2	Maximum delay table continuation	II

List of Tables

3.1	Maximum Delay	31
3.2	Low intensity attempts	31
3.3	High intensity attempts	32
4.1	Result for each test, Question 1	49
4.2	Result for each test, Question 2	49
4.3	Result for each test, Question 3	49
4.4	Result for each test, Question 4	50
4.5	Result for each test, Question 5	50

Terminology / Abbreviations

- CAPL - Communication Access Programming Language
- CAN - Controller Area Network
- ECU - Electrical Control Units
- PSCM - Power Steering Control Module
- SAS - Steering Angle Sensor
- VDDM - Vehicle Dynamic Domain Master
- LIN - Local Interconnect Network
- PAP - Parking Assist Pilot
- DIM - Driver Information Module
- SUM - Suspension Module
- LAN - Local Area Network
- SAP - Steering Assist Pilot
- GUI - Graphical User Interface
- Wi-Fi - Wi-Fi is a family of wireless network protocols
- SASChasFr01 - Steering Angle Sensor Chassis Frame 01
- PinionSteerAG1 - Pinion Steering Angle 1
- VDDMChasiFr15 - Vehicle Dynamic Domain Master Chassis Frame 15
- ParkAssiPinionAgReq - Park Assistant Pinion Angle Required
- SteerWhlAgSafe - Steering Wheel Angle Safe
- SPA - Scalable Product Architecture
- UDP - User Datagram Protocol

1

Introduction

The modern society undergoes constant changes in every area regarding it. New inventions and developments of already existing products are invented and new implementations of these developments and inventions are created. There is endless possibilities for the development of new technology and in the current time, there are a lot of focus on the connection and the interaction between humans and technology. Technology that are commonly used for this purpose is for example the touchscreen, which enables the user to physically communicate with its user, and Wi-Fi which enables the user to communicate wirelessly with the product. To be able to communicate from a wireless distance is a powerful tool. This thesis examines if it is reliable enough to send information over Wi-fi in a safe and secure manor with as low latency as possible.

1.1 Background

The car industry is evolving at a high velocity and the direction the development is going is reflected in the cars produced. Nowadays the interior components of the car heavily consist of electronic components with an increased number of interactive features which allows an improved interaction between the driver and the car. The developments have gotten as far as designing and researching in self driving, autonomous cars controlled by algorithms. Therefore, there is an increasing need of exploring the wireless possibilities of driving a car to explore the possibilities of doing so as an alternative way of steering a car if needed. The possibilities such as how much latency there is before turning unstable and if steering the car over Wi-Fi is manageable, or if the time delay will be too great to be able to steer the car in any relevant velocity. The intention behind this thesis is therefore to research in this area and explore how well this will work, both in theory and in practise.

1.2 Aim

The objective of this thesis is to explore how well steering a car over Wi-fi, using UDP data transmission, will work. Will the time delay by doing so be too great or will it be small enough so that steering the car will end up manageable? Will this way of wireless information delivery be reliable enough to not jeopardize the safety of the driver and its surroundings while driving the car? These are the questions that are going to be answered in this report. The anticipated result of this project is to get a clear answer, whether it is possible to steer the car or not in a relevant

speed. Hopefully, the result of this thesis end up in a positive manner where driving the car with the implemented setup translates to be with high measure of drivability, while being tested safely. This with low latency so that it is possible to steer the car at a relevant velocity.

1.3 Limitations

With a detailed and thorough investigation of the areas touched upon in the thesis, the possibilities and practical implementations of analysing an alternative way of steering a Volvo XC60 will be made. To create a scenario where the desired result is achievable, regarding the size of the thesis, some limitations are needed. These limitations are stated below.

- The only test drivers performing the tests will be the authors.
- The Driver's visual perspective while maneuvering the vehicle remotely will only be a linear perspective from the front end of the car.
- The high intensity test will only cover simulated time-delays in the range of 0-50 ms and the low intensity test within the interval of 0-150 ms.
- The maximum velocity of the performed tests during this thesis, will be a velocity of 40 km/h.
- The tests will only be implemented and performed on a Volvo XC60.
- This report will not cover any implementation of how well controlling the throttling or breaking will work over Wi-Fi.
- This report will only cover the usage of data transportation over UDP-protocols.
- The focus on this report will be on the proof of concept. A demonstration and a verification of a concept or theory to review its practical potential.

1.4 Specification of issue under investigation

To create a scenario where steering a Volvo XC60 over Wi-Fi is achievable, a formulation of the specification of issues under investigation is a requirement. There is a great importance in specifying these issues in such a way that make it easy and straightforward to judge if the process of the thesis is going in the right direction. Following below are the problem statements formulated for this thesis.

- Will it be possible to manoeuvre a car implemented with remote steering using UDP?
- How does one modify and create a setup applied on the XC60 to enable the possibility to steer the car remotely?
- How does one create a valid and functional test track environment?
- How well will manoeuvring a Volvo XC60 through a created test track do, regarding the implementation of different time-delays and different velocities applied during these tests?

2

Theory

The following chapter of the thesis presents the prior knowledge and theory required for understanding the thesis.

2.1 Python

The programming language Python is partly based on the usage of a vast number of open libraries and its included classes and functions. With the help of these classes and functions it gives the user the possibility to write and design any desired program to perform the task intended for it. Furthermore, Python is an object-oriented programming language that focuses on objects and classes rather than functions. This allows the user to create and manipulate classes and objects which makes it beneficial to write large and complex programs that are easy to understand and edit. Therefore, Python allows the user to create and write programs swiftly and efficiently with a straightforward code construction that emphasizes on readability and accessibility [1]. In this thesis, three main libraries are used; Socket library, Time library and Pygame library.

The socket library allows the user to use the socket objects which enables the usage of different data transmission protocols and other TCP/IP transportation layers [2]. Additionally, the time library gives the user the possibility to use different time-related objects and functions [3]. The Pygame library allows the user to interact with gamepads and joysticks devices.

2.1.1 Pygame

Pygame is a free and open-source python-based programming language library [4]. This means that anyone can use its content if a reference is made. The library is a great tool to use for these exact purposes of creating inputs and outputs for a game-like programming script or similar. Regarding this thesis, the functions from the joystick module from the library was used. This module is specifically created for interacting with gamepad and joystick devices. It includes multiple functions intended to manipulate the plugged-in joystick device.

2.2 Matlab

Matlab is one of two main softwares included in Mathworks. The founders, Jack Little and Cleve Moler, developed and founded this software in 1984. Matlab is a well-known programming and numeric computing platform, designed to allow the user to use all their content for math, graphics, and programming [9]. It is a tool for engineers and scientists to develop their own programs and to analyse data. Furthermore, it gives the user the possibility to create their own algorithms and models.

2.3 OBS Studio

OBS Studio is an application for video recording and live streaming built on an application programming interface that allows the user to interact with multiple software applications at the same time. OBS Studio was founded and still maintained by Hugh “Jim” Bailey and developed by the community utilizing the software [11].

2.4 Network Latencies

All data transmissions utilizes different means of network transportation. The most common networks used are Wi-Fi, LAN, internet, mobile 4G network and mobile 5G network. All these different networks are optimized for different scenarios and purposes. Furthermore, each of them has different settings and specifications and therefore different latencies for end-to-end data transmission. The two relevant latencies used during the thesis is Wi-Fi and LAN. The fastest method with the lowest latency is a wired one, due to few latency demanding steps between the sending and the receiving client. For the Wi-Fi communication, the data transmission first has to go through the Wi-Fi router that later on sends it to the receiving client. By having these extra steps in the transportation duration, a higher latency is therefore achieved.

Other types of latencies are video-based latencies regarding the time-delay between camera and display. Due to the large amount of data required to be transferred over Wi-Fi or LAN, the latency will be larger compared to sending a small value or similar data. By observing the OBS studio software, the live streaming latency can be adjusted in the advanced settings. The lowest latency achievable while utilizing this software is a latency of around 500 ms.

2.5 UDP: User Datagram Protocol

The user datagram protocol (UDP) is one of the two most common protocols under the transport layer in the TCP/IP model [5]. The TCP/IP model exists of five layers which are application layer, transport layer, network layer, data link layer and the physical layer. All five layers handles different tasks in the TCP/IP model, where

the transport layer provides the wireless end-to-end data transmission between two units. The UDP protocol is the more unreliable one of the two common protocols in the transport layer. This due to its protocol structure and how the process of sending and receiving a data package works.

The structure of the UDP-package consists out of a header which contains two 4 bytes sequences of necessary information, see figure 2.1. Every sequence includes two ports of 16 unsigned bits and the first 4 bytes contains information such as the source port and the destination port. This is where the UDP-package receives its information regarding the sending and receiving process. The first 16 bits of the second sequence consists of the length of the sequences of the UDP-package and the UDP checksum on the remaining bits, which checks if the package is corrupt or not. The following sequences contains the user defined data, that is the message that the user desires to send using this protocol [7].

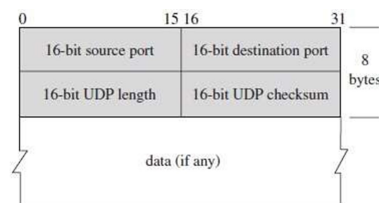


Figure 2.1: Typical architecture of a User Datagram protocol message.

Initializing of a socket is needed for data transmission between two clients. Also, a socket type is required to be determined. There are different types of sockets. The UDP protocol is using a message-oriented socket type called datagram sockets, for data transportation between two devices. By sending data using this socket type, the packages are sent one at a time without a receiving confirmation from the opposing end of the transmission [8]. The client in both end of the end-to-end communication is required to create a socket with purpose to either receive from or send data to the UDP server. Afterwards, the socket of each client is required to be defined to the same port, to enable data transmission with the defined server. As seen in figure 2.2, the sending client is requesting a connection to the UDP server to send the data package. The server then defines a new socket and start to listen for incoming requests from the receiving client. When the connection been made with a receiving client a transmission of data can be made.

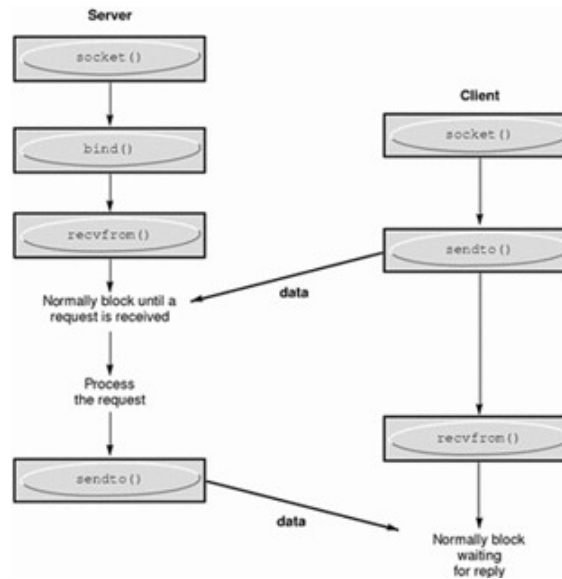


Figure 2.2: Visualizing the sending structure of data transportation over UDP

The client utilizes the function `sendto()` to send data packages over to the server, as shown in figure 2.2. The server receives the package by calling the function `recvfrom()` and then initialize a process of waiting for a receiving client to request for a data transfer. The receiving client then calls the function `recvfrom()` and gives the server the request required to enable the final data transfer.

2.6 Transfer function

Achieving deeper understanding of how different signals behave and how its respective output signals vary after undergoing a certain process, can be useful. There is a great importance in acquiring correct and rich data to thereby achieve knowledge otherwise unattainable through our own eyes. A Transfer functions is a great tool for signal processing and to understand how a system behaves and affects the corresponding inputs and outputs. It is also often utilized for measurement purposes and data analysing within a bode plot. It compares the input and output signals of a system to visually and theoretically compare the achieved data from different tests on a bode plot.

2.7 Bode Diagram

It is possible to estimate a system with the help of its in and output signals. With the help of a bode plot it is then possible to visualise the characteristic's between the system's in and output. From the characteristic's, data can be extracted to later on be able to draw conclusions regarding the system's abilities [19].

2.8 Coherence

One way of judging the correlation of two signals is to measure the coherence between them. The coherence is explained as a relation between two propositions and the correlation between them. The two propositions can be anything free of choice, the coherence still measures the amount of proposition A that is in proposition B [6]. According to [18], coherence can be used to study if there is a linear relation between two signals. Which values that are seen as good measures is up for debate and depends on the situation, but if the coherence is equal to one the signals are in perfect linear relation.

If the coherence value drops below one it means that the signals are not in a perfect linear relation. Reasons behind this can be due to that the input signal does not contain some of the specific frequencies. Which means that the data collected only consist of noise for the specific frequencies. Or if there is a non-linearity, which means that it does not exist an linear relation between the signals.

2.9 CANoe

CANoe is a software tool for analysis, test and development of entire Electrical Control Units (ECU) networks and individual ECUs. ECUs send and receive signals with the help of CAN, LIN, and FLEXRay for instance. In CANoe it is possible to create simulation models that simulate the behaviour of a real ECU. CANoe has its own Communication Access Programming Language (CAPL). Which is very similar to C-programming language. CAPL offers the possibilities to manually program the test, and manually set up your ECU [10].

2.9.1 Controller Area Network

Controller Area Network (CAN) was introduced by Robert Bosch in 1986, due to obtain robust communication between the rising electrical control units used in cars. CAN is nowadays applied in every automotive vehicle and electromechanical components for communication between the different ECUs. In 1993 the CAN standard ISO 11898 was released [12]. CAN is not a master/slave system, all control units (nodes) can send and receive a message. A CAN message consists of several bits and is also called a frame. There are four types of frames. Data frames that contain information from a source sent to receiver/receivers. Remote frames which are used to request transmission of a data frame. Error frames that are sent out if an error is detected on the network. Overload frames which are controlling the flow and can if needed request time delay. Also, Data frames can be divided into two different types of frames. The standard one where the Identifier is 11-bits and the extended where the Identifier is 29-bits. Figure 2.3 shown below illustrates the typical architecture of a CAN-based system.

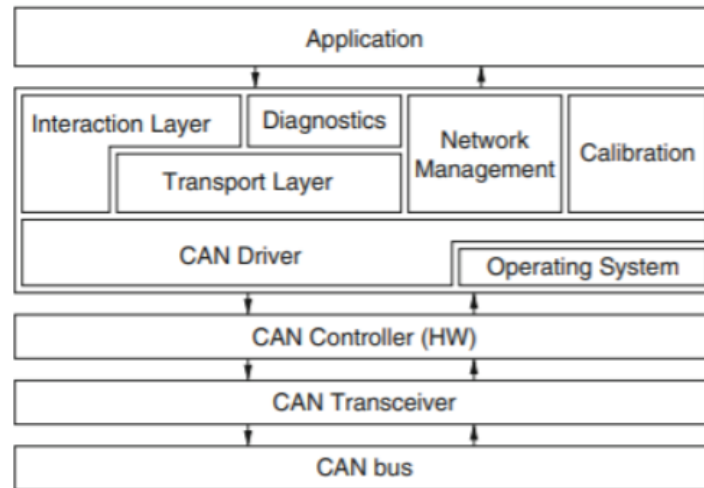


Figure 2.3: Typical architecture of a CAN-based system

2.10 VN-module

To establish connection between external software and the bus system integrated in a car a VN-module can be used. In this project the advanced VN-8911 was suitable due to that it has the possibility of internet connection. A VN-8911 can best be described as a small computer with an Intel ATOM processor. It is designed to communicate with control units in any application. It can be run in several modes [16]. The mode real time processing is used in this thesis due to the need of very low latency. It is used together with CANoe. VN-module execute real-time simulation and test functions and its graphic interface is displayed on the computer connected to it. Figure 2.4 below illustrates its connection. The VN-module is in this case connected to the CAN-bus to read or write to frames sent out by PSCM and SAS. Which handles the frames necessary for steering.

2.10.1 VN-8911 Setup

As described earlier, the VN-module can be seen as a small computer. In order for it to work properly with our CANoe configuration and rig/car. A few settings need to be set. It is done through CANoe under the tab Vector Hardware Configuration. Once it is open the user can choose which mode the module should operate in and configure its IP-address. Configuration of its IP-address is important due to this project use of UDP and LAN to send and receive data. Figure 2.5 below shows how the specified settings are done.

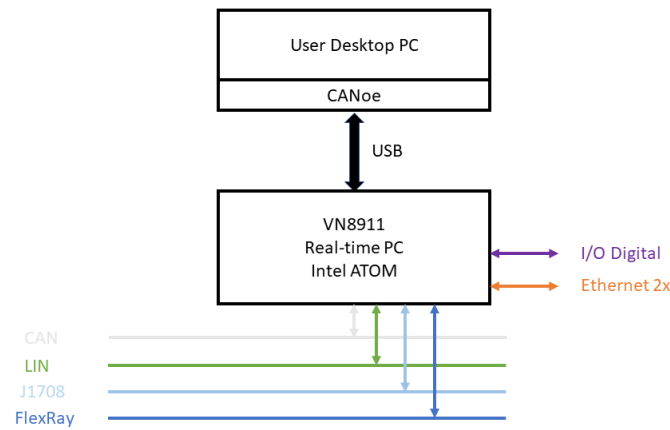


Figure 2.4: Vn-module and user desktop connection.

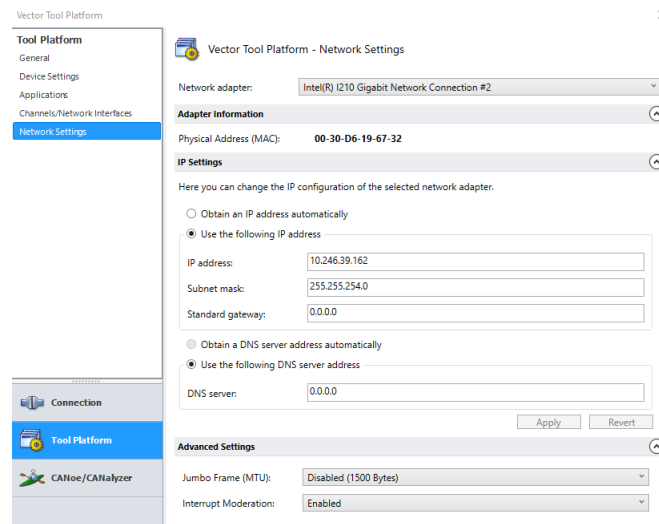


Figure 2.5: IP-address configuration of VN8911

2.11 Electrical Control Units and Interface

ECUs that handle frames related to steering is Power Steering Control Module (PSCM), Vehicle Dynamic Domain Master (VDDM) and Steering Angle Sensor (SAS) measure the steering column angle. They transmit and receive frames related to steering to one and other. Actions taken by the ECUs is depending on what interface that is active. ECUs above communicate with each other by transmitting and receiving frames to ensure that steering works properly. Parking Assist Pilot (PAP) is a form of interface. It is constructed to find and maneuver your car when parking. It can also assist when steering out of a parallel parking spot. PAP is impossible to activate when a trailer is attached. On every occasion it does only

assist the steering, braking, gas and supervision is the driver's responsibility. It is possible for PAP to handle steering because of its programmed controller [14].

3

Methods

There are multiple ways of achieving a desired result. The thesis is divided into two different sections/chapters. In the first section a small quantitative oriented survey was used to find the answer to the problem that was implied in the first problem statement. The statement about being able to manoeuvre a car through a test track while controlling the steering rack with a remote steering wheel, but through a purely technical viewpoint. Plenty of layers of investigation was needed to achieve a result, which gave the opportunity to explore the following section of the thesis. In the second section the methodology used where more qualitative oriented and less data heavy, where the focus was rather based on a personal point of view and impression on whether the car was manoeuvrable through the created test tracks or not.

First and foremost, to have the possibility to conduct any tests at all, a lot of problems of different technical areas must be solved. The whole thesis itself is reliant on all these solutions to cooperate to create a scenario where the alternative way of steering a XC60, which this report is based on, becomes a reality. During the planning period of the thesis a basic idea was created of how the problem statements were to be solved. The objective was to send a steering angle value over Wi-Fi using UDP to steer a car. To be able to send the value remotely, a script written in Python was used for reading the steering angle value generated from an alternative hardware and later send it. The receiving end of the UDP-protocol is the XC60 internal software. To enable the data transmission between the python script and the internal software of the car, a VN8911 module must be utilized as a bridge between the car and UDP packages. When the data has been received to the VN8911 module, the real value of the steering rack can then be overwritten by modifying the value of the input signal received by the VN8911 module to the output signal.

Secondly, after a solution has been accomplished and manoeuvring a car is possible, the second chapter of the report can be initiated. The chapter more focused on a qualitative oriented method approach, where tests are used to examine whether the previous technical part of the report works in practise. This is where the second and third problem statement is answered. To be able to examine the functionality and effectiveness of the implementations, the tests is required to be designed in such way where it produces accurate data. The objective with the thesis is to see how well the car can be manoeuvred in a real-life scenario. In a scenario where the way of driving is not necessarily an intense one where a lot of quick and powerful manoeuvres are

performed. To create test environments where accurate data can be analysed and where conclusions can be made on which driving scenarios suits the implementations of the car better or worse. To determine this, two different tests were decided to be used. One for high intensity driving and one for low intensity. The test drivers then determine the feel of the car based on their own intuition and judgement.

3.1 Hardware implementations

To be able to create an environment for test of remote steering. Different hardware was needed to be included and remodelled to work with each other. At first the test of different solutions was conducted on a rig. The best working solution was later transferred to a car. Below is a brief description of each hardware and its setup.

3.1.1 Rig

A former test rig that is designed to test mechanical hardware and software updates related to steering. The rig is based on Volvo's SPA platform and is equipped with some of the ECUs found in a car. Power Steering Control Module (PSCM, CAN) which contains the autopilot (traffic jam assist) function. Steering Angle Sensor (SAS, CAN) the measured steering angle. Driver Information Module (DIM, FlexRay) shows vehicle speed, gear etc [15]. Due that both the Rig and car used in this thesis is based on the SPA platform transition back and forth are possible. Figure 3.1 shows how the configuration in CANoe were constructed to be able to read and write to specific frames. Some of the ECUs had to be simulated to represent the non existing.

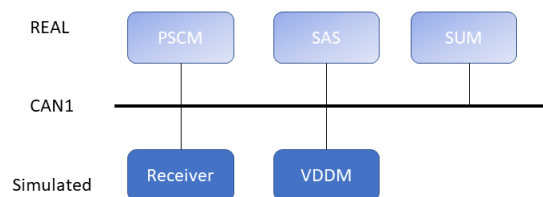


Figure 3.1: Network setup without breakout cable.

Before the rig was able to be used a series of small modifications were to be done. The wires connected to the power steering motor were replaced to better suit our

As can be seen above in the figure 3.3 the frames sent out and received by the PSCM travels through the VN-module. Which in this case makes it possible to modify the frames needed for the steering rack to move in the specified direction. As well as have the car believe that it is a correct action. Figure 3.4 display how the CANoe configuration were modified to have the possibility to change specific frames.

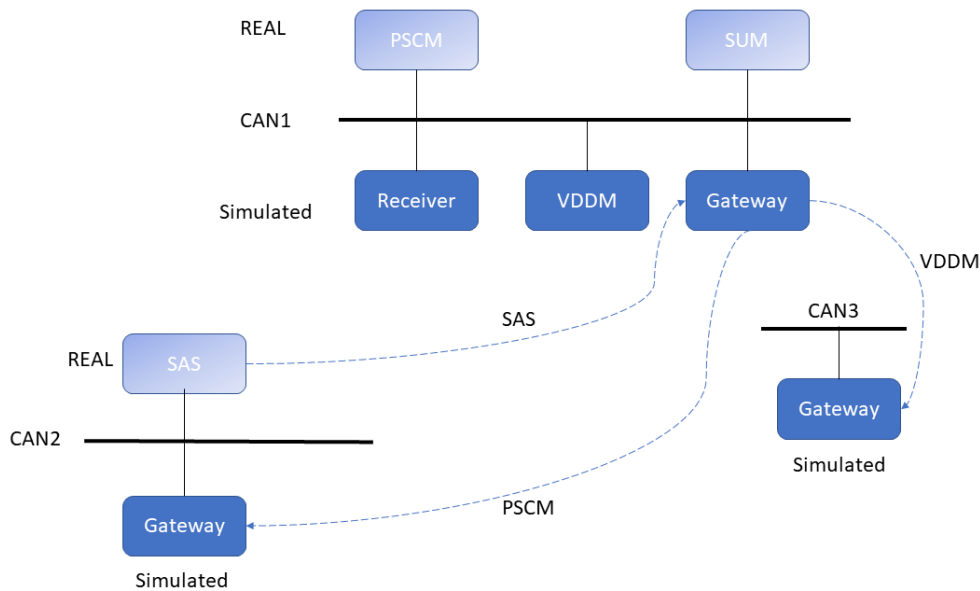


Figure 3.4: CANoe configuration network setup with the breakout cable.

Figure 3.5 below demonstrates the connection setup for the rig when the new cabling kit is installed. KL15 is the wire used for ignition. Which is sending a signal to PSCM to 'start up'. PSCM is built inside of the power steering motor but its socket is located at the outside.

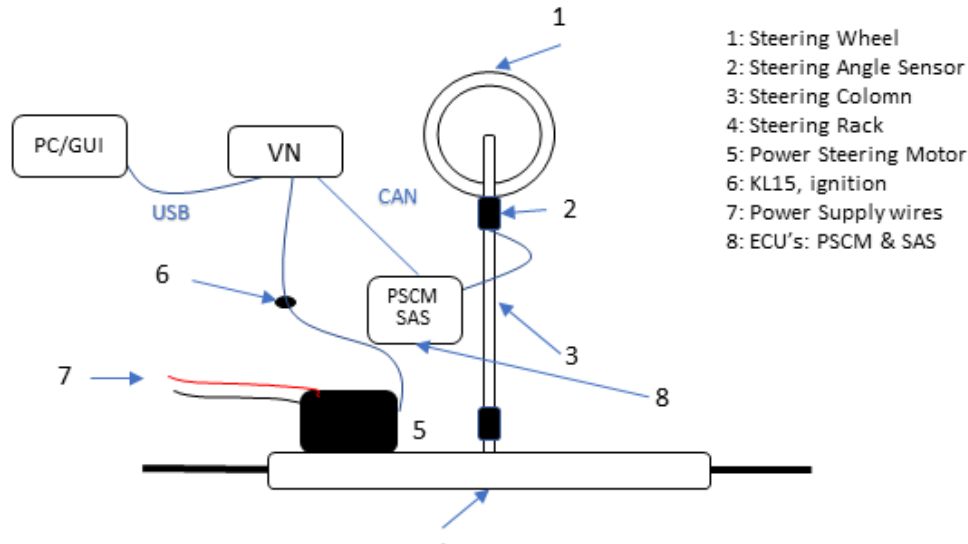


Figure 3.5: Connection between rig, VN8911 and computer with breakout cabling kit

3.1.3 Remote Steering Wheel

For remote steering to be possible an external steering wheel has to be used. In this case it is a Thrustmaster T300 RS that is originally designed for gaming. It has its own software that offers the possibility to simulate feedback in terms of increasing the torque when turning. In this case feedback from the car is not sent out to the remote steering wheel and simulating the feedback is needed.

3.1.4 GoPro

To be able to steer remotely a view of what is ahead of the vehicle maneuvered, is needed. For this a camera that can live stream in different perspective as wide and linear view is needed. The camera should also be easy to mount in the front of the car and water resistant. To fulfill all these needs a GoPro Hero 8 was chosen.

3.1.5 Additional Hardware

Beyond all the necessary hardware for steering, a few more components is needed to be used. All of the electronics used need power supplies in some sort. Therefore, converters and distributors are inserted when installing everything inside of the car. Every item used is stated below.

Power supplies

- 12 V computer charger.
- 12 V VN-power supplier.
- 230V to 12 V converter.
- regular computer charger.
- Socket for 230 V to power remote steering wheel was already installed.
- 12 distributor.

Wires and converters

- Ethernet to USB converter.
- GoPro cable.
- USB distributor.

3.1.6 Car Setup

Once all hardware is gathered and configured to work properly with each other and the break out cabling kit is installed. Everything needs to be connected to each other. Figure 3.6 below displays connection of the complete system inserted and connected inside of the car. The VN-module runs the same CANoe configuration used to control the rig after break out cabling kit is installed.

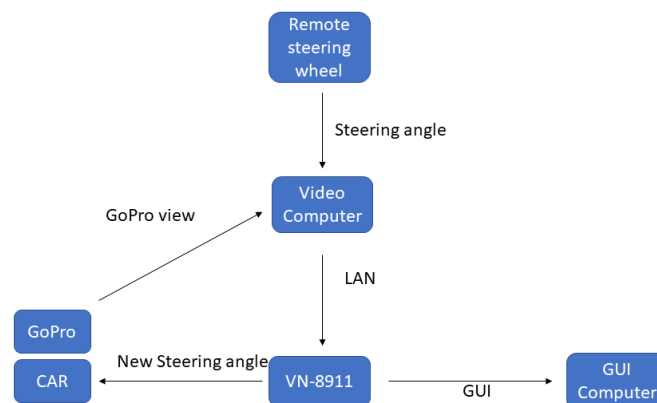


Figure 3.6: Displays the setup for steering a car

3.2 Software implementations

The following chapter of the thesis presents the software developed during this thesis from the different tests performed with the applied implementations.

3.2.1 Reading the steering angle

The first thing that is required to be solved to enable remote steering of the XC60, is to generate a steering angle value remotely from an alternative hardware. The Thrustmaster T300 RS steering wheel was used for this purpose. The Thrustmaster steering wheel is mainly designed for gaming purposes and not for implementations on a real physical car. For the purpose of this thesis, a gaming steering wheel was the perfect hardware to utilize. It is designed to provide an input steering angle value for a chosen computer application and to gain access to this value. The open-source Python programming language library Pygame was utilized for this purpose. By having access to the functions in the joystick module of Pygame, a script that reads the input of the Thrustmaster T300 RS can be written. With the following lines of code, and by implementing the library “import pygame” in the script header, the functions are enabled and ready to be used. The complete code exists in appendix B.

```

14 import pygame
15
16     pygame.init()                # Initialize the Pygame library.
17     pygame.joystick.init()        # Initialize the joystick module.
18     horizontal_axis = 0           # The input value of the Thrustmaster T300 RS
19     x_coordinate = 0              # The x-axis coordinate that are to be sent

```

Figure 3.7: Code describing the initialization of the Pygame library and the joystick module.

In figure 3.7, the system variable named *horizontal_axis* is the receiving variable of the input steering angle value from the Remote steering wheel. The system variable named *x-coordinate* is the modified variable with the purpose of being sent over UDP. The following step is to initialize the Thrustmaster as a joystick to enable data transmission between the hardware and the python script. Figure 3.8 shows that on row 22 the number of joysticks initialized are counted. If the number of joysticks is zero, the program will stop running. Otherwise, if the value is not zero, the program will bind the Thrustmaster T300 RS to the first slot of joysticks in the Pygame joystick module and initialize it on row 29 and 30.

```

21 # Count the joysticks the computer has
22 joystick_count = pygame.joystick.get_count()
23 print("Number of joysticks found", joystick_count)
24 if joystick_count == 0:
25     # No joysticks were found!
26     print("Error, no joysticks has been identified.")
27 else:
28     # Use joystick #0 and initialize it
29     Thrustmaster_T300_RS = pygame.joystick.Joystick(0)
30     Thrustmaster_T300_RS.init()
31     print("found a joystick")
32
33 is_init = pygame.joystick.get_init()
34 print(is_init)

```

returns true if a joystick is initialized, false otherwise

Figure 3.8: Code describing the counting and initialization of the connected joystick devices.

The internal steering angle value of the Thrustmaster will now be possible to attain as an input to the Python program when the initialization of the joystick device has been completed. To achieve this, the function *Thrustmaster_T300_RS.get_axis(0)* is implemented on row 62 in the figure 3.8. The function reads the value of the steering wheel angle as a float value and overwrites the value of the system variable *horizontal_axis*. The value is interpreted as a Gamepad joystick with a horizontal value which converts the angular value from the Thrustmaster steering device to a horizontal value between the interval of -1 and +1.

To design the python program to produce a continuous steering angle value to transfer to the VN8911, an indefinite loop is needed. This is achieved by the usage of a while loop with a terminating parameter if something would go wrong and an emergency stop is required. As shown in figure 3.9 The value from the variable *horizontal_axis* is then transferred to the *x_coordinate* variable. To allow the Volvo XC60 internal software to receive the data sent from the Python program, the UDP package is required to be a string datatype variable and consist of a precise size of 16 bytes. The programming operation to converting a float value to a string is done by a cast function *str(x_coordinate)* on row 71. The Thrustmaster generates a varying output size. Therefore, every data package is required to be filled up with zeros from the right side of the *x_coordinate value*.

The test rig's steering wheel takes the opposite value for a right-hand turn and vice versa for a left-hand turn. The *x_coordinate* value must then change sign to achieve a true synergy between the remote steering wheel and the test rig's physical one. As shown on figure 3.10, an if statement was utilized for this purpose.


```

45 while not done:
46     #Skapa en timer-interrupt med en timer, IRQ:
47     start = time.perf_counter()
48     time.sleep(0.010000 - ((time.time() - InitiationTime) % 0.010000))
49     irq_timedelay_ms()
50
51     i = 0
52     j = 16
53     x_coordinate = 0
54     for event in pygame.event.get():
55         if event.type == pygame.QUIT:
56             done = True
57
58     # As long as there is a joystick
59     if joystick_count != 0:
60         # This gets the position of the axis on the game controller
61         # It returns a number between -1.0 and +1.0
62         horizontal_axis = Thrustmaster.T300.R5.get.axis(0)
63         x_coordinate = x_coordinate + horizontal_axis
64         irq_timedelay_ms()
65         if keyboard.is_pressed('q'):
66             done = True

```

Figure 3.9: Code describing the method of how the program attains the input value of the remote steering wheel.

```

68 if x_coordinate < 0:
69     x_coordinate = -x_coordinate
70     x_coordinate_string = str(x_coordinate)
71     x_coordinate_string = x_coordinate_string.zfill(16)[1:]
72     x_coordinate_string = bytes(x_coordinate_string, 'utf-8')
73     sock.sendto(x_coordinate_string, server_address)
74     print(x_coordinate_string)
75
76 elif x_coordinate >= 0:
77     x_coordinate = -x_coordinate
78     x_coordinate_string = str(x_coordinate)
79     x_coordinate_string = x_coordinate_string.zfill(16)[1:]
80     x_coordinate_string = bytes(x_coordinate_string, 'utf-8')
81     sock.sendto(x_coordinate_string, server_address)
82     print(x_coordinate_string)

```

Figure 3.10: Code describing the manipulation of the steering angle value in Python, to a value susceptible for CAPL.

3.2.2 Sending the value over UDP

The second thing that must be solved is to enable wireless data transportation over Wi-Fi between a computer and the VN8911 module. This requires both a programming script implementing the sending part of UDP written in Python and a script of the receiving part written in CAPL. The CANoe software is running on the VN8911 module which then executes the CAPL receiving node as shown in figure 3.11. The VN8911 is running in distributed mode instead of standalone mode. By running in distributed mode, the VN8911 module is required to have a graphical user interface (GUI) operated on an external GUI device. This in the form of another laptop.

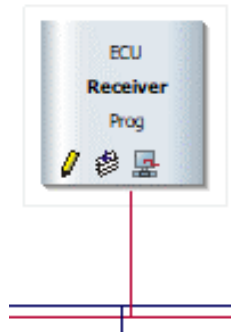


Figure 3.11: The receiving node from the simulation setup in CANoe.

3.2.2.1 UDP - Python

The sending part of the UDP is implemented in the Python programming script. This is written in the same program as where the input value from the Thrustmaster T300 RS is generated and modified. To enable data transmission using UDP protocol, a datagram socket must be declared. This is done by the declaration of the variable *sock* with the function initialization *socket.socket(socket.AF_INET, socket.SOCK_DGRAM)* as shown in figure 3.12. The first parameter of the function informs the created socket the ipv4 family address is used. The second parameter declares the created socket as a datagram socket. With this done, the UDP sending client is declared and a UDP server is created. To specify a mutual server address and port between the sending and receiving client, a declaration of the destination port and the source port is done on row 40.

```

26 #Creates UDP sockets
27 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)           # internet, UDP
28
29 #Connects the socket to a port and an IP-address
40 server_address = ('10.246.39.162', 40001)                         # specifies IP-address for port and server
41
42 print('UDP server address {}, port {}'.format(*server_address))

```

Figure 3.12: Code describing the declaration of datagram socket following the UDP structure.

The *sendto()* function sends a request to the UDP server to enable the data transportation between the client and the server. As shown in figure 3.10, this function is being used on row 74 and 83. The first input parameter of the function contains the modified steering angle value, as earlier explained, and the second parameter is the destination server address where the data package is delivered to. This enables the possibility for the requesting client to receive the data package being sent.

3.2.2.2 UDP - CAPL

To transfer the data package to the VN8911 module over UDP communication, a receiving programming script is needed. The VN8911 uses CANoe for this purpose, the same internal software implemented in the Volvo XC60. The receiving UDP client of the CANoe simulation setup is the receiving node that is shown in the figure 3.11. The internal programming language used in the CANoe software is CAPL, a programming language very similar to the C programming language. The receiving node contains the program written in CAPL that receives the data package requested. To start the data transportation between the UDP server and the requesting client, a declaration of the server address and the source port is needed. This is achieved by code line 20 where the UDP socket is being declared with the same IP-address and port as the UDP server. To initialize the communication between the UDP server and the receiving client, the built in function *ReceiveFrom()* from the socket communication library is used to enable the transport layer protocol data transmission communication. The input parameters of *ReceiveFrom()* gives the CAPL program a buffer capable of taking in data packages of size 16 bytes.

```

11 // on myvar_update myvar::Receive::Open
12 {
13     // on open button down...
14     if (!this == 1)
15     {
16         // Open an UDP socket. As source address 0.0.0.0 is used, this means that
17         // the configure address of the TCP/IP stack is used. See TCP/IP stack
18         // configuration dialog in the simulation setup
19         // On UDP port 40001 we want to receive UDP packets.
20         gSocket = UdpSocket::Open( IP_Endpoint(169.254.202.224:40001) ); //Changed from 10.246.39.142:40001 on 2021-04-19 (11:32), 169.254.202.224:40001 IAN
21
22         if (IpGetLastError() != 0)
23         {
24             // if UdpSocket::Open fails, we print a message to the write window
25             write( "<NAME_FILE_NAME> UdpSocket::Open failed with result %d", IpGetLastError() );
26             return;
27         }
28         // To receive data with the created socket, we have to call ReceiveFrom.
29         gSocket.ReceiveFrom( gRxBuffer, sizeof(gRxBuffer) );
30
31         // if ReceiveFrom does not immediately copy to gRxBuffer, it returns 997 to
32         // indicate it will call the callback function OnUdpReceiveFrom later.
33         if (gSocket.GetLastError() != 0) && (gSocket.GetLastError() != 997)
34         {
35             char errorString[100];
36             // if ReceiveFrom fails, we print a message to the write window
37             gSocket.GetLastErrorAsString( errorString, sizeof(errorString) );
38             write( "<NAME_FILE_NAME> ReceiveFrom failed with result %d (%s)", IpGetLastError(), errorString );
39         }
40
41         // update panel controls state
42         EnableControl( "Receiver", "OpenButton", 0 );
43         EnableControl( "Receiver", "CloseButton", 1 );
44     }
45 }

```

Figure 3.13: Code describing the implementation of *OPEN* and *CLOSE* button on the test page.

Previously mentioned in the report, the data package sent over UDP is modified to be a string of 16 bytes. In order to send the message over the CAN bus of CANoe, the string of 16 bytes is required to change data type according to a certain bit setup. The data type required is a double number where the bytes of the number need to be allocated as follows: Blank space - Sign- Digits - . Digits. As seen in figure 3.14, this allocation is made by the typecast function *atodbl*. The modified value is then transferred to the desired variable *emphParkAssiPinionAgReq* and is thereafter sent to the CAN bus.

```

69 void OnUDPReceiveFrom (ushort socket, long result, ipEndpoint remoteEndpoint, char buffer[], ushort size)
70 {
71     write("received package"); // Print if package is received
72     if (result == 0)
73     {
74         char endpointString[30];
75         remoteEndpoint.ToString(endpointString);
76         // set syvars to display receive data in Receiver panel
77         sysSetVariableString( sysvar::Receiver::RxAddress, endpointString );
78         sysSetVariableString( sysvar::Receiver::RxText, buffer );
79
80         if (abs(SCanChassisCANh::PSCM::PSCMChasFr02::SteerWhlTq) > 3) // IF FAP is in use and wheel gets Tq over 2 Nm, CAN 1
81             //if(abs(SCAN2_Networks::PSCM::PSCMChasFr02::SteerWhlTq) > 2)
82             {
83                 SCanChassisCANh::VDCM::VDCMChasFr07::LatCtrlModReqSafe = 0; // change from ParkAssist to regular mode , CAN 1
84                 //SCAN2_Networks::VDCM::VDCMChasFr07::LatCtrlModReqSafe = 0;
85             }
86         // added 2 * 2021-03-25
87         SteeringAngle_double_old = SteeringAngle_double;
88         SteeringAngle_double = (SteeringRatio*atodbl(buffer)); // Converts the string buffer into a double number
89         if( abs(SteeringAngle_double-SteeringAngle_double_old) > 0.2)
90         {
91             SteeringAngle_double = SteeringAngle_double_old;
92         }
93         SteeringAngle_double = (SteeringRatio*atodbl(buffer));
94         IFParkAssiFinionAgReq = SteeringAngle_double; // writes the steering angel when Park Assist active
95
96         //IFKeyFinionAgReq = SteeringAngle_double; // writes the steering angel when trafficjam active
97         //write("sa", SteeringAngle_double); // Prints steering angel in write window
98     }
99 }
100
101 // To receive more data, we have to call ReceiveFrom again.
102 gSocket.ReceiveFrom( gRxBuffer, elcount(gRxBuffer) );
103 }

```

Figure 3.14: Code describing the final modification before sending the steering angle value to the CAN bus.

3.2.3 The Rig and the whole setup

By being able to generate the input value of the remote steering wheel and enabling the UDP communication, the focus shifts from the individual solution to a more complete scenario. A scenario where the focus now lies on the objective to make the physical steering wheel of the rig move accordingly to the remote one. With the right hardware configuration and UDP communication, a complete simulation setup for testing the rig can be created. On the test rig, there are some physical nodes producing real values to the VN8911 module. These physical nodes are the PSCM, SAS and SUM. As mentioned in the hardware chapter 3.1, the frames from the PSCM are produced from the ECU located inside of the power steering motor. The SAS and SUM frames are coming from the steering wheel of the test rig. All other ECUs are simulated in the simulation setup, this with the purpose to simulate a real testing environment to fool the test rig to believe it is a real car. This is shown by figure 3.15 where the faded squares are the real ECUs and the remaining ones are simulated. The simulated ECUs are the missing physical components not implemented on the test rig, from the ChassisCAN Gateway. The ChassisCAN Gateway is a minority of CAN networks inside of a XC60, but it includes the ECUs needed for the purpose of this thesis.

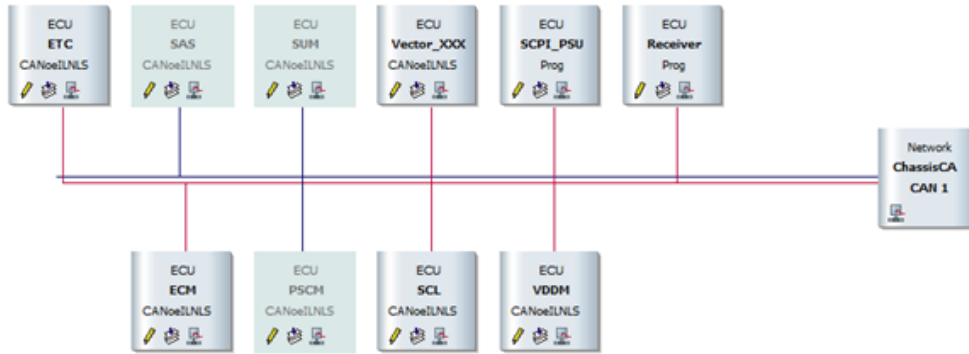


Figure 3.15: Visual representation of the created simulation setup utilized on the tests performed on the test rig.

To make the steering wheel mounted on the test rig to turn with the help of the simulation setup in figure 3.15, an understanding of what is required to make the steering wheel turn, is necessary. The XC60 is chosen to be in park assist driving interface, this is done due to two main reasons. The first reason is that the behaviour of the car is unknown while overwriting system variables and signals in normal driving interface. Therefore, this is done with the purpose to not interfere with the XC60 internal driving system. Simply put, it is a delimitation at the start of the thesis. The other reason is, while in park assist driving interface, the internal system for controlling the steering wheel will not intervene with the actual steering wheel in a way that is hindering the wheel to turn as intended. There will be no controller systems in park assist driving interface that will prevent the steering wheel of the car to move accordingly to the remote steering wheel. In other words, the steering wheel will be very accurate to the movement of the remote one. In order to do this, fooling the car that the velocity is below 5 kilometres an hour is necessary. Otherwise, the car will enter a different driving interface and the created configuration will not work as intended.

In park assist pilot interface, the received signal that instructs the steering rack to turn, is being held by the system variable *PinionSteerAG1*. This variable is one of the variables included in *PSCMChasiFr01* which is the first frame from the PSCM ECU. The value of *PinionSteerAG1* is received from the VDDM system variable *ParkAssiPinionAgReq* from the VDDM frame *VDDMChasiFr15*. As previously explained, *ParkAssiPinionAgReq* obtains its value from the UDP data transmission. But in order to change the value of *PinionSteerAG1*, the *ParkAssiPinionAgReq* must be compared with *SteerWhlAgSafe* from *SASChasFr01*. This comparison is made due to safety measures e.g., if the deviation between the two variables is too great, the value of *PinionSteerAG1* will not be overwritten. If this occurs during simulation tests on the test rig, the test rig will shut down the simulation. This is due to how the simulation setup is programmed to change the driving interface from park assist pilot interface to the regular driving interface. This is partly caused by how

the internal software of Volvo XC60 car configuration handles the comparison of the *SteerWhlAgSafe* and *ParkAssiPinionAgReq* variable.

3.2.4 LAN and Wi-Fi tests

By this stage, a complete scenario has been created and tests can be performed on the rig. Two main tests are performed. The first test was executed without the usage of the created cabling kit, explained in the hardware chapter previously. This test was made with both LAN connection and Wi-Fi connection between the VN8911 and the second computer operating the Python script. The second test was executed with the same prerequisites except this time performed with the created cabling kit implemented on the rig. Between the two tests, the configuration of the hardware differs. The IP-address will need to change accordingly to whether the test will be performed over LAN or over Wi-Fi connection. This change is also required in both the sending client Python script and the receiving client CAPL script to make the test possible to perform.

Both these tests are done using a quantitative way of method where the data from the tests will be collected and analysed. The primary goals of performed tests are to manage to steer the steering rack and so also the physical steering wheel with the remote steering wheel. But at the same time measure the different time-delays occurring over different parts of the tests. Time-delays such as the delay from the computer executing the Python program to the program implemented in the VN8911, and the differences between these time delays using LAN connection and Wi-Fi connection.

3.2.5 Test page explanation

Initialization of the simulation is done by the computer directly connected with the VN8911 over USB. The computer handles the GUI of the VN8911 with the CANoe software. The GUI allows the user to interact with the VN8911 and change between different interfaces and other possible interactions are possible, such as generating data graphs and other valuable data. As seen in figure 3.16 below, some of the important GUI interactions of the simulation setup are shown. As previously mentioned, both the tests with the car and the test rig need to operate in the park assist pilot interface. To achieve this, the VDDM is required to be in driving interface, which is shown as the *UsgModDrv*_*UsgModSts* number 13. While in driving interface, the *LatCtrlMod* can be changed to *LatCtrlMod4_PrkgAut*, which forces the car into the SAP interface.

An interactable panel where the system variables *Open* or *Close* is available is shown in figure 3.16. This panel enables the CAPL program to know if the UDP receiving socket is open and susceptible to receiving UDP packages or closed. The value of the UDP packages can be observed on the receiver panel by the system variable

RxText. Other options and interactions are also available, options such as regulating a simulated vehicle speed.

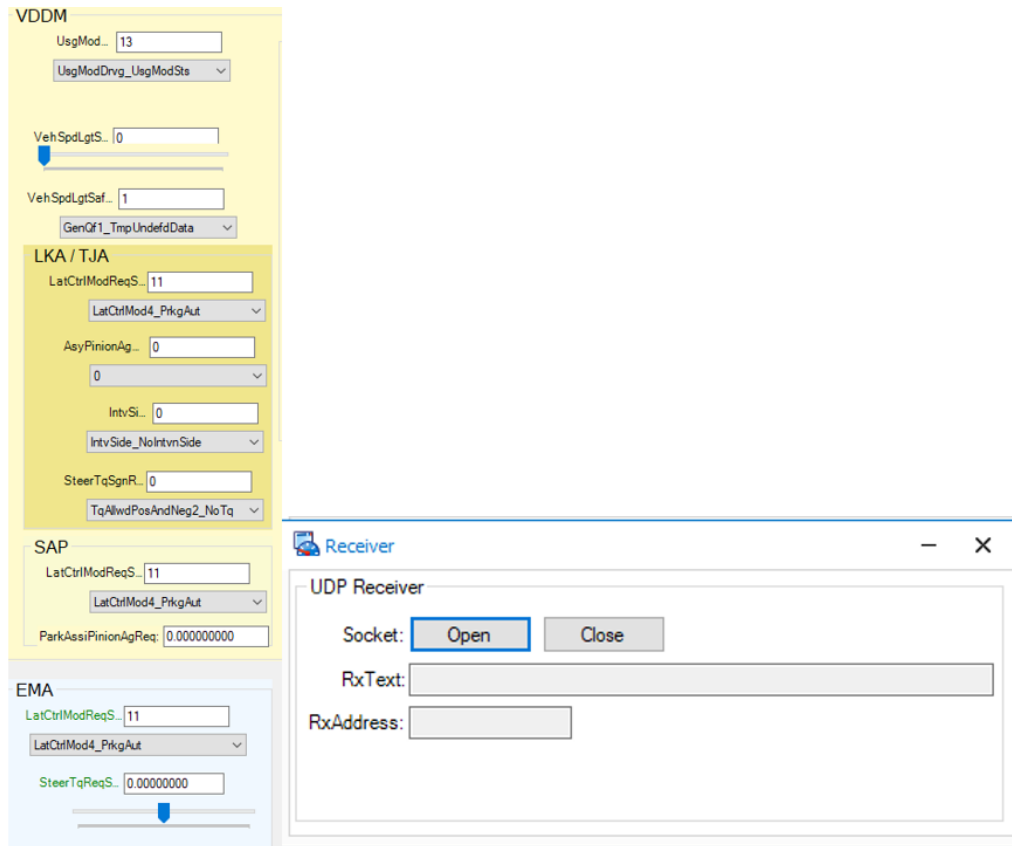


Figure 3.16: Visual GUI representation of the test page in CANoe, used for direct interaction between the car and the GUI.

3.2.6 Simulation setup for car implementation

After successfully steering the test rig remotely, the next phase of the thesis can be initiated. The next phase involves the previous implementations tested on the test rig but applied on a Volvo XC60. Therefore, the breakout cabling kit of the PSCM CAN bus network needs to be implemented and mounted on the car by Volvo mechanics. VN8911 is connected in such a way that channel 1 is connected to the CAN network from the real PSCM of the car and channel 2 and 3 is directly connected to the CAN network communicating with the rest of the car. All messages and signals sent or received to channel 1 are therefore between the PSCM and the VN8911 and the messages and signals sent or received over channel 2 and 3 are between the VN8911 and the car.

Additionally, some adjustments in the simulation setup are required. The XC60 differs a lot from the test rig and some signals and frames are needed to be allocated differently by the VN8911. As shown in figure 3.17, the majority of ECUs are faded

and are therefore real ECUs sending real values generated from the car. The VDDM ECU is the only simulated ECU required. As mentioned previously in the report, the VDDM frame *VDDMChasFr15* consist of the system variable *ParkAssiPinionAgReq* that compares its value with the corresponding variable from SAS and PSCM. The PSCM system variable *PinionSteerAg1* will therefore be overwritten and the PSCM frames can be sent through the VN8911 on to the XC60. As shown on figure 3.17 and 3.18, two additional CAN networks and a gateway ECU between the three CAN networks is needed to enable an exchange of signals between the hijacked PSCM ECU and the car that allows the comparison.



Figure 3.17: Visual representation of the created simulation setup utilized on the tests performed on the XC60.

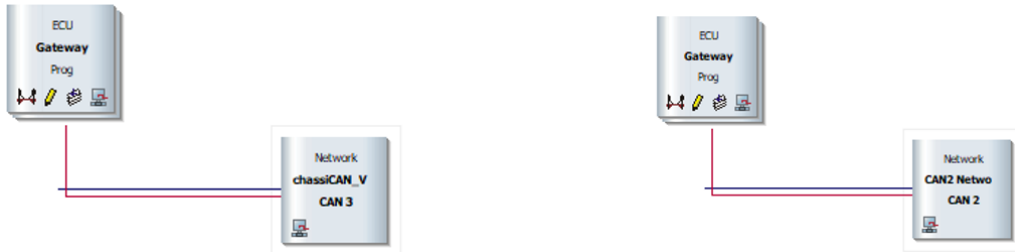


Figure 3.18: Visual representation of CAN2_Networks and ChassiCAN_VDDM CAN busses in the simulation setup.

The gateway ECU is programmed to send *PSCMChasiFr01*, *PSCMChasiFr02*, *PSCMChasiFr03* and *PSCMChasiFr04* to the CAN network *CAN2_Networks* plugged into channel 2. These frames are the modified frames including the overwritten system variable *PinionSteerAg1*. Additionally, the ECU is programmed to send the simulated frame *VDDMChasFr15* to *ChassiCAN_VDDM* over channel 2 and the SAS frame *SASChasFr01* to the PSCM through channel 1 over ChassisCANhs. The code lines written to execute the previous operations are greatly similar and an example of how *VDDMChasFr15* is sent, is shown in Fig 3.19.

The *on message* function initializes the code lines inside of its brackets by checking if the message/signals is active on the desired location. In this case, the following location was used; *ChassisCANhs::VDDM::VDDMChasFr15*. The *ChassiCANhs* is the CAN network linked to channel 1 informs the CAPL program where to check if the message *VDDM::VDDMChasFr15* is active. The message *chassiCAN_VDDM.*m*; tells the program the location where message *m* are being sent during the if statement. The if statements checks the incoming message from *on message* if the message is a receiving or transmitted signal and which CAN network it is being sent from.

```

78 on message ChassisCANhs::VDDM::VDDMChasFr15
79 {
80     message ChassiCAN_VDDM.* m;
81     //if ChassisCANhs
82     //if
83     if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received on CAN 1, to avoid duplicating frames that were transmitted by CANoe, Rx receive, tx, transmit
84     {
85         mchasi; // copy data from the received message to a new message
86         output(m); //send the new message on CAN 2
87     }
88 }

```

Figure 3.19: Code describing an example of sending Frames and signal between CAN networks programmed in the Gateway node.

3.2.7 Matlab code

To be able to analyse the data received from the tests performed, the system variables *ParkAssiPinionAgReq* and *PinionSteerAg1* were required. These variables are, as mentioned earlier in the thesis, the input and output signals regarding the steering of the car. After each test, these two variables containing all signals received by the car during the test, were saved and later used as two large arrays of complex coordinate values. The function of *tffestimate* and *mscohere* was later used in the Matlab script as shown in figure 3.20, to generate an estimated transfer function and to gain the linear relation between the two variables.

```
1
2 %Bachelor's thesis:
3 - [txy,f1] = tfestimate(ParkAssiPinionAgReq,PinionSteerAg1,[1024],[[],[],100]);
4 - [cxy,f2] = mscohere(ParkAssiPinionAgReq,PinionSteerAg1,[1024],[[],[],100]);
5 - subplot(2,1,1);
6 - plot(f1,mag2db(abs(txy)))
7 - xlabel('Frequency [Hz]'), ylabel('Magnititude [Db]')
8 - hold on
9 - grid on
10 - subplot(2,1,2);
11 - plot(f1,(180/pi)*angle(txy))
12 - xlabel('Frequency [Hz]'), ylabel('Phase [deg]')
13 - hold on
14 - grid on
15 - figure()
16 - subplot(2,1,1);
17 - plot(f2,abs(cxy))
18 - xlabel('Frequency [Hz]'), ylabel('Coherence value [0-1]')
19 - hold on
20 - grid on
21 %figure()
22 - subplot(2,1,2);
23 - plot(Time,ParkAssiPinionAgReq,'r',Time,PinionSteerAg1,'b')
24 - xlabel('Time [s]'), ylabel('Angle [rad]')
25 - grid on
```

Figure 3.20: Code describing the functions generating the data analysis from the different tests performed.

3.3 Tests and Simulations

Test and simulation chapter explains what experiment method that was used. Why it is used and what the result will narrate.

3.3.1 Sending and Receiving

As mentioned under LAN and WIFI test. It is important to determine the existing delay when sending signals over WIFI and LAN. To later on be able to set up the other experiments correctly. To measure the built-in delay a timer was implemented in Python. The timer starts when the computer that runs python is sending a value to the VN-module. The configuration loaded at the VN-Module receives the value and then sends it back immediately. When the computer that runs python receives the value the timer stops. The total time was then divided by two to get the latency of one way. When the delay for sending data over either WIFI or LAN is known. It is possible to work out the test with different time delays implemented.

3.3.2 Rig And Car Test

To create and measure how well the car's steering works when a new way of steering is applied requires accurate test methods. At first some minor tests were performed on the rig to verify if steering with the chosen solution even would be possible to implement in a car. To measure how well the actual steering angle followed the requested steering angle comparison needed to be done. In order to do so the reference signal was compared with the system's present value. In this case *ParkAssiPinion-AgReq* represents the requested value and *PinionSteerAg1* is the system's output. CANoe has the possibility to collect data from each signal and plot them. From CANoe you can export the data to Matlab where it is possible to use different built-in math functions to analyze the result.

3.3.2.1 Rig test

The rig is a powerful tool to try out the solutions before moving on to a real car. In this case the rig were not used for any further analysis in Matlab. The test was to by eye try to recognise any latency between the different steering wheels. A Final test was constructed as following. Every hardware is setup and connected to the rig together with the CANoe configuration. The steering motion back and forth started with slow movements but was successively increased. The steering movement was increased until the point where it was visible that rig's steering wheel is pointing in the wrong direction compared to the remote steering wheel.

3.3.2.2 Car Test

Experiments performed with the car were divided in to the two categories low intensity and high intensity. To better get an understating of maneuvering possibilities in different scenarios. Hardware setup and connections were the same in both cases and the exact same CANoe configuration was used.

The remote steering wheel was placed in the backseat. Above the steering wheel a computer was placed. A GoPro was mounted on the windshield and its view was displayed on the computer placed above the remote steering wheel. Figure 3.21 below is displaying the setup from the backseat. With the help of that view the test driver tried to navigate around the courses. The view was a bit delayed. It did not have the same delay as when sending the steering angle. A safety driver was located in the front seat to handle gas and brake. As well as have the possibility to overtake the steering of the car in case of emergency.



Figure 3.21: Remote steering wheel and computer that display the view placement

Figure 3.21 above shows how the remote steering wheel was mounted in the backseat and where the computer that displayed the view were placed. As can be seen the safety driver is located in the front seat to handle gas and brake.

3.3.2.2.1 Maximum Delay Before moving on to experiments devoted to low or high intensity maneuvering. It was necessary to get an understanding of how different delays regarding sending the remote steering wheels steering angle affects maneuvering. To later on be able to choose between what steering delays low intensity and high intensity experiments should be conducted with. Delays chosen during this test was related to latency's that can occur when using WiFi. But simplified due to that it will not fluctuate during the tests.

For this experiment the setup was a bit different, no camera was attached to the windshield. Driver that was maneuvering with the help of the remote steering wheel did instead look directly at the road. Which means that there were not any delays regarding view and the test did only evaluate how well the driver could maneuver when a fixed delay was applied. This experiment was performed at the circuit seen in Figure 3.22.



Figure 3.22: Figure Demonstrating test track at PV-north.

Experiment was divided into two attempts depending on velocity. Delays on sending the steering angle were in both cases first set to zero the successively increased till

it felt impossible to steer in a safe way. It was increased with 50 ms after each lap around the track. Table 3.1 illustrates the attempts.

Table 3.1: Maximum Delay

attempt	velocity	delay
1	Curve = 15 km/h, Straight = 30 km/h	0 -> impossible
2	Curve = 15 km/h, Straight = 30 km/h	0 -> impossible

3.3.2.2.2 Low Intensity An experiment to simulate driving on a country road that does not have the need of fast maneuvering. To exemplify and test driving in low intensity situations. Figure 3.23 below demonstrates the track driven. As can be seen the track is colored in blue and red. The use of different colors is due to the different velocities applied during the track.

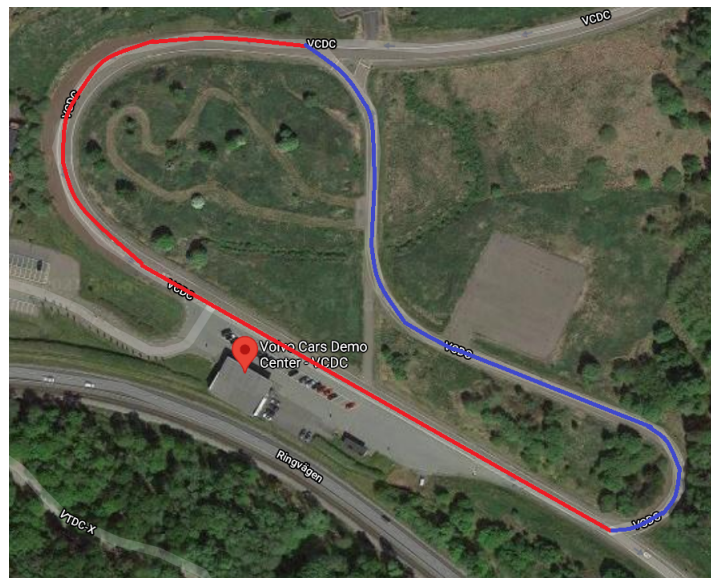


Figure 3.23: Caption

The low intensity test track can be seen above. Red represents a velocity of 40 km/h and blue 30 km/h. The test was divided into two attempts, the track was run with the same velocity changes in each attempt but different delays on sending the steering angle was used in each attempt. If the car crossed the road lines or the safety driver had to interfere, the attempt was marked as failed. In this experiment maximum delay was set to 50ms due to the risk of injury.

Table 3.2: Low intensity attempts

attempt	velocity	delay
1	depending	0 ms
2	depending	50 ms

3.3.2.2.3 High Intensity An experiment to test and simulate driving in situations when faster manoeuvring is needed. To test higher intensity a concourse was put together. Figure 3.24 below demonstrates the cone course.

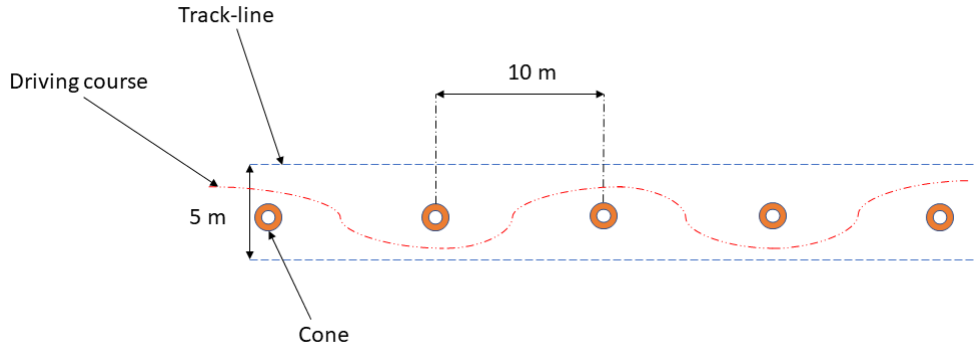


Figure 3.24: Illustrating concourse for high intensity test

As can be seen in the figure 3.24 above, the cone course had track-lines that represent the edges. If a track line was crossed with all four wheels or if a cone was hit, the attempt was marked as failed. High intensity test was divided into three different attempts.

Table 3.3: High intensity attempts

attempt	velocity	delay
1	15 km/h	0 ms
2	15 km/h	50 ms
3	15 km/h	150 ms

3.3.3 Evaluation

To better get an understanding of how well the solution performed during the tests. Evaluation of each test is important. It was done theoretical and by taking the drivers feelings regarding steering into consideration. Therefore, measuring of the reference signal *ParkAssiPinionAgReq* and the system output signal *PinionSteerAg1* were done during each test. The measured value were later exported to MATLAB for further analysis. After the high- and low intensity tests were performed a sheet was answered by the participants. It included questions related to steering to better get an understanding of how the handling felt. As well as what expectations participants had. Questions included in the sheet are stated below.

What was the experience of the deviation between screen and the remote steering wheel? 1 (Not noticeable) - 5 (Very noticeable)

How did it feel to steer while only watching a screen? When not the same field of view was possible.

1 (Not noticeable) - 5 (Very noticeable)

Question 3: How was the feeling of turning the car with the remote steering wheel with its settings and moment of inertia? Compared with the actual one.

1 (Not Strange) - 5 (Very Strange)

What were the expectations before the test? Knowing steering will be delayed.

1(Not Nervous) - (Very nervous)

How noticeable was the delay when maneuvering?

1(Not noticeable at all) - 5(Impossible to maneuver)

4

Results

The following chapter of the thesis presents the results achieved from the different tests performed with the applied implementations.

4.1 Latency tests of LAN and Wi-Fi

This thesis is based on evaluating the possibilities of steering a Volvo XC60 remotely over Wi-Fi. One of the areas of greatest importance is therefore the evaluation of the different latencies occurring while using the implementation developed during this thesis. Due to the limitations of the thesis, a simulated time delay in the python program has been utilized to measure the extreme latencies appendix C. By testing these boundaries of when the car gets undrivable according to the tests performed, one can judge the possibilities of remote steering. But these simulated latencies applied to the program itself, are not the only existing latencies. There is also the latency of the UDP data transfer from the Thrustmaster R300 TS to the actual steering wheel. These latencies were achieved, as mentioned earlier in the thesis, on the test rig using the sending node implemented in the CANoe simulation setup. As seen in figure 4.1, the Wi-Fi latency is a bit larger than the LAN latency, with an average latency of 2.325 ms for a LAN connection and 3.4112 ms for a Wi-Fi connection between the computer executing the Python program and the VN8911. The main idea behind the simulated time delay is to practically evaluate the safety margin of the length of a Wi-Fi connection delay is possible.

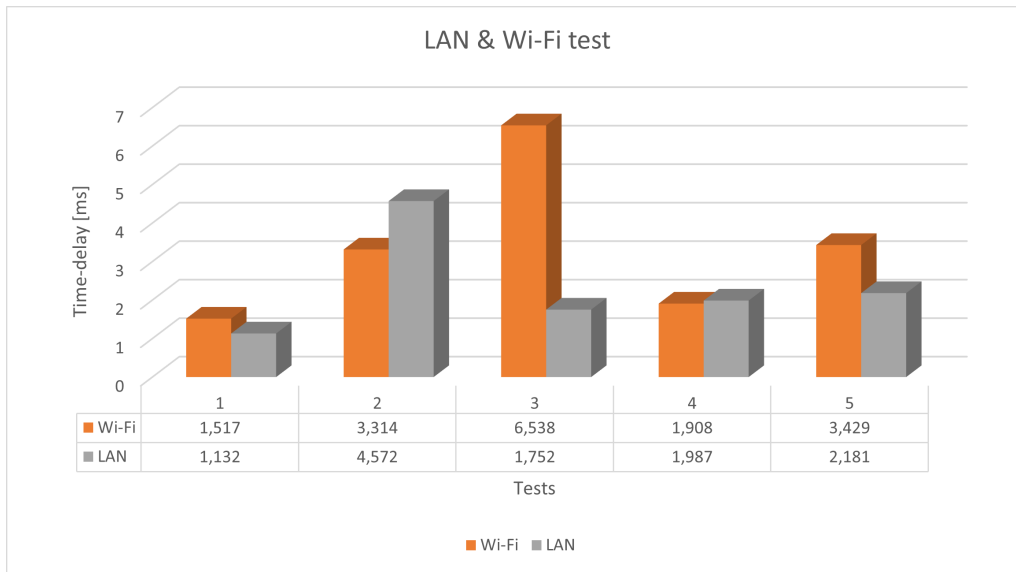


Figure 4.1: Bar graph visualizing the difference in latency between Wi-Fi and LAN performed 5 times.

This section of the chapter describes the outcomes from the signal processing analysis of the implementations on both the test rig and the car. The signal processing analysis is achieved by four graphs describing the characteristics of the differences between the input and output signals of the system in different situations. These graphs consist of Bode, coherence and timespan plots. These plots are used with the purpose of achieving data and information otherwise unattainable through humans' eyes. With the usage of the information gained, a deeper understanding of how well the hardware and software implementations on the XC60 works in practise, can be understood. Is it drivable? Is it possible to steer the car in any relevant velocity without losing control?

4.2 Vehicle test results

As previously explained in the test and simulation chapter, the two tests consist of a low intensity country road test and a high intensity cone test.

4.2.1 Low intensity test results

By observing the second plot of figure 4.2 and 4.3, the timespan of the individual tests and value of both the input variable *ParkAssiPinionAgReq* as red and output variable *PinionSteerAg1* as blue. The difference between the two signals is almost unnoticeable. By observing the difference in time for the zero-crossings between the two signals, the difference in time describes the total time-delay of the system. The time-delay including the simulated one and the overall delay between the system variable *PinionSteerAg1* is received and the steering rack starts to turn. The general size of the time-delay was approximately 126 ms with 0 ms simulated time-delay implemented. Furthermore, at a few moments during the tests performed, the graph

of the input value is clearly visible. An interesting observation can be made from figure 4.4 4.3 4.4, where two arbitrary value spikes occur from the *ParkAssiPinionAgReq* variable. The reason behind the safety precautions made during the low intensity test of only implementing the low and medium time-delays. Was due to the large and sudden value changes from the sent value of the remote steering wheel. As seen in figure 4.4, the real steering value of the car, attempts to obtain the same value through the variable *PinionSteerAg1*. Luckily, the small static sampling time of CANoe of 30 ms prevented this. The value then relatively fast tried to change the value back to a relevant one, but a small movement of the steering rack of the car still occurred. This small movement will grow larger depending on the simulated time-delay length. In higher velocities this movement could end up dangerous and therefore the safety margins applied on the tests were applied.

Additionally, the rate and steering wheel torque limiter was applied in the CAPL programming script to prevent the arbitrary value changes happening, see appendix D. The implementation of the previous was shown unsuccessful and not completely compatible with the settings of the real car. Strange behaviours from the car occurred as the result of this. Due to lack of test time, the right values or modifications of these implementations were never successfully applied during any tests.

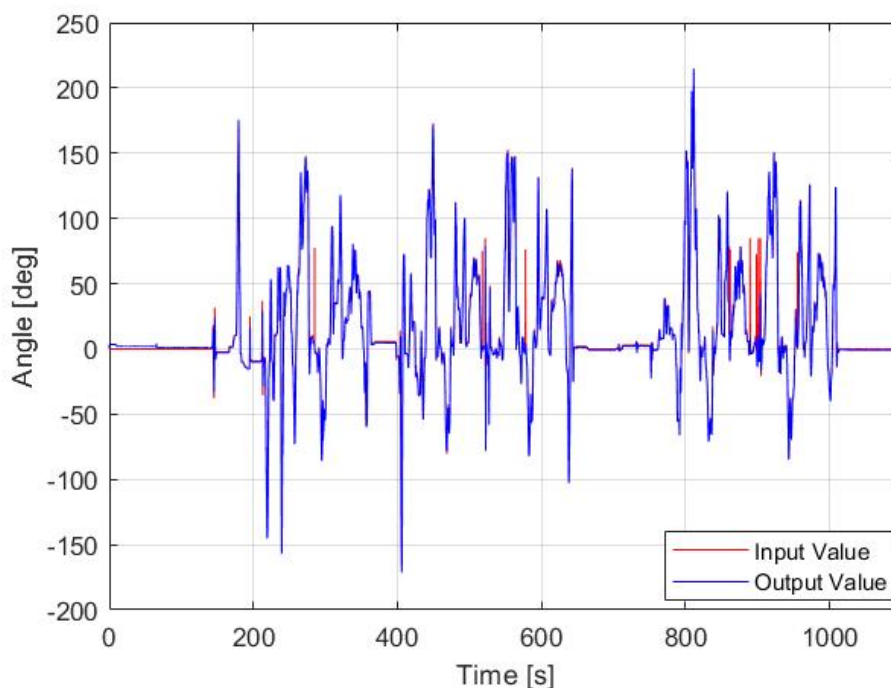


Figure 4.2: Timespan plot for 0 ms simulated time-delay during the low intensity test.

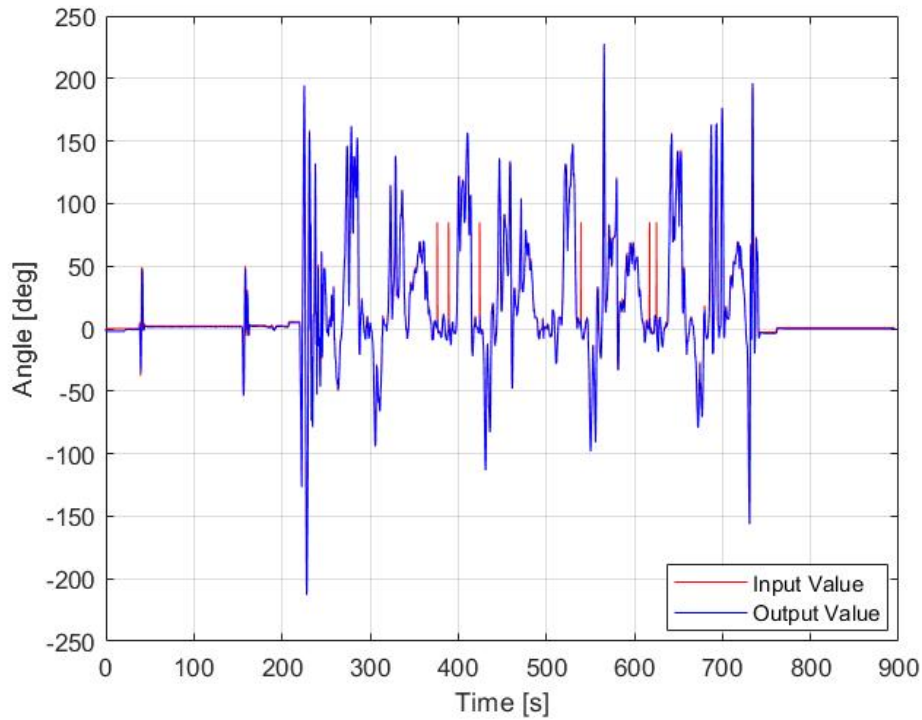


Figure 4.3: Timespan plot for 50 ms simulated time-delay during the low intensity test.

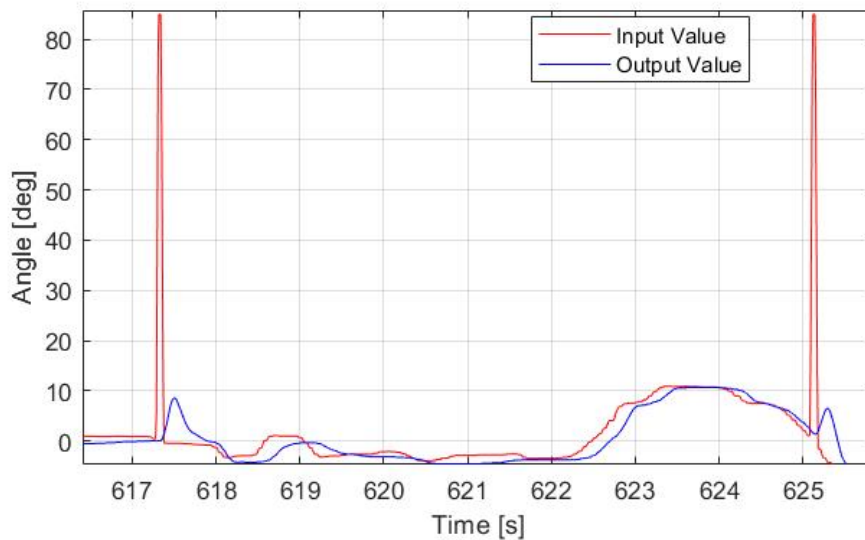


Figure 4.4: Angular difference in the timespan plot during 50 ms low intensity test, between input and output steering angle

The first three graphs regarding a more in depth signal processing analysis performed during the low intensity country road test, are the visualization of the Bode plot and the coherence between the input and the output signals. As mentioned in

chapter 3.3.2.2.2, the extreme simulated time-delay was considered too dangerous for the safety of the test. Figure 4.5 and 4.6 is providing a theoretical and visual explanation of the behaviour of the car during the low intensity test. For both the test implemented with a simulated time-delay of 0ms and 50ms added with the actual latency of the implemented system over LAN.

There is a great importance of the system having as good characteristics as possible for the relevant frequencies of steering the car, with a minor phase shift and difference in magnitude. According to [17], 3 hertz are the maximum extreme frequency that a human being can control when steering a car without losing control while keeping the car on track. Therefore, acquiring a system that has beneficial characteristics for frequencies as close to 3 hertz as possible is essential. By observing both bode plots, more noise is affecting the system after a certain frequency. The probable reason behind this behaviour was due to higher steering angle frequencies never were used which provided less accurate data. The system will therefore be perceived as signal noisy. Which is affecting the system in an inaccurate way with a low coherence value for frequencies where the signal noise is considered substantial. Considering each gain plot, the system for 0 ms time-delay consisted of more accurate data and less signal noise and gave the graph better attributes than the one regarding the 50 ms time-delay, during higher frequencies. The blue signal visualizes the average value of the transfer signal and the general direction the graph is headed towards. Otherwise explained as an easier signal to read compared to the red signal. The red signal of the bode plot visualizes the real unfiltered signal that tends to be volatile and fluctuate plenty and is therefore harder to analyse. The red signal attains these characteristics during the frequencies with a low coherence value. Additionally, the gain between the input and output signal for the system regarding 0 ms, does not fluctuate and spike as high or as much. The gain value is more consistent around the 0-decibel mark until around 1.25 hertz compared to the system regarding the larger time-delay. The blue signal concerning figure 4.6 can be observed to separate from the 0-decibel mark earlier, around 1 hertz before being more affected by signal noise. The simulated time-delay is small enough to not have a great impact on the outcome of the graphs. Therefore, the two graphs regarding the gain plot should be almost identical. As mentioned before, the probable reason behind this is rather the difference in driving and steering the car during the low intensity test than a result indicating on poor attributes for frequencies over 1 hertz.

By observing the phase plot, a dissimilarity can be recognised in the steepness in which the signals are descending after around 1 hertz. By considering the gain plot that was more stable for a longer amount of time for both simulated time-delays, the phase graphs differ accordingly. The phase margin for figure 4.5 acquired the value of 106 degrees and for figure 4.6 the value of 107 degrees. The system in theory turns unstable when the phase shift crosses the value of -180 degrees, but this comparison is not directly applicable in this occasion, due to a lot more factors are affecting the result. But the general idea is having as small phase shift possible for as high frequencies possible. This to gain desired characteristics and an easier time controlling the steering of the car. For the test performed with the simulated time-

delay of 0 ms, if the filtered estimated signal is the signal taken under consideration. The gain margin was around 16,5 decibel and reached the -180 degrees mark at 2,42 Hertz. For the test with 50 ms time-delay applied, the gain margin was around 18,7 decibel and reached the -180 degrees mark at 2,7 Hertz. This result compared to the maximum possible frequency of 3 hertz, is close enough to increase the credibility of the outcome from the performed tests [17]. To take into account, the outcome of the tests is based on the steering frequencies that actually were utilized during the tests in the XC60. The values shown in the figures would not be accurate over these frequencies. A conclusion can be made that when the unfiltered estimated phase shift is starting to be excessively affected by signal noise and gain unreasonable values, the attributes of system would no longer be attainable by observing the plotted graphs. By observing the graph when the red signal behaves like mentioned, the system can be analysed for frequencies up to around 1,5 Hertz for 0 ms simulated time-delay and 1.6 Hertz for the larger one.

The third plot in figure 4.5 and 4.6, shows the statistics of the coherence value and its linear relation between the input and output signal. For the simulated latency of 0 ms and 50 ms applied in the Python programming script. According to [18], if the value of the coherence between the steering signals and the actual value of the steering rack in a Volvo XC60 are dropping below the value of 0.9. The frequencies generating a value that low will be considered as irrelevant during the analyse of the outcome from the tests performed. The frequencies that fulfils this outcome required, while applying a simulated delay of 0 ms and 50 ms, lies within the interval of 0-81 Hz (for 0 ms) and 0-79 Hz (for 50 ms) by an observation of the third graph of figure 4.5 and 4.6. By evaluating these values from the previous statement, the only valid frequencies of turning the car from the remote steering wheel would be within these intervals, for the corresponding time-delay while striving for a high linear relation. But as mentioned earlier, the likely reason behind the low frequency intervals with a low coherence value, for both implementations. Is probably due to the lack of use of higher steering angle frequencies in a way that would gather imprecise data and therefore would have an impact on the outcome with a lower frequency range.

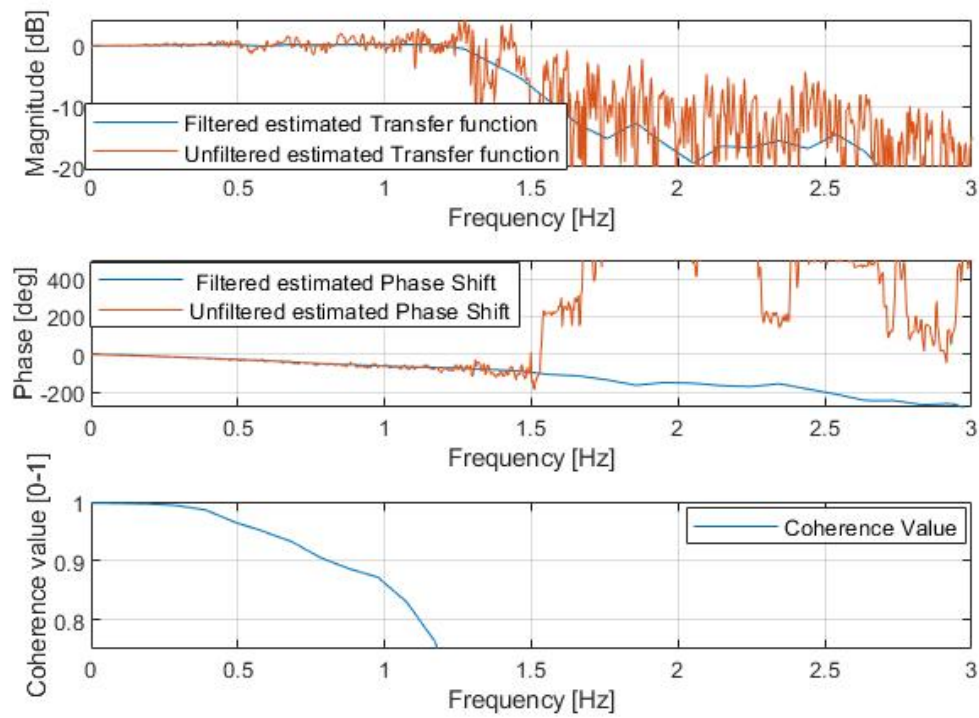


Figure 4.5: Bode plot and coherence plot with the time-delay 0 ms during the low intensity test.

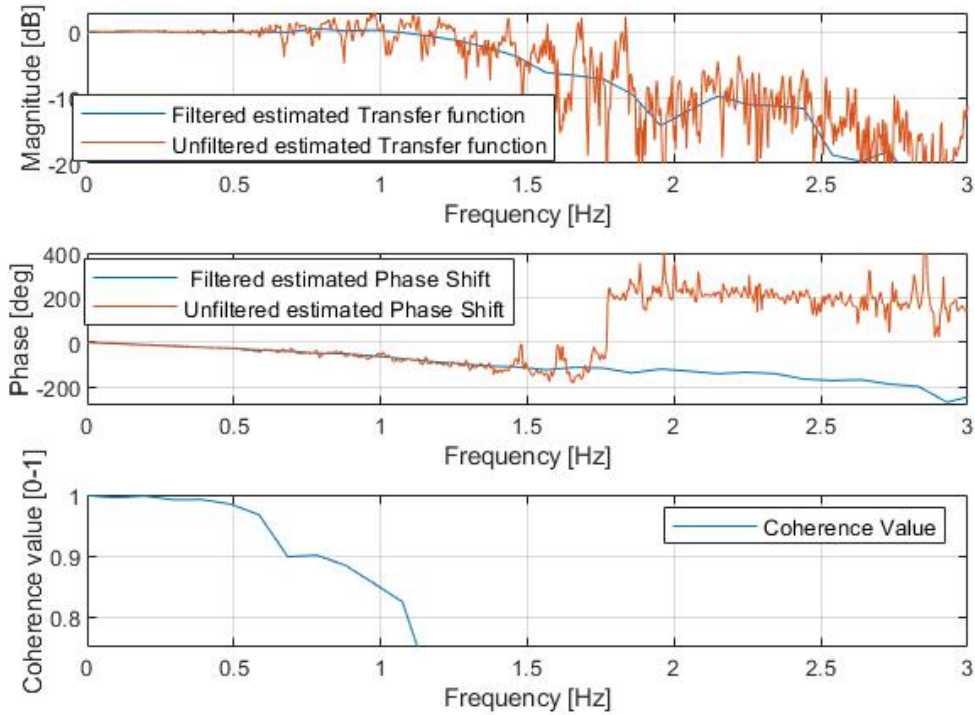


Figure 4.6: Bode plot and coherence plot with the time-delay 50 ms during the low intensity test.

4.2.2 High intensity test results

The result from the high intensity test differs significantly from the low intensity test. As mentioned in the test and simulation chapter 3.3.2.2.3, three different simulated time-delays were evaluated during the high intensity test. From figure 4.7, 4.8 and 4.9, the difference between the three time-delays were visualized in timespan plots. As mentioned for the low intensity test, the input and output signals are visualized in these figures. The zero crossing time delay was extremely similar to the low intensity one. Which is demonstrating on a system that is functioning well in both scenarios, regarding this area. A big difference in outcome of the high intensity tests, is that the arbitrary value spikes seem to occur less frequently. In fact, not once during any of the high intensity tests. The reason behind this result could potentially be that those values spikes only occur during straights or when the angle value has not changed for a small amount of time. By observing figure 4.9, a significant and clear difference between the input and output signal can be observed. The test performed with the simulated time-delay of 150 ms was a failed test where a physical overtaking of the actual steering wheel was forced by the safety driver in the driver seat of the XC60. The reason behind the value spike was due to the change of value being too great, leaving the steering rack of the car unavailable to attain the next upcoming value and leaving the park assist pilot interface to activate the driving interface *LatCtrlModRegSafe* number 0 instead of 11. This can be observed by the graph regarding where the input value containing the same value

throughout the rest of the test, due to the input from the remote steering wheel is no longer necessary.

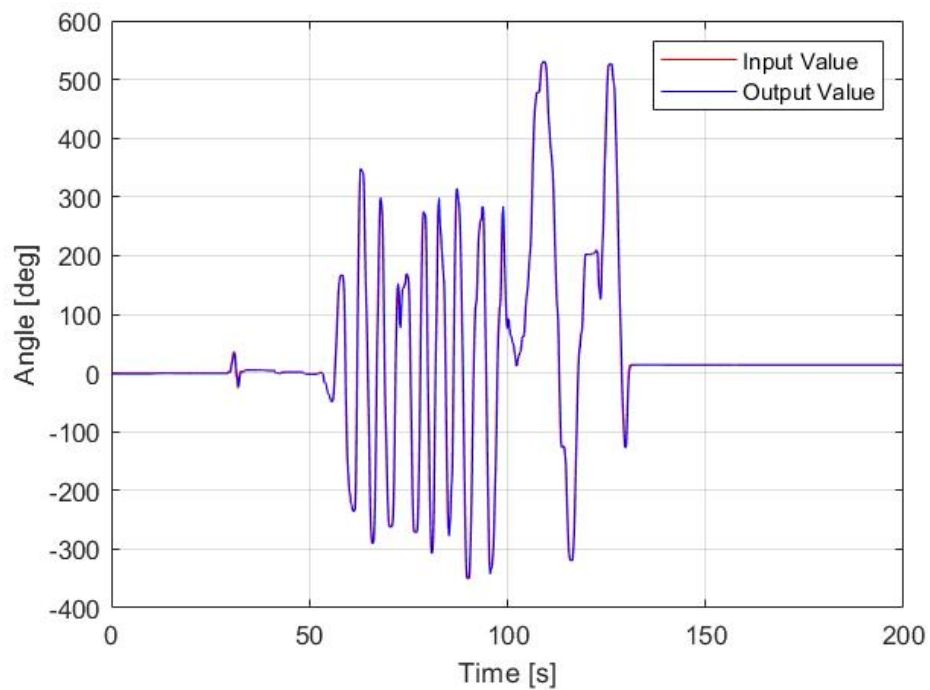


Figure 4.7: Timespan plot for 0 ms simulated time-delay during the high intensity test.

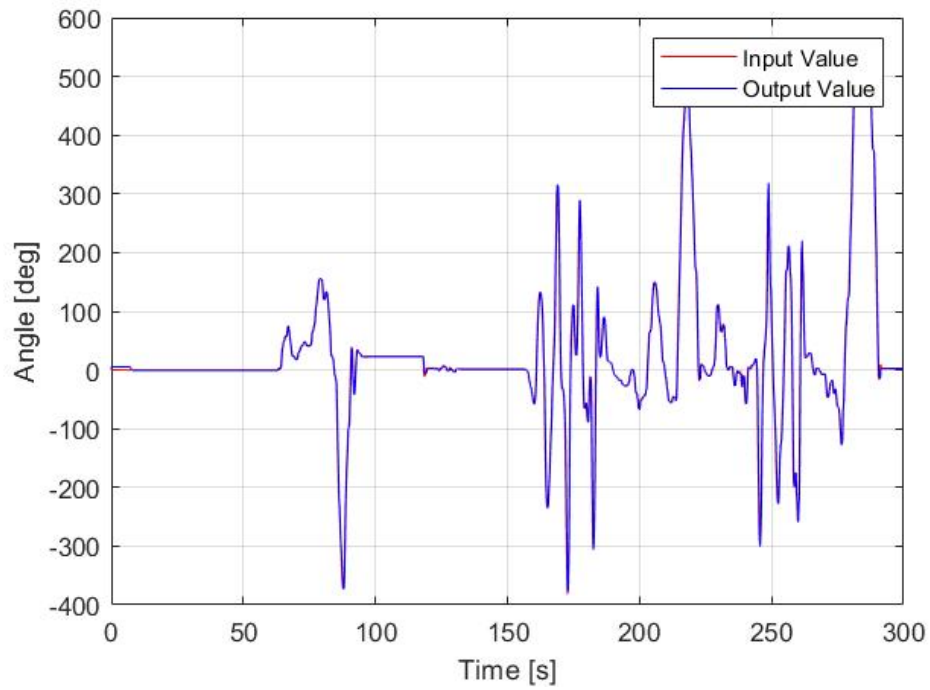


Figure 4.8: Timespan plot for 50 ms simulated time-delay during the high intensity test.

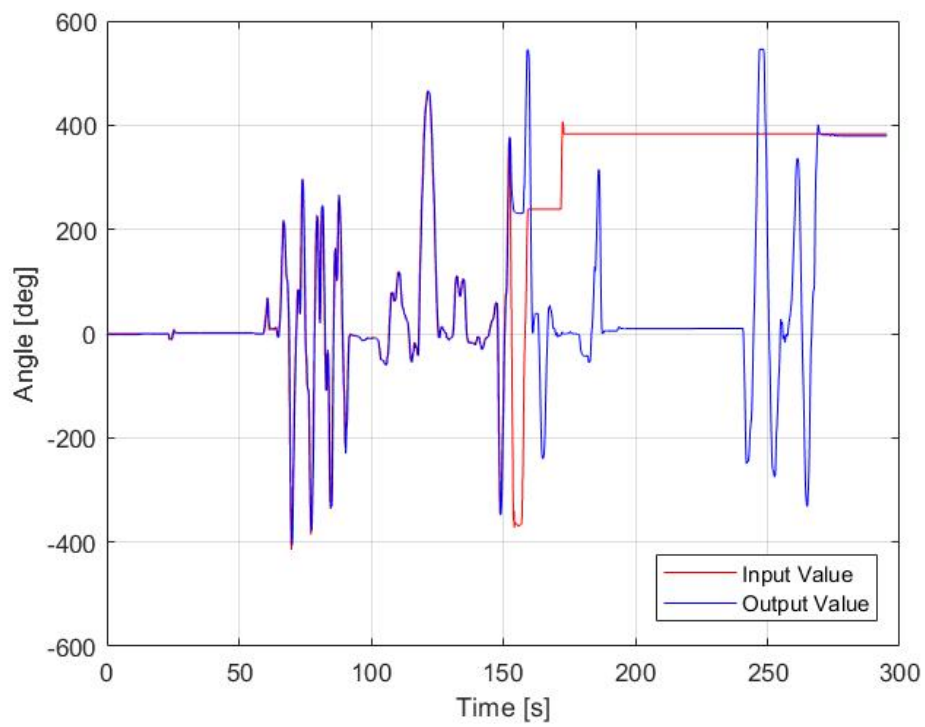


Figure 4.9: Timespan plot for 150 ms simulated time-delay during the high intensity test.

By observing figure 4.10, 4.11 and 4.12, the outcome of the gain plot during the high intensity test implemented with 0 ms time-delay, resulted in a relatively consistent system around the 0 decibel mark. Between the interval of frequencies 0-1.75 Hertz with fairly small deviations from the 0 db magnitude level. Considering an extreme value of the steering angle frequency is 3 Hertz according to [17], this outcome is positive. Compared to figure 4.11, the gain plot concerning the time-delay of 50 ms is more affected by signal noise in lower frequencies. Furthermore, the system has desirable characteristics during a relatively small interval of 0-1.2 Hertz with acceptable small deviations of the graph. By comparing the results, a clear difference can be seen. The lower the time-delay, the closer the transfer function between the input and output signal is to the gain of 1 and 0 decibel line. But during higher frequencies the outcome is the same as for the low intensity test with most likely the same reason behind this as for the low intensity test as well with less noise affecting the result.

By observing the phase plot of the bode diagram for figure 4.10 and 4.11, the phase shift is successively increasing which means the difference between the input steering angle and the output steering angle is increasing as well. The larger phase shift, the harder it gets controlling the car through the high intensity test track and execute the manoeuvres necessary. The phase margin of the system for 0 ms was 90.6 degrees and for the system with 50 ms was 134 degrees, which is a value too great for achieving desired characteristics for the implemented system. The gain margin was 3,5 decibel from the test performed with a simulated time-delay of 0 ms. For the test regarding the 50 ms time-delay, the filtered signal of the phase graph never achieved the value of -180 degrees, but by observing the figure 4.11 the unfiltered one seems to turn reach -180 degrees at around 2-2,5 Hertz. Although, the outcome regarding the mentioned frequency interval is likely to be inaccurate due to the frequency interval being too great compare to the outcome to the gain plot being affected by signal noise for significantly lower frequencies. The outcome regarding the aforementioned analyse is probably caused by the same reason as the outcome from the low intensity tests. With that being said, the reason behind the signal noise in the relevant frequency interval is because the higher frequencies are not behind performed during the tests.

As shown in figure 4.12, a more extreme delay was applied, the bode plot shows clear signs of the driver struggling to control the car over the course of the test track. By observing the graph, heavy signal noises can be noticed for all frequencies within the 0-3 hertz interval. Therefore, the graph is completely incomprehensible and irrelevant to analyse any further regarding the phase and gain plot. During the test, the car was still drivable but according to the outcome of the graph, an implementation of this delay should be avoided.

Figure 4.10, 4.11 and 4.12 demonstrates the coherence of the systems. The acceptable frequency interval for the coherence value for all three systems analysed are: for 0 ms the interval was 0-0.93 Hertz, for 50 ms the interval was 0-0.89 Hertz and for 150 ms the coherence value never reaches an desirable value of 0.9. The outcome shows that the longer delay, the more signal noise there is affecting the linearity of

4. Results

the system. For safety reasons and drivability, the extreme delay turned out to be too large of a delay to gain any valuable information.

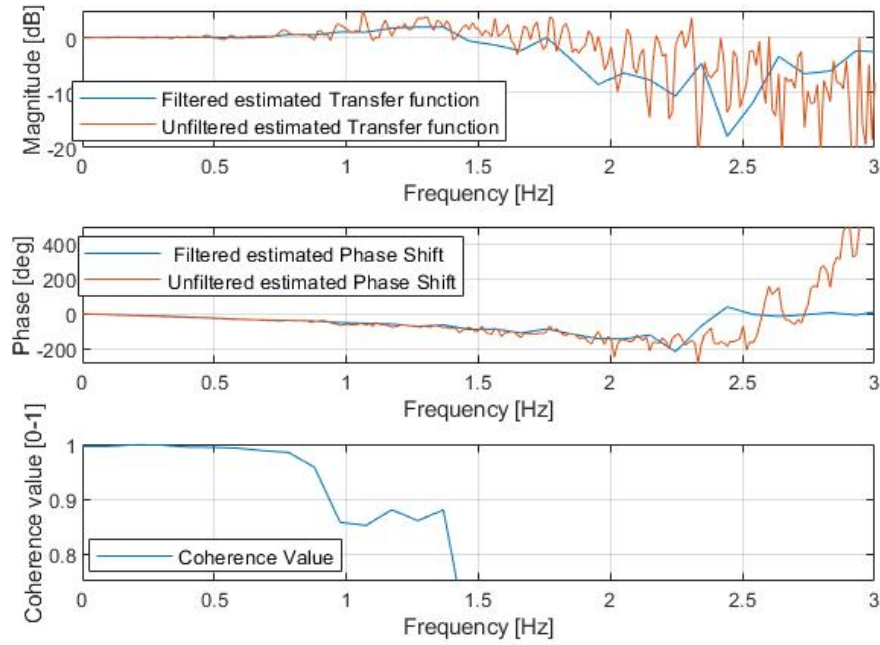


Figure 4.10: Bode plot and coherence plot with the time-delay 0 ms during the high intensity test.

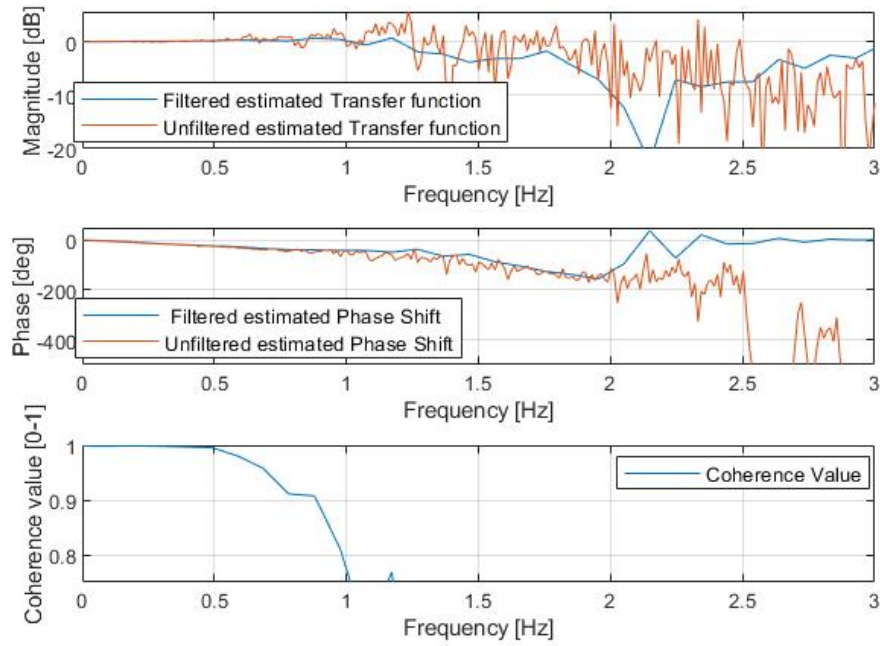


Figure 4.11: Bode plot and coherence plot with the time-delay 50 ms during the high intensity test.

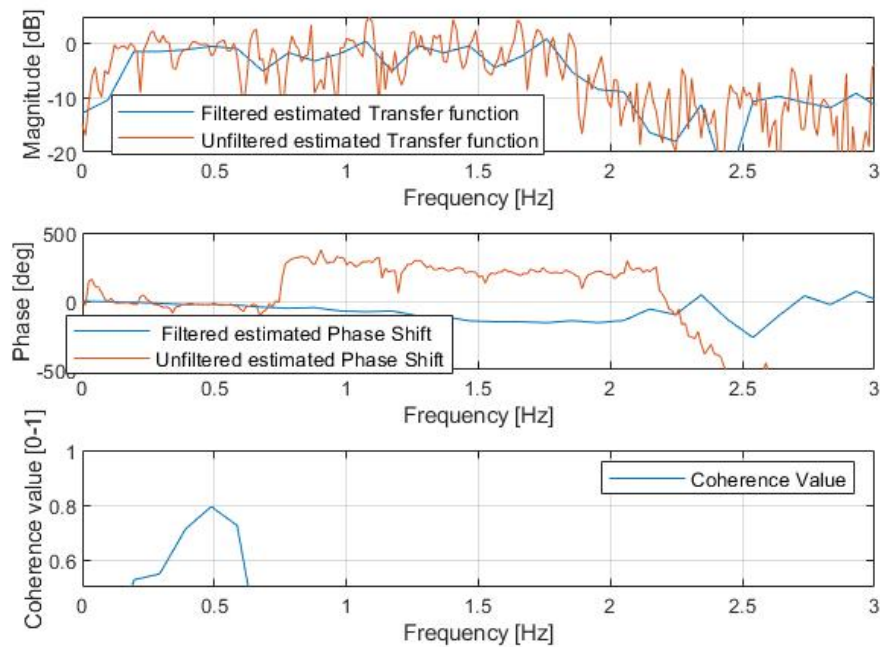


Figure 4.12: Bode plot and coherence plot with the time-delay 150 ms during the high intensity test.

4.3 Test Sheet and Inaccuracies.

After each test the participants answered a sheet with questions regarding steering characteristics. Due to the lack of time, only a small amount of people have answered. In addition to the sheet, test drivers overall feelings were taken in account to better get an understanding of how the driver perceives the changes of delay time

4.3.1 Inaccuracies noticed

As described earlier, large arbitrary angle values sent to PSCM resulted in the car to turn without any notice. It would usually emerge on a straight when the steering value has remained unchanged for a while. How powerful of a turn it resulted in was related to velocity and what steering delay was used. Greater steering delay resulted in more powerful sudden turns. Probably because the time between the arbitrary value and the correct value was greater. Furthermore, higher velocity would result in the turn being perceived as stronger.

The large arbitrary angle value did in some cases result in that the safety driver had to interfere and take control of the steering. Once the safety driver had taken over the steering, the car changed the driving interface to a safety mode. Once the car was in safety mode it signaled that city safety needed support. It resulted in difficulties for the car to regain correct settings and interface needed for remote steering to be possible. It was solved by disconnecting the modified cabling kit and use it as an extension to drive around the course steering with the traditional steering wheel. The message about city safety then disappeared and it was possible to reconnect the setup and choose correct settings and interfaces.

4.3.2 General thoughts

Regardless which test that was performed, some general thoughts were collected and it is summarised below. Steering a car when the field of view was limited to only a screen is difficult as well as it gave the test drivers a strange feeling of not having the possibility to look around. Even if the camera had a linear view, distance assessment was hard. It resulted in turns being taken in the last minute regardless test. But it should also be mentioned that the test driver got used to the condition and could better adapt to the environment during the tests.

4.3.3 Evaluation of Maximum Delay

As explained under the test and simulations chapter a test to find the maximum delay to where it would no longer be possible to steer the car was conducted. It was divided into several attempts. During this test no camera was used which meant that delays regarding view were not taken into account. This test was conducted to later on be able to choose reasonable delays when performing the high and low intensity tests. The focus was not to determine how well steering works in different situations. Delays in the range of 50 – 350 ms were examined. Test drivers comments

on the examined delays can be read in appendix A.

4.3.4 Evaluation sheet

Sheet contained five questions that were to be ranked from one to five. Ranking value is an average of the two participants answers.

Question 1: What was the experience of the deviation between screen and remote steering wheel?

1 (Not noticeable) - 5 (Very noticeable)

Table 4.1: Result for each test, Question 1

What kind of test	Delay [ms]	Ranking Value
Low	0	4
Low	50	4
High	0	2
High	50	2
High	150	4

Question 2: How did it feel to steer while only watching a screen? When not the same field of view was possible.

1 (As usual) - 5 (Impossible)

Table 4.2: Result for each test, Question 2

What kind of test	Delay [ms]	Ranking Value
Low	0	3
Low	50	3
High	0	2
High	50	2
High	150	2

Question 3: How was the feeling of turning the car with the remote steering wheel with its settings and moment of inertia? Compared with the actual one.

1 (Not Strange) - 5 (Very Strange)

Table 4.3: Result for each test, Question 3

What kind of test	Delay [ms]	Ranking Value
Low	0	4
Low	50	4
High	0	4
High	50	4
High	150	4

Question 4: What were the expectations before the test? Knowing steering will be delayed.

1 (Not Nervously) - 5 (Very nervous)

Table 4.4: Result for each test, Question 4

What kind of test	Delay [ms]	Ranking Value
Low	0	1
Low	50	2
High	0	1
High	50	1
High	150	2

Question 5: How noticeable was the delay when manoeuvring?

1 (Not noticeable) - 5 (Very noticeable)

Table 4.5: Result for each test, Question 5

What kind of test	Delay [ms]	Ranking Value
Low	0	1
Low	50	1
High	0	1
High	50	1
High	150	4

4.3.5 Further Thoughts Regarding Tests

Low Intensity test

- 0 ms

The field of view is the most challenging part as well as the delay regarding the live stream. As mentioned in 2.4 the delay regarding the live stream is 500ms. To better get an understanding of how it affects. It can be said that if the car travels at 40 km/h it will travel a distance of 5,5 m before the view on the screen is updated. Which resulted in difficulties on keeping a straight line when steering. Several small turns were needed during a long sweeping turn instead of a long smooth turn. The simulated steering delay is not yet noticeable.

- 50 ms

The simulated steering delay began to be noticeable and resulted in turning back and forth on the road instead of keeping a straight line.

High Intensity

- 0 ms

Field of view had less influence when trying to manoeuvre between the cones. The delay regarding view was less noticeable then in low intensity tests, probably because traveling at 15km/h resulted in a distance of 2,1 m had been traveled before the view updates. But it did have the effect of late turning in general between the cones. Beyond that, turning between cones worked out fine.

- 50 ms

When steering delay was set to 50 ms it did influence the manoeuvring in the fact of how smooth the turns were performed. The driver did end up performing wider turns around the cones.

- 150 ms

A delay of 150 ms did affect the manoeuvring in many ways. It was clearly noticeable and the turns became very wide. Due to the latency before actions were done by the system drivers tried to fend off during turns which resulted in hitting cones and crossing the track line. Once the track-line was crossed with all four wheels or a cone was hit, the test was marked as failed. It was clear that manoeuvring a cone course at 15 km/h with a steering delay of 150 ms can be set as a limit. Figure 4.13 below illustrates how the car traveled through the course with different steering delays.

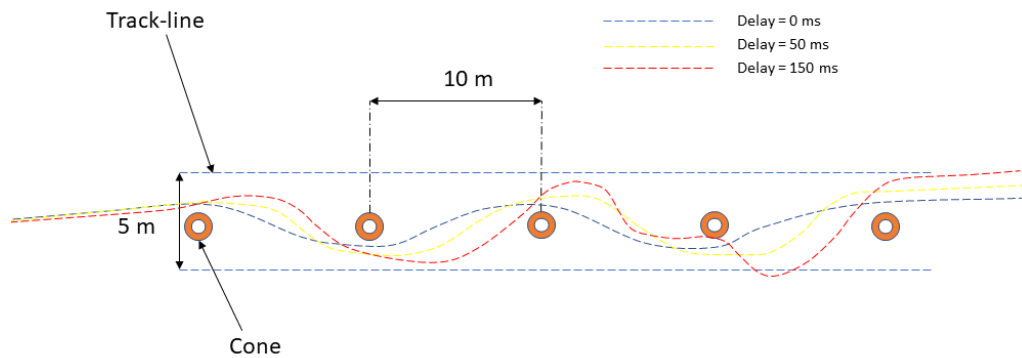


Figure 4.13: Illustrations of how the car was steered through cone track with different delays

5

Discussion

The following chapter of the thesis presents the discussion and conclusions of the results achieved during the thesis.

5.1 Overall conclusion

During this thesis, the followed the early planning of the development process in most areas. Both tests and implementations of the hardware and software setup were applied successfully to both the test rig and the car and functioned as intended. Steering the rack remotely over UDP of both the test rig and the car, was done efficiently with no larger complications. Attaining the input steering value from the remote steering wheel was done successfully and sending the value over UDP worked as intended. The implemented simulated time-delay proved to be low enough to provide a fully manageable drivability of the car during the different tests performed. The real estimated latency of sending data over Wi-Fi resulted in a far lower delay then the acceptable and manageable for the simulated one. This outcome was greatly positive due to the difference being that large. The Wi-Fi delay measured on the internal Volvo Wi-Fi network tested on the test rig setup, was only a delay of average 3.412 ms. Compared to the simulated time-delay of 50 ms. The practical marginal of the latency to achieve an acceptable performance during the test, was therefore a significant contrast.

By observing the results gained from a theoretical point of view, the aim of the thesis was achieved. The outcome of the data gained from the tests was showing great promise and indicated on a technical area full of possibilities. With the prerequisites of this thesis, there are still areas of improvement of making the car more drivable and efficient in higher frequencies. Still, the implementations made on the car functioned as intended and were shown to be drivable remotely while operated relatively efficiently, according to the data gained from the tests.

5.2 Hardware implementations

To find the right configuration of the different hardware utilized during the thesis and figuring out what hardware was required for the implementations on both the rig and the car, was time demanding. During the thesis, a better understanding of what was required was needed and all the hardware, together with the software,

started to come together. Therefore the outcome was a complete and fully functional implementation setup for both the test rig and for the car.

Whilst performing the tests, the setup undeniably could have been better. The safety driver in the driving seat of the car is forced to physically hold the laptop running the live stream from the GoPro. A better solution for this could have been achieved, where the laptop would have been more securely stationed in the car to prevent incidents regarding the laptop and the drivers' health. The remote steering wheel was placed in between the middle backseat and the console, which also could have been placed more efficiently.

From the result of the tests, an illustrative figure of how the car was manoeuvring during the high intensity test was created. This provided a feeling of how the outcome of the test was, but due to the limited time of the bachelor's thesis, a solution for a more precise analysis regarding the previous was excluded. By having more precise data from the tests, a more accurate result could have been gained.

Since the start of the thesis, there have been issues getting the right hardware when needed. Obtaining the hardware proved to be more time demanding than anticipated. A lesson learned regarding this area for future thesis work, to be more aware of the actual time it takes to obtain the right hardware in time. Additionally, getting the right configuration on the VN module and to make it fully functional for the created hardware and software setup, was something that was proving to be a huge struggle over a large duration of the thesis. There are always difficulties in estimating the amount of time needed in achieving different sub goals during a thesis work. This is a lesson learned for the future, to do things more efficiently.

5.3 Software implementations

During this thesis, the implementation of the different sub goals concerning the software aspect of the thesis has proven to be successful. The different programming scripts of both Python and CAPL worked as intended for sending the input value of the remote steering wheel and by doing so over UDP. Every step worked with no larger issues, but the road to achieve these results have not been without setbacks. During the creation of the simulation setup and the writing of the different programs implemented during the thesis, the hardware issues resulted in a postponed development due to the time demanding steps getting the right configuration and hardware. Due to the lack of understanding what issues needed to be solved for a successful implementation of the setup in the Volvo XC60, a lot of time was spent on testing different setups and solutions over different areas, during the duration of the thesis.

Moreover, during the first implementation of the complete setup developed during the thesis applied on the car. The car accepted the breakout cabling kit and the simulation setup developed for the purpose of fooling the car into believing the park assist pilot interface was active. This implementation had a successful outcome,

due to the car showing no signs of repulsion of the applied software and hardware setup on the XC60. By doing so, the signals and messages regarding the VDDM and PSCM frames will be modified and the first expectations concerning this was that the car would show great signs of repulsion. This enabled the time gained to be focused on developing and investigating in other areas of the thesis.

Throughout the different tests performed, some safety issues were affecting the outcome of the tests and forced adjustments to be applied to the design of the tests executed. For example, due to the underlying danger of a large arbitrary value change, only the 0 ms and 50 ms simulated time-delays were performed during the low intensity test. By having more time for developments and adjustments for a rate limiter in CAPL, might have provided an extra layer of safety that would enable a test regarding an extreme simulated time-delay to be performed, and more data to analyse.

As mentioned previously in the result chapter, the car was forced into a city safety interface when the safety driver had to intervene during a failed test, to by force physically take over the steering wheel. By having more time for developments and testing the implementation of a steering wheel torque limiter that would change the *LatCtrlModRegSafe* to the regular driving interface when a take-over is needed during the tests, a better outcome could have been achieved.

Furthermore, an issue that was realized during the tests was an issue related to the reason why a rate limiter was needed. The remote steering wheel was not completely proportional to the real steering wheel of the car. This due to the remote steering wheel having no feedback available both regarding the steering wheel itself but also a software implementation of feedback to the remote steering wheel itself. The car could therefore behave unpredictable during high intensity manoeuvres.

5.4 Further developments

As mentioned during the discussion chapter so far, there are multiple areas of improvements. First and foremost, the whole thesis is based on steering a car remotely over Wi-Fi with the help of UDP. Due to the limited time of the bachelor's thesis a decision was made to make a limitation of the thesis to exchange the Wi-Fi latency with a simulated time-delay over a LAN connection between the external laptop running the Python program and the VN8911. The ideal scenario would instead be to apply a central control unit where the remote steering angles were sent from a far, over Wi-Fi. This could easily have been solved without the lack of time during the thesis. The solution would have been to implement two Wi-Fi access points, one on the car and one by the central control unit. Moreover, the same software and hardware implementation and all other areas of the thesis is based on being able to steer the car over Wi-Fi and would support this solution. The test rig is a proof of this statement, where the whole setup is fully functional over a Wi-Fi connection.

Additionally, to be able to produce better test results and a better test environment,

a more suitable camera equipment for livestreaming with a much lower latency would have been needed as an improvement. Performing the tests would have been easier and more accurate and the analyses gained might prove that the car could have been manoeuvred more efficiently during larger simulated time-delays and latencies. A better camera equipment could also provide a more realistic video livestream perspective. The perspective used during the tests, was something the test drivers reacted to as being a nuisance while trying to perceive different distances and overall perspective of the environment.

Regarding the hardware setup in the car, a computer table for both laptops and some sort of a more stable setup for the remote steering wheel would have improved the experience and the safety of the tests performed. This is something that can be improved for future works inspired by this thesis. Furthermore, for a better result regarding more accurate data gathered during the tests and a better correlation between the remote steering wheel and the steering rack of the car, implementing feedback to the remote steering wheel would be preferred and might be a capable solution. This could be done by either replacing the remote steering wheel itself for a far more optimal one, or by somehow implementing feedback in the software programming script.

Moreover, to produce more accurate data analyses and to design better and more extensive test courses, more than just a few test drivers is needed. By having a larger amount of test drivers, more quantitative, correct, and rich data would be obtained. The outcome would therefore be based on the general population rather than a few individuals. A result produced by a quantitative test is more reliable and trustworthy and would therefore be truly valuable and increase the credibility for this thesis.

Bibliography

- [1] *Python* Accessed:Jan, 2021. [Online]. Available:<https://www.python.org/>
- [2] *Using socket in Python* Accessed:Feb 2021. [Online]. Available: <https://docs.python.org/3/library/socket.html>
- [3] *Insert Timer Python* Accessed:May 2021. [Online]. Available: <https://docs.python.org/3/library/time.html?highlight=time#module-time>
- [4] *Pygame library in Python* Accessed:Mar 2021. [Online]. Available: <https://www.pygame.org/docs/ref/joystick.html>
- [5] *UDP* Accessed:Feb 2021. [Online]. Available: <https://www.javatpoint.com/computer-network-tcp-ip-model>
- [6] P. Thagard. (1989) *A theory of explanatory coherence. Explanatory coherence*. Cognitive Science Laboratory, Princeton University, 221 Nassau St., Princeton, NJ 08540. <http://www.arts.uwaterloo.ca/~pthagard/Articles/1989.explanatory.pdf>
- [7] *UDP Architecture* Accessed:Feb 2021. [Online]. Available: <https://www.eetimes.com/guide-to-embedded-systems-architecture-part-3-transport-layer-udp-a>
- [8] *UDP* Accessed:Feb 2021. [Online]. Available: <https://internetstiftelsen.se/guide/introduktion-till-ip-internet-protocol/tcp-och-udp-nivan/>
- [9] *Matlab* Accessed:May 2021. [Online]. Available: https://se.mathworks.com/company.html?s_tid=hp_ff_a_company
- [10] Vector. Stuttgart, Germany, *CANoe (2021)*. Accessed:Apr, 2021. [online]. Available:<https://www.vector.com/int/en/products/products-a-z/software/canoe/>
- [11] *OBS studio* Accessed: April 2021. [Online]. Available: <https://obsproject.com/sv>
- [12] *History of CAN technology* Accessed:May 2021.[Online]. Available: <https://www.can-cia.org/can-knowledge/can/can-history/>
- [13] M.D.Natale, H.Zeng, P.Giusto, A.Ghosal, *Understanding and Using the Controller Area Network Communication Protocol.*, Springer, New York
- [14] Volvo Car Corporation, Sweden, *Park Assist Pilot (2019-11)* Accessed: June, 2021. [Online]. Available: <https://www.volvocars.com/se/support/topics/ansanda-din-bil/bilfunktioner/park-assist-pilot>
- [15] *Volvo Car Steering Rig*, Vector Sweden, Gothenburg, Sweden, 2017
- [16] Vector InformatikGmbH, *VN8900 Interface Family Manual v6.4*, (2019), [Online], Available: https://assets.vector.com/cms/content/products/VN89xx/docs/VN8900_Manual_EN.pdf

- [17] M. Harrer , P. Pfeffer , HH Braess . (2017) *Steering-Feel, Interaction Between Driver and Car. Steering Handbook*. Springer, Cham. https://doi.org/10.1007/978-3-319-05449-0_7
- [18] L. Ljung. (1999) *System Identification: Theory for the User*. Linköping, Sweden: https://books.google.se/books/about/System_Identification.html?id=nHfoQgAACAAJ&redir_esc=y
- [19] B. Thomas, Liber AB, "Frekvensanalys, Beräkning av egenskaper hos regler-system," in *Modern Reglerteknik*, 5th ed. Göteborg, Sweden: 2016, pp. 133-191.

A

Appendix A

2	Test: Drive the entire course at PV norra, without C							
3	Delay: The added steering delay, the delay to send values were not taken in to account.							
4								
5	Date:		Test Driver	Test nr:	Delay	Velocity	Feeling	Special event:
6	4/27/2021	Torsland a	Tobias	1	0	30 km/h straight, 15 km/h track/curve	Normal	no
7	4/27/2021	Torsland a	Tobias	1	25ms	30 km/h straight, 15 km/h track/curve	Normal	no
8	4/27/2021	Torsland a	Tobias	1	50ms	30 km/h straight, 15 km/h	No noticeable diffrent	no
9	4/27/2021	Torsland a	Tobias	1	75ms	30 km/h straight, 15 km/h track/curve	Delay was noticeable, but worked fine to steer.	Car did a weird snath at 30 km/h on straight going back.
10	4/27/2021	Torsland a	Tobias	1	100ms	30 km/h straight, 15 km/h track/curve	Delayen a bit more noticeable, ut still possible to manuver.	Car did a weird snath at 30 km/h on straight going back.
11	4/27/2021	Torsland a	Tobias	1	125ms	30 km/h straight, 15 km/h track/curve	Delayen a bit more noticeable, ut still possible to manuver.	Car did a weird snath at 30 km/h on straight going back.
12	4/27/2021	Torsland a	Tobias	1	200ms	30 km/h straight, 15 km/h track/curve	Delay is noticeable, cars steering wheel is acting jerky at som points. Hard to performe sweeping turns.	no
13	4/27/2021	Torsland a	Tobias	1	275ms	30 km/h raksträcka, 15 km/h bana/kurva	It it starting to get uncomfortable to steer the car. Especially in the begining before you	no

Figure A.1: Maximum delay table

14	4/27/2021	Torsland a	Tobias	1	350ms	30 km/h straight, 15 km/h track/curve	Very uncomfortable to steer the car. It was impossibale to navigate the course at the choosen	no
15	4/27/2021	Torsland a	Ossian	1	0	40 km/h straight, 20 km/h	No	no
16	4/27/2021	Torsland a	Ossian	1	25ms	40 km/h straight, 20 km/h	No	no
17	4/27/2021	Torsland a	Ossian	1	50ms	40 km/h straight, 20 km/h track/curve	No noticeable diffrent.	Car did a weird snath at 30 km/h on straight going back.
18								
19								
20								

Figure A.2: Maximum delay table continuation

B

Appendix B

```
import pygame
import keyboard
import socket
import struct
import threading
import time

# Timer Interrupts:
InitiationTime = time.time()
def irq_timedelay_ms():
    # determines the lenght of delay wanted
    time.sleep(0.5)

# Initialize the Pygame library
pygame.init()
# Initialize the joystick module.
pygame.joystick.init()
# The input value of the Thrustmaster T300 RS
horizontal_axis = 0
# The x-axis coordinate that are to be sent
x_coordinate = 0

done = False

# Count the joysticks the computer has
joystick_count = pygame.joystick.get_count()
print("Number of joysticks found", joystick_count)
if joystick_count == 0:
    # No joysticks were found!
    print("Error , no joysticks has been identified.")
else:
    # Use joystick #0 and initialize it
    Thrustmaster_T300_RS = pygame.joystick.Joystick(0)
    Thrustmaster_T300_RS.init()
    print("found a joystick")
```

```
is_init = pygame.joystick.get_init()
# returns true if a joystick is initialized, false otherwise
print(is_init)

# Creates UDP sockets
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# internet, UDP

# Connects the socket to a port and an IP-address
# specifies IP-address for port and server
# Wi-Fi
server_address = ('10.246.39.162', 40001)
# LAN
# server_address = ('169.254.202.226', 40001)

print('UDP_server_address_{}_port_{}'.format(*server_address))

while not done:
    # Sending the steering angle with the intervall of 10ms
    time.sleep(0.010000 - ((time.time() - InitiationTime) % 0.010000))

    i = 0
    j = 16
    x_coordinate = 0
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    # As long as there is a joystick
    if joystick_count != 0:
        # This gets the position of the axis on the game controller
        # It returns a number between -1.0 and +1.0
        # 0 for horizontal axis and 1 for vertical axis
        horizontal_axis = Thrustmaster_T300_RS.get_axis(0)
        x_coordinate = x_coordinate + horizontal_axis
        # An intentional time delay between reading the steering angle v
        irq_timedelay_ms()
        if keyboard.is_pressed('q'):
            done = True

    if x_coordinate < 0:

        x_coordinate = -x_coordinate
        # convert to a string
        x_coordinate_string = str(x_coordinate)
        x_coordinate_string = x_coordinate_string.zfill(16)[i:j]
```

```
    # converts the string to bytes, to then send via UDP
    x_coordinate_string = bytes(x_coordinate_string, 'utf-8')
    # Send the UDP package to designated server address
    sock.sendto(x_coordinate_string, server_address)
    print(x_coordinate_string)

elif x_coordinate >= 0:

    x_coordinate = -x_coordinate
    # convert to a string
    x_coordinate_string = str(x_coordinate)
    x_coordinate_string = x_coordinate_string.zfill(16)[i:j]
    # converts the string to bytes, to then send via UDP
    x_coordinate_string = bytes(x_coordinate_string, 'utf-8')
    # Send the UDP package to designated server address
    sock.sendto(x_coordinate_string, server_address)
    # Print to see the values sent in the terminal
    print(x_coordinate_string)

pygame.quit()
```


C

Appendix C

```
import pygame
import keyboard
import socket
import struct
import threading
import time

# Timer Interrupts:
InitiationTime = time.time()
def irq_timedelay_ms():
    # determines the lenght of delay wanted
    time.sleep(0.5)

# Initialize the Pygame library
pygame.init()
# Initialize the joystick module.
pygame.joystick.init()
# The input value of the Thrustmaster T300 RS
horizontal_axis = 0
# The x-axis coordinate that are to be sent
x_coordinate = 0

done = False

# Count the joysticks the computer has
joystick_count = pygame.joystick.get_count()
print("Number of joysticks found", joystick_count)
if joystick_count == 0:
    # No joysticks were found!
    print("Error , no joysticks has been identified.")
else:
    # Use joystick #0 and initialize it
    Thrustmaster_T300_RS = pygame.joystick.Joystick(0)
    Thrustmaster_T300_RS.init()
    print("found a joystick")
```

```
is_init = pygame.joystick.get_init()
# returns true if a joystick is initialized, false otherwise
print(is_init)

# Creates UDP sockets
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# internet, UDP

# Connects the socket to a port and an IP-address
# specifies IP-address for port and server
# Wi-Fi
server_address = ('10.246.39.162', 40001)
# LAN
# server_address = ('169.254.202.226', 40001)

print('UDP_server_address{},port{}'.format(*server_address))

while not done: # This while loop might not be needed in UDP, bcs of UDP

    # measure the time between start and end
    start = time.perf_counter()
    # measure the time between start and end in nanoseconds
    # start = time.perf_counter_ns()
    # Sending the steering angle every 10ms
    time.sleep(0.010000 - ((time.time() - InitiationTime) % 0.010000))
    #time.sleep(1 - ((time.time() - InitiationTime) % 1)) # Sending the
    irq_timedelay_ms()

    i = 0
    j = 16
    x_coordinate = 0

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

        # Possible joystick actions: JOYAXISMOTION JOYBALLMOTION JOYBUTTON
        # if event.type == pygame.JOYAXISMOTION: #om man vill se att det

    # As long as there is a joystick
    if joystick_count != 0:
        # This gets the position of the axis on the game controller
        # It returns a number between -1.0 and +1.0
        horizontal_axis = Logitech_G920.get_axis(0) # 0 for horizontal o

        #x_coordinate = x_coordinate + horizontal_axis
```

```

if keyboard.is_pressed('q'):
    done = True

x_coordinate = -1
if x_coordinate < 0:

    x_coordinate = -x_coordinate
    x_coordinate_string = str(x_coordinate) # convert to a string
    x_coordinate_string = x_coordinate_string.zfill(16)[i:j]
    x_coordinate_string = bytes(x_coordinate_string, 'utf-8')
# converts the string to bytes, to then send via UDP
    sock.sendto(x_coordinate_string, server_address)
# Send the UDP package to designated server address
    print(x_coordinate_string)

elif x_coordinate >= 0:

    x_coordinate = -x_coordinate
    x_coordinate_string = str(x_coordinate) # convert to a string
    x_coordinate_string = x_coordinate_string.zfill(16)[i:j]
    x_coordinate_string = bytes(x_coordinate_string, 'utf-8')
# converts the string to bytes, to then send via UDP
    sock.sendto(x_coordinate_string, server_address)
# Send the UDP package to designated server address
    print(x_coordinate_string)

# internet, UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# specifies IP-address for port and server, for receiving UDP-package
# server_address_receiving = ('10.246.39.178', 40010)
# specifies IP-address for port and server, for receiving UDP-package
server_address_receiving = ('169.254.21.61', 40010)
sock.bind(server_address_receiving)

# The receiving part: used to see the delay-time
#while True:
print("Waiting to receive data")
data = 0
# buffer size is 1024 bytes
data, address = sock.recvfrom(1024)
# Writes the received data
print("Steering angle: {}, received from {}".format(data, address))
if data != 0:
    print("Intervall of sending UDP package:")
    end = time.perf_counter()
    print(end - start)

```

```
    if not data:
        break

    #Compare the angle value that were sent and the value that were recieved
    print("Steering_angle_sent: {}, Steering_angle_recieved {}".format(data['Steering_angle_sent'], data['Steering_angle_recieved']))

pygame.quit()
```

D

Appendix D

```

variables
{
    UdpSocket gSocket;
    char      gRxBuffer[16]; // changed from 1500 to 8. again 2021-03-25 to
    double SteeringAngle_double; // Variable that receives steering angel f
    double SteeringAngle_double_old;
    int SteeringRatio = 10; // Steering ration between, remot steeringwheel
}

on sysvar_update sysvar::Receiver::Open
{
    // on open button down...
    if (@this == 1)
    {
        // Open an UDP socket. As source address 0.0.0.0 is used,
        //this means that
        // the configure address of the TCP/IP stack is used.
        //See TCP/IP stack
        // configuration dialog in the simulation setup
        // On UDP port 40001 we want to receive UDP pacekts.
        gSocket = UdpSocket::Open( IP_Endpoint(169.254.202.226:40001) );
        //Changed from 10.246.39.162:40001 on 2021-04-19 (11:32), 169.254.202.226
        //169.254.202.226
        if (IpGetLastError() != 0)
        {
            // if UdpSocket::Open fails , we print a message to the write window
            write( "<%BASE_FILE_NAME%>_UdpSocket::Open_failed_with_reauls_%d",
            return;
        }
        // To receive data with the created socket,
        //we have to call ReceiveFrom.
        gSocket.ReceiveFrom( gRxBuffer, elcount(gRxBuffer) );

        // if ReceiveFrom does not immediatelly copy to to gRxBuffer,
        // it returns 997 to
        // indicate it will call the callback function
    }
}

```

```

//OnUdpReceiveFrom later.
if ((gSocket.GetLastError() != 0) &&
(gSocket.GetLastError() != 997))
{
    char errorString[100];
    // if ReceiveFrom fails, we print a message to the write window
    gSocket.GetLastErrorAsString( errorString, elcount
    (errorString) );
    write( "<%BASE_FILE_NAME%>_ReceiveFrom_failed_with_result_%d_(%s)",
    IpGetLastError(), errorString );
}

// update panel controls state
EnableControl( "Receiver", "OpenButton", 0 );
EnableControl( "Receiver", "CloseButton", 1 );
}
}

on sysvar_update sysvar::Receiver::Close
{
    // on open button down...
    if (@this == 1)
    {
        // Close socket
        gSocket.Close();

        // update panel controls state
        EnableControl( "Receiver", "OpenButton", 1 );
        EnableControl( "Receiver", "CloseButton", 0 );
    }
}

on preStop
{
    // Close socket on measurement stop
    gSocket.Close();
}
// Implement OLd vs Newvalue, and citysafe to break earlier..
// Callback function, which is called if a UDP packet is received
void OnUdpReceiveFrom( dword socket, long result, ip_Endpoint
remoteEndpoint, char buffer[], dword size)
{
    write("recived_package"); // Print if package is received
    if (result == 0)
    {
        char endpointString[30];

```

```

remoteEndpoint.PrintEndpointToString( endpointString );
// set sysvars to display receive data in Receiver panels
sysSetVariableString( sysvar::Receiver::RxAddress, endpointString );
sysSetVariableString( sysvar::Receiver::RxText, buffer );

// If PAP is in use and wheel gets Tq over 2 Nm, CAN 1
if ( abs( $ChassisCANhs::PSCM::PSCMChasFr02::SteerWhlTq) > 3)
//if ( abs( $CAN2_Networks::PSCM::PSCMChasFr02::SteerWhlTq) > 2)
{
// change from ParkAssist to regular mode , CAN 1
$ChassisCANhs::VDDM::VDDMChasFr07::LatCtrlModReqSafe = 0;
//$CAN2_Networks::VDDM::VDDMChasFr07::LatCtrlModReqSafe = 0;
}/*else{
// added 2 * 2021-03-25
SteeringAngle_double_old = SteeringAngle_double;
// Converts the string buffer into a double number
SteeringAngle_double = (SteeringRatio*atodbl(buffer));
if( abs(SteeringAngle_double-SteeringAngle_double_old) > 0.2)
{
SteeringAngle_double = SteeringAngle_double_old;
}*/
SteeringAngle_double = (SteeringRatio*atodbl(buffer));
// writes the steering angel when Park Assist active
$ParkAssiPinionAgReq = SteeringAngle_double;

// writes the steering angel when trafficjam active
//$AsyPinionAgReq = SteeringAngle_double;
// Prints steering angel in write window
//write("%s", SteeringAngle_double);
// }
}

// To receive more data, we have to call ReceiveFrom again.
gSocket.ReceiveFrom( gRxBuffer , elcount(gRxBuffer) );
}

```


E

Appendix E

```

includes
{

}

variables
{

}
//test 2021-04-23
/*
on message CAN2_Networks::SAS::SASChasFr01
{
    $ChassisCANhs::SAS::SASChasFr01::SteerWHLAgSafe = $CAN2_Networks::SAS::
    //$ChassisCANhs::PSCMChasFr01::PinionSteerAg1 = $ChassisCANhs::SAS::SA
}
*/
//-----# CAN1 --> CAN2 #-----
//on message ChassisCANhs
on message ChassisCANhs::PSCM::PSCMChasFr01
{
    message CAN2_Networks.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
}
on message ChassisCANhs::PSCM::PSCMChasFr02
{
    message CAN2_Networks.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received

```

```
{
    m=this; // copy data from the received message to a new message
    output(m); //send the new message on CAN 2
}
}
on message ChassisCANhs::PSCM::PSCMChasFr03
{
    message CAN2_Networks.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
on message ChassisCANhs::PSCM::PSCMChasFr04
{
    message CAN2_Networks.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
}
on message ChassisCANhs::PSCM::PscmChasNMFr
{
    message CAN2_Networks.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
}
on message ChassisCANhs::VDDM::VDDMChasFr15
{
    message chassiCAN_VDDM.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received
```



```

    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}

on message ChassisCANhs.VddmChasNMFr
{
    message chassiCAN_VDDM.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==1)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
//-----# CAN2 --> CAN1 #-----
//on message chassiCAN_VDDM.SASChasFr01
on message CAN2_Networks.SASChasFr01
{
    message ChassisCANhs.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==2)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
//on message chassiCAN_VDDM.SasChasNMFr
on message CAN2_Networks.SasChasNMFr
{
    message ChassisCANhs.* m;
    //if ChassisCANhs
    //{
    if((this.DIR == RX) && (this.CAN==2)) //check if the frame was received
    {
        m=this; // copy data from the received message to a new message
        output(m); //send the new message on CAN 2
    }
}
}

```


F

Appendix F

```
variables
{
    UdpSocket gSocket;
    int var = 0;
    char text[16];
}

//on start
//on sysvar_update sysvar::Receiver::Open
on sysvar_update sysvar::Sender::RxText
{
    // Open an UDP socket. As source address 0.0.0.0 is used, this means th
    // the configure address of the TCP/IP stack is used. See TCP/IP stack
    // configuration dialog in the simulation setup
    // As port no ist used, this means a source port is dynamically assign
    // by the TCP/IP stack.
    gSocket = UdpSocket::Open( IP_Endpoint(169.254.21.61) ); // changed fro

    if (IpGetLastError() != 0)
    {
        // if UdpSocket::Open fails, we print a message to the write window
        write( "<%BASE_FILE_NAME%>_UdpSocket::Open_failed_with_result_%d", Ip
    }
}

on preStop
{
    // Close socket on measurement stop
    gSocket.Close();
}
/*on sysvar sysvar::Receiver::Open
{
    // send on button down
    if (@this == 1)
    {
        char text[16]; // changed from 200 2021-04-13 13:13
```

```
// get string from sysvar
sysGetVariableString( sysvar::Receiver::RxText, text, elcount(text) ,
write("%s", text);

// send text to IP address/UDP port of the receiver
gSocket.SendTo( IP_Endpoint(10.246.39.178:40010), text, strlen(text)
}
}*/
//on signal ParkAssiPinionAgReq
// bort kommenterat 2021-05-21

//on message VDDMChasFr15
on message PSCMChasFr02
{
    if($ParkAssiPinionAgReq != 0)
    {

        // get string from sysvar
        //sysGetVariableString( sysvar::Sender::RxText, text, elcount(text) ,
        text[0] = '1';

        // send text to IP address/UDP port of the receiver
        gSocket.SendTo( IP_Endpoint(169.254.21.61:40010), text, strlen(text)
169.254.21.61
        write("Package sent"); // Print if package sent
    }
}
```



CHALMERS