Master's thesis

2022

Alexey Gusev

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Alexey Gusev

Remote Control of Autonomous Vehicle Through 5G Network With Integrated Operator-Assistance System

July 2022

NTNU
Norwegian University of
Science and Technology

Remote Control of Autonomous Vehicle Through 5G Network With Integrated Operator-Assistance System

Alexey Gusev

Computer Science
Submission date: July 2022
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth

Norwegian University of Science and Technology
Department of Computer Science

Abstract

The promise of autonomous technology has long been tempting. It has the potential to transform our experience of travel, remove people from high-risk working environments, help with transport, streamline our industries and more. Self-driving vehicles are slowly but surely starting to become a reality despite the many obstacles still to be overcome. Especially in aspects as object detection in bad weather, full autonomous control, reaction to sudden issues or unpredicted situation where the vehicle can end up doing harm. In all of those situations it is important to have a possibility to take manually control over the vehicle in order to safely bring it back to a normal state or condition. This thesis aims to explore and determine how to build such an operator-assistance system for real-time remote control of AV, and show the feasibility of it.

Different methods was used when exploring possible implementations of the system such as connection, control, object detection, steering, video transmission and a GUI for the operator. The result was an operator-assistance system build on top of a Nvidia Drive platform on a full scale electric KIA Niro and a ARM-based Linux computer, NVIDIA DRIVE AGX Xavier. The connection between operator and the autonomous vehicle was realized through Wireguard VPN over a 5G cellular network with a specialized 5G-router mounted into the car, in order to minimize transmission delay and jitter. By further using ROS as a base for controlling and sending data it was made possible to remote control the AV and transmit video. In order to integrate an operator-assistant in the future, helping avoid bad maneuvers, object detection with Yolo5 was tested, implemented and added onto the received data. All of this was later collected, automated and presented in a graphical interface available for operator.

Sammendrag

Løftet om autonom teknologi har lenge vært fristende. Den har potensial til å transformere opplevelsen vår av reise, fjerne folk fra høyrisikoarbeidsmiljøer, hjelpe til med transport, effektivisere bransjene våre og mer. Selvkjørende kjøretøy begynner sakte men sikkert å bli en realitet til tross for de mange hindringene som fortsatt må overvinnes. Spesielt i aspekter som gjenstandsdeteksjon i dårlig vær, full autonom kontroll, reaksjon på plutselige problemer eller uforutsette situasjoner der kjøretøyet kan ende opp med å gjøre skade. I alle disse situasjonene er det viktig å ha en mulighet til å ta manuell kontroll over kjøretøyet for å trygt bringe det tilbake til normale omgivelser og tilstand . Dette prosjektet tar sikte på å utforske og bestemme hvordan man kan bygge et slikt operatørassistansesystem for sanntids fjernkontroll av autonome kjøretøy, og vise gjennomførbarheten av det.

Ulike metoder ble brukt under utforskning av mulige implementeringer av systemet; høyhastighetstilkobling i sanntid, kontroll, objektdeteksjon, styring, video-overføring og en GUI til bruk for operatøren var blant det som skulle være operativt. Resultatet var et operatørassistansesystem bygget på toppen av en Nvidia Drive-plattform på en fullskala elektrisk KIA Niro og en ARM-basert Linux datamaskin, NVIDIA DRIVE AGX Xavier. Forbindelsen mellom operatøren og det autonome kjøretøyet ble realisert gjennom Wireguard VPN over et 5G mobilnettverk med en spesialisert ruter montert i bilen, for å minimere overføringsforsinkelser og jitter. Ved ytterligere å bruke ROS som en base for å kontrollere og sende data ble det mulig å fjernstyre bilen og overføre video. For å hjelpe operatøren med å unngå dårlige manøvrer, ble objektdeteksjon med Yolo5 implementert og lagt til de mottatte dataene. Alt dette ble senere samlet, automatisert og presentert i et grafisk grensesnitt tilgjengelig for operatøren.

Preface

This specialization project is a part of NAP-lab (NTNU Autonomous Perception Lab) research group located at the Norwegian University of Science and Technology (NTNU).

I am thankful to my supervisor Gabriel Kiss and co-superviser Frank Lindseth for providing invaluable feedback throughout the project and also for giving me access to the required resources. A big thanks goes also to "Værnes Garnison" who let us test the automated vehicle at Værnes airport in Trondheim.

Nomenclature

Acronyms and Abbreviations:

$5G$	Fifth-generation technology standard for broadband cellular networks
CAN	Controller Area Network
ESC	Electronic Speed Controller
FOV	Field Of View
$GMSL$	Gigabit Multimedia Serial Link
IMU	Inertial Measurement Unit
$LiDAR$	Light Detection And Ranging Sensor
$MQTT$	Message Queuing Telemetry Transport
$NAP-Lab$	NTNU Autonomous Perception Laboratory
OS	Operative System
PID	Proportional Integral Derivative Controller
R/C	Radio Control
ROS	Robot Operating System
$SLAM$	Simultaneous Localization and Mapping
SSH	Secure Shell
VCU	Vehicle Control Unit
VPN	Virtual private network
$YOLO$	You Only Look Once (family of compound-scaled object detection models)

Contents

Figures

Chapter 1

Introduction

This chapter introduces the project and will briefly cover the thesis's motivation, goals and research method. Due to one of the main focuses of this paper being the use of autonomous vehicles in cities and other public areas, this aspect, its dependencies and today's as-is situation are introduced thoroughly. In addition, the goals and research challenges will be set and discussed.

1.1 Motivation

Self-driving vehicles are slowly but surely starting to become a reality despite the many obstacles still to be overcome. This means that in the near future, the world could change in some unexpected ways. It is already observed that many companies invest big money in their self-driving car projects, such companies as Waymo(Google), Uber, Tesla Yandex, BMW and Audi, to mention a few.

This technology can be the key to building future cities where our reliance and relationship with vehicles are redefined. In addition, it can lower carbon emissions and pave the way for more sustainable ways of living. Nevertheless, the biggest problem for the engineers in the autonomous technology industry is getting the cars to operate safely and effectively in complex and unpredictable human environments and bad, snowy weather conditions. As a result, autonomous cars will inevitably end up in a complex situation where some sensor information is wrong, limited, or something unreasonable happens on the road, leading the vehicle into a possible "dead-lock". In this situation, no progress is possible because two or more competing actions are waiting for the other to finish.

This paper will concentrate on developing a proof-of-concept solution for the dead-lock problem on a full-scale vehicle, contributing to the development of AV. The proposed solution is an integration of a remote control system as an indispensable part of every AV system over the new 5G cellular network for low latency telemetry and an additional operator-assistance system in order to counteract any collapse of the system or bad manoeuvres by the operator when burst packet loss occurs, or operator makes any mistake. With such a system, edge cases and dead-locks should no longer have to be a limitation or threat for self-driving cars.

1.2 Goals and research challenges

The goals of this thesis are introduced and described in this section. The formation of such questions provides an important driving force for the project. Furthermore, provide clarity as to the goals sought. The later results presented in chapter 5 will be solely based on goals and research challenges set in this section. It is worth mentioning that this project combines exploration, different methods, modern technologies, and knowledge followed by consistent testing into a complete system. This way visualizes in full scale the concept of remote-controlled cars that later can be used for further research or development. Figure 1.1 also shows a feature tree diagram with proposed feature to develop and test for the system.

> ***Goal 1:*** *Explore and show a possible architecture and functionality of a full-scale autonomous vehicle based on Nvidia Drive Platform*

The project will be based on the resources provided by the research group NAPLab, the Kia Niro with integrated Nvidia Drive Platform, and all necessary sensors for making the car fully autonomous. A further technical specification is shown in section 3.4. Knowing the car's architecture is crucial for further research and development.

> ***Goal 2:*** *Explore, build and test a complete system that enables remote control of an autonomous vehicle as a proof-of-concept solution for occurring edge cases in traffic such as "dead-locks (**??**).*

Nowadays, autonomous cars still need much human support, mainly when a problem or an unfamiliar situation occurs. A human can always be inside a vehicle and control the situation if something happens, but what if the vehicle is so small that it lacks space for a human operator, or the vehicle is made for driving fully autonomous; no human operator will be by its side. The company Yandex have been developing autonomous cars and rovers for delivering food for many years. When a unique situation occurs without the rover detecting what to do, the control is handed over to a control centre over LTE. Operators can see real-time video and help the rover out.[1]

This technology is used for low speed and will need less latency and higher network speed to work in real-time with a full-scale autonomous car. Therefore, it will be appropriate to contribute to this area of development and research by experimenting with a remote-controlled vehicle with the new 5G network.

> ***RC2.1:*** *Ensuring secure communication and data transfer*

Teleoperation can control autonomous vehicles when edge cases or the need for it occurs (2.1.3). However, that requires real-time information about traffic situations, status and vehicle control to be sent over from the car to the operator. Therefore, the remote controller must retain all necessary information and data over a secure, stable, high-performing communication channel (4.1).

> ***RC2.2:*** *Enabling brake, steering and throttle control of a car over IP.*

In an autonomous car's software or hardware stack, there are various parts, all important for the system to function. However, some are still more dominant, and a part of the core, like the control system, which is especially crucial for the remote control system of this thesis [2]. For the operator to remotely control the car, all vital parts such as steering, brakes and throttle must be done by wire and not physically in the car. The operator also needs to have a high response time from the system to control it at high-speed 2.1.4.

> ***RC2.3:*** *Realizing real-time video transmission from an Nvidia Drive Platform.*

Video transmission in real-time is crucial for the operator to manoeuvre the vehicle remotely. The operator is wholly dependent on the video stream delivered by the system to entail a high response time in any situation. In the research done by M. Claypool and D. Finkel [3] any first-person game is both the most sensitive class of games and the one having the most significant negative impact by increased latency. The remote control system with a possible user interface can be seen as similar to a first-person game. Therefore, the system must implement a stable and high-quality video stream.

> ***RC2.4:*** *Designing GUI for the remote control system, such that an operator can have reliable control of the vehicle*

A graphical user interface plays a vital role for the end-user. Therefore, the design needs to support and highlight the potential of the remote control system, and GUI itself needs to be highly functional, aesthetically pleasing and user-friendly. The aforementioned is essential because of the real-time environment, where stressing or poorly designed UI plays a critical role in user performance and efficiency and can entail a bad user experience and, in the worst case, lead to possible accidents [4].

> ***RC2.5:*** *Mapping out methods and algorithms for object detection in real-time, that later can be available for operator assistance*

Operator-assistance system is important as a part of the remote control system to counteract any system collapse or bad manoeuvres by the operator when burst packet loss occurs, or the operator makes any mistakes. Object detection sub-task is one of the most critical prerequisites in many autonomous driving systems and vehicles. The task allows the car controller to account for obstacles when considering the diversity of future trajectories to follow and is therefore crucial for the autonomous navigation [5].

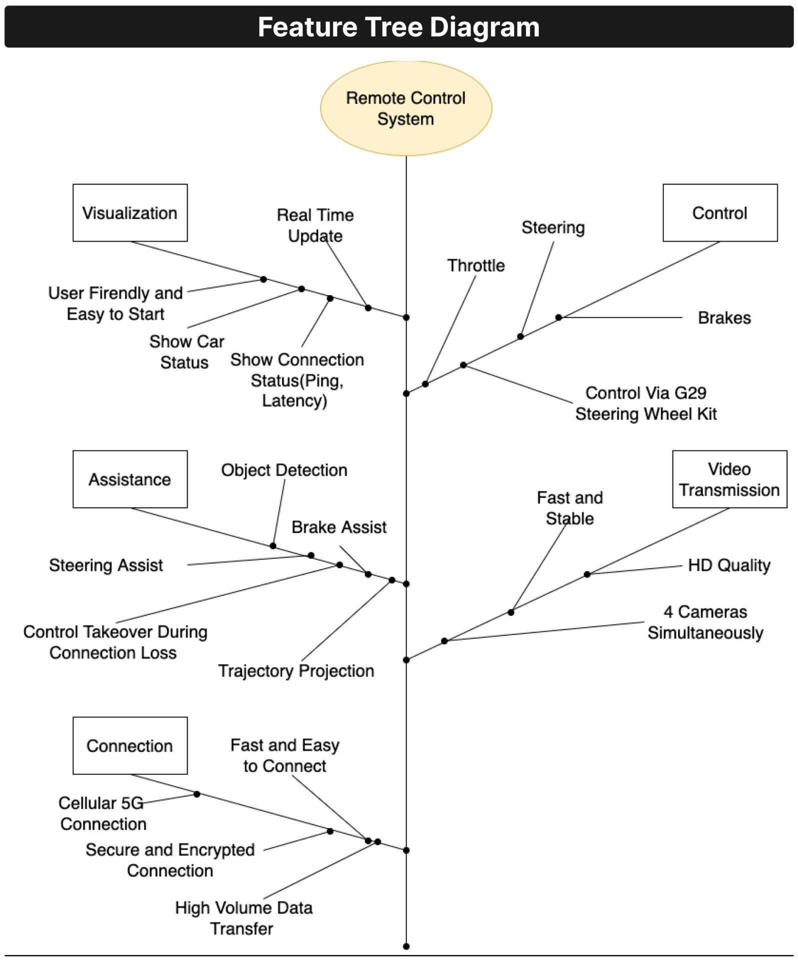


Figure 1.1: Feature tree diagram containing the proposed features of the Remote Control System that are desired to be developed.

1.3 Research Method

This chapter contains a brief account of methodological choices. The thesis is a qualitative study, a standard research method to use when studying a phenomenon in depth [6]. In this case, the phenomenon is to get insight into an autonomous vehicle and understand how to control it remotely over IP, as explained in section 1.2. Most of the thesis is based on a qualitative literature study - a process of studying various documents to try to answer the overall research challenge [7]. The paper is also a feasibility and experimental study, which means that technical aspects of the concept are taken into account and also tested [8].

In addition to previous research papers related to the topic, various databases and search engines were used to find sources, such as Google Scholar and Oria. The relevance and credibility of the sources were assessed before use. When obtaining sources, various articles and journals were first assessed for relevance by the title. Then the summary and the introduction were read so that the source's relevance concerning the problem could be assessed. If a source was relevant, then credibility was assessed. Peer-reviewed sources and official reports from large organizations were considered the most credible. In the absence of such sources, less credible sources were used, including master's theses and news articles. Finally, the information from the most credible source was extracted in case of conflicting information. The project is pervasive and more immense than it is possible to complete with the available resources, especially in terms of time and human resources.

Flexible working methods have been used in preparing the concepts before the experiments. It is about continuous iteration through the project, which means repeating a particular process several times. This gradual procedure involves undergoing various changes and adaptations along the way. The process is often divided into several work blocks where one in each block repeats the same activities, such as planning and analysis [9]. Iterative usage of creative design methods, such as early prototyping and problem reframing, was a vital tool for innovation and problem-solving, known as the design thinking process. In order to arrive at a final working concept, it was relevant to evaluate possible solutions several times to see what was suitable. This method allowed testing and comparing different ideas to achieve the best possible result. This study was also experimental, which means the experimental method was used. It involves manipulating one variable while studying the others and seeing if any changes are happening simultaneously and if they have any correlation. This method relies on controlled methods, random assignment and the manipulation of variables to test a hypothesis.

This thesis aimed to develop a full-scale product (remote control system) that could be tested and act as a proof-of-concept. Therefore, the work implies concentrating on a fully working concept, targeting a professional end-user. Thus, it is worth noting that both the methods and results are practical and concentrated on the entire system functionality and user experience rather than each part's theory and high-end solutions. In addition, because the system was developed on an already built vehicle hardware and software stack, the variety of choices and information was somewhat limited.

1.4 Contribution

This thesis and its experiments construct a system that can remote control a full-scale car through the 5G cellular network. Much of the technology and theory used for building the remote control operating system is widely known. Nevertheless, the system in its entirety is unique and similar projects have rarely been built before in full scale. The evaluation of the project delivered many good results and was one of the first of its kind tested in Norway, see chapter 5. By showing the feasibility of the system and proof-of-concept, the thesis contributes to knowledge mainly with the following aspects and methods:

- Low latency remote control system through 4G/5G with possibility for future operator assistance as a complete and working system
- Local and global data flow through CAN Bus and ROS.
- Object detection with Yolo v3 and v5 in real-time.
- Automated P2P WireGuard VPN connection over 5G and 4G cellular network.
- PID controller for precise by wire manoeuvrability, with Logitech G29 steering wheel and pedals.
- Automated system of several interconnected ROS nodes and ".launch" files for remote control of the vehicle over IP.
- Video Transmission with ROS over IP and H.264 compression of 4 GMSL cameras.
- Desktop app in Kivy displaying real-time data and video streams in a clean and eye-catching graphical user interface.

1.5 Thesis Overview

This section contains a short description of the chapters in this report:

1. **Introduction**
   The introduction to the project covers the motivation, goals and research method of the thesis. In addition, the goals and research challenges are set and discussed here.

2. **Background Theory**
   Background theory includes the essential theory and information needed to understand the project in the different disciplines it crosses. The chapter also describes the hardware components and the software stack of the NAPLab car and the whole system.

3. **Methods**
   This part of the report will present methods to answer and fulfil the goals in section 1.2. It mainly includes the implementation and discussion of all parts constituting the remote control system developed for this thesis. In addition, at the end of every section is an architecture overview with graphs and diagrams corresponding to each part. All sections together form the entire remote control system, and it is worth noting that all architectural overviews have a different level of abstraction of the system.
   - Connection and Communication Between Car and Operator
   - Vehicle Control System
   - Real-Time Video Transmission Over IP
   - Object Detection in Real-Time
   - Graphical User Interface

4. **Results**
   Results were collected during the development, implementation and testing of all system parts. They are all presented with respect to the goals and research challenges earlier determined. The Results are also showed in a video uploaded to Youtube, can be seen here [10].

5. **Discussion**
   Evaluation and discussions of the results from chapter 5, along with suggested improvements for the system.

6. **Conclusion**
   The conclusion of the project

Chapter 2

Background

This chapter will explain the background for this thesis in-depth, the potential of autonomous vehicles and their significant vulnerabilities. In addition, a possible solution for some of the challenges will be proposed.

2.1 Potential and Challenges In Autonomous Driving

2.1.1 Potential of Autonomous Vehicles

The promise of autonomous technology has long been tempting. Potentially it can transform our experience of travel, remove people from high-risk working environments, help with transport and streamline our industries. This technology can be the key to building future cities where our reliance and relationship with vehicles are redefined. In addition, it can lower carbon emissions and pave the way for more sustainable ways of living. Only by reducing the time a car uses for parking search could achieve 5-11% emission reduction, stated by "Brown A [11]. Travel could also become much safer than it is today. According to the World Health Organization, more than 1.3 million people die each year due to road traffic incidents, and 94% of crashes involve human error as a contributing factor [12]. However, for autonomous vehicles to become mainstream, much needs to be changed, developed and tested. [13]

2.1.2 Challenges in Autonomous Driving

The biggest problem for the engineers in the autonomous technology industry is getting the cars to safely and effectively operate in complex and unpredictable human environments. Because not every human driver will behave calmly, serenely and understandable for the autonomous vehicle. Artificial intelligence software development for object detection and manoeuvring is also far from finished. [13]

Another major challenge for fully autonomous vehicles is navigating in bad and especially snowy weather conditions. Snow can clog the sensors or confound crucial sensor data. As a result, the vehicle can lose its depth vision, the ability to detect obstacles or, for example, keep on the right side of the road [14]. Most testing of driverless cars until now has been in sunny, dry climates, but for the technology to be helpful in all climates and conditions - that will have to change. A spokesperson for Argo, a company backed by Ford, said that their technology will handle a light rain, but "for heavier rains and snow, there still needs to be advancements in both hardware and software." [15] It is clear how the winter weather is increasing the complexity of developing an automation technology and that we need more testing in ice-weather conditions.

Autonomous cars will inevitably end up in a complex situation where some sensor information is wrong, limited, or something unreasonable happens on the road, leading the vehicle to a possible "deadlock". A deadlock is a situation where no progress is possible because two or more competing actions are waiting for the other to finish [16]. Such situations where the car cannot make decisions independently will need human intervention, which entails a driver taking control of the vehicle until the car can drive autonomously again. Of course, this is not an issue when a driver is present in the car, and the driver can take control of the vehicle when needed. On the other hand, when the car is completely driverless and fully autonomous, a "deadlock" can inflict everything from a traffic jam to human injuries and fatal consequences. An example of that can be the incident in Arizona, where an autonomous Waymo tried to run away from its support crew after getting stuck in traffic before completely blocking a three-lane highway [17]. This and many other incidents show that an autonomous car these days can not be wholly left to itself and still needs human support [18]. Waymo is still using safety drivers for many of its tests in San Francisco, so cracking those parts of the puzzle can be a significant step in the coming years.

2.1.3 Possible Solution to The "Dead-Lock" Problem

Cooperative driving, where multiple vehicles are automatically and remotely operated through low-latency communication, has recently caught much attention. That is mainly because of the ongoing build-up around the next-generation cellular network - 5G[19]. Cooperative driving itself can be defined as "Systems that provide a collection of sensor information, mediation, and control of wide-spread multiple vehicles with ultra-low latency" [20]. It starts to be interesting because 5G can offer much lower latency than its predecessor and even more bandwidth and speed, which will infer a stable cooperative driving system for the first time. Although self-autonomous vehicles are required to manage cooperative driving safely, and the system will enable self-autonomous vehicles to acquire sensor information from peripheral vehicles, it is still somewhat tricky for distributed control by autonomous driving vehicles, at least in the phase of development, to achieve cooperative driving in scale.

A possible solution to the deadlock situation as described in subsection 2.1.2 can be an integration of a remote control system as an indispensable part of every AV system. By letting a human operator control the car remotely, such edge cases and deadlocks no longer have to be a limitation or threat for self-driving cars. That makes integrating self-driving cars onto the roads and into cooperative driving even before the cars are fully automated, without requiring a safety driver in the vehicle. It is worth mentioning that to run a car safely over telemetry and with low network latency, a 5G cellular network would probably be needed.

In addition, an operator-assistance system is suggested as part of the remote control system, such that any possible mistake made by the operator would not lead to a fatal accident. An AI should accomplish this assistance with object detection by helping the operator in all situations and working seamlessly together. For example, when burst packet loss occurs, a remote control system tends to collapse or be interrupted, which can be highly undesirable at high speed. In such cases, the control would be carefully taken over by the operator's assistant, waiting for the situation to stabilize before returning the control to the operator. That leads to remote control of AV through a 5G network with an integrated operator-assistance system, where AI and operator work together.

In this paper, the remote control will be further defined as the control system by which external equipment, such as a server or computer, with two-way communication, collects sensor information from a vehicle and seamlessly controls it.

2.1.4 Connection and Communication Between Car and Operator

Remote-controlled vehicles with teleoperating systems for steering first appeared around 1900, but they were not commonly used before the 1970s. The application for teleoperating vehicles has different purposes, such as inspecting potentially dangerous environments or exploring places hard to reach, such as space or other planets [21]. However, for these purposes, there is often no need to consider other vehicles; therefore, communication quality and latency is not as critical as in traffic situations where the reaction time must be as short as possible. As already discussed in subsection 2.1.3, teleoperation can be used for controlling autonomous vehicles when edge cases occur. Such telemetry requires real-time information about traffic situations, status and control of the car. Measurements done by M. Claypool and D. Finkel in 2014 [3] on the subjective and objective effects of latency in cloud-based games show that both user performance and quality of experience degrade linearly with an increase in latency. Therefore, the remote controller must retain all necessary information and data over a secure, stable, high-performing communication channel, with latency lower than 50ms [22] for accurate manoeuvring and satisfying operator experience. Figure 2.1 shows a comparison between the impact of different latency on a car driving at 60 km/h.

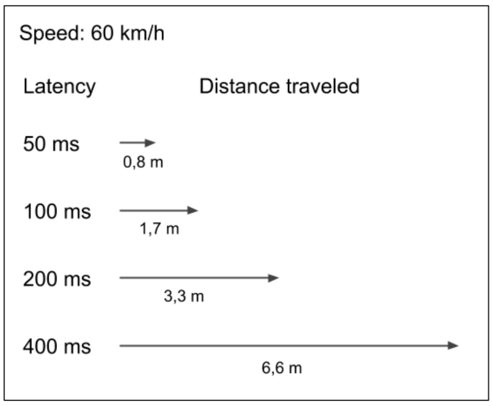


Figure 2.1: Illustration showing the correlation between vehicle's travelled distance and latency with a driving speed of 60km/h. Source of image [23]

Projection and future of driverless cars

Several analysts predict that SAE Level 5 vehicles will only be in great use in a decade [24]. Global data has forecasted the expected number of cars, SAE levels 4 and 5, to be 2.3 million cars produced by the year 2033/2034 [25], shown in Figure 2.2. The forecast for 2020 is more conservative than 2019, reflecting the industry, which has generally had a less optimistic view of the time perspective for vehicles of level 5 [26]. However, if new challenges related to level 5 emerge, the current forecast may become even more conservative, and the future with level 5 vehicles may be even further away.

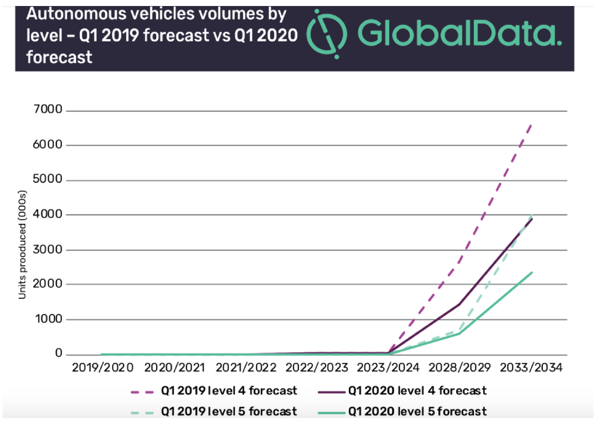


Figure 2.2: Forecast of the number of SAE level 4 and 5 vehicles produced from 2019[25]

Chapter 3

Theory

This chapter will go through the most important theory, information, and terminology needed to understand the project in the different disciplines it crosses. The theory covered here is closely related to the implementation of the remote control system developed for the thesis and general knowledge about autonomous vehicles and subsequent terminology. The chapter also describes hardware components and the architecture of the NAPLab car, in addition to the software stack used in implementing the remote control system. The NAP-lab car will, in this text, further be referenced as the car and the remote control system as the system developed for this thesis.

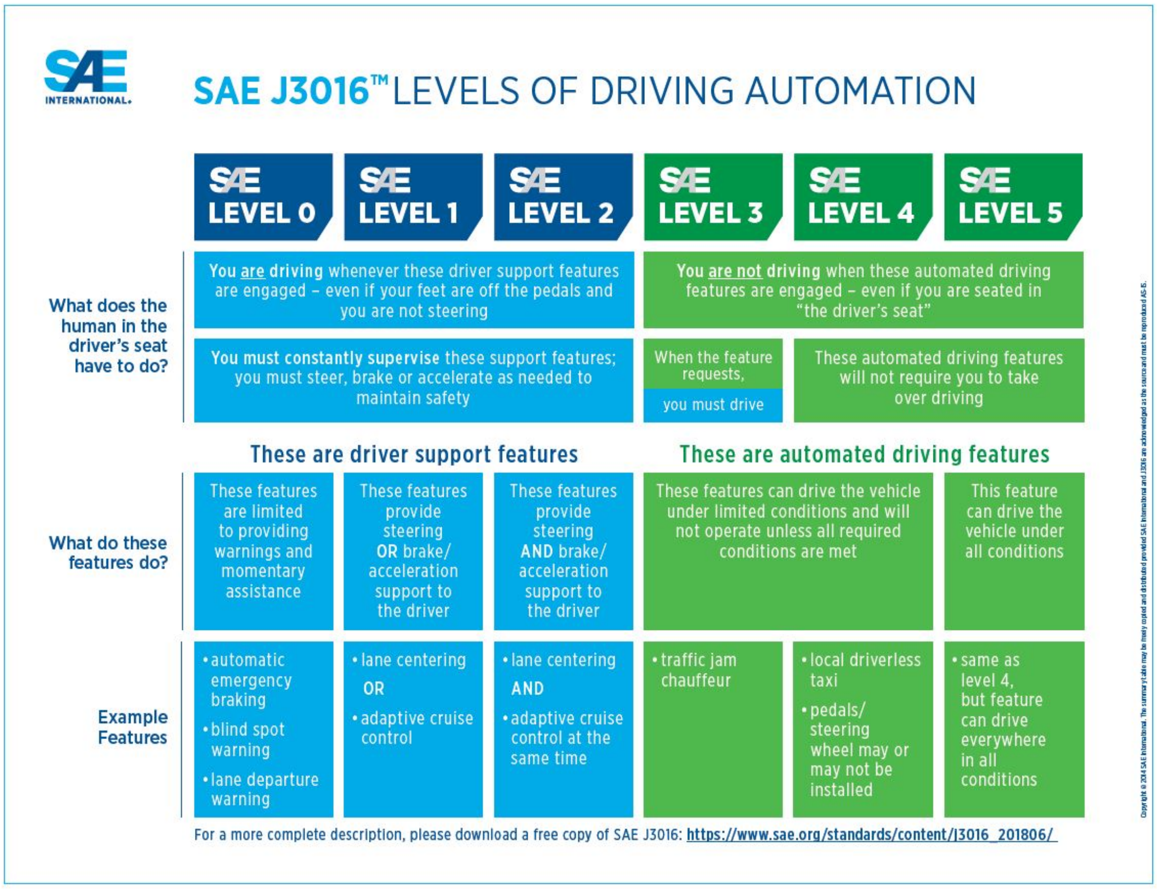
3.1 Autonomous and Cooperative Driving

An autonomous vehicle (AV), also known as a self-driving car, driver-less or robotic car, is a vehicle incorporating vehicular automation, that is, a ground vehicle capable of perceiving its environment while moving safely with little or no human input at all, depending on the autonomous level of the agent. By combining various sensors, the AV can sense its surroundings; such sensors can be radars, lidars, thermographic cameras, image cameras, odometry, sonar, GNSS and inertial measurement units. In addition, advanced control and computing systems can interpret sensory information to identify appropriate navigation paths, relevant signage and possible obstacles. A self-driving car is usually developed with five main components in mind, where all of which are equally important to make the car fully autonomous [27]:

- Computer vision - how the car sees
- Sensor fusion - how the car understands its environment
- Localization - how the car knows its location
- Path planning - how the car plans the most optimal route
- Control - how the car steers the wheel and pedals

AV is often divided into six levels according to a system developed by SAE International (SAE J3016, revised and updated periodically) to categorize the state of how comprehensive an autonomous agent is. The SAE levels are illustrated in Figure 3.1 and also roughly described below: [28]:

- Level 0 - no automation
- Level 1 - hands-on/shared control
- Level 2 - hands off
- Level 3 - eyes off
- Level 4 - mind off
- Level 5 - steering wheel optional



SAE INTERNATIONAL.

SAE J3016™ LEVELS OF DRIVING AUTOMATION

	SAE LEVEL 0	SAE LEVEL 1	SAE LEVEL 2	SAE LEVEL 3	SAE LEVEL 4	SAE LEVEL 5
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

For a more complete description, please download a free copy of SAE J3016: https://www.sae.org/standards/content/J3016_201806/

Figure 3.1: Figure illustrates the different SAE level that categorize the autonomy state of a vehicle. Source: SAE website[28]

In order to enable a car to travel without any driver embedded within the vehicle but still have human control over telemetry, as this project was aiming at, it can no longer be called a fully autonomous vehicle according to SAE.

If driving automation systems can perform their functions independently and self-sufficiently, they may be autonomous. However, if they depend on communication or cooperation with some outside entities, for example, remote control operators, they should be considered cooperative rather than autonomous [28]. Back in 2003, professors at Nagoya University [21] discussed the cooperative type of driving and believed that cooperative driving is the ultimate behaviour in traffic were not only remote control available but mainly cooperation between cars and agents on the road. They stated that telematics opens the path to an application domain - cooperative driving by extending the onboard sensors, permitting communication intentions, and facilitating road courtesy. This means gathering information and sharing it with other drivers and operators to adapt to the surrounding traffic and environmental conditions and obey traffic rules and regulations. According to the professors, the most important goal of cooperative driving was to increase traffic safety and flow. [21]. It is important to point out that this can also be used to describe cooperative systems like the one in this thesis, where the remote control system should work together with an autonomous system in the vehicle, making it a cooperative driving system.

3.2 Next Gen. 5G Cellular Network

5G is the fifth generation of the world's wireless network technology, the descendant of the earlier but still mainly used 4G technology. Shortly speaking, 5G is a collection of different technologies and updates that delivers ultra-low-latency with an ultra-reliable and ultra-fast signal with a potential of being 10-100 times faster than the previous standard. That is achieved using higher spectral efficiency, operating at frequencies of about 28 GHz and 39GHz, using advanced mobile technology and newer network architecture, such as Edge Computing. 5G can handle bandwidth as high as 20 gigabytes per second (Gbps), whereas the previous generation is limited to speeds of around 100 megabits per second (Mbps). [29][30]

In ways of radio technology, it is still important to notice that 5G is very similar to its predecessor and uses much of the same methods. Like other cellular networks, 5G networks use a system of cell sites. Transceivers in each cell divide their territory into different sectors and send encoded data through radio waves. For this to work, each cell site must be connected to a network backbone, whether done by wire or with a wireless backhaul connection. On top of that, 5G changes how data is encoded, offering the carriers more options in terms of airwaves to use. [30]

With this technology, new forms of vehicle-to-vehicle (V2V) communications can already become available. For example, two cars approaching each other from different directions would let their onboard computing units determine which vehicle should yield for the other at the location where their paths cross. With such communications, vehicle-to-everything (V2X) will also become available; cars will be able to communicate with objects, pedestrians and traffic management systems at intersections.

Remote control of cars in real-time would also be an easy task and will allow for great advances in both safety and usability of autonomous cars.
Cars will be able to sense hazards farther ahead and use automated systems to apply brakes, throttle, steering or, when necessary, let a remote operator take control of the vehicle seamlessly. [21]

One major drawback of 5G is that it achieves its speed with millimetre waves, which max out at a few hundred meters and are quickly stopped by obstacles and even heavy rain. This means the 5G network needs more infrastructure to cover the cities than the lower band 4G. In addition, the latency of the network is highly dependent on the processing speed that happens at the base stations or a central server. If the network uses old methods in the way that data is first sent to a centralized computing server like in Norway at the time this paper is written, the latency will still be significantly higher than the one 5G can achieve. Telia and Telenor are building the 5G cellular network in Norway. Both of them are routing their data through a server in Oslo, meaning that the signal must travel first to Oslo and then to where the end-user is located. That process can eat up the low latency promised by 5G. The result is that the processing needs to be closer to the end-user, for example, with edge-computing, and not placed far away to minimize the latency with 5G. [29] [30]

Advantages and disadvantages of 5G summarized:

- Advantages:
  - Speed
  - Security
  - New generation of AI- and machine learning-based services
  - Capacity
  - Low latency - 5G will significantly reduce the time it takes for network devices to respond to commands. Compared with 4G which latency ranges from approximately 60ms - 98 ms according to [31] and 30ms-50ms in Norway [29]. 5G can reduce latency to less than 5 ms, but the ultimate target is to go under 2 ms [19].
- Disadvantages:
  - Uneven coverage
  - Need a lot of transceivers(Line of sight)
  - Need a lot of computing power
  - Capex/Opex
  - IOT Security

3.3 VPN

Since the early start of the World Wide Web, there has been a goal to protect and encrypt internet browser data. During the 1960s, the US Department of Defense was involved in projects with encryption of internet communication data, and the battle for safe browsing is still ongoing. The somewhat old but still the de facto standard TCP/IP protocol has four levels: Link, network, transport and application. At the network level, local networks and devices can be connected to the universal network – where the risk of exposure is evident. A team from Columbia University and ATT Bell Labs succeeded in 1993 in creating an early version of the modern VPN known as swIPe: Software IP encryption protocol. That was the start of a possible solution to a secure gateway to the World Wide Web.

A VPN establishes a secure connection between a client and the internet, often called a tunnel. Via the VPN, all data traffic from and to the client is routed through an encrypted virtual tunnel. This will disguise the client's IP address for the rest of the internet, making its actual location invisible to everyone. The VPN tunnel uses different encryption methods to reduce the risk of data leakage; therefore, only accessible by the client itself and will be secure against external attacks. This means that even the user's Internet Service Provider (ISP) and other third parties cannot see which websites the user is visiting or what data is sent and received online. It is like a filter that turns all data into "gibberish", and even if someone does intercept the stream and get their hands onto the user's data, it would be useless as it needs to be decrypted with the client's private keys (Public-key cryptography). Without the key, it could take millions of years for a computer these days to decipher the code with a brute force attack.

3.4 Hardware and Architecture of NAPLab Car

Architecture exploration was done by searching and investigating the documentation of the different components and parts, in addition to the car itself. Testing and exploring the equipment with an experimental research method were also done. It is worth mentioning that this study will not cover all sensors and parts in-depth, but primarily those relevant for further research and development based on the goals from section 1.2. The overview of the whole system is illustrated in Figure 3.4.

3.4.1 NAP-Lab car

Sensors allow vehicles to see and sense everything on the road and collect vital information for safe driving. Path-building from one point to another, for example, is made based on the sensor data, such as location and surroundings. Furthermore, all this information is processed and analyzed by the car's computer, in real-time, like an infinite loop. Most of today's autonomous car manufacturers commonly use cameras, radars, and lidars as the three primary sensors in autonomous

(a) Front view of NAP-Lab car **(b)** Rear view of NAP-Lab car

Figure 3.2: Overview of NAP-Lab car

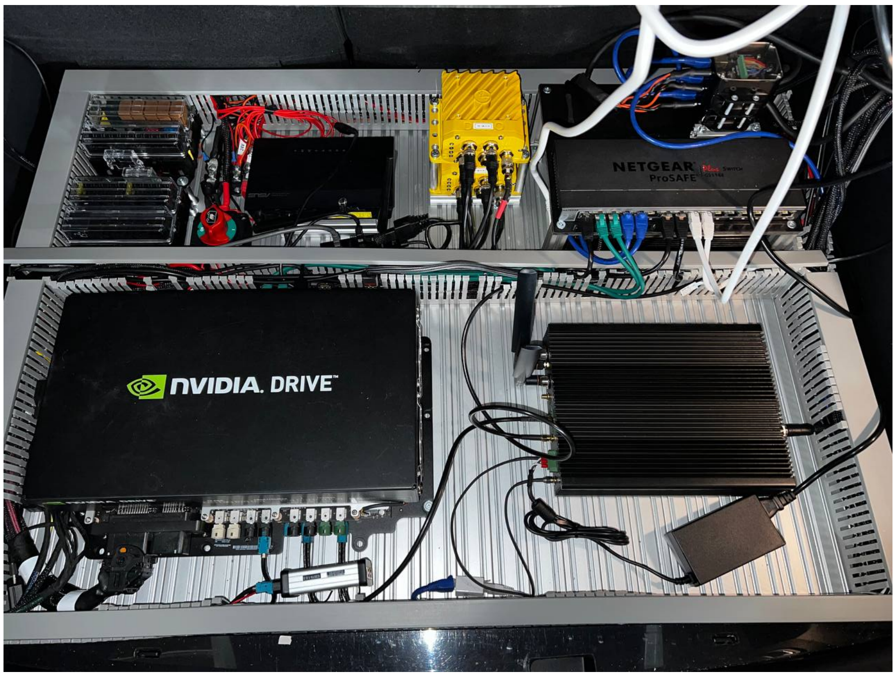


Figure 3.3: Figure illustrates an overview of the trunk area with the mounted equipment. Nvidia Drive on the left, Arcus Router on the right, GNSS receivers and switch in the back.

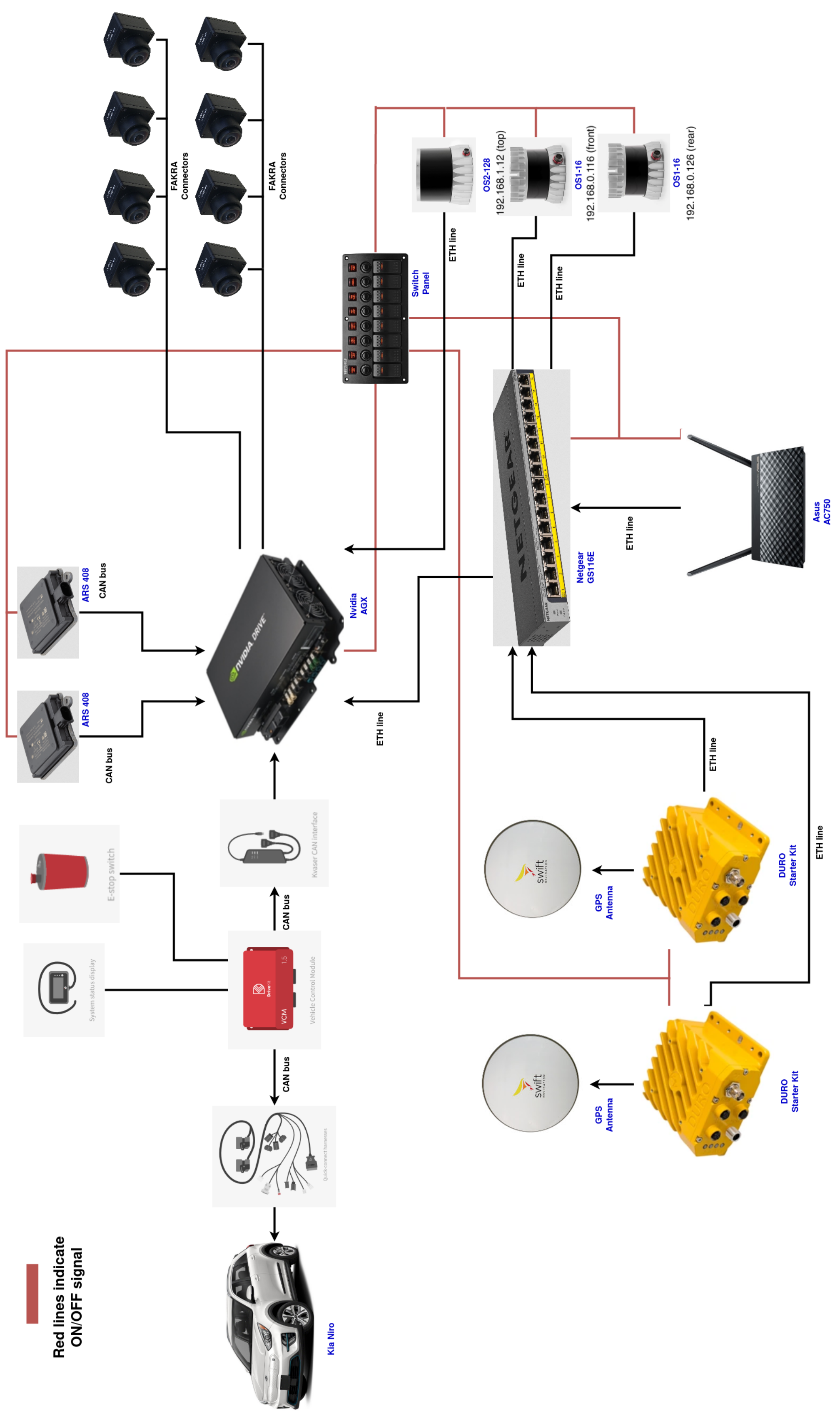


Figure 3.4: Figure illustrates an overview over the architecture of the upgraded Kia Niro by NAPLab. The Asus router illustrated is old setup, and is now changed to Arcus 5G router

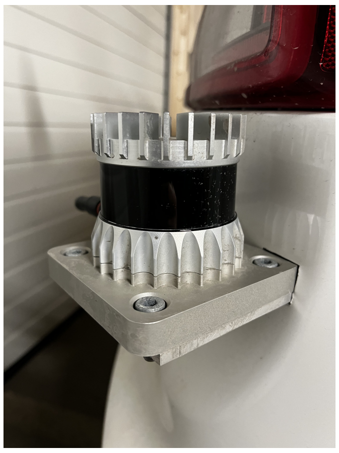
vehicles[32]. As mentioned, the car used in this study is a Kia Niro, modified by the research group NAP-Lab. It contains many of the most critical sensors, computers and kits to drive fully autonomous. For the project of making the remote control system, the most fundamental parts are the image sensor - camera, the 5G router, vehicle control module, the computing unit and the further development of the assistance system, the lidar and radar. The reasons behind it and the discussion of why are described in chapter 4.

3.4.2 Lidar

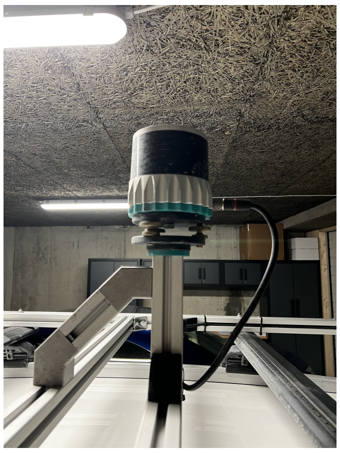
On the car, three lidars by the company Ouster are present, two of which are smaller with only 16 channels or beams (OS1-16) and one big with 128 beams (OS2-128). One of the small lidars is mounted into the front of the car, see Figure 4.24a, to see the object, especially people coming close to the car. The second of the two small lidars is located on the rear right side; see Figure 4.24b. This one will detect people or cars approaching from the right side and behind the vehicle. Since cars in Norway are driving on the right side of the road, it is more likely that pedestrians and sidewalks will be to the right of the car, making that side more important to monitorize. The last, big lidar, is mounted on top of the roof, shown in Figure 4.24a and has the mission of holding an overview of the surroundings. With its 128 beams, it is stronger and does collect data with higher resolution. The range of the lidars can be up to 260 meters for the top one and 50 meters for the smaller one. [33]

The connection between each of the smaller 16 beams lidar and the heart of this car's autonomous system, the Nvidia Drive (AGX Xavier), is set up with a 100Gb switch. The bigger one OS2-128 beams lidar is connected directly to the Nvidia AGX unit, as shown Figure 3.4. The data in point cloud packets are sent via UDP protocol. Incoming packets are handled by the ROS nodes written by the producer, Ouster. The LiDAR recording format is DriveWorks specific. When using the recording tool provided by Nvidia, it will create a DW LiDAR recorded file. This data can be used directly or stored and later analyzed through the Ouster Studio software.

(a) Front lidar



(b) Right side lidar



(c) Top lidar

Figure 3.5: A figure showing the different lidars installed.

3.4.3 Camera

The Kia Niro is also loaded with cameras. In total, there are 8 GMSL cameras of two types mounted onto the car: Sekonix Nvidia Drive Camera SF3325 with FOV 60° and SF3324 with FOV 120°, both having a 1928 x 1208 resolution, RGB sensor with 12-bit depth. They are mounted all around the car in order to cover all points of view. Three in front, two on each side mirror, pointing in different directions and one in the back, can be seen in Figure 3.6 and on the **??**. The video format when recording from each of the cameras is h.264. They are connected through a Fakra port and in separate channels, recording a separate video.

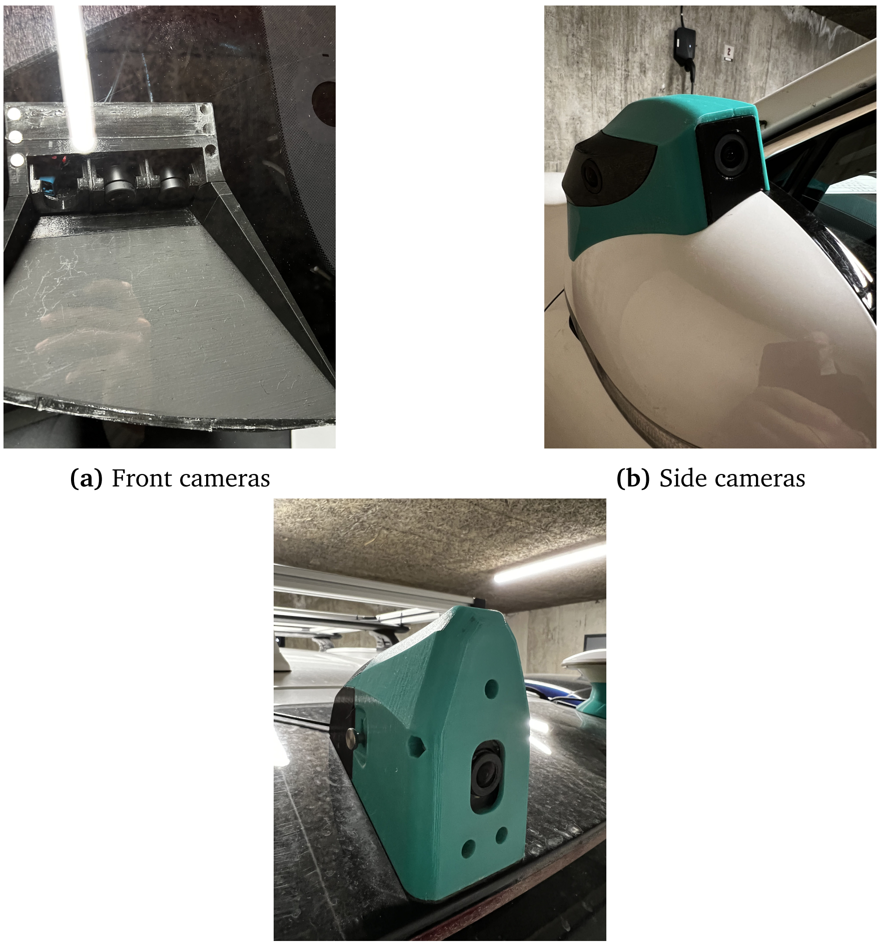

(a) Front cameras (b) Side cameras (c) Rear camera

Figure 3.6: A figure shows the different cameras.

3.4.4 Vehicle Control Module - Drive Kit

Another essential part of an autonomous car is the control and drive unit. The one that manoeuvres and controls the car's throttle, brakes and steering. In this car, a drive kit, OSCC by PolySync, is installed and connected to all mentioned parts by a CAN bus. ROS subscribers, publishers, and nodes can get commands from the central computer, the Nvidia Drive. The car did have a pre-installed feature with the possibility of controlling the car with an Xbox one controller that was directly connected to Nvidia Drive through a USB. The Xbox controller was made as a "ROS joy node", the Nvidia drive as a publisher and the drive kit as a subscriber. The current part is in-depth in **??**.

3.4.5 Radar and GNSS

The radar attached to the car is Continental ARS 408-21 Long Range 77Ghz. There are two of them, one in front and one in the rear. Continental ARS 408-21 77GHz is a long-range millimetre-wave radar sensor. The car is also equipped with two Swift GNSS GPS/GLONASS/Galileo/BeiDou mini-survey antennas on the roof for localization, see in Figure 3.4.

3.4.6 Router

The car is equipped with Arcus 5G router by Celerway. It is a powerful 5G and 4G LTE, modular ruggedized router, enabling up to 7 simultaneous WAN connections, of which three are cellular. Its main objective is to enable high-capacity connectivity on the move or outdoors and to promote performance for latency-sensitive applications. The router supports a maximum load balancing capacity of 990Mbps and 800+Mbps using VPN encryption. This makes the Arcus router especially suited for installation in vehicles, trains, ships, proven in multiple transportation scenarios [34]. The router also features direct VPN connections through Open VPN, IPSec, WireGuard, Celerway Phantom and more. It was installed explicitly for this project mainly because of its 5G cellular compatibility, and the possibility of creating direct VPN channels with low latency [35].

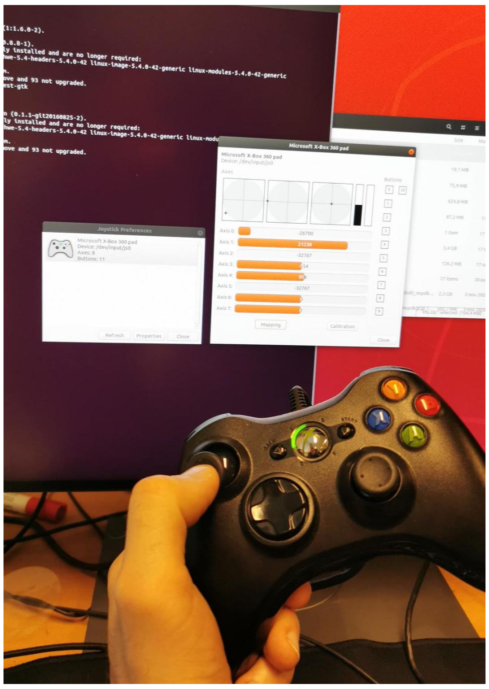
The cellular 4G/5G signal is received by four antennas of the type Panorama EF-6-60 attached to the front window and on the small rear windows in the back, see Figure 3.8. The antenna covers the frequency bands 617-960 / 1427-6000 MHz, which means all 4G and 5G bands. The reason for installing four antennas rather than just one is to increase the bandwidth and the reliability of the signal, [36].

3.4.7 Steering Wheel and joystick

In order to remote control the car, a joystick or steering wheel is needed. For this project, a Logitech g29 PS3/PS4 wheel was used together with pedals connected through USB on the remote client machine. An Xbox One/360 joystick was also used in the early phase of the testing, both connected to the car and the remote client machine.

Some specification of the g29 set:

- Rotation: 900 degrees lock-to-lock
- Hall-effect steering sensor
- Dual-Motor Force Feedback
- Nonlinear brake pedal
- Self-calibrating


(a) Setup of the Xbox controller

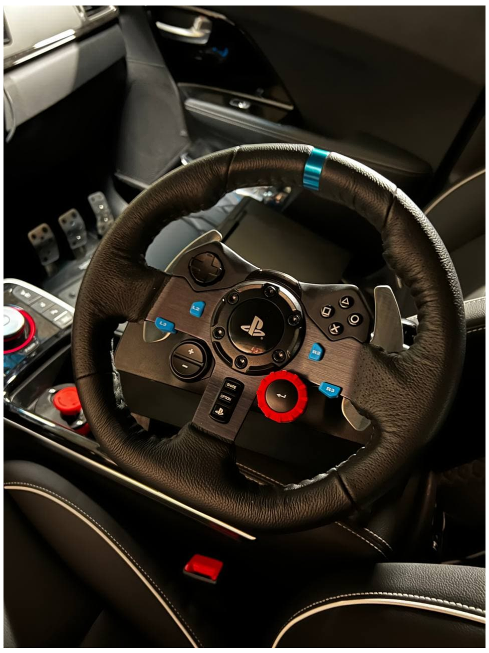

(b) Setup of the Logitech g29 steering wheel in the car.

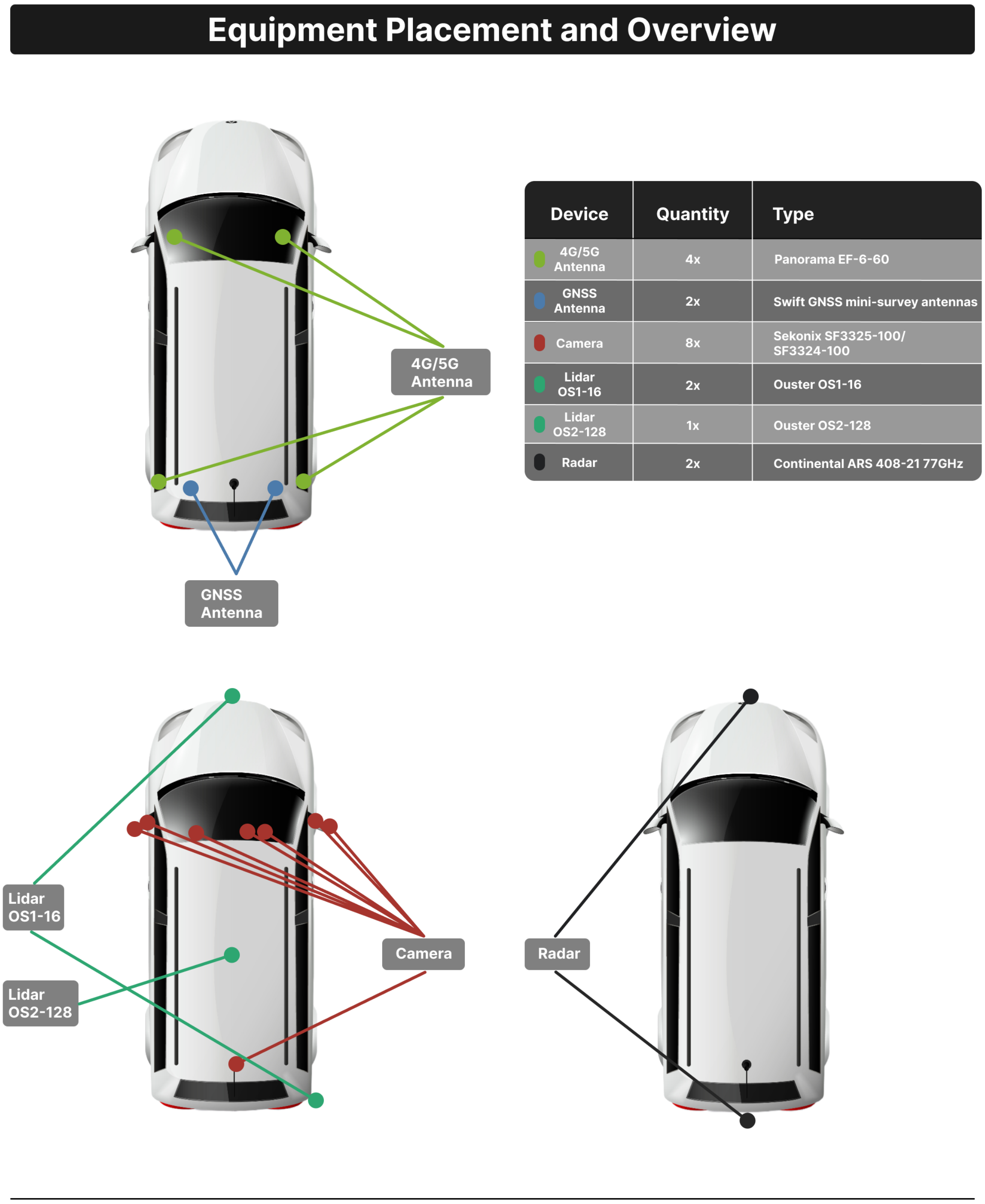
Figure 3.7: A figure showing the setup of the remote controllers.

3.5 Software

3.5.1 Nvidia Drive SDK - Drive OS and DriveWorks

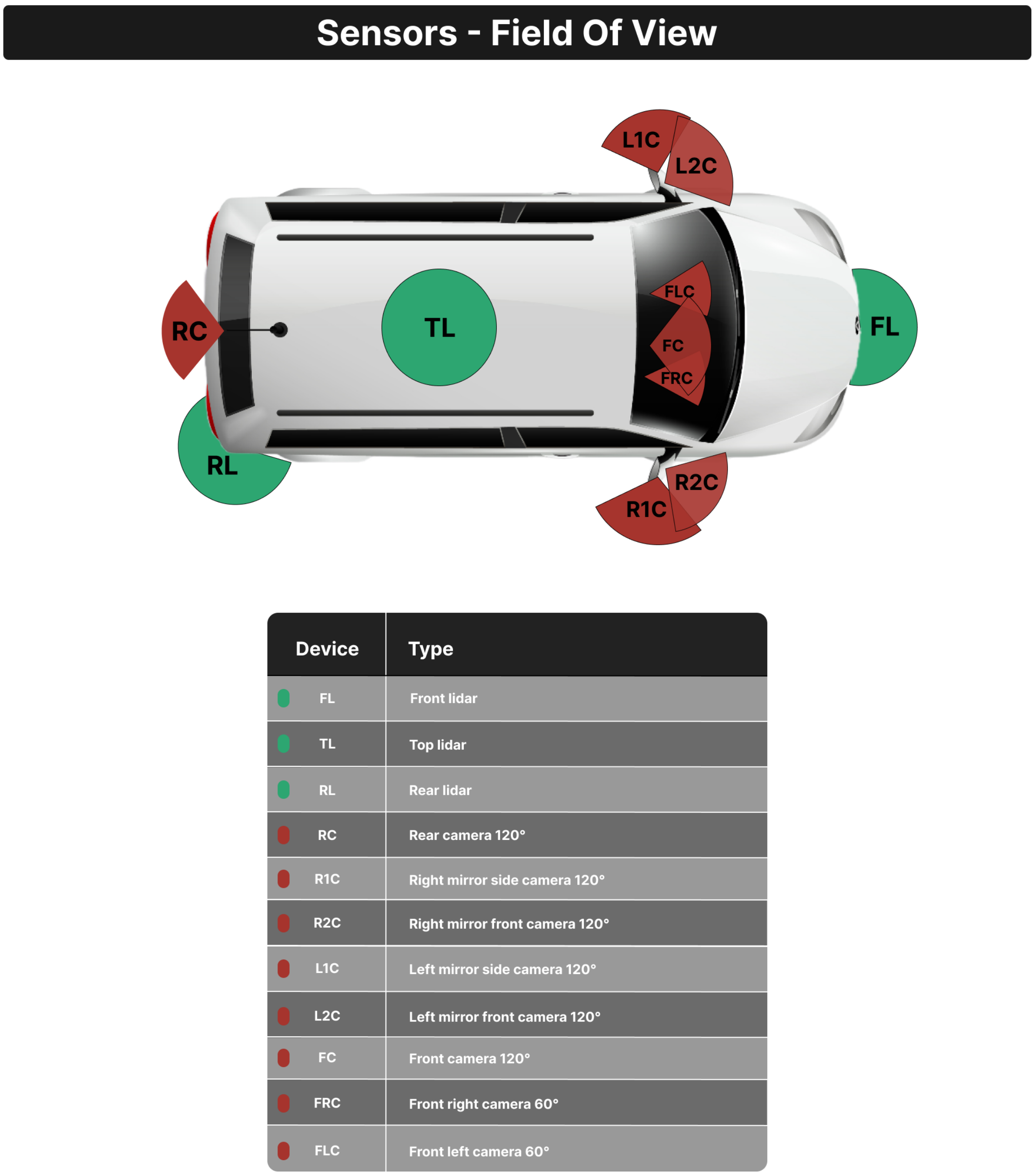
"The open Nvidia Drive SDK gives developers all the building blocks and algorithmic stacks needed for autonomous driving. It empowers developers to build and deploy a variety of state-of-the-art AV applications more efficiently, including perception, localization and mapping, planning and control, driver monitoring, and natural language processing" this is the way Nvidia describes the Drive SDK on their website [37]. Making it possible to build and develop a self-driving car is the main objective of the SDK, but it is worth mentioning the fact that this is mainly made to be working together with Nvidia's Drive platform and hardware. The SDK consists of different parts, but the main ones used in the car are Drive OS and DriveWorks, installed on Nvidia Drive AGX Xavier. The stack is illustrated in Figure 3.10.

Nvidia Drive OS is a software stack acting as a foundation that consists of RTOS - embedded real-time operating system, Nvidia Hypervisor, Nvidia CUDA libraries, NVIDIA TensorRT, and other modules providing access to the hardware engines. Drive OS should offer a safe and secure execution environment for applications such as secure boot, security services, firewall, and over-the-air updates and is designed to develop and deploy autonomous vehicle applications. [38]



Device	Quantity	Type
🟩 4G/5G Antenna	4x	Panorama EF-6-60
🟦 GNSS Antenna	2x	Swift GNSS mini-survey antennas
🟥 Camera	8x	Sekonix SF3325-100/ SF3324-100
🟩 Lidar OS1-16	2x	Ouster OS1-16
🟩 Lidar OS2-128	1x	Ouster OS2-128
⬛ Radar	2x	Continental ARS 408-21 77GHz

Figure 3.8: Figure illustrates an overview over the type and placement of the equipment on the car. Different colors indicate different type of equipment.



Device	Type
🟩 FL	Front lidar
🟩 TL	Top lidar
🟩 RL	Rear lidar
🟥 RC	Rear camera 120°
🟥 R1C	Right mirror side camera 120°
🟥 R2C	Right mirror front camera 120°
🟥 L1C	Left mirror side camera 120°
🟥 L2C	Left mirror front camera 120°
🟥 FC	Front camera 120°
🟥 FRC	Front right camera 60°
🟥 FLC	Front left camera 60°

Figure 3.9: Figure illustrates the field of view of each camera and lidar, with respect to it placement. Different colors indicate different type of equipment.

The Nvidia DriveWorks SDK provides a large set of fundamental capabilities, including processing modules, tools and frameworks required for advanced AV development on the Nvidia Drive platform. DriveWorks sits on top of the Drive OS and gives a developer the right tools for developing a self-driving car. The purpose of the software is to offload the developers from spending time on basic low-level functionality. DriveWorks is modular, open, and readily customizable. That means a developer can use only one module within their software stack to achieve a specific function or multiple modules to accomplish a higher-level objective. The DriveWorks and Drive OS gives together a foundation for self-driving car development. [39]

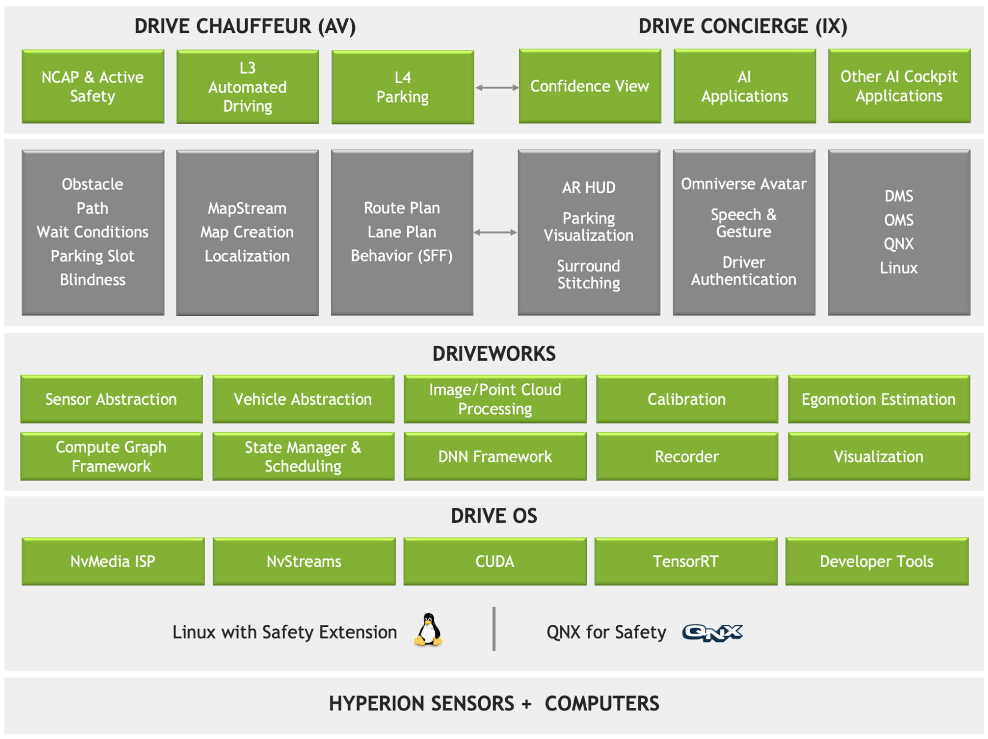


Figure 3.10: Figure illustrates the stack of Nvidia Drive SDK. Source of image [38]

3.5.2 Kivy

Kivy is an open-source and free Python multi-platform GUI development library and framework. It is made for developing desktop GUI, mobile apps and other multi-touch application software with a natural user interface. Kivy helps to develop applications that make use of innovative, multi-touch GUI. The fundamental idea behind the library is to enable the developer to build an app once and then use it across all other devices, making the code reusable and deployable. It does also allow for quick and easy interaction design and rapid prototyping. Kivy does not use CSS or HTML code structures or components, it has its way of functioning. However, Kivy can still be in many parts similar to web HTML and JavaScript based web frameworks. It can run on Android, iOS, Linux, macOS, and Windows and is s distributed under the terms of the MIT License. For this project, Kivy 2.1.0 with Python 3.8 was used.

3.5.3 OpenCV

OpenCV - an open-source computer vision and machine learning software library. It can be used in C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. It can also be compiled with support for CUDA, which can use the CUDA cores on the Nvidia Drive AGX or an Nvidia GPU to achieve GPU-accelerated functionality. The library can also be used for displaying, rendering and transforming images, which was its primary objective in this project. For this thesis, OpenCV 4.5.5 was used.

3.5.4 ROS

The Robot Operating System (ROS) is an open-source meta-operating system developed specifically for robotics. It is a set of software libraries and tools that helps build robot applications; it includes everything from drivers to algorithms and developer tools, such as libraries to visualize data and run code on multiple platforms, including embedded computers and micro-controllers. ROS implements low-level device control, hardware abstraction, message-passing between processes, implementation of commonly-used functionality, and package management. ROS functions as a peer-to-peer network of loosely coupled processes using the ROS communication infrastructure. It can be similar to how the MQTT protocol works with subscribers, publishers and topics. ROS delivers several different styles of communication, which include the synchronous RPC-style communication over services, asynchronous streaming of data through topics, and data storage on a Parameter Server.[40]

ROS comes in two main versions, ROS 1 and ROS 2. The two versions are almost like two different systems and do not share much of their functionality. ROS 1 Melodic was used for this project, even though a newer version was available and ROS 2 was better suited for real-time applications. But the reason was possible compatibility problems with earlier projects and software in the car and the fact that ROS 1 is now more widely used and has a lot more documentation than ROS 2. [40]

Core elements of the ROS system and extra ROS packages important for the project are listed below:

- **Catkin** is the successor to the original ROS build system, rosbuild, which is now the ROS's main and official build system.

A general build system is usually responsible for generating "targets" from any raw source code that an end-user can use. In ROS-specific terminology, source code is organized into "packages", where each package can consist of one or more targets when built. The catkin build system is specifically made for ROS but inherits much from CMake. On top of CMake's normal workflow, it combines CMake macros and Python scripts to provide some functionality. When Catkin was designed, it was planned to be more conventional than rosbuild, allowing for better cross-compiling support, better distribution of packages and better portability. The workflow of Catkin and CMake is very similar, but it does add support for automatic 'find package' infrastructure and support for building multiple, dependent projects simultaneously. [41]

- **Packages** organize the software of ROS. It might contain the ROS run-time processes called nodes, libraries, a third-party piece of software, or anything seen as a useful module usefully organized together. The goal of these packages is to provide functionality in an easy-to-consume manner so that others can easily reuse software. ROS packages can be seen to follow a "Goldilocks" principle in general, meaning that it has enough functionality to be useful but not so much that the package is difficult for other programmers to understand. Packages can be created by hand or by tools like catkin_create_pkg. [42]

- **Nodes** are ROS run-time processes that perform computation and can publish and subscribe to topics for inter-node communication. A full ROS system usually has many different nodes, often started with roslaunch with multiple parameters nodes doing specific tasks. [43]

- **Master** is responsible for name registration and lookup to the rest of the nodes in the ROS system. It can run across multiple machines for communication via Ethernet or WiFi and does so that the nodes can find each other and send messages. It tracks subscribers and publishers to the different topics as well as services. The main role of the ROS master is to make it possible for ROS nodes to locate each other in the network. It could look like an MQTT broker, but it is more like a DNS server for lookup and localization. Once these nodes have located each other, the further communication is in a peer-to-peer fashion, while in MQTT, it still goes through the broker. [44]

- **Messages** in ROS are, in general, a way to standardize communication between nodes. There are many standard message formats for different needs, such as int, float, sensors, and navigation. ROS makes it also possible to have custom formats of messages; an example is illustrated in Figure 3.11a where a Joy message of the type sensor_msgs is shown. [45]

- **Topics** are channels exchanging messages between nodes. The messaging is made in a publisher and subscriber fashion over the given topic. A node can publish a message on a given topic, and all nodes subscribed to it will receive the message. The topic has an identifier that allows other nodes through the ROS master to identify and subscribe to it. In general, nodes do not get the information about whom they are communicating with because they are intended for unidirectional streaming communication. However, via the ros master, it is possible to get information about who is subscribing and publishing to a topic. It is worth mentioning that ROS currently supports mostly TCP/IP-based and some UDP-based message transport but only in roscpp. ROS streams message data over persistent TCP/IP connections and is known as TCPROS. [46]

- **Roslaunch** is a simple but powerful tool for launching multiple ROS nodes locally and remotely, for example, via SSH, and setting parameters for specific nodes on the Parameter Server under the launch. It also includes options to automatically re-spawn already dead processes. By taking one or more XML configuration files with the .launch extension, roslaunch can specify what parameters to set and what nodes to launch, as well as the machines on which they should be run onto. This package was used to start multiple local and remote nodes, discussed in depth in chapter 4. [47]

- **Rqt** is a plotting data tool made to automatically display the connections between nodes. [40]

- **Rosbags** are used for saving ROS message data from topics, with a possibility of playing it back. [40]

- **Rviz** is a tool for visualizing ROS messages and transformations and can be used to visualize radar scans, laser scans, maps, paths, current positions and orientations, to mention a few. [40]

- **sensor_msgs** is a package that defines messages for commonly used sensors, including cameras and scanning laser rangefinders. Inside this package, it is possible to find raw message definitions for many different sensors and equipment, such as for sending images or, as an example, the Joy message used in this project for interacting with a joystick or steering wheel; the definition can be seen illustrated in Figure 3.11a. [48]

- **image_transport** is a package that uses the prior sensor_msgs providing transparent support for transporting images in low-bandwidth compressed formats. Specialized transport strategies, such as image compression or streaming video codecs, are often preferred when working with images.

Image_transport is abstracting the complexity of compressing and decompressing and transporting images in arbitrary over-the-wire representations so that the developer only sees familiar sensor_msgs/Image messages. [49]

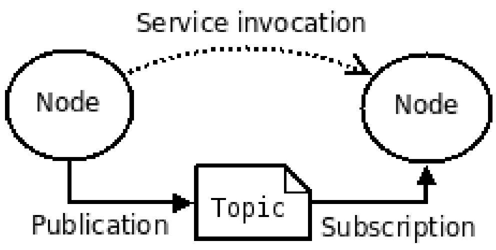
- **Joy and joystick_drivers** is the ROS driver for a generic joystick supported by Linux. The joy package contains joy_node, which interfaces a Linux joystick to ROS. This node publishes a sensor_msgs/Joy message on the topic "joy", which contains the current state of the joystick's buttons and axes, see Figure 3.11a. This node should work with any joystick that Linux supports. [50]

- **usb_cam** The usb_cam node interfaces with standard USB or built-in cameras using libusb_cam and publishes images as sensor_msgs/Image to different topics, based on the type of compression, for example,/raw or /theora, together with the use of the image_transport library to allow compressed image transport. In the project, this was mainly used on the operator/client computer to debug the GUI. [51]

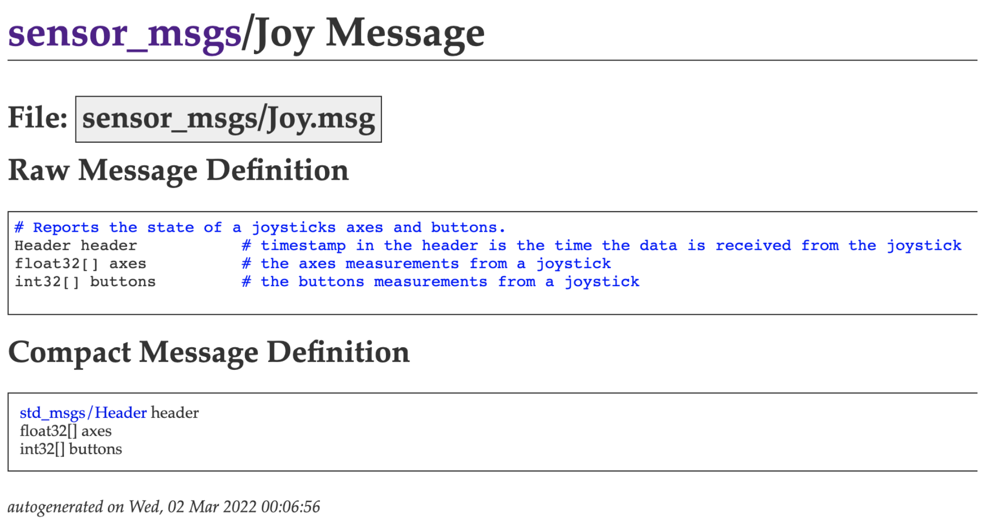
- **nvidia_gmsl_driver_ros** is a package made by UT-ADL of a university in Estonia; it is free to use and available on Github [**utAdl**]. The purpose of the package is to make it possible to take out camera images from the GMSl camera connected to Nvidia Drive and processed by DriveWorks and convert it into ros messages published on a topic in the form of sensor_msgs/Image. This was used in the project in order to have easy access to the cameras. [52]

- **h264_image_transport** is a H264 plugin for the ROS image transport package. This repo was created by UT-ADL to be used in pair with the prior vidia_gmsl_driver_ros mainly but can work standalone also. It decodes h264 encoded image packets published with the correct message type. [53]

- **image_view** is a simple viewer for ROS image topics. It allows viewing and visualizing a camera image published on a ROS topic. [54]



(a) Figure illustrates the main workflow of ROS nodes.



sensor_msgs/Joy Message

File: sensor_msgs/Joy.msg

Raw Message Definition

```
# Reports the state of a joysticks axes and buttons.
Header header           # timestamp in the header is the time the data is received from the joystick
float32[] axes          # the axes measurements from a joystick
int32[] buttons         # the buttons measurements from a joystick
```

Compact Message Definition

```
std_msgs/Header header
float32[] axes
int32[] buttons
```

autogenerated on Wed, 02 Mar 2022 00:06:56

(b) Figure shows a autogenerated overiew over a sensor message of type Joy, used for managing joystick commands.

Figure 3.11: Figures illustrates ROS workflow and a message example

3.5.5 WireGaurd VPN

WireGuard was created by Jason A. Donenfeld, also known as "zx2c4", with the first release on 09 December 2016 [55]. WireGuard is a communication protocol that is free and open-source, designed as a universal VPN for operation on embedded devices and supercomputers. Its software implements encrypted virtual private networks (VPN), with design goals that aim to possess low attack surface, high-speed performance and ease of use. WireGuard claims to have better performance and more power than the two standard tunnelling protocols; IPsec and OpenVPN. That information is also backed up by different reviews and tests. An early review from Ars Technica did, for example, found out that WireGuard connected and reconnected much faster than other protocols in the test and that its cryptographical choices, in other words, its security, pointed out to be better [56]. In May 2019, a machine-checked proof of the WireGuard protocol was published by researchers from INRIA; it was produced using the CryptoVerif proof assistant[57]. WireGuard is also a straightforward protocol and VPN to understand. It incorporates many modern VPN solutions and, in contrast to many other protocols, only delivers over UDP, which uses no handshake protocols, entirely skipping TCP support. Such a choice is explained as necessary according to WireGuard; due to the classically terrible network performance of tunnelling TCP-over-TCP [58].

WireGuard includes the following[59]:

- Curve25519 for key exchange
- ChaCha20 for symmetric encryption
- Poly1305 for message authentication codes
- SipHash for hashtable keys
- BLAKE2s for cryptographic hash function
- UDP-based only

Chapter 4

Methods

This chapter features the main methods used to answer and fulfil the goals in section 1.2. The methods will mainly be concentrated on building the software architecture for the remote control system, as stated in goal 2. Each section will discuss and explain the implementation and solutions of the different parts of the total system. At the end of each section, there will also be an architecture overview containing graphs and charts of that particular part. It is essential to understand that every graph or diagram has its level of abstraction, meaning that it can be wrong to compare them directly, but together they all form the complete remote control system. The architectural aspects will be covered from either some or all of the following points of view:

- **The physical view** describes the hardware allocation of each component of the system.
- **The process view** explains the run-time operations of the system and its processes, usually a sequence or process diagram.
- **The development view** shows the organization of software in the development of the system, usually a deployment diagram.

4.1 Connection and Communication Between Car and Operator

Teleoperation can be used for controlling autonomous vehicles when edge cases occur, as already discussed in subsection 2.1.4. However, for an operator to accurately manoeuvre a vehicle over IP, the remote controller must retain all necessary information and data over a secure, stable, high-performing communication channel, with latency lower than 50ms [22]. This chapter will address developing and establishing such a communication channel.

4.1.1 Peer to Peer(P2P)

Peer-to-peer (P2P) systems show numerous advantages over centralized systems, such as load balancing, scalability, and fault tolerance. [60]. In addition, no processing was needed at any possible server for this task and project. Then by removing the possibility of having a server as both a bottleneck in speed and a possible single point of failure, the direct communication between the operator and the autonomous agent would be the proposed solution.

The easiest way to connect to the car's computer from the outside WAN was by opening ports in the router placed in the car and sending packages directly between the operator and the car, letting them traverse the firewall and NAT. That was needed because all routers have a firewall that, by default, blocks or drops all incoming WAN traffic and only allows traffic outbound initiated by the internal LAN side [61]. In addition, port forwarding was needed to traverse Network address translation (NAT) inside the router, a method of remapping IP address space by modifying the network address in the IP header of packets during their transit across any routing device. [62].

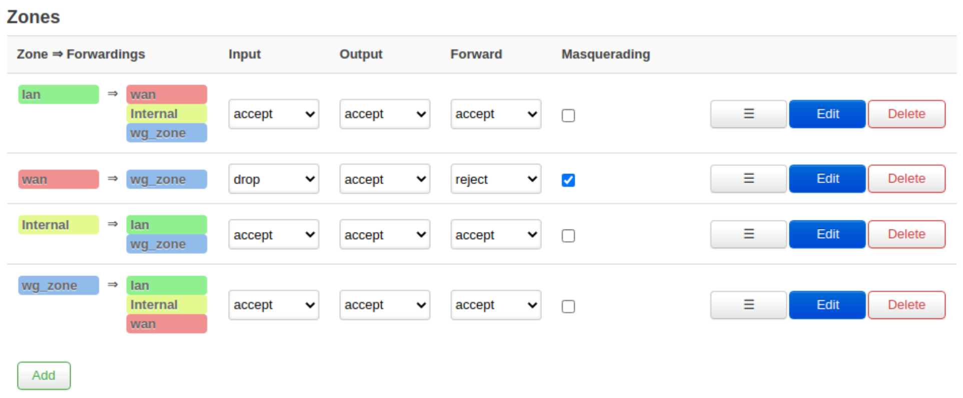
The mentioned is not enough to make a P2P connection between the operator and the car. In this case, the Internet Service Provider(ISP) Telenor does its network address translation on top of the one done locally in the router. The indicated induces a known issue of being behind double NAT, where the ISP has a carrier-grade or large-scale NAT (CGN). The CGN maps IPs and ports of the user devices in the network into other IPs and ports in its external interface. Even though the car router might be configured correctly for port forwarding, the ISP "master router" running the carrier-grade NAT will never let the packets travel to the destination. Therefore, the IP address and port seen from the operator's perspective would not be the actual IP and port given to the car's router. In addition, all IP addresses are usually dynamic and will change steadily.

To address the NAT issues, ISP (Telenor) was contacted, and a unique static IP was obtained on the SIM card placed inside the car's router, with their CGN in bridge mode on this particular IP. As a result, the CGN was disabled, and the translation of IP addresses by the ISP was stopped. Along with port forwarding from WAN to LAN in the car's router, the operator could reach the car at the same public IP and onto the same port at any time.

4.1.2 VPN

The solution with P2P is handy but unfortunately carries many security issues that need to be fixed. The first main issues are the open ports in the router interface, which can be attacked by first doing a port scanning and then exploiting the open port (the outcome will depend on the port type and what it can give access to). The second issue, and maybe the most dangerous, is the fact that the data transfer between the two hosts, the operator and the car, is not encrypted at all and can easily be exploited in attacks such as "man in the middle attack", eavesdropping, sniffing and spoofing, to mention a few. In the worst case, this can let the attacker gain control over the car[63]. Therefore this vulnerability was addressed with a solution chosen to be a VPN. The VPN would be a secure channel from the operator to the car, encrypting all data sent between the devices with end-to-end public-key cryptography. Hence, it alone can restrict eavesdropping, sniffing, and other attacks where the main target is to obtain transferred data.

The open ports problem was addressed by creating a special VPN zone inside the Arcus router rather than inside the car's computer. Consequently, incoming connections from WAN are not accepted into the port forwarding mechanism, except for the VPN; see Figure 4.1. Such a setup made the router act as a server, always listening to possible client VPN connections. If an operator was trying to connect to the car, a secure connection must be established with the VPN zone first; only possible for hosts with the right encrypting keys. After the connection is ready, the operator is inside the VPN zone the appropriate port, this case SSH and ROS are opened for the operator, and the car's computer could be reached; see Figure 4.2.

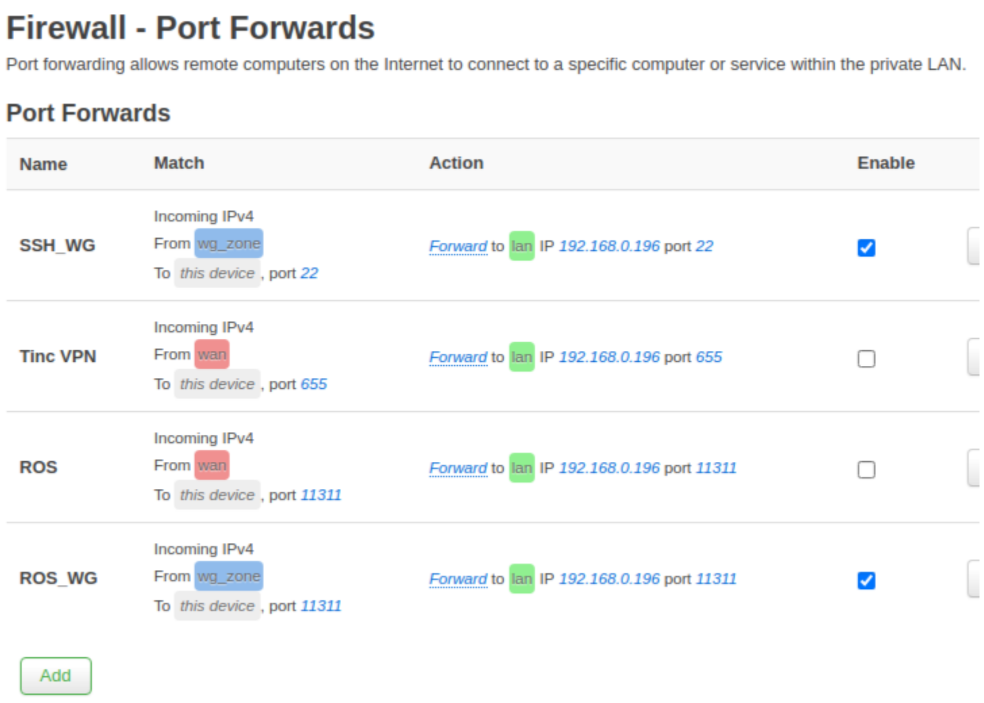


Zones

Zone ⇒ Forwardings	Input	Output	Forward	Masquerading
lan ⇒ wan, Internal, wg_zone	accept	accept	accept	☐
wan ⇒ wg_zone	drop	accept	reject	☑
Internal ⇒ lan, wg_zone	accept	accept	accept	☐
wg_zone ⇒ lan, Internal, wan	accept	accept	accept	☐

Figure 4.1: Illustrates the setup of WireGuard zone, the one in blue. WAN massages has only access to the WireGuard zone, the router and LAN more secure to threats.

The choice for a suitable VPN protocol was set between using OpenVPN with its out-of-the-box solution with graphical user interface, the lightweight TincVPN daemon and the new WireGuard VPN protocol. When implementation started, OpenVPN became a non-working solution very fast. Because DriveWorks has a limited version of Linux by design, OpenVPN is not installable. Even though it would have worked, OpenVPN seems to lose to TincVPN and especially to the WireGuard VPN in speed, simplicity and controllability[64][65]. The choice was then between Tinc and WireGuard.



Firewall - Port Forwards

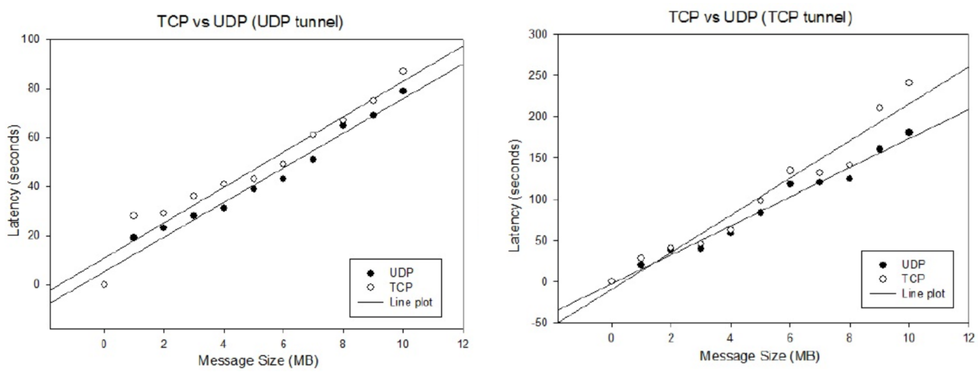
Port forwarding allows remote computers on the Internet to connect to a specific computer or service within the private LAN.

Port Forwards

Name	Match	Action	Enable
SSH_WG	Incoming IPv4 From wg_zone To *this device* , port *22*	*Forward* to lan IP *192.168.0.196* port *22*	✓
Tinc VPN	Incoming IPv4 From wan To *this device* , port *655*	*Forward* to lan IP *192.168.0.196* port *655*	☐
ROS	Incoming IPv4 From wan To *this device* , port *11311*	*Forward* to lan IP *192.168.0.196* port *11311*	☐
ROS_WG	Incoming IPv4 From wg_zone To *this device* , port *11311*	*Forward* to lan IP *192.168.0.196* port *11311*	✓

Figure 4.2: Illustrates the setup of WireGuard zone port forwarding, the one in blue. When the any host has connection established with the VPN zone(wg_zone) only ROS and SSH messages to the specific ports are possible to use. Giving an additional layer of security.

After doing more research and setting up both, it was clear WireGuard had some extra advantages over Tinc. WireGuard could deliver even more speed than Tinc because of its UDP-only design, whereas Tinc could go into using TCP. TCP is more secure and desirable in many situations but not in real-time applications where speed is more important than a lost packet. Tinc's TCP design could become a problem due to the remote control system using ROS for controlling the car (**??**). ROS is a robot operating system explained in both chapter 2 that mainly uses TCP to send commands. Having TCP both in the outer and inner tunnel could induce an issue known as TCP Meltdown [66]. TCP-over-TCP could make one of the layers overcompensate when detecting a problem because they have the same logic. In addition, as TCP is way slower than UDP, doubling up with it would make the connection have more latency, which is not desirable for this system. The chosen WireGuard was, therefore, a better solution given the TCP-over-UDP design of the remote control system. Tinc VPN was also not possible to use as a VPN zone inside the Arcus router, making the problem of open ports applicable again. The chapter 5 shows the difference in the performance of Tinc and WireGuard[67]. See Figure 4.3 for a comparison between a TCP-over-UDP and TCP-over-UDP and Figure 4.5 for a simple deployment view over the system setup.



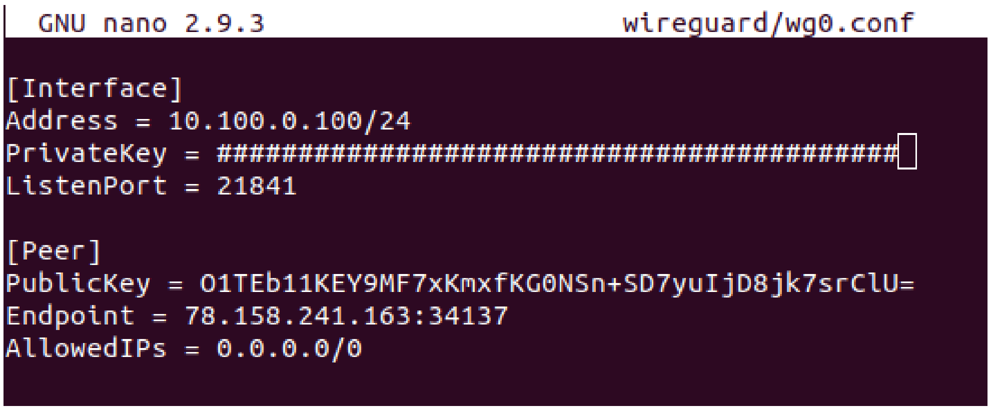
(a) UDP and TCP inside a UDP tunnel. **(b)** UDP and TCP inside a TCP tunnel.

Figure 4.3: A figure showing a latency comparison between TCP and UDP tunnels in WAN. Source [67]

4.1.3 Automated Connection

Since WireGuard is only a protocol with no GUI, everything concerning the connection must be done manually in CLI. That, together with ROS-dependent software, discussed in **??**, makes it difficult and not user-friendly to start a session with the proposed remote control system. The issue confrontation was done with a couple of ".bash" scripts, running different commands automatically in a given order. It was done on both the car's and operator's computer. One script named "autoStartAll.sh", ran by the operator, fires a variety of other scripts, establishing the VPN connection and starting the system in the car and the operator's device.

In order to start scripts from one host on the other, a specific script establishes an SSH connection through the VPN, which makes it possible to dictate orders to the car's computer and, for example, run the "setupAndStart.sh" script that will start all necessary ROS based programs and make the car ready to drive. The sequence diagram in Figure 4.8 illustrates how the first operation of establishing a VPN and SSH connection is realized, and **??** shows a screenshot of the particular WireGuard VPN interface settings used for the connection. Altogether, those scripts make the system much more user-friendly since only one script needs to be run with a password prompt from SSH, and the rest is taken care of automatically.



```
  GNU nano 2.9.3                          wireguard/wg0.conf

[Interface]
Address = 10.100.0.100/24
PrivateKey = ##########################################
ListenPort = 21841

[Peer]
PublicKey = O1TEb11KEY9MF7xKmxfKG0NSn+SD7yuIjD8jk7srClU=
Endpoint = 78.158.241.163:34137
AllowedIPs = 0.0.0.0/0
```

Figure 4.4: Figure is screenshot of the the WireGuard interface setup "wg0" on the remote machine. This is used in establishing VPN connection to the WG interface inside the car´s router. (The private key on the picture is hidden)

4.1.4 Architecture Overview

Deployment Diagram

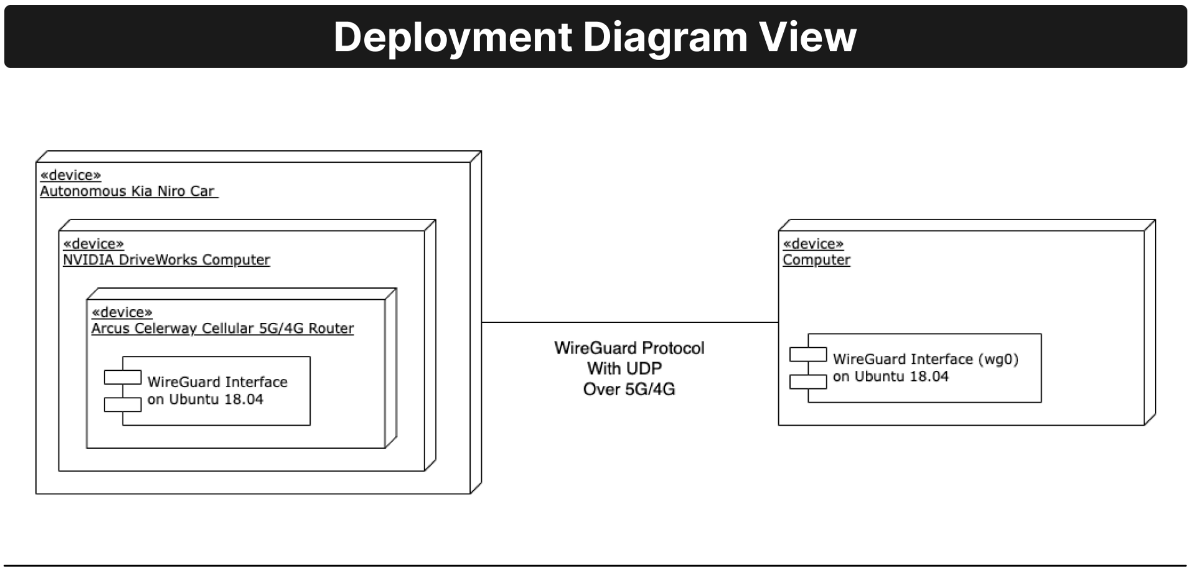

Deployment Diagram View
«device»
Autonomous Kia Niro Car
«device»
NVIDIA DriveWorks Computer
«device»
Arcus Celerway Cellular 5G/4G Router
WireGuard Interface on Ubuntu 18.04
WireGuard Protocol With UDP Over 5G/4G
«device»
Computer
WireGuard Interface (wg0) on Ubuntu 18.04

Figure 4.5: Figure showing a deployment diagram view with only relevant components of the remote control system network architecture.

Network Diagram

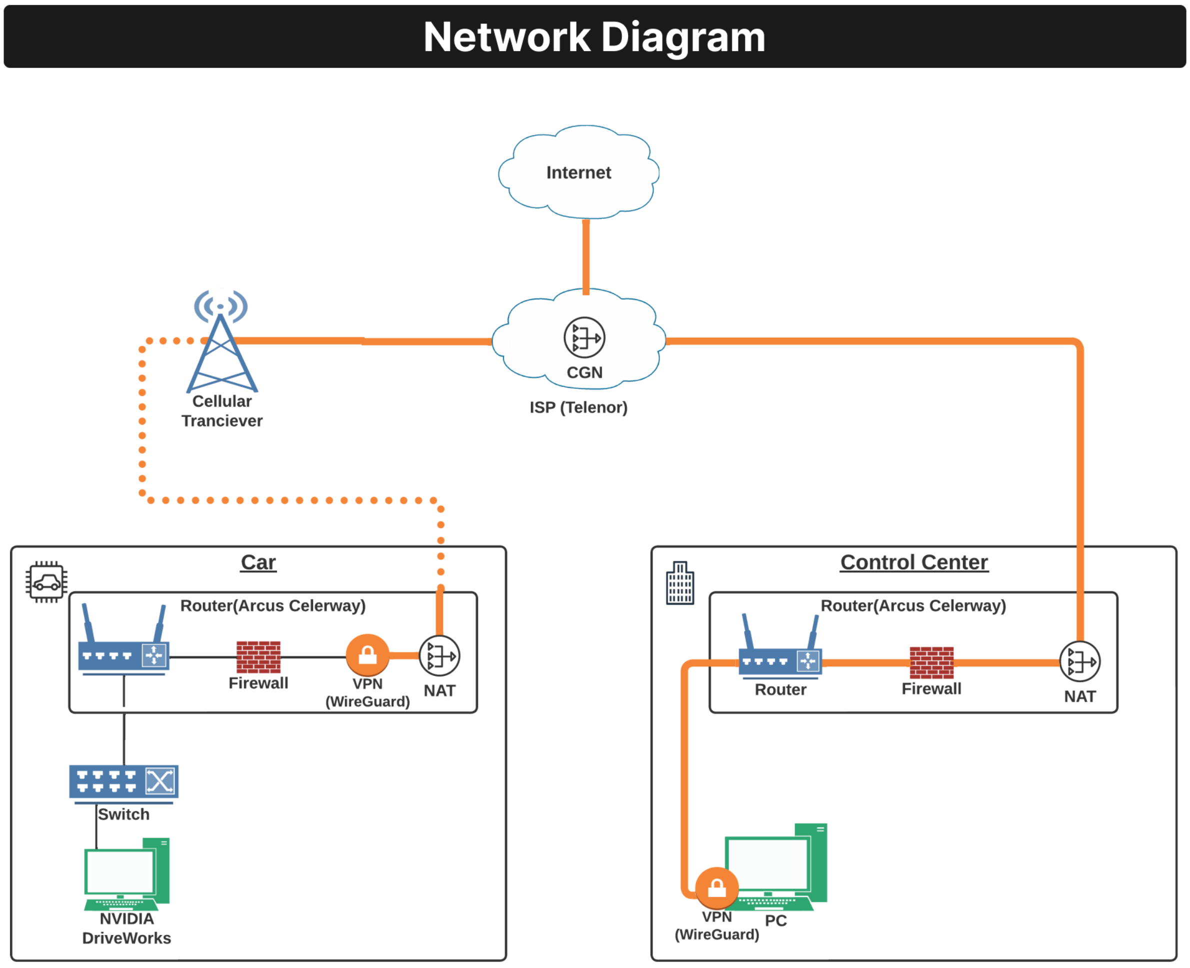


Figure 4.6: Figure showing a network diagram with only relevant components of the remote control system network architecture.

Logical Network Diagram

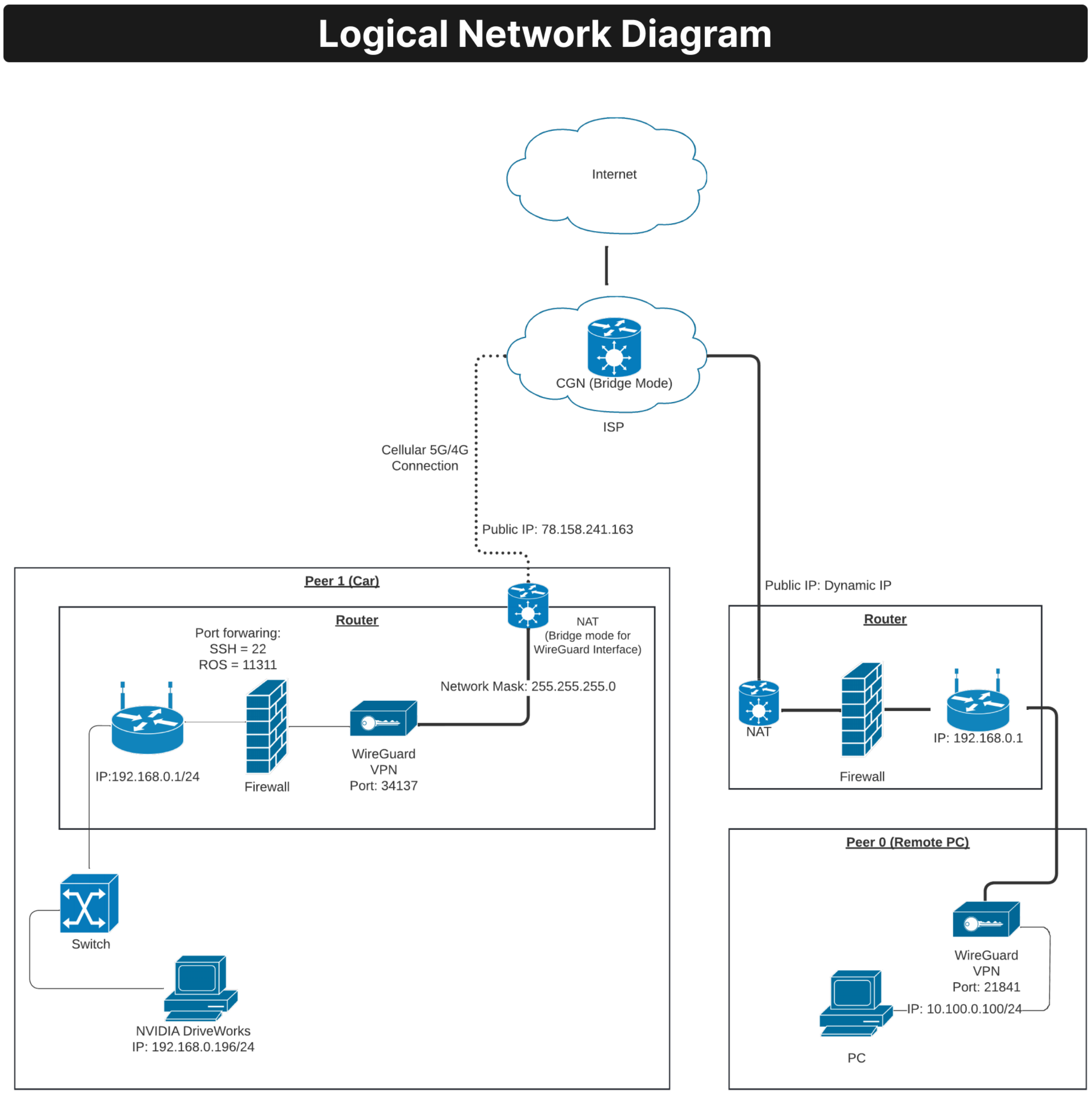


Figure 4.7: Figure showing a logical view network diagram with only relevant components of the remote control system P2P network architecture. Note that the shown CGN is in bridge mode only for this particular static IP and that the IP address of remote PC is a logical IP set by WireGuard, the physical is arbitrary for this view.

Sequence Diagram of VPN and SSH Connection

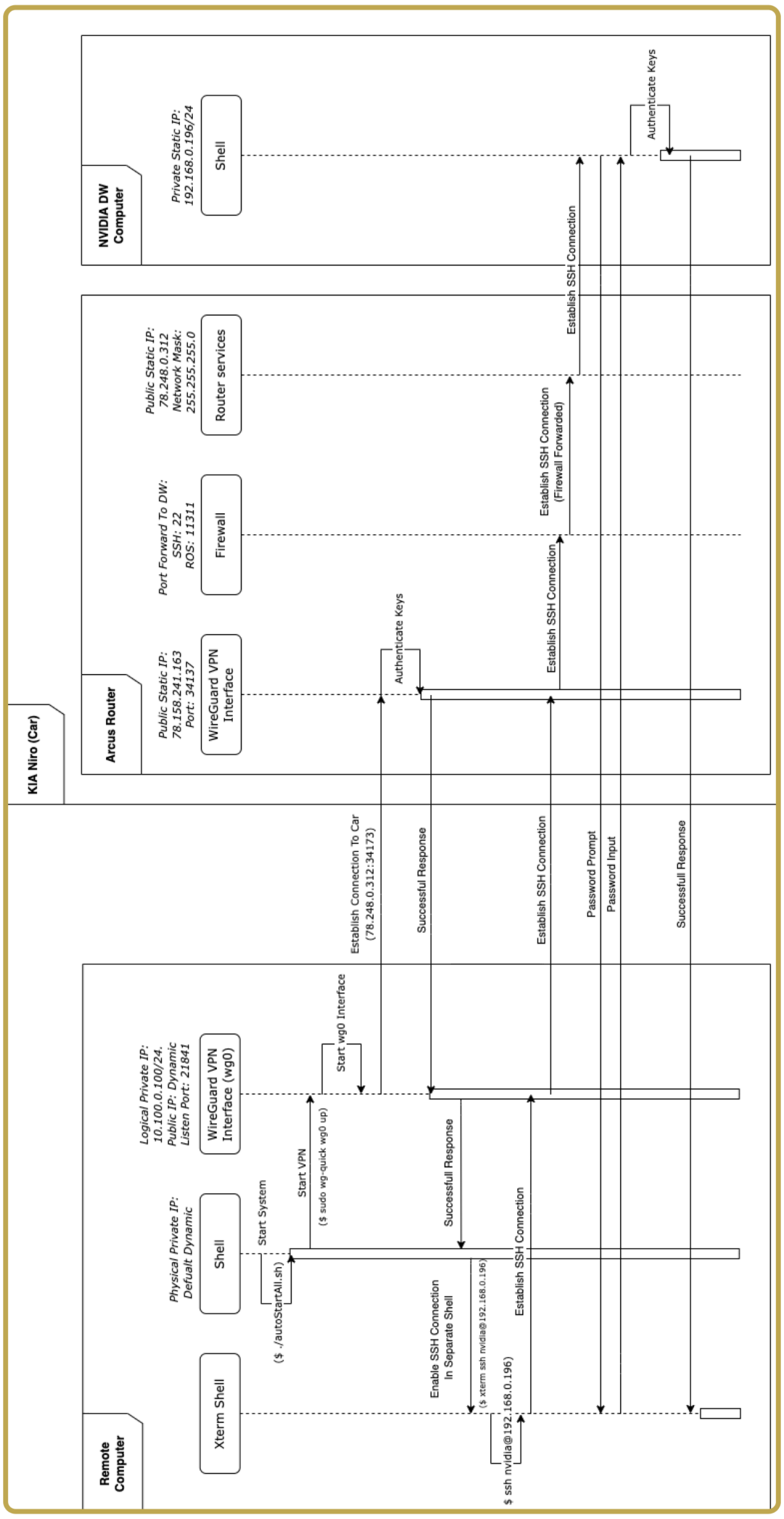

KIA Niro (Car)
NVIDIA DW Computer
Private Static IP: 192.168.0.196/24
Shell
Authenticate Keys
Arcus Router
Public Static IP: 78.158.241.163 Port: 34137
WireGuard VPN Interface
Port Forward To DW: SSH: 22 ROS: 11311
Firewall
Public Static IP: 78.248.0.312 Network Mask: 255.255.255.0
Router services
Authenticate Keys
Establish SSH Connection
Establish SSH Connection (Firewall Forwarded)
Establish SSH Connection
Establish Connection To Car (78.248.0.312:34173)
Successful Response
Establish SSH Connection
Password Prompt
Password Input
Successfull Response
Remote Computer
Xterm Shell
Physical Private IP: Defualt Dynamic
Shell
Logical Private IP: 10.100.0.100/24. Public IP: Dynamic Listen Port: 21841
WireGuard VPN Interface (wg0)
Start System
($ ./autoStartAll.sh)
Start VPN
($ sudo wg-quick wg0 up)
Start wg0 Interface
Successfull Response
Enable SSH Connection In Separate Shell
($ xterm ssh nvidia@192.168.0.196)
Establish SSH Connection
$ ssh nvidia@192.168.0.196)

Figure 4.8: Figure showing a sequence diagram view with only relevant components for the VPN and SSH connection establishment. The diagram shows also commands during the process. After the connection is established it is possible to use the connection directly end-to-end without going through the steps again

4.2 Vehicle Control System

An autonomous car's software or hardware stack is built of various parts, where many are critical for the system to function. However, some are still more dominant, and a part of the core, like the control system, which is especially crucial for the remote control system of this thesis [2]. In this section, its realization will be presented. It is worthy of note that this section covers the software primarily since this project did not include working with the hardware.

4.2.1 Drive Kit and CAN bus wiring

As mentioned in Figure 3.4.3, the car already had a pre-installed OSCC Drive Kit by Polysync before this project started. This kit is connected to the car's CAN bus to receive essential data and possibly take over the control of the car. Kia Niro's car has a handful of CAN buses on board. The OBD-II CAN network - the vehicle's self-diagnostic and report-capable bus, contains vehicle state information such as steering wheel angle, wheel speeds, and brake pressure. This information is helpful for algorithms like PID control or displaying data in the graphical user interface.

Rather than sharing the vehicle's OBD-II bus and possibly interfering with the vehicle's native messages, Polysync OSCC Drive Kit has made its CAN bus called Control-CAN. Through this bus, all commands and reports are sent and received. This is illustrated in the Figure 4.9 by Polysync [68].

The CAN Gateway module bridges the vehicle's native built-in OBD-II bus and the Control-CAN. The gateway forwards all relevant vehicle messages from the OBD-II bus to the Control-CAN bus, which can later be consumed by applications subscribing to the OBD messages, such as the GUI or PID controller. By already mentioned diagram in Figure 4.9 shows that CAN messages are only published in one direction: towards the Control-CAN bus from the OBD-II CAN bus, and all overriding control commands are sent directly to the respective component, such as the steering wheel motor from the Control-CAN.

4.2.2 Utilize CAN Data in ROS

In order to utilize the data from the OBD CAN bus, it needs to be read and converted to ROS. ROS acts as a backbone framework for control and communication in this project between the remote computer and the car systems. ROS is used as the top layer for communication and control, whereas the Drive Kit and CAN bus are the lower layers directly communicating with the car's internal systems, such as sending torque value to the power steering column. Therefore all data from and to the Drive Kit must be converted into the respective type. However, this paper will not cover how the converting happens since it is the work of Polysync and not developed as a part of this project.

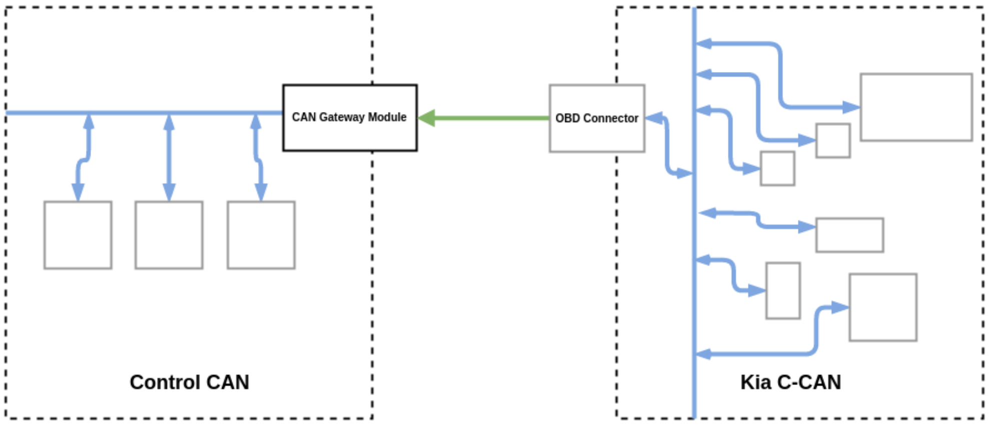
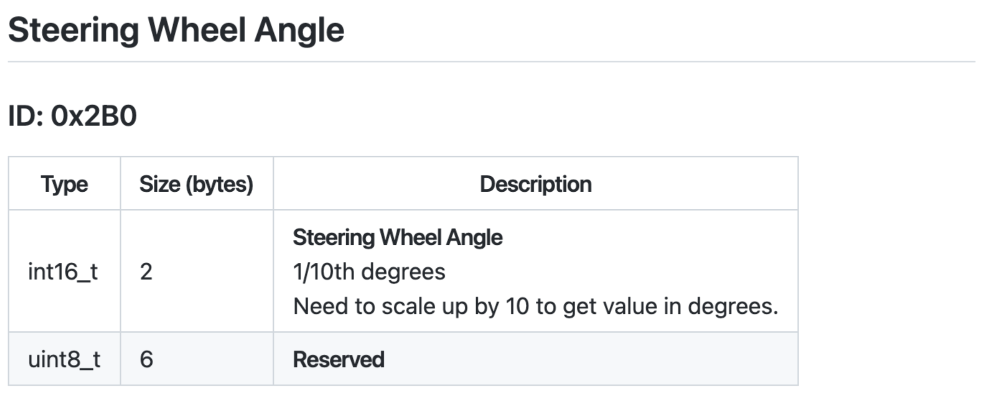


Figure 4.9: Figure illustrates the wiring of Drive Kit into the car, though CAN bus. Source [68]

The data is gathered from the OBD CAN bus and then republished as a ROS topic on a specific node. By running the command "candump can0," it is possible to see various hexadecimal numbers. Those are all different ids in the CAN bus, each connected to a specific data stream running through the bus. The corresponding id needs to be known to use any specific data stream. This Figure 4.10 shows a table with information from Polysync on what id wheel angle corresponds to in the CAN bus. This way, data for steering angle, throttle position, brake position, and speed is known and can be taken out from the CAN bus and then published on a ROS topic for the GUI **??** to subscribe on. On Figure 4.11 it is a plot showing the data taken out from the CAN bus with speed on the top followed by throttle, braking and steering wheel angle as the last with a middle point as 0 degrees. A similar way is used to send the data into the Drive Kit and the car's components. Nevertheless, that is mainly done automatically by the Polysync Drive Kit and is not covered in this paper. The important part is that code for receiving ROS msg, converting and sending it is available for the project inside the "teleop.cpp" file. Moreover, the full Github for the Drive Kit is available online [68].

4.2.3 Wired Control Over ROS With Xbox Joystick

During the beginning of this project, the car already had some example functionality embedded, from both the Polysync Drive Kit, Nvidia DriveWorks and some made by NAP-lab. The wired joystick control is the most relevant example code and is used as a base for the remote control system. The example contained one script file called "runJoystickDrive.sh" located inside the Joystick catkin workspace in the car's computer. This script first sources the ROS environment required by ROS because ROS relies on the notion of combining spaces using the shell environment. By doing that, all variables connect to the ROS environment in that specific shell. In addition, it allows access to all packages of ROS even though they are not in the same folder.



Steering Wheel Angle

ID: 0x2B0

Type	Size (bytes)	Description
int16_t	2	**Steering Wheel Angle** 1/10th degrees Need to scale up by 10 to get value in degrees.
uint8_t	6	**Reserved**

Figure 4.10: Figure illustrates the wiring of Drive Kit into the car, though CAN bus. Source [68]

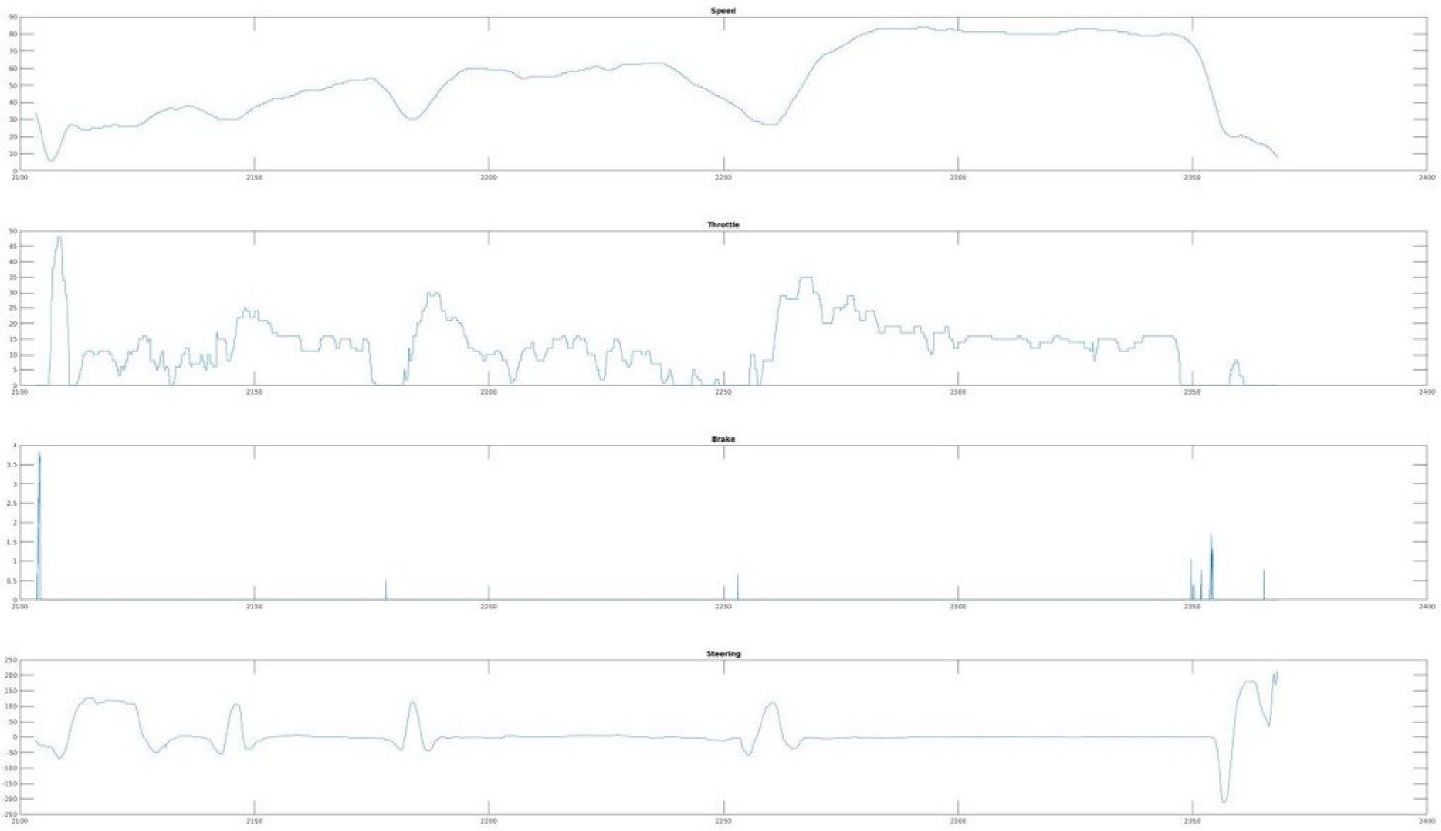


Figure 4.11: Figure illustrates plot of respectively from the top; speed, throttle, brake and steering from the OBD CAN bus, during a short manual drive with the car According to vehicle internal metrics.

After the sourcing is done, a launch file "example.launch" runs, and launches three nodes: "roscco_node.cpp", roscco_node.cpp and "joy_node". The Joy node has the mission of sending joy_msgs messages, earlier shown in Figure 3.11a, when an Xbox joystick is connected to the car's computer and initialized. Then the Teleop node does different computations, mapping and calibration for the commands to make sense for the car and the Drive Kit before publishing four different topics. That is the steering, throttle, brake and enable_disable topic. Those topics are then subscribed on by rossco_node, converted into data used by the OSCC Drive Kit, and sent as control commands through the Control-CAN bus and into the car's components.

This process is essential and described here because it was used as the base for developing the controlling part of the remote control system with the G93 steering wheel.

4.2.4 Vehicle Control Over IP From LAN

As explained in subsection 4.2.3, the car did have some functionality for controlling the car with a joystick. However, it did not have the feature necessary for taking control over the car from another host, on either LAN or WAN. It was therefore chosen to manipulate, enhance and facilitate the "joystick.launch" file and its dependencies into working over the Internet, rather than only local. Therefore, before using the developed VPN and P2P secure connection from WAN (**??**), the objective was to make a setup that could work on the car's LAN.

The Control from LAN was realized by taking apart the "example.launch" and dividing it into two parts: Teleop and Roscco node formed the receiving part while Joy node became the transmitting part. The launch files became "example_nojoy.launch" and "example_onlyjoy.launch" with their respective bash scripts "runJoystickDriveNoJoy.sh" and "runJoystickDriveOnlyJoy.sh". The name "example" was not changed because it is used as the base; therefore, it remained for this project to keep the code consistent. The scripts and launch files were placed at each end of the system; the car and the remote computer. This way, the remote computer could publish joy_msgs messages on a topic called joy. The receiving part could then subscribe to the topic and extract the data for further processing, as explained in **??**, and control the car. Figure 4.12 shows an RQT graph of the nodes, topics and their interaction when the "runJoystickDrive.launch", that means before it was divided into two parts, but the interactions are still the same. In another diagram, however, the entire sequence of the process is illustrated; see Figure 4.17

In the theory of ROS, this setup only works if the nodes know exactly where the ROS master is located, more precisely, having its IP address. In addition, the IP of ROS-master should not be "localhost", which is standard. For those criteria to be met, ROS_MASTER_IP and URI must be specified on both the car's and operator's computer. It was chosen to have ROS-master running in the car since it makes more sense than on the remote PC. If the connection with the operator is broken,

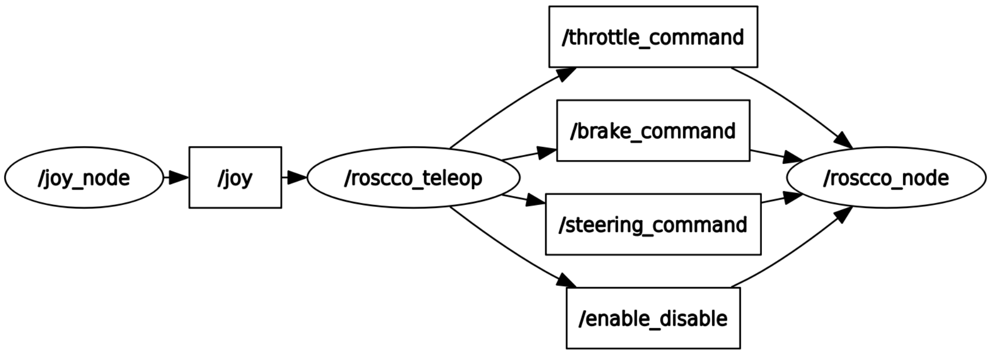


Figure 4.12: Figure shows a RQT graph of the nodes, topics and their interaction when the "runJoystickDrive.launch" is run

the nodes and system running in the car will not collapse. Therefore, the master URI and IP in the car are set to the car's local IP. In the remote PC, the URI is also set to the car's local IP; this way, the ROS nodes running at the remote control PC can locate the ROS-master. The system was then fully functional, and the car could be controlled from an Xbox controller over LAN connected remote control PC.

4.2.5 Vehicle Control Over IP From WAN

Change of ROS-master URI and IP

The next step after achieving fully functional remote Control from LAN was to make the system work from WAN, i.e. having the remote computer on another network. A VPN and P2P connection were needed to grasp the task. The system connection scripts and steps described in **??** was applied, and the P2P connection was set. Since the remote computer is connecting to the WireGuard interface in the router, seeing it as a peer and not the car's computer directly means that all devices connected to the router will be available to the remote computer peer through their local IP address, as long as the respective ports in the router are open. From the car's perspective, the remote PC is in the same network but at another IP address; the one set in the WireGuard settings on the remote machine, even though it is located on a completely different network. Having that information is vital because of the ROS-master IP and URI settings. Those are the only parts that need to be slightly changed in order to make the system work over IP. The ROS-master URI will be the same as already explained, but the ROS IP will be changed on the remote PC into the one set by WireGuard.

Latency Caused Drive Kit Collapse

When this setup was tested, it did work as intended for ROS functionality. However, the Drive Kit was not made to take care of a situation when latency occurs, forcing the Drive Kit to disable itself every time the connection was initialized. The Drive Kit was originally only intended to be used by wire; therefore, its threshold for latency between one command to another is, as stated by Polysync maximum of 50ms. Consequently, it needs to be fed by new commands at a frequency of 20Hz. By testing, this showed to be wrong, and the latency threshold looked to be around 20-30ms. Because all communication over IP does have at least some latency, often called ping, this infers into a problem where such low latency is very hard to achieve. Even with the 5G cellular network, this can be difficult since there is always a chance of packet loss or lousy network connection leading to uncertainties and sudden jumps in ping, called jitter.

Command Repeater

To deal with the possible jitter problem and latency intolerance, the usual approach is to use a jitter buffer. VoIP often uses that to smooth any variations in the packet stream. However, measuring the average ping through VPN made it clear that a buffer would not help since it was about 40-60ms on 4G and 15-30ms on 5G, sometimes worse. In addition, the worst that could happen is that a buffer could be exhausted before a significant packet loss, and the car could stop mid-driving. The choice was then to make a simple repeater that would repeat the last command received from the "Joy" topic at 50Hz, meaning a maximum latency of 20ms independent of the packet stream. That would make the Drive Kit always receive a command in time even though there is high jitter in the network. The Drive Kit threshold was, in this case, not altered because the Drive Kit system is an extensive system with not only software but also hardware parts, which means that a change of the threshold could interfere with or cause problems to some hardwired parts in the kit.

4.2.6 System Adaption for G29 Steering Wheel and Pedals

Steering the car with an Xbox controller(3.7a) was not optimal because any slight joystick movement could propagate into a swift movement of the car's steering wheel. It was therefore decided to introduce and embed the G29 steering wheel (3.7b) with accompanying pedals into the remote control system. The G29 did not need more drivers than the Xbox controller on the remote Linux machine, and the only thing installed as extra was "jstest-gtk", a tool for testing and watching the status of the steering wheel position [69].

Mapping the G29 Kit

As the steering wheel kit was recognized and worked in the "jstest-gtk" tool, the next step was to incorporate it into the ROS code. By changing the mappings of all buttons, pedals and the steering column in "teleop.cpp", the node started to receive ROS Joy messages. The earlier mapping was done with respect to the Xbox controller, which had fewer buttons and a different setup.
In order to determine to what ID every button corresponded to on the steering wheel, a Joy node was started and by using the "rostopic echo joy" command in the CLI, it was possible to check every button and watch its response and boundaries manually. Then, the mapping method was set inside the "teleop.cpp". Therefore, every command from the steering wheel was interpreted as intended by the Polysync OSCC Drive Kit and sent further to the car's components.

4.2.7 Steering Control Strategy

Introducing the G29 kit into the system did work but only to the point of not crashing it or having wrong mapping of buttons or pedals. In addition, the driving and steering itself were not as intended.

Manual Steering

The Xbox controller, as used in subsection 4.2.3 did incorporate a model in which a movement of the small joystick directly corresponded to a specific amount of torque sent into the Drive Kit and to the power steering column. That did work as long as the joystick could self-centre itself, which it usually does on an Xbox controller. This means that any move to the left or right would make the power steering column apply a force on the steering as long as the operator was pushing on the joystick. With a steering wheel, on the other hand, namely the G29, it will always remain at the place operator left it on, leading to a constant torque applied to the power steering column. The torque would also become larger for every steering wheel angle when moved to any side.

The used model can be interpreted to be directly proportional to the operator's input without having any correcting methods for changes in the environment. The operator is also unaware of how much the steering wheel will move and if less or more torque should be applied to get the desired trajectory. This system was, therefore, not suitable for the remote control system.

Control Algorithm Comparison

There are, of course, a variety of different controller algorithms. Some are better suited than others in each situation but generally when it comes to traditional control strategies, PI or PID controllers are the primary choice for speed control and other types of longitudinal control. In contrast, MPC or a geometric controller are usually employed for lateral control, such as steering.

That, however, depends upon the specifics of the problem statement. In this case, an operator will always have control of the vehicle, only remotely. Hence any complex algorithms would be overkill in terms of technical complexity, computational overhead and tuning [70]. At the same time, the algorithms with low complexity could be not enough for smooth and stable steering and driving experience. In Figure 4.13 two diagrams compares the most popular control strategies available for an AV based on research done by Cornell University, New York [70].

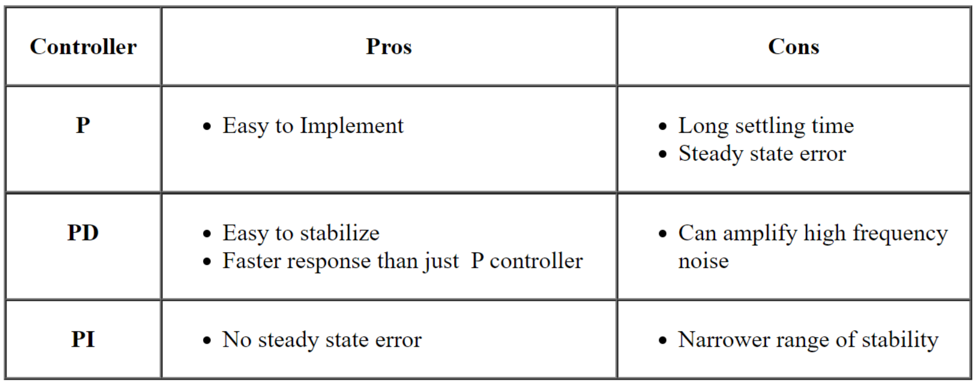
Any controller algorithm can be either an open or closed-loop controller. That refers to electronically controlled devices and whether they can take feedback from the device. Devices with closed-loop controls get feedback from their sensors and adjust the response and operations based on current device conditions. On the other hand, open-loop controls do not receive feedback and purely do what they are programmed to do without regard for current device condition [70]. An open-loop controller in this project would not be enough for the remote control system since it would do basically what the manual steering torque applier was doing without knowing how much force to apply. Therefore a closed-loop controller is needed, and based on the comparison in Figure 4.13 the best would be either PID or a Geometric controller, but since the PID controller is more widely used, it is the one chosen for this project. [71]

PID implementation and tuning

Inside the PID controller, there are three variables; proportional (P), integral (I) and derivative (D). In tandem, they work to give rise to a much more efficient PID controller than only using a plain P, PD or PI. The PID controller takes advantage of all the three primary controllers in order to generate a more sophisticated control action that proportionally corrects the variable error. It also dampens the resulting overshoots and reduces a possible steady-state error over time. Nevertheless, all these three variables need to be tuned after the implementation, either automatically assisted or manually. [70].

Comparison Metric	**Bang-Bang Control**	**PID Control**	**Geometric Control**	**Model Predictive Control**	**Imitation Learning Based Control**	**Reinforcement Learning Based Control**
Principle of Operation	Error Driven (FSM)	Error Driven (Correcitve)	Model Based (Corrective)	Model Based (Optimal)	Supervised Learning	Reinforcement Learning
Tracking	Poor	Good	Very Good	Excellent	Good	-
Robustness	Poor	Very Good	Very Good	Excellent	Good	Good
Stability	Poor	Very Good	Very Good	Excellent	Good	Good
Reliability	Very Low	Very High	Very High	Extremely High	Low	Low
Technical Complexity	Very Low	Low	Low	Very High	Low	High
Computational Overhead	Very Low	Low	Low	Very High	High	High
Constraint Satisfaction	Poor	Good	Good	Excellent	-	-
Technical Maturity	Very High	Very High	High	Moderate	Low	Very Low

Figure 4.13: Summary of Control Strategies for Autonomous Vehicles. [72]

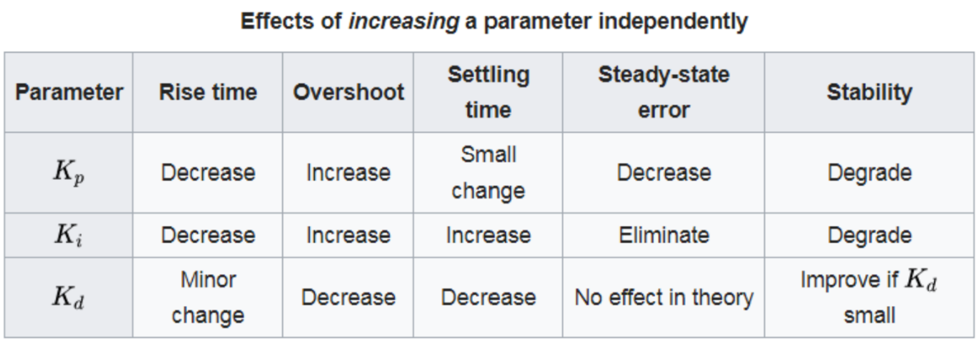


Controller	Pros	Cons
P	• Easy to Implement	• Long settling time • Steady state error
PD	• Easy to stabilize • Faster response than just P controller	• Can amplify high frequency noise
PI	• No steady state error	• Narrower range of stability

Figure 4.14: P, PD and PI controller comparison. [72]

The PID steering controller algorithm was implemented inside the "teleop.cpp" and in "pid_controller.cpp". The controller takes input from the ROS commands from the G29 kit and gets input from the OBD CAN bus. This information is used to alter the torque sent to the power steering column to match the wheel angle given by the CAN bus and the one sent via ROS from G23.

The tuning of this PID controller was done manually because any automatic tuning in simulated environments requires a realistic model of the car, which was not made and could take too much time. So instead, an extra feature was implemented into "teleop.cpp", which was real-time adjustments of the three parameters Kp, Kd and Ki. This could be done right from the G29 steering wheel by pressing down a square, triangle or circle button, mapped to Kp, Kd and Ki, and then pressing plus/minus on the wheel. For example, this would add/subtract 0.0005 to the initial value. By doing this, it was possible to test the rotation of the car's steering wheel and, at the same time, adjust the parameters accordingly to how the response was. In the end, the controller was tuned and smoothly working. Figure 4.15 shows how the different parameters' tuning impacts the performance.



Effects of *increasing* a parameter independently

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

Figure 4.15: Figure illustrates showing how tuning of the different parameters in a PID controller impact its performance. [73]

4.2.8 Architecture Overview

RQT Graph

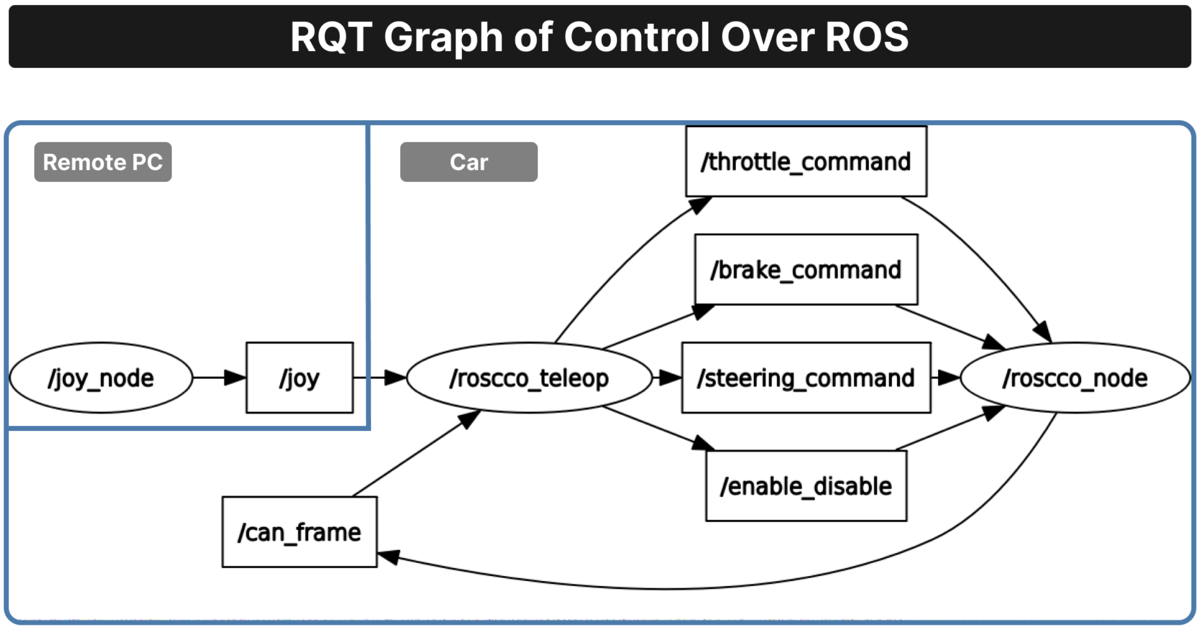


Figure 4.16: Figure showing a RQT graph also called ROS graph, which illustrates nodes that subscribes or publishes topics related to the control part in the full remote control system. The topic "can_frame" is used for publishing information regarding car status data such as speed.

Sequence Diagram

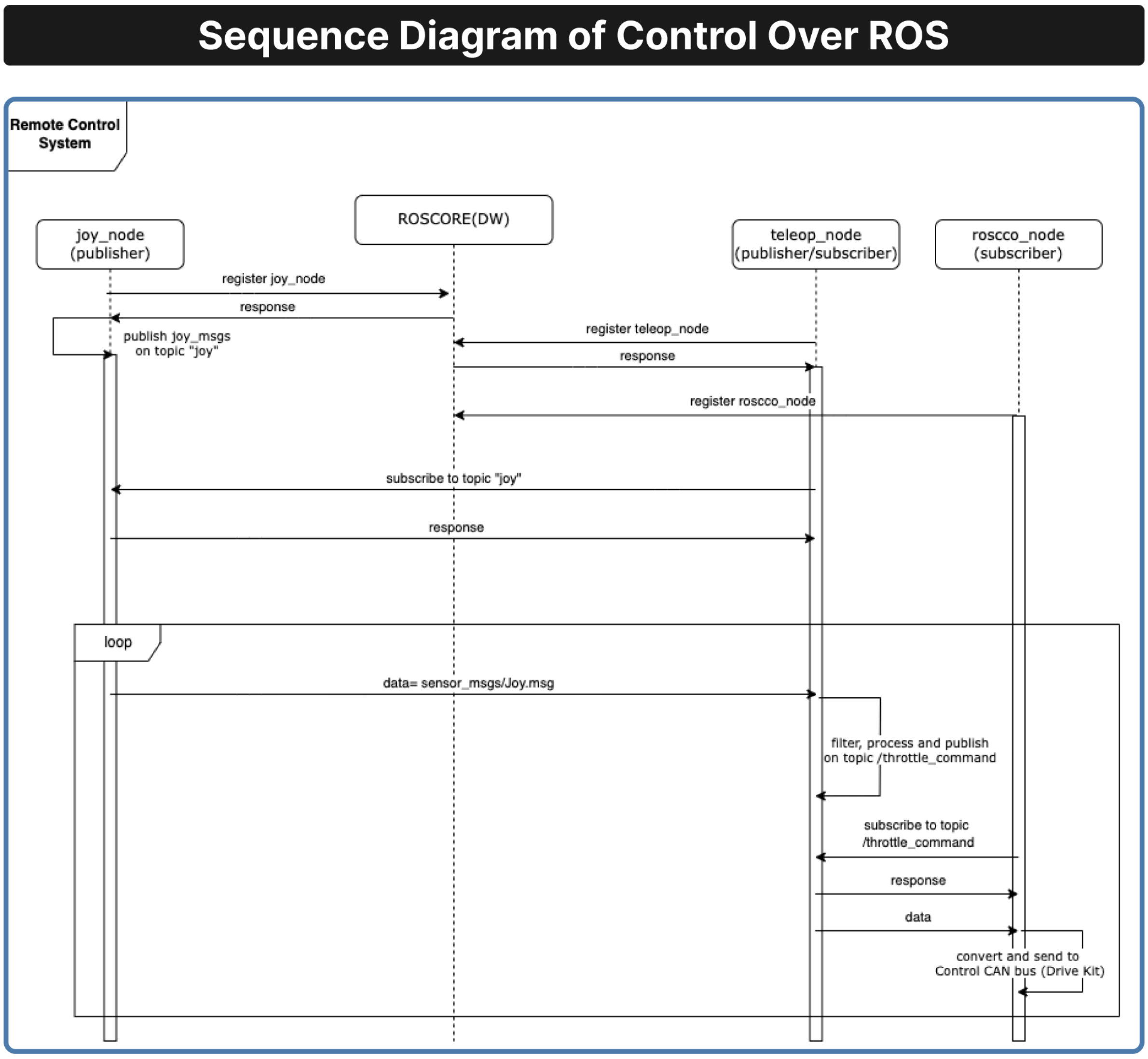

Sequence Diagram of Control Over ROS
Remote Control System
joy_node (publisher)
ROSCORE(DW)
teleop_node (publisher/subscriber)
roscco_node (subscriber)
register joy_node
response
publish joy_msgs on topic "joy"
register teleop_node
response
register roscco_node
subscribe to topic "joy"
response
loop
data= sensor_msgs/Joy.msg
filter, process and publish on topic /throttle_command
subscribe to topic /throttle_command
response
data
convert and send to Control CAN bus (Drive Kit)

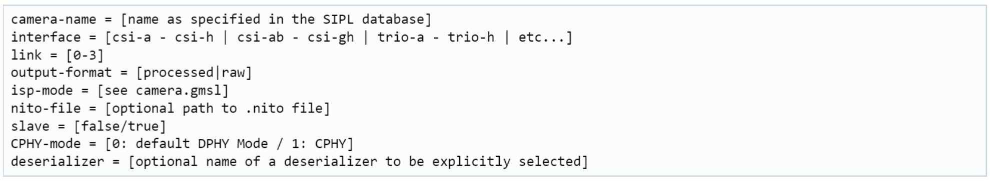
Figure 4.17: Figure showing a sequence diagram with only relevant components regarding ROS nodes used in the remote control process of the vehicle. The diagram shows an abstraction of only ROS processes and assumes that the VPN and SSH connection showed in Figure 4.8 has already been established.

4.3 Real Time Video Transmission Over IP

New technologies have allowed even professional broadcasters requiring high bandwidth and low latency to migrate to IP networks for video transport, even in real-time. With the 5G cellular network emerging, P2P, also called device-to-device (D2D) communications, has been an unfolding technique anticipated to provide many mobile users with new advanced services in 5G networks. As mentioned in section 3.2, 5G is perfect for the task of this project, especially for sending a large amount of data with low latency, high speed and over a stable connection. This section describes how video transmission from the car to the remote computer was realized over a cellular network and in real-time. It is important to understand that architectural choices were explicitly made concerning the available equipment and software in the car. [74]

4.3.1 Obtaining Image Data From the Cameras

When this project started, the car already had all cameras installed, connected and initialized. All cameras (3.4.3) were using the GMSL protocol and connected to the Nvidia Drive Platform through a FAKRA port. On Figure 4.20 an overview of all parameters a camera contains in the Nvidia Drive OS (**??**). NAP-lab had three main front cameras and one rear camera connected into interface csi-a and link 0-3. The other four side cameras were connected to the interface csi-c with a similar link 0-3. All cameras were available through built-in DriveWorks commands such as the "sample_camera" used for testing [75]. The successful sample test ensured that everything was correct and that cameras were ready to be used in further development.



```
camera-name = [name as specified in the SIPL database]
interface = [csi-a - csi-h | csi-ab - csi-gh | trio-a - trio-h | etc...]
link = [0-3]
output-format = [processed|raw]
isp-mode = [see camera.gmsl]
nito-file = [optional path to .nito file]
slave = [false/true]
CPHY-mode = [0: default DPHY Mode / 1: CPHY]
deserializer = [optional name of a deserializer to be explicitly selected]
```

Figure 4.18: Figure shows the parameters for camera.gmsl in Nvidia Drive SDK. Source of image [75]

4.3.2 Compression Algorithm

Video as a media has the nature of being needy in both storage and bandwidth. The setup of cameras designed in the mock-up, later described in section 4.5, stipulates the use of four cameras. That means four full HD cameras at 30 frames per second (3.4.3) being sent over IP in real-time. In order to grasp the bandwidth necessary for the video transmission, a calculation on the raw video was conducted.

Five relevant factors in every digital video signal determine how much bitrate, and thus bandwidth is required: video resolution, frame rate, colour space, chroma sub-sampling and colour bit-depth. Together, they form the base on how much data will be required to send and process. Equation 1 in (4.19) shows a simple mathematical formula used to calculate the needed bandwidth [76]. By filling out all variables based on information from subsection 3.4.3, and assuming that the cameras are not doing sub-sampling, meaning that they reserve full-colour depth, we get the following equation 2 in 4.19. The results of it in equation 4 (4.19) show an insane bandwidth of 2.52 Gbps per camera, in total 10.08 Gbps with four cameras. This illustrates that it would have been impossible to send any image in the "raw" format over the IP when using any WiFi or Cellular network. The maximum speed of a high-speed wireless network is usually around 1 Gbps, the same as a high-end 5G cellular network these days.

$$\begin{aligned} Bandwidth &= (Video\ Resolution) * (Frame\ Rate) * (Color\ Space) * \\ &\quad (Chroma\ SubSampling) * (Color\ BitDepth) \end{aligned} \tag{1}$$

$$Bandwidth = 1080p * 30\ Hz * RGB * 12bit \tag{2}$$

$$2515345920\ bps = 1928 * 1208 * 30 * 3 * 12 \tag{3}$$

$$2515345920\ bps = 2.52\ Gbps \tag{4}$$

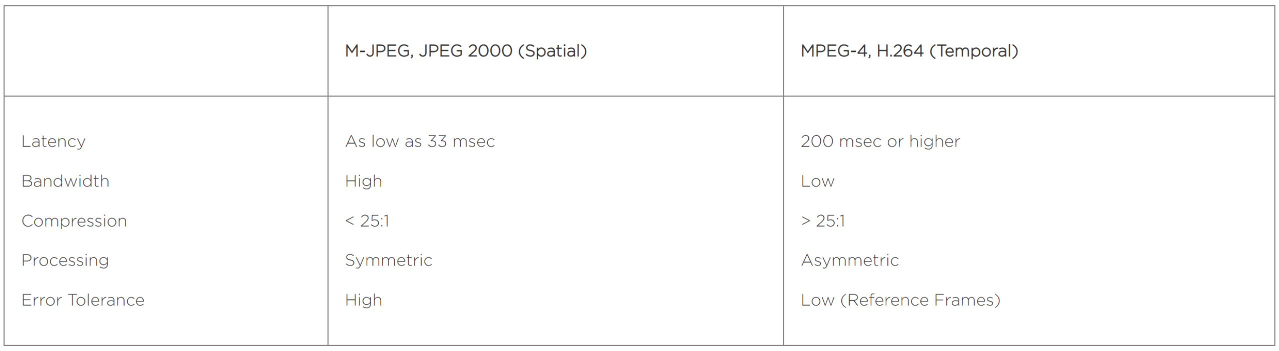
Figure 4.19: Equations for calculating bandwidth usage during video transmission. [76]

The calculation example above shows that encoding is highly needed for this remote control system to include even one camera. Hence, a search was done in order to find the best-suited codec for this task. There are many different encoding algorithms, all delivering different performances. Nevertheless, the most prevalent standards are MJPEG, MPEG4, and H.264. In addition, the choice is highly limited when depending on ROS-supported compression algorithms in tandem with the GMSL cameras and Driveworks. The only directly supported type of image compression algorithm is MJPEG, often referred to as "type=compressed" in ROS. Testing the MJPEG compression and displaying the frames through ROS locally showed a considerable use of CPU power in the car, letting the computer freeze if more than three cameras were encoding simultaneously. However, that did not prove the MJPEG as bad; it showed the problems it entails when in this specific setup of software and hardware. Therefore the H.264 looked more promising since the algorithm should consume less CPU.

4.3.3 Video Encoding and Transmission From The Car

H.264 compression was added to ROS through two ROS libraries "h264_image-_transport" and "nvidia_gmsl_driver_ros" 3.5.4 made by an Estonian University, and shared freely on Github [52]. Those packages made it possible to encode, decode and transport frames compressed with H.264, which is normally not supported in ROS. In this package, it was also possible to fine-tune the compression bitrate, which again was vital for this project. After some testing, the bitrate for the compression was chosen to be as low as 2 Mbps for each camera. The reason was mainly the high CPU usage and the fact of using an additional 4G in places without 5G, which could sometimes have low bandwidth. The picture was good enough for seeing important details, and a higher bitrate was not as critical for this task.

The H.264 encoding is done by launching the nvidia_gmsl_driver_ros.launch file inside its catkin workspace, which starts several nodes with different tasks. All cameras specified in a ".yaml" parameter file are then checked, and frame stream is enabled. All data is then compressed into h.264 and published on a ROS topic corresponding to every camera, for example, "interface/link0/camera/image/h264" showing the camera on interface "A" at link 0, which in this case is the front camera 1. The ROS transport method for publishing the topic is "image_transport", which sends "sensor_msgs/Image". More exact information about the work of this library is available on their GitHub [52].



	M-JPEG, JPEG 2000 (Spatial)	MPEG-4, H.264 (Temporal)
Latency	As low as 33 msec	200 msec or higher
Bandwidth	High	Low
Compression	< 25:1	> 25:1
Processing	Symmetric	Asymmetric
Error Tolerance	High	Low (Reference Frames)

Figure 4.20: Figure shows a comparison between codecs. Source of image [76]

4.3.4 Video Decoding at Remote

When the graphical user interface wants to show the video streamed from the car, it needs to use its "gui_node" to subscribe to the topic on which the car is publishing the camera frames and reading them. The problem is that the video frames published on the topic are compressed in h.264. This data needs to first be decompressed, then read and displayed.

An extra library from UT-ADL for ROS was installed to achieve decompression on the remote computer. This library called "h264_image_transport" adds to the existing ROS "image_transport" ability to decompress H.264. Since four streams are being published on four different topics, they must all be decompressed. By developing a ".launch" file containing four republisher nodes that use the "image_transport" with H.264 decompression and create four brand new topics containing the decompressed image stream called "interface/link0/camera/image/decoded" for each of the cameras corresponding to the link number. That makes it possible for the GUI to subscribe to the four topics and receive the desired camera frames.

4.3.5 Architecture Overview

RQT Graph

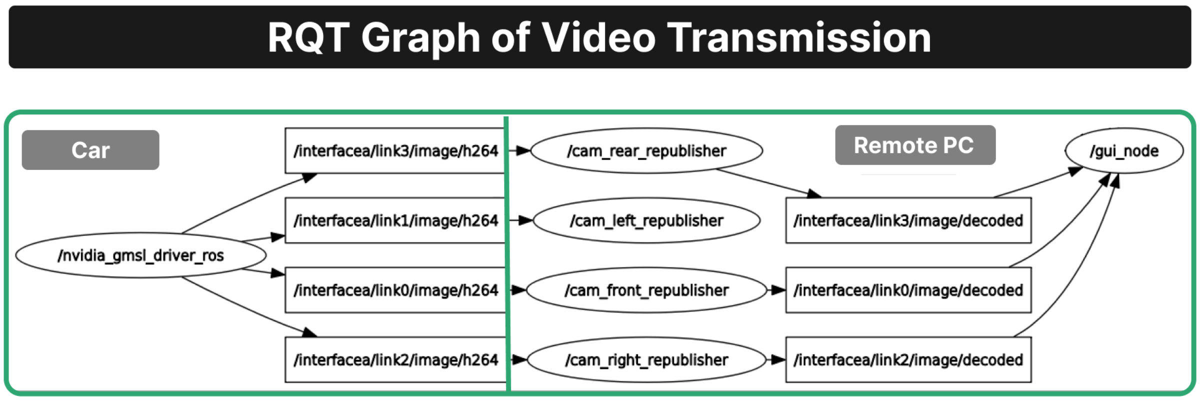

RQT Graph of Video Transmission
Car
Remote PC
/nvidia_gmsl_driver_ros
/interfacea/link3/image/h264
/interfacea/link1/image/h264
/interfacea/link0/image/h264
/interfacea/link2/image/h264
/cam_rear_republisher
/cam_left_republisher
/cam_front_republisher
/cam_right_republisher
/interfacea/link3/image/decoded
/interfacea/link0/image/decoded
/interfacea/link2/image/decoded
/gui_node

Figure 4.21: Figure showing a RQT graph also called ROS graph, which illustrates nodes that subscribes or publishes topics related to video transmission in the remote control system. Link 0-4 corresponds to camera 0-4, and this particular setup shows only 3 cameras are subscribed and used by the GUI

Sequence Diagram

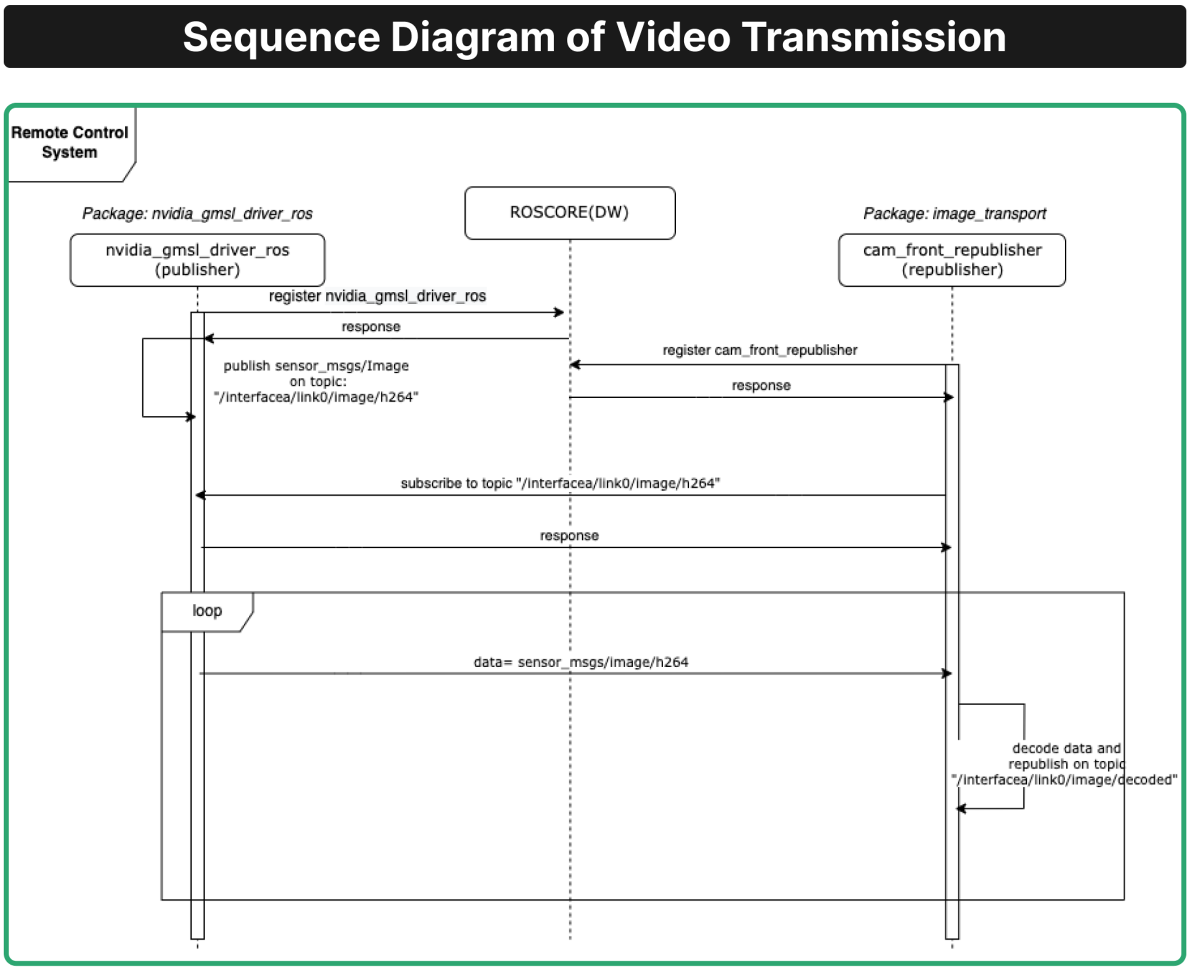

Sequence Diagram of Video Transmission
Remote Control System
Package: nvidia_gmsl_driver_ros
nvidia_gmsl_driver_ros (publisher)
ROSCORE(DW)
Package: image_transport
cam_front_republisher (republisher)
register nvidia_gmsl_driver_ros
response
publish sensor_msgs/Image on topic: "/interfacea/link0/image/h264"
register cam_front_republisher
response
subscribe to topic "/interfacea/link0/image/h264"
response
loop
data= sensor_msgs/image/h264
decode data and republish on topic "/interfacea/link0/image/decoded"

Figure 4.22: Figure shows a sequence diagram with only relevant components regarding ROS nodes used in the video transmission processes. The diagram shows an abstraction of only ROS processes and considers that the VPN and SSH connection showed in Figure 4.8 has already been established. The diagram also illustrates only one republisher and one camera being used. In the system, there are four of them (front, right, left, rear), but they do the exact same thing.

4.4 Methods and Algorithms for Real-Time Object Detection

This section features concepts for object detection and their implementation. This methods are planned in future development to be embedded as a part of the operator-assistant in the complete remote control system. The goal of this system is to be a safety guard for the operator when situations with packet loss occurs, or the operator simply does a wrong move, result in fatal consequences.

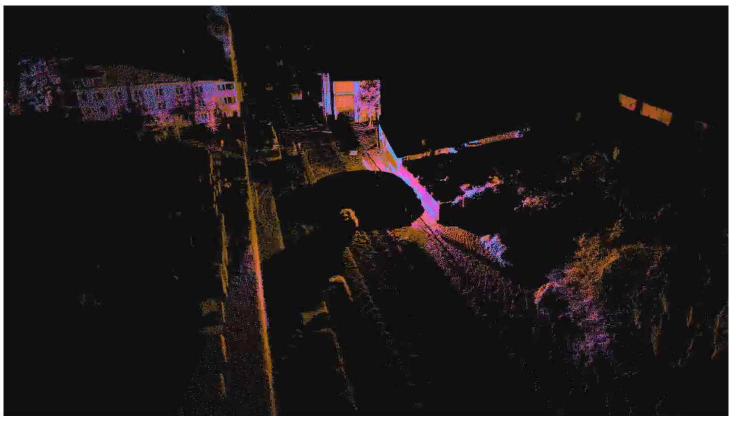
In order to map as much as possible in the field of object detection and later answer RC 2.5 in section 1.2, the objective for this section was divided into smaller parts as follows; see the list **??**. The experiment will give information about the procedure of recording, segmentation and making use of lidar data for real-time object detection and also explore different methods. It was chosen to use the lidar instead of the cameras in order to explore more of the subject and also map the possibilities and future potential of the operator-assistance system. [77]

4.4.1 Collection and Segmentation of Lidar Data

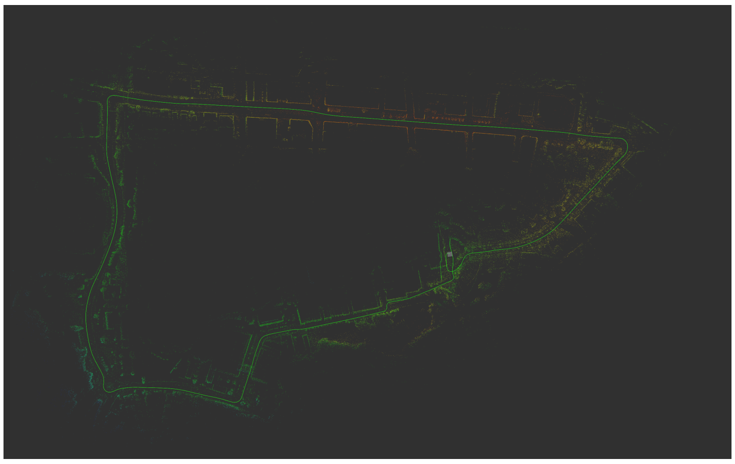
The lidar data collection was done by driving a car around in Trondheim while recording the top lidar (OS2-128) see Figure 4.23a for raw 3D Lidar data and Figure 4.25b for illustration of the driven route with SLAM, done by NAPLab. Later because of several complications with formats in Drive Works and Ouster Studio, segmentation was done with the help of the NAPLab research group; that made a small plugin for Nvidia DriveWorks to get the right outcome. This is important in order to have object detection software such as Yolo or SSD to work, since they are not able to detect object in 3D environment [77]. In addition, the process would be the same as performing it for a camera, which means that the cameras used for video streams and sent to the remote computer could also be used for object detection. The result was 19 videos with 100 frames each, in three variants of visualization: intensity, range and ambient, whereof all videos are in black and white. Figure 4.24 shows all three visualizations of the 2D space lidar image from one of the 19 videos, namely the one chosen to later be the test video.

4.4.2 Label Objects on Videos

In order to label all the respective videos in time, it was done together with the help of the NAPLab group. The labelling was done in CVAT, free, easy use and well-known annotation tool for images and videos [78]. It was chosen to use a car, truck, bus, motorcycle, bicycle, scooter, person and rider as the classes for the labelling because those are the primary moving objects to be detected when driving a car, see Figure 4.25a. This labelled data was then ready for being used in either training or testing with an object detection algorithm, Figure 4.25 shows the annotated data.

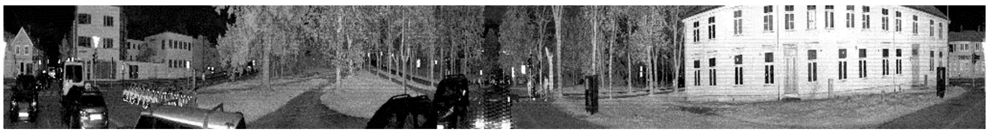


(a) Lidar data in 3D format

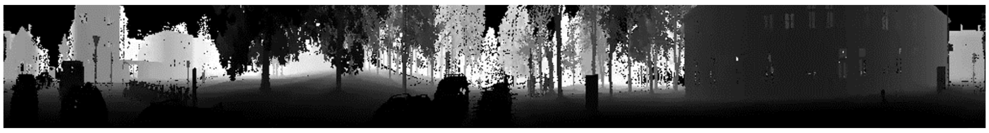


(b) SLAM on the driven route

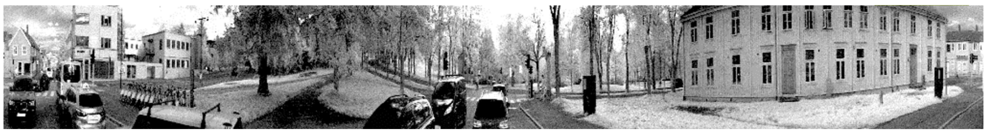
Figure 4.23: A figure showing the 3D lidar data and the driven route illustrated through SLAM.



(a) 2D lidar picture in intensity variant

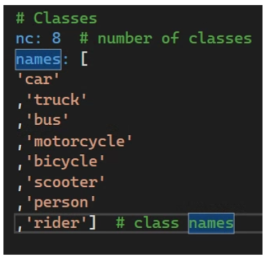


(b) 2D lidar picture in range variant



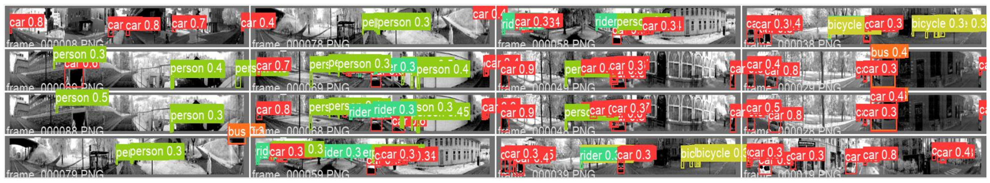
(c) 2D lidar picture in ambient variant

Figure 4.24: A figure showing the lidar data segmented into 2D space images, in three variants of visualization.



```
# Classes
nc: 8  # number of classes
names: [
'car'
,'truck'
,'bus'
,'motorcycle'
,'bicycle'
,'scooter'
,'person'
,'rider']  # class names
```

(a) Chosen annotation classes



(b) Labels on a frame

Figure 4.25: A figure showing annotation of video with the chosen classes and labeling.

4.4.3 Object Detection Algorithm

It was chosen to use Yolov3, Yolov3 Tiny and Yolov5 for these experiments. The reason was that those are popular, simple, fast and easy to work with [79].

The implementation of the object detection algorithm was done with plain Yolov3 and the Coco dataset, downloaded from the original website together with the Darknet framework. To load the frames, a small algorithm was written. It used functions in OpenCV to store the CNN and scaled the frame to the appropriate 416x416. The scaling was done not by cropping but by crushing down the image. This way, all the details should follow as in the original. The bounding boxes were also drawn by coding a box generator that took anchor points given from the prediction of Yolov3 as input. After the first run, this showed to be a failure because none of the objects was predicted, and none of the boxes was drawn. The problem lied in the scaling of the image or rather the crushing of the frame. Since the images are of the wide format, 1024x128, when crushing them, almost everything will look like a brick without any normal details. The solution could be to either train Yolov3 on those scaled images or try something like patching of splitting [80]. Patching was then implemented, and the images were split into four parts, fed into the prediction algorithm and concatenated back into the original picture, but now with the attached bounding boxes. In the results chapter section 5.7, Figure 5.9 shows the to different results, with and without the patching and also rest of the experiment.

4.5 Graphical User Interface

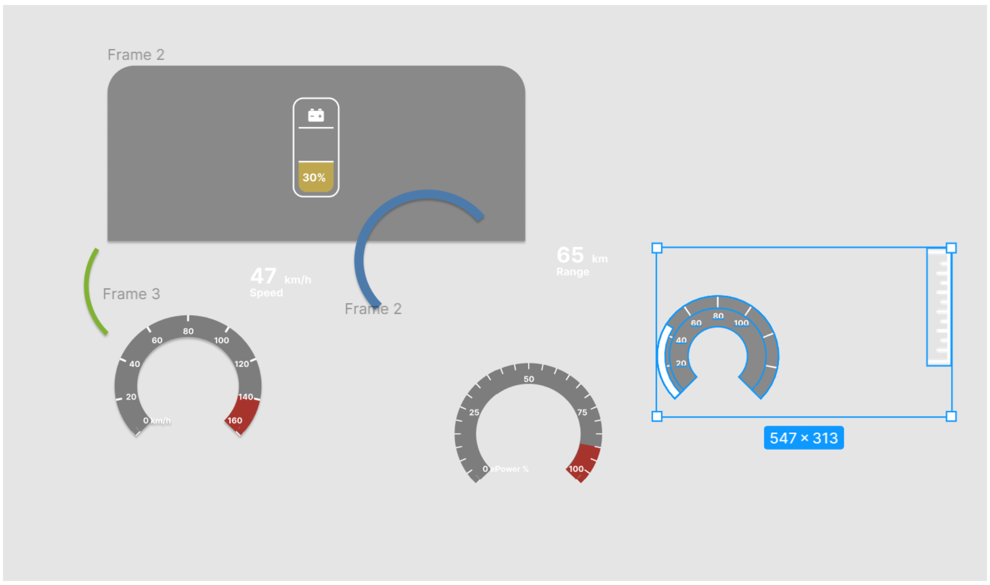
As the project's objective was to create a product prototype that would act as a proof of concept, a graphical user interface plays a vital role for the end-user. Therefore design and testing became an essential part of the development process. To ensure that the design would support and highlight the potential of the remote control system, it was chosen to focus on making the GUI highly functional, aesthetically pleasing and user-friendly. To achieve these criteria, a design-thinking process was followed rather than a user-centred design process because of the main concentration on functionality, quality and innovation. Several mock-ups were made, iteratively tested, and enhanced based on the results, followed by implementation and real data integration into a desktop application. [4] [81]

4.5.1 First Design Phase: Overall Conceptual Design

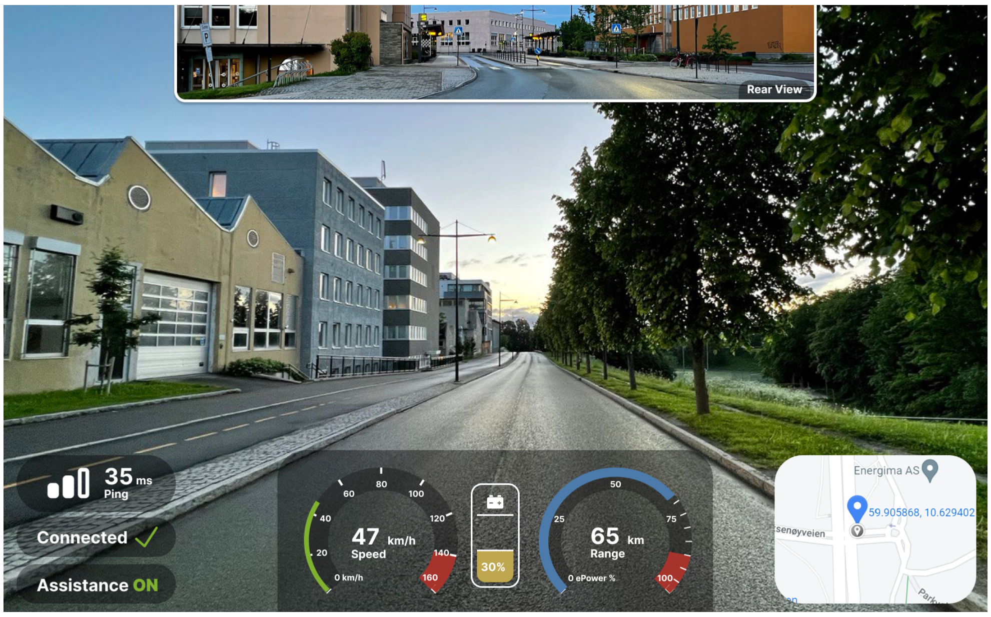
The first mock-up was designed using Figma - a vector graphics editor and prototyping tool. A prototype was designed to show the application's general idea and central aspects, everything from how a camera could be shown to how a gauge cluster could be designed. This mock-up did not contain actual data; all buttons and text were placeholders. The idea was to understand the interface's look and get a starting point before implementing a functional version. The colour palette and general design solutions like border-radius, and corners, to mention a few, were set and standardized at the start for consistent use in the whole interface.

The first overall conceptual mock-up is presented in Figure 4.28. The design choices were not random but carefully considered by research and inspiration from major game manufacturers making simulator racing games, such as Gran Turismo for Playstation. Inspiration was also taken from the car industry, for example, on how to make a proper gauge cluster. Games were considered an inspiration because they have been developing over a long period, a layout that can give any player a good overview of the road, the surroundings and data from the digitally made car. This correlates with the same objective set for this GUI; control a car remotely.

The mock-up in Figure 4.28 consists of a front camera image taking up the entire screen, because of its importance, a rear camera on top, mimicking a real rear mirror, a gauge cluster showing speed, range, "ePower" (Throttle) and battery level in the middle, followed by a map with the car's location and essential information on each side of the cluster. Altogether it was enough to see what a possible solution could look like.



(a) A figure showing elements designed in Figma, used to later construct the gauge cluster for the mock-up.



(b) A figure showing first conceptual design of the GUI.

Figure 4.26: Figure showing the design of the first Mock-Up of the Remote Control System as a graphical user interface.

4.5.2 Second Design Phase: Functional Design

The second design phase was aimed toward the functional parts of the design. It was important to understand what information was needed on the screen for a driver to feel comfortable driving the car. Usually, such a functional mock-up can be made in Figma and tested by users, but in this case, all data, images and the driving experience are impossible to simulate in Figma. The solution was to implement one part at a time and test its functionality by me; this again shows the not user-centred approach but rather a design-thinking process. (The map was not implemented because it is just an extra feature.)

The first implementation and tests led to issues such as:

- Not enough information about the final or total delay.
- Not enough view around the car. Extremely hard to navigate.
- Difficult to see how much throttle is applied.
- Not possible to view the position of the real steering wheel inside the car.
- When are brakes applied, and how much?

A new mock-up was made to address all the issues presented, shown in Figure 4.27. Steering wheel position, brake pressure, and total delay were added to the interface. Two more rear camera views were integrated into the gauge cluster, now presented as an instrument panel. The inside view was added on the right side in order to watch what is happening inside the car, but it could also be shifted out for an extra-wide front camera. This design was maybe a bit more crowded, but all this information was considered essential to make safe driving possible for the operator, based on qualitative testing.

4.5.3 Visual Design Principles

The basic five visual design principles found in 5principlesUX and their use during the design phase are explained in the following paragraphs.

Scale: *This design principle refers to using relative size to signal importance and rank in a composition of different components, and it is generally recommended to only have about three different sizes for elements inside a component*. It was chosen to use three to four, as the interface would have a lot of data inputs and information that needed to be held visually separated. This visual hierarchy can be seen in the speedometer and the torque gauge Figure 4.28a. The speed and the torque value in each of the gauges are the most prominent elements, followed by the info, the measuring unit and the values around the gauge. This gives any operator of the system the most valuable information as standing out as the most significant elements, followed by more minor, less essential elements.

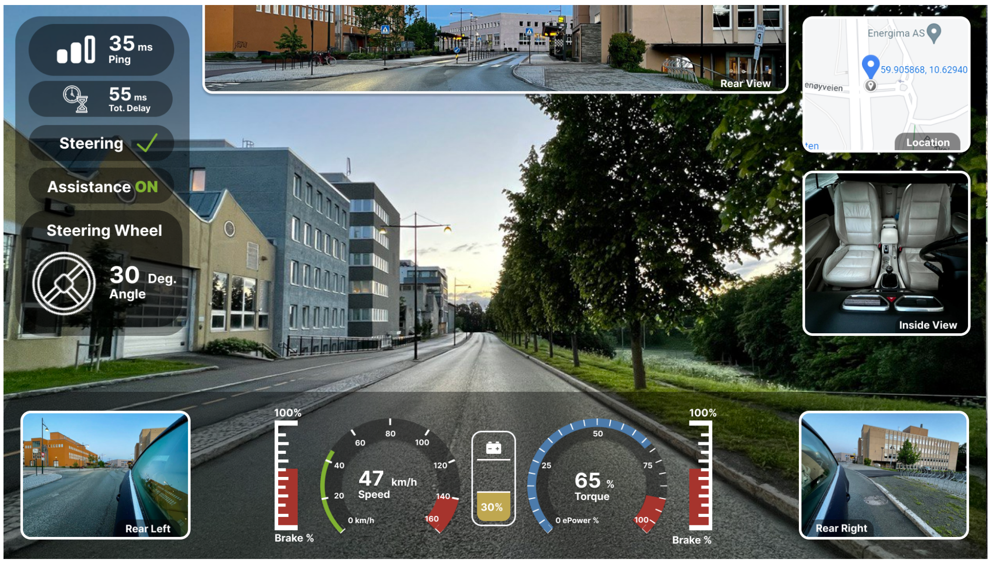


Figure 4.27: Figure illustrates second version of mock-up, this time with more functional design and fixes of the issues in the first mock-up.

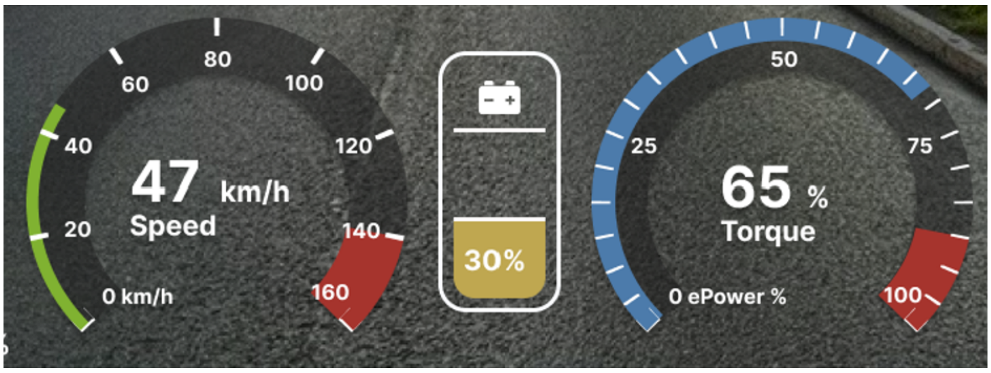
Visual Hierarchy: *A principle addressing how to intentionally design web pages in order to deliver the hierarchy of element importance to users.* It should take a minimum of time for a user to locate essential elements on a page. To ensure that this graphical interface delivered a good visual hierarchy, the page was divided into parts where all of them were placed on a partly transparent background, see Figure 4.27. This creates a separation between one section and another making it easier for users to spot different functionality. The same is done inside the information panelFigure 4.28c, where every group of elements with its own information are placed in their own rectangle. Furthermore, the information panel is located in the top left corner because this is where website users tend to look first [82], making the most critical information read at first.

Balance: *Satisfying arrangement or proportion of design elements.* Since the intended usage area of this system is primarily remote control of a car in traffic; it had to be informative and balanced, without any unnecessary elements. One exception was the difference between the left information panel and the right camera and location panel. They were designed to be strongly symmetric, but the data inside is highly asymmetric and gives an unbalanced overview. That is inevitable because of the different information they need to deliver to the operator. At the same time, this makes the interface more engaging without disrupting its informative intentions, as asymmetry creates a sense of energy and movement [83]. Consequentially, the other elements are all centred, both with respect to their containers and horizontally on the page. However, they encapsulate some asymmetric elements to make the design more interesting. An example is the speed value, which is asymmetrically placed to the left inside the middle of the gauge, while the informative text is placed under and on the right side. This makes the elements appear not centred, making the design more exciting.

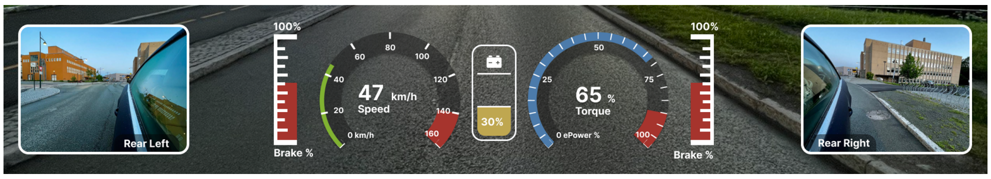
Contrast: *Makes certain parts of your design stand out to end-users.* As already touched upon in the paragraph on Visual Hierarchy, visually guiding users towards the information they are looking for is essential. Therefore, contrast is used frequently in this project to ensure that the most critical information and buttons stand out. This is done using white text on a partly transparent black background, making the black and white contrast. This is done so that information is readable even if the camera shows a bright image, but at the same time, it is not covering all views of the image completely.

Gestalt: *Captures the user's tendency to perceive the whole as opposed to the individual elements.* The Gestalt principles are used in many of the components; the different panels containing data give the user the understanding that all the text inside one rectangle corresponds to the value in the same rectangle, see Figure 4.28c. Additionally, the Gestalt proximity principle is used a lot in the design. One example is the labels on every camera view; they are inside or close to their corresponding boxes, making it intuitive and easy to understand what it labels.

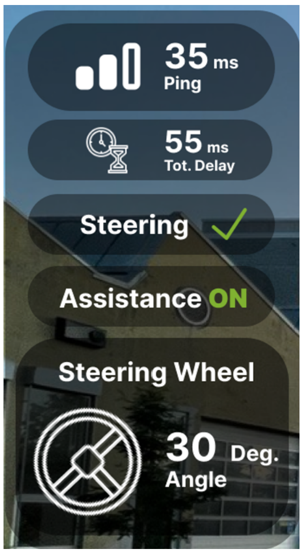
(Norman's Design Principles) To make a product have high usability, it relied on Norman's design principles, in addition to the Visual Design Principles, as a guide during the design process [84]. Norman's principles are well known, designed for humans, and provide guidance in establishing clarity and improving decision-making. Norman depicts six principles: Visibility, Feedback, Affordance, Mapping, Constraints, and Consistency.



(a) Gauge cluster with speedometer on the left, battery status in the middle and throttle-meter on the right



(b) Design of the the instrument panel with the gauge cluster in the middle and rear side views on th



(c) A figure showing the information panel

Figure 4.28: Figure showing different parts of the GUI.

4.5.4 Implementation and Architecture Overview

This sub-section covers the architectural aspects of the desktop application with a graphical user interface and the technologies it is made of.

Desktop App

During the first design phase of the GUI, a decision about making the interface a desktop app was taken. This was a better choice than developing a web interface for the remote control system because of its complexity, or rather the lack of it. Since the remote control system uses ROS and a couple of scripts as backbones for the control and data transmission, it would be a highly complex solution to let the web interface interact with the computer's local scripts, CLI and gather information from ROS. It would also need a client-server setup where the server could introduce more latency, security issues and complexity to the whole system. The system should therefore be somewhat redesigned at its root in order to work properly with a web interface. It is possible and maybe the most modern solution, but not necessary for this project where the subject is a proof of concept and not a widely used commercial product.

Python and Kivy

It was chosen to use Python as the language because it supported ROS, machine learning libraries, object detection and, of course, the possibility of making a good-looking GUI with the framework KivyFigure 3.5.1. The other possible candidate was C++, but it did not have good frameworks for making graphical interfaces. Any other language does not support ROS.

Background and Foreground Processes

In order for the remote control system to work, it is divided into two types of processes; background processes - running independently of a user and needed for the app to work, and the foreground processes - interactable by the user and are mainly the desktop app. Ideally, all processes should have been started automatically by starting the foreground process, but since this is a proof of concept not made for a non-professional, the background processes are also started by the user. Therefore, after starting the background processes with a CLI command and filling in the SSH password, the foreground processes can be started, and the Kivy desktop app will appear.

App Architecture

As good practice, a Kivy application is divided into at least two pieces; the functional "main" Python file and the graphical layout ".kv" file written in the Kivy syntax, which is a Kivy app can be seen as the CSS file in a web-page architecture. The main processes and functionality is happening inside the "main.py" file.

The idea of the desktop app is to have multiple camera streams, car data, information and driver assistance presented and updated in real-time. All the data is served by and retrieved from the background processes section 4.5.4 constantly running.

"**__main__**" **method:** This is the primary method that initializes and runs the whole application. It also creates a ROS node, used for subscribing to different ROS topics, running as part of background processes. Those are needed to get the data sent from the car. The ROS node is called "gui_node" and can be seen in the RQT graph in Figure 4.29.

Camera class: The application consists of multiple camera classes, one for each camera stream with its own updating mechanism - callback. This callback is called every time information is received by "gui_node", subscribing to the particular decoded ROS camera stream. Inside the callback, every frame goes through a reading process, resizing with interpolation and converting from a Numpy array into a Kivy texture, before being visualized as an asynchronous image in the GUI. It is worth mentioning that every callback is running as "@mainthread", meaning that it will never run in the back as a daemon due to the stop of frame update. Figure 4.29 illustrates how the gui_node gets image data through ROS from the car.

"**MainWidget**" **class:** The data such as steering wheel angle, throttle position and speed are also run with their callback functions, only they are inside their own joint class, the "MainWidget". This class stores and updates all kinds of variables needed in the interface, which are not a part of the camera views. In the same class, ping and latency are also evaluated. Those methods show the average ping and total image latency every 3 seconds. The ping is tested with a library called Pingparser, which uses the car's IP address as a reference point. The image latency is calculated every time a callback function with frame processing is called by the node subscribing to the decoded image topic.

"**drivegui.kv**" **file:** This is the file in Kivy syntax used for making the layout, design and animations of the app. It contains a structure of different widgets and classes, together with design-specific variables such as the elements' colour, size, placement, and shape.

RQT Graph

RQT Graph of Full Remote Control System

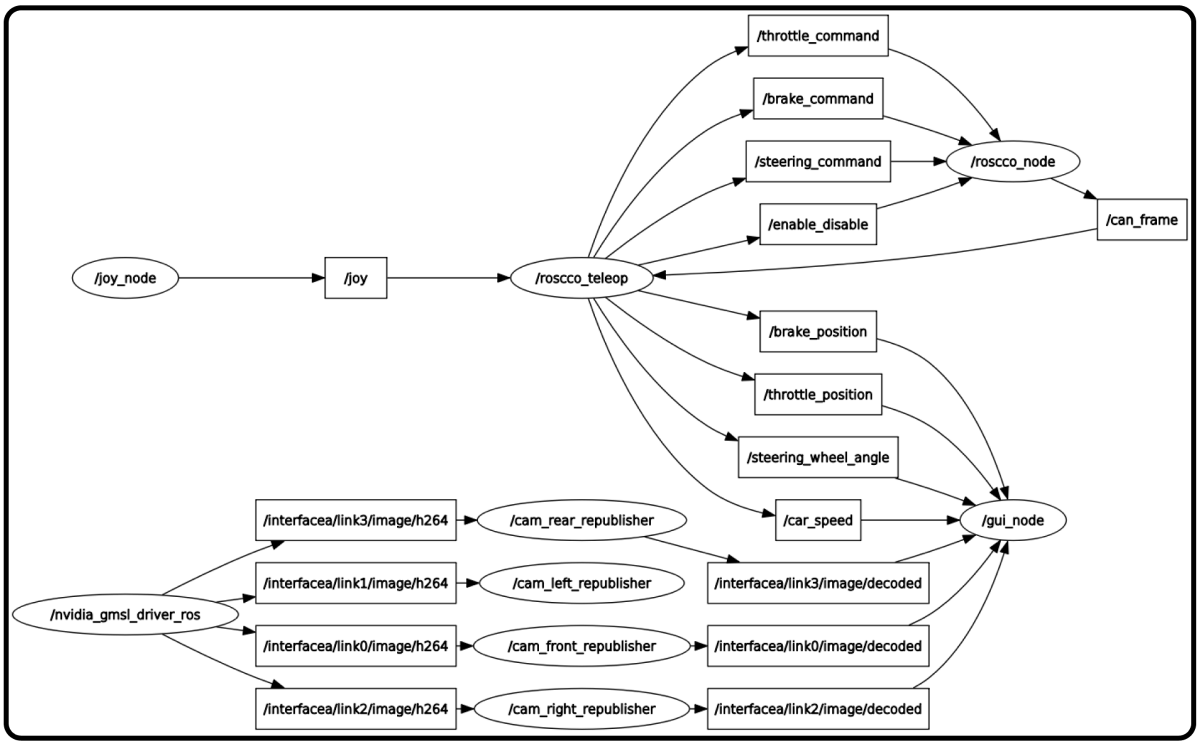


Figure 4.29: Figure showing a RQT graph also called ROS graph, which illustrates nodes that subscribes and publishes topics related to the remote control system included GUI. Link 0-4 corresponds to camera 0-4. This graph shows the entire system with all nodes, topics and connections

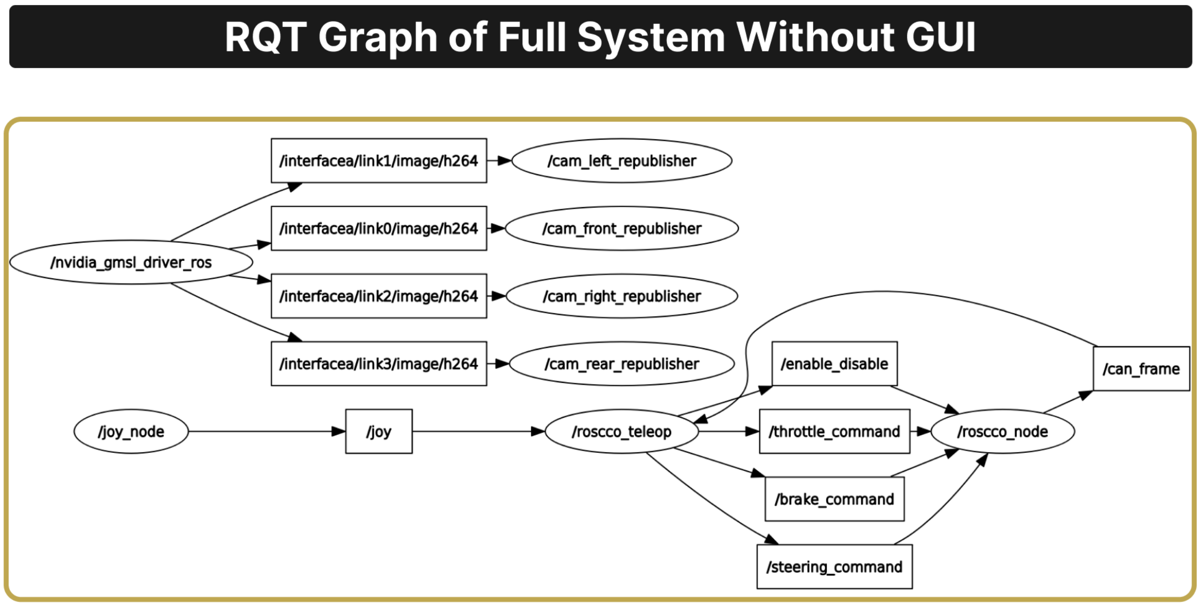


Figure 4.30: Figure showing a RQT graph also called ROS graph, which illustrates nodes that subscribes and publishes topics related to the remote control system without the graphical user interface. This graph shows all nodes and topics except of the graphical user interface and the connection to it.

Chapter 5

Results

The following chapter presents the final results of this project collected during tests and development of the remote control system. The results are set up with respect to goals, research challenges and desired features.

The first goal in section 1.2 was mainly to present and map the architecture of the available NAP-lab vehicle; it is therefore answered throughout the document by the presented hardware architecture of the car in theory section 3.4, and by going through the different parts in methods chapter 4. The second goal in section 1.2 targeted the development process of a proof-of-concept that also needed to be tested and evaluated. The testing and results of the concept are thoroughly described in the following sections.

It is important to notice that a YouTube video was made specifically to cover all the results and tests. This chapter is therefore accompanied by the video in this link, [10]

5.1 Stationary and non stationary test execution

Several tests were carried out to evaluate both individual parts of the architecture and the complete system. However, because the car could not legally drive on a public road or any public place, all driving tests were performed in a private and closed area. That was primarily due to the use of equipment non-certified by the local authorities, such as the developed remote control system. This limitation entailed only one test conducted with the entire system running in its intended and realistic environment, while the other test was done with the vehicle either stationary or with somewhat limited driving space.

The main test was done at the landing site of the Værnes airport in Trondheim, Norway. Its objectives were to remotely operate the car from a nearby office, demonstrate the concept, test its functionality, measure user and usability satisfaction, and stress test the equipment.

It must be noted that there were only three test users of the remote control system: a developer, a professor and a TV reporter. The reason for having a small number of test users is because the concept is in an early phase of the development process, and at this stage, qualitative feedback has a lot more value than quantitative, as stated in the theory of iterative design thinking process [81].

When the test was performed, the landing site at Værnes Airport was unfortunately only equipped with 4G and 4G+ technology, making it impossible to test the 5G connection. Therefore, all 5G testing was done at other places with a 5G connection and with the vehicle either stationary or slowly moving. Nevertheless, the test results gave the developer a large amount of essential data that later can be used for further development.

5.2 The Entire Remote Control System

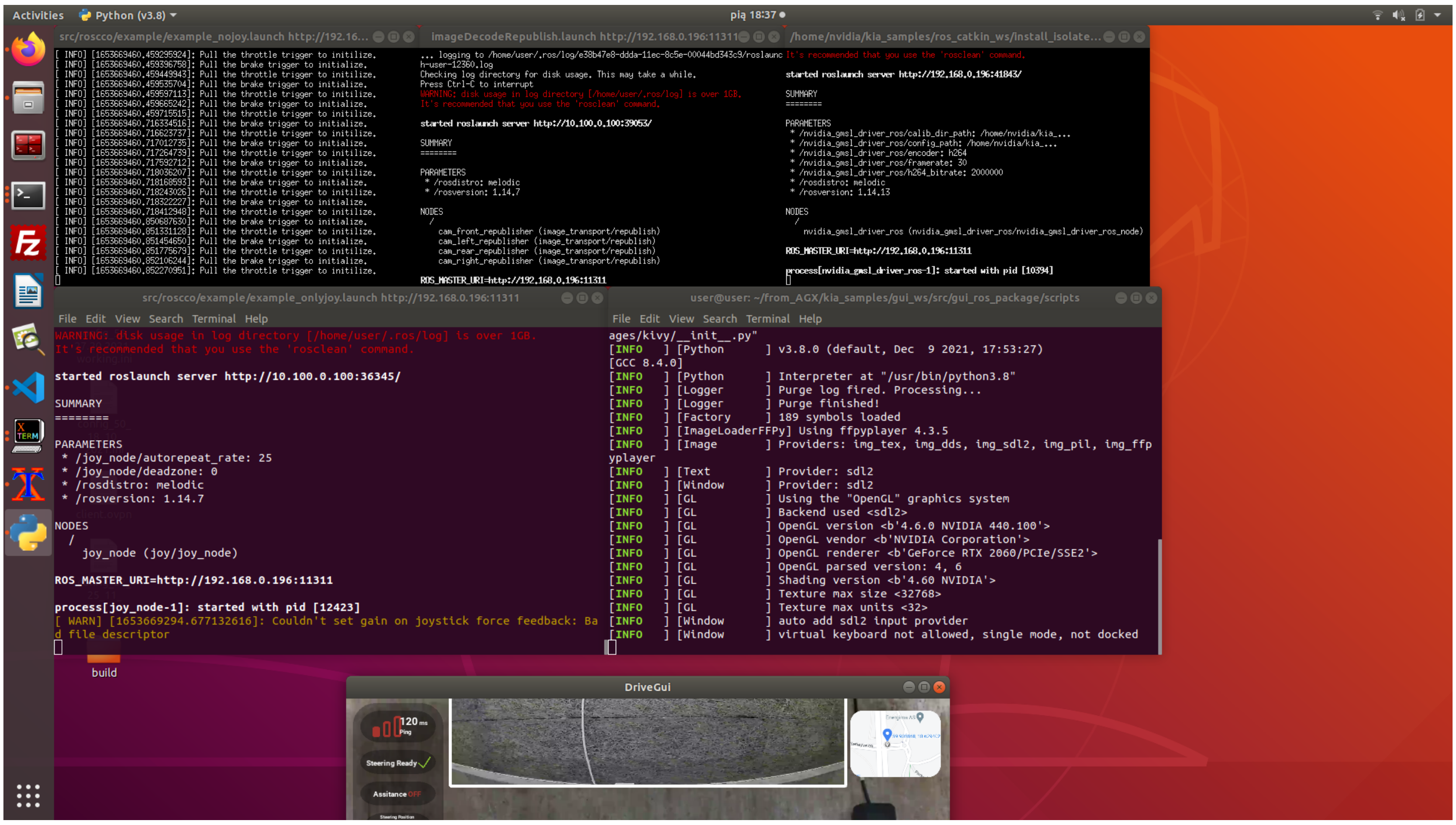
The second goal was to build a complete remote control system that enabled an operator to manoeuvre an autonomous vehicle in an edge case situation such as a "dead-lock". In addition, several features were proposed for the remote control system to make it a complete proof-of-concept.

5.2.1 Implemented features

- Automated P2P WireGuard VPN connection over 5G and 4G cellular network.
- PID controller for precise by wire manoeuvrability, with Logitech G29 steering wheel and pedals.
- Automated system of several interconnected ROS nodes and ".launch" files for remote control of the vehicle over IP.
- Video Transmission with ROS over IP and H.264 compression of 4 GMSL cameras.
- Desktop app displaying real-time data and video streams in a clean and eye-catching graphical user interface.

Assembling all parts explained in chapter 4 together leads to a complete remote control system for the NAP-lab AV. With the use of ROS and a PID controller, the system controls the car from a G29 kit in real-time, while video from 4 cameras mounted on the car is encoded in H.264, transmitted via ROS, decoded and displayed in an informative designed graphical user interface. The connection between the operator and the car is realized with a secure P2P VPN tunnel with a fast UDP WireGuard protocol directly through the 5G/4G cellular network.

All software parts in the remote control system are interconnected through several "bash" scripts and start automatically by running two commands in the CLI: "./autoStartAll" starts all background processes and establishes a connection between operator and car, whereas "./runDriveGui" launches the Kivy app. When both the commands are run, and the password is entered, the result is shown in Figure 5.1. Altogether this forms a completed second goal, with the remote control system being built.


Activities
Python (v3.8)
pią 18:37
src/roscco/example/example_nojoy.launch http://192.16...
imageDecodeRepublish.launch http://192.168.0.196:11311
/home/nvidia/kia_samples/ros_catkin_ws/install_isolate...
src/roscco/example/example_onlyjoy.launch http://192.168.0.196:11311
user@user: ~/from_AGX/kia_samples/gui_ws/src/gui_ros_package/scripts
File Edit View Search Terminal Help
build
DriveGui
120 ms Ping
Steering Ready
Assitance OFF

```
[ INFO] [1653669460.459295924]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.459396758]: Pull the brake trigger to initialize.
[ INFO] [1653669460.459449943]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.459535704]: Pull the brake trigger to initialize.
[ INFO] [1653669460.459597113]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.459665242]: Pull the brake trigger to initialize.
[ INFO] [1653669460.459715515]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.716334516]: Pull the brake trigger to initialize.
[ INFO] [1653669460.716623737]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.717012735]: Pull the brake trigger to initialize.
[ INFO] [1653669460.717264739]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.717592712]: Pull the brake trigger to initialize.
[ INFO] [1653669460.718036207]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.718168593]: Pull the brake trigger to initialize.
[ INFO] [1653669460.718243026]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.718322227]: Pull the brake trigger to initialize.
[ INFO] [1653669460.718412948]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.850687630]: Pull the brake trigger to initialize.
[ INFO] [1653669460.851331128]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.851454650]: Pull the brake trigger to initialize.
[ INFO] [1653669460.851775679]: Pull the throttle trigger to initilize.
[ INFO] [1653669460.852106244]: Pull the brake trigger to initialize.
[ INFO] [1653669460.852270951]: Pull the throttle trigger to initilize.
```

```
... logging to /home/user/.ros/log/e38b47e8-ddda-11ec-8c5e-00044bd343c9/roslaunc
h-user-12360.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
WARNING: disk usage in log directory [/home/user/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://10.100.0.100:39053/

SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.7

NODES
  /
    cam_front_republisher (image_transport/republish)
    cam_left_republisher (image_transport/republish)
    cam_rear_republisher (image_transport/republish)
    cam_right_republisher (image_transport/republish)

ROS_MASTER_URI=http://192.168.0.196:11311
```

```
It's recommended that you use the 'rosclean' command.

started roslaunch server http://192.168.0.196:41843/

SUMMARY
========

PARAMETERS
 * /nvidia_gmsl_driver_ros/calib_dir_path: /home/nvidia/kia_...
 * /nvidia_gmsl_driver_ros/config_path: /home/nvidia/kia_...
 * /nvidia_gmsl_driver_ros/encoder: h264
 * /nvidia_gmsl_driver_ros/framerate: 30
 * /nvidia_gmsl_driver_ros/h264_bitrate: 2000000
 * /rosdistro: melodic
 * /rosversion: 1.14.13

NODES
  /
    nvidia_gmsl_driver_ros (nvidia_gmsl_driver_ros/nvidia_gmsl_driver_ros_node)

ROS_MASTER_URI=http://192.168.0.196:11311

process[nvidia_gmsl_driver_ros-1]: started with pid [10394]
```

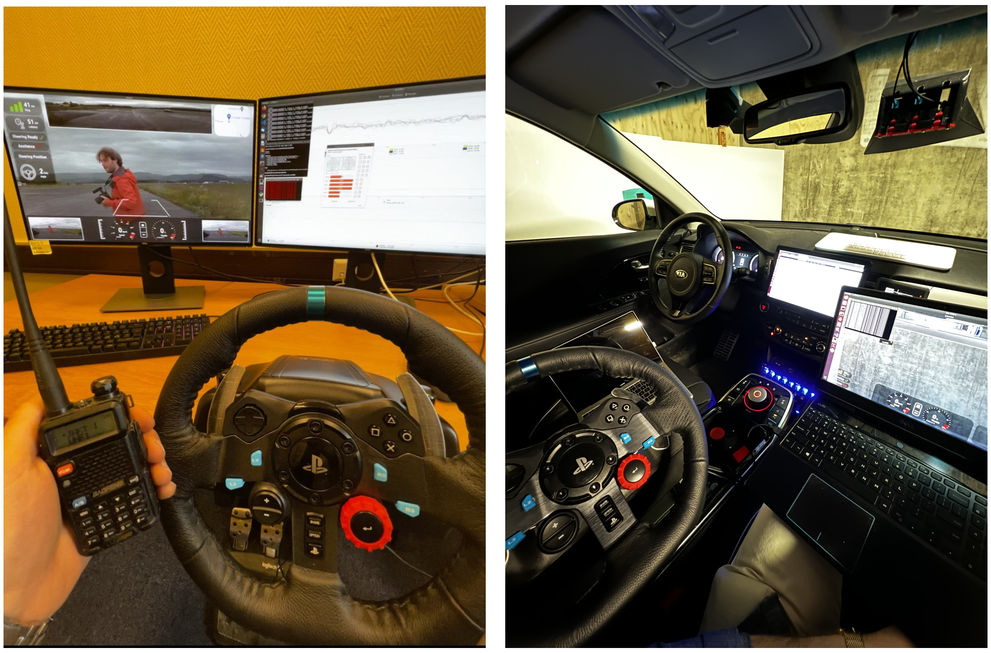
```
WARNING: disk usage in log directory [/home/user/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://10.100.0.100:36345/

SUMMARY
========

PARAMETERS
 * /joy_node/autorepeat_rate: 25
 * /joy_node/deadzone: 0
 * /rosdistro: melodic
 * /rosversion: 1.14.7

NODES
  /
    joy_node (joy/joy_node)

ROS_MASTER_URI=http://192.168.0.196:11311

process[joy_node-1]: started with pid [12423]
[ WARN] [1653669294.677132616]: Couldn't set gain on joystick force feedback: Ba
d file descriptor
```

```
ages/kivy/__init__.py"
[INFO   ] [Python      ] v3.8.0 (default, Dec  9 2021, 17:53:27)
[GCC 8.4.0]
[INFO   ] [Python      ] Interpreter at "/usr/bin/python3.8"
[INFO   ] [Logger      ] Purge log fired. Processing...
[INFO   ] [Logger      ] Purge finished!
[INFO   ] [Factory     ] 189 symbols loaded
[INFO   ] [ImageLoaderFFPy] Using ffpyplayer 4.3.5
[INFO   ] [Image       ] Providers: img_tex, img_dds, img_sdl2, img_pil, img_ffp
yplayer
[INFO   ] [Text        ] Provider: sdl2
[INFO   ] [Window      ] Provider: sdl2
[INFO   ] [GL          ] Using the "OpenGL" graphics system
[INFO   ] [GL          ] Backend used <sdl2>
[INFO   ] [GL          ] OpenGL version <b'4.6.0 NVIDIA 440.100'>
[INFO   ] [GL          ] OpenGL vendor <b'NVIDIA Corporation'>
[INFO   ] [GL          ] OpenGL renderer <b'GeForce RTX 2060/PCIe/SSE2'>
[INFO   ] [GL          ] OpenGL parsed version: 4, 6
[INFO   ] [GL          ] Shading version <b'4.60 NVIDIA'>
[INFO   ] [GL          ] Texture max size <32768>
[INFO   ] [GL          ] Texture max units <32>
[INFO   ] [Window      ] auto add sdl2 input provider
[INFO   ] [Window      ] virtual keyboard not allowed, single mode, not docked
```

Figure 5.1: Figure showing both the background processes and the app running. All systems are nominal and the control over the car is established.

The feature tree presented in the first chapter in Figure 1.1 was almost fulfilled, and the features were successfully implemented onto the concept. However, there were still some discrepancies, especially within the assistance part. Due to limited time and features connected to the operator assistance being highly dependent on object detection software, the operator assistance was not implemented into the system. Nevertheless, a lot of research and exploring was carried out into its components. This part can, therefore, in further development, be addressed and implemented.

Figure 5.2 shows a picture from the operator's point of view when the system is up and running. The screen 5.6c displays the graphical user interface and developer information such as background processes. Figure 5.6d shows the setup inside the car during LAN tests and system development. The graphical user interface is run from a portable computer connected to either the car's router or a 5G hot-spot, depending on the test.



(a) Setup during test of the system in the office nearby the test track.

(b) Setup inside the car during development and tests.

Figure 5.2: Figure shows the operator's setup during test of the system.

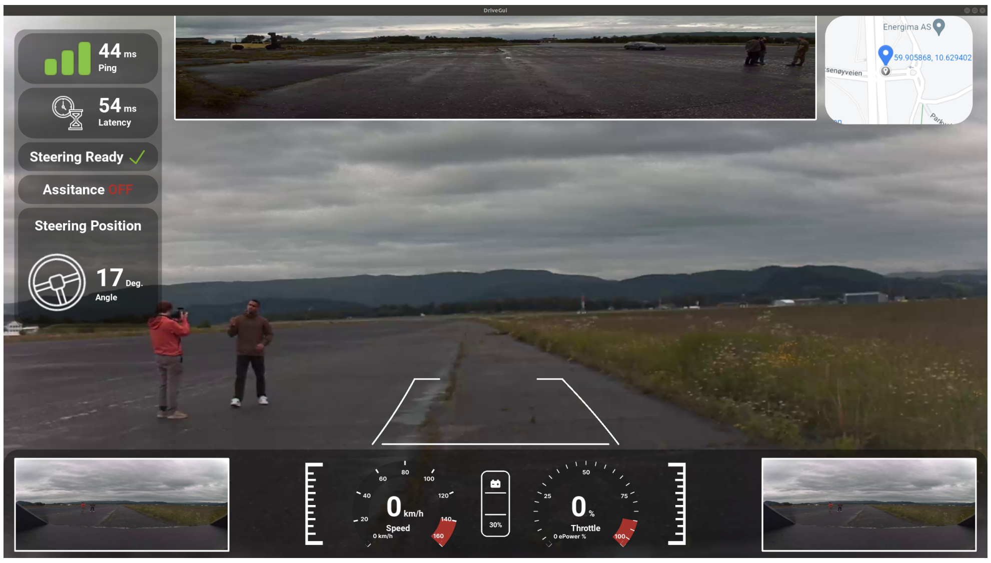


Figure 5.3: Operator's point of view in the GUI of the remote control system. Side cameras is changed to front wide cameras.

5.2.2 Tests

The system was tested for multiple parameters and different cases; whether the connection was LAN, a slow 4G, fast 4G, slow 5G or a fast 5G. That would show how the technology of cellular connection can impact a remote-controlled vehicle and its systems. All test parameters are explained below.

Test parameters with respect to connection type:

- **User Satisfaction** - Measures how satisfied the operator is by using a remote control system.
- **Usability** - Measures how usable the system is, in its intended environment, which in this case is a real traffic situation.
- **Total Latency** - Round trip time (ping) in addition to the camera processing latency on the remote computer (Process latency calculated by metering time it takes for a camera callback function to process one frame on the remote computer)
- **Ping** - Round trip time from the remote to the car and back.
- **Total Satisfaction** - Measures the total satisfaction of the system based on the given results in the rest of the table.

As this test is a proof-of-concept, all satisfaction measurements are exclusively qualitative and based on the developer's opinion. User satisfaction, usability and total satisfaction use the satisfaction scale from 1 to 5, where one means that the system performed poorly and five is that the performance was outstanding. During the main test, the operator drove the car in straight lines at higher speeds and slower speeds on winding tracks. Everything such as steering tests, ping and latency, except for the actual driving, was performed on the stationary experiments. The objective was to understand how well the system behaves in different situations.

5.2.3 Results

Figure 5.4 shows a table with all the results of the performed tests for the entire system. The results are discussed and explained in the subsections below, divided into connection types.

LAN

On LAN, which means that the operator is inside the car and connected to the car's internal router, the results were the best, which was expected. However, it is still important because it shows that the system's overhead is low enough to not impose any problems on the functionality, thereby receiving a high score on both usability, total satisfaction and user satisfaction.

On the other hand, the user satisfaction was not outstanding because the user interface and camera placement were shown to have some design flaws that made it hard to operate a vehicle at low speed and corners.

5G

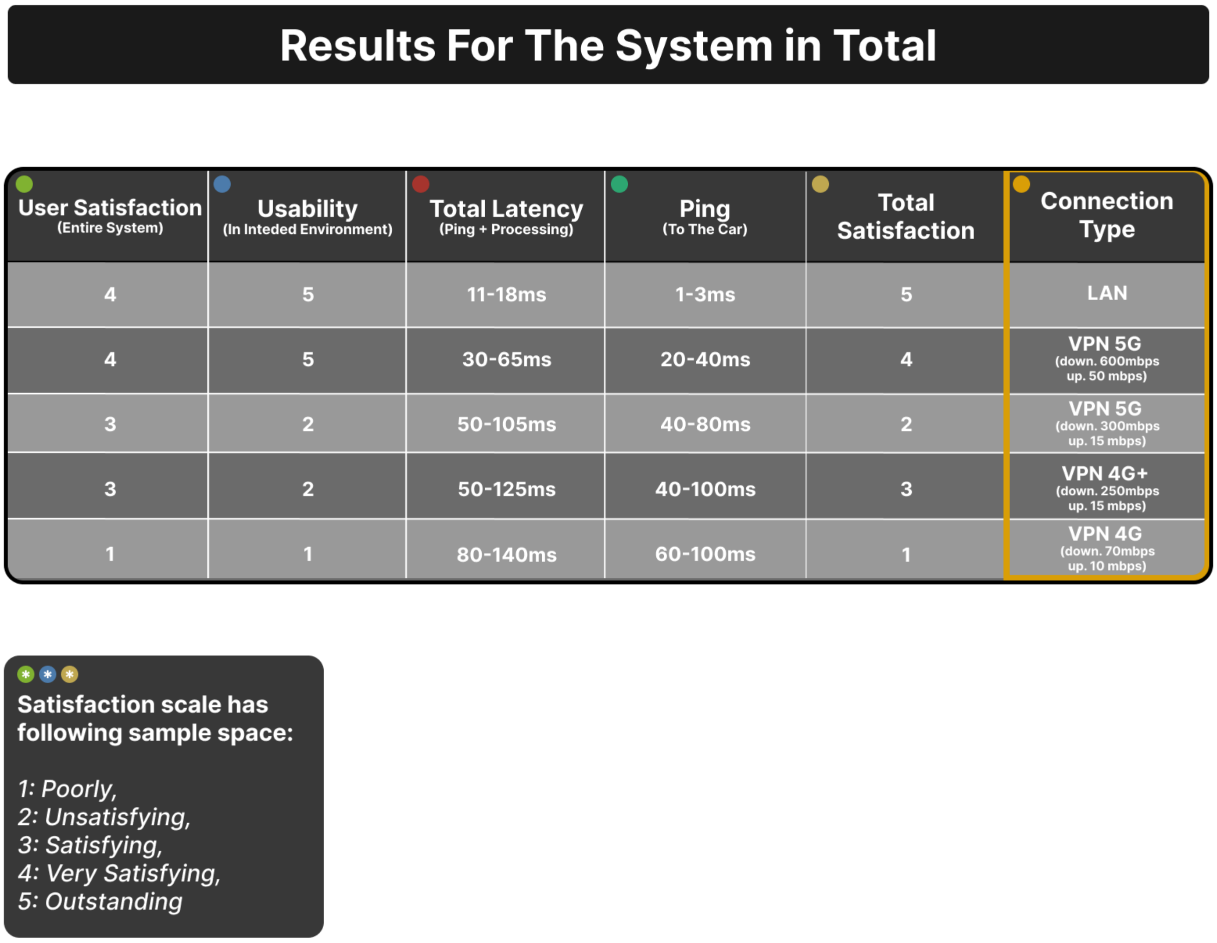
On the 5G with a high download and upload speed, the results are almost identical except for the latency and ping. The processing is still happening in the same amount of time on the remote machine, but the round trip time is significantly higher than on LAN. Nevertheless, the results are still very acceptable, with the total latency only being between 30-65ms, making the driving easy and without feeling lagging. Still, the goal for total latency is to be below 50ms for real-time operation, considering that any higher than 50ms in 60 km/h will lead to 0.8m or more distance being travelled by car. That can entail a threat of being unable to manoeuvre the car correctly, as earlier discussed in subsection 2.1.4. That leads to the "total satisfaction" missing the point.

On the 5G with lower bandwidth and higher ping, the system started to obtain a high latency, which implicated difficulties in the car's remote manoeuvrability and dropped satisfaction points on all parameters. With this latency, driving the car on the road was possible, and the system worked. However, it was slightly more difficult, especially when the latency could peak at 100ms. Using the system in its intended environment is therefore not optimal but possible.

4G

Using the faster 4G connection did not change the performance much compared to the low-end 5G. Ping did not have a considerable increase, making it highly plausible to operate the car, but not as an optimal solution. On the "airport test", the car drove at 60km/h and higher with the operator still having control, which shows that the performance is the same as on low-end 5G.

The lower 4G, on the other hand, was not usable for driving. The ping and the total latency increased to between 80-140ms, making it harder for the operator to react in real-time. However, the worst part was probably the amount of packet drop induced by the low upload speed. In the worst case, the picture could freeze and the system collapse in the absence of a stable packet stream. All this gives 4G, the lowest score in the test, making it unusable for such real-time operations in traffic.



Results For The System in Total

User Satisfaction (Entire System)	Usability (In Inteded Environment)	Total Latency (Ping + Processing)	Ping (To The Car)	Total Satisfaction	Connection Type
4	5	11-18ms	1-3ms	5	LAN
4	5	30-65ms	20-40ms	4	VPN 5G (down. 600mbps up. 50 mbps)
3	2	50-105ms	40-80ms	2	VPN 5G (down. 300mbps up. 15 mbps)
3	2	50-125ms	40-100ms	3	VPN 4G+ (down. 250mbps up. 15 mbps)
1	1	80-140ms	60-100ms	1	VPN 4G (down. 70mbps up. 10 mbps)

Satisfaction scale has following sample space:

1: Poorly,
2: Unsatisfying,
3: Satisfying,
4: Very Satisfying,
5: Outstanding

Figure 5.4: Figure showing a table with results of ping, total latency, user satisfaction, usability of the system and total satisfaction of the system with regard to the connection type.

5.3 P2P WireGuard VPN Connection Over 5G Cellular Network

The research challenge 2.1 set in section 1.2 was about making a secure, stable and fast connection from the remote control computer to the car. As described in depth in section 4.1 the connection was established through a VPN with a P2P design, using SSH to enable the launch of systems related to the control of the car.

5.3.1 Tests

The system was tested for two parameters; whether the connection was through a VPN tunnel or not. The test aimed to understand how the VPN performs and its average latency on both the vehicle and the remote computer. All test parameters are explained below.

Test parameters with respect to connection type:

- **Vehicle Computer** - Latency in milliseconds inside the vehicle computer on a given test.
- **Remote Computer** - Latency in milliseconds inside the remote computer on a given test.
- **Test Type** - Type of the test that is ran both devices.
- **Connection Type** - Shows whether the connection is through the VPN tunnel or not.

Together with the system testing, there was also a test measuring cellular connection in the city of Trondheim at two different places; at the NTNU university and Ila. The measurements were done on both 4G and 5G, showing low-end and high-end 4G/5G. The total results are shown in Figure 5.6.

5.3.2 Results

The VPN connection result shown by the table in Figure 5.5 gives a good overview of different latency present in the system. Looking at the lower table, the connection speed on the remote computer is high due to the use of Ethernet. The vehicle computer also has a high download speed but a low upload speed because of the use of low-end 5G and the test facility being located inside an underground garage. Nevertheless, the upper table does show important results related to how much overhead the VPN tunnel can create. By comparing the latency of pinging "google.com" on remote and on the vehicle, the difference is about 37ms. That means the tunnel creates an overhead of around 21ms; 37ms minus the 16ms ping without VPN. The same results are revealed when comparing the delay of one ROS joystick message minus the latency of pinging "google.com" in the vehicle. These results show an overhead created by the VPN tunnel with around 20-30ms on top of the ISP-induced latency. These are promising results when considering a high-end 5G cellular network that can have a 10-20ms ping (5.6) and, together with VPN, sum up to about 40-50ms. As long as the total delay of the system is 50 or under 50 ms, the car is highly manoeuvrable by the operator [23].

5.4 PID Controller and ROS System For Manoeuvring Vehicle Over IP

The research challenge 2.2 set in section 1.2 was to enable vehicle brake, steering and throttle control over IP. That was done by building an interconnected ROS system with an additional PID controller inside the car for precise manoeuvring, described in depth in section 4.2. This section shows the control part of the system going through multiple tests.

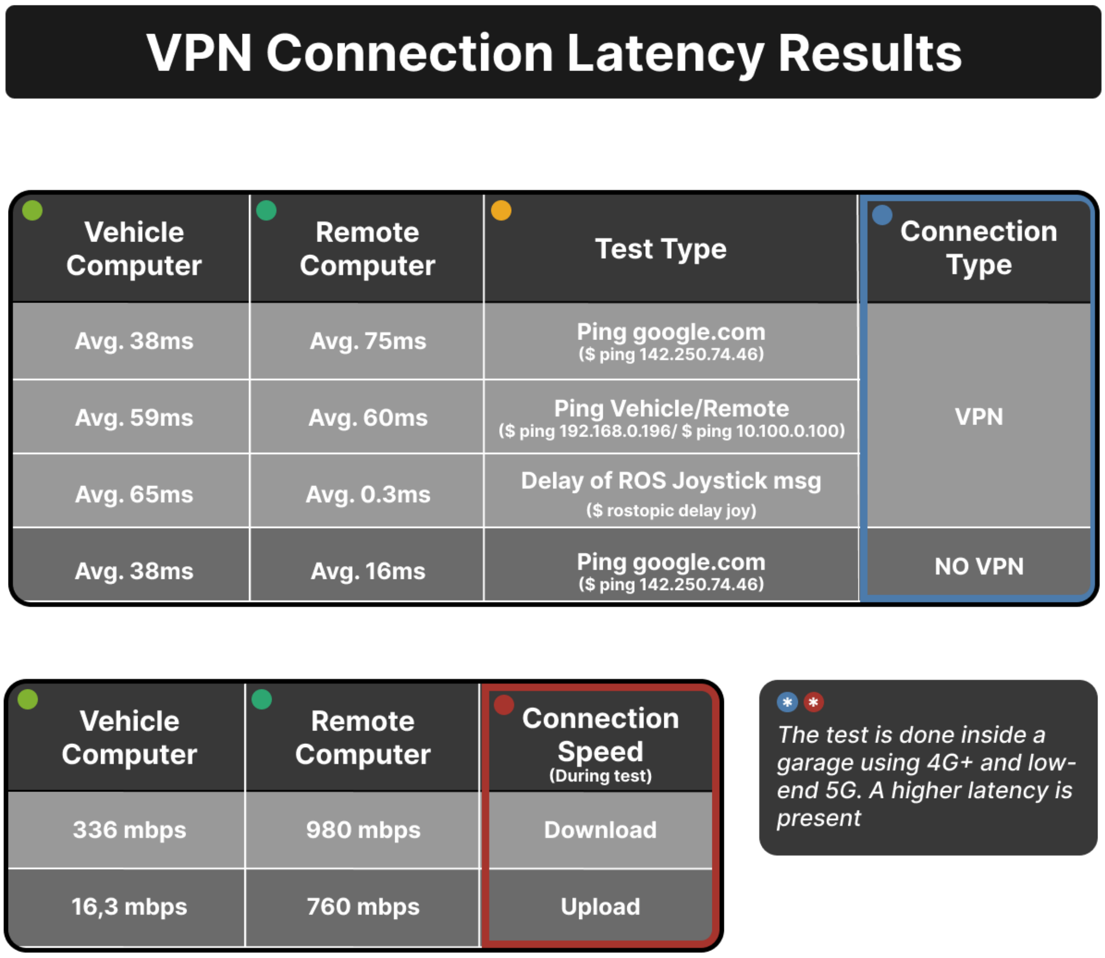
5.4.1 Tests

The vehicle control system was tested for multiple parameters and, for different cases, different driving speeds, ranging from 0-60km/h. That would show how good and precise the system is at manoeuvring and controlling the car. All tests were performed with the car connected to LAN, 4G+ and 5G cellular networks. As a result, the test shows an average value of several different tests.

Test parameters with respect to speed:

- **User Satisfaction** - Measures the operator's satisfaction with the control system's manoeuvrability, response and feel.
- **Steering Angle Deviation** - Divination between the operator's G29 steering wheel and the native wheel inside the car.
- **Steering** - Operator's satisfaction with using the proposed control system with the G29 steering wheel to control the car over IP.
- **Brakes** - Operator's satisfaction with using the proposed control system with the G29 brakes to stop the car over IP.
- **Throttle** - Operator's satisfaction with using the proposed control system with the G29 gas pedal to accelerate the car over IP.
- **Total Satisfaction** - Measures the total satisfaction of the system based on the given results in the rest of the table.

As this tests a proof-of-concept, all satisfaction measurements are exclusively qualitative and based on the developer's opinion. User satisfaction, usability and total satisfaction use the satisfaction scale from 1 to 5, where one means that the system performed poorly and five is that the performance was outstanding. The test was carried out by driving the vehicle around, making different manoeuvres, braking and accelerating at different speeds.



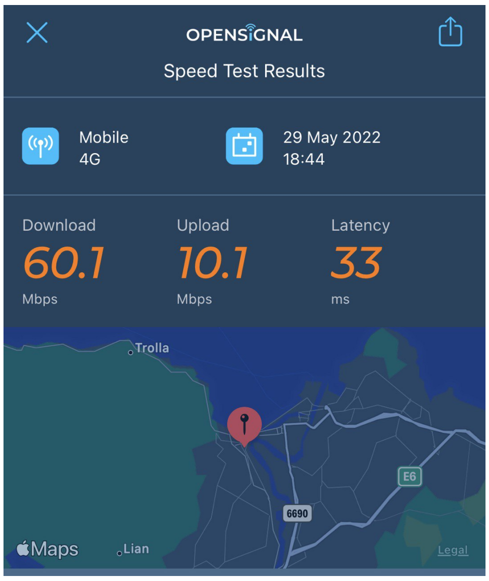
VPN Connection Latency Results

Vehicle Computer	Remote Computer	Test Type	Connection Type
Avg. 38ms	Avg. 75ms	Ping google.com ($ ping 142.250.74.46)	VPN
Avg. 59ms	Avg. 60ms	Ping Vehicle/Remote ($ ping 192.168.0.196/ $ ping 10.100.0.100)	
Avg. 65ms	Avg. 0.3ms	Delay of ROS Joystick msg ($ rostopic delay joy)	
Avg. 38ms	Avg. 16ms	Ping google.com ($ ping 142.250.74.46)	NO VPN

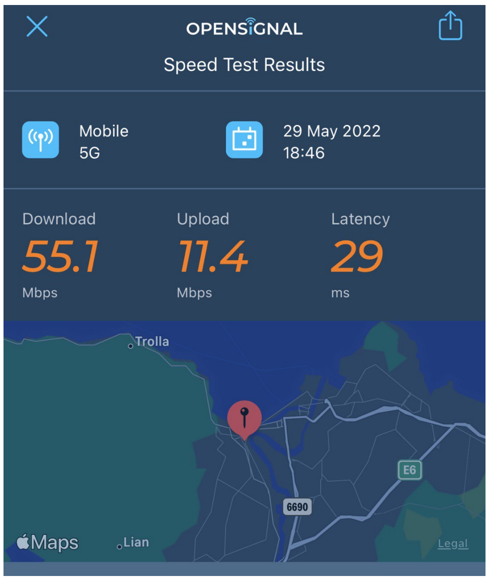
Vehicle Computer	Remote Computer	Connection Speed (During test)
336 mbps	980 mbps	Download
16,3 mbps	760 mbps	Upload

* * *The test is done inside a garage using 4G+ and low-end 5G. A higher latency is present*

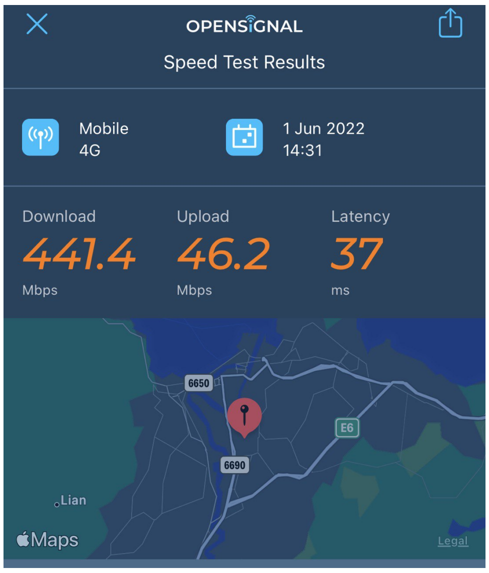
Figure 5.5: In this figure there are 2 tables, the upper one showing Connection latency with or without use of the VPN on both the vehicle computer and the remote computer. The lower table shows connections speed on both computers during the testing.


OPENSIGNAL
Speed Test Results
Mobile 4G
29 May 2022 18:44
Download 60.1 Mbps
Upload 10.1 Mbps
Latency 33 ms

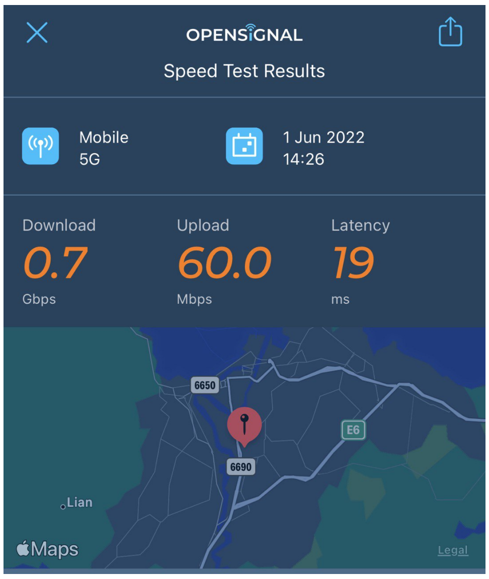
(a) Measurements of 4G connection with OpenSignal at Ila, Trondheim.


OPENSIGNAL
Speed Test Results
Mobile 5G
29 May 2022 18:46
Download 55.1 Mbps
Upload 11.4 Mbps
Latency 29 ms

(b) Measurements of 5G connection with OpenSignal at Ila, Trondheim.


OPENSIGNAL
Speed Test Results
Mobile 4G
1 Jun 2022 14:31
Download 441.4 Mbps
Upload 46.2 Mbps
Latency 37 ms

(c) Measurements of 4G with OpenSignal at NTNU, Trondheim.


OPENSIGNAL
Speed Test Results
Mobile 5G
1 Jun 2022 14:26
Download 0.7 Gbps
Upload 60.0 Mbps
Latency 19 ms

(d) Measurements of 5G with OpenSignal at NTNU, Trondheim.

Figure 5.6: Figure shows 4 measurments at two different locations in Trondheim with both 4G and 5G in order to compare the connection type on speed and latecny.

5.4.2 Results

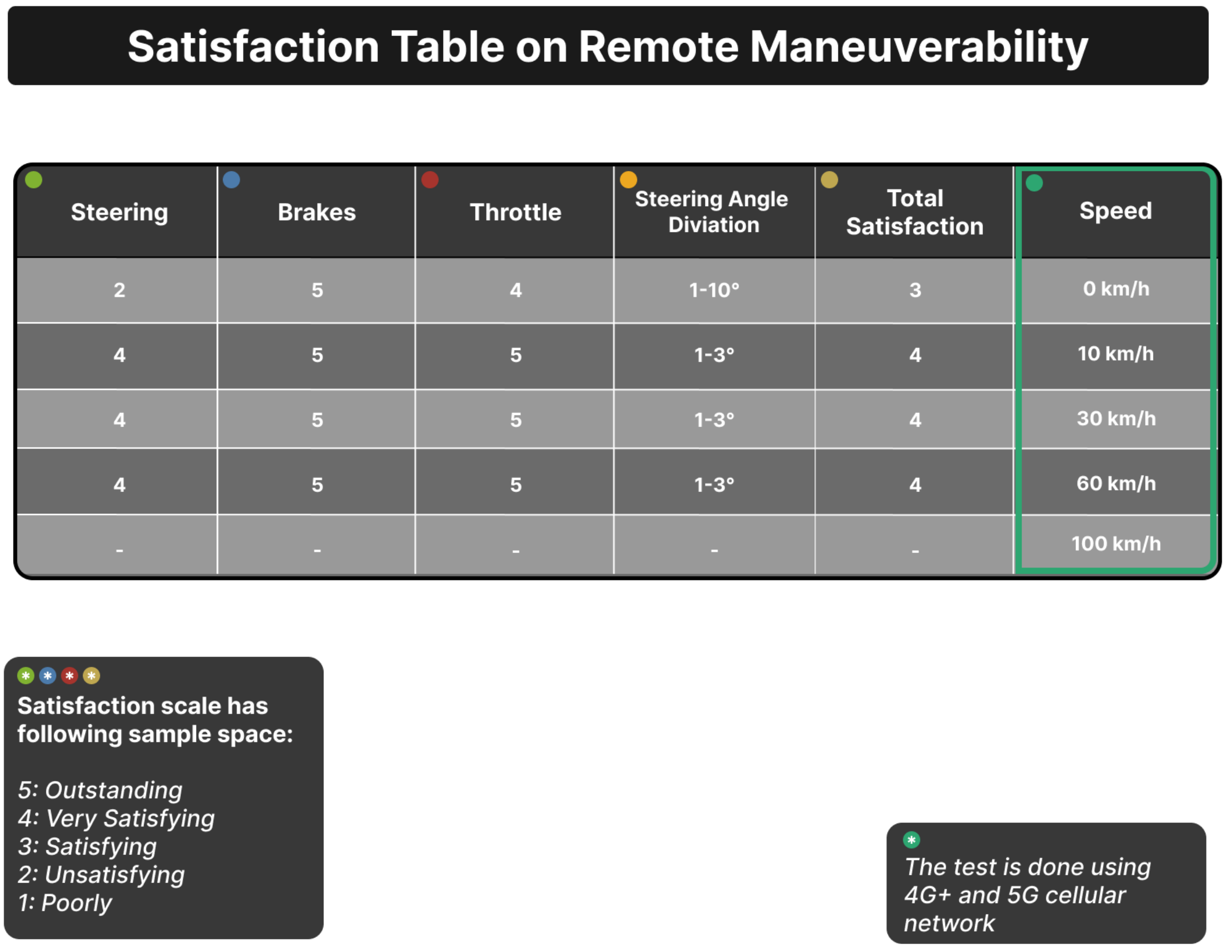
The results of the performed remote manoeuvrability test are presented in the table in Figure 5.7, and all were promising. When the system was tested in non-stationary, all parts worked well, with the steering wheel angle showing a deviation of about 1-3 degrees if started at identical positions.
There was no delay in the system, and all parts felt exactly like they were working in real-time, even when driving at high speeds such as 60 km/h. However, if the remote G29 steering wheel was offset compared to the native steering wheel in the car, it could have a larger offset during the rest of the session. In addition, at a standstill, the PID controller did struggle a bit to hold the desired angle. The car's steering wheel could start wobbling when turning the remote wheel in a very fast or a very slow manner. The reason is probably the high friction from the tires not letting the wheels turn evenly. That is why steering did not get an outstanding satisfaction factor in the tests, especially not in the stationary case. Nevertheless, the brakes and throttle did surprisingly good and had an outstanding result in all cases. The remote pedals had a response factor that could be close to the native inside the car and were very easy to interact with.

5.5 Video Transmission Over ROS With H.264 Encoding

The research challenge 2.3 set in section 1.2 ensure a stable real-time video stream from the cameras mounted on the car and connected to the Nvidia Drive platform. That was done using a third-party library "nvidia_gmsl_driver_ros" to encode and convert the GMSL camera stream into ROS "sensor_msgs/Image" messages before being transported to the remote computer and decoded. The detailed explanation of the method can be read in section 4.3.

5.5.1 Tests

This part of the system was somewhat challenging to test. Therefore, the results are based on qualitative visual and logical interpretation, in addition to the overall performance. That is because the main work related to encoding and decoding is either done by the "nvidia_gmsl_driver_ros" library or ROS transport libraries; testing the process in detail is therefore not accessible.



Satisfaction Table on Remote Maneuverability

● Steering	● Brakes	● Throttle	● Steering Angle Diviation	● Total Satisfaction	● Speed
2	5	4	1-10°	3	0 km/h
4	5	5	1-3°	4	10 km/h
4	5	5	1-3°	4	30 km/h
4	5	5	1-3°	4	60 km/h
-	-	-	-	-	100 km/h

* * * *
Satisfaction scale has following sample space:

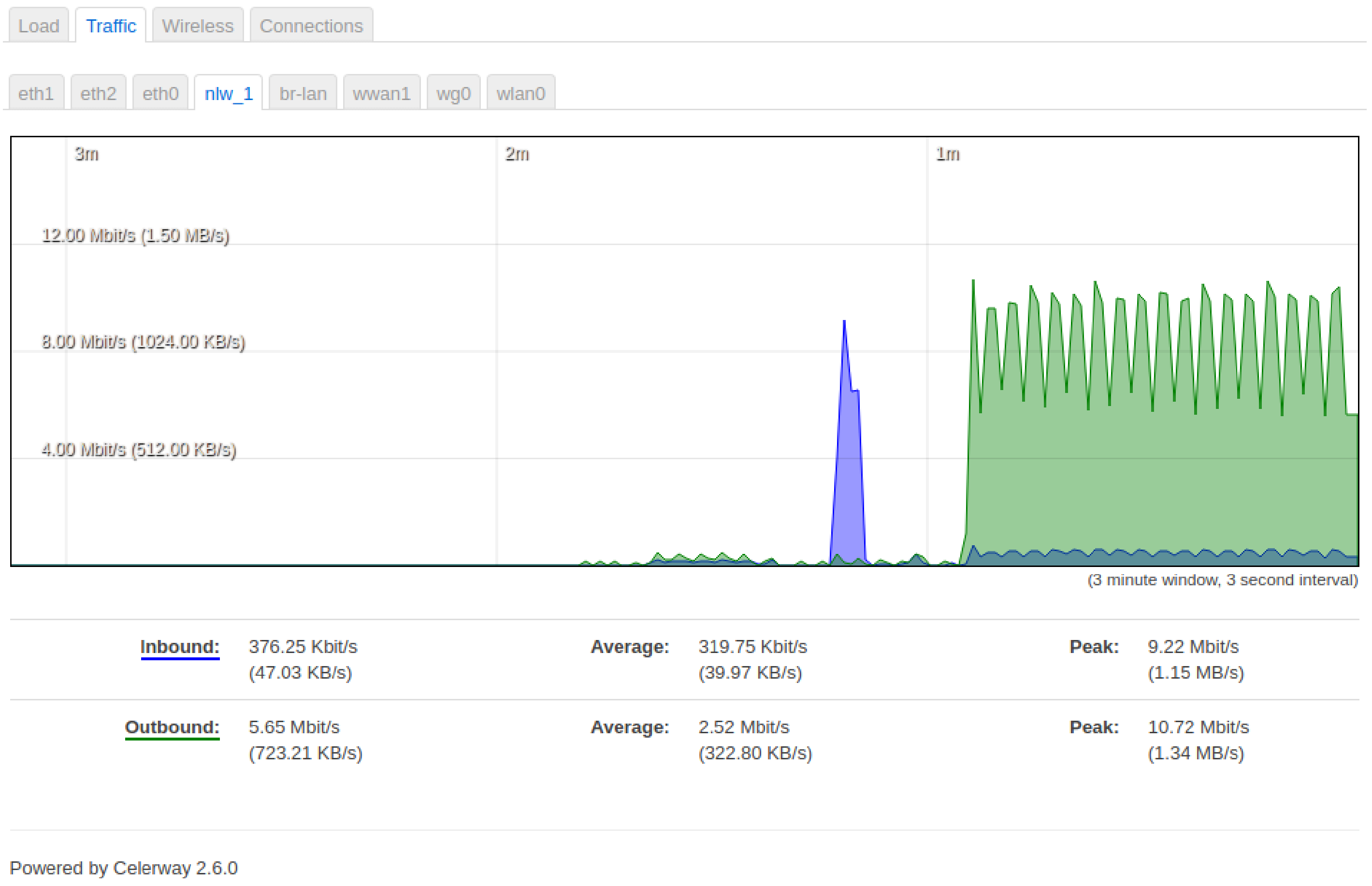
5: Outstanding
4: Very Satisfying
3: Satisfying
2: Unsatisfying
1: Poorly

*
The test is done using 4G+ and 5G cellular network

Figure 5.7: Figure with a table showing the results of testing of the remote maneuverability. All tests are done using 4G+ and 5G.

5.5.2 Results

The video transmission is, as mentioned in subsection 4.3.2 compressed in order to be sent over IP. The compression and conversion are CPU and GPU intensive, showing that up 60% of the CPU in the vehicle is loaded. That makes more than 4-5 cameras taught for the machine. On the remote computer, it is the displaying and not the decoding that is heavily GPU and CPU intensive, showing that a PC without a dedicated GPU cannot display more than one camera. The bandwidth used by the video transmission is about 8mbps, shown in Figure 5.8. That infers a bandwidth shortage due to the cellular network, especially 4G, having much lower upload than download speed. Four cameras displayed simultaneously is therefore only possible with a powerful remote computer and a fast 4G+ or 5G connection, where upload speed is higher than 12mbps. The measurement in Figure 5.6 shows that this is not always achievable, resulting in slow video transmission or even a system collapse.



Inbound:	376.25 Kbit/s (47.03 KB/s)	**Average:**	319.75 Kbit/s (39.97 KB/s)	**Peak:**	9.22 Mbit/s (1.15 MB/s)
Outbound:	5.65 Mbit/s (723.21 KB/s)	**Average:**	2.52 Mbit/s (322.80 KB/s)	**Peak:**	10.72 Mbit/s (1.34 MB/s)

Powered by Celerway 2.6.0

Figure 5.8: The bandwidth used by the video transmission is shown on this screenshot from the Celerway Arcus router's interface.

5.6 GUI Desktop App in Kivy

The research challenge 2.4 set in section 1.2 describes the need for a graphical user interface for the remote control system in order for the operator to achieve reliable control. That was carried out by developing a desktop app with all the necessary information to achieve a good overview of the system and integrate the other parts. The desktop app is described in depth in section 4.5.

5.6.1 Tests

The desktop app testing was done by the same tests when testing the complete system, as described in section 5.2. User satisfaction was based on the system and how it was to interact with the graphical user interface. Optimally, a user interface should be tested by many users to detect errors or misleading design elements. However, since this is a proof-of-concept developed in a matter of design thinking, the feedback on GUI is highly qualitative and received from the developer, professor and a reporter on the test.

5.6.2 Results

The satisfaction results can be seen in Figure 5.4 showing that the operators were mostly happy with the proposed interface, but only with a good cellular connection. The reason was laggy performance and the risk of the system crashing when the connection was at the lower end. Nevertheless, the app fully displayed four cameras simultaneously with a lot of information updating in real-time. That was marked as working smoothly all the time, such as the speedometer and the steering angle.

Regarding design elements, the feedback was positive, and the interface was mostly easy to use. However, the battery indicator showing the current state of the charge was not fully implemented and not working, with the map at the top right corner being only a mock-up. The implemented trajectory projections were also not working as intended and showed the wrong trajectory when turning the steering wheel more than 45 degrees. However, the biggest problem was the camera placement and its displaying. Because the primary front camera did not show the vehicle's dimensions or parts, it was rather challenging to understand whether the car would crash or drive beside an object. The trajectory projections were, therefore, not enough even when they worked as intended. See Figure 5.3

5.7 Object Detection With Yolo V5 in Real Time

The research challenge 2.5 set in section 1.2 was to explore and map possible methods and algorithms that can be used as a part of a future operator-assistance system. That has been done by testing different algorithms, both pre-trained and trained by developer's data, collected from driving the NAP-lab car. In addition, an additional test was conducted to see object detection potential as an integrated part of the remote control system.

5.7.1 Tests

At first, all three object detection algorithms, Yolov3 and Yolov3 Tiny and Yolov5, were tested pre-trained with the COCO dataset. That was done to check which would perform the best and whether the patching affects the performance.

Then, to take the exploration a bit further, the train and validation set was created and divided as shown in Figure 5.12a, 100 frames for training and 20 for validation for ten different videos. This dataset was then loaded into Yolo5 and trained in the network. That was done to see what performance object detection can have if it is trained specifically for its purpose.

Lastly, the operator assistant is considered an essential part of the remote control system in further development. Therefore an extra experiment was carried out to prove its potential. First, Yolov5 was installed and integrated into the graphical user interface to test if it could work with the whole remote control system. For this experiment, Yolo5 was used as pre-trained and ran on GPU with CUDA. Then, it was implemented into the callback that processes the primary video stream from the camera inside the remote computer. That was done before the frame was fully processed and displayed in the desktop app; see Figure 5.13 for the result.

5.7.2 Results

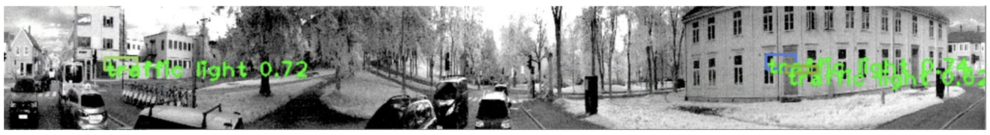
In order to make simulated real-time testing, some extra functionality was added in the form of a loop where every frame was loaded as fast as the CPU could and processed the same way as described earlier. The list subsection 5.7.2 shows all the different trails together. In addition, Figure 5.9 and Figure 5.10 illustrate the trials with screenshots of one frame from the simulated object detection. It is worth mentioning that Yolov5 was more of a black box compared to Yolov3, so many of the functions like scaling were done automatically by Yolo5, and results show, therefore, that patching on Yolov5 was not efficient as on v3, instead made it worse because of the lower speed.

The results was very good on the validation set, 42% mAP, showed in Figure 5.12b, but not so good on the test set, only 14%, in Figure 5.11c. Either way, it was still better than all the Yolo3 variants and the pre-trained Yolov5 when looking at the screenshot, comparing it to the others, in Figure 5.10c. One of the possible reasons that Yolov5 worked better on the trained data without any patching or pre-processing is that it has implemented the Auto anchor functionality, where the code automatically looks at the given anchors(labels) and compares them against the data. If any of them fall below a certain threshold, the algorithm will change them, make new, more appropriate anchors, and train the model on them instead.

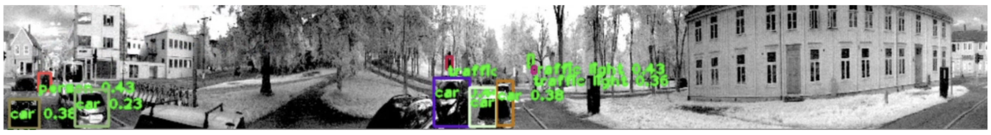
- **Pretrained Yolov3 *without* patching**
  - Pretrained on COCO 2012
  - Prediction as black box(By Yolo)
  - Bounding box and label generation and image crop/scaling by own algorithm in python with OpenCV.
  - **Results**:
    - Predictions: Few and all are bad
    - Speed: To slow in real-time(On CPU)
    - Illustrated in Figure 5.9a
- **Pretrained Yolov3 *with* patching**
  - Pretrained on COCO 2012
  - Prediction as black box(By Yolo)
  - Bounding box and label generation and image crop/scaling by own algorithm in python with OpenCV.
  - **Results**:
    - Predictions: Many and more correct
    - Speed: To slow in real-time(On CPU)
    - Illustrated in Figure 5.9b
- **Pretrained Yolov3 Tiny *with* patching**
  - Pretrained on COCO 2012
  - Prediction as black box(By Yolo)
  - Bounding box and label generation, patching and image crop/scaling by own algorithm in python with OpenCV.
  - **Results**:
    - Predictions: Few and somewhat correct
    - Speed: Very fast in real-time(On CPU)
    - Illustrated in Figure 5.9c

- **Pretrained Yolov5 *without* patching**
  - Pretrained on COCO 2017
  - Prediction as black box(By Yolo)
  - Bounding box and label generation as black box(By Yolo)
  - **Results**:
    - Predictions: Medium amount and many correct
    - Speed: Very fast in real-time(On CPU)
    - Illustrated in Figure 5.10a
- **Pretrained Yolov5 *with* patching**
  - Pretrained on COCO 2017
  - Prediction as black box(By Yolo)
  - Bounding box and label generation as black box(By Yolo)
  - Patching by own algorithm in python with OpenCV.
  - **Results**:
    - Predictions: Medium amount and many correct
    - Speed: Very slow in real-time(On CPU)
    - Illustrated in Figure 5.10b
- **Trained Yolov5 *without* patching**
  - Trained on custom dataset, see Figure 5.12a
  - Prediction as black box(By Yolo)
  - Bounding box and label generation as black box(By Yolo)
  - Patching by own algorithm in python with OpenCV.
  - **Results**:
    - Predictions: Many and a lot correct
    - Speed: Very fast in real-time(On CPU)
    - Illustrated in Figure 5.10c

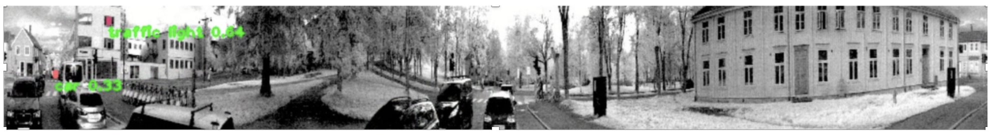
At last, in this section, the results of object detection with Yolov5 being run as a part of the remote control system showed that it works, but only when computing one camera. Even though the computer was using GPU for object detection, several streams, together with processing and rendering, were computationally heavy for any of the tested computers. The GPU used in the test was RTX 2060, both in the laptop and the desktop computer. The extra overhead and latency created were around 22ms if the object detection was turned on, which can be concluded as fast and suitable for real-time remote control, but only if the cellular connection is not having high latency. Because it could sum up to very high latency in tandem. Figure 5.13 shows a picture of the object detection perform detection in real-time inside the remote control GUI.



(a) Pretrained Yolov3 with no patching

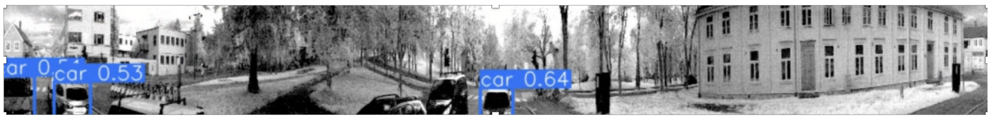


(b) Pretrained Yolov3 with patching

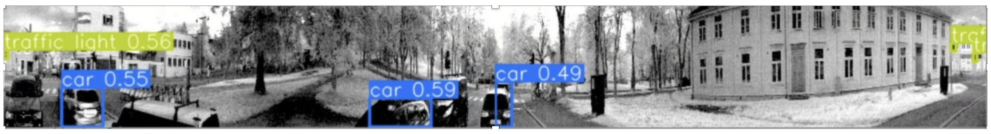


(c) Pretrained Yolov3 Tiny with patching

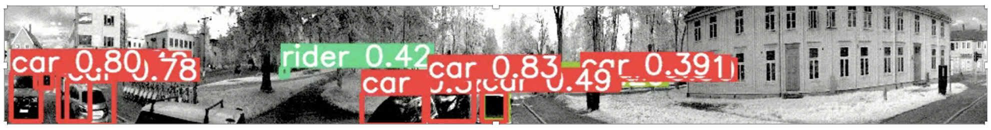
Figure 5.9: A figure showing the labels predicted by different version of pre-processing with Yolov3 and Yolov3 Tiny.



(a) Pretrained Yolov5 with no patching

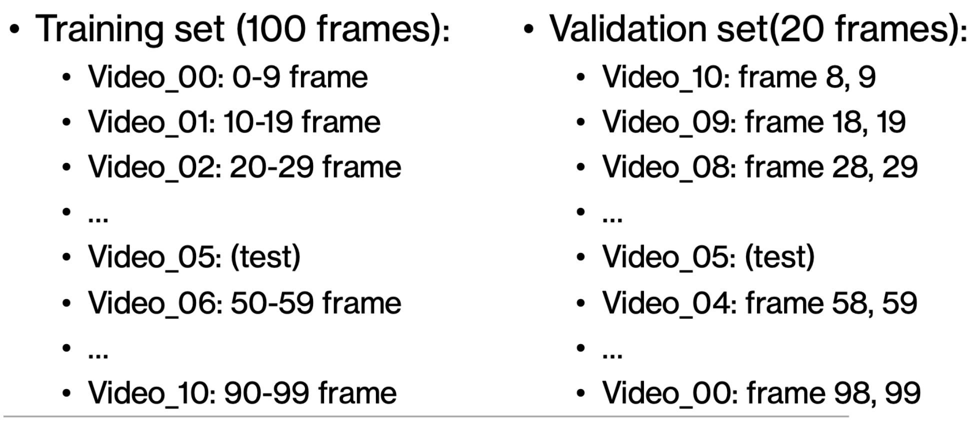


(b) Pretrained Yolov5 with patching



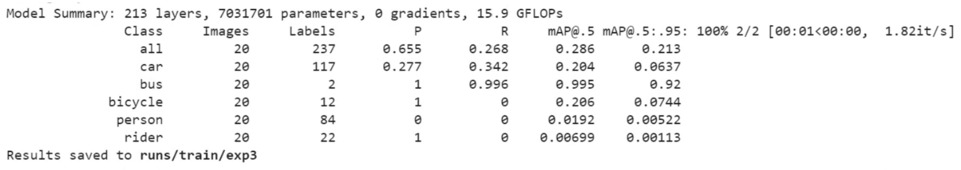
(c) Trained Yolov5 with no patching

Figure 5.10: A figure showing the labels predicted by different version of pre-processing, training with Yolov5.



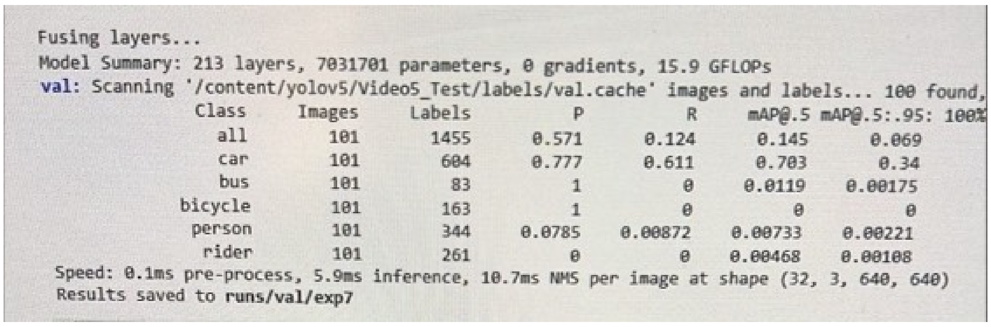
- Training set (100 frames):
  - Video_00: 0-9 frame
  - Video_01: 10-19 frame
  - Video_02: 20-29 frame
  - ...
  - Video_05: (test)
  - Video_06: 50-59 frame
  - ...
  - Video_10: 90-99 frame
- Validation set(20 frames):
  - Video_10: frame 8, 9
  - Video_09: frame 18, 19
  - Video_08: frame 28, 29
  - ...
  - Video_05: (test)
  - Video_04: frame 58, 59
  - ...
  - Video_00: frame 98, 99

(a) Figure showing the distribution of frames on the training and validation set.



```
Model Summary: 213 layers, 7031701 parameters, 0 gradients, 15.9 GFLOPs
             Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 2/2 [00:01<00:00,  1.82it/s]
               all         20        237      0.655      0.268      0.286      0.213
               car         20        117      0.277      0.342      0.204     0.0637
               bus         20          2          1      0.996      0.995       0.92
           bicycle         20         12          1          0      0.206     0.0744
            person         20         84          0          0     0.0192    0.00522
             rider         20         22          1          0    0.00699    0.00113
Results saved to runs/train/exp3
```

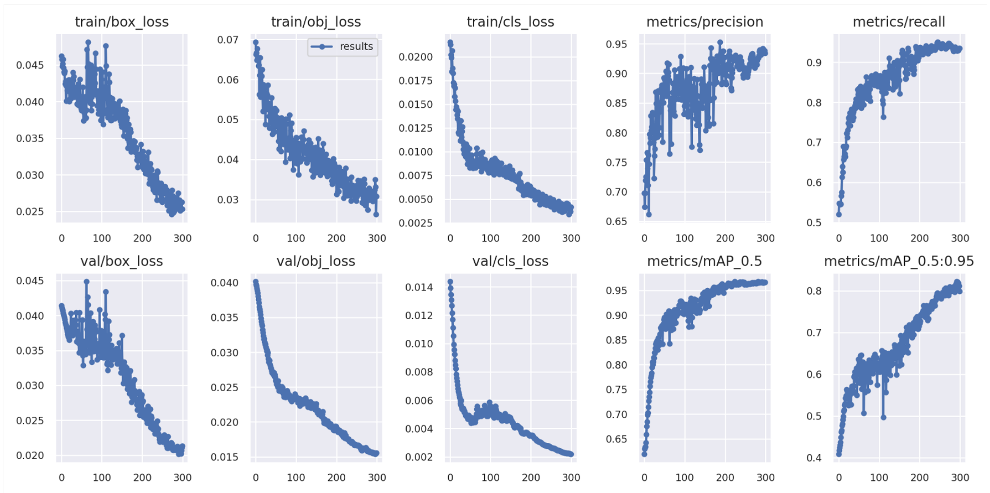
(b) Results of on the validation set



```
Fusing layers...
Model Summary: 213 layers, 7031701 parameters, 0 gradients, 15.9 GFLOPs
val: Scanning '/content/yolov5/Video5_Test/labels/val.cache' images and labels... 100 found,
             Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100%
               all        101       1455      0.571      0.124      0.145      0.069
               car        101        604      0.777      0.611      0.703       0.34
               bus        101         83          1          0     0.0119    0.00175
           bicycle        101        163          1          0          0          0
            person        101        344     0.0785    0.00872    0.00733    0.00221
             rider        101        261          0          0    0.00468    0.00108
  Speed: 0.1ms pre-process, 5.9ms inference, 10.7ms NMS per image at shape (32, 3, 640, 640)
  Results saved to runs/val/exp7
```

(c) Trained Yolov5 with no patching

Figure 5.11: A figure showing the training and validation dataset together with the results.



(a) A figure showing graphed results of Yolo5 on validation set.

(b) A figure showing the predictions on the validation set.

Figure 5.12: A figure showing graphed results of Yolo5 and the predictions labels on the validation set.

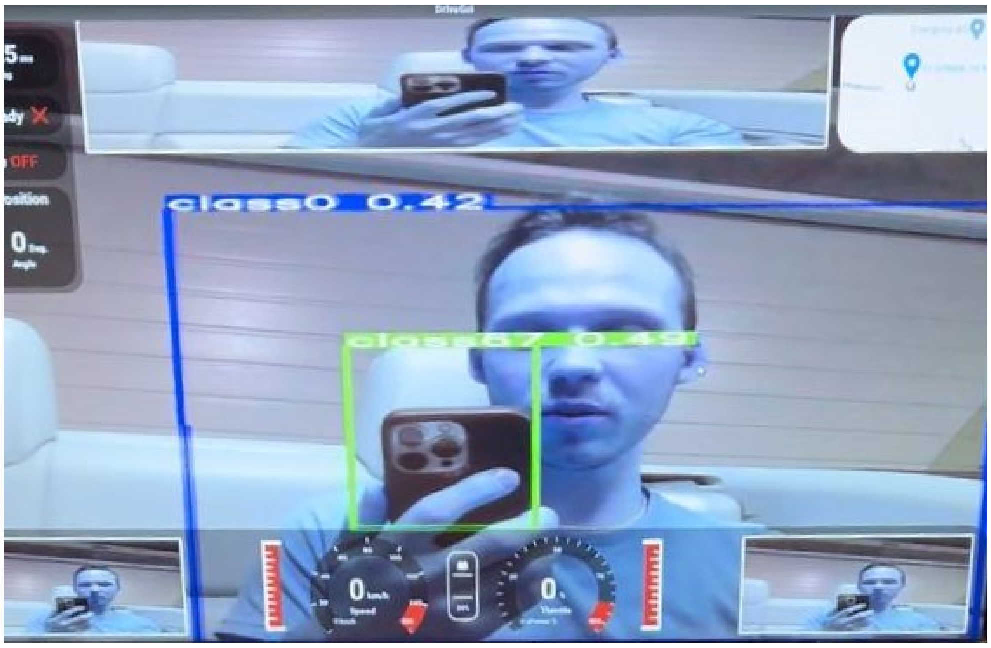


Figure 5.13: Testing of the implemented object detection with Yolo5, that is embedded into the remote control system as seen on picture.

5.8 Expert Feedback

Based on the given results presented in section 5.3 expert feedback was needed to fully grasp the connection challenges with the cellular network. In a mail correspondence with Lars Inge Graabak, the Service Manager of Voice and Internet Computing in Telia, the system was presented shortly, and the results were shown. Then a question was asked on why the 4G and 5G sometimes can have the same low speed and latency, on which L. Graabak responded:

> "*Indeed, there can often be equal speeds of 4G and 5G. That is because some frequency bands are used by both of them due to the same amount of resources for 4G and 5G. Today we run 4G on the following frequencies: 700MHz, 800MHz, 900MHz, 1800MHz, 2100MHz and 2600MHz. The frequency band 700MHz is used by both 4G and 5G and 3600MHz only for 5G. That will change as the use of 5G increases. How much speed we produce depends on the bandwidth. The 700MHz frequency band produces about 80Mbps while the 3600MHz band up to 1,500 Mbps. What also complicates it a bit is that we use Carrier Aggregation, i.e. we connect and use several frequency bands simultaneously. Under favourable conditions, it is possible to get up to 800Mbs on 4G.* "
>
> "*The reason why latency is quite similar between 4G/5G is that the 5G network in the current version uses the earlier 4G network for signalling. The version everyone uses on public 5G networks today is called Non-Stand-Alone (NSA), and in about one year, Stand-Alone 5G will be introduced, and then you will typically get latency down to expected 7-9 ms on 5G. After that, possible edge computing will be developed and send the latency down to 2-3ms in special cases.*"
>
> **See complete mail correspondence in Appendix A**

Chapter 6

Discussion

The developed system showed its effectiveness, performance, and usability and performed above all expectations with the vehicle control realized from a nearby office, with only a 4G+ connection. The operators could ensure that this proof-of-concept did its job and showed the system's potential. All parts were easy to run by the automated scripts described in subsection 4.1.3 and a desktop app with an adequately designed graphical user interface, endowing the operator with a complete overview of the vehicle status and movement.

However, not everything worked as planned, and improvements to the system are needed to be taken into account. The robustness was unfortunately below the threshold for the system to be used in its intended environment; traffic [23]. Furthermore, the use of 4G did not deliver a stable connection, and the latency could sometimes peak at over 100ms, as shown in section 5.3, making it rather difficult for the operator to manoeuvre the vehicle. High bandwidth usage also impacted the system and could overflow the buffers when the available upload speed was lower than 12mbps (5.5). Data packets could therefore be lost, thus lowering the stable video refresh rate of 30 frames per second. In the worst case, this would completely stop the delivery of frames or joystick control messages, resulting in the system collapsing. That affects especially the Drive Kit mounted inside the car and the desktop app on the remote computer, as both are in a stage of receiving messages.

The lack of any mechanism that could stop the car from rolling away or restart the collapsed desktop app automatically made the system at this stage of development risky to use without a safety driver. One of the possible solutions is an integrated operator assistant that was already proposed and planned but unfortunately not finished and remained in the development stage. This assistant could be sure to either safely stop the vehicle or hold the trajectory with object avoidance and lane assistance before the connection was re-established.

Nevertheless, even operator assistant system measurements in Figure 5.6 show that 4G+ and low-end 5G still have high latency, low upload speed and somewhat unstable connection. That is entirely out of the scope of the remote control system and is dependent on the roll-out and development of the 5G transceivers. In section 5.8 the response of the service manager from the ISP Telia explains the possibility of using a new version of 5G in 2023 in Norway, which should make the connection more stable, faster and with lower latency, down to 7-9ms. If this happens, the remote control system would not need to change or downgrade its image quality to lower the bandwidth as it could be a solution for lower latency because of less processing and lower bitrate of the sent video stream.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Self-driving vehicles are slowly but surely starting to become a reality despite the many obstacles still to be overcome. A significant challenge for engineers working on this development is getting the cars to safely and effectively operate in complex and unpredictable human environments. Environments where autonomous cars inevitably will end up in a complex situation where some sensor information is wrong, limited, or something unreasonable happens on the road, leading the vehicle to a possible "deadlock" or an edge case, as discussed in the Background chapter (2). As the main driving force behind this thesis, a remote control system was proposed as a possible solution for dealing with unpredictable edge cases. To prove such a solution, goals, research challenges, and essential features were set in the introduction 1.2 and acted as an important guide throughout the project to develop a proof-of-concept.

Several parts were explored, developed and tested. Together they formed the entire remote control system for the NAP-lab autonomous vehicle. With the use of ROS and a PID controller, the system controls the car from a G29 kit in real-time, while video from 4 cameras mounted on the car is encoded in H.264, transmitted via ROS, decoded and displayed in an informative designed graphical user interface as a desktop app in Kivy. The connection between the operator and the car is realized with a secure P2P VPN tunnel with a fast UDP WireGuard protocol directly through the 5G/4G cellular network. All this is automated with scripts and started by only two CLI commands.

The project results showed that a remote control system can be installed and developed with today's available technology, with both 4G and 5G cellular networks. However, for this system to be used on the road in real traffic situations, 4G is not enough for a stable and robust connection. Furthermore, as of today, 5G is still under development, as stated by the ISP Telia(A).

Therefore the remote control system needs some additional functionality in order to use the older 4G safely. An example is the proposed integrated operator-assistance system that could be vital if some possible collapse of the system occurs. Because of the limited amount of time for this project, the operator assistant was unfortunately not developed and implemented. However, some essential parts were explored and tested, such as the object detection and trajectory projections in the GUI.

To sum up, the complete remote control system did work during the testing and got positive feedback from the operators, but it also needs to be further developed and tested to be safely used in traffic. Per now, it is still considered a proof-of-concept and not a commercial product, but the system did its intended objective in this project; it showed its performance in a real test and proved the potential and feasibility of a remote control system for an autonomous vehicle.

7.2 Future Work

Although the system is fully functional and showed promising results during the tests, some improvements should still be made. As mentioned in the Conclusion section, the system needs a support system that can counteract any lousy network connection or possible wrong manoeuvres by the operator. The development of the operator assistant system was already somewhat started during this project but only at an exploring and test stage. In the future, this system should be developed completely and added to the remote control system. That could help using not only the 5G connection but also realize possible use of the older 4G connection, which was somewhat tricky based on the results in this project.

Another part that needs further development and enhancement is the graphical user interface together with the camera placement on the car. The satisfaction score in Results did never show outstanding user satisfaction, which often was based on connection losses, system collapse and the GUI that did show the front camera view somewhat strange and did not have any reasonable trajectory projections. All these parts need to be fixed and again user-tested on multiple operators.

The last part that also should go through an enhancement is the ROS transfer protocol. Even though the VPN uses UDP, the inside of ROS uses TCP. That could be a part that slows down all the video transmission even more than the latency of the ISP itself. A solution here that could be developed is turning ROS into using UDP rather than TCP, thus making a faster connection. Beyond the mentioned, there are, of course, overall further testing and enhancement of the concept to increase its robustness and potential.

Bibliography

[1] Yandex. 'Yandex self-driving group and grubhub partner for robot delivery on us college campuses.' (), [Online]. Available: `https://yandex.com/company/press_center/press_releases/2021/07-06-2021` (visited on 13/12/2021).

[2] K. Jo, J. Kim, D. Kim, C. Jang and M. Sunwoo, 'Development of autonomous car—part ii: A case study on the implementation of an autonomous driving system based on distributed architecture,' *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5119–5132, 2015. DOI: `10.1109/TIE.2015.2410258`.

[3] M. Claypool and D. Finkel, 'The effects of latency on player performance in cloud-based games,' pp. 1–6, 2014. DOI: `10.1109/NetGames.2014.7008964`.

[4] T. Cerny and M. J. Donahoo, 'Impact of remote user interface design and delivery on energy demand,' pp. 1–4, 2015. DOI: `10.1109/ICISSEC.2015.7371005`.

[5] G. Lewis. 'Object detection for autonomous vehicles.' (), [Online]. Available: `https://web.stanford.edu/class/cs231a/prev_projects_2016/object-detection-autonomous.pdf` (visited on 13/12/2021).

[6] R. J. Krumsvik, *Forskningsdesign og kvalitativ metode - ei innføring*. Fagbokforlaget, 2014, p. 16.

[7] R. J. Krumsvik, *Forskningsdesign og kvalitativ metode - ei innføring*. Fagbokforlaget, 2014, pp. 145–146.

[8] O. Bukve, *Forstå, forklare, forandre. Om design av samfunnsvitskapelege forskingsprosjekt.* Oslo: Universitetsforlaget, 2016, p. 121.

[9] D. Muslihat. 'Agile methodology: An overview.' (), [Online]. Available: `https://medium.com/zenkit/agile-methodology-an-overview-7c7e3b398c3d` (visited on 13/12/2021).

[10] A. Gusev. 'Remote Controll System of Autonomous Vehicle With Operator Assistance over 5G cellular network.' [Online; accessed 6. Jul. 2022]. (Jul. 2022), [Online]. Available: `https://www.youtube.com/watch?v=cehT9yf8Ljs&ab_channel=AlexeyGusev`.

[11] R. B. "Brown A Gonder J, "*An Analysis of Possible Energy Impacts of Automated Vehicles*". "Springer International Publishing", 2014.

[12] T. W. Organization. 'Road traffic injuries.' (), [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries (visited on 12/12/2021).

[13] J. Cusack. 'How driverless cars will change our world.' (), [Online]. Available: https://www.bbc.com/future/article/20211126-how-driverless-cars-will-change-our-world (visited on 12/12/2021).

[14] A. Mills. 'Driving in the snow is a team effort for ai sensors.' (), [Online]. Available: https://www.sciencedaily.com/releases/2021/05/210527172545.htm (visited on 12/12/2021).

[15] W. Knight. 'Snow and ice pose a vexing obstacle for self-driving cars.' (), [Online]. Available: https://www.wired.com/story/snow-ice-pose-vexing-obstacle-self-driving-cars/ (visited on 12/12/2021).

[16] M. J. E. E. G. Coffman and A. Shoshani, 'System deadlocks,' *ACM Comput. Surv.*, vol. 3, no. 2, pp. 67–78, Jun. 1971.

[17] A. J. Hawkins. 'A driverless waymo got stuck in traffic and then tried to run away from its support crew.' (), [Online]. Available: https://www.theverge.com/2021/5/14/22436584/waymo-driverless-stuck-traffic-roadside-assistance-video (visited on 12/12/2021).

[18] R. Stern. 'Angry residents, abrupt stops: Waymo vehicles are still causing problems in arizona.' (), [Online]. Available: https://www.phoenixnewtimes.com/news/waymo-arizona-abrupt-stops-angry-residents-are-still-a-problem-11541896 (visited on 12/12/2021).

[19] S. Jun, Y. Kang, J. Kim and C. Kim, 'Ultra-low-latency services in 5G systems: A perspective from 3GPP standards,' *ETRI Journal*, vol. 42, no. 5, pp. 721–733, Oct. 2020, ISSN: 1225-6463. DOI: 10.4218/etrij.2020-0200.

[20] K. Sasaki, S. Makido and A. Nakao, 'Vehicle control system for cooperative driving coordinated multi -layered edge servers,' in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, 2018, pp. 1–7. DOI: 10.1109/CloudNet.2018.8549396.

[21] M. S. "M. Fukui Y. Sugiyama and D. Wolf", "*Cooperative Driving: Taking Telematics to the Next Level*". "Springer International Publishing", 2003.

[22] X. Ge, 'Ultra-reliable low-latency communications in autonomous vehicular networks,' *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5005–5016, 2019. DOI: 10.1109/TVT.2019.2903793.

[23] T. Blomqvist, 'Video latency in a vehicle's remote operating system,' *Metropolia University of Applied Sciences*, p. 38, 2018.

[24] IDC. 'Self-driving vehicles will emerge, but only gradually, idc says.' (), [Online]. Available: `https://www.fierceelectronics.com/electronics/self-driving-vehicles-will-emerge-but-only-gradually-idc-says` (visited on 12/12/2021).

[25] Globaldata. 'Sobering attitudes towards autonomous vehicles see long-term forecasts cut by almost half.' (), [Online]. Available: `https://www.globaldata.com/sobering-attitudes-towards-autonomous-vehicles-see-long-term-forecasts-cut-almost-half/` (visited on 27/04/2021).

[26] C. Murray. 'Automakers are rethinking the timetable for fully autonomous cars.' (), [Online]. Available: `https://www.plasticstoday.com/electronics-test/automakers-are-rethinking-timetable-fully-autonomous-cars` (visited on 27/04/2021).

[27] W. Law, 'An Introduction to Autonomous Vehicles - Towards Data Science,' *Medium*, Dec. 2021, ISSN: 9161-8140. [Online]. Available: `https://towardsdatascience.com/an-introduction-to-autonomous-vehicles-91d61ff81a40`.

[28] *SAE J3016 automated-driving graphic*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic`.

[29] *Forskning.no*, [Online; accessed 6. Jul. 2022], Mar. 2021. [Online]. Available: `https://forskning.no/internett-mobiltelefon/hva-er-egentlig-5g/1813874`.

[30] *Read @Kearney: 5G: a key requirement for autonomous driving—really?* [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://www.kearney.com/communications-media-technology/article/-/insights/5g-a-key-requirement-for-autonomous-driving-really-`.

[31] C. Moozakis, 'The pros and cons of 5G networks,' *SearchNetworking*, Apr. 2022. [Online]. Available: `https://www.techtarget.com/searchnetworking/feature/The-pros-and-cons-of-5G-networks`.

[32] M. W. Vick Yu. '3 types of autonomous vehicle sensors in self-driving cars.' (), [Online]. Available: `https://www.itransition.com/blog/autonomous-vehicle-sensors` (visited on 14/12/2021).

[33] *OS2 Long-range lidar sensor for autonomous vehicles, trucking, and drones*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://ouster.com/products/scanning-lidar/os2-sensor`.

[34] The Superyacht Group, 'British-Norwegian technology partnership provides superyachts with secure super-speed internet,' *Superyacht News*, Sep. 2021. [Online]. Available: `https://www.superyachtnews.com/technology/british-norwegian-technology-partnership-provides-superyachts-with-secure-super-speed-internet`.

[35] *Arcus*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://www.celerway.com/products/arcus`.

[36] R. Ricky, 'Why Do Routers Have Multiple Antennas? - NetWork From Home,' *NetWork From Home*, Dec. 2021. [Online]. Available: https://network-from-home.com/home-network/why-do-routers-have-multiple-antennas.

[37] *NVIDIA DRIVE Software for Autonomous Machines*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-platform/software.

[38] *NVIDIA DRIVE SDK*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: https://developer.nvidia.com/drive/drive-sdk.

[39] *DriveWorks SDK Reference: Getting Started*, [Online; accessed 6. Jul. 2022], Jun. 2020. [Online]. Available: https://docs.nvidia.com/drive/archive/driveworks-3.0/dwx_devguide_getting_started.html.

[40] *ROS/Introduction - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/ROS/Introduction.

[41] S. Kavanagh. '5g vs 4g: No contest.' (), [Online]. Available: http://wiki.ros.org/catkin/conceptual_overview (visited on 15/12/2021).

[42] *Packages - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/Packages.

[43] *Nodes - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/Nodes.

[44] *Master - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/Master.

[45] *Messages - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/Messages.

[46] *Topics - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/Topics.

[47] *roslaunch - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/roslaunch.

[48] *sensor_msgs - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/sensor_msgs.

[49] *image_transport - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/image_transport.

[50] *joy - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/joy.

[51] *usb_cam - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: http://wiki.ros.org/usb_cam.

[52] Ut-Adl, *nvidia_gmsl_driver_ros*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: https://github.com/UT-ADL/nvidia_gmsl_driver_ros.

[53] Ut-Adl, *h264_image_transport*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://github.com/UT-ADL/h264_image_transport`.

[54] *image_view - ROS Wiki*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `http://wiki.ros.org/image_view`.

[55] *wireguard-monolithic-historical - Historical monolithic WireGuard repository, split into wireguard-tools, wireguard-linux, and wireguard-linux-compat.* [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://git.zx2c4.com/wireguard-monolithic-historical/tag/?h=0.0.20161209`.

[56] *WireGuard VPN review: A new type of VPN offers serious advantages*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen`.

[57] [Online; accessed 6. Jul. 2022], Jun. 2022. [Online]. Available: `https://hal.inria.fr/hal-02100345v3/document`.

[58] J. A. Donenfeld, *Known Limitations - WireGuard*, [Online; accessed 6. Jul. 2022], May 2022. [Online]. Available: `https://www.wireguard.com/known-limitations`.

[59] J. A. Donenfeld, *WireGuard: fast, modern, secure VPN tunnel*, [Online; accessed 6. Jul. 2022], May 2022. [Online]. Available: `https://www.wireguard.com`.

[60] D. Kondo, Y. Hirota, A. Fujimoto, H. Tode and K. Murakami, 'P2p live streaming system for multi-view video with fast switching,' in *2014 16th International Telecommunications Network Strategy and Planning Symposium (Networks)*, 2014, pp. 1–7. DOI: `10.1109/NETWKS.2014.6959253`.

[61] *How To Block Incoming Traffic Firewall?* [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://www.nstec.com/how-to-block-incoming-traffic-firewall`.

[62] T. Bocek, E. Hunt, D. Hausheer and B. Stiller, 'Fast similarity search in peer-to-peer networks,' in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, 2008, pp. 240–247. DOI: `10.1109/NOMS.2008.4575140`.

[63] A. Summers. 'Self-driving cars to bring a new form of hacking.' (), [Online]. Available: `https://www.le-vpn.com/self-driving-cars-bring-new-form-hacking/` (visited on 15/12/2021).

[64] H. Long, 'WireGuard vs OpenVPN in 2022: 7 Big Differences,' *RestorePrivacy*, Apr. 2022. [Online]. Available: `https://restoreprivacy.com/vpn/wireguard-vs-openvpn`.

[65] B. Smith. 'Compare tinc vs openvpn- which one?' (), [Online]. Available: `https://internet-access-guide.com/tinc-vs-openvpn/` (visited on 15/12/2021).

[66] *What Is TCP Meltdown? | OpenVPN*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://openvpn.net/faq/what-is-tcp-meltdown`.

[67] I. Coonjah, P. Catherine and K. Soyjaudah, 'Experimental performance comparison between tcp vs udp tunnel using openvpn,' pp. 1–5, Dec. 2015. DOI: `10.1109/CCCS.2015.7374133`.

[68] PolySync, *oscc*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://github.com/PolySync/oscc/wiki/Hardware-Gateway`.

[69] *jstest-gtk - A joystick testing and configuration tool for Linux*, [Online; accessed 6. Jul. 2022], Feb. 2021. [Online]. Available: `https://jstest-gtk.gitlab.io`.

[70] S. K. "Chinmay Samak Tanmay Samak, "*CONTROL STRATEGIES FOR AUTONOMOUS VEHICLES*". "Carnell University", 2021.

[71] x-engineer. org, *Open loop vs. closed loop control systems (with Xcos simulations) – x-engineer.org*, [Online; accessed 6. Jul. 2022], Jul. 2022. [Online]. Available: `https://x-engineer.org/open-loop-vs-closed-loop-control-systems`.

[72] C. V. Samak, T. V. Samak and S. Kandhasamy, 'Control Strategies for Autonomous Vehicles,' *arXiv*, Nov. 2020. DOI: `10.48550/arXiv.2011.08729`. eprint: `2011.08729`.

[73] *PID Controller Manual tuning - DCS - Engineers Community*, [Online; accessed 6. Jul. 2022], May 2018. [Online]. Available: `https://engineerscommunity.com/t/pid-controller-manual-tuning/4043`.

[74] H. Yamauchi and A. Luštica, 'Audio and video over ip technology,' in *2015 57th International Symposium ELMAR (ELMAR)*, 2015, pp. 125–128. DOI: `10.1109/ELMAR.2015.7334512`.

[75] *DriveWorks SDK Reference: Camera*, [Online; accessed 6. Jul. 2022], Jun. 2020. [Online]. Available: `https://docs.nvidia.com/drive/archive/driveworks-3.0/camera_mainsection.html`.

[76] Biamp, 'Video Basics,' *Biamp Cornerstone*, Jun. 2019. [Online]. Available: `https://support.biamp.com/General/Video/Video_Basics`.

[77] Z. Z. Yecheng Lyu Xinming Huang. 'Learning to segment 3d point clouds in 2d image space.' (), [Online]. Available: `https://arxiv.org/abs/2003.05593` (visited on 14/12/2021).

[78] G. Boesch. 'Computer vision annotation tool (cvat) – 2021 overview.' (), [Online]. Available: `https://viso.ai/computer-vision/cvat-computer-vision-annotation-tool/` (visited on 14/12/2021).

[79] 'What is YOLO Algorithm? | Baeldung on Computer Science.' [Online; accessed 4. Jul. 2022]. (Jul. 2022), [Online]. Available: `https://www.baeldung.com/cs/yolo-algorithm`.

[80] deep-plant-phenomics. ‘Automatic image patching.’ (), [Online]. Available: `https://deep-plant-phenomics.readthedocs.io/en/latest/Automatic-Image-Patching/` (visited on 14/12/2021).

[81] A. Combelles, C. Ebert and P. Lucena, ‘Design thinking,’ *IEEE Software*, vol. 37, no. 2, pp. 21–24, 2020. DOI: `10.1109/MS.2019.2959328`.

[82] *Where do users look first? | GazeHawk Blog*, [Online; accessed 6. Jul. 2022], Sep. 2021. [Online]. Available: `https://gazehawk.com/blog/where-do-users-look-first`.

[83] K. Gordon, ‘5 Principles of Visual Design in UX,’ *Nielsen Norman Group*, Mar. 2020, Accessed: 2020-11-13. [Online]. Available: `https://www.nngroup.com/articles/principles-visual-design/`.

[84] H. S. J. Preece Y. Rogers, *Interaction Design: Beyond Human-Computer Interaction*, 5. edition. John Wiley & Sons Inc, 2019.

Appendix A

Additional Material

Fra: Graabak, Lars Inge
Sendt: onsdag 29. juni 2022 kl. 19:54
Til: Alexey Gusev; Frydenlund, Espen
Emne: Re: Masterprosjekt - Spørsmål til Telia om 5G

Absolutt! Helt greit.

Få Outlook for Android

From: Alexey Gusev <alexeygu@stud.ntnu.no>
Sent: Wednesday, June 29, 2022 7:28:44 PM
To: Graabak, Lars Inge <lars-inge.graabak@telia.no>; Frydenlund, Espen <espen.frydenlund@telia.no>
Subject: Re: Masterprosjekt - Spørsmål til Telia om 5G

Hei, Tusen takk for så rask respons og svar på spørsmålene! Er det greit at jeg legger ved denne mailen som referanse i masteren? :)

Mvh Alexey Gusev

Fra: Graabak, Lars Inge <lars-inge.graabak@telia.no>
Sendt: Monday, June 27, 2022 3:52:04 PM
Til: Frydenlund, Espen <espen.frydenlund@telia.no>; Alexey Gusev <alexeygu@stud.ntnu.no>
Emne: RE: Masterprosjekt - Spørsmål til Telia om 5G

Hei
Har kommentert med rød skrift under spørsmålene. Har prøvd å lage en kort forklaring på hver enkelt

Lars Inge Graabak
Service manager Voice | NSD
+47 92695830
lig&telia.no

Telia

Skonnertvegen 7, 7053 Trondheim
www.telia.no

Telia Norge AS 981 929 055

From: Frydenlund, Espen <espen.frydenlund@telia.no>
Sent: mandag 27. juni 2022 13:23
To: Graabak, Lars Inge <lars-inge.graabak@telia.no>
Subject: FW: Masterprosjekt - Spørsmål til Telia om 5G

Hei Lars!

Alexey er en NTNU-student som skriver en masteroppgave rundt autonome kjøretøy (Remote Control Systems) og spør om vi kan besvare spørsmålene nedenfor.
Ettersom du sitter med bedre kompetanse og informasjon enn det jeg gjør – kunne du hjulpet til å med å besvare de? 😊

Mvh

Espen Frydenlund
Senior Sales Executive | Telia Enterprise
tlf: +47 958 19 909
epost: espen.frydenlund@telia.no

From: Alexey Gusev <alexeygu@stud.ntnu.no>
Sent: mandag 27. juni 2022 12:32
To: Frydenlund, Espen <espen.frydenlund@telia.no>
Subject: Masterprosjekt - Spørsmål til Telia om 5G

Hei Espen,

Jeg har da som tidligere nevnt et par spørsmål som kan anses som noe tekniske til dere i Telia, angående 5G nettet. Jeg tror mest sannsynlig at du kanskje vet svaret på de fleste, og jeg selv har en anelse på dem, men det er fint å få et korrekt svar direkte fra dere. 😜

Jeg leverer masteren den 30. juni, på torsdag. Så om dere får til å svare før den tid så legger jeg ved svaret deres på mailen til som en referanse i masteroppgaven. Det trengs ikke veldig lange svar, men bare noe som gir en forståelse. 😊

1. *Etter å ha kjørt rundt om i byen Trondheim for å teste 5G og 4G nettet med vanlig mobil og applikasjonen OpenSignal, så viser det seg at flere steder i byen det er lik hastighet og latency mellom 4G og 5G. Hvorfor er det slik? (Bilder lagt ved fra 3 ulike steder i Trondheim)*

Det er riktig at det mange ganger kan være lik hastighet på 4 og 5G fordi de i enkelte frekvens bånd er delt. DVS det er like mye resurser tilgjengelig for 5G som 4G. I dag kjører vi 4G på følgende frekvenser 700 800 900 1800 2100 og 2600. Mens vi i dag bruker 700 (delt med 4G) og 3600 for 5G. Dette vil endre seg etter hvert som bruken av 5G øker. Hvor mye hastighet vi klatrer å produsere er avhengig av båndbredden. På 700 er den på 10 MHz mens på 3600 på 100MHz. 700 frekvensen klarer å produsere ca 80Mbs mens 3600 opp til 1 500 Mbs. Det som i tillegg kompliserer det litt er at vi bruker noe som heter Carrier Aggregasjon, dvs vi kobler sammen og bruker flere frekvensbånd samtidig. Under gunstige forhold kan du da på opp mot 800Mbs på 4G. Det samme er vi i gang med på UL, men ennå i litt begrenset omfang.
I deler av Oslo (og snart Trondheim) får du over 180 Mbs på UL
Årsaken til at latency er ganske lik er at 4G nettet i dagens versjon av 5G brukes til signalering. Den versjonen alle bruker på offentlige 5G nett i dag kalles Non Stand Alone (NSA) og om et års tid kommer Stand Alone 5G og da vil du typisk komme ned i 7-9 ms, inntil Edge computing kommer og sender den ned mot 2-3 i svært spesielle tilfeller.

2. *Det er også slik at på alle testene så har 5G men spesielt 4G veldig lav «Upload» hastighet i forhold til «Download». Det er først og fremst denne som kan sette begrensninger for mitt system når det gjelder bruk av 4G ettersom systemet, ved bruk av høykomprimert video, kan sende mellom 16mbps – 8mbps. Ved høyere kvalitet og flere kameraer så vil dette selvsagt*

øke. På noen av testene ser vi en opplastingshastighet på 12mbps som er lavere enn det som sendes. Kort sagt hvorfor er det slik og vil denne i fremtiden øke på 5G nettet?

Årsaken er en skjevfordeling av UL og DL. Den aller meste av trafikken er DL dvs at kundene surfer eller ser på video, derfor bruker vi mange flere blokker til DL. Nå har vi begynt å aktivere carrier Aggregasjon på Up Link også så hastigheten vil øke på sikt etter hvert som vi bruker flere ressurser til 5G

3. *Jeg forstå det fra nevnelser av Telenor og artikkel på forskning.no at dagens 5G i Norge bruker underliggende 4G teknologi i form av at all prosessering for trafikk foregår i servere i Oslo. Det fører da til latency i Trondheim kan være betydelig høyere en 5G har mulighet til. Enkelt forklart, hvordan foregår denne prosessen og hvorfor er det egentlig slik?*

Årsaken er at vi bruker 4G til signalering, og det er mulig Telenor bare har kjerne nett i Oslo. Vi har i dag 3 lokasjoner hvorav en ligger i Trondheim. Hoveddelen av trafikken i Trondheim rutes direkte hit, men det er lastdeling mellom nodene slik at i perioder kan det mye rutes via Oslo

4. *Hvor langt har man kommet med utvikling av 5G nettet i Norge i dag?*

Sammenlignet med mange andre land kommet ganske langt. Radiodelen av nettet skal være rullet ut over hele landet i løpet av neste år for Telia og Telenor året etter . Når det gjelder kjerne nettet skal vi ha SA klar i slutten av året og tilby nye tjeneste som skivedeling på nyåret

5. *Hvor mange år framover antar vi at vi har et stabilt 5G nett i for eksempel Oslo eller Trondheim uten at latency eller hastighet faller ned til 4G kvalitet?*

Du vil aldri få et nett uten latency, men med stabil 7-9 ms i løpet av neste år. Og når vi kommer litt fram i tiden ned mot 2-3ms i spesielle situasjoner. I forbindelse med utrullingen av 5G oppgraderer vi 4G nettet slik at kvaliteten på 4G nettet øker betraktelig. Det er mange ting som påvirker latency. Avstand til basestasjon, hvordan transmisjonen er mellom basestasjonen og kjernenettet. Trafikken ut på internett og hvordan leverandøren av måletjenesten er satt opp. Ser at når jeg kjører speed test, at resultatet blir forskjellig om jeg tester mot NTNU eller mot en server som står på Blindern. Jeg sitter i Trondheim

6. *Hvordan ser Telia på den type teknologi som utvikles som del av dette masterprosjektet. Kunne et slikt fjernstyrings system (Remote Control System) hypotetisk sett blitt brukt for fremtidens autonome biler bli brukt i Norge?*

Det kan så absolutt tenkes at denne teknologien kan brukes på autonome biler i fremtiden. Det pågår mange store prosjekter på dette ute i verden og det diskuteres tungt hvilken protokoll som skal brukes. I dag er det V2V og V2I som er mest aktuell. Her har jeg sett noen pappers fra NTNU også med betraktninger.

På forhånd takk

Mvh Alexey Gusev
Masterstudent og forskningsassistent ved NTNU