Remote Control for Rail Vehicles

Simon Jürgensen

Bachelor's thesis
March 2025

Prof. Dr. Reinhard von Hanxleden
Real-Time and Embedded Systems Group
Department of Computer Science
Kiel University

Advised by
Dr.-Ing. Alexander Schulz-Rosengarten

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weiterhin erkläre ich, dass die digitale Fassung dieser Arbeit, die dem Prüfungsamt per E-Mail zugegangen ist, der vorliegenden schriftlichen Fassung entspricht.

Kiel,

Abstract

Digitizing and better utilizing the railway network in Germany is a big focus for the railway industry in general. With Automatic Train Operation (ATO) the railway traffic can be done more efficient and a high Grade of Automation (GoA) is key for achieving this goal. With the stages of the GoA describing a transition from human controlled trains to fully autonomous trains, utilizing autonomous rail vehicles, is a very important focus for the future of railway traffic. To ensure this transition is safe at all stages, it is essential to have a means of intervening from a distance in case of an emergency. A remote control system is a way to achieve this safety during the development of an autonomous vehicle and during intermediate stages of automation. Additionally, a remote control system can be a fallback solution for emergencies in later stages of automation.

This thesis develops a remote control system for rail vehicles, using the REAKTOR as an example use case. The REAKTOR project is part of the REAKT initiative[1], which aims to reactivate abandoned tracks in rural regions with innovative and smart vehicles, achieving a better utilization of the existing railway network in the process. As a part of the initiative an autonomous railway vehicle, the REAKTOR, is being developed and is the centrepiece of the REAKTOR project. The remote control system gives the user the means to take control of the vehicle and decide on the speed and direction. This is supported by a livestreamed camera from the vehicle to give the user access to real time information from the track, which is essential for the user to utilize the system in the intended way. Another part of this thesis is a concept for using the remote control system on the full test track for the REAKT initiative between Malente and Lütjenburg and an assessment of the network needed for the system to be used is presented as well.

The concept for the remote control system developed in this thesis is scalable and applicable for different use cases in the context of autonomous railway traffic, enhancing their safety and flexibility.

Acknowledgements

Firstly, I would like to thank Prof. Dr. Reinhard von Hanxleden for giving me the opportunity to write this thesis and work on the REAKTOR project.

Furthermore, I want to thank my advisor, Dr.-Ing. Alexander Schulz-Rosengarten, who helped guide me a lot throughout the work on the project and writing this thesis. Moreover, I would like to thank Birkan Denizer for giving me insights and figures for the networking part in this thesis, which was a great help for me.

[1] `https://reakt.sh/`

Finally, I am very thankful for the working environment created by the Real-Time and Embedded Systems Group and especially for the company of Kevin Ebsen, Lorenz Tiedemann, Rasmus Janssen, and Nig Rambow during the joint work on the REAKTOR project.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
ATO	Automatic Train Operation
ARTE	Autonomous Regional Train Evolution
CAU	Christian-Albrechts-Universität zu Kiel/ Kiel University
CSS	Cascading Style Sheet
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DSD	Digitale Schiene Deutschland
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
EU	European Union
GoA	Grade of Automation
GPIO	General Purpose Input/Output
GPS	Global Positioning System

HTML	Hypertext Markup Language
HTTP	Hyptertext Transfer Protocol
JSON	JavaScript Object Notation
km/h	kilometers per hour
LTE	Long Term Evolution
NGT	Next Generation Train
OS	Operating System
RTT	Round Trip Time
SQL	Structured Query Language
STTT	Single-Track Transfer Traffic
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Ressource Locator
WSGI	Python Web Server Gateway Interface

Chapter 1

Introduction

In terms of public transport, the railway is one of the most important ways of travelling for a big part of the population in Germany. The load on the railway network is increasing through a growing number of passengers and freight transport. In contrast to that, the railway network itself did not grow in size adequately[1]. According to the Allianz pro Schiene[2], the railway network even decreased in size by 11.8 percent since 1995, with the existing railway network being overloaded. Thus, the existing railway network must be utilized in a more efficient way, as well as increase in size, in order to meet the requirements in the future. A critical point for the efficiency of railway traffic is a shortage of skilled workers, especially drivers.

Digitizing the existing railway network is one approach to overcome these problems. With ATO over European Train Control System (ETCS), a GoA 4 for the railway network in Germany leads to minimal staff required for rail traffic. Vehicles with a GoA 4 do not require a driver or staff on board the rail vehicle, but can be monitored from a distance. This solves the shortage of drivers, but creates a need for humans to take control of the vehicles in case of an emergency.

For autonomous vehicles, it is important to mention that autonomy does not mean that there is no more human element involved. It rather means that it is difficult or not even possible for a human to take over driving control and solve problems for the vehicle. Errors of autonomous systems can still be considered human errors by extension, because humans designed and constructed those systems and any flaws in the system could be argued to be routed in a human error of logic or judgement [Ahv16]. The key difference is that the ability to handle an error is no longer an option for the human in autonomous systems. Achieving a gradual transition from human controlled vehicles to autonomous vehicles is a natural way of minimizing errors in the system step by step.

This means for the design of an autonomous vehicle that a remote control system is the next logical step on the way to reaching GoA 4. Myhrvold states that a remote control system can be a step towards full autonomy [Myh23] and is also commonly used for safety while testing the behaviour of autonomous vehicles. With a goal of a GoA 4 it is crucial to have a way for a human to intervene and support the system in the intermediate levels of automation [CM23; MRB25]. Additionally, a remote control system can be a supplement even to a fully autonomous driving system [Myh23]. A similar idea of a remote control system as a fallback system and an additional safety layer for autonomous vehicles is also presented by Cogan and Milius [CM23]. As such, the goal of the system is to support autonomous systems while

[1] https://www.bundesregierung.de/breg-de/aktuelles/faq-schienenverkehr-2170596

[2] https://www.allianz-pro-schiene.de/themen/infrastruktur/schienennetz/

reaching full autonomy and ensuring safety in the process.

Another way to relieve the pressure on the current railway network is to expand it. An efficient way of doing so is to integrate existing tracks that were abandoned previously, back into the current railway network. Such tracks exist mostly in rural regions, and this is exactly where the REAKT initiative starts. These tracks were usually abandoned for economic reasons and to reactivate them, a more economical solution is required. An autonomous vehicle is a solution to this problem, as significantly less staff is required for maintaining it in relation to a human driven train. As autonomous vehicles need a way for a human to take control in case of unforeseen circumstances, the REAKTOR is a great use case for the remote control system developed in this thesis.

1.1 Preleminaries

Firstly, some important basics and systems for autonomous rail traffic need to be discussed, in order to understand the goals and demands for the digitization of the railway network.

1.1.1 ETCS

The ETCS[3] is a part of the European Rail Traffic Management System (ERTMS) and is carried out by the European Union (EU). The goal for ETCS is to establish a standard for railway traffic in the EU. ETCS differentiates between different levels of this standardization that differ in the infrastructure on the specific track.

1.1.2 ATO

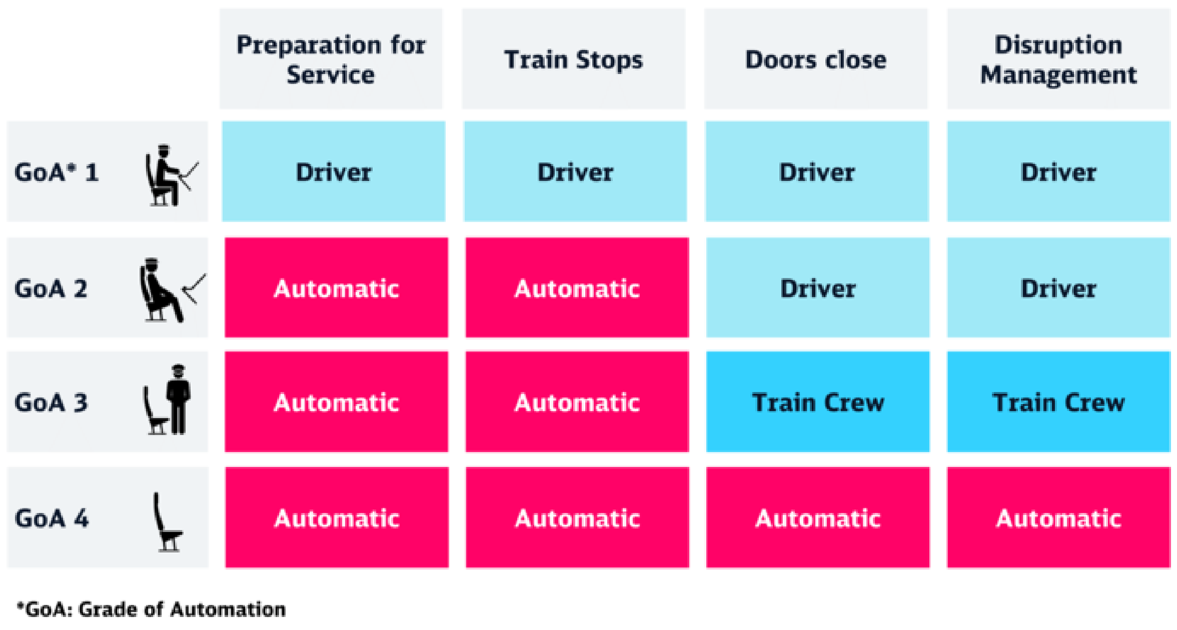
ATO[4] is a technology that centres around automatic train traffic in general. With the integration of ATO over ETCS, the infrastructure according to the ETCS level can be used for the automation of the rail vehicles. At ETCS Level 2, a constant connection via radio from the ETCS headquarters to the vehicle be used to enable vehicle automation by receiving orders from the headquarters. To what degree this is possible is described by the GoA.

1.1.3 GoA

The GoA has 4 levels, which describe a gradual transition from human driven trains to automatic trains, as can be seen in Figure 1.1. At GoA 1, the Driver is responsible for the preparation for service, train stops, doors closing, and disruption management on the track. At GoA 2, preparation for service and the train stops are done automatically, while the driver still has responsibility for the doors closing and general disruption management. At GoA 3, the doors closing and disruption management can be handled by the train crew. At this

[3]https://digitale-schiene-deutschland.de/de/technologien/ETCS

[4]https://digitale-schiene-deutschland.de/en/Automatic-Train-Operation



	Preparation for Service	Train Stops	Doors close	Disruption Management
GoA* 1	Driver	Driver	Driver	Driver
GoA 2	Automatic	Automatic	Driver	Driver
GoA 3	Automatic	Automatic	Train Crew	Train Crew
GoA 4	Automatic	Automatic	Automatic	Automatic

*GoA: Grade of Automation

Figure 1.1. GoA. Source: `https://digitale-schiene-deutschland.de/en/Automatic-Train-Operation`

stage, the train is driverless, but not unattended. Finally, at GoA 4, all these features are done automatically, leaving the train unattended and classifying it as a fully automatic vehicle.

1.2 The REAKT Initiative

The goal of the REAKT initiative is to reactivate abandoned tracks and to better connect rural regions into the railway network. The initiative aims to achieve innovative solutions for railway traffic that are climate friendly, sustainable and attractive for potential passengers to use. Currently, new concepts and ideas for railway traffic are being tested on the abandoned single track between Malente and Lütjenburg. This 17 km long track is currently driven on by tourists on non-motorized trolleys but is also used as a real-live laboratory for multiple projects and types of vehicles by the initiative. One of these projects is the REAKTOR project.

1.3 The REAKTOR Project

The REAKTOR project's aim is to invent a new form of utilizing single tracks for railway traffic with the greater goal to have efficient, climate friendly and passenger friendly railway traffic. Thus connecting rural areas better to the railway network and also making the railway itself more attractive for passengers to use. The current goal for the project is to conduct a feasibility study with a prototype for the REAKTOR, collecting data and checking for safety.

The idea is to have multiple autonomous vehicles run on the same track and perform Single-Track Transfer Traffic (STTT) to allow passengers to transfer directly from one vehicle to another and continue their journey in the same direction while the vehicles themselves drive in the opposite direction after the STTT is performed. Additionally, the passengers should be able to not only enter the REAKTOR on a train station but also on multiple suitable points of interest on the track. For this matter, the user should be able to call the nearest vehicle to the nearest possible point of interest to board. This is how on-demand traffic is also part of the REAKTOR project. With multiple vehicles, the STTT and on-demand traffic being pillars of the project, it is clear that efficient scheduling and also an intelligent REAKTOR itself are essential for the REAKTOR project.

Multiple theses have been written concerning parts of the project. An autonomous controller is being implemented to achieve the goal concerning fully automatic vehicles on the track. To support the controller, another thesis is being written concerning Artificial Intelligence (AI)-based obstacle detection. A thesis that implements an on-demand app creates a way for passengers to utilize the traffic in the intended way in the future of the project. Moreover, a thesis concerning a digital twin for the REAKTOR is being written, with the digital twin intended to simulate behaviour for the vehicles on the real track and therefore also to be used as a testing environment for other parts of the project in the future.

The REAKTOR should run fully autonomously in the future with the GoA 4. As a means for safety during development stages, a remote control system is being implemented for the REAKTOR. Furthermore, the remote control is designed to be used as a backup in the future as a way for humans to interact from a distance with fully autonomous vehicles on the track. This thesis implements a scalable remote control system with the specific use case being the REAKTOR, interfacing with the autonomous controller.

1.4 Problem Statement

A remote control system is important for autonomous vehicles on intermediate stages of autonomy and as a fallback system in case of an emergency. A remote control system provides a way for a human to intervene from afar and deal with unforeseen circumstances.

The remote control system is designed to give the user the means to control the speed, brakes and direction of the rail vehicle on the full track while providing real time information. Another requirement for the system is scalability to other types of rail vehicles and trains. Additionally, it is essential for the system to interact with other parts of the system of the autonomous rail vehicle in a clean and logical way. Lastly, the remote control system needs to provide safety for the vehicle and the full autonomous system while taking control, releasing control, during remote control and in case of a network error.

To achieve these goals, this thesis implements a remote control system that gives the user remote access to speed, brakes and direction of a rail vehicle and is supported by a livestream from a camera. The system is designed to be scalable for real trains and multiple vehicles, interfacing with the autonomous controller of the vehicle. An assessment for the network, needed for the system is also part of this thesis. These requirements are satisfied in the concepts for the system in general and in the implementation for the use case REAKTOR especially.

1.5 Outline

In Chapter 2 the different technologies used in this thesis are being introduced and justified. Following that, Chapter 3 reviews a number of different projects regarding autonomous vehicles and remote control. Chapter 4 explains the concepts for the thesis and Chapter 5 how the implementation for the remote control system is done in detail. The following chapter, Chapter 6, performs an assessment for the network needed for the system and an analysis for the network on the track between Malente and Lütjenburg on the base of the work from Birkan Denizer, from CAU. Finally, Chapter 7 summarizes the work of this thesis and gives insight into possible additions and in future work.

Chapter 2

Technologies

In this chapter, the languages, frameworks, and tools that are used in the remote control system are being introduced. These technologies have been picked because of the flexibility in development and usability that they offer, as well as their scalability options for further uses of the system. In the following sections, the advantages of the different technologies and the reasons why they were chosen are being discussed.

2.1 Raspberry Pi

There are multiple different models of single-board computers, named Raspberry Pi[1] that can perform various tasks. The variety of uses ranges from industrial use over private projects of single developers to educational uses. The possible uses for a Raspberry Pi can be extended with different modules, such as a camera module. The single-board computer can also interact with external hardware via General Purpose Input/Output (GPIO) pins, which are easily accessible for developers. Additionally, the default operating system running on a Raspberry Pi, Raspberry Pi OS, is based on Linux and provides a very flexible environment for developing. More on the range on what a Raspberry Pi can and should be used can be read in a review article by Mathe et al. [MKV+24].

The current use case for the remote control system developed in this thesis, the demonstrators for the REAKTOR utilize different models of Raspberry Pi for specifically for controlling them and utilizing the livestream from the camera, as well as possible other sensors in the future. Thus, the remote control system interacts with the autonomous controller and the livestream on a Raspberry Pi in this use case.

2.2 Languages

The following sections discuss the different programming languages that this thesis uses and why they were chosen.

[1]https://www.raspberrypi.com/

2.2.1 Python

Python[2] is a programming language that supports many different programming paradigms, works on different platforms and is relatively easy to learn. This means that interfaces between different parts of the project are made easier by using Python.

Furthermore, Python is also supported by Raspberry Pi OS, which makes it easy to test a version of a program. The python code for this thesis is written with the version Python 3.13.0.

2.2.2 HTML

Hypertext Markup Language (HTML) is used for defining static parts of websites, such as text and pictures. A web client in the browser of a user can interact with a server via Hyptertext Transfer Protocol (HTTP) requests. Dynamic content can also be included, for example via JavaScript.

2.2.3 CSS

Cascading Style Sheet (CSS) is the default language to format the content of HTML documents. All HTML elements can be customized visually and the overall layout of the website can be defined with HTML and CSS combined.

2.2.4 JavaScript

JavaScript[3] is a flexible, lightweight programming language that supports multiple programming paradigms. Most importantly, it is used for dynamic content on websites. External JavaScript files can be included as well as directly written into the HTML files.

2.3 Flask

Flask[4] is a framework for programming web applications, using the Python Web Server Gateway Interface (WSGI). Flask makes simple applications easy to start with, but can also scale in complexity accordingly if needed. This means that prototypes and early versions for applications can be written relatively fast and tested easily.

In Flask, different HTML templates can be created for defining different pages for the web application and with the routing decorator they can be accessed at a specific Uniform Ressource Locator (URL). If the URL is accessed, the specific function bound to the route is being called and can return the right page accordingly. These routes give the called function the ability to determine which HTTP request was used when accessing the URL and change

[2]https://www.python.org/
[3]https://www.javascript.com/
[4]https://flask.palletsprojects.com/en/stable/

the return value accordingly. Moreover, the routes can be organized in blueprints and the full application can be hosted on a development server at a specific port for easy testing.

The easy implementation and testing possibilities with Flask, as well as the ability to scale up for complexer applications makes the framework a good choice for this thesis.

2.4 OpenCV

OpenCV[5] is an open source computer vision library which makes accessing and utilizing visual data such as a recording or a stream from a camera, easier for developers. It is also supported by Python, which makes it the obvious choice for the part of the remote control system that is related to utilizing the camera on board the REAKTOR.

2.5 Docker

Docker[6] is a tool that is used for deploying software on different platforms. This is done by writing a set of instructions for running the application in a Dockerfile and then building a template called a Docker image from these instructions. An instance of this application can then be made by creating a Docker container and running the container starts the application in the environment specified by its image.

A Docker container uses container-based virtualization, which makes the containerized application runnable on different Operating System (OS). Multiple different containers can be run on the same host OS, isolated from each other, while sharing a OS kernel. This makes the application portable and deployable on different platforms. This is important for the scalability of the remote control system.

[5]https://opencv.org/
[6]https://www.docker.com/

Chapter 3

Related Work

This chapter reviews different existing projects implementing remote control systems as well as projects that relate to the specific use case in the railway context. Projects with an intersection or that are an inspiration for the work of this thesis specifically are also being discussed in this chapter.

3.1 Remote Control

The following sections discuss articles and projects with existing remote control systems with a similar use case. Systems for remote controlled vehicles as well as the networks for the remote control systems are being reviewed.

3.1.1 Safety

As described in Chapter 1, a remote control system ensures safety during a transition from human driven vehicles to autonomous vehicles in the different stages of the GoA and also can be used as a fallback system in later stages of automation. Based on that, safety is a key aspect for remote control systems in the context of autonomous vehicles in general, but by extension the safety of the remote control system itself is also very critical.

In their paper on standardization considerations for autonomous train control [PHL22], Peleska et al. define different modes for autonomous trains. Based on the general safety considerations for remote driving by Tonk et al. [TBB+21], the authors focus on the operational safety for remote control systems, which describes the safety of the system in interactions with its environment and other systems. In their approach, the operational mode for autonomous trains is changed, based on the availability of the sub-functions of the autonomous system. If the train is fully functional, it drives autonomously, but in case of sub-functions of the system being unavailable, the train changes the mode to be driven remotely and not autonomously. This describes an exception handling from the autonomous system and a remote control system is the solution for this emergency. The authors state further, that the architecture for autonomous trains presented in their article is only applicable for freight trains and metro trains currently, as no reliable solution for obstacle detection in high speed trains is available currently.

3.1.2 Connectivity

Connectivity is a big part for a remote control system to function properly. In his thesis on remote controlled train operations [Myh23], Myhrvold does a comparison of the network generations in their applicability for remote control systems. The author also discusses different network protocols and requirements for video streaming in the context of remote control. This work is very valuable for this thesis, because of the similar use case and especially the work done in context of a network needed for a remote control system.

In regard to the system architecture of a remote control system, Hwang and Yu present a client-server system using specifically ZigBee technology [HY12]. For this thesis, the system architecture and structure is very similar and relevant for the concepts.

Liu et al. performed an investigation on remote control using a Long Term Evolution (LTE) network [LKD+17]. They state that an LTE network is generally bound to have too much delay for a remote control system that needs to operate in real time but can be used with specific alterations on the frame arrangement in video streaming. This is important for this thesis for the assessment of the network.

The remote control setup by den Ouden et al. [OHS+22] has a very similar approach to the work in this thesis. Their setup can be connected via 4G and 5G cellular network and their measurements for latency and packet loss are valuable information for the assessment of the network for this thesis. The authors state that the system can be implemented on 4G, but the latencies have dropped to half when utilizing 5G in their testings.

3.1.3 MONOCAB

MONOCAB[1] is a research project with the technical university, Technische Hochschule Ostwestfalen-Lippe, having the lead in the project with support from other German universities and societies. The project is set to add to the general railway traffic with a small and efficient rail vehicles. These vehicles are able to drive on a single rail and therefore cover another niche of the railway traffic in Germany in general. On demand traffic is a focus for MONOCAB as well, giving it a very similar idea to the REAKTOR project in general. Key differences for the projects are in the way that traffic in opposite directions is handled, as well as the size of the vehicles.

For this thesis, the communication network [BNS+23] is very relevant. The authors of the article discuss a communication concept using a 5G network and cover communication between multiple vehicles, as well as from a vehicle to the infrastructure of the rail system. A use case regarding remote control from an external control centre is also part of the article and especially relevant for this thesis. The authors focus on the data throughput needed for the system and the problem of latency, which is key for a human to be able to react to real time obstacles on the track.

[1]https://www.monocab-owl.de/

3.1.4 CAPTN Förde5G

The goal of the Förde5G[2] project is to develop solutions for autonomous and safe traffic with ferries. The focus of the project is set on connectivity with a 5G network and autonomous traffic on the water supported by sensors and cameras. A land based control centre is another objective for Förde5G. This project is very relevant for the context of this thesis in terms of remote control as a main goal for the project and remote connectivity being a main problem to solve.

Smirnov and Tomforde explore in their article [ST24] on data transmission in 5G networks. They have a focus on livestreaming data, and the bandwidth and maximal latency needed for a remote control system to function properly. This study is especially relevant for the network assessment part of this thesis.

3.2 REAKTOR

In the following sections, relevant projects in the context of digitizing the railway network and their intersections in communication with rail vehicles, remote control, and with REAKTOR in general, are being discussed.

3.2.1 The ARTE Project

A project that centres around ATO over ETCS is the Autonomous Regional Train Evolution (ARTE)[3] project by the French group Alstom which specializes in railway traffic and technology. The goal for the project is to revolutionize the German rail network and Alstom partnered with multiple German institutions to achieve it. The project aims to utilize the existing railway network more efficiently, by using automation technologies on existing trains, achieving the GoA 4.

Another focus for the project is a fleet management solution for trains. This is about real time diagnostics for the functionality of the trains, using remote monitoring. This relates to the content of this thesis in terms of achieving remote connectivity for trains at all times as well as in terms of the use case, which is also part of digitizing the railway network.

3.2.2 DSD

The Digitale Schiene Deutschland (DSD)[4] is a program that aims to digitize the railway network in Germany as well, carrying out many different projects concerning digitization of the infrastructure, automatic trains AI supported traffic control and improving data transfer and connectivity [FWH+24]. The goal for DSD is a GoA 4 for railway traffic. The infrastructure

[2]https://captn.sh/foerde-5g/
[3]https://www.alstom.com/arte-pioneering-automated-regional-trains
[4]https://digitale-schiene-deutschland.de/de

needed for remote connectivity for rail vehicles on the track at all times is critical for achieving this goal.

Especially the focus on data transfer and connectivity, as well as automatic train traffic with different stages of automation relates well to the key problems in the context of this thesis and also with the REAKTOR project as a similar use case.

3.2.3 NGT

For railway traffic in rural regions, the Deutsches Zentrum für Luft- und Raumfahrt (DLR) is working on a project called Next Generation Train (NGT)[5]. This innovative rail vehicle has a modular approach to its construction and thus can be adapted to the frequency of traffic on different tracks. It is automatic, lightweight, climate friendly and constructed for tracks in rural regions.

This project is relevant for the context of this thesis through the modular approach in construction and communication. The DLR implemented a system for telecontrolled coupling of vehicles to a clutch. The trainset driver executes control command on the different modules of the train via radio communication [WKK+12]. This has a similar structure to a remote control system with a central unit to take control of multiple vehicles, and thus is relevant to the concepts of the thesis, as well as a similar use case in the railway context.

3.3 Remote Control Panel

For the UI of a remote control system in general, it is important for the system to provide the remote driver with the necessary tools to do the job as intended. Luke et al. state in their article [LBP+06] that train driving is primarily a visual task and it is key for the driver to have clear vision on the track and on signals that might occur.

In another article by Michel et al. [MRB25], a concept for a remote control desk for train drivers is presented. In the approach, a substitute system for each of the human senses is integrated into the remote control desk, in order to give the driver the ability to drive remotely in a way that relates to physical controlling the vehicle as much as possible. A key focus is the visual system with the systems for the other senses supporting it.

The following sections of review projects that were an inspiration for the work of this thesis and the planning of a remote control panel in general.

3.3.1 Siemens Mobility GmbH

The Siemens Mobility GmbH[6] carries out many different projects in the greater scope of digitizing and advancing the railway general in many countries. They work on a wide variety of subjects around the railway system, ranging from different models of railway engines,

[5]https://www.dlr.de/de/fk/forschung-transfer/projekte/innovative-fahrzeugkonzepte/ngt-taxi
[6]https://www.mobility.siemens.com/global/de.html

over new systems for realizing railway traffic in a more efficient way to autonomous trains in general.

Specifically, ATO over the ETCS is a focus for the Siemens Mobility GmbH that is similar to the REAKTOR project. On the InnoTrans 2024[7], the Siemens Mobility GmbH presented a remote control panel that inspired the work specifically, of this thesis and can be seen in Figure 3.1. The idea of a remote control panel with a livestream from a camera to a dedicated device for remote controlling, the REAKTOR has it's origin in this event.

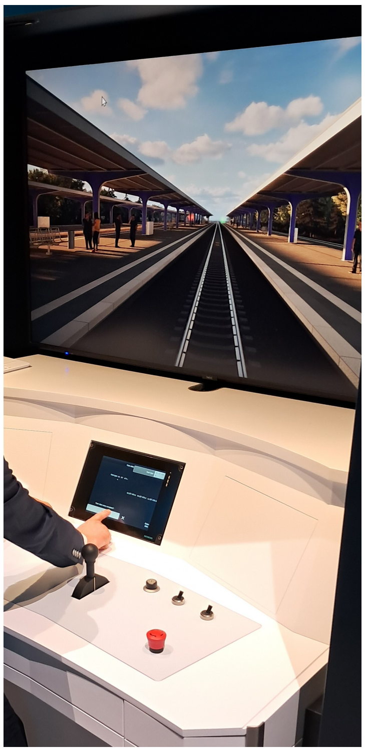


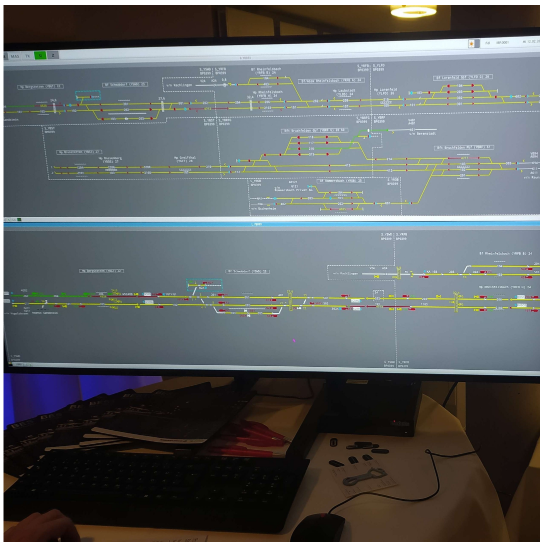
Figure 3.1. Inspiration for a design of a Remote Control Panel, based on a stand of the Siemens Mobility GmbH at the InnoTrans 2024, Figure credit: Dr.-Ing. Alexander Schulz-Rosengarten, CAU

3.4 Scheidt & Bachmann GmbH

The Scheidt & Bachmann GmbH is another company with a focus on railway technology. Especially the training system for control centres[8] which the company presented, as seen in Figure 3.2, is a design inspiration for a future version of a remote control panel as presented in this thesis.

[7]https://www.innotrans.de/

[8]https://www.scheidt-bachmann.de/es/signalling-systems/products-solutions/simulation-systems


(a) Information on the track

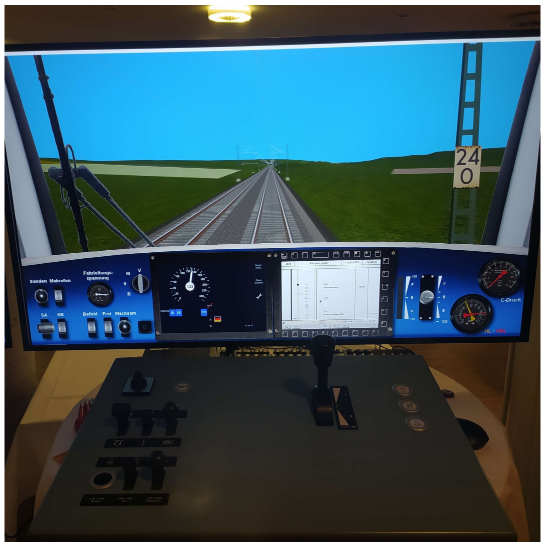

(b) UI for the vehicle

Figure 3.2. Inspiration for a design of a Remote Control Panel on the base of the training system for control centres presented by Scheidt & Bachmann GmbH

Chapter 4

Concepts

Remote control systems in the context of rail vehicles are mainly used to ensure safety during development stages and during a transition on the way of reaching GoA 4, as introduced in Chapter 1. The other main use case for a remote control system is as a fallback system in later stages of automation [CM23]. This makes a safe interaction of the remote control system with other components of an autonomous vehicle, a critical aspect for concepts and implementation. Safety in terms of the interactions in the remote control system itself is a crucial aspect by extension as well.

The first section of this chapter discusses the key aspects and requirements for a remote control system that the concepts must cover. In the following sections, the concepts for the remote control system presented in this thesis are discussed, referencing the key aspects from the previous section accordingly. Firstly, the architecture of the system is discussed, followed by concepts for handling interfacing with other components of an autonomous controller, which handles autonomous driving on the vehicle side. Concepts for the UI, as the way a remote operator can interact with the system are being presented, and lastly, how the system is intended to be used regarding connectivity is described. The remote operator in the context of this remote control system is hereafter referred to as the user of this system.

4.1 System Requirements

As mentioned earlier, the safety of the system is a key requirement in interaction with other components of an autonomous vehicle, mainly the autonomous controller, as the component that decides on the behaviour of the autonomous vehicle on the track. The interaction between the components of the remote control system itself must be safe as well. In addition to that, security is an important aspect for any system in an actual use case. The concepts presented in this thesis describe a way, security features can be implemented, but the other requirements are in focus, as security is less important during research and development stages.

Another requirement for a remote control system can be described with functionality of the system. The system has to give the user the controls and information necessary, to be able to use the system in the intended way. This means for a remote control system, access to speed and brakes for the vehicle, and a livestream as means to remotely control the vehicle on sight with real time information from the track. A livestream from at least two different sources is required, one for each of the two directions, a rail vehicle is able to drive in. Real time information is not limited to a livestream, but could include additional information as well, which is discussed further in section 4.4.1 and in section 7.2. A means to remote control multiple vehicles in one system is a requirement for the REAKTOR project as a use case especially, but also a useful feature for a remote control system in general. Another aspect of the functionality is a way for the system to handle multiple users. Passenger comfort is an aspect for further stages of automation and thus is only discussed at a basic level in this thesis.

Reliability is the third key requirement for a remote control system. This means, that the system must be available at all times and on the full track for the specific use case. For this system, this means mostly connectivity in regard to the network that is available on the track.

4.2 System Architecture & Dataflow

One of the key requirements presented in the previous section is the overall functionality of the system. This includes the integration of multiple vehicles and multiple users. To accommodate for that, the structure of the remote control system presented in this thesis, is based on a classic client-server-architecture. The specific architecture for this system can be seen in Figure 4.1.

The three main components of the system consist of the user side, the vehicle side, and the server side. Both the user and the vehicle are communicating with the server, and the server coordinates the dataflow. Every autonomous vehicle that is integrated in the system has an autonomous controller, a program that decides on the behavior of the vehicle on the track. The vehicle side component needs to have an interface with the autonomous controller and needs to be located on the same vehicle. Every copy of the vehicle side component of this system then needs to communicate to the same server in order to give a user the ability to remote control different vehicles. On the user side, a UI on a Web Client is part of the concept, which also communicate to the same server. Thus, the server needs to coordinate requests and data from multiple users and vehicles.

As can be seen in Figure 4.1, a user interacts with the remote control system via a Web Client. With the UI provided to the user through the Web Client, the user can choose to take control of a specific vehicle and set its target speed and direction and to release the control if no longer needed. An emergency stop button to set the target speed to zero immediately and activating brakes of the vehicle is another action for the user. On the server side, this input from the user is read in the Web Server.

The data from the user is read by the Web Server and then sent to the RC Server, which coordinates the data sent between vehicle and user and also coordinates remote control for different vehicles. The RC Server sends this information to it's counterpart, the RC Client on the vehicle. The RC Client is a program that is integrated in the autonomous controller on the vehicle and organizes the data flow on the side of the vehicle. Lastly, the Cam Client is a program on the vehicle as well. The livestream from the camera on board the vehicle is handled by the Cam Client and everything is coordinated by the RC Client. In parallel to the livestream from the camera, the current speed of the vehicle is sent back to the server and

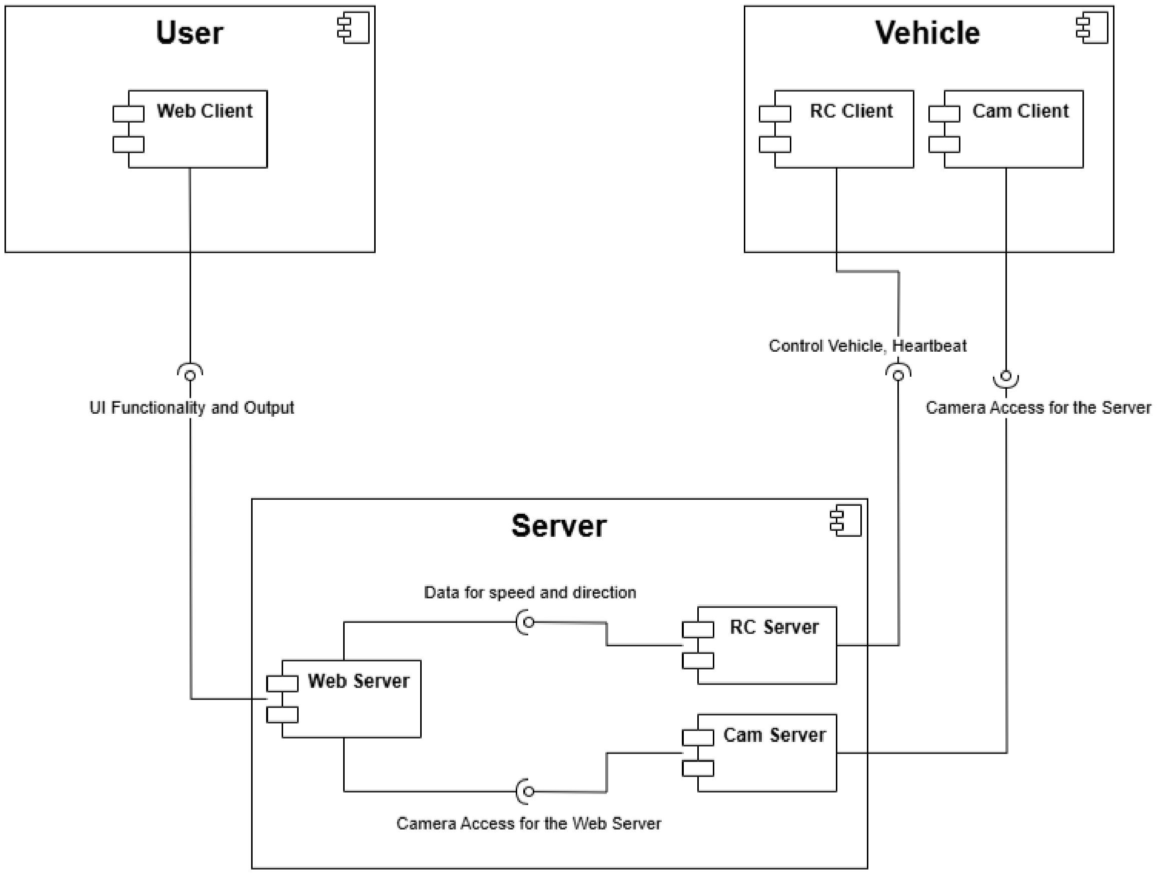

User
Web Client
Vehicle
RC Client
Cam Client
UI Functionality and Output
Control Vehicle, Heartbeat
Camera Access for the Server
Server
Data for speed and direction
RC Server
Web Server
Cam Server
Camera Access for the Web Server

Figure 4.1. Component diagram for the current system

everything is provided from the server to the user via the UI on the `Web Client`. As stated in section 4.1, each vehicle integrated in the system must have at least two different sources for livestreams from different cameras. Those have to be either changed manually on the user side in the UI, or changed by the system, according to the driving direction from the vehicle. The second option is the one that is more applicable to a UI via a `Web Client`, while the first option could be a valuable addition for a UI via a remote control panel. Changing the source according to the driving direction is information that the `RC Client` as the program coordinating the dataflow on the vehicle side can send to the `RC Server`, which can provide the data to the `Web Sever`, which can change the output for the user.

For safety considerations of the system itself, it is critical that any case of network error or lost connection in general is handled safely by the remote control system. To achieve this, the data on target speed, target direction, current speed, and current direction is sent between the `RC Client` and the `RC Server` in a continuous interval, in addition to the communication initiated by the user. The dataflow in this interval is realized as a `Heartbeat` function for the vehicle, as can be seen in Figure 4.1. If the information does not reach the vehicle in a fixed interval, the connection is assumed to be broken and the RC Client stops the motors for the vehicle. Thus, any variant of connection failure is handled, while integrated into the necessary exchange of information. Similarly to this, the RC Client also sets the target speed for the vehicle to zero, when control is released by the user.

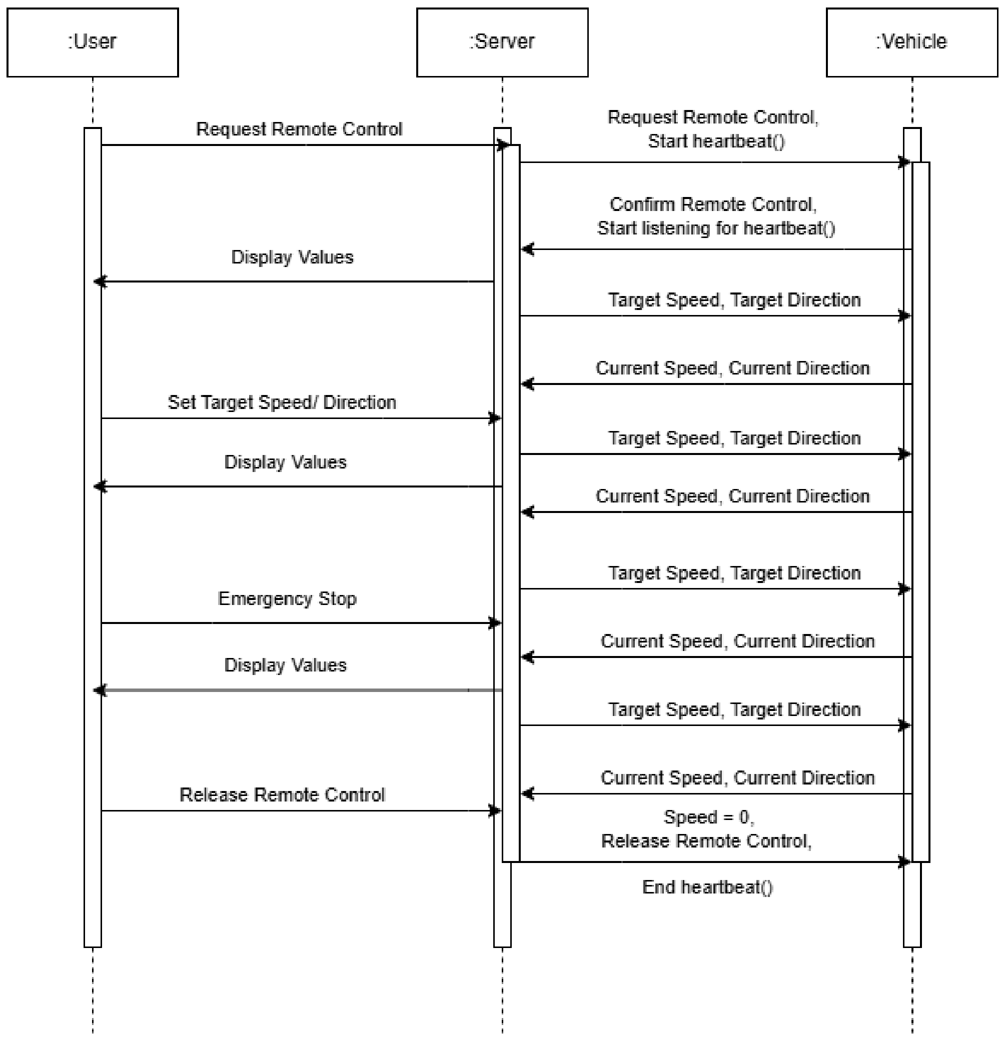

:User
:Server
:Vehicle
Request Remote Control
Request Remote Control, Start heartbeat()
Confirm Remote Control, Start listening for heartbeat()
Display Values
Target Speed, Target Direction
Current Speed, Current Direction
Set Target Speed/ Direction
Target Speed, Target Direction
Display Values
Current Speed, Current Direction
Target Speed, Target Direction
Emergency Stop
Current Speed, Current Direction
Display Values
Target Speed, Target Direction
Current Speed, Current Direction
Release Remote Control
Speed = 0, Release Remote Control, End heartbeat()

Figure 4.2. Sequence diagram, presenting the Heartbeat function of the system

For internal safety reasons, but also as a start for covering the passenger comfort as a soft requirement, the system only allows the direction to be changed when the current speed is at

0. Users should really be avoiding a situation like this on their own, but in case of an error by the user, the system should not be able to reach such a state by its definition.

Figure 4.2 shows how the data is sent between the user, the server and the vehicle. The Heartbeat function is activated once remote control is taken and the data between server and vehicle are exchanged in the interval defined by this function. If the user releases control, the Heartbeat function ends, and the server goes in standby regarding this vehicle.

Regarding multiple users, the server must have restrictions for remote control access for multiple users on the same vehicle. This can be achieved through server side constraints, using semaphores or other means for locking variables. Multiple users must be forbidden from accessing the same vehicle at the same time on the server side. The structure of the system is modular on the server side and vehicle side, adding to the scalability of it. Lastly, security for the system can be implemented as a stage before, the user is able to remotely access a vehicle. A basic way is a username and password combination for accessing each vehicle or the system itself.

4.3 Interfacing

On an autonomous vehicle, the RC Client needs to interact safely with other programs. The most important being the autonomous controller itself. This means for the system to notify the autonomous controller on the request by the user to remote control and also to return the control of the vehicle to the autonomous controller with the system in a defined state. This safety requirement is satisfied by stopping the vehicle fully, before control is given back to the autonomous controller. Peleska et al. state in their paper on standardization for autonomous train control [PHL22], that the autonomous train should have different modes that should change based on the availability of critical system sub-functions. The system should then automatically go into a mode where it is only allowed to be remote controlled and no longer run autonomously. The system presented in this thesis focuses more on the user deciding when to take remote control and when not to.

In both cases, the communication with the autonomous controller on the rail vehicle needs to be sound to make changes in how the vehicle is controlled as smooth as possible. In this remote control system, the user requests control of the vehicle and the request is sent to the RC Client on the vehicle via the RC Server. The RC Client then requests control from the autonomous controller in which it is integrated. This interaction can be seen in Figure 4.3. The autonomous controller should then handle a safe transition from an autonomous driving mode to a remote controlled mode, where the vehicle only takes input from the RC Client. On a connection error, as well as on releasing control as a user, the RC Client sets the target speed for the autonomous controller to zero and then notifies it that remote control is released, as can be seen in Figure 4.3. The controller should then return to an autonomous driving mode if all sub-functions are available as expected.

In the REAKTOR project, as the current use case for this remote control system, the RC Client notifies the autonomous controller about a request to take remote control and the

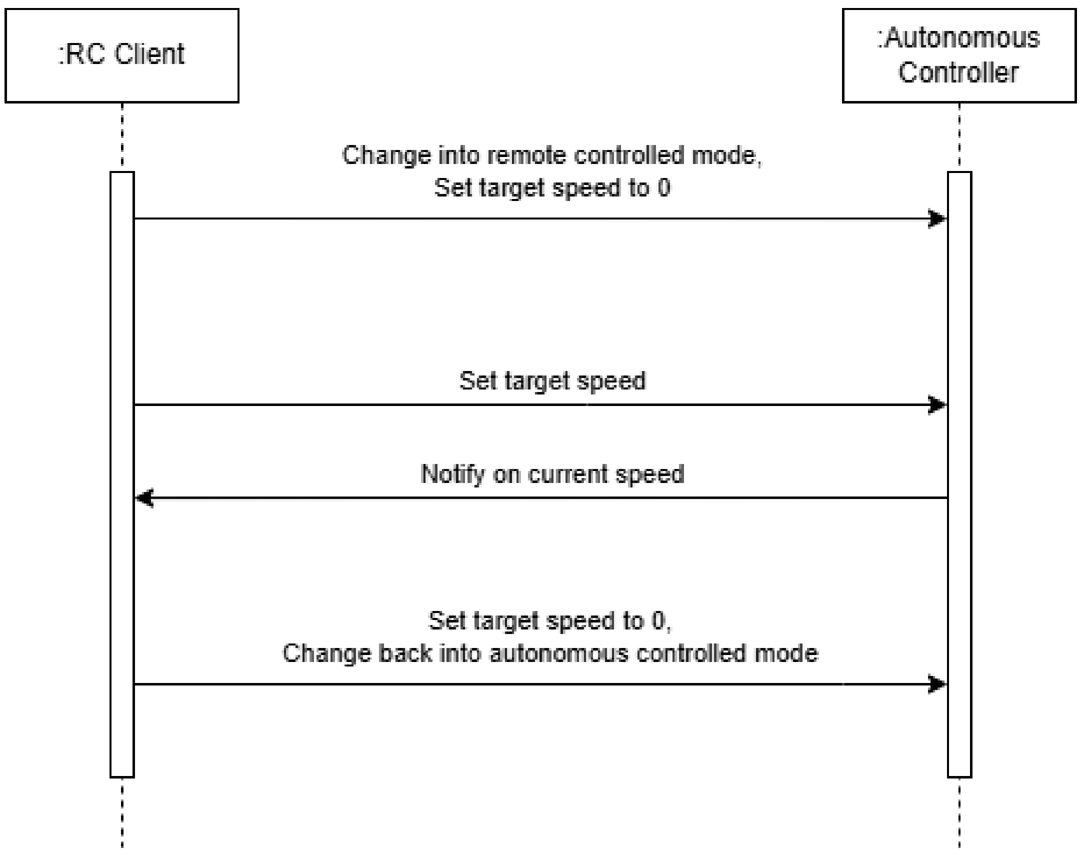

:RC Client
:Autonomous Controller
Change into remote controlled mode, Set target speed to 0
Set target speed
Notify on current speed
Set target speed to 0, Change back into autonomous controlled mode

Figure 4.3. Sequence diagram, showing the safe interfacing on the vehicle side

autonomous controller grants this control, setting the REAKTOR in a remote controlled mode. The autonomous controller provides the RC Client with the current speed and direction as well.

Both the remote control system and the AI-based obstacle detection for the REAKTOR need access to a livestream from the camera on the vehicle. This is achieved by utilizing a single livestream from the camera and using it as a source for the different projects to use in parallel. Interactions between the remote control system and the other parts of the REAKTOR projects as the current use case are handled by the autonomous controller. This includes the digital twin and the on-demand app.

4.4 UI

With the human role in a remote control system being very similar to physical control of the vehicle [Ahv16], it is essential for the remote driver to have access to a quality of information that is as close as possible to the physical information a driver on board the vehicle would have. This information has to be provided to the remote driver by a UI. Ahvenjärvi states further that the user needs to have up-to-date information and essential information.

The most important information for a train driver is anything that happens in front of the vehicle in driving direction. With train driving being primarily a visual task [LBP+06], the most important part of the UI is a visual image of the relevant information provided by the system. Essential is sight on the track in front of the rail vehicle, as well as any kind of obstacles that are on the track or could appear on the track in the future. In addition to that, any kind of railway signals must be visible and recognizable to the user.

In a human driven train, the driver sits in the cockpit of the vehicle and has access to physical controls. Those controls, in addition to the sensory perception of the human driver on the track, need to be substituted by the UI for a remote control system. Simulating a real cockpit as a UI for the system in a replica would be the most precise way to achieve similar circumstances for a remote driver compared to a human driver, but this would be a stationary and inflexible way to for a system that is designed for development of autonomous vehicles as well. A more flexible way with a set of controls that is more focused on the specific controls needed for a vehicle is a remote control panel, as can be seen in Figures 3.1 and 3.2. A concept for a remote control panel for the system presented in this thesis is presented in the next section. A more simplified and flexible approach is a UI via a Web Client. This way, the controls can be implemented and adapted according to the requirements of the specific system, that might change during development stages. Moreover, a Web Client allows for a user to request remote control of a vehicle while physically on the track, as the UI is not stationary. A concept for a UI on a Web Client is provided in section 4.4.2, and is implemented for the specific use case REAKTOR in section 5.1.

4.4.1 Remote Control Panel

As stated earlier, a remote control panel is an approach for a UI of a remote control system that is more flexible in location and specific design compared to a real cockpit or a replica, but maintains physical components for controls. This gives the user an easier way to utilize existing knowledge as a train driver and has the option to integrate additional information. A monitor for a livestream is one of the most crucial parts for the UI. Michel et al. discuss in their article on a remote control desk for train driver [MRB25], the importance of integrating input for the different human senses, to supplement the input a driver would have while physically driving a train. They state as well that the visual system is the most important aspect, but can be reinforced by additional sensory systems.

Compared to a UI on a website, the remote control panel is better dedicated for the job and can integrate other systems to give the user an experience that is as similar as possible to physically controlling the vehicle. In contrast, the website holds the advantage over a remote control panel in terms of its flexibility in location and devices in general. The user can physically be beside or on the vehicle while remotely controlling it and still have the option to take remote control from a distance. This is especially valuable during development and in

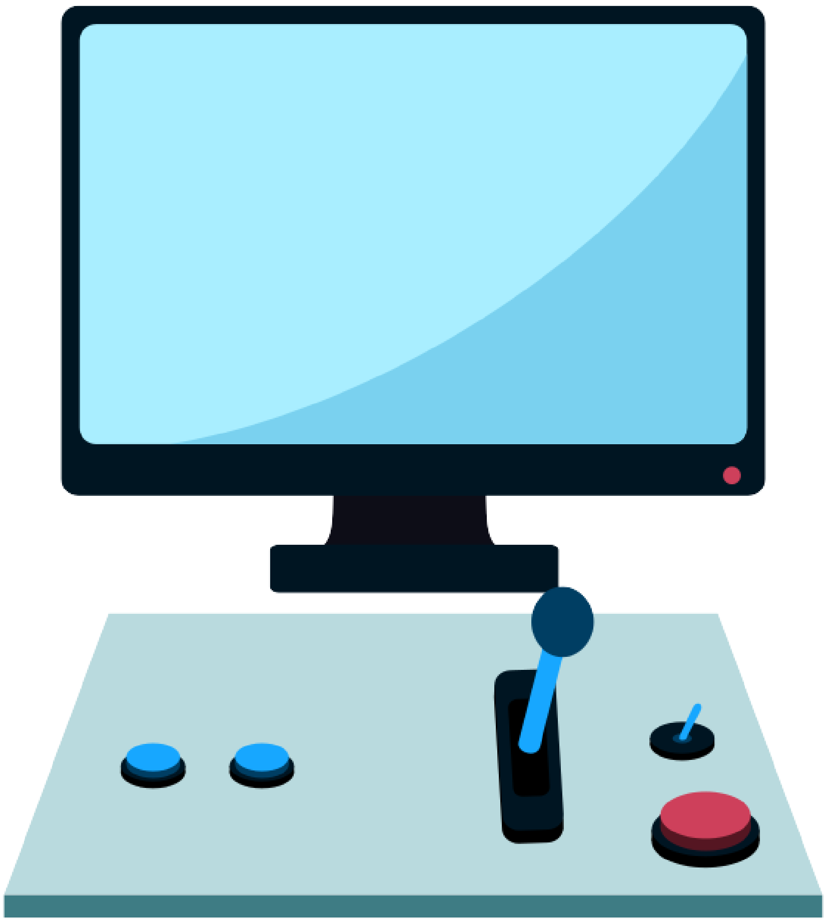


Figure 4.4. Design for a Remote Control Panel

testing stages for an autonomous vehicle. In contrast to that it is possible for a remote control panel to be better dedicated for the job and integrate other systems beside a visual system to aid the user.

Currently, a remote control panel is planned for future work of this thesis, designed for the REAKTOR as a current use case, but applicable for other use cases in the autonomous railway context as well. A possible design can be seen in Figure 4.4. The panel consists mainly of a monitor with a variation of the surface from the website, with the livestreamed video from the camera in the centre and a display for target speed and current speed below. The hardware components consist of a drive control lever for setting the target speed via acceleration and a button for an emergency brake in red. On the left side of the panel is space for multiple buttons. The two buttons shown in Figure 4.4 are used to take control of the vehicle and release control back to the autonomous controller. Moreover, a mouse and a keyboard are connected to the system, to login at the start and switch between multiple vehicles in a list. These external hardware components can be stored in a shelf below the other hardware components.

This concept for a remote control panel contains the key components in terms of functionality for the user. Livestream, controls for speed and brakes are integrated as well as a switch for remote controlling multiple vehicles. The physical components of the panel can be supplemented and adapted for the controls that a specific rail vehicle that is integrated in the system needs. Further ways to expand on the primary functionality of the remote control panel is discussed in 7.2.1.

4.4.2 Web Client

On a Web Client, a UI can be provided to a user in a more simplistic and flexible way, and can be easily adapted for additional controls and information. This is especially helpful during earlier stages of development, as discussed earlier. Thus, a UI on a Web Client is the first version of a UI, that is implemented. In later stages of development, a remote control panel might be the preferred variant, as a dedicated device. On a Web Client, the user has the option to control the vehicle, independent of the location that the user is in. During early development stages, this is especially helpful to provide a means of intervening during tests of other components of autonomous vehicles on the track.

The design for a UI presented in this thesis, shows the information presented in the section 4.4.1 on the monitor. The physical controls for speed and brakes, as well as switching between cameras and vehicles, are integrated in on the Web Client, as buttons. A button to take control and a button to release it is therefore part of the design for the UI. Additionally, it is needed for the user to have a way to set the speed and direction of the vehicle. This is done in a target speed and current speed format for the user, to present a way to give exact values from the user to the vehicle in real time. The target speed describes the speed value that the user wants the vehicle to achieve, while current speed describes the actual speed of the vehicle at this moment. Target speed is read from the Web Client by the server and forwarded to the vehicle to be implemented there. Current speed is sent from the vehicle to the server and then provided to the user on the Web Client. The buttons on the UI allow the user to influence the target speed by adding to it and subtracting from it. This ensures a smooth acceleration and deceleration, big differences between the target speed and current speed of the vehicle can not happen suddenly. Both the target speed and current speed values are displayed for the user constantly below the livestreamed camera. As a safety insurance for any UI of a remote control system, an emergency stop button is integrated.

This is the basic concept for a UI of a remote control system with the essential functions. There is a multitude of possible additions available for the UI in general, including a dedicated device, a remote control panel, which are discussed further in 7.2.1.

4.5 Connectivity

This remote control system is designed to work on a cellular network, though it's current tests are only via Wi-Fi connection. The main requirements for a remote control system are to be functional, safe to use and reliable. For safety reasons and most importantly for the reliability of the system, it is crucial for the system to run on a network capable of transferring the needed data reliable at all times. With the design for a remote control desk by Michel et al. [MRB25] in mind and also for the design of the remote control panel and the UI on the Web Client in this thesis, the main task for the network is the livestreamed video.

The tests for the system are currently done on a Wi-Fi network with reliability as a focus. For this reason, Transmission Control Protocol (TCP) is the protocol used for communication in this remote control system. This is especially critical for the Heartbeat function and the communication of the system in terms of taking or releasing remote control, as well as the speed and direction data sent between vehicle and server. For the livestreamed video, another protocol could provide less latency in further testing. User Datagram Protocol (UDP) is an example for this, as Myhrvold describes in his thesis [Myh23]. He compares different network generations, protocols, and the requirements for video streaming. Compared with the studies by Liu et al. [LKD+17] on LTE network and den Ouden et al. [OHS+22] on 4G and 5G cellular network in the context of remote control, this is the base for the assessment of the network for the remote control system in this thesis.

As the system has not been tested on a cellular network on a track yet, most of the details concerning the network needed are done on a theoretical basis in Chapter 6, based on the work of Birkan Denizer, from CAU.

Chapter 5

Implementation

This chapter describes the current implementation of the concepts for the remote control system and discusses them further. While the concepts are applicable for different rail vehicles, the implementation is done for the specific use case for the REAKTOR project specifically. The cases for multiple users and multiple vehicles are not implemented yet. As described in section 4.2, the implementation follows a client-server-architecture on the baseline.

In this chapter, the implementation details are being discussed in layers. Firstly the view of the user with the visual components is described. Following the data flow from user to vehicle, the server side is the next part of the implementation for this chapter. This includes the storage of important data and different interfaces for the different clients of the system. How the server can be deployed on different platforms is also part of this section. Next, the vehicle side of the implementation follows. This includes the clients for the communication of speed and direction for the vehicle, as well as the camera. The program on the vehicle side is also integrated in the greater system for the autonomous controller of the vehicle. These sections explain how the different components for the remote control system are implemented and how they interact with each other.

Scalability options as described in Chapter 4 are being discussed for this implementation specifically. Most importantly, this includes multiple cameras and vehicles, but also additional information provided to the user.

With the use case of a remote control system in general being autonomous rail vehicles, the final sections for this chapter discuss the integration of this remote control system in the software for the autonomous vehicle of the REAKTOR project specifically. The sections then explain the current testing environment for the system, being a demonstrator for the REAKTOR.

5.1 User Side

The user interacts with the system using a Web Client. The UI is defined by HTML templates in a Flask application on the server side. A login page is a placeholder for potential additional rail vehicles. As there is a single demonstrator available for testing, the UI for remote control currently only has a single page implemented, which can be seen in Figure 5.1.

On this page, the livestream is provided to the user in a window in the centre. Below that, the values for target speed and current speed are displayed. The target speed is a value that the user can determine and is sent to the vehicle continuously while remote control is taken.

In contrast, the current speed can not be altered by the user and is instead determined by the data sent from the vehicle. This also happens continuously, and the page is refreshed in a fixed interval to show changes for the current speed of the vehicle. A more dynamic version of the implementation for the Web Client is part of future work on the system presented in this thesis and described in section 7.2.1.

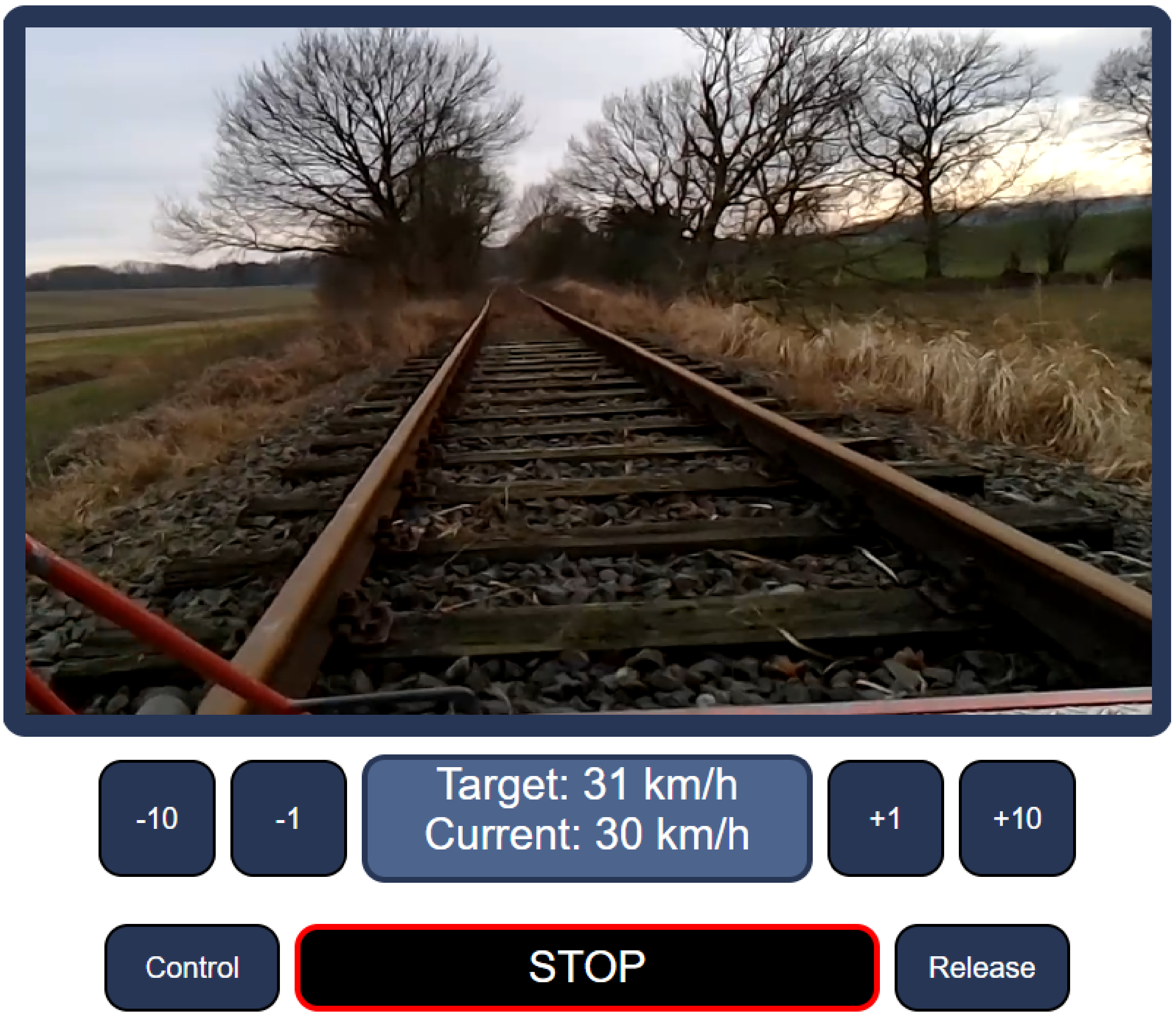


Figure 5.1. Mockup for use of the UI via a Web Client on a track

The user then has access to 7 different buttons. These are for taking control of the vehicle, releasing this control, adding and subtracting from the target speed for the vehicle and an emergency stop button. When the user presses any of the buttons, a HTTP POST request is sent from the Web Client to the server. Determined by the specific button, the server updates the data for target speed and direction as the user intended and responds with an updated page in the Web Client. When the user presses a button to take control or to release control, the server sets the target speed to zero. This is done to ensure a safe transition from autonomous driving to remote control. The autonomous controller should have its own mechanisms in place, to avoid the vehicle being remote controlled while driving. Still this remote control system starts sending target speed and direction continuously and thus, the first order when taking control is for the vehicle to stop and wait for the instructions intended by the user. In the same way, the user of the remote control system should always have the vehicle stop before releasing remote control to give the autonomous controller a safe starting point before resuming its task. In case the user forgets, the system sets the target speed for the vehicle to zero before releasing remote control.

For setting the target speed and direction for the vehicle, the UI provides the user with 4 buttons. They add 1 or 10 to the target speed value or subtract 1 or 10 from it. This provides the user with a way to set the speed for the vehicle in a way that resembles a cruise control system. The system lets the user change the target speed for the vehicle while the controller on the vehicle attempts to align its current speed with the target speed. The speed values as shown in the UI have the unit kilometers per hour (km/h) and can be set positive or negative. If the value is positive, the system sets the direction to forward and a negative speed value sets the direction to backward.

5.2 Server Side

On the server side, the data from the user and the vehicle are organized and exchanged. The interfaces between the different components of this system are defined at three different parts of the server side program. A Flask application defines the Web Server and by extension the Web Client and the interaction with the user. In addition to that, the interaction with the vehicle is done via different interfaces for the exchange of data for speed and direction and for the livestream of the camera, respectively. The RC Server handles communication with the RC Client and the Cam Server handles the livestream provided by the Cam Client on the vehicle. These three parts together form the system architecture on the server side. This server can also be deployed externally utilizing Docker containers. In section 5.2.4, this part of the implementation is discussed further. The internal structure of the implementation of the server, can be seen in Figure 5.2.

5.2.1 Web Server

The Web Server component of the remote control system is the part that handles communication with the user via the Web Client on the server side. For this purpose a Flask application is implemented. Flask as a framework for Python can be used for developing simple web applications easily, as well as being able to expand them for different use cases and general scalability. For scalability considerations discussed earlier in the system architecture part of the Concepts chapter, the Flask application factory pattern is utilized for the implementation

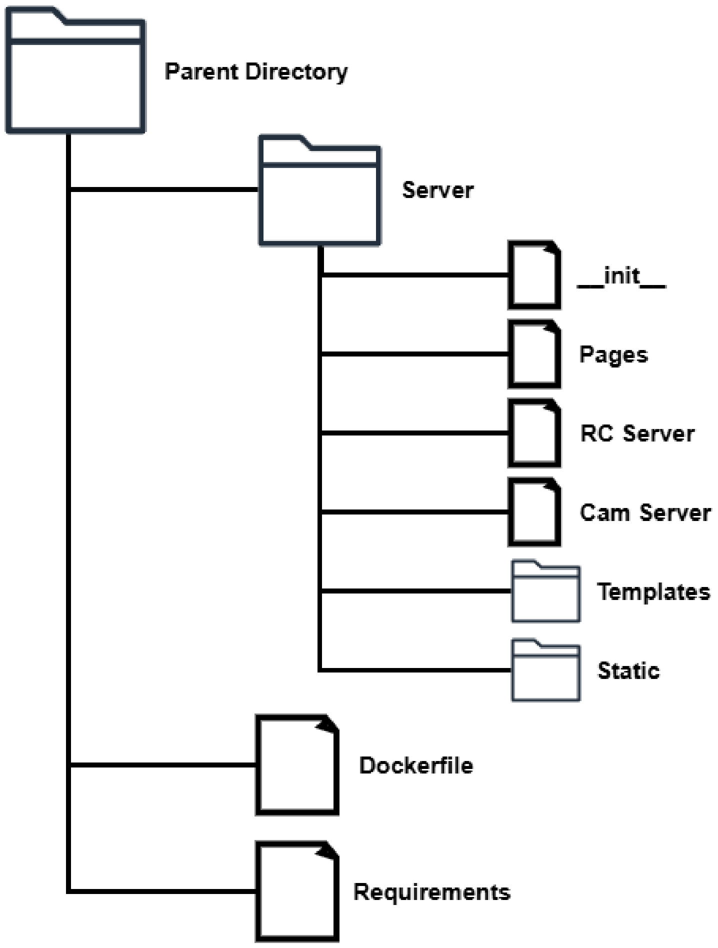


Figure 5.2. Structure of the server components

of the web server[1].

This pattern allows for the application to be run with one console command, as well as being more structured for developing, utilizing a modular approach for the different features of the server. The structure of the Flask application consists of a directory which holds the complete server structure and an `__init__` file just below, as can be seen in Figure 5.2. The `__init__` file is the access point for the program, when executed, as this implementation specifically is written mostly in Python code. Next to it is a folder holding static content for the website and another folder holding HTML templates for the different pages. In addition to that is a `Pages` file, which handles the requests from the user on the server side. The `__init__` file serves as a starting point for running the application, importing a Blueprint[2].

[1]https://flask.palletsprojects.com/en/stable/patterns/appfactories/
[2]https://flask.palletsprojects.com/en/stable/tutorial/views/

Different view functions handle how the server responds to requests from the web client. These view functions can be organized in blueprints for modularity and purposes of reusing these blueprints when expanding the system. Running the app is done with a terminal command provided by Flask and the website then can be accessed on a hostname and code specified in the command.

The design for the different pages of the website are done in the templates folder using multiple HTML templates. This includes a base template, which sets the structure for the other pages and imports a stylesheet, which is a CSS file from the static folder. In this stylesheet, the colours and overall design for the pages is down, while the HTML templates define the specific text, buttons, and the livestream for the user. Additional content in the static folder can be put into JavaScript files, for smaller dynamic updates. More on this is discussed in the section for the RC Server and also for future work. The most important template for the remote control system is the `rc template`, which defines the page for the UI when remote controlling the vehicle as a user. With the `url_for()` function provided by Flask, this template can integrate the livestream from a URL defined in the Pages file. External JavaScript files can be included in the same way, which is done in this system with a file, used for accessing the target speed and current speed data from the server in the Web Client. Furthermore, the `rc template` defines the different buttons in the UI, which can then be accessed by name in the Pages file.

Most importantly for the web server is the Pages file, which holds the interactions with the Web Client and also coordinating the RC Server and Cam Server, utilizing their data on the server side. For this purpose, the RC Server and Cam Server files are imported into the Flask application. In the Pages file, the access points for the different webpages of the system are implemented with the route decorator. In the parameters for the different routes, specific URL for the pages are defined and the different requests of the Web Client with the server can be differentiated. For the most important page of the current implementation, the `rc page`, reactions for the methods GET and POST in HTTP requests are being differentiated. For HTTP requests on any page, a view function for the specific URL is being called. As a response to a GET request, the corresponding HTML template is being returned by the view function and shown on the Web Client.

The login page and the`rc page` differentiate between GET requests and POST requests in their view functions. The user is only granted access to the `rc page`, if the correct username and password have been accessed in the same session before. This is defined in a supporting function, the `login_required()` function. Logging into multiple vehicles is a way to implement the usage of the system and a starting point to cover the security requirement for the system in further development, but currently it is just a placeholder for this. If the username and password are valid is hardcoded currently, but Flask provides the possibility of integrating a Structured Query Language (SQL) database, which ensures the scalability of the system for multiple vehicles. In the view function for the `rc page`, the buttons pressed by the user can be recognized by their name in the request form on the server side. The web server sets the variables in the RC Server accordingly. A specific route for the livestream is defined in the Pages file as well and then integrated into the rc template. A generate() function, as a supporting function, utilizes the individual frames from another function from the Cam Server and makes the livestream visible for the user.

Both the RC Server and the Cam Server are imported into the Pages file and started in separate threads with the daemon property. This means that starting the Flask application for the Web Server starts the RC Server and the Cam Server as well. Moreover, the two programs do not have to be ended manually but end as soon as the Web Server is exited because of the daemon property of the thread they run in. All views of the Pages file are registered with a blueprint and imported into the __init__ file. This way all the server components can be started in a single command, but still operate in separate threads.

5.2.2 RC Server

The RC Server program is the program communicating with the autonomous controller of the vehicle through the RC Client. Server side variables for target speed, target direction, current speed and current direction are stored in this Python module. Communication with the RC Client is achieved through the Python socket module.

A key requirement for the system is safety, as discussed in 4.1. The Heartbeat function as shown in Figure 4.1 and Figure 4.2, is key to a safe transition from the autonomous controller to remote control by an external user and critical to handle connection errors for the remote control system safely on the vehicle side. The Heartbeat function is running on the server side but has a counterpart on the vehicle side, as discussed in section 5.3.1.

Being the centrepiece for this program, the Heartbeat function is started on a thread in the Pages file at the start of the application. The Heartbeat function initiates the server side of the connection via Python socket and waits for a connection from the RC Client. When connected, the Heartbeat function awaits the input from the Web Server that remote control, has been requested by the user. It then starts sending the data for the target speed and direction to the RC Client and awaits the feedback concerning current speed and current direction. As the RC Client needs to provide the autonomous controller in the implementation for the current use case, REAKTOR, with separate values for speed and direction, a negative value in the target speed is translated into an absolute value and the direction backward. When reading the values from the RC Client, a backward direction is translated back into a negative value for speed. This calculation is not mandatory for every specific implementation of this remote control system, but has to be done in this specific use case. Both target speed and current speed are written into a JavaScript file, which is integrated in the rc HTML template and displays the values to the user. If remote control is set to be released by the Web Server, the Heartbeat function ends and the RC Server is set into standby until remote control is taken again by the user.

The data is sent between the RC Server and RC Client in a JavaScript Object Notation (JSON) format in a fixed interval. The connection itself is set up with the SOCK_STREAM type, utilizing a TCP connection for a reliable transfer of data.

5.2.3 Cam Server

As a threadable Python class, the job for the `Cam Server` is to read the livestream from the vehicle provided by the `Cam Client` and providing it to the `Web Server` in a usable way. Similarly to the `RC Server`, the `Cam Server` initiates the server side of a Python socket connection on creation and then waits for the `Cam Client` to connect. The `Cam Server` is initiated when the Flask application for the web server is being executed, its daemon thread is started when the `rc template` is first being accessed by the user.

When started, the `Cam Server` starts the Python socket for the `Cam Client` on the server side and accepts the connection from the `Cam Client` on the vehicle. It then reads the frames that are being sent by the `Cam Client` and encodes them into a readable format for the Flask application. In the `Web Server`, the stream then is provided for the `Web Client` on the rc webpage. The socket connection currently uses the TCP protocol for reliability reasons, but UDP might be the better option on cellular network in the future.

5.2.4 Container

To be able to deploy the complete server side code on different platforms, instructions for building a Docker container are part of the implementation. For this purpose, all dependencies for the system are written in a requirements text file. A Dockerfile is defined with the instructions for creating a Docker image. When building the image, the instructions in the Dockerfile are being executed, defining the environment for the project, installing the requirements defined in the text file and storing the command line arguments for running the code. A Docker container can be built from the image and then run in the host environment via the Docker run command.

5.3 Vehicle Side

On the vehicle side, the data sent from the server is forwarded to the vehicle and the data from the vehicle is sent back to the server. The programs on the vehicle side of this remote control system functions as an interface between the server and the autonomous controller on the vehicle. The part of the remote control system on the vehicle side consists of the `RC Client`, the program handling the data transfer for speed and direction between vehicle and server, and the `Cam Client`, provides the livestream of the camera from the vehicle for the server.

5.3.1 RC Client

As the counterpart for the `RC Server` on the server side, the centrepiece for the `RC Client` is the `Read Heartbeat` function. This function is started on the initiation of the program and tries to connect to the remote control server on the socket opened by the `RC Server`. If connected, the `RC Client` waits for input from the server and if remote control is requested by the server, the `RC Client` notifies the autonomous controller.

The RC Client is set to be integrated into the autonomous controller for communication with the remote control server and therefore has an Application Programming Interface (API) with the controller. In the REAKTOR project, as the current use case, this API consists of three functions, the RC Client imports from the controller. These functions are `remoteSetMode()`, for setting the direction of the vehicle, `remoteSetSpeed()`, for changing the target speed of the vehicle, and `setRemoteControl()`, for initiating a remote control mode in the autonomous controller. When the server requests remote control of the vehicle, the RC Client then notifies the autonomous controller, utilizing the setRemoteControl() function. It then continuously reads the target speed and target direction being sent by the server, and notifies the autonomous controller via `remoteSetMode()` and `remoteSetSpeed()`. If a connection error occurs, the RC Client sets the speed of the vehicle to zero and awaits the connection to the server. For redundancy in safety, the speed for the vehicle is set to zero every time remote control is taken or released as well. The Cam Client is started on initiation by the RC Client as well as a separate thread with the daemon property.

5.3.2 Cam Client

The Cam Client waits until it is able to connect to the Cam Server when it's thread is started by the RC Client. If connected, the Cam Client reads the source of the camera on the vehicle, transforms the frames into a byte array, and sends it through the Python socket connection to the Cam Server. The source for the livestream of the camera on board the REAKTOR, as the current use case, is an URL that is being streamed from another part of the project. The stream then is read by the Cam Client using OpenCV. If the connection is lost, the Cam Client stops reading from the source and tries to connect to the server again.

5.4 Scalability

For scalability of the implementation, the server side of the system must be in the focus. The RC Client and Cam Client are integrated in the autonomous controller. If the system is scaled for multiple vehicles, these programs would be integrated in every vehicle connected and therefore only be slightly altered dependent on the API with the respective autonomous controller. The communication with the RC Server could also change in terms of more values for additional information. Multiple URLs for cameras could be integrated in the Cam Client as well and for every vehicle, an additional port would have to be opened by the server. The scalability of the Web Client has been discussed in the UI section on the base of the article by Michel et al. [MRB25] already. Important for the server would be additional buttons, variables and the information would have to be sent in a JSON format that is expanded on the existing one.

Multiple vehicles integrated into the system would mean for the server to expand in terms of pages, at least. Every vehicle would have its own UI on a `rc page` at least, which can be done in a Flask application using different variables as a part for the URL in the routes and filling them in based on the vehicle that the user wants to control. This can be done in the login page by checking a valid name and password for each vehicle in a database in the background, as described in Chapter 5.2.1. Another solution would be giving the user a choice on the starting page of the website and having different blueprints for the pages for each vehicle.

For multiple Web Clients accessing the same URL the server must implement constraints, locking the data for each user through sessions and forbidding multiple users from accessing the same vehicle. With sessions already integrated in the flask application factory, this can be achieved. An idea for integrating multiple cameras for a vehicle would be to change the streaming URL on the server side based on the current direction of the vehicle. Another way would be for the Cam Client, to read a different camera based on the current direction on the vehicle side. Either way, the UI could remain the same.

5.5 Demonstrators

Testing ground and base for the current implementation of the remote control system are the demonstrators for the REAKTOR. A small 1:32 demonstrator has been tested with the system in a laboratory context and the system is also implemented to work on a bigger demonstrator on the testing track between Malente and Lütjenburg. Both these demonstrators are set to be running with a Raspberry Pi and with similar information being needed from the vehicle. For the big demonstrator, concepts for the system running on the cellular network must be implemented first.

Chapter 6

Network Assessment

This chapter reviews several analyses for the applicability of remote control systems on different networks. This includes an analysis of the amount of data that has to be transferred reliably, different network protocols and generations. The network requirements for video streaming in general and the latency and delay for real time applications are especially interesting. In section 6.2, the network currently available on the track between Malente and Lütjenburg for the current use case of the remote control system, the REAKTOR project, is being analysed, based on measurements and assessment by Birkan Denizer, from CAU.

6.1 Needed Network

The main problem for remote driving is the processing and communication delay, as stated by den Ouden et al. [OHS+22]. In remote control systems, the user must be aware of some latency and act accordingly [MRB25] but a loss of performance of the user starts at a delay of 300 ms and at a delay of 1000 ms remote control becomes unfeasible [OHS+22]. In his thesis [Myh23], Myhrvold discusses the importance of a reliable network for remote control systems. He states that the average upload speed is lower than the average download speed and thus, the issues for video streaming are mostly related to the upload speed.

As the remote control system in this thesis is designed to run on the cellular network, the different generations must be compared for capability. A comparison of the capabilities of different network generations, based on the work of Myhrvold in his thesis [Myh23], can be seen in Table 6.1. With Myhrvold stating, that 5G is often referred to as the requirement for remote control, but still can be achieved LTE with a lower reliability, Liu et al. state that it is in fact feasible on LTE and present a frame arrangement solution to accommodate for the inherently higher delay compared to 5G [LKD+17]. Their solution is based around predicting the delay of the network. The authors also state that lowering the frame rate of the video stream can result into a less severe impact for the user than a higher delay. The system by den Ouden et al. runs on 5G as well as on LTE and they state in their article [OHS+22] that their system consumes between 25 and 50 Mbit/s in the upload, which are between 3.125 Mb/ and 6.25 Mb/s. They state that remote control on LTE is feasible, but only at a speed below 40 km/h.

Combining the values from the comparison of the capabilities of different network generations by Myhrvold [Myh23] with the upload speed stated by den Ouden et al. [OHS+22] as an example, reinforces the estimation that a remote control system could be used on 4G, but

Table 6.1. Comparison of the capabilities of different generations of networks, based on [Myh23]

Technology	3G	3,5G	4G	4G+	5G
Max downlink speed	3 Mb/s	40 Mb/s	100-150 Mb/s	1 Gb/s	20 Gb/s
Average downlink speed	1 Mb/s	4 Mb/s	10 Mb/s	30 Mb/s	260 Mb/s
Max uplink speed	500 Kb/s	11 Mb/s	50 Mb/s	1 Gb/s	10 Gb/s
Average uplink speed	100 Kb/s	500 Kb/s	3 Mb/s	8 Mb/s	34,97 Mb/s
Meant for	SD video streaming	SD video streaming	HD video streaming	1080p video streaming	4K video streaming

not in a way that meets the reliability requirements fully. The average uplink speed for 4G in the work of Myhrvold is at 3 Mb/s and for 5G at 34.97 Mb/s. The minimal requirement for upload speed in the system by den Ouden et al. [OHS+22], is at a similar value to the average value of uplink speed of 4G based on the comparison in Table 6.1. At average, 4G barely meets the requirements for this example, but the values are under the required minimum for at least some of the time. This makes 4G not a reliable option for this example. 4G+ and 5G however meet the requirements for uplink speed for this example at least.

Another interesting point is the network protocol used for the connection. While TCP ensures stability and structure in the stream, UDP sacrifices this for more speed in transmitting data. The current version of the system utilizes a TCP stream, but since this protocol may struggle with bandwidth on long distance network paths [Myh23], UDP might prove to be the more efficient protocol for further testing.

The remote control system presented in this thesis is not tested for the specific uplink speed needed and currently runs on a Wi-Fi network. As an estimation based on the example values, the system might be viable on a LTE network with alterations or constraints for the user. The frame rate of the transmission can be lowered, as well as the resolution of the livestream. A protocol for a dynamic solution for the resolution according to the upload speed would be another approach. The maximal driving speed allowed for a user can also be capped at a lower level, as a lower driving speed can make a higher delay feasible for a remote control system, The slower speed decreases the distance before standstill [OHS+22].

While these constraints could make the system viable on LTE this remains to be tested and a 5G network is most likely the network quality needed for running this system reliably.

6.2 Current Network

For the current use case, REAKTOR, the quality of the network currently available on the track between Malente and Lütjenburg is the most relevant. The system is not tested on the track currently, but this is the next logical step and the network analysis in this chapter supports this step.

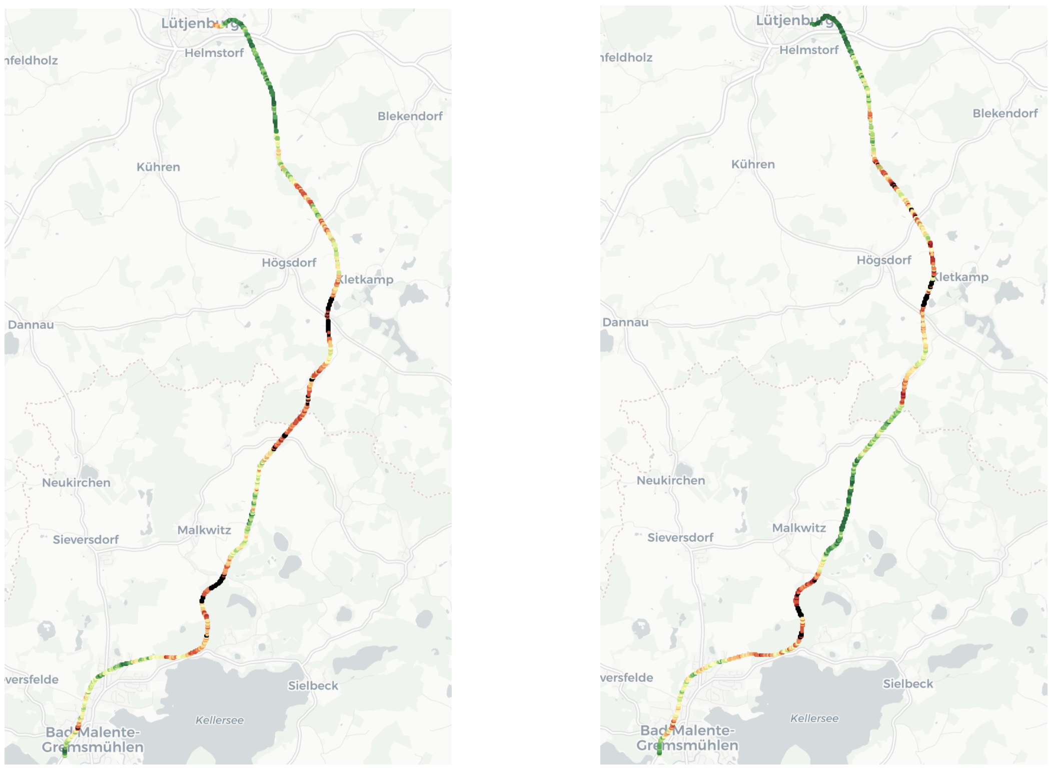


Figure 6.1. Channel quality of different operators on the track between Malente and Lütjenburg, Figure credit: Birkan Denizer, CAU

An analysis of the channel quality for two different operators in the real-live laboratory on the track between Malente and Lütjenburg can be seen in Figure 6.1. On the parts that are marked green on the track, the channel quality is the best, while red and black colours mean the worst signal. Birkan Denizer from CAU conducted the analysis and shared the network coverage percentages for the track as well. The 5G coverage on the full track is between 60 % and 65 %. The coverage of 5G and 4G networks combined reaches a coverage between 97.7 % and 99.8 % of the track, depending on the operator.

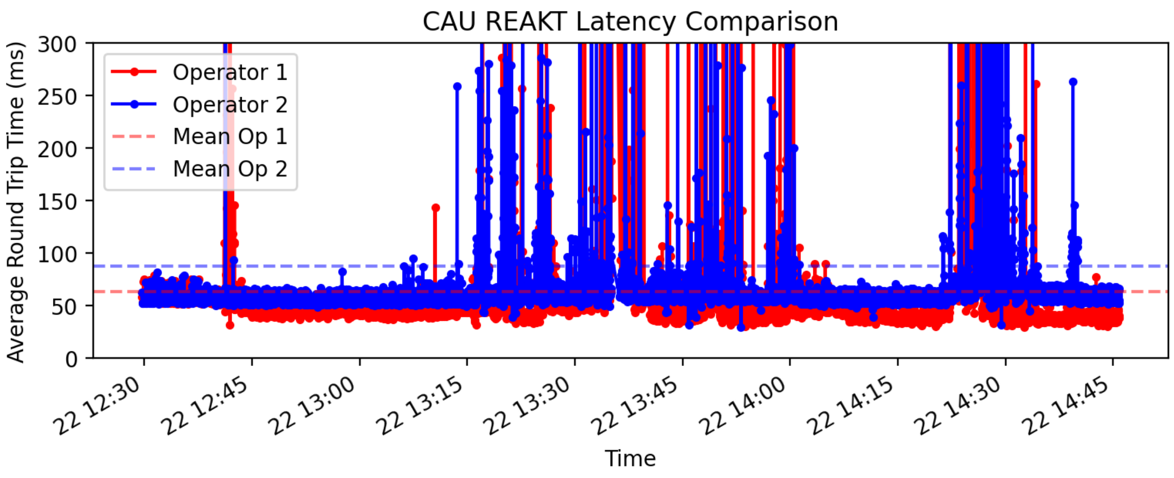


Figure 6.2. Latency of the network for different operators on the track between Malente and Lütjenburg, Figure credit: Birkan Denizer, CAU

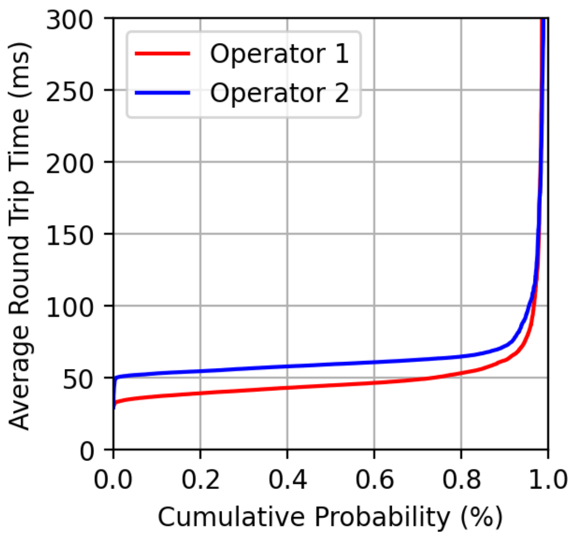


Figure 6.3. Cumulative probability for average RTT,
Figure credit: Birkan Denizer, CAU

With the delay for video streaming being the main problem for the applicability of the remote control system on the track, the latency of the network available on the track is critical for testing on the track. As can be seen in Figure 6.2, the latency on parts of the track has average values below 100 ms RTT, which are reasonable values for remote control as stated by den Ouden et al. [OHS+22]. On big parts of the track however, the latency values spike up, reaching values of over 1 second latency, which makes the system infeasible to use on the full track. Around the parts of the track near Malente, on the green to yellow parts near the middle of the track in Figure 6.1 and especially near Lütjenburg, the system could be tested for the REAKTOR project in the future. The analysis by Denizer shows that the average latency values for the full track are between 63 ms and 87 ms, depending on the operator. The latency is under 300 ms for 98 % of the time, and for 99.7 % of the track, the latency is below 1 s, as can partly be seen in Figure 6.3.

The work of den Ouden et al. [OHS+22] states that at a RTT greater than 300 ms, the remote driver experiences a loss of performance, and at 1 s delay, the system is not feasible to be used at all. Additionally, the full track would have to reach 100 % coverage for 4G and 5G combined at least, to be usable on the full track. The analysis by Denizer, combined with estimations by den Ouden et al., show potential for the system to be tested on specific parts of the track, but the system cannot be used on the full track reliably.

Chapter 7

Conclusion

In conclusion, the concepts for the remote control system are made to be applicable for different rail vehicles, while further testing must be done for a specific implementation. In the implementation in this thesis, the concepts are applied as far as possible, but concerning the network and scalability remains more testing to be done.

The first section of this chapter summarizes the work of this thesis, centring on the concepts. The implementation for the current use case and the assessment for the needed network are part of this summary as well. Following that, the next section discusses the possibilities and ideas for future work on the system. This includes the scalability considerations for the concepts and implementation.

7.1 Summary

The remote control system for rail vehicles that is presented in this thesis is designed to grant the user remote access to a rail vehicle, utilizing a cellular network. High flexibility in location and device is provided to the user via a UI on a Web Client. This is especially useful for testing features on the track during development stages of autonomous rail vehicles. For more dedicated usage of the system, a remote control panel is planned to provide optimized access to the user in later stages of automation. With a client-server-architecture as a base for the system, it is structured with a client on the user side, another client on the vehicle side, and a server to organize and forward the data between user and the vehicle. On the user side, the system can be used to monitor the ride of the vehicle through the constant livestream from a camera on board the vehicle. When remote control is being taken, the target speed and direction given by the user are sent in a continuous interval from the Web Client, through the server to the client on the vehicle. On the vehicle, the target speed is being forwarded to the controller to implement, and current speed and direction are sent back in the same continuous interval to the server and is provided to the user in the UI, while ensuring safety and functionality of the system in the process. The livestream from the camera is sent from the vehicle to the user in parallel. Designed to be deployed externally, the server side of the system is portable via a Docker container.

A basic assessment of network requirements for reliable application of the system on the track is being done in this thesis and the network available on the track for the current use case is also discussed in that context, on the base of an analysis by Birkan Denizer, from CAU. 5G being the suggested network generation for the remote control system, LTE might be feasible with constraints in streaming quality and a capped driving speed for the user. Concluding this thesis, a variety of paths for future work and testing on the system is being discussed.

7.2 Future Work

As the remote control system has only been tested in a laboratory environment with a 1:32 demonstrator for a rail vehicle, some of the key aspects for the system remain not sufficiently tested. For full functionality, additional work and testing needs to be done in the future.

7.2.1 Improved UI

On the user side, the UI can be improved in the future by including additional input for the user in the form of data from additional sensors. An example would be the integration of data from a radar sensor or lidar sensor into the UI. This features are highly dependable on the equipment of the rail vehicle. Additional controls for specific rail vehicles can be added to the UI via buttons, but adding controls to the UI leads to less space for the existing features. Thus, for more complex controls for rail vehicles, the construction of a remote control panel leads to a more effective use of space in the UI, as the controls can be physical components, independent of the space on the monitor. The remote control panel also provides the possibility of integrating other features, such as a haptic system, giving the user potential vibrotactile feedback for speed controls or alerting for danger [MRB25].

For an improved visual experience, a roadmap with the track and Global Positioning System (GPS) trackers for vehicles would be a great feature that could be integrated into the UI if provided by the rail vehicles. Other vehicles and points of interest could be marked on the map, giving the user valuable real time information on the track. This information could also be shown on a secondary monitor, integrated in the design in section 4.4.1.

The UI for the Web Client and the remote control panel would profit from the integration of the AI-based obstacle detection if possible on the network. Beside the visual approach, information on the vehicles and the track could also be given auditively. The current UI on the Web Client was chosen for the flexibility on location and devices, but does not change visually depending on the actual device of the user currently, so a responsive design for the Web Client is another goal for future work. Functionality of the system can and should also be improved, by implementing the website dynamically. The system currently refreshes the page in a continuous interval, but the display for current speed and target speed should update on change separately.

7.2.2 System Scalability

Implementing the scalability considerations for the current implementation as described in the concepts chapter, are part of future work on the system. This includes constraints in the implementation to forbid multiple users from accessing the same vehicle and integrating separate pages on the website for each of the potential vehicles integrated in the system. A server side way and a vehicle side way to integrate multiple cameras for a single vehicle was discussed earlier and in case a rail vehicle has additional controls, the data and interfaces for them need to be integrated on the server side as well. The `Heartbeat` function can be scaled up to hold additional values as well. It could be used to compare the value for target speed on the vehicle side with the value on the server side by adding it to the communication in both directions. The Python framework Flask has partly been chosen to accommodate for this scalability work in the future. The remote control server is designed to be deployed externally, but still has to be tested on different platforms.

7.2.3 Connectivity

In Chapter 6, 5G is suggested for the system, to ensure a reliable connection for livestreaming the video. A LTE network might be feasible as well, with a speed limit for the user and possible lower frame rate and resolution for the livestream. As the system currently only is tested on a Wi-Fi connection, part of future work on this system is testing it on different generations of the cellular network, to verify the assessment. Following that, the system needs to be tested on a real track. While doing that, TCP, UDP and possibly other protocols need to be tested and compared for performance in the livestream, to assess which is the best for this system. Focus for the overall performance should be the delay of the video stream for the user, which should be minimal.

Bibliography

[Ahv16] Sauli Ahvenjärvi. “The human element and autonomous ships”. In: *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation* 10.3 (2016), pp. 517–521.

[BNS+23] Andre Bröring, Arne Neumann, Andreas Schmelter, and Jürgen Jasperneite. “A communication concept using 5g for the automated driving monorail vehicle monocab”. In: (2023). DOI: http://dx.doi.org/10.25673/111745.

[CM23] Baris Cogan and Birgit Milius. “Remote control concept for automated trains as a fallback system: needs and preferences of future operators”. In: *Smart and Resilient Transportation* 5.2 (2023), pp. 50–69.

[FWH+24] Achim Fiack, Franziska Weller, Moritz Heimes, and Thomas Laux. “Digitale schiene deutschland-zukunftstechnologien für das bahnsystem”. In: *Eisenbahn Ingenieur Kompendium* (2024), pp. 198–208.

[HY12] Soyoung Hwang and Donghui Yu. “Remote monitoring and controlling system based on zigbee networks”. In: *International Journal of Software Engineering and Its Applications* 6.3 (2012), pp. 35–42.

[LBP+06] T Luke, Nikki Brook-Carter, Andrew M Parkes, E Grimes, and A Mills. “An investigation of train driver visual strategies”. In: *Cognition, Technology & Work* 8 (2006), pp. 15–29.

[LKD+17] Ruilin Liu, Daehan Kwak, Srinivas Devarakonda, Kostas Bekris, and Liviu Iftode. “Investigating remote driving over the lte network”. In: *Proceedings of the 9th international conference on automotive user interfaces and interactive vehicular applications*. 2017, pp. 264–269.

[MKV+24] Sudha Ellison Mathe, Hari Kishan Kondaveeti, Suseela Vappangi, Sunny Dayal Vanambathina, and Nandeesh Kumar Kumaravelu. “A comprehensive review on applications of raspberry pi”. In: *Computer Science Review* 52 (2024), p. 100636. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2024.100636. URL: https://www.sciencedirect.com/science/article/pii/S1574013724000200.

[MRB25] Emelyne Michel, Philippe Richard, and Quentin Berdal. “Remote control desk in industry 4.0 for train driver: an ergonomics perspective”. In: *Procedia Computer Science* 253 (2025), pp. 1045–1054.

[Myh23] Øyvind Thingstad Myhrvold. “Remote controlled train operation: an analysis of remote driving technology and a possible pilot project in norway”. MA thesis. NTNU, 2023. URL: https://hdl.handle.net/11250/3091468.

[OHS+22] Jos den Ouden, Victor Ho, Tijs van der Smagt, Geerd Kakes, Simon Rommel, Igor Passchier, Jakub Juza, and Idelfonso Tafur Monroy. "Design and evaluation of remote driving architecture on 4g and 5g mobile networks". In: *Frontiers in Future Transportation* 2 (2022), p. 801567.

[PHL22] Jan Peleska, Anne E Haxthausen, and Thierry Lecomte. "Standardisation considerations for autonomous train control". In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. 2022, pp. 286–307.

[ST24] Nikita Smirnov and Sven Tomforde. "Exploring the dynamics of data transmission in 5g networks: a conceptual analysis". In: *arXiv preprint arXiv:2404.16508* (2024).

[TBB+21] Abhimanyu Tonk, Abderraouf Boussif, Julie Beugin, and Simon Collart-Dutilleul. "Towards a specified operational design domain for a safe remote driving of trains". In: *Proceedings of the 31st European Safety and Reliability Conference, Angers, France*. 2021, pp. 19–23.

[WKK+12] Joachim Winter, Jens König, Gerhard Kopp, and Holger Dittus. "Next generation train. the revolution". In: *Railvolution* 4 (2012), pp. 28–37.