An Autonomous Train Controller with Risk Analysis using System-Theoretic Process Analysis

Rasmus Niels Janssen

Bachelor 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
M.Sc. Jette Petzold

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

The REAKT project is an initiative aimed at developing new mobility concepts for public transport in rural areas by reactivating old shut-down railway tracks. The project aims to build small, autonomous, on-demand train carriages that can be called via an app. To be able to have multiple trains drive on the same route, which are mostly single-track lines, Single-Track Transfer Traffic is proposed. During this, trains safely dock together anywhere on the track to exchange passengers, thereby allowing multiple vehicles on the same track.

This thesis is part of the REAKTOR student project, which is a first attempt to develop concepts and implementations to make an autonomous vehicle function as proposed and consist of multiple theses. In this thesis, an autonomous train controller is developed which is able to control a train model and is able to transport passengers safely to their destination at a reasonable pace. To ensure the controllers safety a System-Theoretic Process Analysis (STPA) is done, which is a risk analysis technique developed by Leveson and based on system theory. A Safe Behavior Model (SBM) is generated based on the STPA, to ensure safe behavior of the controller and thus safe behavior of the train. Using the SBM and a control structure modeled in the STPA an implementation of the autonomous controller for different test environments is presented. Further implemented is the interface of the controller to other theses of REAKTOR.

In addition to the controller implementation, a concept for Single-Track Transfer Traffic is proposed and a separate STPA is build, analyzing the risks of the concept.

Acknowledgements

First of all, I want to thank Prof. Dr. Reinhard von Hanxleden for the opportunity to write this thesis and be part of the REAKTOR project.

I would further like to thank the entire Real-Time and Embedded Systems group for their friendliness and the constructive and helpful feedback I received during this bachelor's thesis. In particular, I want to thank my advisors Dr.-Ing. Alexander Schulz-Rosengarten and M. Sc. Jette Petzhold for their continuous guidance and help throughout the research and writing process.

Furthermore, I want to express my thanks to my fellow students in the REAKTOR student project, for making this whole process a fun, interesting and valuable experience.

Lastly, I want to express my utmost gratitude to my family, who always supported me throughout my studies and helped whenever they could.

Contents

List of Figures

Chapter 1

Introduction

Climate change is one of the biggest and hardest problem the current and future generations have to solve. Reducing the carbon-dioxide footprint of humanity is one of the ways to combat climate change that can be applied to almost every aspect of society. One of these aspects is mobility. In Germany, a mobility transition is happening. In parts, it is trying to shift from cars with combustion engines to electric vehicles. Another idea is trying to reduce the amount of cars in general by expanding and developing public transport, which generates less CO_2 per person[1]. Although expanding public transport in urban areas with a lot of commutes and making it more accessible is a feasible option, the same approach is not feasible in rural areas with a much lower daily demand. On top of that, there is already an existing shortage of skilled workers in Germany that does not allow increased train throughput. However, not all is negative. Since 1990, many rural railway tracks were shut down in Germany but not removed [2], meaning infrastructure for trains still exists, that public transport initiatives can reactivate and use.

The REAKT project[3] is such an initiative to develop new mobility in rural areas by reactivating these tracks. The goal is to develop small autonomous passenger vessels that can be called on-demand via an app. Most of the shutdown tracks are single-track lines, but having only one train for the track will be problematic, resulting in long waiting times for passengers if multiple demand it at the same time at different parts of the track. To be able to have multiple trains drive simultaneously on one track, Single-Track Transfer Traffic is proposed. During this, the trains shall safely dock together at any point on the track, exchanging passengers. Once all passengers are transferred to their respective vehicle, they reverse directions and take the exchanged passengers to their destination. The project has access to a 17-kilometer-long, shut-down railway track between Malente and Lütjenburg that is used for researching and testing these concepts[4]. By developing small autonomous on-demand vehicles, this

[1] https://www.de-hub.de/en/blog/post/mobility-turnaround-in-germany-less-traffic-more-networking/

[2] https://www.eba.bund.de/DE/Themen/Stilllegung/ListenStatistiken/listenstatistiken_node.html;jsessionid=B273C648819AE255A1CE9898CCFE20BE.live21301

[3] reakt.sh

[4] https://www.schiene-m-l.de/

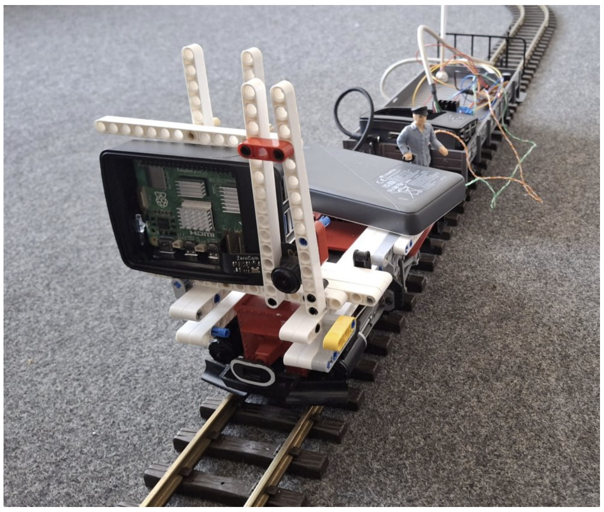


Figure 1.1. 1:32 REAKTOR prototype

project avoids the problems of lower demand and skilled worker shortages while simultaneously making it accessible every time of the day at any point of the track.

This bachelor's thesis is part of the REAKTOR project, a student project aiming to make an autonomous vehicle function as proposed. The student project consists of multiple other bachelor's and master's theses that all interact with each other to create a concept and the first implementation of a working autonomous train. To test this project, a 1:32 scale prototype, as can be seen in Figure 1.1, and a digital twin of the Malente-Lütjenburg railway track are used. In addition, a full-scale prototype is developed by the Real-Time Systems and Embedded Systems work group.

1.1 Problem Statement

This bachelor's thesis builds a safe autonomous train controller. The controller must be able to take control of the train, take input from various other components, and give accordingly output to the physical units of the train. It must be scalable to work on all three proposed testing environments, and more. The controller must engage in a safe docking procedure with another train for the Single-Track Transfer Traffic, when commanded to do so. To ensure safety, a risk analysis using System-Theoretic Process Analysis (STPA) is used to identify potential losses and hazards and minimize their risk. Further a Safe Behavior Model (SBM) based on the analysis is generated, so that the identified unsafe behavior does not occur [PH25].

1.2 Outline

The next chapter introduces the foundation of this thesis, such as the connection to the other theses of REAKTOR, the process of STPA and other technologies and programs used. Chapter 3 reviews related work on autonomous train vehicles and existing projects for on-demand public transport. In Chapter 4, the STPA for the train is performed, as well as an separate STPA for the safe docking procedure. Chapter 5 covers the Safe Behavior Model, which will be generated based of the STPA. In Chapter 6, the implementation of the controller in Python is described. This includes a deeper look into the implementation of the modulation of the control structure and the interface to the other thesis in the REAKTOR student project. Chapter 7 then evaluates the functionality of the controller to control a train, the functionality of the interface to the programs developed in other REAKTOR theses, and the scalability of the controller. In the last chapter, the thesis is concluded with a summary and future work.

Chapter 2

Preliminaries

This chapter introduces the technology concepts used in this bachelor's thesis. In Section 2.1, a deeper insight into STPA is given; Section 2.2 introduces the technologies used in this thesis. Lastly, Section 2.3 displays and explains the REAKTOR student project and how the theses are interconnected with each other.

2.1 System-Theoretic Process Analysis

System-Theoretic Process Analysis (STPA) is a relatively new risk analysis technique developed by Leveson and Thomas [LT18]. It is based on System-Theoretic Accident Model and Processes (STAMP) [Lev12]. STAMP itself is based on system theory, where the system is treated as a whole and not just as a result of many independent components. In system theory, properties emerge not only from the individual components but also from the interactions and relationships between them. STAMP is a model or a set of assumptions on how accidents occur based on system theory and is used by STPA as a foundation.

This foundation gives STPA an advantage compared to other more traditional risk analysis methods such as Fault Tree Analysis (FTA) [HRV+81], Failure Modes and Effects Criticality Analysis (FMECA) [BPR93], Event Tree Analysis (ETA) [II05], or Hazard and Operability Analysis (HAZOP) [Kle99], where hazards and risk arise only from failures of individual components. STPA is able to identify all the risk scenarios identified by these traditional techniques, as well as more previously undetected scenarios that arise from unsafe interactions between components. The results of STPA can be used to create new restrictions, evaluate design decisions, identify leading indicators for potential accidents, create test cases, etc. [LT18].STPA consists of four consecutive steps:

1. Define Purpose of the Analysis

2. Model the Control Structure

3. Identify Unsafe Control Actions

4. Identify Loss Scenarios

It can be iterated as often as modifications are necessary and can be applied during the whole development process.

2.1.1 Define Purpose of the Analysis

The first step of STPA is defining the purpose of the analysis. In this step, stakeholders, losses, and hazards are identified. Stakeholders are any kind of people that are involved in the system, such as users, the government, etc. Losses involve something that is of any value to a stakeholder. This can be grave losses such as the loss of life, but also less grave losses such as the loss of a mission or customer satisfaction.

After that, system-level hazards can be identified. Hazards are any kind of state in the system which, in a worst-case scenario, can lead to losses. With the hazards identified, System-level Constraint (SC) are created that specify conditions or behavior for the system, which prevents hazards. Each constraint should be linked to the hazard, it prevents. As an example, an automatic train door results in the following [Tho13]:
Losses:
L-1: Loss of life or injury
Hazards:
H-1: Doors close on a person in the doorway [L-1]
H-2: Doors open when the train is moving or not in a station [L-1]
H-3: People are unable to exit during an emergency [L-1]
System-level constraints:
SC-1: Doors must stay open if a person is in the doorway [H-1]
SC-2: Doors must stay closed when the train is moving or not in a station [H-2]
SC-3: Doors must be able to open during an emergency [H-3]

2.1.2 Model the Control Structure

In the second step, a control structure is built. It is an abstract model that defines system components and the interactions and relationships between them. This control structure is hierarchical and generally consists of the following elements: Controllers, control actions, feedback, other input or output from components that is neither a control action nor feedback, and controlled processes. The vertical structure represents control and authority. Components in higher layers have authority over those in lower layers. Higher components can send control actions down to components in lower layers. Lower-level components can give feedback to higher-level components about their current status. This creates a control-feedback loop between the controller and

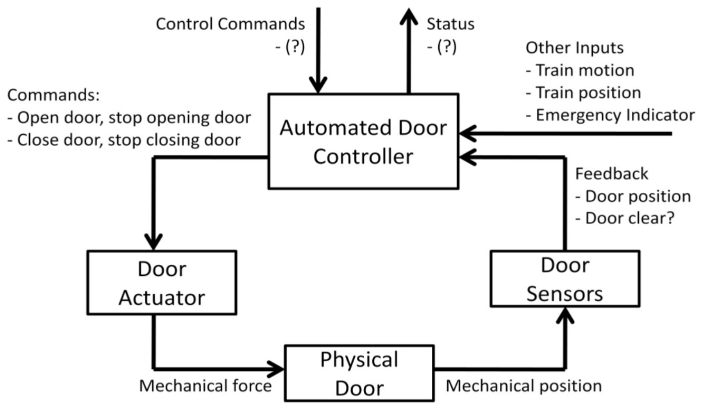


Figure 2.1. Example control structure of an automatic door [Tho13]

the controlled process. Further, controllers on the same level can communicate with each other outside of this control-feedback loop.

After modeling the control structure, responsibilities are assigned to the different components. These responsibilities are refined system-level constraints that specify what each component needs to do to fulfill the constraint.

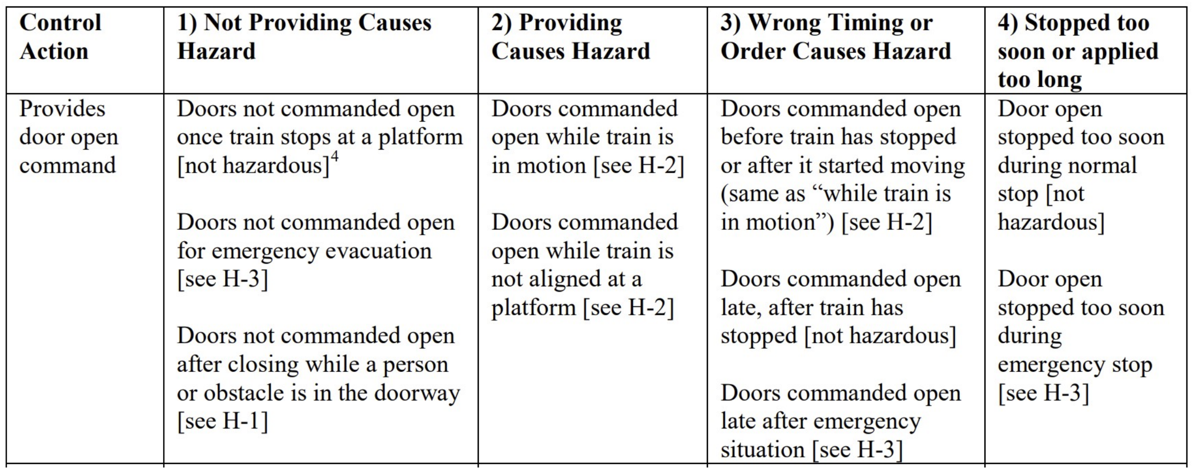
An example of a control structure for the automatic train door can be seen in Figure 2.1. In the figure, the *AutomatedDoorController* is able to send control actions to open/close the door to the *DoorActuator*, which then mechanically moves the *PhysicalDoor*. The position of the door can then be read by the *DoorSensors* that give feedback to the controller about the position of the doors and if it is clear. Thereby a control-feedback loop is constructed. A possible responsibility of the *DoorSensors* could then be:

R-1: If the door is not clear, it may not be closed [SC-2]

2.1.3 Identify Unsafe Control Actions

The third step is to identify UCA. An UCA consist of a control action, and a context in which it, becomes dangerous and leads to a hazard. There are four ways a UCA can occur:

- ▻ Not providing control action
- ▻ Providing control action
- ▻ Providing control action too early, too late, or in the wrong order



Control Action	1) Not Providing Causes Hazard	2) Providing Causes Hazard	3) Wrong Timing or Order Causes Hazard	4) Stopped too soon or applied too long
Provides door open command	Doors not commanded open once train stops at a platform [not hazardous][4] Doors not commanded open for emergency evacuation [see H-3] Doors not commanded open after closing while a person or obstacle is in the doorway [see H-1]	Doors commanded open while train is in motion [see H-2] Doors commanded open while train is not aligned at a platform [see H-2]	Doors commanded open before train has stopped or after it started moving (same as "while train is in motion") [see H-2] Doors commanded open late, after train has stopped [not hazardous] Doors commanded open late after emergency situation [see H-3]	Door open stopped too soon during normal stop [not hazardous] Door open stopped too soon during emergency stop [see H-3]

Figure 2.2. Example UCAs for the open door control action [Tho13]

▻ Providing control action for too long or stopping too soon

These types need to be considered for all control actions. It is possible to have multiple UCAs for the same type while not having any for another type. An example of UCAs for providing the open door command can be seen in Figure 2.2. The figure shows UCAs of all four types and in which hazards they result. A UCA for not opening the door is, for example, in the case of emergency evacuation.

Similarly to SCs, Controller Constraints (CCs) can be defined to specify the controller's behavior to prevent UCAs.

2.1.4 Identifying Loss Scenarios

The last step is to identify loss scenarios. Loss scenarios describe the real-life factors that lead to UCAs and hazards. Losses can be differentiated into two categories: scenarios that lead to UCAs and scenarios in which control actions are improperly executed.

For the first, scenarios can generally be identified by working backward from UCAs, and they can be further categorized into unsafe controller behavior and causes of inadequate feedback and information. There are four general reasons for controller malfunctions that lead to loss scenarios: physical controller failures, inadequate control algorithms, unsafe controller input, and an inadequate process model. To find causes of inadequate feedback and information, the source of the feedback needs to be examined.

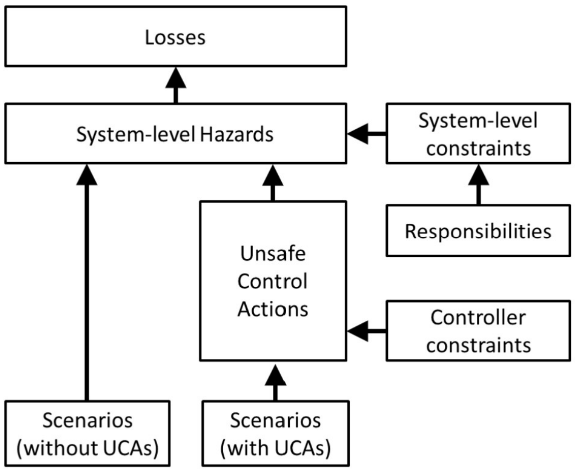


Figure 2.3. Traceability diagram for STPA [LT18]

The second type of scenario can also be divided into two types: scenarios involving the control path and scenarios related to the controlled process. The first generally identifies whether something is faulty between sending and receiving the control action. If this is not the case, the second type could apply, where control actions are transferred successfully but not executed effectively or are overridden. Some example scenarios for UCA-4 identified in Figure 2.2 are:

UCA-4: Doors commanded open while train is in motion [H-2]

Scenario 1 for UCA-4: While the train is in motion, the door actuators have a mechanical malfunction, causing the doors to open [UCA-4]. As a result, passengers may fall off the train. [H-2]

Scenario 2 for UCA-4: While the train is in motion, the automatic door controller receives inadequate feedback about the train motion, thinking that the train stands still [UCA-4]. As a result, passengers may fall off the train. [H-2]

2.1.5 STPA Outputs and Traceability

Figure 2.3 shows the traceability diagram of the STPA analysis, giving an overview of how the results of the STPA are connected to each other. The control structure is thereby closely connected to every STPA output, although it is not explicitly visualized. The results of the STPA can be used in various ways, such as finding design recommendations, driving new design decisions, creating requirements, evaluating existing designs, etc. [LT18].

2.2 Used Technologies

Multiple preexisting technologies are used for the STPA process and SBM generation and execution in this thesis.

2.2.1 KIELER

The KIELER project[1] is a research project about enhancing the graphical model-based design of complex software systems. It is developed by the Real-Time and Embedded System group at Kiel University. It researches among other things Sequentially Constructive Charts (SCCharts), which are a synchronous state chart dialect with sequentially constructive semantics. These SCCharts are used in this thesis to model a SBM, to ensure the safety of the autonomous controller.

2.2.2 Pragmatic Automated System-Theoretic Process Analysis

Pragmatic Automated System-Theoretic Process Analysis (PASTA)[2] is a tool implemented as a Visual Studio Code (VS Code) extension by Petzold et al. [PKH23]. PASTA provides a domain-specific-language for STPA and provides the user with validity checks and automation. The main advantage of PASTA in comparison to other tools is the visualization of STPA. The visualization is split into two graphs: The control structure and a traceability diagram similar to that in Figure 2.3 [PKH23]. These graphs give an overview of relationships in the STPA and help to make them much more comprehensible. In addition, options are provided to hide specific subcomponents, adjust how they are visualized, or change the color style of the visualization.

PASTA also offers safe behavior model generation, creating SCCharts based on the STPA or for a specific controller in it, by generating Linear Temporal Logic (LTL) formulas out of UCAs. The generated SBM is not necessarily complete but provides a good foundation that covers safety as well as aliveness properties [PH25].

2.3 REAKTOR Student Project

This section presents the interconnections between the different theses of the REAKTOR student project and the connection of them to this thesis. A traceability diagram is depicted in Figure 2.4 giving an overview of the project and is further discussed in the next sections.

[1] `https://github.com/kieler`

[2] `https://marketplace.visualstudio.com/items?itemName=kieler.pasta`

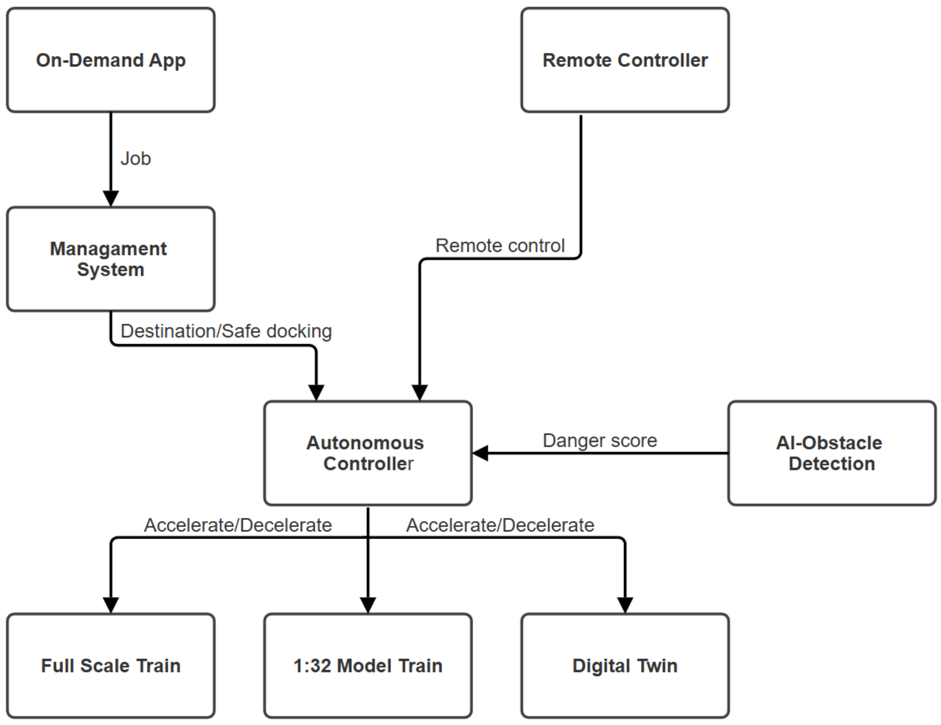


Figure 2.4. Traceability diagram of theses in the REAKTOR student project

2.3.1 On-Demand App

The on-demand app provides a user interface where passengers can call the train on demand to a position on the track, get in and then drive to their destination. After being used to call the train, the app sends a job with all the important details to the management system.

2.3.2 Management System

The management system is responsible for assigning the job from the app to the correct vehicle and giving the train controller the train's destination. It passes information about the train's status between it and the app user.

The management system is also responsible for identifying when Single-Track Transfer Traffic must be initiated between two trains and then providing the command to the train controller to safely dock together to exchange passengers.

2.3.3 Remote Controller

The remote controller can take over the train and control it remotely if necessary. This is useful in emergency situations or when train components fail. For this, the autonomous controller must be able to be switched on and off to allow the remote controller to have full control over the vehicle and to release the control so that the autonomous controller can continue after the problems have been resolved.

2.3.4 AI-Obstacle Detection

The AI obstacle detection identifies potential objects on the track using cameras and sensors. If any objects are identified on the track or close to it, a signal is given to the autonomous controller to inform it about the potential danger.

2.3.5 Physical Units

The physical units are the different test environments for the REAKTOR project. It includes a 1:32 model train with a Raspberry Pi, a digital twin, and in the future, a prototype of a full-scale version. The digital twin will be able to simulate multiple small vehicles on a digital recreation of the railway track between Malente and Lütjenburg. This recreation allows the autonomous controller to control the digital vehicles to simulate the real-world application of the project.

Chapter 3

Related Work

In Germany several other projects are working to revive train transport in rural areas of Germany. One of these projects is MONOCAB discussed in Section 3.1. In addition, there have been multiple other projects that have analyzed the safety of autonomous train controllers and proposed a design for commercial use. Some of them are introduced and discussed further in Section 3.2. Lastly, a comparison with the autonomous controller proposed in this thesis and the different approaches presented in this chapter is made in Section 3.3.

3.1 MONOCAB

MONOCAB[1] is a project developed by the OWL University of Applied Sciences and Arts, Hochschule Bielefeld (HSBI), and Frauenhofer IOSB-INA under the direction of Prof. Dr.-Ing. Thomas Schulte.

Similarly to the REAKT project, the objective of MONOCAB is to make public transport more accessible in rural areas by reactivating shut-down railway tracks. They develop small gyro-stabilized vehicles that are able to balance on just one rail of the traditional railway track, as shown in Figure 3.1. They are thin enough that two vehicles can pass each other on a single track line. These small vehicles are supposed to drive on-demand around the clock for 365 days a year and can hold between 4-6 passengers each.

The MONOCAB can operate completely autonomously using advanced environmental sensing systems, including radar and camera sensors, to ensure the safety of passengers. To ensure smooth operation, the MONOCAB is equipped with communication systems for vehicle-to-vehicle communication, as well as vehicle-to-infrastructure communication, enabling remote control and management of the MONOCAB fleet.

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

[2] https://www.monocab-owl.de/presse-downloads

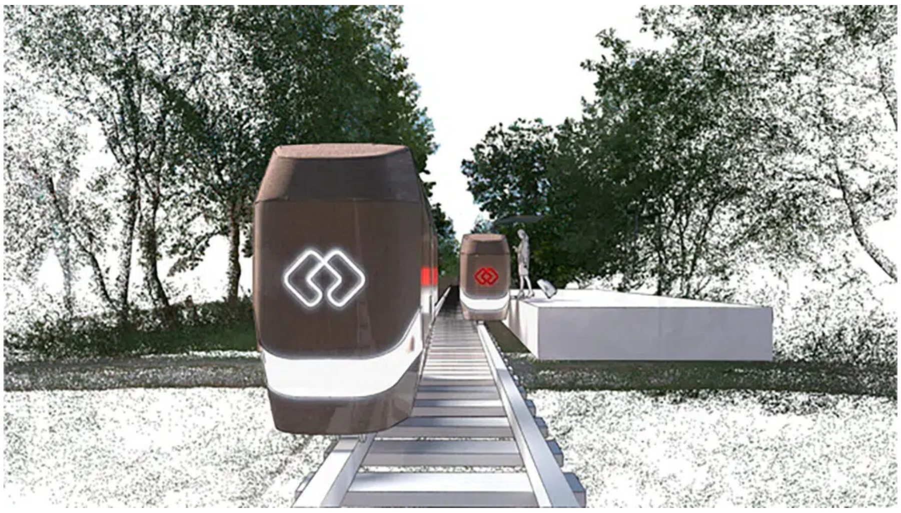


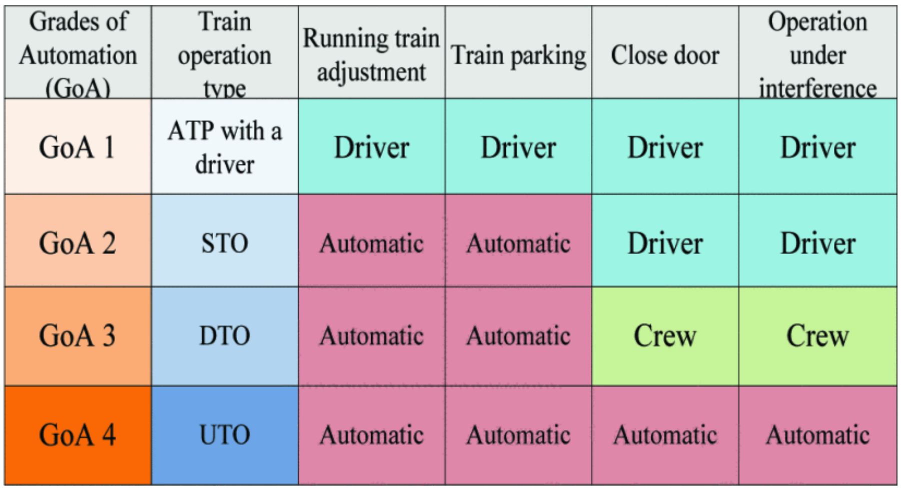
Figure 3.1. Two MONOCAB vehicles driving side by side[2].

3.2 Autonomous Train Concepts

There have already been multiple successfully implemented autonomous train systems, starting all the way in 1968 with the London Victoria Line. Although this underground line still required a driver to open and close doors, it was the first train to drive autonomously. Since then, many metros have used Autonomous Train Operators (ATOs), meaning a system responsible for operating trains without direct human intervention, with different amounts of automatization. To categorize this automatization for trains, Fei Yan et al. [YZT19] have proposed a Grade of Automation (GoA) system, depicting the different grades of automatization as shown in Figure 3.2. In the figure, STO stands for Semi-automated Train Operation, DTO stands for Driver-less Train Operation, and UTO stands for Unsupervised Train Operation. The London Victoria Line has a GoA of 2, since only driving and stopping are done automatically. Today's projects, such as REAKT or MONOCAB, aim to achieve a GoA of 4 for their vehicles.

With autonomous controllers trying to achieve higher GoAs the safety of these trains is also further discussed. Tonk et al. [TCB+23] proposed a methodology to be used to ensure the safety of autonomous trains. The methodology divides the train into three hierarchical system levels:

1. Overall System level
2. AI-Based Component level
3. AI/ML Software level



Grades of Automation (GoA)	Train operation type	Running train adjustment	Train parking	Close door	Operation under interference
GoA 1	ATP with a driver	Driver	Driver	Driver	Driver
GoA 2	STO	Automatic	Automatic	Driver	Driver
GoA 3	DTO	Automatic	Automatic	Crew	Crew
GoA 4	UTO	Automatic	Automatic	Automatic	Automatic

Figure 3.2. Grade of automation for train control system [YZT19]

The lowest level concerns the safety issues related to the learning-based software used in safety-related applications. The middle level concerns the risks emerging due to insufficient performance of the components. The highest level represents the autonomous train. It determines GoA, the specific environment of the train, and it identifies hazards related to the aspects of autonomy. To fulfill the challenges in this top level of the methodology, a safety analysis, similar to the one in this thesis, must be done, where especially the trains environment and the hazards are identified.

Trentesaux et al. [TDO+18] propose a way to slowly integrate automatization into existing train operation with an iterative design process to be able to test and validate intermediate stages of the process. They propose two different methods for this step.

The first method will progressively transfer driving tasks from the driver to the autonomous train while simultaneously progressively transferring supervision tasks to the driver. Thereby typically following the GoA levels.

The second method is to immediately transfer all driving tasks away from the driver, giving him only supervision tasks from the beginning. In this method, there are two approaches to implement the autonomous train. The first approach implements a simplified version of the complete decision-making and learning process, including everything, and then slowly improves on simple scenarios. After that, it improves on more complex ones until the train is working in the entire environment. The second approach consists of sequentially implementing each function of the decision progress of the train, only moving to the next one once the current one has been fully validated.

Trentesaux et al. [TDO+18] also propose a design approach for an autonomous train as shown in Figure 3.3. The train follows a predefined operation plan while

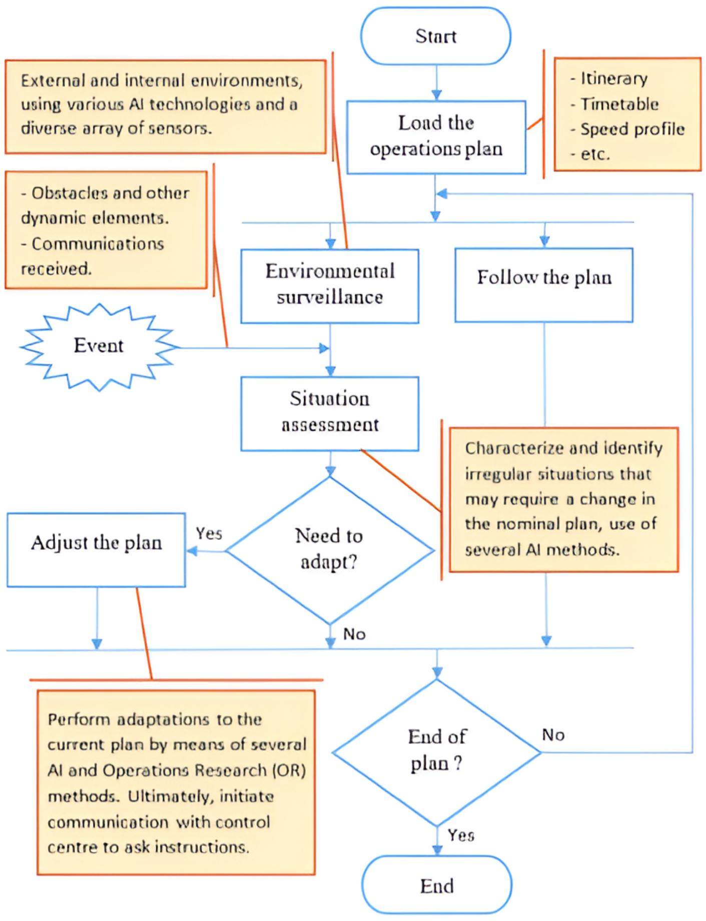

Start
Load the operations plan
- Itinerary
- Timetable
- Speed profile
- etc.
External and internal environments, using various AI technologies and a diverse array of sensors.
- Obstacles and other dynamic elements.
- Communications received.
Environmental surveillance
Follow the plan
Event
Situation assessment
Characterize and identify irregular situations that may require a change in the nominal plan, use of several AI methods.
Adjust the plan
Yes
Need to adapt?
No
Perform adaptations to the current plan by means of several AI and Operations Research (OR) methods. Ultimately, initiate communication with control centre to ask instructions.
End of plan ?
No
Yes
End

Figure 3.3. Autonomous train design approach by Trentsaux et al. [TDO+18].

constantly checking for environmental changes to which the train needs to adapt. If an unusual situation occurs, the train adapts its operating plan accordingly if necessary.

Peleska et al. [PHL22] propose another approach to an autonomous train controller with GoA 4, with a lot of design restrictions, such as vehicle-to-vehicle communication. The deliberately conservative architecture they propose serves as a thought experiment, whether such a GoA 4 system could be certified on the basics of the CENELEC standards [EN11; Kar24; Std19] and the ANSI/UL 4600 pre-standard [Koo22], being the first "fairly-complete" document addressing system-level safety of autonomous vehicles.

In addition, they propose an architecture with a kernel at its center. The architecture also includes multiple positioning systems, communication systems, juridical recording, and different obstacle detection methods such as cameras, radar, and

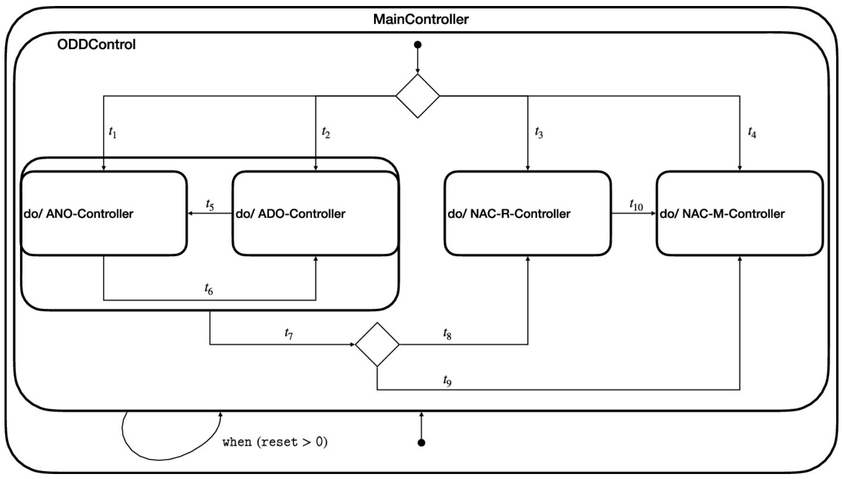


Figure 3.4. Operational modes of main controller [PHL22]

lasers. In addition, it consists of multiple supervision components securing passenger safety. The central kernel makes its autonomous decision based on the input of the other modules that are executed via a train interface module. The kernel consists of four operating modes, which can be seen in Figure 3.4. In the Autonomous Normal Operation (ANO) mode, the train is fully functional as an autonomous train, driving normally. In the Autonomous Degraded Operation (ADO) mode, the train is still driven autonomously by the controller, but with a reduced performance, for example, a lower speed. In case of failure, where the train can no longer be driven autonomously, the kernel switches to the Non-Autonomous Control (NAC) modes. In the NAC-R-Controller, the controller can be remotely controlled by a person, while for the NAC-M-Controller, a train driver must physically board the train and drive it manually. The approach is developed for freight trains and metros, while declaring itself infeasible for high-speed passenger trains, as existing obstacle detection can only be used reliably at speeds of less than 120 km/h.

3.3 Comparison

This thesis proposes a controller for autonomous trains with a hierarchical control structure. The main controller, being the decision making controller at the highest level of the structure. Similarly to the main controller proposed by Peleska et al. [PHL22], this thesis controller consists of multiple driving modes, depending on the outside environment. In this way, the train can operate autonomously with reduced speed, when some smaller obstacles are detected close to the track, without instantaneously needing human assistance. However, there is also the possibility of human interference in emergency situations, using the remote controller. The flow of the controller process is implemented similarly to the approach presented by Trentsaux et al. [TDO+18], but in a less abstract way. The difference from these systems is having a clearly ordered hierarchical control structure divided into different modules. The proposed one implements the trains ability to drive and navigate itself autonomously, but it is possible to add further subsystems that keep communicating with the controller to a minimum and fulfill their own responsibilities. A future controller for light could, for example, be responsible for everything regarding light and handle minor problems that occur. Communication to the train controller can then be kept minimal, making intervention only necessary when the light controller is in an emergency mode. The approach in this thesis also creates a controller that is easily scalable for different vehicles when combined with the other theses of the REAKTOR student projects. This way, the interface to the management system, remote controller and AI-obstacle detection does not need to change.

Chapter 4

System Theoretic Process Analysis

The goal of this thesis is to build an autonomous train controller. The highest priority for this is not only to build a controller that is able to drive a train but also to ensure maximum safety. To ensure the safety of the vehicle and passengers, it is imperative that a risk analysis is performed. The autonomous controller needs to know how it is supposed to react in every possible situation, as well as avoiding to do things that would fall under common sense of a human driver, such as accelerating while braking. For this, I performed an STPA, as described in Section 2.1.1 using PASTA. The analysis is divided into two use cases. A general STPA of the controller and its control of the train is performed in Section 4.1. A second STPA, for the use case scenario of Single-Track Transfer Traffic, where the trains safely dock together to exchange passengers on the track, is performed in Section 4.2. The reason for doing two separate STPAs is that the use cases are too different and analyzing both in the same analysis would drastically increase complexity and reduce clarity.

4.1 STPA for Autonomous Controller

I conducted an STPA for an autonomous train driving on a single-line track. The main goal is to drive safely and fulfill the following objectives:

- Drive from A to B
- Track its own position
- Accelerate gently
- Stop safely at its destination
- React accordingly to obstacles
- Remote control

With these basic goals for the train, I performed a STPA to ensure that the controller executes them safely.

4.1.1 Define Purpose of the Analysis

The first step is to identify the stakeholders. I considered the possible users and operators of the train and determined their stake in the system. The main stakeholders are passengers, the operating group, and the manufacturer. Passengers are interested in safe, swift, and efficient travel between two points. Operators and manufacturers are interested in satisfied customers. These stakes can be converted into losses, as seen in Listing 4.1.

```
Losses
L1 "Loss of life or injury"
L2 "Loss of operating components, vehicle or infrastructure"
L3 "Loss of Communication"
L4 "Loss of mission"
L5 "Loss of customer satisfaction"
```

Listing 4.1. STPA Losses for the autonomous controller

The losses are hierarchically ordered from worst to least worst. The worst possible loss for stakeholders is an accident that leads to injury or death. Other losses include costly damage to the incorporeal structure, the train losing its ability to communicate, or the train not being able to reach its destination.

Afterwords, the hazards leading to these losses are identified as shown in Listing 4.2.

```
Hazards
H1 "System integrity is lost" [L1,L2,L3,L4,L5]
H2 "Vehicle exceeds safe-operating envelope for its environment    (
    Speed, Lateral/Horizontal forces)" [L1,L2]{
    H2.1 "Vehicle exceeds safe speed limit of the track"
    H2.2 "Vehicle accelerates too fast"
    H2.3 "Vehicle decelerates too fast, while not performing a
        Emergency Stop"}
H3 "Vehicle comes too close to objects on track" [L1,L2]
H4 "Vehicle loses communication to infrastructure or other vehicles"
    [L3,L4]
H5 "Vehicle loses connection while being controlled by remote
    controller" [L1,L2,L3,L4]
H6 "Insufficient positional awareness of the vehicle on the track" [L1
    ,L2,L4]
```

Listing 4.2. Hazards of the autonomous controller

Hazards and losses are kept to a minimum, which is the common practice for STPAs. The first hazard describes any part of system integrity that is lost, meaning that one or multiple system components experience failure, which can lead to the whole system failing and there by endangering passengers and outsiders. The second hazard has been divided into *sub-hazards* for the different ways, the train risks losses when driving and stopping without constraints. *H*2.3 for example, can lead to passengers flying through the train when it decelerates as fast as possible. *H*4 is the risk of losses when communication fails. This includes communication to the management system, meaning that the train cannot receive more destinations to drive to, and it includes the risk of losses during Single-Track Transfer Traffic. For each hazard, a System-level Constraint (SC) is defined. The SC establishes basic conditions to prevent the hazard from occurring. As an example, the SC to prevent *H*2 and its *sub-hazards* can be seen in Listing 4.3. The SCs describe abstractly how these hazards can be prevented. For example, *H*2.3 can be prevented by stopping the train at an acceptable speed when not making an emergency stop. The full list of SCs can be found on `github`.

```
SystemConstraints
SC2 "Vehicle may not exceed the safe-operating envelope for its
    environment" [H2]{
    SC2.1 "Vehicle may not exceed the safe speed limit of the track" [
        H2.1]
    SC2.2 "Vehicle may not accelerate too fast" [H2.2]
    SC2.3 "Vehicle my not decccelerate too fast, while not performing a
         Emergency Stop" [H2.3]
}
```

Listing 4.3. System-level constraints for H2.3 of the autonomous controller

4.1.2 Modeling the Control Structure

With the losses and hazards identified, the next step is to model the control structure of the autonomous controller. The control structure is not an implementation, but a functional model of the controller implemented later. The modeled control structure for the controller is shown in Figure 4.1. The structure is divided into two sections: the off-board section and the on-board section.

The off-board section contains everything that is not physically on-board the train but interacting with it and includes the Remote Controller and the Management System of the trains. These components are implemented in other theses of the REAKTOR student project and need to have an interface to the autonomous controller.

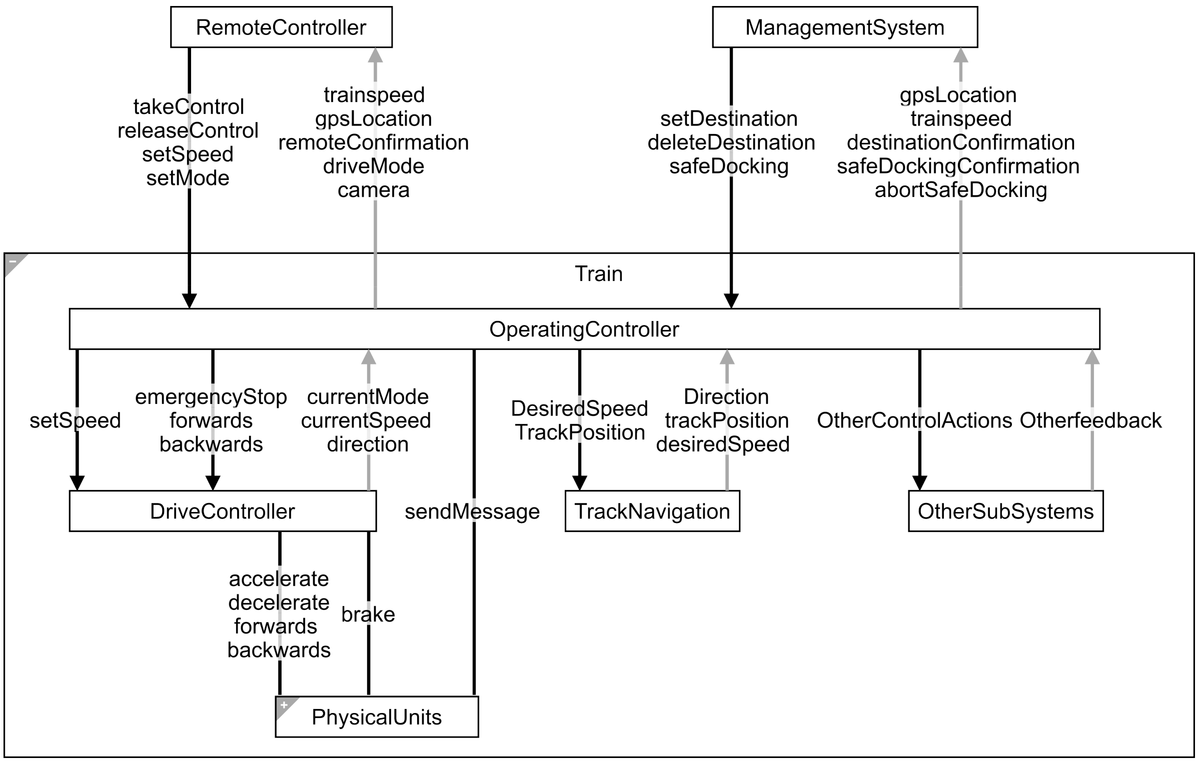


Figure 4.1. Control structure of the autonomous controller.

The remote controller must be able to take over the train remotely, control it, and then release it again. This is done in emergency situations where it is not possible for the autonomous controller to drive safely due to unknown factors. These factors could be objects on the track or system components that are failing. In this case, the human controlling the remote controller will have full authority over the train without the autonomous controller interfering. To overtake the train, the remote controller can send control actions to overtake the train. If this control action succeeds, the train sends a confirmation that the overtaking was successful. When controlling the train remotely, it has the possibility to adjust the speed and driving mode, which means the direction or an emergency stop. To successfully control the train, the remote controller receives feedback from the train about the train's current speed, location, and drive mode. It also receives a connection to the train's cameras for a live video broadcast of the track in front of the train, but no further information by the AI-obstacle detection or other subsystems supporting the autonomous driving of the train, as it is unknown how reliable their information are in emergency situations. The full responsibility lies within the decision making of the human driver steering the train remotely. If the remote control is finished, it can then release its control of the train and let the train drive autonomously again.

The management system must be able to set the destination of the train, finish or cancel it, and give the train commands related to safe docking, which will be further discussed in the STPA regarding the Single-Track Transfer Traffic in Section 4.2. To perform its duties, it has the corresponding control actions. In return, the management system receives information about the trains current track position and speed. It also receives confirmation of the control actions it has sent.

The on-board section contains everything that is physically on-board the train. At the lowest hierarchy level are the physical units. These include every physical controlled process on the train itself, e.g. the motor, brakes, communication systems, GPS sensors, cameras, etc. The physical units give feedback to the different higher-level components and only receive control actions to execute by them. The autonomous controller itself is divided into a module for track navigation, which allows for more modularity regarding information about the railway track and two controllers: *OperatingController* and *DriveController*.

Operating Controller

The operating controller is the decision-making controller. It receives input and feedback from several other components, such as AI-obstacle detection, drive controller, management system, and remote controller. Based on these inputs, it determines the behavior of the train. The process model of the operating controller is shown in Figure 4.2. The process model consists of a lot of variables needed by the operating controller to calculate the trains behavior. *mode* indicates the current mode of the operating controller, similar to the main controller proposed by Peleska et al. [PHL22]. At first, the controller is initializing itself and other train systems over which it has control. Then it switches between normal, degraded, and emergency mode, depending on the input it receives from AI-obstacle detection. In normal mode, the train drives normally; the degraded mode is a mode defined to ensure that the train can drive autonomously with caution when potential objects are close to the train. In this mode, the train drives at reduced speed, depending on the risk of danger the controller has received, to be able to stop quickly if an emergency occurs. If the controller switches to emergency mode, an emergency brake is immediately applied stopping the train. This is of course a drastic measure that is very uncomfortable for passengers and violates $H2$. Therefore, it must be ensured that the train only makes an emergency stop when it is really necessary. The toggleable variables *remotecontrol* and $safeDocking$ can be changed by the off-board section to put the train in a special mode for remote control and Single-Track Transfer Traffic. These variables are separate from *mode*, to prevent the possibility that the autonomous

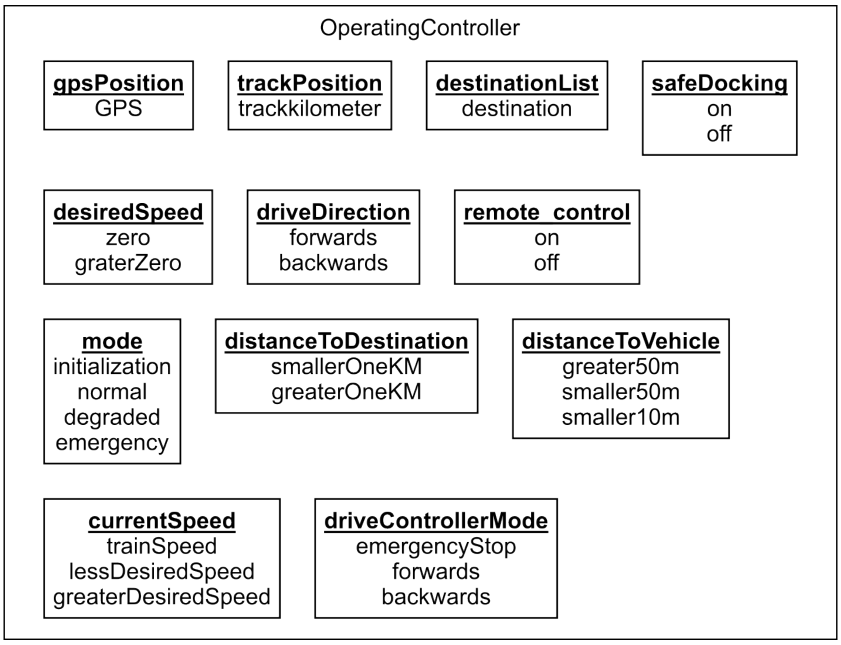


Figure 4.2. Process model of the operating controller.

controller can change them unintentionally. The process model also includes multiple variables that store feedback received from other components used to determine the controllers behavior. These include the current mode of the drive controller, the train's position, possibly measured using GPS, its destinations, current speed, and driving direction. In addition, it calculates the distance to the train destination to reduce the speed accordingly and stop gently at the destination, not violating $H2$. The operating controller can send *gpsPosition* to the *TrackNavigation* module where it is converted to track kilometers. Track kilometers ease calculations of distances on the tracks much easier than using raw GPS coordinates or other positioning technologies. With the help of this module, the train can also receive the desired speed it needs to drive depending on its operating mode.

The operating controller itself does not directly command the motor to accelerate, decelerate, etc. Instead, it sets the speed and driving mode of the second controller: the drive controller.

Drive Controller

The drive controller is the controller that directly controls the physical motor and brakes. The drive controller has variables storing the current speed of the train received as feedback from the physical parts, and has three different operating modes:

driving forward, driving backward, and emergency stop. This mode is set directly by the operating controller. The drive controller also receives the desired speed the train should accelerate/decelerate to. It is the drive controllers responsibility to slowly reach this desired speed. For example, if the train is driving with 30 km/h forward and receives a desired speed of 70 km/h from the operating controller, the train should not try to accelerate to 70 km/h as fast as possible, but at a reasonable pace. The drive controller is also responsible for ensuring that there are no sudden changes of the driving directions and that the train is fully stopped before changing directions. A normal, non-emergency stop is made, by decelerating to 0 km/h.

The drive controller is also responsible for converting the speed from km/h into the corresponding value the physical unit can process. For example, km/h must be converted into PWM for the 1:32 REAKTOR prototype. The drive controller will later be implemented as SBM, which is further discussed in Chapter 5.

Track Navigation

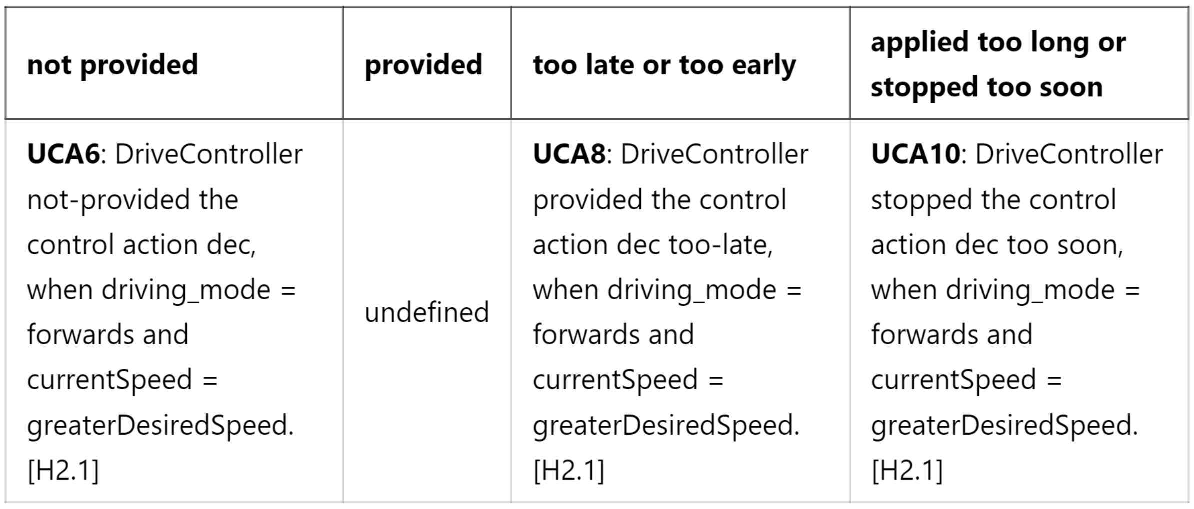
The module *TrackNavigation* is responsible for all the information on the physical track the train is driving on. This unit stores information about the length of the track and divides it into intervals of speed limit, telling the operating controller how fast the train can drive. This module is also responsible for converting the GPS position into the corresponding position on the track as a track kilometer and gives the operating controller the direction in which it needs to drive to reach its destination.

Other Subsystems

Other subsystems include systems on the train with which the operating controller needs to interact. As of know, this includes the AI-obstacle detection, but it can also be expanded with future sub-systems in the train, such as a possible door controller or lights controller.

The last step in modeling the control structure is to assign refined SCs as responsibilities to the different controllers. All responsibilities are publicly available on `github`, but an example of a responsibility for the operating controller implementing SC 2.1 is:

```
R1 "Give control action to reduce speed, if vehicle goes over desired
    speed, while not being remote controlled."[SC2.1]
```



not provided	provided	too late or too early	applied too long or stopped too soon
UCA6: DriveController not-provided the control action dec, when driving_mode = forwards and currentSpeed = greaterDesiredSpeed. [H2.1]	undefined	**UCA8**: DriveController provided the control action dec too-late, when driving_mode = forwards and currentSpeed = greaterDesiredSpeed. [H2.1]	**UCA10**: DriveController stopped the control action dec too soon, when driving_mode = forwards and currentSpeed = greaterDesiredSpeed. [H2.1]

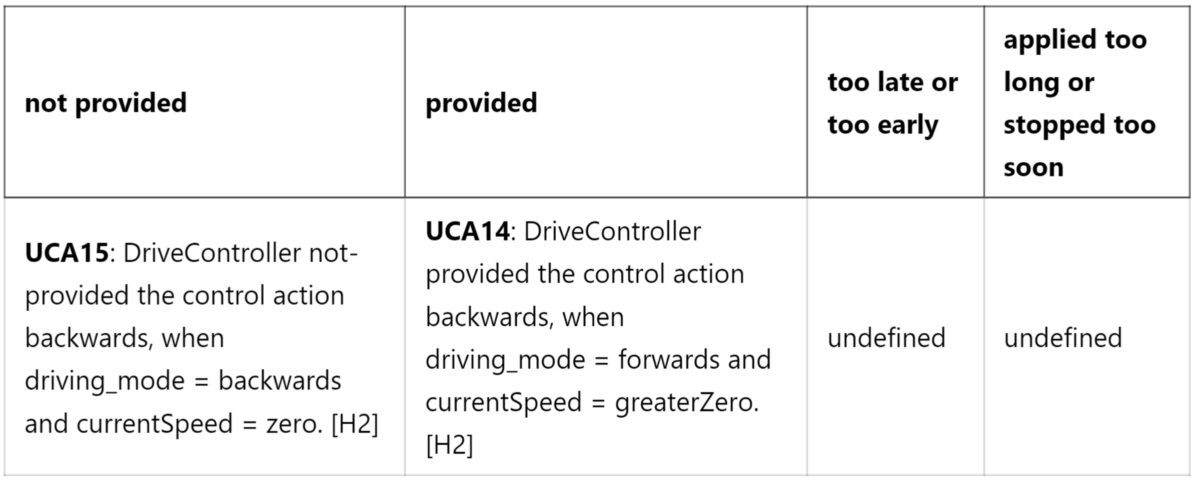
Figure 4.3. UCAs of Drive Controller for decelerating.

4.1.3 Identifying Unsafe Control Actions

In the third step Unsafe Control Actions (UCAs) are identified. For this, every single control action in the control structure is examined and possible UCAs are identified in the different ways they can occur. The UCAs have been implemented in PASTA via context tables, which contain concrete values of the process model variables, which lead to the control action being an UCA. These are also needed to generate a SBM for the drive controller based on the STPA. The UCAs for being in the *normal* or *degraded* operating mode and for driving backward and forward are identical, except for this variable. This is because the responsibilities of the train do not change based on the direction it drives and the desired speed is adjusted in the *degraded* mode, meaning that the train should also not drive above this adjusted speed then. Therefore, I will only show the UCAs for the normal mode while driving forward. The complete list of UCAs can be found on `github`.

Here, I will only show a few examples of UCAs. The first are UCAs that directly influence the motor and brakes of the train, i.e. the control actions provided by the drive controller. In Figure 4.3 the UCAs for the drive controller decelerating are shown.

The control action to decelerate contains multiple UCAs. UCAs have been identified for not decelerating, decelerating too late, or stopping too soon. All these cases happen if the train is driving faster than desired. These UCAs make it, so that the train continues to drive too fast, putting both the train and the passengers at risk. There has not been an UCA identified for when deceleration is provided, as it may be uncomfortable and unnecessary at certain times, but it is not hazardous or unsafe.



not provided	**provided**	**too late or too early**	**applied too long or stopped too soon**
UCA15: DriveController not-provided the control action backwards, when driving_mode = backwards and currentSpeed = zero. [H2]	**UCA14**: DriveController provided the control action backwards, when driving_mode = forwards and currentSpeed = greaterZero. [H2]	undefined	undefined

Figure 4.4. UCAs for drive controller changing driving direction to backwards.

For acceleration, the identified UCAs belong to the opposite UCA types, compared to deceleration. There have been identified multiple UCAs for providing acceleration, such as accelerating during an emergency stop or accelerating when the train has already reached its desired speed. On the contrary, there have not been defined any UCAs for not accelerating, accelerating too late, etc. For example, if the train's desired speed is 50 km/h, but it only drives 30 km/h and does not accelerate, it is again not dangerous, but merely inconvenient.

More UCAs have been identified when changing directions. UCAs for changing the direction from forward to backward are shown in Figure 4.4. $UCA15$ shows, that it is dangerous not to change direction when told and the train is at a complete stop. This is because if the train was to accelerate again, it would be in the other direction then intended, leading to unknown consequences. Another UCA is changing directions, while the train has not reached a complete stop, which would lead to extremely harsh direction change and possible vehicle damage. Preventing this UCA prevents a possible cause of $H2$ in a way that extreme forces do not affect passengers. Further UCAs have been identified for the drive controller to ensure that an emergency stop is taken immediately when ordered and to ensure that the train arrives at a complete stop.

For the operating controller the identified UCAs for the control actions to the track navigation module are mainly specified to ensure that the train regularly updates its current position and asks for the speed limit it is allowed to drive. The tracking of the trains position is relevant in all situations, regardless of whether the train is driven autonomously or remotely. Requesting the speed limit is not necessary during remote control, as the controller should only be driven remotely in emergency situations, where it is unknown how reliable the onboard subsystems are. This model therefore assumes responsibility for a well-trained human driver to be able to access the situation accordingly. The operating controllers actions to the drive controller are used to set the speed of the train, the driving direction, and to initiate emergency brakes. For setting the speed, again it is not unsafe to have a speed below the speed limit, but it is undesired, as it means that the train needs much more time than necessary. However, it is unsafe to not reduce speed if the train goes above the speed limit. In addition, it is dangerous to set the speed above 0 km/h during the emergency mode, as this risks that the train does not stop as quickly as possible. The UCAs regarding the acceleration of the drive controller should be able to correct this UCA, but it is best to prevent it from possibly being able to happen.

Other UCAs include the controller not performing the actions to reduce the speed in time. UCAs identified for initiating an emergency stop happen, when the controller gives out the command while not in the correct mode or similarly, does not initiate an emergency stop, or does it too late, if one is needed. The full list of UCAs can be seen on `github`.

Any UCAs related to Single-Track Transfer Traffic are discussed in the separate STPA in Section 4.2. UCAs regarding to the management system and remote controller are only identified in a simplistic way, as their process models are not within the scope of this thesis.

Further this STPA includes Desired Control Actions (DCAs), which contrary to UCAs are control actions that are desired to occur. These are identified to ensure that the above-mentioned uncomfortable and inconvenient, but not unsafe actions can be prevented. For example, is it never unsafe that the train does not accelerate, even though its speed is below the limit, but it is desired to do so. Examples of DCAs are, accelerating when driving to slow, not decelerating when driving the speed limit, and not performing an emergency stop, when it is not necessary. These DCAs are also used, to generate the SBM discussed in Chapter 5.

Lastly, Controller Constraints (CCs) have been defined for each UCA identified, similar to the SCs defined in the first step. These CCs aim to constrain the corresponding control action, so that UCAs are prevented from occurring. An example of an CC for $UCA1$ is shown in Listing 4.2.

```
    ControllerConstraints
C1 "Drive Controller may not accelerate during an emergency stop" [UCA1]
```

4.1.4 Identifying Loss Scenarios

In the last step, this thesis defined loss scenarios. Each scenario is tied to a UCA and the respective hazard created. Some examples of scenarios for the drive controller can be seen in Listing 4.4.

```
LossScenarios
Scenario1 for UCA1
"An control action to accelerate has been delayed because of a failure in
    the system and is now send while performing an emergency stop."[H1,H3]
Scenario2 for UCA1
"The controller receives an action, setting the desired speed above 0 and
    thereby accelerating, before changing the drive mode from '
    EmergencyStop'."[H1]
Scenario3 for UCA1
"The operating controller has send an action to swtich mode to forward/
    backward and accelerate, but the action to switch mode has been
    delayed or is lost."[H1]
Scenario5 for UCA2
"The controller has received an incorrect current speed from the physical
    units, leading it to accelerate above the desired speed. The reason
    for this can be:
     - Measuring has been unprecise due bad calibration
     - Information about the speed has been delayed, causing the controller
          to recieve out of date speed values"[H2.1]
Scenario49 for UCA32
"Controller incorrectly believes, the drive controller is already driving
    forwards, causing it not to send the action to change."[H2]
```

Listing 4.4. Example loss scenarios

The first three scenarios describe a possible cause for the drive controller to accelerate during an emergency stop, namely a physical controller failure and unsafe controller input. *Scenario*5 describes a possible cause for the controller to accelerate above the desired speed and the last scenario shows that the operating controller does not change the direction in which the train is driving, because it thinks that the train is already driving this direction. This can happen due to an inadequate process model or control algorithm.

Scenarios usually have all kinds of different causes. Most controller scenarios are caused by unsafe controller input, delays, or incorrect feedback received. Unsafe inputs can be received from inadequate control algorithms that give an action that is

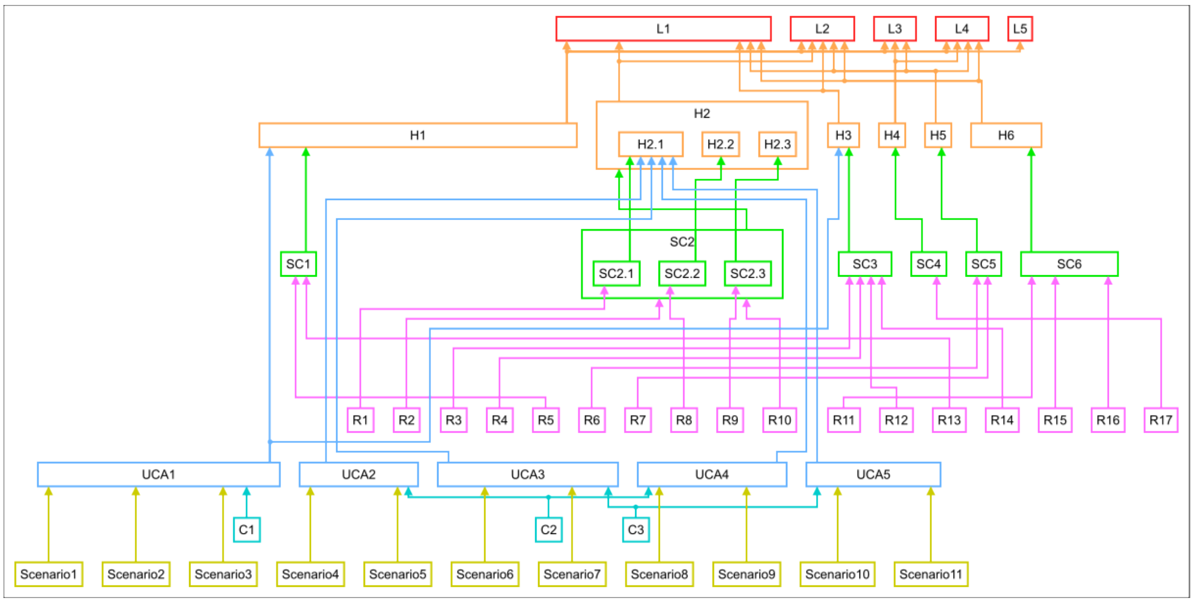


Figure 4.5. Diagram showing how the identified components of the STPA are connected.

generally unsafe, such as in *Scenario*2, where the drive controller receives the input to accelerate while still in emergency mode. With a correct control algorithm, the operating controller should never send out such an action but always switch the modes before accelerating. An example of a UCA resulting from delays can be seen in *Scenario*3. Here, the operating controller has a correct control algorithm and sends out the action to switch the mode of the drive controller before setting the desired speed. The problem arises because the first action is delayed and arrives after the second. An example of receiving incorrect feedback can be seen in *Scenario*5. Here, the measured speed of the train does not match the real speed. The reason for this could be many. The scenario mentions two possible reasons, poor calibration of the measuring unit and a delay on the feedback causing the received train speed to be out of date. Most other identified scenarios are usually also related to receiving incorrect feedback or unsafe controller input. This feedback and input is sent by inadequate control algorithms or controller failures. An overview how all the Scenarios, UCAs, hazards, etc. are connected to each other is shown in Figure 4.5. This figure only shows the UCAs regarding the drive controller's control action to accelerate. The complete diagram shows the connections between the components for all control actions.

4.2 Single-Track Transfer Traffic STPA

A second separate STPA has been conducted for the special case of Single-Track Transfer Traffic, where the trains must dock safely together, so that passengers can switch vehicles. The train's capabilities to drive during this procedure, has been discussed in the original STPA in Section 4.1 applies. This STPA identifies only the specific risks that apply during Single-Track Transfer Traffic in addition to the risks during normal driving.

The process proposed by this thesis for Single-Track Transfer Traffic is the following: The management system identifies when Single-Track Transfer Traffic is necessary and notifies both trains. The notification includes the means for the trains to establish a connection for communication with each other. Both trains need to confirm safe docking, switch into a safe docking mode, and establish communication between each other. The trains then regularly exchange their current position on the track as track kilometers and adjust their speed to the distance between them. At some point, the trains come too close for accurate positioning using GPS or something similar. At this point, the trains switch to distance sensors, which will measure the distance between the trains, using different sensors based on distance. The trains are steadily decreasing speed until they stop and make contact. Upon contact, different sensors confirm the successful maneuver. Lastly, before departing and breaking the connection, both trains need to switch destinations, as they resume driving in the direction they came from.

If one of the trains is remotely controlled, this thesis proposes two different concepts on how it should affect Single-Track Transfer Traffic. The first is simply that the remote driver performs the maneuver manually. This requires the driver to know that the train he steers is engaging in Single-Track Transfer Traffic. A second concept would need the train to abort safe docking when it is taken over remotely. Since remote takeover is only supposed to happen in emergency cases, it is likely that the train is already unable to dock safely against another train for unknown reasons. Therefore this STPA uses the second concept.

4.2.1 Define Purpose of the Analysis

The stakeholders and losses are unchanged from the general STPA discussed in Section 4.1.1 and the hazards identified there generally also apply during driving. Specific hazards related to Single-Track Transfer Traffic have been identified additionally in Listing 4.5

```
Hazards
```

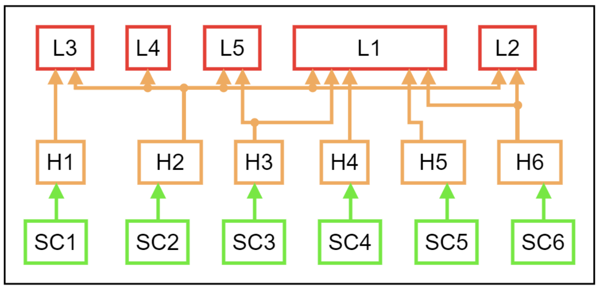


Figure 4.6. Relationship graph of Single-Track Transfer Traffic STPA.

```
H1 "Vehicle loses communication to other vehicle during safe-docking-
   procedure" [L3]
H2 "Vehicles dock with too much speed against each other" [L1,L2,L3,L4,L5]
H3 "Vehicle thinks it docked succesfully, while not being docked" [L1,L5]
H4 "Vehicles are not informed correctly that they need to dock" [L1]
H5 "One or both trains are unable to perform the maneuver"[L1]
H6 "Vehicles calculate incorrect distances between each other"[L1,L2]
```

Listing 4.5. Identified hazards related to safe docking of the trains.

All these identified hazards can be seen as *sub-hazards* of the original hazards in the first STPA. As an example, *H*1 is a specification of the original hazard: "Vehicle loses communication to infrastructure or other vehicles". If this hazard occurs, both trains are no longer able to get accurate information on the distance of the other vehicle. This leads to them unable to complete the procedure safely, as they cannot accurately calculate the distance between them, and thus decelerate to the correct speed. A direct consequence of this is in the worst case *H*2, where the trains crash. Further hazards can arise due to sensor failures or intern train problems discussed in the first STPA. For each hazard a SC is defined, stating constraints to prevent them. The connection between the losses, hazards and SCs can be seen as a relationship graph in Figure 4.6.

4.2.2 Model the Control Structure

For the control structure, the management system must be able to signal both trains that a Single-Track Transfer Traffic is initiated. Both trains must communicate with each other, passing different information needed for the docking. A minimized control structure can be seen in Figure 4.7. The control structure of each train is the same as in the first STPA, seen in Figure 4.2. However, in this one, there is a different focus on the physical units. The focus shifts away from the interaction with the motor, brakes, and sensors used for obstacle detection and instead focuses on sensors used

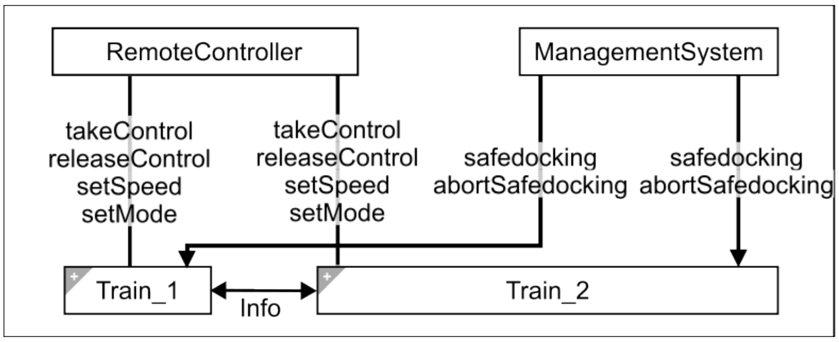


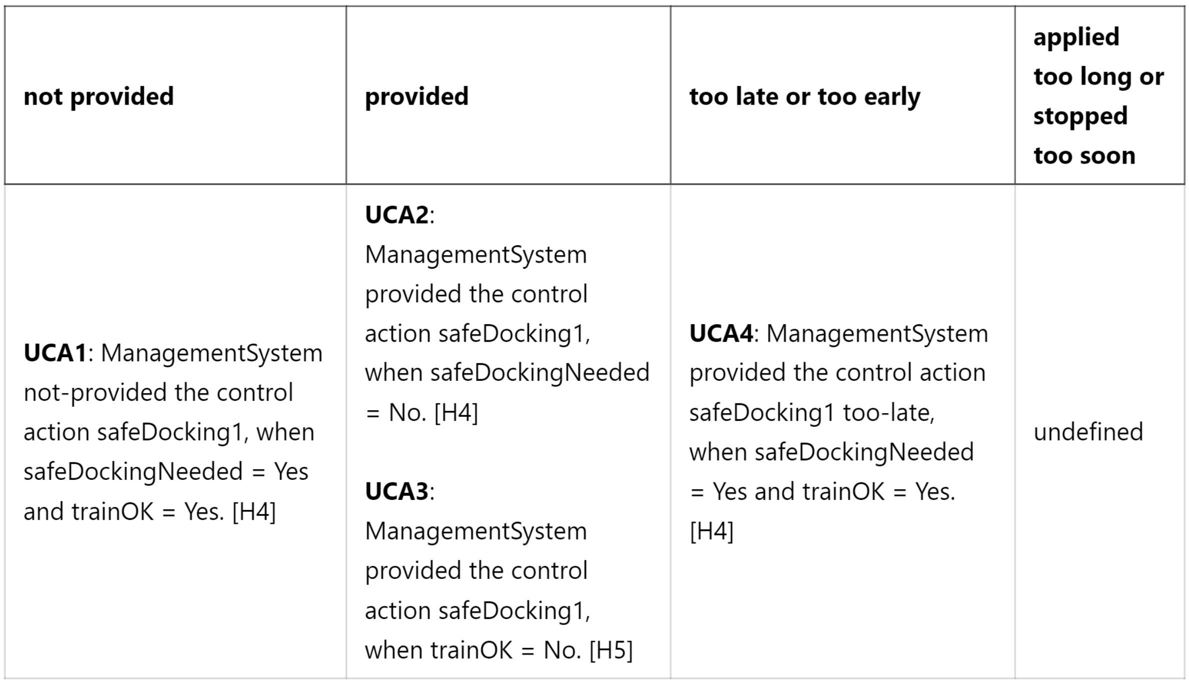
Figure 4.7. Control Structure for Single-Track Transfer Traffic

to measure distance during safe docking and whether the vehicles have docked together successfully. Multiple distance sensors are used for the precision needed when docking. These include longer range sensors, such as laser sensors with a range of about 50 meters. For closer ranges, ultrasonic sensors can be used, which are extremely accurate but are limited to a range of only a few meters at maximum.

The remote controller functions the same as during normal driving, allowing it to take over the train at any time. If the remote controller takes over the train during safe docking, it will be canceled. The management system can send control actions to initiate and abort safe docking and receives confirmations from the train on the status. It receives constant updates on the position and speed of the train, as it does during normal driving, enabling the system to track the train in real time. The trains can establish a connection with each other, allowing them to exchange information about their location needed to calculate the distance between them. In addition, they can inform each other about a possible abort if one of the trains cannot perform a safe docking for some reason. Lastly, responsibilities have been assigned to the different components. The responsibilities for the distance and docking sensors are shown in Listing 4.6.

```
DockingSensors{
    R8 "Confrim if the vehicles have docked together"[SC3]
    R9 "Confirm that docking was at a safe pace, without damaging to
       vehicles"[SC2]
}
DistanceSensors{
    R10 "Give the correct distance between vehicles, when they are close
       enough to measure"[SC6]
}
```

Listing 4.6. Responsibilities of the different components during safe docking.



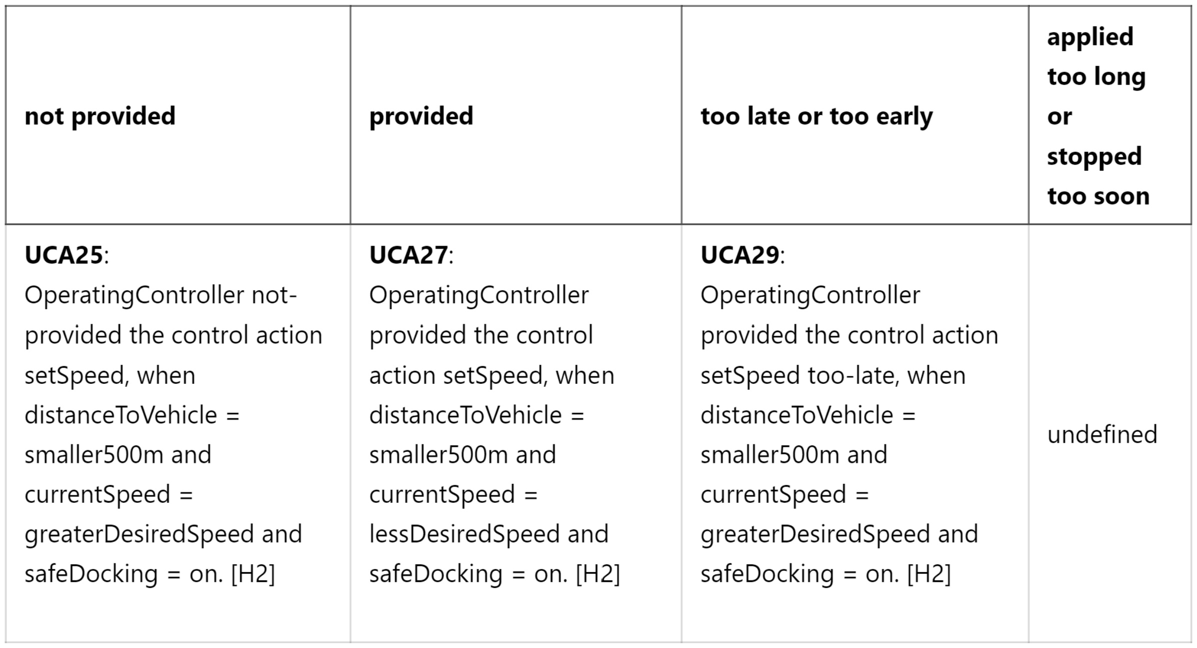
not provided	**provided**	**too late or too early**	**applied too long or stopped too soon**
UCA1: ManagementSystem not-provided the control action safeDocking1, when safeDockingNeeded = Yes and trainOK = Yes. [H4]	**UCA2**: ManagementSystem provided the control action safeDocking1, when safeDockingNeeded = No. [H4] **UCA3**: ManagementSystem provided the control action safeDocking1, when trainOK = No. [H5]	**UCA4**: ManagementSystem provided the control action safeDocking1 too-late, when safeDockingNeeded = Yes and trainOK = Yes. [H4]	undefined

Figure 4.8. UCAs for management system sending notification to the trains to dock together.

The sensor's responsibilities are to provide the controller with accurate information about the environment of the trains. Responsibilities for the management systems are to manage the Single-Track Transfer Traffic by notifying the right trains at the right time. The train controller's responsibility is to adjust the train's speed based on the distance between vehicles and to ensure that safe docking can be performed without interruption. The full list of responsibilities can be found on `github`.

4.2.3 Identify Unsafe Control Actions

The next step is to identify UCAs. This STPA analyses only a concept for Single-Track Transfer Traffic, without an exact process model for the management system or exact units to communicate. Therefore, this identification of UCAs, especially for UCAs send by the management system, is a bit more abstract. The identified UCAs for the management system notifying the trains, to safely dock together are shown in Figure 4.8. These UCAs show, that it is extremely important for the management system to inform the trains about Single-Track Transfer Traffic at the correct time, and only if it is necessary. If the trains are not notified or are notified too late, the best case is that the trains driving operation comes to an unexpected halt. The worst case is that both trains crash. However, it is also unsafe to provide the action



not provided	**provided**	**too late or too early**	**applied too long or stopped too soon**
UCA25: OperatingController not-provided the control action setSpeed, when distanceToVehicle = smaller500m and currentSpeed = greaterDesiredSpeed and safeDocking = on. [H2]	**UCA27**: OperatingController provided the control action setSpeed, when distanceToVehicle = smaller500m and currentSpeed = lessDesiredSpeed and safeDocking = on. [H2]	**UCA29**: OperatingController provided the control action setSpeed too-late, when distanceToVehicle = smaller500m and currentSpeed = greaterDesiredSpeed and safeDocking = on. [H2]	undefined

Figure 4.9. UCAs for the train adjusting speed during safe docking

when it is not necessary or the trains are unable to carry it out, as this leads to unknown consequences. UCAs related to aborting safe docking, whether the abort comes from the management system or one of the trains, are pretty similar to the UCAs in Figure 4.8. It is unsafe not to abort the procedure or to abort it too late when the trains cannot perform it, but it is also unsafe to abort it without proper reason, as this leads to unknown consequences.

Further UCAs are related to not sending information on the position of trains on the track during the entire process, starting too late or stopping too soon. It is extremely important that the trains have accurate knowledge about the distance between them, to minimize risk when docking together. The operating controller also needs to adjust the trains speed when both vehicles approach each other. UCAs identified for speed adjustment are shown in Figure 4.9. These UCAs are all related to *H*2, which means that if they occur, the trains risk crashing together. Therefore, it is vital that the controller has a good control algorithm to prevent these UCAs from occurring. These UCAs occur during the calculation of the distance to the vehicle using the GPS position calculated to a track position of the other vehicle, where the train does not adjust its speed correctly. Additional UCAs occur when distance sensors are used to identify the opposing train. At that point, the trains must decelerate to a minimum speed when approaching. Going faster could result in a to hard contact when docking, risking damaging the train or passengers.

In the end CCs for each UCA have been defined, to constrain the UCAs from occurring, similar to how the SCs were defined.

4.2.4 Identify Loss Scenarios

The last step is to identify loss scenarios. Most scenarios in which actions are not provided, even though they should be, can be traced back to inadequate feedback and information. Examples of this can be seen in Listing 4.7.

```
Scenario28 for UCA25
"The train receives incorrect or outdated positioning information from the
    other train, leading it to calculate a distance between them, that
   does not correspond to reality"[H2]
Scenario29 for UCA26
"The distance sensors do not identify the other train correctly, leading
   the train to not slow down further"[H2]
Scenario30 for UCA26
"The distance sensors measure an incorrect distance to the other train,
   that is way to far away."[H2]
```

Listing 4.7. Loss scenarios for UCA23 and UCA24

In these scenarios, the operating controller receives incorrect information about its position relative to the other vehicle. This leads the controller in the worst case to think that the vehicles are still far from each other, and therefore leads the train to not decelerate or even to start accelerating. Other scenarios for the controller not providing actions, such as sending an abort request when necessary, can be due to an inadequate control algorithm or process model, where the train already thinks it send the signal. A vital piece of this control structure, where a lot of UCAs can arise is the connection that must be established between vehicles. A lot of UCAs occur in scenarios where there is an unstable connection, such as the scenarios which can be seen in Listing 4.8

```
Scenario19 for UCA21
"The train wants to send out the notification to abort, but needs to
   establish a connection with them first because:
    - The connection was lost prior
    - A connection was never established
    - A former connection had faults and needed to be re-established
This leading to delayed notifations."[H2]
Scenario21 for UCA22
```

```
"The connection is establish incorrect or lost, therefore the train can
    not send its position on the track."[H2,H6]
```

Listing 4.8. Loss scenarios

*Scenario*19 shows a scenario in which the signal to abort is sent too late to the other train, because it first must establish a new connection, as the former was lost or faulty. This could cost valuable time, where the trains do not decelerate when moving towards each other. *Scenario*21 shows a scenario, where the train must not abort but can not send its position to the other train because the connection is not established correctly. This way the trains do not know how far away they are from each other and therefore they can not slow down appropriately, and in a worst-case scenario, they trains are too fast once the distance sensors detect them and cannot slow down enough to safely dock together. The full list of scenarios can be found on `github`.

Chapter 5

Safe Behavior Model

In this chapter, a Safe Behavior Model (SBM) is constructed for the implementation of a safe drive controller, that ensure that the UCAs and DCAs identified in the STPA hold. Using the drive controller constructed in the STPA and its related UCAs and DCAs, PASTA provides generation of a SBM as an SCChart. Since the SBM is based on LTL formulas generated out of the UCAs and DCAs of the drive controller, the generated SBM is correct by construction [PH25]. Thus, a SBM has been generated covering safety properties. I further added initialization of variables and calculations of intern values. The SBM receives two inputs, the desired speed the train should accelerate to and the desired driving mode, e.g. forward, backward, or emergency stop. These are the values of the control actions that the drive controller receives from the operating controller. The output values are the current speed to which the train should accelerate and the current drive direction to which the train should drive, which are the values that the drive controller converts into a format the physical units of the train can process and then gives these converted values to the physical units, thereby changing the train's speed or direction. A visualization of the full SBM can be seen in Figure 5.1. During initialization, the desired speed is set to 0 km/h and the driving mode is set to forward. After initialization, the SBM starts in the initial state $NoAction$. In this state, the outputs do not change and remain the same. When the SBM receives an input changing the desired speed, the train changes to the states $accforwards$ or $decforwards$ seen in Figure 5.2. These states are for accelerating and, respectively, decelerating while driving forward. There are separate states for driving backward, as a way to ensure that if the SBM receives the input to change direction, while driving faster than 0 km/h, the train slowly decelerates to 0 km/h driving the old direction before switching, thus avoiding any harsh direction changes. When entering the state, the output drive mode is set to 2, meaning forward. If the train was already moving forward, nothing changes, but if it was in a different mode before, the train now switches direction to driving forward. During acceleration, the train speed is then increased by one for every step taken by the SBM until it is equal to the desired speed. In that case, the train returns to $NoAction$. If the SBM receives a new input on the desired speed, it will go to the correct state to reach the new desired speed. The same applies to decelerating, with the only difference being that the train

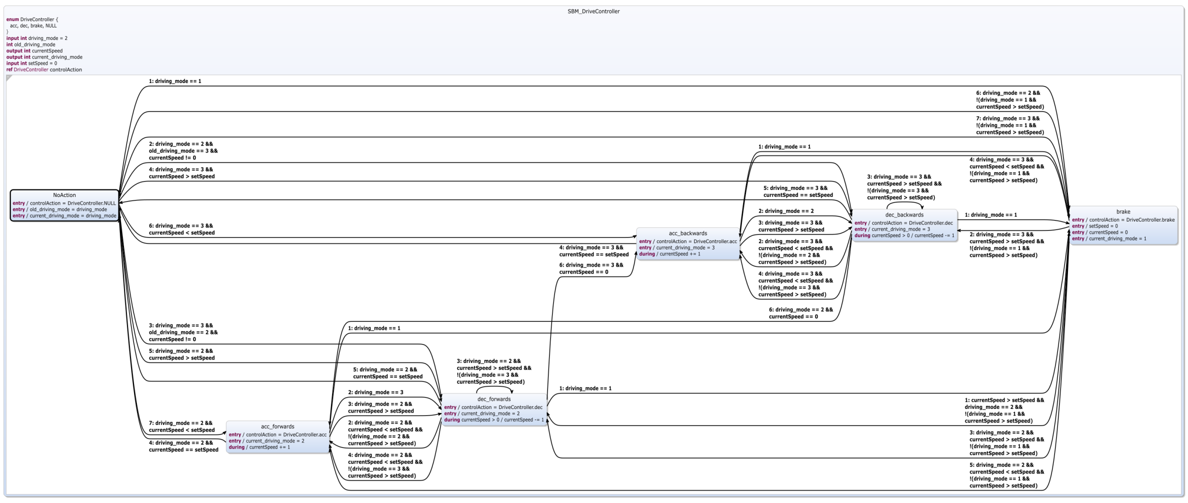


Figure 5.1. Visualization of the SBM for the Drive Controller as SCChart.

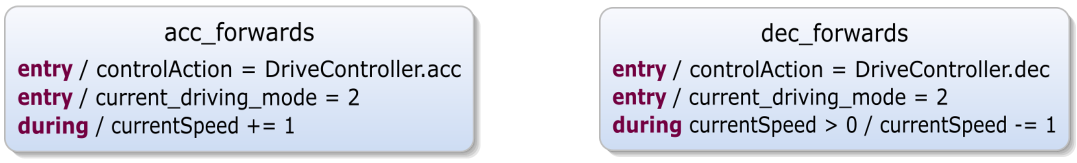

acc_forwards
entry / controlAction = DriveController.acc
entry / current_driving_mode = 2
during / currentSpeed += 1
dec_forwards
entry / controlAction = DriveController.dec
entry / current_driving_mode = 2
during currentSpeed > 0 / currentSpeed -= 1

Figure 5.2. States for acceleration and deceleration while driving forwards.

speed is reduced by one per step and not below zero. The states *accbackwards* and *decbackwards* function in the same way when the train is traveling backward.

If the SBM receives the input changing the driving mode to emergency stop, the SBM will immediately switch to the *brake* state. In this state, the train will immediately set the speed output to 0 and the train mode output to 1, giving the command to make an emergency stop. The SBM will not switch out of this state until the train's drive mode is changed.

Using Kiel Integrated Environment for Layout Eclipse Rich Client (KIELER), the SBM has been converted from a functional SCChart into a Python class. This class gives full functionality of the SBM for the implementation of the drive controller, ensuring that the control actions given to the physical units do not violate any UCAs and follow the DCAs. The class consists of class functions that allow the user to reset the SBM and to simulate a step of the SBM with the given input.

Chapter 6

Implementation

The implementation of the controller as a program is done in Python. I built a foundational implementation that is used to implement the controller for the specific test models. Based on this foundation I have further implemented specific implementations for the 1:32 model REAKTOR prototype and the digital twin. An implementation for the full-scale REAKTOR prototype has not been done, as the prototype has not been built yet. Further, it should be noted that the concepts of Single-Track Transfer Traffic are noted in the implementation, but not implemented. The reason for this is that the prototypes used for testing are not ready to test two trains safely docking together.

The controller process is shown using a flow diagram which can be seen in Figure 6.1. The controller starts by initiating. Then, it is checked whether the train is controlled remotely. If this is the case, the controller checks whether this is the first iteration, where the train is being controlled remotely. In case this is true, a flag is set, and additional info needed by the remote controller is sent. Then it is the controller's task to pass along the desired speed and mode from the remote controller to the train. If the train is not controlled remotely, the controller checks, if it was notified by the management system, to perform Single-Track Transfer Traffic. When the controller was notified, the train must follow the safe docking process proposed in Section 4.2. This concept is not implemented for the train as of yet, as indicated by the green box, and remains for future work. If the train is not supposed to dock to another train, it checks if it has a destination to drive to. If not, it waits until it receives one from the management system. Once it has a destination, it checks the outcome of the AI-obstacle detection represented as a danger score. This score of 0 to 100 shows the risk of obstacles, identified by cameras and sensors. When the score is higher than 90, the train switches to emergency mode and makes an emergency stop. When it is between 60 and 90, the train switches to degraded mode. When the score is below that, it drives in normal mode. The train then gets its position on the track, for example as GPS coordinates, and converts it into a position represented as track kilometers. Afterwords, the controller checks if it is still driving in the correct direction. If this is not the case, it needs to change direction. After that, it calculates how far it is from its destination. When the train is close to its destination, about 1 km

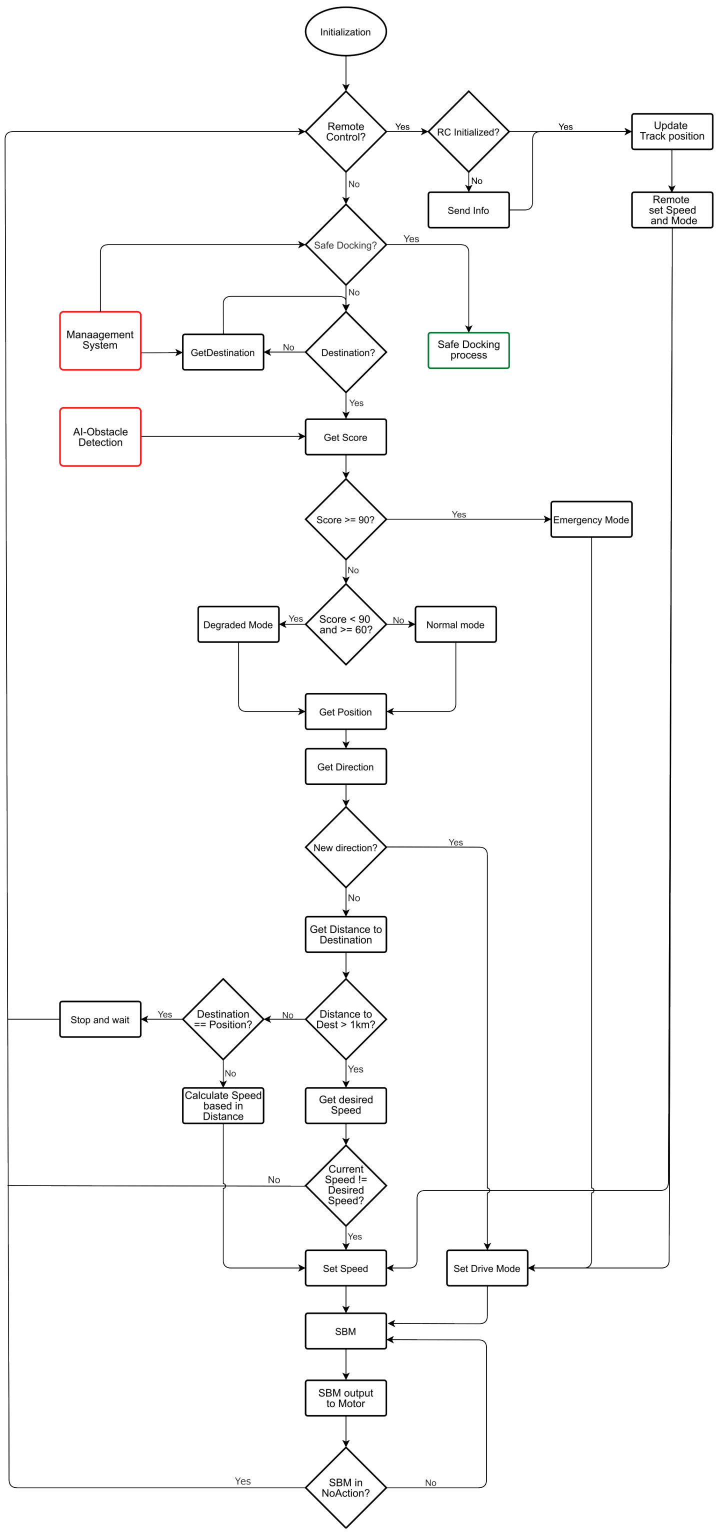


Figure 6.1. Flow diagram for autonomous controller

away, the train reduces its speed, calculating its desired speed based on the allowed speed limit and the remaining distance. As soon as it has reached its destination, it stops and waits for a new destination. When the train is more than a kilometer away from its destination, the speed limit for the train on this part of the track is identified. If it differs from the current speed that the train is driving, the speed is changed. Any speed changes, changes in drive direction, or emergency stops declared by the controller in this iteration will be given as input to the SBM, which generates the speed and direction, which should be set. In the end, these outputs are converted into a form that can be given to the motor. This is done as long as the SBM is still in an accelerating or decelerating state. Once the SBM is in the state *NoAction*, which means that the train has reached its desired speed and direction, the process starts from the beginning with a new iteration, checking whether the train is being controlled remotely.

6.1 Module Implementation

As shown in the control structure of the STPA, the controller is divided into three main modules: *OperatingController*, *DriveController*, and *TrackNavigation*. Further, there are clients for the remote controller and management system on the same device, allowing them, to request information and set specific variables, such as a destination, in the operating controller. This implementation for the autonomous controller described here is a foundation from which the specific implementations for the different environments can be built.

Operating Controller

The operating controller consists of a main function that implements the flow diagram up to the SBM. The implementation varies very little from the flow diagram, the only difference being that acceleration or deceleration is limited to 5 km/h for each iteration. The reason for this is because the implementation is only done using one thread, if acceleration or deceleration was not limited, the train would reach its speed limit, which can take a while, before starting a new iteration in the process flow. If AI-obstacle detection detects something during this, the controller can only react to the input in the next iteration. As an example, if the train is supposed to accelerate from 0 km/h to 50 km/h in one step, and the AI-obstacle detection detects mid-way something in the track with a score high enough to force an adjustment, the train would only react to it after reaching the desired speed of 50 km/h. This is, of course, far too late. By accelerating in steps of 5 km/h, the controller can check the danger assessment sent by the AI-obstacle detection 10 times in the same time frame. In addition, the arrival of the train at its destination is implemented with the precision of stopping the train within 10 meters. The controller can be started by starting this main function. Once started, it will control the train autonomously without needing additional user input in a continuous loop. To stop this loop and, thereby, the program, the keyboard interrupt is used, which gently stops the train and then correctly exits any existing programs used.

The management system's and remote controller's clients can receive and set different variables in the operating controller's process model. For the management system, this includes setting, getting and deleting the destination, getting the current track position, and getting the current train speed. The remote controller has the ability to take over and release the autonomous controller and to set the speed and mode of the train, receiving these values back as feedback. Lastly, it consists of a connection to the AI-Obstacle detection. The connection is built using a web-socket and then requests the current danger score identified by the detection. There is an existing risk of false positives. To avoid making an emergency stop for a false positive, multiple scores are requested, and the average score is returned for the autonomous controller to use.

Drive Controller

The functionality of the drive controller has already been implemented via the SBM discussed in Chapter 5. Now the only thing left is to implement a wrapper for the drive controller, updating the SBM when a change in speed or mode is sent by the operating controller. For this, two functions $setDriveControllerMode(mode : int)$ and $setSpeed(currentSpeed : int, desiredSpeed : int)$ are available. These functions receive the input values for the SBM from the operating controller. After computing the next step of the SBM, the output values received from it must then be transformed into a format that the physical units can read and then send to them to execute. This has to be done individually for each implementation for different vehicles. Lastly, the wrapper consists of an $initialize()$ and $exitController()$ function, where the SBM and other programs used in the wrapper are initialized at the beginning and shut down correctly at the end.

Track Navigation

This module stores track information and is used by the operating controller to calculate or obtain different important values. The track is stored as a dictionary, split into different intervals with a respective speed limit. For this implementation, a simplistic version of the 17 kilometer long Malente-Lütjenburg track was taken, dividing it into 3 different speed intervals, as this was enough to test the controllers capabilities. The knowledge of the track must be obtained before driving. To obtain the speed limit, the function $getDesiredSpeed(trackPosition)$ identifies the current interval in which the train is in and returns the speed limit of it. The function $findTrackPosition(GPSPosition)$ finds the current track position based on the positioning system used and needs to be adjusted accordingly to the system used. Other functions allow the operating controller to calculate the reduced speed when in degraded mode, obtain the driving direction, and calculate the distance to the destination.

Clients for Remote Controller and Management System

These clients each operate on their own thread and are in constant connection to the servers of the remote controller and the management system. They are able to set variables and obtain the value of variables using the functions in the operating controller, to fulfill the purposes of their respective component. Using this interface design allows for a capsulation of the different components. Further, additional clients for different systems can easily be implemented and have the possibility to receive the values of variables and set them. When doing this, possible conflicts between clients must, of course, be analyzed. The implementation of the clients has been done by their respective thesis.

6.2 1:32 Model REAKTOR Implementation

The controller has been implemented for the 1:32 model REAKTOR prototype. The prototype uses a Raspberry Pi 3, which is connected to a stripped-down engine. In the above section, the foundation for the autonomous controller was already built following the flow diagram in Figure 6.1. This implementation uses the foundation and builds upon it by implementing a drive controller wrapper for the model train. To control the inputs and outputs of the Pi, a Pigpio daemon is used. It is initialized together with the SBM, setting the correct pins. The train mode is set by adjusting pins 19 and 26. The train speed is set by adjusting the Pulse Width Modulation (PWM) set in a range of 0 and 100. The wrapper takes the output of the SBM and converts it into input for the pins and into PWM. For converting km/h into the correct PWM a linear function is used. Using $exitController()$ when interrupting the operating controller allows us to properly shutdown Pigpio before exiting the program.

Tracking the location of the train is difficult. Since the model track is only a small circle a few meters in length, using any positioning system such as GPS for this is not an option. Instead, I tracked the position using time, measuring the time between each loop of the operating controller. Using this time difference, the old position of the train, the train speed and driving direction, the trains position can be tracked as track kilometers. The formula when driving forward is:

$$oldPosition + (currentSpeed * (\frac{timePassed}{3600}))$$

The result is then rounded down to four decimals, and in case the train is driving backward, the position is subtracted instead. If the train is not driving, the position does not change. If the train is driving, the amount of meters is added/subtracted by multiplying the trains' speed with the time difference since the last loop. Since the time difference is expressed as seconds, km/h is converted to km/s. To test this implementation, I have multiplied the position change by a factor of 12, to reduce waiting times during testing.

For testing this implementation, a Pigpio daemon must be started. After that, *Main.py* can start the autonomous controller, the remote client, and the management system's client.

6.3 Digital Twin Implementation

The implementation for the digital twin also uses the foundation built in Section 6.1. In addition, this implementation does not need to convert the position of the train to track kilometers, as the digital twin uses them to position the train already. On request, it then gives the information already in the required form.

To read or set values in the digital twin, a web-socket is used. Connecting to the correct vehicle requires a vehicle ID, which must be known beforehand if there are multiple vehicles simulated. To set the speed and direction, the controller must then post this information using $requests.post()$ to the vehicle. The digital twin takes the speed input in km/h, so it is not necessary to convert the SBM output. To test this implementation, the application for the digital twin must be started on the same computer on which the autonomous controller is run. If a different computer is used, the *URL* used to connect to the digital twins trains must be changed accordingly.

Chapter 7

Evaluation

To evaluate the concepts of the autonomous controller, it is tested. The autonomous controller has been tested both independently and in combination with the other theses of the REAKTOR student project. The testing was done using both the controller implementations of the 1:32 model train and the digital twin.

7.1 Testing the Controller Independently

When I independently tested the controller in the 1:32 model train, the functions of the other REAKTOR theses were mocked in a simplistic way. The testing was done in different stages. In the first test, the controllers ability to drive the train forwards and backwards from point A to B while driving normally was tested. For this, random destinations on the track were generated and the danger score given by the mock AI obstacle detection was set to 0, meaning that the train would drive permanent in *normal* mode during this test. This test showed that the controller was capable of gently accelerating the train, driving to its destination. During driving, it successfully adjusted its speed to the changing speed limits given by the track navigation module. When approaching its destination, the train would slowly reduce its speed and stop within 10 meters of the track kilometer that was its destination. The second stage was to simulate the controller reaction to different inputs from the mocked AI-obstacle detection. For this, I let obstacle detection generate a random danger score between 0 and 100, with different probabilities. When driving, the controller successfully switched to the *degraded* mode, when receiving a high enough danger score, where it reduced its speed according to the score. In addition, it performed an emergency stop, when it received a score greater than 90. During this testing, the controller's ability to reach its destination was not impaired, other than the desired adjustments to the speed. A video of the train driving can be found on `github`.

7.2 Testing with other REAKTOR Theses

Further testing was done in combination with the other theses, with the main goal of testing the functionality of the interface between them and the controller and to test if the controller's ability to control the train is impaired. We tested different scenarios. The first scenario was again the normal use of the train with a clear track. The controller received a destination from the management system and slowly accelerated to the allowed speed limit. When approaching the destination, the controller gently decelerated the train until it stopped at the destination. This test showed that the controller successfully received a destination from the management system, showing that the interface between them functions. Additionally, no impairment of the controllers ability to drive was detected. For the next test, different obstacles were placed on the track, to test the trains reaction to these objects when they were identified by AI-obstacle detection. During this test, the train responded accordingly, by reducing its speed when objects were identified near the track and making an emergency stop for objects identified as on the track by the AI-obstacle detection. After the objects were removed, the train continued to drive normally. During this test, the remote controller also took control of the train. When the remote controller took over, the autonomous train controller did exactly as told. It no longer made its own decisions for the train, but instead just passed the speed and direction given by the remote controller to the train. This test showed that the interfaces with the AI-obstacle detection and remote controller work as intended. Both of these cases were tested for the 1:32 model train and the digital twin, where no differences in controller behavior could be seen, supporting the argument that the controller is easily scalable for different vehicles.

Chapter 8

Conclusion

In this chapter, a short summary of the contributions of this thesis is presented. After that, potential future work for the autonomous controller is presented.

8.1 Summary

In this thesis, I presented the concepts of an autonomous train controller for the REAKTOR student project. Using a STPA, potential hazards and dangerous interactions of the train controller were identified and a control structure was modeled. The controller consists of two main controllers and a module regarding the navigation of the railway track. The operating controller is the decision making controller and receives input from other systems and makes decisions for the train based on them. The drive controller, implemented as an SBM, ensures that these decisions are executed safely and accelerates or decelerates the train. The SBM was generated based on the STPA and then extended further. In addition, concepts for two trains docking together on the tracks are presented to realize Single-Track Transfer Traffic.

The testing of the controller on a 1:32 model train and digital twin revealed that it functioned in different scenarios as intended. The testing further revealed that the interfaces to the management system, AI-obstacle detection, and remote controller function as intended.

8.2 Future Work

Future work for this project includes a controller implementation for testing on the Malente-Lütjenburg track. Currently, a full-scale prototype is under construction that will be able to drive on the research track. To test the controller for this prototype, an implementation must be built. To do this, the foundation I built can be used. Then further, it must be implemented, how the outputs of the SBM in the drive controller are converted into a format, which the motor and brakes of the prototype can process, and how the physical units receive these outputs. In addition, the positioning system used for the train must be converted into track kilometers and the physical track must be implemented in more detail in the track navigation module. This includes detailed track intervals with correct speed limits.

In the future, the concepts for Single-Track Transfer Traffic must be implemented, to be able to realize the concept for rural public transport proposed by the REAKT project. To implement and test the concept of Single-Track Transfer Traffic, two testing vehicles with communication systems are needed. The testing vehicles need to be able to send their own position to the other vehicle and decipher the position of the other vehicle when received. In addition, these vehicles need distance sensors to safely receive the distance to the other vehicle on the last few meters, when geo-positioning gets to imprecise.

Bibliography

[BPR93] Robert J Borgovini, Steve Pemberton, and Michael J Rossi. *Failure mode, effects, and criticality analysis (fmeca)*. 1993. URL: https://api.semanticscholar.org/CorpusID:107361063.

[EN11] BS EN. “50128 (2011). railway applications-communication, signalling and processing systems: software for railway control and protection systems”. In: *International Electrotechnical Commission* (2011).

[HRV+81] D F Haasl, N H Roberts, W E Vesely, and F F Goldberg. *Fault tree handbook*. Tech. rep. Nuclear Regulatory Commission, Washington, DC (USA). Office of Nuclear Regulatory Research, Jan. 1981. URL: https://www.osti.gov/biblio/5762464.

[II05] Clifton A. Ericson II. “Event tree analysis”. In: *Hazard Analysis Techniques for System Safety*. John Wiley Sons, Ltd, 2005. Chap. 12, pp. 223–234. ISBN: 9780471739425. DOI: https://doi.org/10.1002/0471739421.ch12. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471739421.ch12. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471739421.ch12.

[Kar24] Beate Karlsen. *Humans: trouble, or treasure in the eyes of en 50126-1:2017 railway applications - the specification and demonstration of reliability, availability, maintainability and safety (rams).* eng. Student Paper. 2024.

[Kle99] Trevor A. Kletz. *Hazop hazan: identifying and assessing process industry hazards*. CRC Press, 1999. DOI: https://doi.org/10.1201/9780203752227.

[Koo22] Philip Koopman. *Ul 4600: standard for safety for the evaluation of autonomous products*. https://users.ece.cmu.edu/~koopman/ul4600/index.html. Dec. 2022.

[Lev12] Nancy G. Leveson. *Engineering a safer world: systems thinking applied to safety*. The MIT Press, Jan. 2012. DOI: 10.7551/mitpress/8179.001.0001. URL: https://doi.org/10.7551/mitpress/8179.001.0001.

[LT18] Nancy G. Leveson and John P. Thomas. *STPA handbook*. 2018. URL: https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

[PH25] Jette Petzold and Reinhard von Hanxleden. “Safe behavior model synthesis: from STPA to LTL to SCCharts”. In: *13th International Conference on Model-Based Software and System Engineering*. Jan. 2025, pp. 133–140. DOI: 10.5220/0013091600003896.

[PHL22] Jan Peleska, Anne E. Haxthausen, and Thierry Lecomte. "Standardisation considerations for autonomous train control". In: *Leveraging Applications of Formal Methods, Verification and Validation. Practice*. Ed. by Tiziana Margaria and Bernhard Steffen. Springer Nature Switzerland, 2022, pp. 286–307.

[PKH23] Jette Petzold, Jana Kreiß, and Reinhard von Hanxleden. "Pasta: pragmatic automated system-theoretic process analysis". In: *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2023, pp. 559–567. DOI: 10.1109/DSN58367.2023.00058.

[Std19] EN50129 Std. "Railway applications-communications, signalling and processing systems-safety related electronic systems for signalling". In: *European Committee for Electrotechnical Standardisation (CENELEC)* (2019).

[TCB+23] Abhimanyu Tonk, Mohammed Chelouati, Abderraouf Boussif, Julie Beugin, and Miloudi El Koursi. "A safety assurance methodology for autonomous trains". In: *Transportation Research Procedia* 72 (2023). TRA Lisbon 2022 Conference Proceedings Transport Research Arena (TRA Lisbon 2022),14th-17th November 2022, Lisboa, Portugal, pp. 3016–3023. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2023.11.849. URL: https://www.sciencedirect.com/science/article/pii/S235214652301147X.

[TDO+18] Damien Trentesaux, Rudy Dahyot, Abel Ouedraogo, Diego Arenas, Sébastien Lefebvre, Walter Schön, Benjamin Lussier, and Hugues Chéritel. "The autonomous train". In: *2018 13th Annual Conference on System of Systems Engineering (SoSE)*. 2018, pp. 514–520. DOI: 10.1109/SYSOSE.2018.8428771.

[Tho13] John P. Thomas. "Extending and automating a systems-theoretic hazard analysis for requirements generation and analysis". PhD thesis. Massachusetts Institute of Technology, 2013, pp. 64–68.

[YZT19] Fei Yan, Shijie Zhang, and Tao Tang. "Autonomous train operational safety assurance by accidental scenarios searching". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019, pp. 3488–3495. DOI: 10.1109/ITSC.2019.8917006.

List of Abbreviations

ATO	Autonomous Train Operator
CC	Controller Constraint
DCA	Desired Control Action
GoA	Grade of Automation
KIELER	Kiel Integrated Environment for Layout Eclipse Rich Client
LTL	Linear Temporal Logic
PASTA	Pragmatic Automated System-Theoretic Process Analysis
PWM	Pulse Width Modulation
SBM	Safe Behavior Model
SCChart	Sequentially Constructive Chart
SC	System-level Constraint
STAMP	System-Theoretic Accident Model and Processes
STPA	System-Theoretic Process Analysis

UCA	Unsafe Control Action
VS Code	Visual Studio Code