UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Laima Anna Dalbina

Simulation-based Safety Testing of an Automated Driving System (ADS)

Master's Thesis (30 ECTS)

Supervisor: Dietmar Alfred Paul Kurt Pfahl, PhD
Institute of Computer Science
University of Tartu
Estonia

Tartu 2024

Simulation-based Safety Testing of an Automated Driving System (ADS)

Abstract:

An Automated Driving System (ADS) is challenging to test due to the inherent complexity of the interaction between its components. Moreover, the real-world traffic in which an ADS is expected to perform is difficult due to the variety and highly dynamic interaction with other traffic agents. These systems have to be tested to ensure that the ADS maintains a desired level of safety during real-time operations. Testing an ADS in the real world incurs high costs and might be risky due to dangerous and unpredictable situations. This thesis focuses on an approach that uses simulation to test an ADS. The contributions of this thesis are two-fold. First, a method for simulation-based safety testing of ADS using the CARLA simulation environment is developed. The second contribution is the design and execution of a set of experiments to validate the method, whose results have been analyzed. The experiments were used to evaluate the ADS autonomy software currently under development in the Autonomous Driving Lab (ADL) of the University of Tartu. The thesis discusses the results of the experiments and the limitations of the developed method for simulation-based safety testing.

Keywords:

safety, testing, ADS, autonomy, simulation, CARLA

CERCS: P175

Isejuhtiva süsteemi (IJS) simulatsioonipõhine ohutustestimine

Lühikokkuvõte:

Isejuhtivat süsteemi (IJS) on keeruline testida selle komponentide vaheliste seoste keerukuse tõttu. Veel enam, peab isejuhtiv süsteem toimima reaalses liikluses, mis on väga mitmekesine ja paljude osapooltega, mille omavahelised interaktsioonid on äärmiselt dünaamilised. Sellepärast tuleb neid süsteeme testida tagamaks, et IJS säilitab päriselus töötamise ajal soovitud ohutustaseme. Isejuhtiva süsteemi testimine päris maailmas nõuab suuri kulusid ning võib olla riskantne ohtlike ja ettearvamatute olukordade tõttu. Antud lõputöö keskendub lähenemisviisile, mis kasutab IJSi testimiseks simulatsiooni. Esmalt töötatakse välja meetod isejuhtiva süsteemi simulatsioonipõhiseks ohutustestimiseks, kasutades CARLA simulatsioonikeskkonda. Teiseks kavandatakse katsete komplekt meetodi valideerimiseks, viiakse läbi vastavad katsed ja analüüsitakse tulemusi. Katsetulemustest lähtuvalt hinnati Tartu Ülikooli isejuhtivate sõidukite labori arendamisel olevat autonoomia tarkvara. Lõputöös käsitletakse katsete tulemusi ja väljatöötatud meetodi piiranguid simulatsioonipõhiseks ohutustestimiseks.

Võtmesõnad:

ohutus, testimine, IJS, autonoomia, simulatsioon, CARLA

CERCS: P175

Table of Contents

List of Figures

List of Tables

List of Abbreviations

3D	three dimensions or three dimensional
ACC	Adaptive Cruise Control
ADL	Autonomous Driving Lab
ADS	Automated Driving Systems
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AV	Autonomous Vehicle
CAV	Connected and Automated Vehicle
DDT	Dynamic Driving Task
ERM	Empirical Reference Mode
GES	General Estimates System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HARA	Hazard And Risk Analysis
IMU	Inertial Measurement Unit
JSON	JavaScript Object Notation
LiDAR	Light Detection And Ranging
MB	Manual Braking
NHTSA	National Highway Traffic Safety Administration
ODD	Operational Design Domain
PCS	Pre-Collision System
PID	Proportional–Integral–Derivative
PRB or PoRiBa	Positive Risk Balance
RADAR	Radio Detection and Ranging
ROS	Robot Operating System
RSS	Responsibility-Sensitive Safety
SAE	Society of Automotive Engineers
SOTIF	Safety of The Intended Functionality
TD	Test Dummy
TRF	Theoretical Reference Mode
VO	Virtual Object
XML	Extensible Markup Language

1 Introduction

The interest in Automated Driving Systems (ADS) has grown in research and industry. Companies like Waymo and Tesla have made their self-driving technologies available for public use, Waymo Driver [1] and Autopilot [2], respectively.

An ADS consists of several sensing, perception, planning, and control modules [3]. The sensing module uses various sensors to gather and pre-process environmental data, including cameras, Light Detection and Ranging (LiDAR), and Radio Detection and Ranging (RADAR). The perception module gathers data from the sensors to comprehend the surroundings. The planning module generates the ideal driving trajectories anticipated to be followed by the ADS based on the output of the perception module. The control module delivers lateral and longitudinal control signals to execute control within the ADS along the predetermined paths. While these modules work together, enabling the vehicle to move autonomously, the collaboration between the modules increases complexity, which causes unexpected problems to appear. Hence, extensive testing is required to test the modules separately and together.

Testing of the system is not only required due to the complexity of modules but also because of the environment around it. The traffic on the road is generally changing, dynamic, and unpredictable. An ADS in traffic has to be safe and reliable. An ADS is required to be safer than a human driver by reducing the human-dependent factors such as perceiving, predicting, deciding, vehicle control performance, and incapacitation [4]. Koopman [5] suggests that humans are good drivers, considering the complexity of the task. Humans are quite good at compensating for vehicle failures and other drivers. In contrast to computers, humans have common sense and are able to adapt to unknown situations. It means that the ADS has to excel at these tasks and situations to be considered as safe as humans.

The high complexity of the ADS and the environment requires extensive testing. The study by Kalra and Paddock [6] shows that an autonomous vehicle must complete hundreds of millions of miles to demonstrate reliability. The study result implies that only real-world testing is insufficient, and methods like accelerated testing, virtual testing and simulations, mathematical modeling and analysis, scenario and behavior testing, pilot studies, and extensive focused testing of hardware and software systems must be used.

Simulation-based testing enables us to examine how ADS behave in various situations, settings, system configurations, and driver characteristics. Setting up the test environment provides stable and repeatable situations to validate and verify the ADS.

The results from testing in the simulators are often binary, indicating whether a collision or any other rule violation (e.g., slipping from the road) happened. These values do not provide enough details to evaluate the safety. The thesis aims to have a detailed analysis of a chosen scenario while referring to the reference modes for evaluating the simulation and the ADS.

The contribution of the thesis is two-fold:

1. A method for simulation-based safety testing of an ADS is developed. The method describes the procedure to implement a scenario, connect the simulated ADS control in the CARLA simulator, and design experiments and measures to compare the safety behavior of the ADS.

2. The execution of a set of experimental simulation runs and the analysis of the results. The part of the thesis serves as a proof-of-concept for the method's applicability and provides examples on the use of the method for future users. Also, the logged data is analyzed to conclude the safety of the ADS in the simulator and the simulator's comparison to the real-world behavior of the vehicle.

The thesis is structured as follows. The Section 2 (Background) discusses the relevant technologies and methodologies. It includes the description of ADS and its safety, scenario-based testing, scenario catalogs and standards, simulation environment, and the ADS in the ADL. The Section 3 (Method) presents the method of the thesis. The simulation setup, experiment design, experimentation, and analysis are described. The Section 4 (Experiments) describes simulation experiments with two versions of implemented autonomy software stacks (SimV1 and SimV2) and the reference modes defined by the chosen scenario. The results of the experiments and comparisons are presented. The Section 5 (Discussion and Limitations) reports the discussion and limitations of the thesis. The Section 6 (Conclusion) consolidates the thesis and provides future ideas.

2 Background

This section describes the methodologies and technologies related to simulation-based safety testing of an ADS. Section 2.1 describes an ADS and ODD, the levels of autonomy, and the typical architecture of an ADS. Section 2.2 explains how the safety of an ADS is evaluated. Such methodologies as human as a baseline, safety metrics, and safety standards are described. Section 2.3 talks about scenario-based testing. Section 2.4 introduces scenario catalogs and standards. Section 2.5 describes a simulation environment and the software required. Section 2.6 mentions the ADS in the Autonomous Driving Lab, University of Tartu.

2.1 Automated Driving System

Automated Driving System (ADS) is "the hardware and software that are collectively capable of performing the entire dynamic driving task (DDT) on a sustained basis, regardless of whether it is limited to a specific operational design domain (ODD)" [7]. ADS refers to a Level 3, Level 4, or Level 5 system. Levels of autonomy are explained in Section 2.1.2.

2.1.1 Operational Design Domain

Operational Design Domain (ODD) is "operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, or the requisite presence or absence of certain traffic or roadway characteristics" [7].

2.1.2 Levels of Autonomy

The Society of Automotive Engineers (SAE) has proposed a taxonomy and definitions related to driving automation systems called *SAE J3016* [7]. It divides the driving automation systems into six levels, from Level 0 with no automation to Level 5 with full automation. The overview of the levels defined by SAE is illustrated in Figure 1.

Level 0 describes a vehicle with no driving automation technology, and the driver takes charge of the control of the vehicle. The vehicle gives warnings or momentary assistance. Examples are blind sport warning, lane departure warning, and automatic emergency braking.

Level 1 vehicles have lateral or longitudinal motion assistance, and the driver is responsible for driving the vehicle. The longitudinal motion of acceleration and braking is managed by adaptive cruise control (ACC), and the lateral motion of steering takes care of lane centering.

A vehicle with Level 2 driving automation supports longitudinal and lateral motion assistance, where the driver supervises the vehicle's actions and acts when required. Both ACC and lane centering are enabled.

Level 3 systems can execute actions autonomously defined within the ODD, although the driver has to be alert when the system requires control. An example of such a system is a traffic jam chauffeur.

Systems with Level 4 automation do not require any human interaction. The vehicle can drive autonomously within a certain ODD. An example of a Level 4 system is a local driverless taxi.

The level 5 system requires no human interaction and can be driven under any conditions.

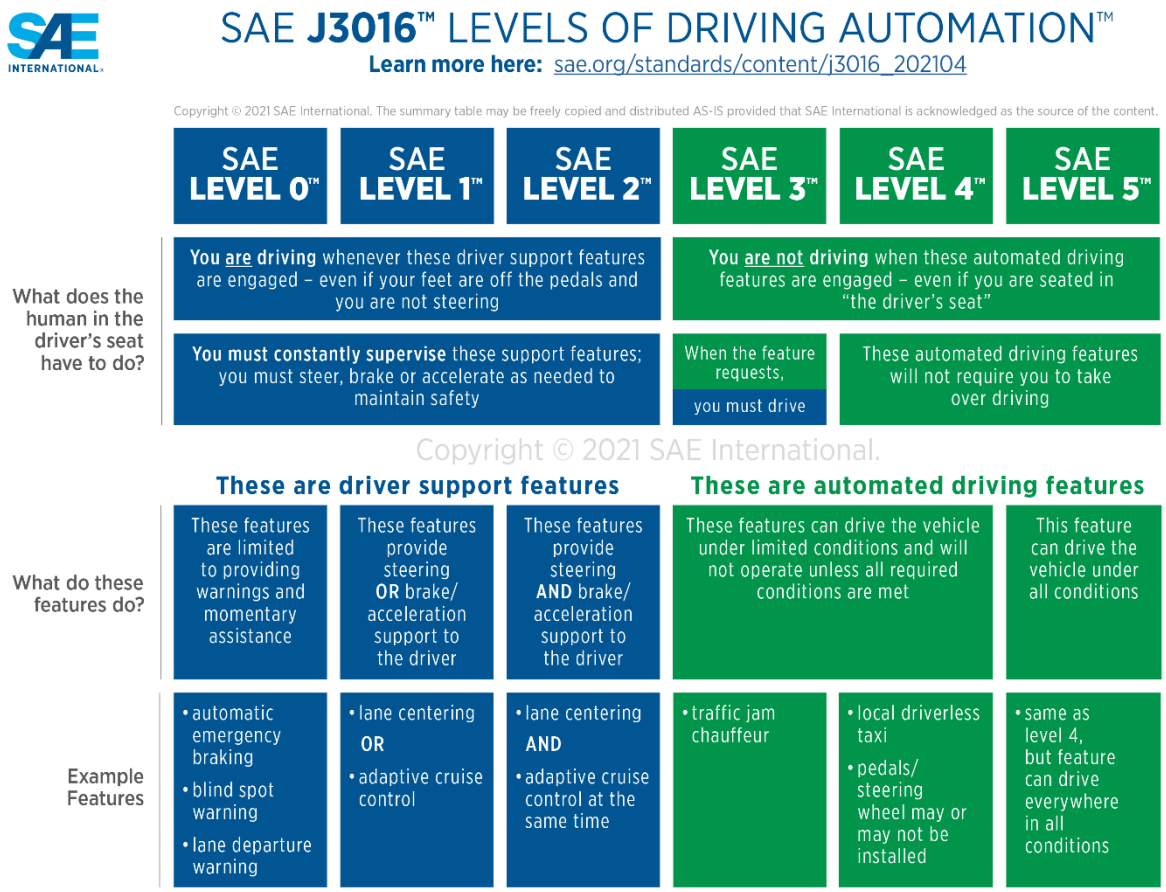

Figure 1. SAE J3016 levels of driving automation [8].

2.1.3 Architecture of ADS

There are two types of architectures for ADS that are most commonly used: modular approach and end-to-end model [9]. The overview of the modules is shown in Figure 2.

2.1.3.1 Modular Approach

The modular model utilizes four modules: the sensing module, the perception module, the planning module, and the control module [9].

In the sensing module, sensors are used to collect and pre-process the data from the environment. Global positioning systems (GPS), inertial measurement units (IMUs), cameras, radar, and LiDAR are standard sensors an ADS utilizes.

The perception module processes sensor data captured by the sensing module, like pictures and 3D point clouds from the sensing module, to perform localization and detection tasks. Localization is the real-time location of the ADS, and detection consists of lane detection, traffic light detection, and object detection.

The planning module uses the perception data as input and decides to control the vehicle. The prediction submodule estimates future trajectories of the objects around it, while the planning submodule generates an optimal trajectory for the vehicle.

The control module takes the output from the planning module and controls the vehicle using it.

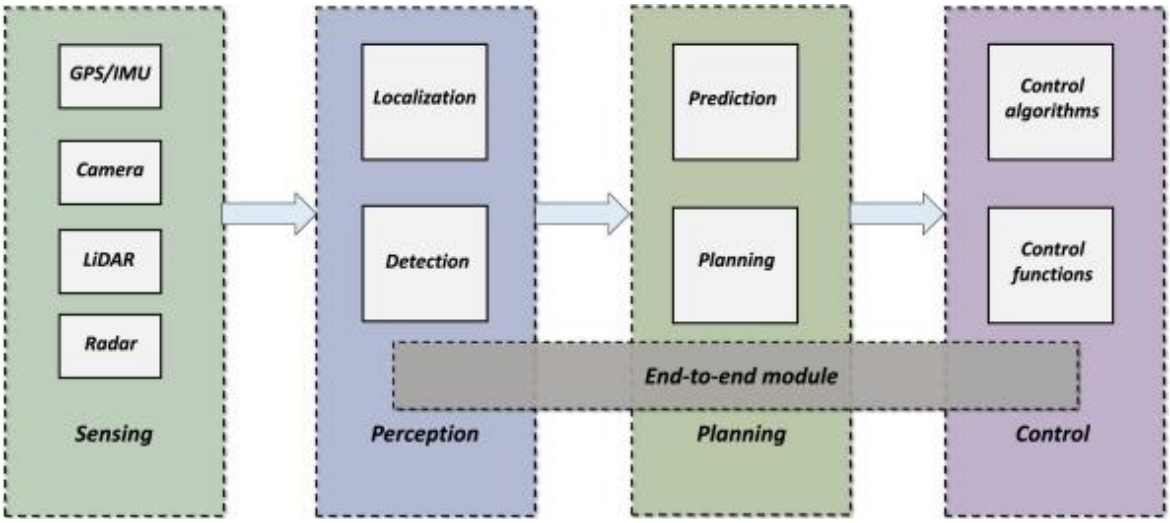

Figure 2. A typical architecture of an ADS [9].

2.1.3.2 End-to-end Approach

As shown in Figure 2, the end-to-end method combines the last three modules of the modular method, namely, perception, planning, and control modules, and combines them in one. The module mainly consists of deep learning models that take input from the sensing module and output the control for the vehicle [9].

2.2 Safety of ADS

Safety can be defined is several ways depending on who defines it. Koopman [5] has defined an Autonomous Vehicle (AV) safety hierarchy of needs, which describes the set of things to consider concerning the safety of an ADS. (In the context of this work, AV is synonymous with ADS)

In Figure 3, the safety hierarchy is visualized. On the bottom, the primary layer, which is the basic driving functionality, is illustrated. The vehicle has to operate in a defined environment without hitting any objects. Once accomplished, the next layer has to be addressed. Defensive driving means that the vehicle has expert driving skills while actively avoiding situations of increased risk. Afterward, systematic hazard analysis is to be performed to analyze and mitigate risks from driving functions and potential technical malfunctions. An example approach is to follow Hazard And Risk Analysis (HARA) from ISO 26262. More about the standard is mentioned in Section 2.2.3.1. On top of hazard analysis comes functional safety, which means that the risks introduced by technical faults in the system have been mitigated to an acceptable level. One of the principle approaches is Automotive Safety Integrity Levels (ASILs) from ISO 26262. The next step is the Safety of The Intended Functionality (SOTIF), which ensures that the unknowns and insufficiencies due to imperfect sensors and actuators have been addressed. It is done in conformance with ISO 21448 (SOTIF) — more about SOTIF in Section 2.2.3.2. The system-level safety level ensures that the safety analysis and risk mitigation have accounted for things beyond the driving task. It might be done in conformance with ANSI/UL 4600. More about ANSI/UL 4600 in Section 2.2.3.3. The social-technical considerations look at the aspects of interaction among ADS, road users, and other stakeholders regarding the safety of ADS. The last level of the hierarchy is safety culture, which refers to the culture of the organizational operations and maintenance of ADS to improve safety in a blame-free manner.

Figure 3. Koopman's autonomous vehicle safety hierarchy of needs [5].

According to Koopman [5], a system that addresses all the levels of the AV safety hierarchy is only one part of proving that the AV is safe. Koopman further mentions the need for testing for "millions of miles" with a low failure rate, the need for simulated billions of miles with a low failure rate, the usage of design approaches for safety, the reduction of risks to acceptable risks, and the expectation to be safer than a human in the relation to crashes, injuries, and fatalities. A combination of these statements might mean that the AV is safe.

2.2.1 Human Driver as Baseline for Safety

European Commission published a report [10] that discusses the ethics of Connected and Automated Vehicles (CAVs) and gives recommendations on road safety, privacy, fairness, explainability, and responsibility. It states that "a minimal requirement for manufacturers and deployers is to ensure that CAVs decrease, or at least do not increase, the amount of physical harm incurred by users of CAVs or other road users that are in interaction with CAVs, compared to the harm that is inflicted on these groups by an appropriately calculated benchmark based on conventional driving." It is termed Positive Risk Balance (PRB, sometimes called PoRiBa [11]). PRB is the idea that the ADS has to be at least as safe as a human driver. In other words, the risk associated with using ADS should be less than that of a vehicle operated by a human.

According to Koopman [5], setting a PRB goal requires three steps: selecting a baseline, accounting for operational conditions, and adding a sufficient engineering margin.

The baseline for a human driver to compare with should not be too simplistic. The national average statistics do not provide enough details. An experienced, undistracted, and unimpaired driver around 50 to 60 years old should be used as the baseline. According to statistics [12], they have the lowest crash and fatality rates when compared to other age groups.

The ADS must be compared in the same weather conditions to account for operational conditions, and the risk should be adjusted based on the locations.

To adjust for the engineering margin, it is necessary to account for uncertainty in the predictability of the deployed safety and add an additional margin to provide a clear improvement compared to a human driver. According to the RAND report [13], in the short term (within 15 years), more lives will be saved if the ADS is 10% safer than a human driver. In the long term (within 30 years), they consider to reach 75% and 90% safety.

2.2.2 Safety Metrics

Safety metrics are used to assess the safety of an ADS [5]. The safety metrics are divided into two categories: vehicle-level metrics and engineering metrics.

2.2.2.1 Vehicle-level Metrics

According to Koopman [5], vehicle-level metrics are based on vehicle measurements. One of the metrics is counting the number of crashes and other loss events, such as traffic law violations. Another metric counts how many miles the vehicle has driven. Disengagements count how many times the safety driver has had to take over, a biased and simplistic metric where the type of road driven and the safety driver differ. A coverage metric evaluates the completeness of the different types of testing. The physics-based risk metric uses Newton's laws of motion to predict and measure the actions and outcomes of the situation.

2.2.2.1.1 Physics-based Metrics

According to Koopman [5], physics-based metrics are based on Newton's laws of motion[1]. The difference stems from how they are applied and what values and actions are considered. One of the points to consider is the near-term changes in behavior. For example, tail-gating a leading vehicle does not give the time to react if the leading vehicle suddenly brakes with full force. Also, various geometries and situations such as following a vehicle, vehicles in the other lane, crossing, and merging are just a few examples to consider. Determining the equations for each behavior separately for each vehicle is possible. Another consideration regarding worst-case actions is the maximum turning, acceleration, and braking capabilities. Besides this, varied participants on the road – cars, trucks, bicycles, and pedestrians – have various maximum values for each attribute. Another factor that needs to be considered is the environmental factors such as friction of the road surface, tire grip, and steepness of the road.

Some approaches attempt to prove that they do not mathematically cause any collisions of their fault. However, a crash can happen if another road user acts improperly according to the rules. Nonetheless, the metrics can provide an in-depth understanding of the real-world and guidance in certain situations.

Another factor to consider is the tradeoff between permissiveness and safety. Permissiveness is the ability to give freedom to the system to perform its tasks [5]. If the system is made to be mathematically safe, then the driving behavior will be so conservative that the users would not use such a system. There will be risks involved in driving because the tradeoff between safety and permissiveness has to be managed.

2.2.2.1.2 Responsibility-Sensitive Safety

Based on Newton's laws of motion, Responsibility-Sensitive Safety (RSS) [14], proposed by Mobileye[2], a company working with driver assist and autonomous driving technologies,

[1] "Newton's laws of motion", Sir Isaac Newton. Available: https://www.physics.utoronto.ca/~jharlow/teaching/everyday06/reading01.htm

[2] Responsibility-Sensitive Safety, Mobileye. Available: https://www.mobileye.com/technology/responsibility-sensitive-safety/

has been defined. RSS is a rigorous mathematical model that assists the AV in responding to hazardous situations and specifies the real-time safety distance it must maintain concerning nearby vehicles. It defines five safety rules made up of formal logic and rules. The five rules and their descriptions are mentioned in Table 1. Mobileye claims that as long as all parties involved in the traffic obey the RSS responsibility principles, the safety regulations guarantee that there will not be any collisions. That means all vehicles have to be autonomous to guarantee the safety of all vehicles involved.

Rule	Description
Don't hit the car in front of you.	Describes the minimum longitudinal distance that has to be kept from the vehicle in front. The formula depends on the speed and acceleration parameters of both vehicles and the reaction time of the rear vehicle.
Don't cut in recklessly.	Describes the lateral distances that have to be kept when changing a lane. The formula depends on the speed and acceleration parameters of both vehicles and the reaction time of the rear vehicle.
Right of way is given, not taken.	Describes the rules that have to be obeyed about the right of way even if others violate this principle.
Be cautious in areas with limited visibility	Describes the rules that have to be obeyed in areas of limited visibility.
If you can avoid a crash without causing another one, you must	Describes the rules that must be obeyed in a dangerous situation that will cause a crash.

Table 1. Five safety rules of RSS and their description

2.2.2.1.3 Reaction Time

The physics-based metrics depend on the reaction time of the human driver. Reaction time is the elapsed time from the event's occurrence to the start of the required action applied. The reaction time is cumulative of perception time and action time. Perception time is the time taken to perceive and understand the event and the action required. For example, it is the time taken to detect that a pedestrian is walking across the road and the decision that an action of braking has to be taken. The action time is the time taken to apply the decided action. For example, to release the accelerator pedal, move the leg to the brake pedal and start the action.

The response time is varied for each driver. It can be affected by various factors, such as the driver's experience, the level of tiredness, distractions, soberness, and age. Yadav et al. [15] showed that alcohol was the most crucial factor impairing driver's reaction time. They showed that blood alcohol concentration levels of 0.03%, 0.05%, and 0.08% resulted in a

36%, 53%, and 94% increase in the driver's reaction time. Another study [16] shows that increased mental workload decreases the reaction time. Rolison et al. [17] show that inexperience and young age of drivers decrease reaction time.

Several studies have performed experiments to measure the reaction time of drivers either in a simulator [15], [16], [18], [19] or in a real-world environment [20]. Cameras and data from the simulations were used to determine the reaction time for the experiments performed in simulations. For the real-world experiment, two cameras were used to determine the time between the start of the event and the start of the foot movement. Also, the time between the start of the foot movement and the start of the movement to press the brake pedal was determined.

Studies have presented their results from the measurements, which vary from one study to another. The study [18] measured the mean reaction time for a concentrated driver to be 0.8 seconds. Another study [19] concludes that the reaction time for the same person varies depending on whether there is no task given or there is a task of reading a message on a screen or repeating numbers out loud. It shows the influence of distractions on the reaction time. They show that the mean reaction time for a particular person without any task is 0.8, although when the repeating numbers task is introduced, it increases to 1.7 seconds. Another study [20] measured the drivers' reaction to a light in the real world. They concluded that the average reaction time was 0.680 seconds.

2.2.2.2 Engineering Metrics

According to Koopman [5], engineering metrics focus on the evaluation of components of the ADS and their performance and the engineering process. A list of engineering metrics:

- Planning metrics – evaluate how well a vehicle can plan its path in the environment. For example, how the vehicle can position itself from other objects on the road.
- Perception metrics – assess the performance of the perception module. For example, evaluating the classification accuracy of the object detection module.
- Prediction metrics – evaluate the capability to predict the motion of other vehicles. For example, evaluating whether the module is able to track the same object over time.
- ODD metrics – evaluate how thoroughly the ODD has been validated and how complete the ODD definition is. For example, if the ODD includes weather conditions with snow, the metric evaluates whether such conditions are covered.
- Surprise metrics – estimate how many unknown situations are left to deal with. For example, to estimate unknown situations, using software reliability growth modeling[3].
- Conformance and engineering metrics – measure the conformance to an appropriate safety standard.

The evaluation and results of these metrics influence the safety of the ADS.

2.2.3 Safety Standards

Standards are required to assure the safety and compliance of the system. To handle safety at the various levels of driving automation, manufacturers must consider various current regulations and develop industry best practices because autonomous technology encompasses software, electrical, and conventional mechanical systems.

[3] https://nap.nationalacademies.org/read/18987/chapter/6

2.2.3.1 Road Vehicles – Functional Safety (ISO 26262)

The standard is for "safety-related systems that include one or more electrical and/or electronic (E/E) systems and that are installed in series production road vehicles, excluding mopeds" [21].

Goals of the standard:

- Provides a lifecycle for automobile safety (management, development, manufacture, operation, servicing, and decommissioning).
- Includes all aspects of functional safety throughout the development process, covering tasks like designing, implementing, integrating, validating, and configuring requirements.
- Offers a risk-based method tailored to the automobile industry for identifying risk classes (Automobile Safety Integrity Levels, or ASILs).
- Specifies the item's essential safety standards using ASILs to achieve an acceptable residual risk.
- Lays out specifications for verification and validation procedures to guarantee that a suitable and sufficient degree of safety is being reached.

2.2.3.2 Safety of The Intended Functionality (ISO 21448)

The Safety of The Intended Functionality (SOTIF) standard [22] provides a general argument framework and guidance on measures ensuring the safety of the intended functionality in the absence of a fault. According to Liu [23], enforcing SOTIF in the real world is challenging because testing for every scenario in a physical setting is nearly impossible. In a virtual environment, it is possible to test an unlimited number of scenarios and variations. The simulation has to have a realistic environment, correct modeling of physical forces, and accurate interactions with other dynamic objects.

2.2.3.3 ANSI/UL 4600

The UL 4600 standard's [24] main goal is to ensure a thorough safety case with supporting documentation, arguments, and safety claims is in place. It is a safety-first, technology-neutral standard tailored for autonomous vehicles and related products. UL 4600 defines the scope as follows:

- The scope of this standard is a generalized framework for autonomous systems, including light autonomous road cars as an example.
- The strategy chosen in this standard (UL 4600) requires a goal-based safety case that includes virtually all of the material required for safety assurance.
- This standard does not describe a process; instead, it establishes assessment criteria for determining the acceptability of a safety scenario.

The Standard covers the following topics: creating safety cases, risk analysis, designing for safety, testing, tool qualification, validating autonomy, data integrity, human-machine interaction (for non-drivers), life cycle issues, metrics, and compliance evaluation.

2.2.3.4 Road vehicles - Scenario Based Safety Evaluation Framework (ISO 34502)

The scenario-based safety evaluation framework standard [25] provides guidance for a scenario-based safety evaluation framework for automated driving systems. The framework explains a scenario-based safety evaluation process applied during product development.

The goals of the standard are:

- to provide an overview of the scenario-based safety evaluation process;
- to explain the relationship between this framework and other standards and legislation.

2.3 Scenario-based Testing

Several approaches have been identified for the safety testing of ADS. According to Khan et al. [26], there are three main safety testing methods: on-road Testing, simulation testing, and scenario-based testing. On-road testing aims to assess the ADS in real-life situations, either on public roads or enclosed areas. Simulation testing is performed in a software-based virtual environment. Scenario-based safety testing uses relevant scenarios to test a specific task of the ADS. Scenario-based testing can be combined with on-road and simulation testing [27].

Various scenarios are developed in scenario-based testing to evaluate the ego vehicle's performance. Compared to physical testing, the benefits of the simulation environment offer the flexibility to examine a wide range of scenarios quickly. These scenarios include pedestrians, traffic, and the ego car's interaction with the surrounding infrastructure.

The scenarios-based testing can be used within high-fidelity simulators that offer in-depth, comprehensive data collection and analysis. The creation of the scenario and the ego car's engagement with the scenario are the critical components of scenario-based testing. More about scenarios and simulators in Sections 2.5 and 2.4.

2.4 Scenario Catalogs and Standards

According to Go and Carroll [28], "a scenario is a description that contains (1) actors, (2) background information on the actors and assumptions about their environment, (3) actors' goals or objectives, and (4) sequences of actions and events." Menzel et al. [29] introduce three abstraction levels for a scenario: functional, logical, and concrete. Functional scenarios are the most abstract level, where the scenario is represented in a natural language easily understandable by human experts. Logical scenarios describe a more detailed functional scenario with variables that describe entities and their relations. Concrete scenarios describe the entities and relations of those entities with exact variables of parameters.

There exist standards and reports about categories of scenarios that give various situations that may occur with a vehicle and help to categorize them. Three of the most prominent scenario catalogs and standards are described below.

National Highway Traffic Safety Administration (NHTSA) has compiled a report, "Pre-Crash Scenario Typology for Crash Avoidance Research" [30]. The typology is based on the General Estimates System (GES) accident database, which includes pre-crash scenarios showing vehicle dynamics and movements and the crucial moment just before a crash. It includes thirty-seven pre-crash scenarios, each with its visualization, typical scenario description, factor over-representation, dynamic variations, scenario severity, and contributing factors.

Nanyang Technological University in Singapore and the Land Transport Authority of Singapore have compiled a report, "Scenario Categories for the Assessment of Automated Vehicles" [31]. The report aims to provide a suitable tagging system and describe scenarios covering many possible changes that could occur in real traffic. Their method generates an overview of 67 scenario types. This overview contains a wide selection of relevant and believable scenarios from which test cases might be developed.

IEEE Standard for Assumptions in Safety-Related Models for Automated Driving Systems [32] (IEEE 2846) defines a minimum set of reasonable assumptions and foreseeable scenarios to consider for ADS safety.

2.5 Simulation

A high-fidelity simulator has to be used to execute scenarios and gain in-depth and comprehensive data. According to [33], a set of requirements has to be defined for an ideal simulator for testing ADS:

- ADS has to perceive the virtual world;
- the simulator has to provide multi-view geometry;
- the simulator has to have interfaces for path-planning algorithms;
- the simulator has to be able to build models and control them;
- the simulator has to provide a realistic 3D environment;
- the simulator has to provide infrastructure for traffic;
- the simulator has to provide API to control the traffic for scenario execution;
- the simulator has to provide ground-truth data;
- the simulator should provide non-functional requirements such as stability, flexibility, portability, and scalability and be open-source.

Kaur et al. describe several simulators, such as Matlab/Simulink[4], CarSim[5], PreScan[6], CARLA [34], Gazebo[7] , and LGSVL[8]. They compare the simulators and conclude that CARLA and LGSVL simulators are the current state-of-the-art simulators for end-to-end testing of self-driving cars when looking at the requirements defined.

CARLA [34] is an open-source simulator for autonomous driving research. CARLA provides a flexible Application Programming Interface (API) that controls all simulation aspects, an autonomous driving sensor suite that users can configure, and traffic scenario simulation using ScenarioRunner.

2.5.1 Scenarios in CARLA

It is necessary to define the scenarios in a format that is readable and executable by a simulator. The ScenarioRunner [35] allows for the definition of scenarios in three ways: the OpenSCENARIO format, Python definition, and routes. All three methods provide visual feedback during the execution and metric feedback after the execution. The examples of all types are available in the ScenarioRunner repository[9].

ASAM OpenSCENARIO 1.0 defines a file format for describing complex and synchronized maneuvers for multiple entities for traffic simulators [36]. It defines the dynamic content of the environment such as entities and their behaviors. The file is structured in an XML format, supporting a hierarchical structure. Section 3.1.1 explains the implementation of a scenario in the OpenSCENARIO format.

[4] https://se.mathworks.com/products/simulink.html
[5] https://www.carsim.com/products/carsim/
[6] https://plm.sw.siemens.com/en-US/simcenter/autonomous-vehicle-solutions/prescan/
[7] https://gazebosim.org/home
[8] https://github.com/lgsvl/simulator
[9] https://github.com/carla-simulator/scenario_runner

Python-based scenarios are defined using *srunner*[10] and *py_trees*[11] libraries. They are implemented by overriding methods from *BasicScenario*[12] that initialize actors, create the behavior, and create test criteria.

Routes-based scenarios consist of the routes files in the XML format and the possible scenarios in various maps in the JSON files. When the ego vehicle drives along the route, it triggers several scenarios in which trigger points are along the route driven.

2.5.2 Autoware Mini Autonomy Software

A vehicle equipped with sensors to sense its surroundings is called an "ego vehicle"[13]. To simulate the ego vehicle, an autonomous driving stack has to be connected to the simulation. Autoware Mini is a minimalistic Python-based autonomy software developed by the Autonomous Driving Lab (ADL) at the University of Tartu [37]. The Autoware Mini autonomy software controls the ego vehicle in the scenarios.

2.5.3 Robot Operating System

Autoware Mini autonomy software uses a Robot Operating System (ROS). The necessary tools provided by ROS enable simple access to sensor data, processing of that data, and creation of a suitable response for the robot's actuators, including its motors.

The data transfer is performed using *topics*[14] intended for unidirectional streaming communication. A *node*[15] performs computation that publishes *messages*[16] to *topics*. A *bag*[17] is a file format that stores ROS message data. *Bags* are primarily used for data logging.

2.6 The Autonomous Driving System in the ADL

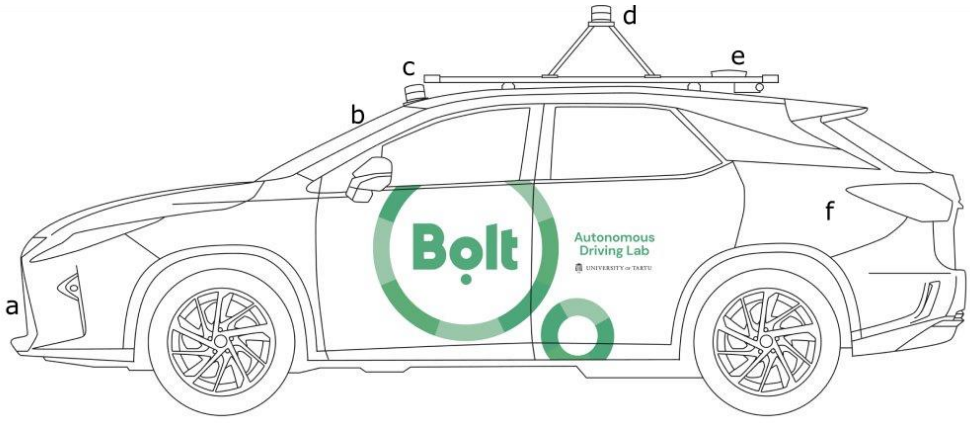

Figure 4. The Lexus RX450h[19] vehicle with its hardware marked a to f.

In real life, the autonomy software Autoware Mini is placed in the ADS. The car and hardware were adapted by AutonomouStuff[18]. The system consists of the Lexus RX450h vehicle with a Spectra computer and various sensors such as LiDARs, cameras, Global Navigation

[10] https://pypi.org/project/srunner/
[11] https://pypi.org/project/py-trees/
[12] https://github.com/carla-simulator/scenario_runner/blob/master/srunner/scenarios/basic_scenario.py
[13] https://se.mathworks.com/help/driving/ug/coordinate-systems.html
[14] https://wiki.ros.org/Topics
[15] https://wiki.ros.org/Nodes
[16] https://wiki.ros.org/Messages
[17] https://wiki.ros.org/Bags
[18] https://autonomoustuff.com/

Satellite System (GNSS), and radar[19]. In the Figure 4, the Lexus RX450h vehicle is illustrated. The hardware components are marked in the figure with letters from a to f. In Table 2, each of the marked components is named and described. In the ADL, it is also called "Autonomous Driving Platform".

The letter marked in the image	Name of the component	Description
a	Aptiv ESR 2.5 radar	The radar uses radio waves to estimate the distance to the objects ahead.
b	Allied Vision Mako G-319C cameras	Two cameras are responsible for the detection of traffic lights. One camera is turned to the left to detect the farther traffic lights, and the right one is pointed to the ones positioned on the right side, which is usually closer.
c	Ouster OS1-128 LiDAR	A short-ranged LiDAR for detecting close-by objects. It has 128 rotating laser beams. The detection range is around 40 meters.
d	Velodyne VLP-32C LiDAR	A long-range LiDAR for detecting farther objects. It has 32 LiDAR beams. The detection range is around 80 meters.
e	NovAtel PwrPak7D GNSS device	The GNSS devices determine the location of the vehicle using satellite positioning. Positioning accuracy is 5 to 10 cm.
f	AStuff Spectra computer	The computer that runs the autonomy software on an Ubuntu operating system.

Table 2. The sensors and hardware components in the Lexus RX450h vehicle[19].

The ADS is made up of three main parts: the vehicle itself, the autonomy software, and the sensors added to the vehicle. The autonomy software uses the input from the sensors to apply the actions to the vehicle. An overview of the architecture can be seen in Figure 5.

[19] https://adl.cs.ut.ee/lab/vehicle

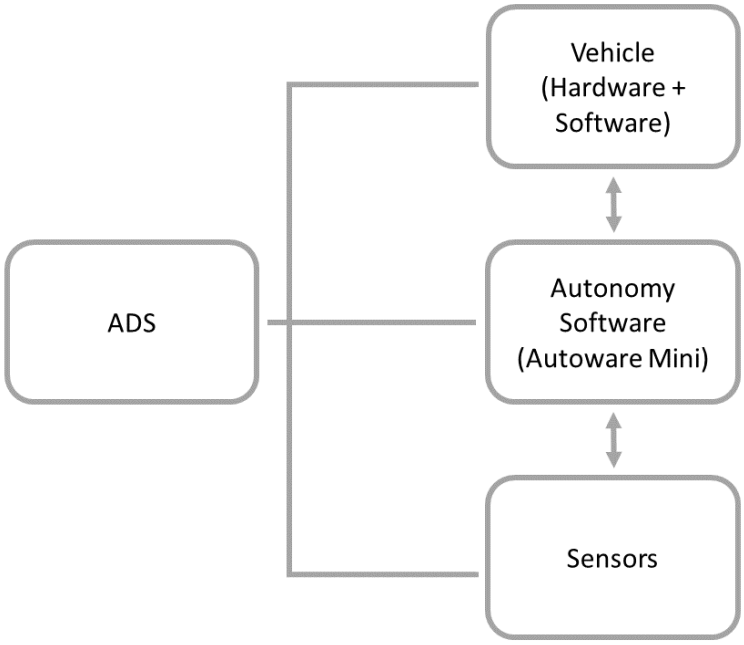

Figure 5. The architecture of the ADS

2.6.1 Built-in Pre-Collision System of Lexus RX450h

The Lexus RX450h vehicle has a driving support system called a Pre-Collision System (PCS) [38]. It uses radar and a front camera to detect objects such as vehicles, pedestrians, and bicycles in front of the vehicle.

The owner's manual [38] of the Lexus RX450h vehicle states, "if the system determines that the possibility of a frontal collision with an object is extremely high, the brakes are automatically applied to help avoid the collision or help reduce the impact of the collision."

2.6.2 Use of Autonomy Software in Simulator and Real-World

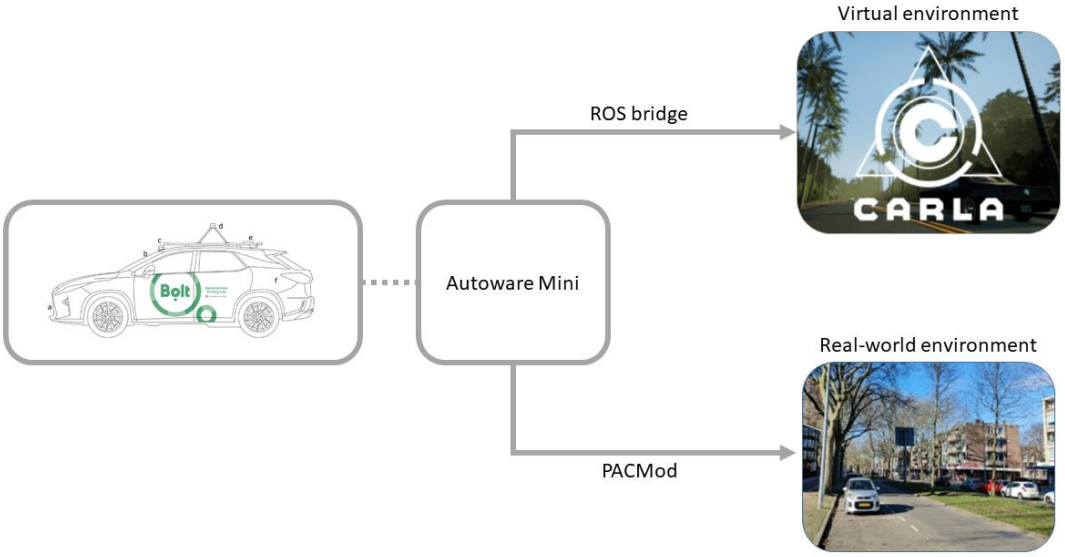

Figure 6. Connection of the autonomy software to the simulator and the real-world

The Autoware Mini autonomy software has been designed to be used in the simulator and the real world. The autonomy software connects to the virtual environment in CARLA using the ROS bridge package that enables two-way communication between both. The connection between the autonomy software and the real-world environment is enabled by PACMod[20], developed by AutonomouStuff. This setup is shown in Figure 6.

[20] https://autonomoustuff.com/velocity-magazine/velocity-2021/setting-a-driverless-course-for-the-future

3 Method

This section describes the method for setting up the simulation-based safety testing environment, the experiments' design, and the analysis of the data gathered.

Section 3.1 describes the method for simulation setup. The method describes implementing a scenario, setting up the simulation environment, and connecting the simulated ADS control in the CARLA simulator. The ADS and the scenarios are separate entities, although they are connected to the simulator.

Section 3.2 describes how to design experiments and measures for comparing the safety behavior of the ego vehicle.

Section 3.3 describes the method for a set of experimental simulation runs and the analysis of the results. The part describes what data to log and how to log it. Also, the analysis of the logged data is performed to conclude the validity of the simulator and the ADS.

A high-level method process is shown in Figure 7. If the simulation has not yet been set up, it must be set up first (Section 3.1). It includes the setup of the simulator, the autonomy software, and the scenario definition. Afterward, the experiment has to be designed (Section 3.2). Then, the experiments are implemented and executed. From the results, it is possible to analyze and draw conclusions (Section 3.3).

The process in Figure 7 can also be iterative and used for regression testing when executing the same experiment on different versions of the ADS system. To achieve this, a part of the experiment design and the experiment execution and analysis can be repeated several times. It provides a way to compare how the new changes have affected certain features and compare them against each other. Another approach for this method is to use it in exploratory testing. Introducing small changes in the system makes it possible to see how the change has affected specific scenarios. This result makes it possible to make new experiments once a deviation from the expected is found.

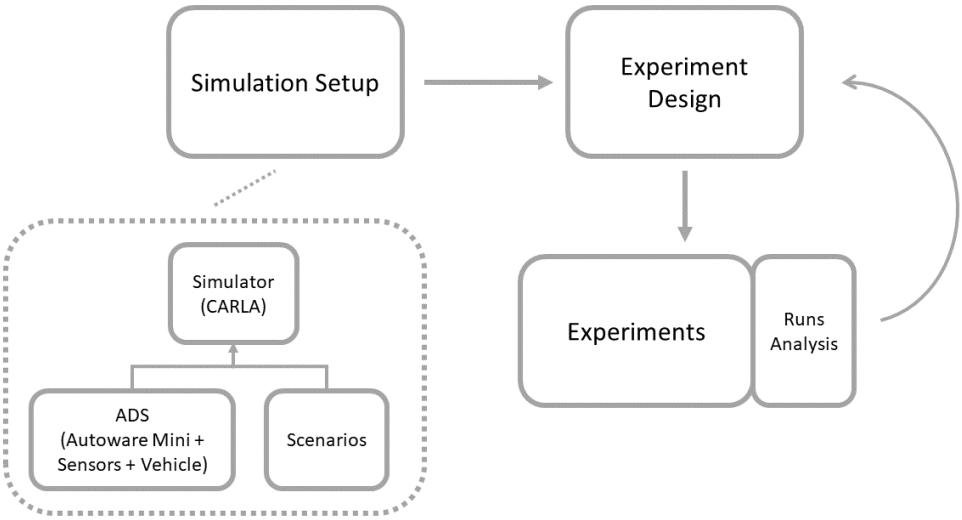

Figure 7. High-level schematic for simulation-based safety testing

3.1 Simulation Setup

This method has three parts: scenario implementation, simulation environment setup, and ego vehicle implementation.

Section 3.1.1 provides a detailed description of each part of the ASAM OpenSCENARIO format and its structure. This scenario format was chosen not only because it is one of the formats that the CARLA simulator can execute but also because it is a format that can be used by simulators such as AWSIM[21], MathWorks Automated Driving Toolbox[22], Prescan[23], and others.

Section 3.1.2 describes the simulation environment and how to implement it. CARLA simulator was chosen because it is one of the few comprehensive open-source simulators with sensor data processing, end-to-end driving policy design, and vehicle control [39].

Section 3.1.3 describes the connection and implementation of the ego vehicle. In addition, the simulator requires the ScenarioRunner, which prides itself on the scenario execution functionality, and ROS bridge[24], which allows to connect a third-party autonomy software that uses ROS.

The simulator (CARLA) simulates the sensors as input for the Autoware Mini autonomy software. The simulator takes the scenario definition and sets up the other participants of the road in the scenario. It also connects the ego vehicle controlled by Autoware Mini to the simulation.

3.1.1 Scenario Definition

The scenario is defined in the ASAM OpenSCENARIO 1.0 format. Currently, CARLA ScenarioRunner supports the OpenSCENARIO 1.0 standard [40]. The ASAM OpenSCENARIO 1.0.0 user guide [41] describes the concepts and scenario creation and presents examples. Every scenario has five mandatory components: *FileHeader*, *CatalogLocations*, *RoadNetwork*, *Entities*, and *Storyboard*. The optional component is *ParameterDeclarations*.

3.1.1.1 FileHeader

FileHeader has five mandatory attributes: *revMajor*, *revMinor*, *date*, *description*, and *author*. Figure 8 gives an example of how a *FileHeader* can be defined.

```
<FileHeader revMajor="1" revMinor="0" date="2023-11-01T12:00:00" descrip-
tion="Longitudinal control after leading vehicle's brake" author=""/>
```

Figure 8. Example code snippet for *FileHeader*

3.1.1.2 CatalogLocations

CatalogLocations is used to define one or multiple catalog locations: *VehicleCatalog*, *ControllerCatalog*, *PedestrianCatalog*, *MiscObjectCatalog*, *EnvironmentCatalog*, *ManeuverCatalog*, *TrajectoryCatalog*, *RouteCatalog*. Catalogs store reusable code for vehicles, controllers, pedestrians, miscellaneous objects, environments, maneuvers, trajectories, and routes. In Figure 9, an example of usage of the *VehicleCatalog* is given.

```
<CatalogLocations>
  <VehicleCatalog>
```

[21] https://github.com/tier4/AWSIM

[22] https://se.mathworks.com/help/driving/index.html

[23] https://plm.sw.siemens.com/en-US/simcenter/autonomous-vehicle-solutions/prescan/

[24] https://github.com/carla-simulator/ros-bridge

```
    <Directory path="catalogs" />
  </VehicleCatalog>
</CatalogLocations>
```

Figure 9. Example code using *VehicleCatalog*

3.1.1.3 RoadNetwork

RoadNetwork defines the *LogicFile* and the *SceneGraphFile*. They provide the road network description as *LogicFile* and the description of the environment as *SceneGraphFile*. If the map is already imported into CARLA, it is required only to give the map's name, as shown in Figure 10.

```
<RoadNetwork>
  <LogicFile filepath="Town01"/>
  <SceneGraphFile filepath=""/>
</RoadNetwork>
```

Figure 10. Example code of *RoadNetwork* definition for a map already imported into CARLA.

If the map is not yet in CARLA, it is possible to provide the paths to *LogicFile* and *SceneGraphFile*, as shown in Figure 11.

```
<RoadNetwork>
   <LogicFile filepath="../xodr/map.xodr"/>
   <SceneGraphFile filepath="../models/map.osgb"/>
</RoadNetwork>
```

Figure 11. Example code of *RoadNetwork* for a map that has not yet been imported into CARLA.

3.1.1.4 Entities

Entities consist of dynamic and static objects called *ScenarioObjects*. Each object has to be defined as either *Vehicle, Pedestrian,* or *MiscObject*.

ScenarioObject of category *Vehicle* requires such properties as *Performance, BoundingBox, Axles,* and *Properties*. Possible categories of vehicles: *car*, *van*, *truck*, *trailer*, *semitrailer*, *bus*, *motorbike*, *bicycle*, *train* or *tram*. An example of a defined ego vehicle is shown in Figure 12.

Category *Pedestrian* requires *BoundingBox* and *Properties*. Possible categories of pedestrians: *pedestrian*, *wheelchair,* or *animal*. An example of a *Pedestrian* definition is shown in Figure 13.

Category *MiscObject* defines other objects that are not vehicles or pedestrians. The types of *MiscObject* are *none*, *obstacle*, *pole*, *tree*, *vegetation*, *barrier*, *building*, *parkingSpace*, *patch*, *railing*, *trafficIsland*, *crosswalk*, *streetlamp*, *gantry*, *soundBarrier*, *wind*, or *roadMark*. An example of a *MiscObject* definition is shown in Figure 14.

```
<ScenarioObject name="hero">
  <Vehicle name="vehicle.lincoln.mkz_2017" vehicleCategory="car">
    <ParameterDeclarations/>
    <Performance maxSpeed="69.444" maxAcceleration="200" maxDecelera-
tion="10.0"/>
    <BoundingBox>
      <Center x="1.5" y="0.0" z="0.9"/>
      <Dimensions width="2.1" length="4.5" height="1.8"/>
    </BoundingBox>
    <Axles>
```

```
      <FrontAxle maxSteering="0.5" wheelDiameter="0.6" trackWidth="1.8" posi-
tionX="3.1" positionZ="0.3"/>
      <RearAxle maxSteering="0.0" wheelDiameter="0.6" trackWidth="1.8" posi-
tionX="0.0" positionZ="0.3"/>
    </Axles>
    <Properties>
      <Property name="type" value="ego_vehicle"/>
      <Property name="color" value="0,0,255"/>
    </Properties>
  </Vehicle>
</ScenarioObject>
```

Figure 12. Example code for a *ScenarioObject* of category *Vehicle*.

```
<ScenarioObject name="pedestrian1">
    <Pedestrian name="walker.pedestrian.0002" mass="90.0" model="walker.pedes-
trian.0002" pedestrianCategory="pedestrian">
        <ParameterDeclarations/>
        <BoundingBox>
            <Center x="0.0" y="0.0" z="0.9" />
            <Dimensions width="1" length="1" height="1.8"/>
        </BoundingBox>
        <Properties>
            <Property name="rolename" value="adversary" />
            <Property name="type" value="simulation" />
        </Properties>
    </Pedestrian>
</ScenarioObject>
```

Figure 13. Example code for a *ScenarioObject* of category *Pedestrian*.

```
<ScenarioObject name="Box">
  <MiscObject name="static.prop.box01" miscObjectCategory="obstacle"
mass="200.0">
      <ParameterDeclarations/>
      <BoundingBox>
          <Center x="0.0" y="0.0" z="0.75"/>
          <Dimensions width="1.0" length="2.0" height="1.2"/>
      </BoundingBox>
      <Properties>
      </Properties>
  </MiscObject>
</ScenarioObject>
```

Figure 14. Example code for a *ScenarioObject* of category *MiscObject*.

3.1.1.5 Storyboard

Storyboard consists of three elements: *Init*, *Story,* and *StopTrigger*. *Init* defines the *Action*(s) that need to be initialized before the scenario. *Story* implements *Act*(s) and *Maneuver*(s) within them to define the flow of the scenario.

Within the *Init* element, GlobalAction and PrivateAction are defined. *GlobalAction* defines global variables such as the environment for weather state, road conditions and time of day, entities for removing or adding entities, parameters for modifying values, infrastructure for modifying traffic signals, and traffic for population ambient traffic. *PrivateAction* defines the actions related to a specific object. For example, teleporting it, assigning a controller, setting a route for the vehicle, changing its longitudinal or lateral values, setting its visibility, and synchronizing it to a reference entity.

The *Story* consists of *Act*(s). Each *Act* has multiple *ManeuverGroup*(s), each of which refers to a set of *Entities*. Each *ManeuverGroup* has one or multiple Maneuver(s). Each *Maneuver*

consists of one or multiple *Event*(s). Each *Event* consists of an *Action* to be made, and a *StartTrigger* will start the Action once fulfilled. *Story* needs a *StartTrigger* and *StopTrigger* to know when to start and end the scenario. *StartTrigger* and *StopTrigger* consist of one or multiple *Condition*(s)

StopTrigger after the *Story* defines the criteria used to assess the finished state of the scenario execution with passed or failed results. This evaluation is displayed after the scenario finishes.

An example of a *Storyboard* can be seen in Appendix I, where an example of a scenario *FollowLeadingVehicle* is implemented.

3.1.1.6 ParameterDeclarations

With the help of *ParameterDeclarations,* it is possible to define a variable that will be used in the scenario and can be passed as a parameter when the scenario is executed. An example of *ParameterDeclarations* is shown in Figure 15.

```
<ParameterDeclarations>
  <ParameterDeclaration name="speed" parameterType="double" value="20.0"/>
  <ParameterDeclaration name="distance" parameterType="double" value="10.0"/>
</ParameterDeclarations>
```

Figure 15. An example of *ParameterDeclarations* with two parameters

3.1.2 Simulation Environment Setup

The simulation is executed in CARLA [34]. First, the requirements of CARLA have to be looked at before the installation, and it must be decided whether the device where the setup will be done is appropriate. Afterward, a version of the CARLA simulator has to be chosen. The installation steps are documented in the CARLA documentation[25]. When choosing the package installation, it is required to visit CARLA's GitHub page[26] and download the required version.

Once the installation is complete, creating a virtual Python environment and installing the requisite packages required to run the simulator is needed. PythonAPI is used to communicate with the simulation server. To install all the required packages, run these commands in the virtual environment:

```
pip3 install <wheel-file-name>.whl
pip3 install –r carla/requirements.txt
pip3 install –r examples/requirements.txt
pip3 install –r util/requirements.txt
```

To run the CARLA server, execute these commands:

```
cd path/to/carla/root

./CarlaUE4.sh
```

In order to test that it is possible to connect to the server using the API, execute these commands:

```
# Terminal A
python3 examples/generate_traffic.py

# Terminal B
python3 examples/manual_control.py
```

[25] https://carla.readthedocs.io/en/0.9.13/start_quickstart/#carla-installation
[26] https://github.com/carla-simulator/carla/blob/master/Docs/download.md

3.1.3 Ego Vehicle Implementation

In order to simulate an ego vehicle in the same way it is in the real world, a model must be made, and its specifications must be defined in the same way. It is required to define all the sensors the real-world vehicle has in the simulator.

To execute the scenario with an ego vehicle controller by Autoware Mini [37], we first need to install it. Before installation, ensure the prerequisites mentioned in the documentation are met. It is required to follow the installation process described in the documentation[27].

3.1.3.1 Implementation of the Ego Vehicle in the Simulator

Following the installation steps for Autoware Mini, there are additional steps to enable the ego vehicle in the simulator. In the Autoware Mini documentation,[28] steps are mentioned regarding the installation of additional assets, such as the ego vehicle modeled the same as the Lexus vehicle (described in Section 2.6) and a Tartu map developed by the ADL.

3.1.3.2 Implementation of Sensors in the Simulator

In order to simulate the ego vehicle accurately in the simulator, the vehicle configuration has to be the same. CARLA *ros_bridge* requires the sensors to be defined in a JSON format that defines the references to sensors. The Autoware Mini developers have defined the corresponding sensors[29] based on the real-life Lexus vehicle to be included in the simulator.

3.1.3.3 Implementation of Scenario Execution for the Autonomy Software

The following step is to set up ScenarioRunner based on the documentation[30]. The ScenarioRunner version has to be the same as for the downloaded CARLA simulator. Step 4 in the documentation shows how to test the setup and ensures that the default scenario can be run.

3.1.3.4 Connection between the Autonomy Software and the Simulator

The connection between the simulator and autonomy software, which is based on ROS, is enabled by *ros-bridge*[31] by CARLA. The documentation[32] explains more about the package.

Autoware Mini is based on ROS 1 (Noetic). The installation of ROS 1 and CARLA *ros_bridge* is already included in the setup steps[27] of the Autoware Mini autonomy software. Autoware Mini requires the default setup of *ros-bridge* module and includes it in a *launch* file. A *launch* file runs the ROS environment for a defined configuration of multiple nodes and parameters. Autoware Mini developers have defined a *carla.launch*[33] file where the required modules of *ros_bridge*, *scenario_runner*, CARLA and the sensor configuration (described in Section 3.1.3.2) are included.

If it is needed to be implemented from scratch for any other autonomy software, then it is possible to follow the guides for installing the required ROS package (ROS 1[34] and ROS

[27] https://github.com/UT-ADL/autoware_mini#installation
[28] https://github.com/UT-ADL/autoware_mini?tab=readme-ov-file#launching-carla-simulation
[29] https://github.com/UT-ADL/autoware_mini/blob/release/config/carla/sensors.json
[30] https://github.com/UT-ADL/autoware_mini#launching-with-scenario-runner
[31] https://github.com/carla-simulator/ros-bridge
[32] https://carla.readthedocs.io/projects/ros-bridge/en/latest/
[33] https://github.com/UT-ADL/autoware_mini/blob/release/launch/platform/carla.launch
[34] https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_installation_ros1/

2[35]). To prepare the ROS environment and run the ROS bridge, the guide[36] must be followed.

3.2 Experiment Design

In order to design the experiment for comparing the safety behavior of the ego vehicle in a chosen scenario, there are three steps needed to be done:

1. Selection of scenarios
2. Definition of reference modes (empirical and theoretical)
3. The scenario implementation and the parameters selection

3.2.1 Selection of the Scenario

In Section 2.4, two scenario catalogs and a standard are defined. The scenario has to be selected with the complexity and restrictions in the simulation and the autonomy software in mind. Each of the autonomy software has a defined ODD in which it operates. Only scenarios possible within the autonomy software's ODD can be created. For example, it is impossible to test that the autonomy software can perform an evasive maneuver to avoid a crash if that has not been implemented within the software itself.

Khan et al. [42] define a process to prioritize and select scenarios for the safety testing of an ADS. The author gives a step-by-step methodology for selecting a scenario catalog, enumerating ODD conditions and filtering based on them, grouping and filtering based on their limitations, and finally, prioritizing and selecting a set of scenarios.

3.2.2 Reference Modes

Generally, the results from testing in the simulators disclose the worst-case behavior. For example, whether the crash happened or quantitative values such as how many times the car ran the red light or how many times the car went out of its lane. These values do not provide enough details to evaluate the safety. The ADS might still behave incorrectly even without crashing, or the tests themselves are not complete enough to show a fault in the system.

In Figure 16, the hierarchy of the reference modes is shown. The reference modes can be divided into two main categories: empirical and theoretical. Empirical reference modes use the data from the real world, while theoretical reference modes take into account the physics-based formulas. Empirical modes can be divided based on the driver: ADS or human driver. The human driver reference modes can be divided into two parts: whether or not the human behavior (e.g., reaction time) is taken into account. The human driver reference mode is divided into two parts because it might not be required or valuable to record human-dependent behavior such as reaction time. It is due to the fact that it might be challenging to capture realistic human behavior if the participants of an experiment already know they are being observed.

The theoretical reference modes are defined using physics-based relations. Depending on the scenario, it is possible to define the behavior of the ADS using physics formulas (e.g., braking of the vehicle). These formulas also take into consideration such parameters as friction and reaction time. An advantage of using the theoretical reference modes is the possibility of parametrizing them and exploring the parameter space to understand the behavior better. More about the definition of theoretical modes can be found in 3.2.2.2.

[35] https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_installation_ros2/
[36] https://carla.readthedocs.io/projects/ros-bridge/en/latest/run_ros/

Depending on the use case, it is possible to compare the reference modes to each other. For example, the empirical ADS and human driver reference modes can be compared if both scenarios are identical. The empirical ADS and theoretical reference modes can be compared if the scenario is the same.

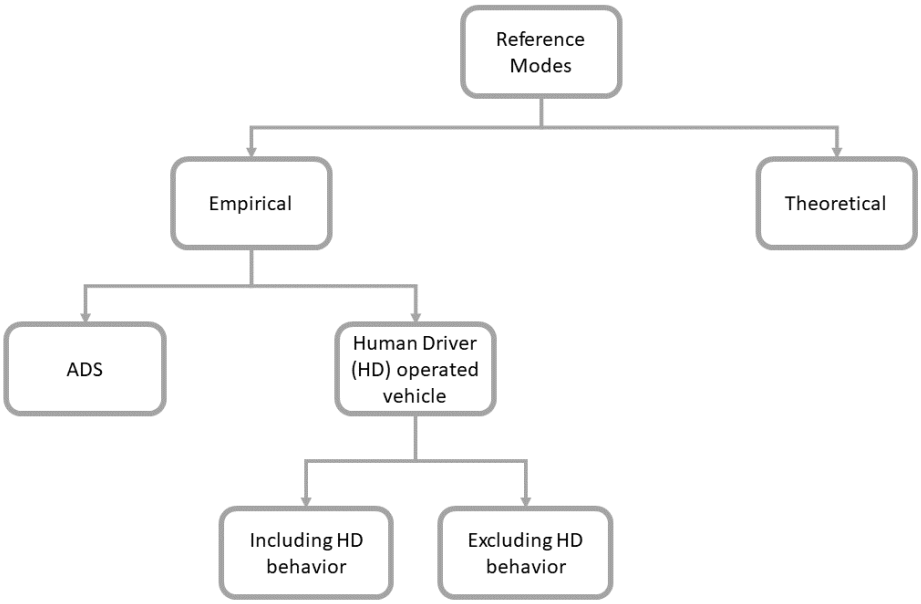

Figure 16. Hierarchy of reference modes

3.2.2.1 Uses for Reference Modes

To conclude the results from the simulations, one has to check whether the simulations are realistic regarding the theoretically *correct* behavior of the simulated ADS (e.g., braking behavior in an emergency situation) and the *correct* representation of the real-world ADS. For this purpose, we define reference modes to evaluate their *correctness*. *Correctness* is defined by the expected behavior of the ADS or the simulator.

One use case of reference modes is for validating the *correctness* of the simulation. This reference mode is derived from the real-life data from the vehicle being placed in real world environment. By performing the same experiment in both real life and the simulation, it is possible to analyze and compare the data to assert *correctness*.

Once the simulation is considered sufficiently *correct*, one can compare the behavior of the simulated ADS with that of human-driven vehicles. These reference modes are meant to check the degree of safety of ADS. For this, it is possible to use situations where the physical formulas are known. Acceleration and deceleration of a car, as well as forces while turning, such as angular velocity and angular acceleration, are some of the examples. Remember that these formulas should depend on parameters such as friction, which differ in various weather conditions.

It is not viable to assume that the simulation will be the same as defined in the reference mode. The reference modes are a good example of driving, and the simulation results are compared to the reference mode.

The definition of reference modes enables to realize how the ADS compares to the expected behavior in the real world and the physics formulas for a specific scenario. It provides an understanding of the ADS behavior in the simulator and the possibility of analyzing points for improvement.

3.2.2.2 Theoretical (Physics-based) Reference Modes

To define a theoretical reference mode, one must assess what kind of scenario is chosen and what actions and forces are applied at what time. For each of these actions, a physics formula should be applied. A different scenario must be chosen if the action is too complex or a physics formula cannot be found in the literature. These formulas must include the reaction time by adding the distance driven before action.

It is possible to define a benchmark for safe driving behavior of how an alert, sober, or distracted human driver behaves. Three different drivers have been defined:

- super-human driver,
- standard good driver,
- tired, distracted, or non-sober driver.

We differentiate them by changing the reaction time it takes to react to the event. A super-human has a 0-second reaction time, where the vehicle will act immediately. The information described in Section 2.2.2.1.3 shows that the average driver's reaction time varies depending on whether it is a static, dynamic, or unexpected task. To summarize the study results, an average concentrated driver's reaction time is 0.7 seconds for a static task and 0.8 for a dynamic task. From this, the better average reaction time is 0.7 seconds; hence, the reaction time for the standard good driver is 0.7 seconds. A distracted driver's reaction time increases to 1.7 seconds or even higher depending on the type of distraction. Similarly, a non-sober driver's reaction time increases. Blood alcohol concentration levels of 0.08% resulted in a 94% increase in reaction time, doubling the average driver's reaction time.

This data gives insight into the fact that the reference modes have to be defined based on the requirements. The reaction time varies due to the driver's task, age, concentration, and soberness. It is advisable to select more reference modes that introduce the varied reaction times of the drivers and select the reference modes that suit the situation to be analyzed. A sensitivity analysis can also be performed to determine how different values affect the outcome.

3.2.2.3 Automatic Assertion

Another way of interpreting and defining a reference mode is RSS [14]. In Section 2.2.2.1.2, the RSS model is described in detail.

This model can be used as a safety metric and formal verification of ADS safety. The minimum longitudinal and lateral distance is an automatic assertion model that can give feedback in real time or afterward. It provides feedback on whether the distance is safe or not.

According to Koopman [5], the issue with formal proofs is the realism of the assumptions for the real world. The assumptions about the front vehicle braking abilities, the slope, and the friction can change drastically. The factors can be accounted for, although the vehicle's behavior is too conservative if the worst-case situation is considered. Eventually, there is uncertainty in the assumptions.

The RSS model is implemented in the CARLA simulator, although it is under-development as of version 0.9.13[37]. Currently, CARLA does not support the RSS model entirely. Only the first two rules are fully implemented. After exploring the RSS module in CARLA, it was decided not to use it for the experiments, as it was unstable and raised unexpected errors.

[37] https://carla.readthedocs.io/en/0.9.13/adv_rss/

3.2.3 Scenario Implementation

The scenario chosen can be implemented in the OpenSCENARIO format, as it complies with the scenario type that the CARLA simulator can execute. The OpenSCENARIO definition is described in Section 3.1.1. The parameters should be chosen according to what is needed to be varied. The variation can be in the speed, distance from the vehicle, or any other scenario parameter.

When implementing a scenario, the ego vehicle requires a starting position and the vehicle definition to be defined in the scenario.

3.3 Experimentation and Analysis

The section explains the setup of the experiment, the type of data that can be gathered, and the analysis to draw conclusions about the validity of the simulator and the ADS.

3.3.1 Setup of the Experiment

The experiments are executed using a simulator (i.e., CARLA), a package enabling the execution of scenarios (i.e., ScenarioRunner), and autonomy software (i.e., Autoware Mini).

At the beginning of the experiments, a version of the autonomy software of the ADS has to be fixed. It enables the comparison of the same version of the autonomy software if experiments have been done to compare different parts of the software. It also enables to compare the versions of the autonomy software if the same experiment was performed on different versions.

The methodology for executing the experiments has to be set according to the scenario. Depending on the scenario's parameters, the initial values or a set of values have to be set. If the variable is independent, it is required to determine the set of values for the experiments. If the parameter is dependent, an initial value is determined for each independent value and adjusted during the experiments. For example, if the braking distance to an unexpected object on the road is measured, the speed and distance parameters from the object to the vehicle must be set. The set of experiments is performed at each speed, for example, 10 km/h, 20 km/h, and 30 km/h. For an emergency braking scenario, exploratory experiments have to be performed where the distance after the vehicle stops and the objects on the road have to be smaller than one meter. The trigger distance value must be adjusted if the distance is greater or a crash happens.

3.3.2 Data Gathering

After every execution, the data from the scenario run in the simulator can be logged as a CSV file with a unique name. Once the scenario has started, data such as time, ego vehicle x, y, and z coordinates, their speed, distance driven, and longitudinal, lateral, and Euclidean distances between each other can be written into the file.

It is also possible to log the data that is present in the autonomy software. ROS framework publishes a set of topics that provide the transport of data between various modules. It is possible to log the data and analyze it according to the needs of the scenario.

3.3.3 Data Analysis

The analysis of the experiments is drawn using the reference modes and experiment executions.

From the scenario executions, it is possible to construct graphs with up to 3 different parameters defined in the scenarios. Using the graphs, it is possible to analyze and compare how

the autonomy software behaved with respect to the reference modes. With the comparison, we can analyze whether the autonomy software is safer than a human-based reference mode.

To assess the safety and correctness of ADS, it is required to analyze and determine whether the autonomy software is acting safely, as defined by the safety measures. This information can be used to modify the algorithm of the Autoware Mini to improve safety in various situations.

To validate the correctness of the simulator, the trustworthiness of the simulation must be evaluated by comparing the results of real-world experiments and the same experiment performed in the simulator.

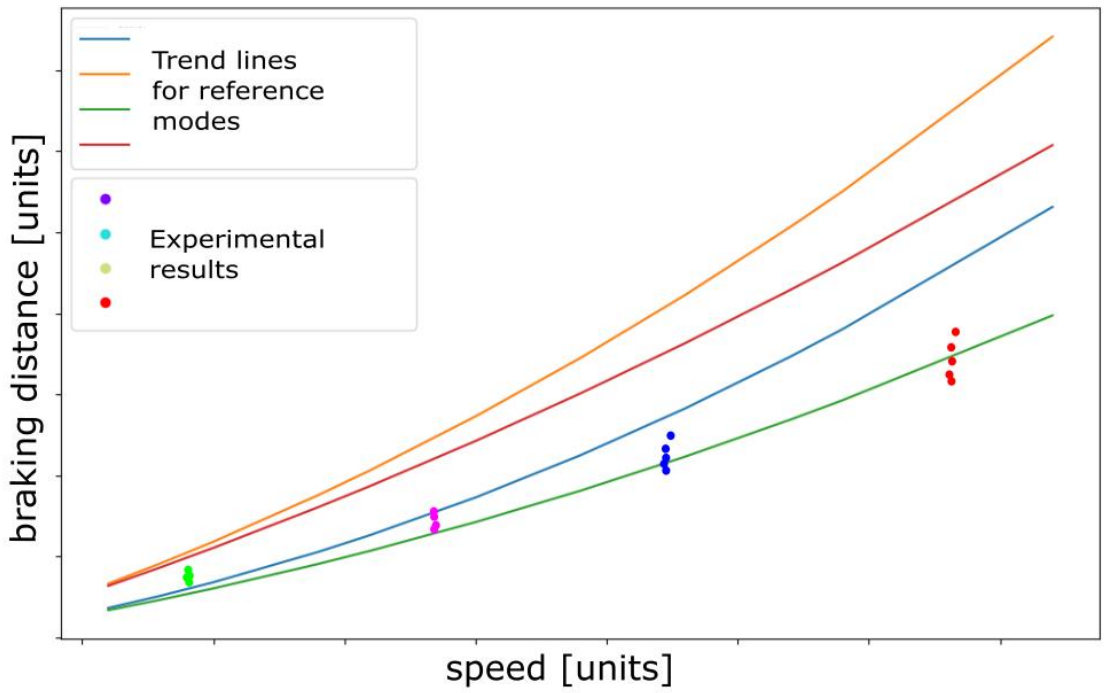

Figure 17. Example of the reference modes (solid lines) and the experiment results (dots)

In Figure 17, an example of results from the experiments and the reference modes are displayed in the same graph. It is possible to compare the experiment data to a particular reference mode and analyze how they differ. Analyzing the data trend and error values for each set of experiments is possible. The example shows that the experiments perform similarly to the green reference mode, although it performs worse for smaller speeds on the x-axis. It also can be observed that the variance of the braking distance values on the y-axis varies more when the independent value (x) is increased. The values on the x and y axis can be changed according to the scenario selected and the data to be analyzed.

A quantitative comparison is required to compare the simulation result to the reference mode. One possible way is to calculate a mean point for the cluster of one speed and compare it to the reference modes by taking the exact value of the x-axis. Statistical measures such as deviation, variance, and standard deviation can be used for this purpose.

4 Experiments

Based on the method in Section 3, two experiments with an emergency braking scenario were conducted. The only difference between the experiments was the version of Autoware Mini (denoted as V1 and V2). Both of them were executed in the simulator CARLA.

Experiment \ Reference Mode	**SimV1 (output) (4.3)**	**SimV2 (output) (4.3)**
Theoretical Reference Modes (4.2.1)	✓	✓
Empirical Reference Mode: Manual Braking (4.2.2)	Not comparable	✓
Empirical Reference Mode: Virtual Object (4.2.2)	Not comparable	✓
Empirical Reference Mode: Test Dummy (4.2.2)	Not comparable	✓
Empirical Reference Mode, including reaction time	Not executed	Not executed

Table 3. An overview of the experiments and reference modes and whether the experiment results and the reference mode are comparable.

An overview of experiments and reference modes is shown in Table 3. The checkmark shows that the experiment result and the reference mode are comparable. In Section 4.3, the SimV1 and SimV2 simulation experiments are described. In the subsections, the selection and implementation of the scenario, the experiment execution, data gathering, and the results from the simulation experiments are described.

For both simulation experiments, several reference modes were defined. Six theoretical reference modes were defined based on the scenario performed in SimV1 and SimV2 (4.2.1). The theoretical reference modes are comparable to both SimV1 and SimV2. Based on the scenario defined in Section 4.1, three empirical reference modes were defined (4.2.2), and data-gathering sessions were planned and executed. Empirical reference modes – Manual Braking, Virtual Object, and Test Dummy - are not comparable to SimV1 because the version of the Autoware Mini is different. The version for SimV2 is the same as for the empirical reference modes mentioned above. The empirical reference mode, which includes reaction time of a human driver, was not executed due to the inability to measure a human's reaction time. In Section 4.4, a comparison of the simulation experiment results and the reference modes defined is performed and described.

The repository with the code for experiment execution and data analysis together with the data gathered from simulation experiments and real-world data-gathering sessions can be found on GitLab[38].

4.1 Scenario Selection

The scenario selection was based on the complexity of the scenario and the autonomy software's ODD. It was decided to start with one of the scenarios that an autonomous vehicle has to encounter and be prepared for to ensure safety. The scenario involves an emergency braking when a pedestrian steps onto the road in front of the vehicle.

4.2 Reference Modes

In order to analyze the results, it is necessary to define reference modes, as discussed in Section 3.2. In relation to the scenario selected, it is possible to define both theoretical and empirical reference modes.

Section 4.2.1 defines the theoretical reference modes for the emergency braking experiments SimV1 and SimV2.

Section 4.2.2 defines the empirical reference modes for the emergency braking experiments SimV2.

The three empirical data-gathering sessions were performed in real life with the Lexus RX450h vehicle from the Autonomous Driving lab at the University of Tartu[39]. The emergency braking scenario was adapted due to safety issues so that it can be executed in real life.

4.2.1 TRM - Theoretical Reference Modes

The scenario performed in both SimV1 and SimV2 is an emergency braking scenario. The action that the ego vehicle performs in this scenario is braking. Total stopping distance D_{total} is defined by the sum of the distance driven during the reaction time (human driver) and the actual distance that it took to halt. The relation is defined in Equation 1, where v is the speed of the vehicle, t_r is the reaction time, μ is the friction coefficient, and g is the gravitational acceleration constant for Earth.

$$D_{total} = vt_r + \frac{v^2}{2\mu g}$$

Equation 1. Total Stopping Distance Formula

According to the Virginia Transportation Research Council [43], the average coefficient of friction for automobiles is 0.7, and for a vehicle equipped with an anti-blocking system, the friction coefficient is 0.9.

The reaction time can be defined by the alertness of the driver. Three drivers with various reaction times were defined: super-human, ideal human, and distracted human driver. Super-human has a reaction time of 0 seconds. Based on the dynamic task of reacting to an unexpected event, the reaction time for an ideal human is 0.7 seconds, and for a distracted human is 2 seconds.

Figure 18 shows the relation between the braking distance of a vehicle and the speed at the time of the braking start.

[38] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/tree/main
[39] https://adl.cs.ut.ee/lab/vehicle

Theoretical reference modes are independent of any simulation or software version.

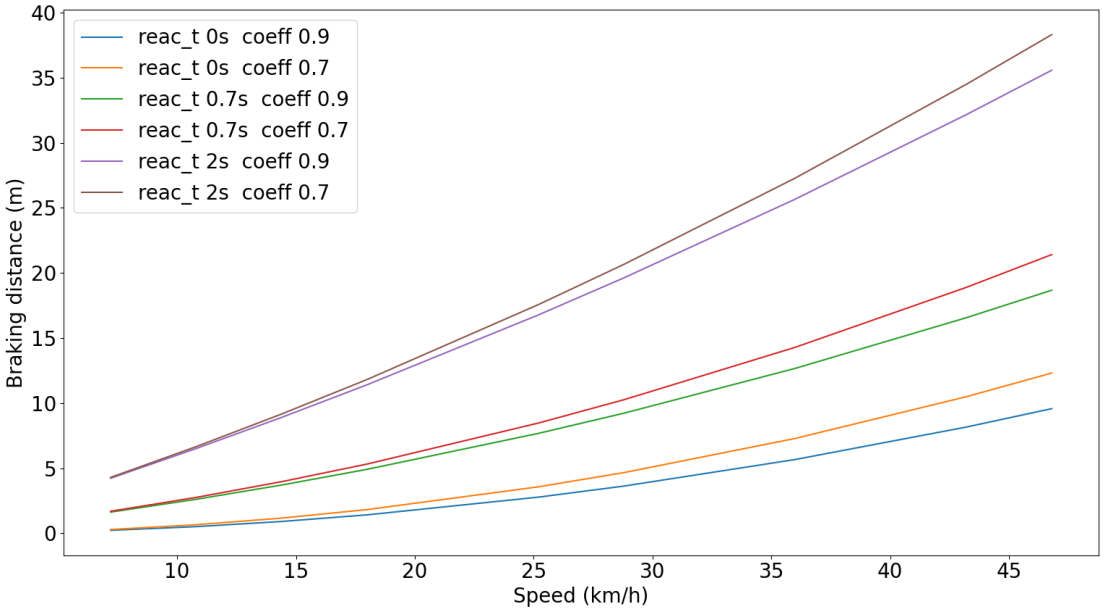

Figure 18. Braking distance against the speed when the braking started.

4.2.2 ERM - Empirical Reference Modes

Three empirical reference modes were designed, and the data-gathering sessions were performed for each.

The first empirical reference mode of manual braking (ERF_MB) aims to measure the maximum capability of the vehicle to brake in an emergency situation.

The second empirical reference mode of virtual obstacle (ERF_VO) aims to measure the braking capability of the ADS in an emergency situation without object detection.

The third empirical reference mode of the test dummy (ERF_TD) aims to measure the braking capability of the ADS in an emergency situation with all systems active.

The data gathering session was performed with the help of ADL members Kertu and Edgar as safety driver and software operator, respectively. The safety driver is responsible for monitoring the actions of the ADS and reacting if required. The software operator observes that the autonomy software is working as expected.

For the results from the empirical reference mode data-gathering sessions to be comparable to simulation experiments, both have to use the same Autoware Mini version.

4.2.2.1 Setup

The data-gathering sessions were performed in a private area, isolated from traffic. The system used was ADS in the Lexus RX450h vehicle (described in Section 2.6). Additionally, Autoware Mini was used for virtual object and test dummy sessions. The Autoware Mini version used for experiments is based on the code[40] committed on the 16th of April, 2024.

4.2.2.2 Scenario Design and Description

The scenario design and description are presented for three empirical reference modes.

[40] https://github.com/UT-ADL/autoware_mini/commit/cb49cc056a45082be64e1ed44699235f88806862

ERF_MB

As the scenario aims to measure the maximum braking capability in an emergency situation, elements such as the autonomy software should not be used for braking. Instead of using the autonomy software to brake, the safety driver is required to brake harshly to simulate an emergency situation.

ERF_VO

The scenario aims to measure the braking capability of the ADS in an emergency situation without object detection. When object detection is not used, such modules as the LiDAR scans with ground removal and clustering of LiDAR points are removed. The removal gives an advantage of the obstacle always being detected by the tracking module and decreases the possibility of errors from the perception module.

This is achieved by simulating an object in front of the vehicle by directly adding it to the object tracking module. The emergency braking situation is achieved by allowing the ADS to react to the object using a range filter for detected objects.

ERF_TD

The scenario aims to measure the braking capability of the ADS in an emergency situation. This reference mode is defined to be the same as the simulation experiment SimV2; only the environment changes from the simulation to the real world.

To achieve a safe data-gathering session, the obstacle detection width was adjusted. Instead of the usual 1.35 meters of lateral stopping distance, it was changed to 3.2 meters to widen the obstacle detection and allow to push the dummy into the zone not directly in front of the vehicle. In addition, this removes any action that the in-built pre-collision system of Lexus RX450h might initiate (described in Section 2.6.1).

4.2.2.2.1 Functional Scenario

The functional scenarios of three empirical reference modes are presented in Table 4. The main differences are highlighted in bold.

ERF_MB	ERF_VO	ERF_TD
The vehicle is placed on a straight and empty road. The vehicle starts driving. Once the desired speed is reached, the **safety driver brakes with full capacity** to simulate an emergency braking situation.	The vehicle is placed on a straight and empty road. A **virtual object is placed** in the software (e.g., RViz) within a distance that allows the vehicle to speed up to the desired speed. **A range filter for detected objects is enabled**. Once the object is set, the vehicle accelerates. **When the object is detected, the ego vehicle is expected to brake.**	The vehicle is placed on a straight and empty road. The vehicle starts driving. **The dummy is pushed onto the road** and stands in the detection lane. **The ego vehicle is expected to perform emergency braking** so as not to hit the dummy.

Table 4. Functional scenarios of empirical reference modes ERF_MB, ERF_VO and ERF_TD

4.2.2.2.2 Logical Scenario

The logical scenarios of three empirical reference modes are presented in Table 5. The main differences are highlighted in bold.

	ERF_MB	ERF_VO	ERF_TD
Weather	Sunny without precipitation	Sunny without precipitation	Sunny without precipitation
Static objects	The static real-world environment consisting of a straight road	The static real-world environment consisting of a straight road	The static real-world environment consisting of a straight road
Dynamic objects	Ego vehicle – moving along a straight road	Ego vehicle – moving along a straight road	Ego vehicle – moving along a straight road **Pedestrian (dummy)** – standing on a sidewalk, afterward pushed onto the road.
Parameters	The vehicle speed at the beginning of braking	The vehicle speed at the beginning of braking	The vehicle speed at the beginning of braking **The distance between the ego vehicle and the dummy** when it has to be pushed onto the detection lane
Pre-conditions		**A virtual object is placed in the software** (e.g., RViz) within a distance that allows the vehicle to speed up to the desired speed. **A range filter for detected objects is enabled**. The range is set based on the speed according to theoretical or simulated braking values.	

Description	1. The ego vehicle is moving along a straight road with a pre-defined speed 2. Once the desired speed is reached, the **safety driver is signaled to brake** 3. The braking is in process until the vehicle stops	1. The ego vehicle is moving along a straight road with a pre-defined speed 2. The **object is detected when it enters the range of detection**	1. The ego vehicle is moving along a straight road at a pre-defined speed. 2. When the required distance between the ego vehicle and the pedestrian is reached, **the dummy is pushed into the detection lane of the vehicle**. 3. Once the dummy enters fully the pre-defined detection lane, it is stopped.

Table 5. Logical scenarios of empirical reference modes ERF_MB, ERF_VO and ERF_TD

4.2.2.2.3 Concrete Scenario

The concrete scenarios of three empirical reference modes are presented in Table 6. The main differences are highlighted in bold.

ERF_MB	ERF_VO	ERF_TD
The three parameters of speed were chosen: 10 km/h, 20 km/h, and 30 km/h. Higher speeds were not chosen due to the discomfort caused by the forces during an emergency braking to the safety driver and software operator.	The three parameters of speed were chosen: 10 km/h, 20 km/h, and 30 km/h. Higher speeds were not chosen due to the discomfort caused by the forces during an emergency braking to the safety driver and software operator. **The calculated theoretical values from TRF determine the initial distance between the ego vehicle and the virtual obstacle**.	The three parameters of speed were chosen: 10 km/h, 20 km/h and 30 km/h. Higher speeds were not chosen due to the discomfort caused by the forces during an emergency braking to the safety driver and software operator. **The initial distance between the ego vehicle and the dummy** when it is being pushed onto the road is determined by the **previous data-gathering session (ERF_VO).**

Table 6. Concrete scenarios of empirical reference modes ERF_MB, ERF_VO and ERF_TD

4.2.2.3 Design and Execution

The data gathering session design and execution are presented for three empirical reference modes.

ERF_MB

Each scenario (10 km/h, 20 km/h, and 30 km/h) was executed twice to average the results gathered.

ERF_VO

For each speed variable (10 km/h, 20 km/h, and 30 km/h), a different detection range was set. The initial distance from the virtual obstacle to the vehicle was defined by theoretical reference mode (Section 4.2.1) using a reaction time of 0.7 seconds and friction coefficient of 0.7. An additional one meter was added for the expected stopping distance to the obstacle. It was adjusted smaller if the vehicle stopped farther than one meter from the virtual obstacle. It was increased if the virtual obstacle was hit. The initial and final distances to the virtual object for each speed can be seen in Table 7.

No.	Speed (km/h)	Virtual obstacle initial distance to the vehicle once detected (m)	Virtual obstacle final distance to the vehicle once detected (m)
1	10	2.5 + 1	2.5
2	20	6.1 + 1	6.5
3	30	16.8 + 1	12.5

Table 7. The initial and final distance to the virtual obstacle from the ERF_VO gathering session

ERF_TD

For each speed variable (10 km/h, 20 km/h, and 30 km/h), the initial distance between the ego vehicle and the dummy when it is being pushed onto the road is changed. The initial variable was defined by the previous data-gathering session (ERF_VO). It was adjusted smaller if the vehicle stopped farther than one meter from the dummy. It was increased if the dummy was hit. The initial and final distances to the dummy for each speed can be seen in Table 8.

No.	Speed (km/h)	Initial distance from the dummy to the vehicle (m)
1	10	2.5 + 1
2	20	6.5 + 1
3	30	12.5 + 1

Table 8. The initial distance to the dummy from the ERF_TD gathering session

4.2.2.4 Data Gathering

Data gathering was performed in the same way for all data-gathering sessions. Each of the execution of the scenario was recorded and saved in a file in *bag* format (described in Section 2.5.3). Reading the data from bags is possible using the Python package *bagpy*[41]. The code to read and save the bag files as *CSV* files is available on GitLab[42].

The goal of data gathering is to find the braking distance, braking time, and speed at the beginning of the braking. The braking time is found by subtracting the time when the braking started (TS1) from the time when the braking ended (TS2). TS1 can be found in the ROS topic /*ssc/brake_feedback* when the brake feedback value starts to increase. TS2 can be found in the ROS topic /*ssc/velocity_accel_cov* when the velocity decreases to 0. The speed at the beginning of braking can be found in the ROS topic /*ssc/velocity_accel_cov* in the column *velocity* at the timestamp TS1. Braking distance is found when calculating the Euclidean distance between coordinates when the braking started and when the braking ended. The x, y, and z position and orientation of the vehicle can be found in the ROS topics */localization/current_pose.* Taking the timestamps TS1 and TS2 from the time column makes it possible to find the x position from column *pose.position.x* and y position from column *pose.position.y* at both timestamps. Afterward, both positions were used in the Euclidean distance [44] formula to find the braking distance. Position on the z-axis is ignored because there was no elevation difference during the execution of the scenario. Orientation is ignored because the vehicle moved straight. An overview of the data required for analysis and where to find it is described in Table 9.

Data required	Topic name	How to find
Time at the beginning of braking (TS1)	/*ssc/brake_feedback*	When the value of the feedback starts to increase
Time at the end of braking (TS2)	/*ssc/velocity_accel_cov*	When velocity decreases to 0
Speed at the timestamp TS1	/*ssc/velocity_accel_cov*	In the column *velocity* at the timestamp TS1
Position of the vehicle at timestamp TS1	*/localization/current_pose*	In column *pose.position.x* at timestamp TS1
Position of the vehicle at timestamp TS2	*/localization/current_pose*	In column *pose.position.y* at timestamp TS1

Table 9. Braking distance and time data gathering from ROS *topics*

4.2.2.5 Analysis Approach

To better analyze the data gathered from the empirical data-gathering sessions, a curve can be fitted to the data points taken from the data. It enables to get continuous data from a discreet set of data points. For this purpose, regression analysis using the least squares method for non-linear data was used. It aims to fit a pre-defined function while minimizing the sum of squares of errors generated by the results of the associated equations [45]. Once the curve is obtained with its parameters, it can be used similarly as a theoretical reference

[41] https://github.com/jmscslgroup/bagpy
[42] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/blob/main/readBagFiles.py

mode and be compared with simulation results. It also can be used to analyze the quality of the results from the empirical gathering sessions. Using the same braking distance function as the theoretical reference mode to fit the curve, certain assumptions of the reaction time and the friction coefficient can be used to analyze the results.

An R-squared (R^2) measure can be used to analyze the regression model's goodness of fit. It is a statistical measure of how well the regression line approximates the actual data [46]. Equation 2 shows the formula to calculate R-squared where y_i is the i^{th} value of the sample, $f(x_i)$ is the predicted value for y_i, n is the number of total values for prediction, and $\bar{y}$ is the mean value of the sample. If the R-squared value is close to 1, the achieved fit is good.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Equation 2. R-squared measure for a regression model.

4.2.2.6 Braking Performance and Analysis

The braking performance is presented for three empirical reference modes.

ERF_MB

The results from the six runs are shown in Table 10. It shows that the average braking distance for a 10 km/h speed is 0.865 meters, for 20 km/h it is 2.69 meters, and for 30 km/h it is 5.615 meters. The deceleration for all runs vary from -5.43 m/s^2 to -6.22 m/s^2. The deceleration average is -5.81 m/s^2.

No.	Speed (m/s)	Speed (km/h)	Braking distance (m)	Average braking distance (m)	Deceleration (m/s^2)
1	2.93	10.54	0.82	0.865	-5.61
2	2.91	10.5	0.91		-5.96
3	5.76	20.72	2.74	2.69	-5.43
4	5.73	20.64	2.64		-5.56
5	8.6	30.96	5.68	5.615	-6.22
6	8.62	31.03	5.55		-6.09

Table 10. Results from ERF_MB data gathering session.

Based on the analysis approach described in Section 4.2.2.5, a curve was fitted to the data obtained, which can be seen in Figure 19. The optimal values for the parameters, reaction time, and friction coefficient were returned after utilizing the least squares method. A reaction time of 0.11 seconds and a friction coefficient of 0.81 was calculated. Using Equation 2, the R-squared value was calculated, which was 0.999099, indicating a perfect fit.

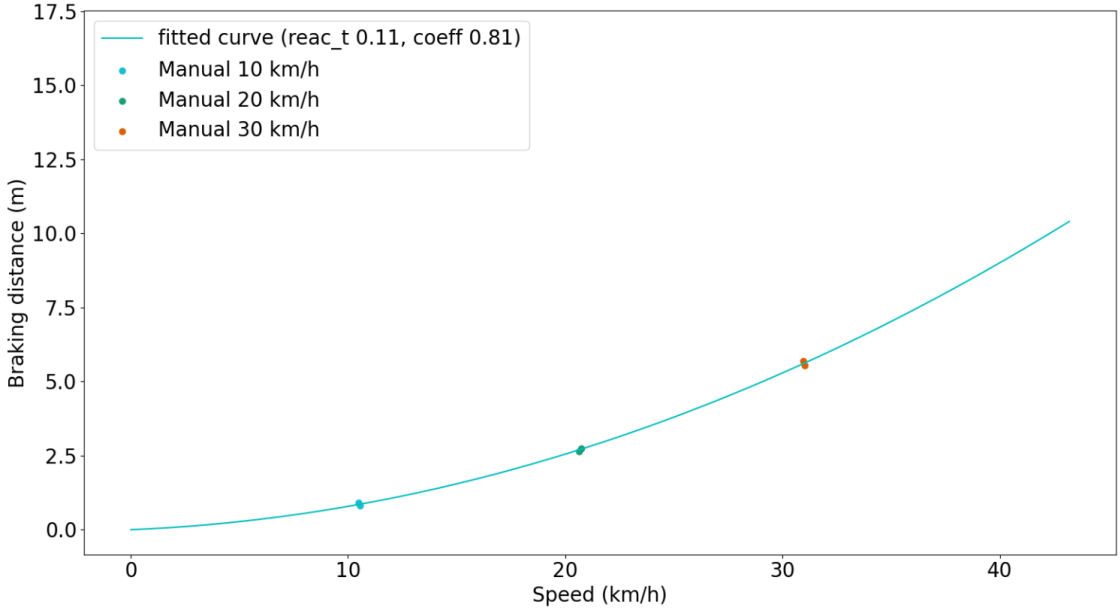

Figure 19. A fitted curve for the data from ERF_MB using the non-linear least squares method.

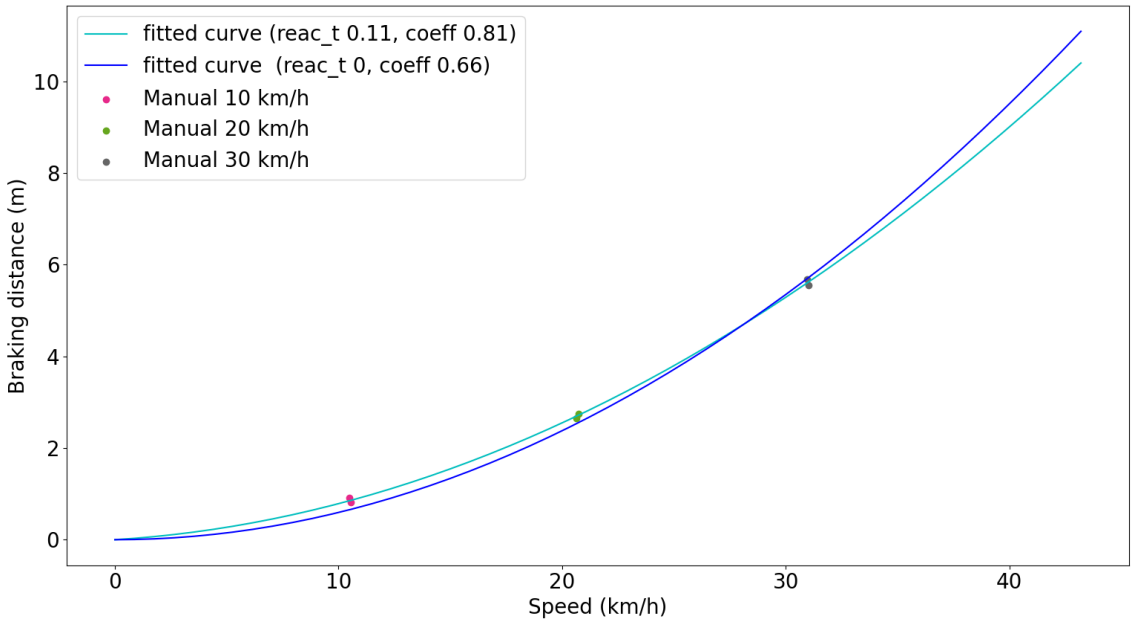

Figure 20. Two fitted curves with (dark blue) and without assumptions (light blue) for data from ERF_MB using the non-linear least squares method.

Figure 20 shows two fitted curves using the non-linear least squares method. The first curve is the same one that was shown in in Figure 19 and described above. The second curve was calculated under the assumption that for the ERF_MB reference mode where the human driver was braking, there was no reaction time involved because it was not measured. Only the actual braking distance was measured and recorded. Hence, the assumption that the reaction time parameter had to be 0 seconds was taken. The coefficient was calculated to be

0.66, and R-squared was 0.994737. It is also a good fit, although more data might be required to achieve the right fit. The data is variable due to minor errors in the localization, as the braking distance was calculated using the data from the localizer.

ERF_VO

The results from the three runs are shown in Table 11. It shows that the braking distance for 10 km/h speed is 1.5 meters, for 20 km/h it is 4.89 meters, and for 30 km/h it is 10.22 meters. The deceleration for all runs vary from -2.53 m/s^2 to -3.47 m/s^2. The deceleration average is -3.06 m/s^2.

No.	Speed (m/s)	Speed (km/h)	Braking distance (m)	Deceleration (m/s^2)
1	2.82	10.15	1.5	-2.53
3	5.79	20.85	4.89	-3.18
5	8.65	31.14	10.22	-3.47

Table 11. Results from ERF_VO data gathering session.

Similarly, as to the calculation above for ERF_MB, a curve was fitted to the data obtained, which can be seen in Figure 21. A reaction time of 0.19 seconds and a friction coefficient of 0.45 was calculated. Using Equation 2, the R-squared value was calculated, which was 0.99982, indicating a perfect fit.

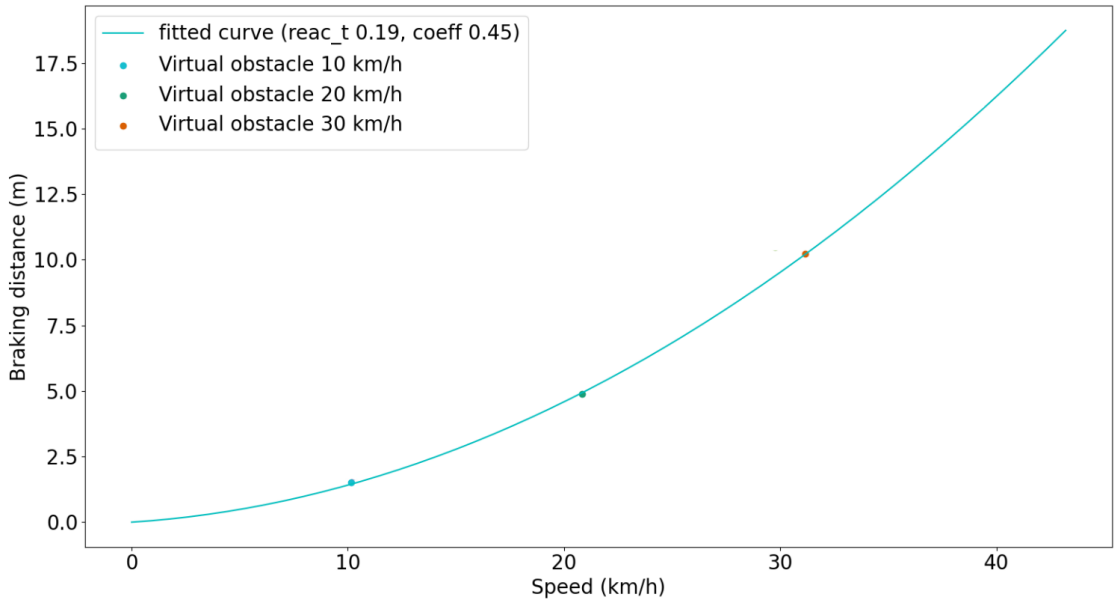

Figure 21. A fitted curve for the data from ERF_VO using the non-linear least squares method.

ERF_TD

The results from the three runs are shown in Table 12. It shows that the braking distance for 10 km/h speed is 1.55 meters, for 20 km/h it is 4.92 meters, and for 30 km/h it is 10.19

meters. The deceleration for all runs vary from -2.93 m/s^2 to -3.73 m/s^2. The deceleration average is -3.39 m/s^2.

No.	Speed (m/s)	Speed (km/h)	Braking distance (m)	Deceleration (m/s^2)
1	3.07	11.06	1.55	-2.93
3	5.91	21.3	4.92	-3.5
5	8.65	31.15	10.19	-3.73

Table 12. Results from ERF_TD data gathering session.

Similarly, as to the calculation above for ERF_MB, a curve was fitted to the data obtained, which can be seen in Figure 22. A reaction time of 0.11 seconds and a friction coefficient of 0.41 was calculated. Using Equation 2, the R-squared value was calculated, which was 0.999884, which indicates a perfect fit.

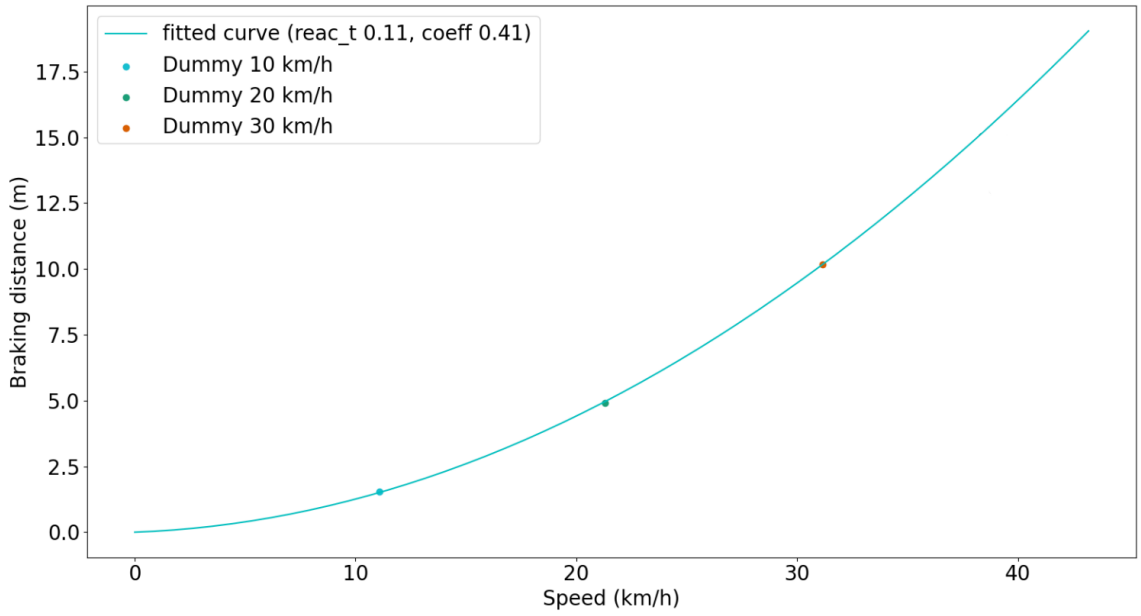

Figure 22. A fitted curve for the data from ERF_TD using non-linear least squares method.

4.2.2.7 Comparison of the Reference Modes

In Figure 23, the empirical reference modes and the theoretical reference modes can be compared. The ERM_MB can be compared to the theoretical reference mode with a reaction time of 0 seconds and a friction coefficient of 0.7. The low friction coefficient could be because the vehicle had winter tires with studs, which might decrease braking performance on non-icy roads.

ERM_VO and ERM_TD are very similar, and they can be compared to theoretical reference mode with a reaction time of 0.7 seconds and friction coefficient of 0.9.

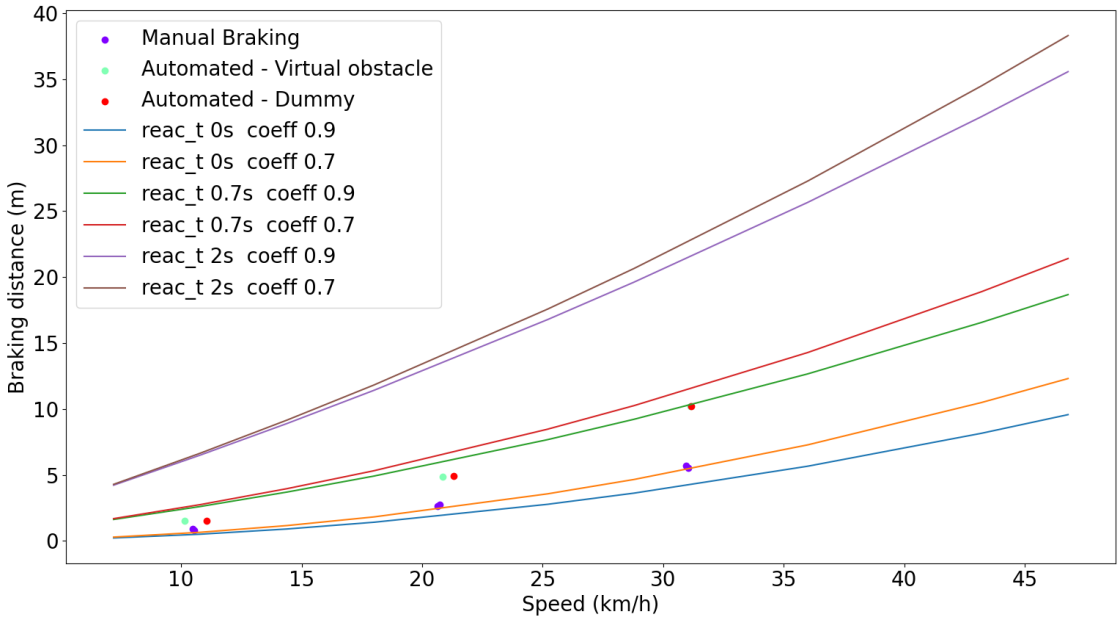

Figure 23. Empirical reference modes ERM_MB, ERM_VO, and ERM_TD (dots) and theoretical reference modes TRF (solid lines) are visualized.

4.3 Safety Evaluation Experiments for Emergency Braking Scenario

The experiments were designed to test the autonomy software capabilities in an emergency situation where the vehicle brakes suddenly when a pedestrian steps onto the road in front of the vehicle. The scenario selection is described in Section 4.1.

There were two experiments performed. The first experiment (SimV1) was designed and executed as described below. The second experiment (SimV2) was designed the same way; only the version of Autoware was changed.

A different Autoware Mini version was used, and the main difference was that the values for the proportional-integral-derivative[43] (PID) controller were changed to improve the vehicle's ability to keep a constant speed. The file that stores the parameters that can be changed for the PID controller can be seen on GitHub[44]. The changes made to the parameters can be seen in Table 13. As the autonomy software is constantly developing, other areas of the software have also been improved. The SimV1 experiment uses the Autoware Mini version based on the code[45] committed on the 23rd of March, 2023. The SimV2 experiments use the Autoware Mini version based on the code[46] committed on the 16th of April, 2024.

SimV1	SimV2
accel_Kp: 0.11 accel_Kd: 0.045	accel_Kp: 0.05 accel_Kd: 0.05

Table 13. The differences for the PID controller parameters in SImV1 and SimV2.

[43] https://www.ni.com/en/shop/labview/pid-theory-explained.html
[44] https://github.com/UT-ADL/autoware_mini/blob/cb49cc056a45082be64e1ed44699235f88806862/config/carla.yaml#L7
[45] https://github.com/UT-ADL/autoware_mini/commit/07be15044f5e3481bd787e03a87aedb5c21f0fd8
[46] https://github.com/UT-ADL/autoware_mini/commit/cb49cc056a45082be64e1ed44699235f88806862

4.3.1 Scenario Implementation

This scenario can be implemented in several ways. The main goal is to have an object in front of the autonomous vehicle appear suddenly. It can be implemented by moving various-sized objects onto the path of the autonomous vehicle. These objects can be humans, animals, other objects on the street like garbage bins, or even tree branches. A human stepping quickly onto the road was chosen. It was due to the high frequency of occurrence of such scenarios.

4.3.1.1 Functional Scenario

The ego vehicle is driving on a straight road. The pedestrian runs on the road and stands in front of the ego vehicle. The ego vehicle is expected to perform emergency braking to avoid hitting the pedestrian.

4.3.1.2 Logical Scenario

Weather:

- Sunny without precipitation

Static objects:

- The static environment consisting of a straight road

Dynamic objects:

- ego vehicle – moving along a straight road
- pedestrian – standing on a sidewalk, afterward running onto the road (in this scenario, running is defined as five m/s speed)

Parameters:

- distance between the ego vehicle and the pedestrian when the pedestrian starts the movement onto the road

Description:

1. The ego vehicle is moving along a straight road with a certain speed
2. When the parametrized distance between the ego vehicle and the pedestrian is reached, the pedestrian runs onto the road at a predefined speed.
3. Once the pedestrian reaches the middle of the lane where the ego vehicle is moving, it stops.

The scenario is implemented in the OpenSCENARIO format and is available on GitLab[47].

4.3.1.3 Concrete Scenario

The parameters are defined based on the theoretical calculations. The starting calculation is taken for an average good human driver with a reaction of 0.7 seconds and a vehicle with a friction coefficient of 0.9. As speed is the variable that changes in the formula, a different braking distance is calculated for each speed. For example, for a speed of 20 km/h, the braking distance of the average human driver would be 5.64 meters. It is the initial distance chosen for the parameter in the scenario based on the speed.

[47] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/blob/main/files/EmergencyBreakExp30.0.xosc

4.3.2 Experiments and Analysis

In this section, the experiment design, execution, and analysis of the results are described.

4.3.2.1 Implementation of the Experiments

As mentioned in Section 4.3.1.3, a speed has to be chosen for each experiment, and the pedestrian's initial trigger distance must be calculated. For this set of experiments, speeds of 15 km/h, 20 km/h, 25 km/h, 30 km/h, and 35 km/h were chosen. The formula from Equation 1 with a reaction time of 0.7 seconds and friction coefficient of 0.7 was used to find the initial trigger distance. Table 14 shows the initial trigger distances for each speed.

A set of exploratory experiments has to be performed to find the final trigger distance. Final trigger distance is the distance that the scenario[48] takes as an input for parameter *pedestrianTriggerLocation.* The initial trigger distance is taken as the first experiment, and the scenario is executed with the calculated stopping distance as the trigger distance parameter. If a crash happens, then the trigger distance has to be increased. Depending on the severity of the crash, the trigger distance can be decreased by 0.2 if the crash is not as severe or by a bigger number if the crash is more severe. The process must be repeated until the ego vehicle stops less than 1 meter from the pedestrian. If the ego vehicle stops less than 1 meter before the pedestrian, the vehicle has to engage the emergency brakes. If the distance is bigger, the ego vehicle has enough time to brake more smoothly. In this case, the trigger distance should be decreased based on the stopping distance from the pedestrian.

Once the final trigger distance is found for a specific speed value, the same has to be repeated for other speeds defined in the set of experiments. Table 14 lists the speed values chosen for the experiments and their corresponding initial and final trigger distances.

Speed (km/h)	Initial trigger distance (m)	Final trigger distance for SimV1 (m)	Final trigger distance for SimV2 (m)
10	2.5	-	5.5
15	4.18	8	-
20	6.14	11	11.5
25	8.38	15	-
30	10.89	19.5	19.5
35	13.7	23	-
40	16.78	-	26

Table 14. A list of speed values and their corresponding initial and final trigger distances for SimV1 and SimV2

[48] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/blob/main/files/EmergencyBreakExp30.0.xosc

The experiments can be executed after the final trigger distance is found for all the speeds. The speed for the autonomy software Autoware Mini can be set in the configuration file for planning[49]. It is possible to limit the global planner so that the maximum speed does not reach higher than the set value. For each of the speed values, ten experiments were executed.

4.3.2.2 Data Gathering

Depending on the requirement, the data can be gathered from two different sources: the CARLA server and the ROS framework.

4.3.2.2.1 CARLA Server

The CARLA server provides API based on Python to control the server and access the data from it. From the server, it is possible to get such data as the position (x, y, and z coordinates) and speed of each dynamic object. At every tick of the server, the data is written to a file together with a timestamp.

4.3.2.2.2 ROS Framework

Autoware Mini is an autonomy software that uses the ROS framework. This means that the framework publishes data available to be accessed on the computer.

One way to access data from ROS is through Python. The library *rospy*[50] is a Python client for ROS. It enables access to ROS topics, services, and parameters. Using this library, a listener was developed to access the data published by the autonomy software and log it to files.

First, the topics where the data is published have to be found. Using the command *rostopic list*, it is possible to find all the topics currently available in the environment. Once the scenario is executed, the command has to be executed and it will output the topics. Based on the output, the topics relevant to the scenario are selected. In Table 15, the chosen topics are listed. Each of the topics has a description stating what data it outputs.

ROS Topic	Description
/carla/ego_vehicle/speedometer	Ego vehicle speed (m/s)
/dashboard/acceleration	Acceleration of the ego vehicle (m/s^2)
/dashboard/closest_object_distance	Distance of the closest object that's detected in the path
/dashboard/target_speed	Target speed of the planner module (km/h)
/clock	Clock value from the start of the scenario execution (s)

Table 15. ROS topics and their description in Autoware Mini

In the Python code, first, a node has to be defined. Then, the chosen topics have to be subscribed to, and a callback is defined for each of the topics. After the scenario has finished,

[49] https://github.com/UT-ADL/autoware_mini/blob/2e17136cbcd2721777c006c0a6a2447454a7a929/config/planning.yaml#L13
[50] http://wiki.ros.org/rospy

an exit handler saves the data in corresponding files with a timestamp. The described code is accessible on GitLab[51].

In order to execute the listener and launch the scenario at the same time, a bash script was developed. It runs both actions in parallel and ends the processes if any actions are finished. This means that the logging will finish once the scenario execution is finished. The code described is accessible on GitLab[52].

4.3.2.3 Analysis Approach

If the set of experiments is not extensive, it is possible to analyze the logged data manually, as the data is logged in a human-understandable way. It is also possible to analyze the data using Python. For each of the scenario executions, data was logged into a file. Files containing information about the closest object distances at every time step are analyzed in a way where the start and end braking distances are gathered, and it is determined whether a crash happened or not. From the speed data, the speed at the time when the vehicle started braking is determined. In the end, *speed*, *braking distance*, and *crash* values are saved to a file. The data is separated into three files – the experiments containing crashes, emergency breaks, and others that do not fit the previous two categories. Once the data is analyzed, the data saved in the file containing emergency braking situations is used. The code that analyzes the data automatically is available on GitLab[53].

To analyze the data for each of the speeds, a cluster of experiment runs was visualized. For each of the cluster, a mean point $(\bar{x}, \bar{y})$ can be found. Equation 3 calculates the arithmetic mean for a set of points in Cartesian coordinates.

$$(\bar{x}, \bar{y}) = \left(\frac{1}{n} \sum_{i=0}^{n} x_i , \frac{1}{n} \sum_{i=0}^{n} y_i \right)$$

Equation 3. Arithmetic mean for Cartesian coordinates (x,y) in two dimensions

For each of the clusters, standard deviation σ of values on the x and y-axis were calculated to show the amount of variation from the mean of the cluster. Equation 4 calculates the standard deviation where x is the set of values the standard deviation is calculated for, n is the total number of values, and $\bar{x}$ is the mean of x values. Equation 5 calculates the standard deviation where y is the set of values the standard deviation is calculated for, n is the total number of values, and $\bar{y}$ is the mean of y values.

$$\sigma_x = \sqrt{\left(\frac{1}{n} \sum_{i=0}^{n} |x - \bar{x}|^2 \right)}$$

Equation 4. The standard deviation for a set of x values.

[51] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/blob/main/listener.py

[52] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/blob/main/files/runAutowareLoggingExp30.0.sh

[53] https://gitlab.cs.ut.ee/dalbina/experimentsthesis/-/blob/main/readLogsAndFilterData.py

$$\sigma_y = \sqrt{\left(\frac{1}{n}\sum_{i=0}^{n}|y-\bar{y}|^2\right)}$$

Equation 5. The standard deviation for a set of y values.

4.3.3 Braking Performance and Analysis

The braking performance for both experiments, SimV1 and SimV2, are described below.

SimV1

Table 16 shows the results from the experiment SimV1 runs for each set speed. The mean of the speed and the braking distance is calculated using Equation 3, while the standard deviation is calculated using Equation 4 for speed and Equation 5 for braking distance.

Speed set for experiments (km/h)	Speed		Braking	
	Mean speed (km/h)	Standard deviation	Mean braking distance (m)	Standard deviation
15	14.56	1.58	2.79	0.37
20	18.51	2.48	4.23	0.49
25	24.53	0.82	5.68	0.97
30	26.93	2.59	7.21	0.7
35	29.16	0.82	8.19	1.08

Table 16. Mean and standard deviation calculations for SimV1 experiment results

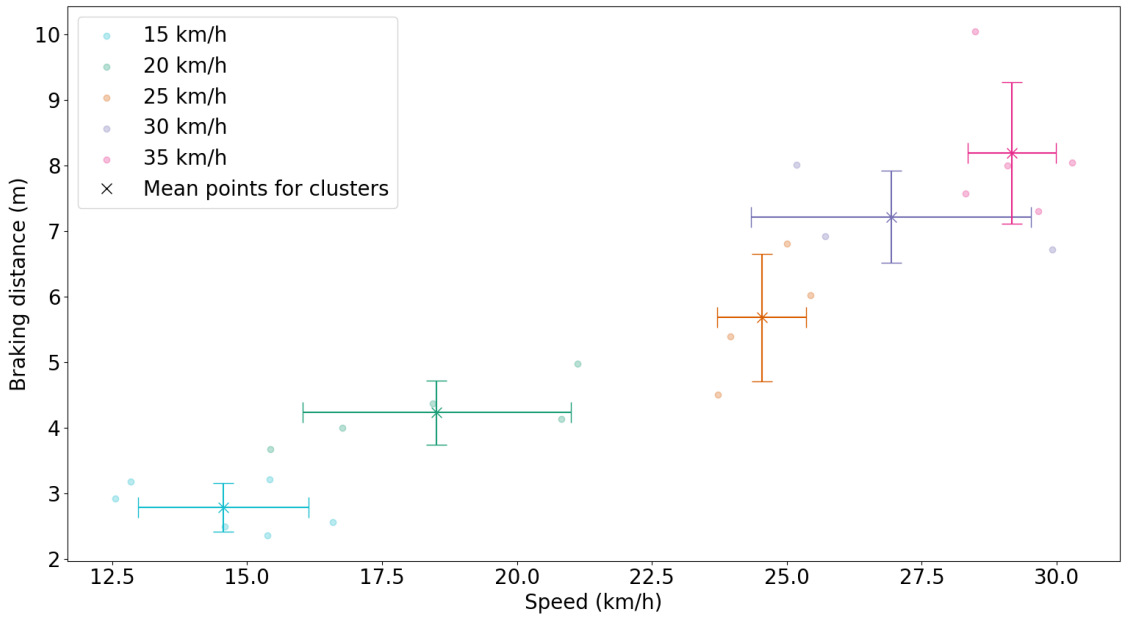

Figure 24. Experiment SimV1 results for braking distance depending on the speed

Figure 24 shows the results of the experiment runs (dots). The x-axis denotes the speed at the beginning of braking. The y-axis shows the braking distance of how many meters it took for the ADS to brake. The calculated values from Table 16 are visualized where each cluster's mean values (crosses) and the standard deviations (error bars) are shown. From the figure, it is seen that the speed of the ADS was unstable for every experiment. The standard deviation for both speed and braking distance is varied and does not seem to have any correlation. As the speed varies, the braking distance becomes very unstable. With the same trigger distance, the ego vehicle was performing differently because, at the time of the pedestrian's detection, the ego vehicle was driving at a different speed and either accelerating or decelerating.

Figure 25 shows how much the speed deviates while the vehicle tries to keep a constant speed of 30 km/h. The target speed (blue color) is set to 30 km/h, and the vehicle's current speed deviation is from 22 km/h to 36 km/h. As the controller is not able to keep the speed constant, each execution of the experiment differs, and the variation in the braking distance increases.

In addition, the ADS could not accelerate to 35 km/h and keep that speed due to fluctuations in the speed. Henceforth, the simulations performed at a speed of 35 km/h were actually around 30 km/h at the time of braking.

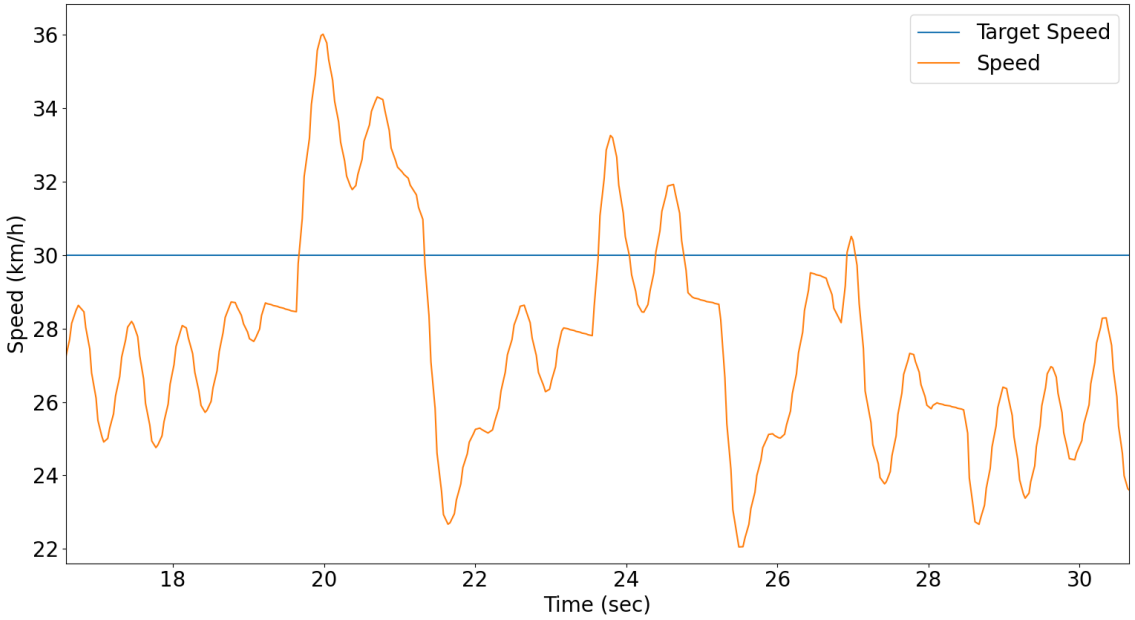

Figure 25. The deviation of speed for the ego vehicle in experiment SimV1 while keeping a constant target speed of 30 km/h.

SimV2

Table 17 shows the results from the experiment SimV2 runs for each set speed. The mean and standard deviation of the speed and the braking distance are calculated using Equation 3 and Equation 4, the same as for SimV1 experiments.

Speed set for experiments (km/h)	Speed		Braking	
	Mean speed (km/h)	Standard deviation	Mean braking distance (m)	Standard deviation
10	10	0.13	2.03	0.2
20	19.95	0.3	6.05	0.15
30	29.95	0.28	11.11	0.48
40	37.95	1.41	15.4	1.5

Table 17. Mean and standard deviation calculations for SimV2 experiment results.

Figure 26 shows the results of the experiment runs (dots). The x-axis denotes the speed at the beginning of braking. The y-axis shows the braking distance of how many meters it took for the ADS to brake. The calculated values from Table 17 are visualized where each cluster's mean values (crosses) and the standard deviations (error bars) are shown.

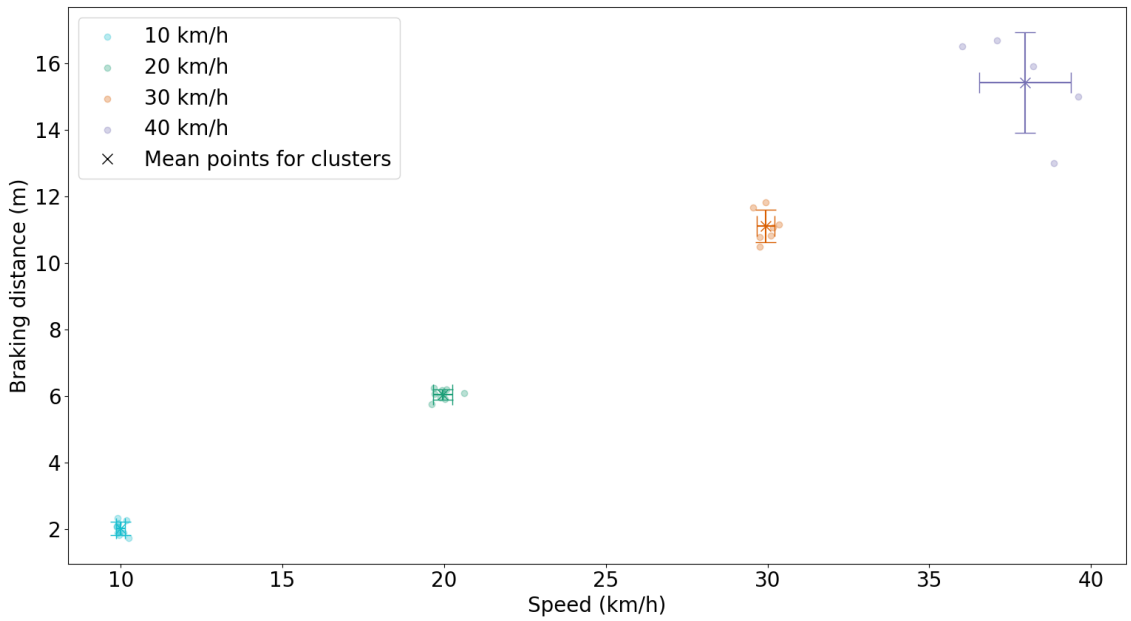

Figure 26. Experiment SimV2 results for braking distance depending on the speed.

In Figure 26, it is visible that the speed of the ADS was becoming unstable after 30 km/h. For speeds less than 30 km/h, it could keep a constant speed set in the planner module. For the clusters of speed less than or equal to 30 km/h, the standard deviation was lesser than for the 40 km/h cluster. When the speed was 40 km/h, a similar issue was observed for the experiment runs of SimV1, where the ADS could not keep the speed stable and fluctuating.

Figure 27 shows how much the speed deviates while the vehicle is trying to keep a constant speed of 30 km/h. The target speed (blue color) is set to 30 km/h, and the vehicle's current speed deviation is from 29.4 km/h to 30.6 km/h. Compared to SimV1 in Figure 25, it can be seen that the deviation of speed is much less and the controller can keep the speed more constant.

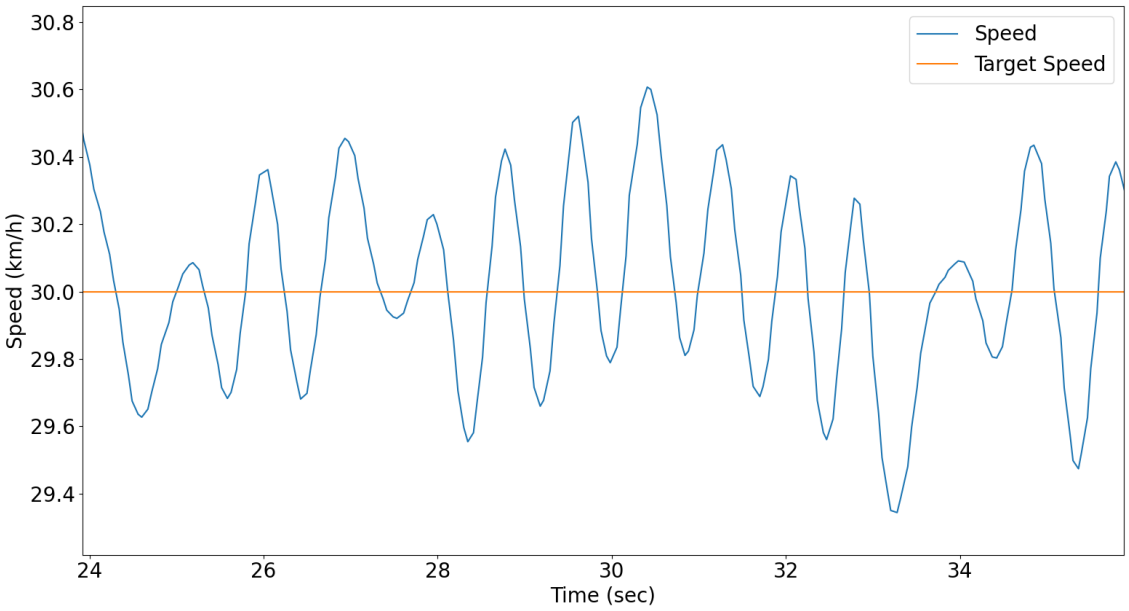

Figure 27. The deviation of speed for the ego vehicle in experiment SimV2 while keeping a constant target speed of 30 km/h.

4.4 Simulation Runs and Comparisons to Reference Mode

The comparison of simulation experiments SimV1 and SimV2 to the reference modes is described in this section.

4.4.1 SimV1 vs. SimV2

Figure 28 shows the results of the SimV1 and SimV2 experiments. On comparing the results from the SimV1 experiment to the results from the SimV2 experiment, it is seen that the braking distance for the SimV2 experiment has increased.

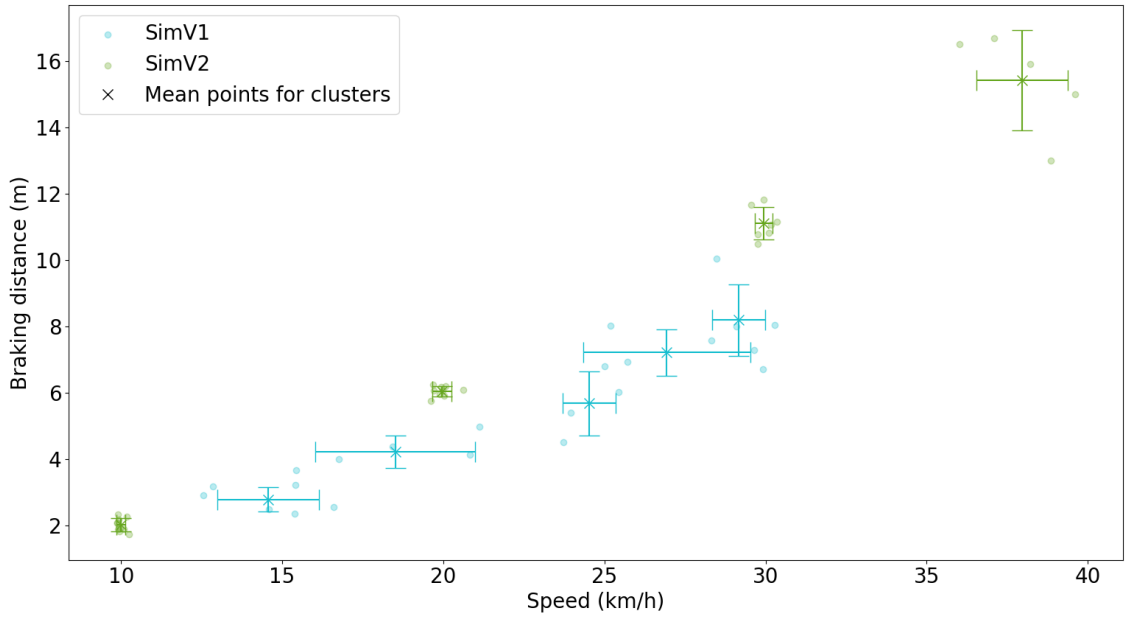

Figure 28. SimV1 and SimV2 experiment results.

On comparing the standard deviation of the clusters for both experiments, it is seen that the results from SimV1 have more variance in both speed and the braking distance for speeds under 30 km/h (including). The variance of braking distance for SimV2 results where the speed is set to 40 km/h is bigger than for any other result.

4.4.2 SimV1 vs. TRM

In Figure 29, results from the experiment SimV1 are shown together with theoretical reference modes. The results of the scenario execution are plotted using the dots. The six lines show the theoretical baselines with reaction times of 0, 0.7 and 2 seconds and friction coefficients of 0.7 and 0.9.

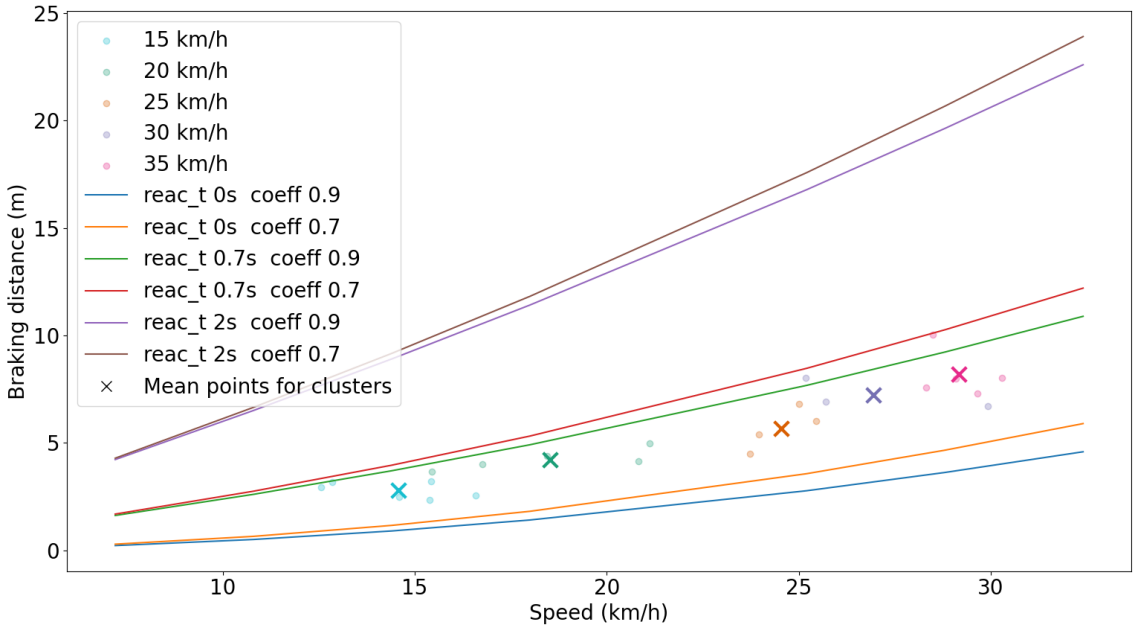

Figure 29. Experiment SimV1 results (dots) and theoretical reference modes TRM (solid lines).

To compare the simulation results to the reference modes more in detail, the deviation of braking distance d from a reference mode for each cluster of speeds is calculated. Equation 6 calculates the deviation of braking distance d from a reference mode where $\bar{y}$ is the mean braking distance for a particular speed, and y is the braking speed taken from the reference mode with the same speed (x) value calculated using the reference mode's formula. Table 18 shows the calculation results for SimV1 results, where the rows have the mean values of the cluster and the columns have a particular reference mode with its reaction time and friction coefficient.

$$d = y - \bar{y}$$

Equation 6. Deviation of braking distance from a reference mode

TRM \ Mean point of a cluster (x, y coordinates)	Reaction time (s)	0		0.7		2	
	Friction coefficient	0.7	0.9	0.7	0.9	0.7	0.9
(14.56, 2.79)		-1.6	-1.86	1.23	0.97	6.49	6.22
(18.51, 4.23)		-2.3	-2.73	1.29	0.87	7.97	7.55
(24.53, 5.68)		-2.3	-3.05	2.47	1.72	11.33	10.58
(26.93, 7.21)		-3.13	-4.04	2.1	1.2	11.83	10.92
(29.16, 8.19)		-3.41	-4.47	2.26	1.2	12.79	11.73

Table 18. Deviation of braking distance for a cluster from a reference mode for SimV1

Using Table 18, it can be observed that the results from simulation experiments can be compared the closest to the reference mode with a reaction time of 0.7 seconds and a friction coefficient of 0.9. The deviation from the reference mode is the smallest for all clusters.

4.4.3 SimV2 vs. TRM

In Figure 30, results from experiment SimV2 are shown together with theoretical reference modes. It can be observed that the results from simulation experiments can be compared to the reference mode with a reaction time of 0.7 seconds and a friction coefficient of 0.7, which is the closest. The results of the scenario execution are plotted using the dots. The six lines show the theoretical baselines with reaction times of 0, 0.7 and 2 seconds and friction coefficients of 0.7 and 0.9.

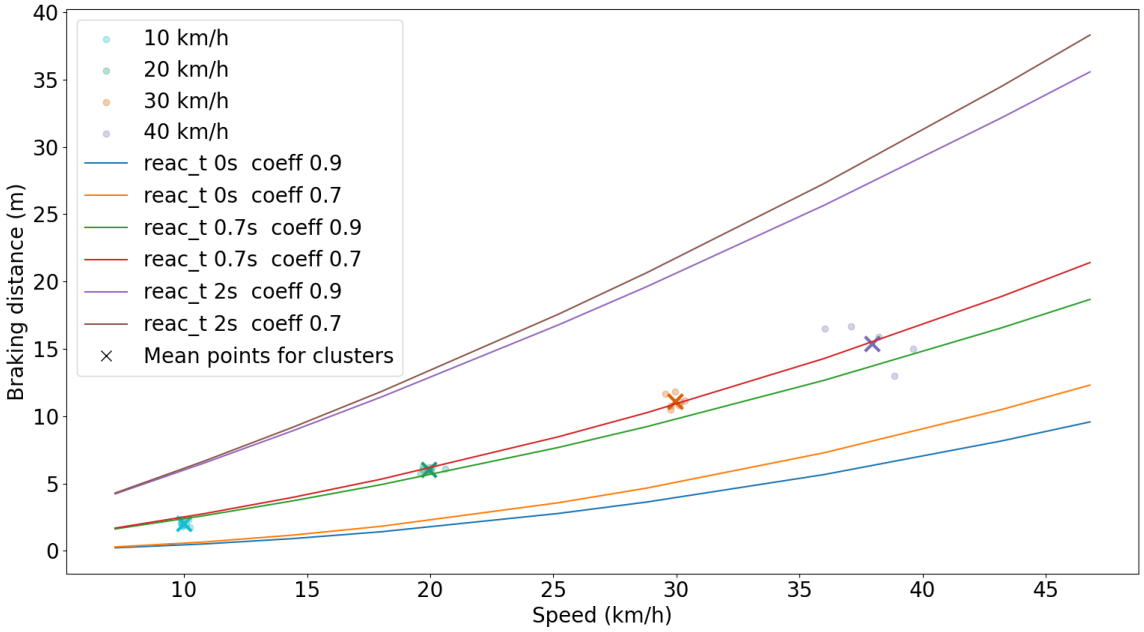

Figure 30. Experiment SimV2 results (dots) and theoretical reference modes TRM (lines).

To compare the simulation results to the reference modes more in detail, the deviation of braking distance d from a reference mode for each cluster of speeds is calculated. Similarly, as in Section 4.4.2, Equation 6 calculates the deviation of braking distance d. Table 19 shows the results of the calculation for SimV2 results, where the rows have the mean values of the cluster and the columns have a particular reference mode with its reaction time and friction coefficient.

TRM \ Mean point of a cluster (x, y coordinates)	**Reaction time (s)**	**0**		**0.7**		**2**	
	Friction coefficient	**0.7**	**0.9**	**0.7**	**0.9**	**0.7**	**0.9**
(10, 2.03)		-1.47	-1.59	0.48	0.35	4.09	3.96
(19.95, 6.05)		-3.81	-4.31	0.06	-0.43	7.27	6.77
(29.95, 11.11)		-6.07	-7.19	-0.25	-1.37	10.57	9.45
(37.95, 15.4)		-7.31	-9.1	0.07	-1.73	13.78	11.98

Table 19. Deviation of braking distance for a cluster from a reference mode for SimV2

Using Table 19, it can be observed that the results from simulation experiments can be compared the closest to the reference mode with a reaction time of 0.7 seconds and a friction coefficient of 0.7. The deviation from the reference mode is the smallest for all clusters.

4.4.4 SimV2 vs. ERF_MB

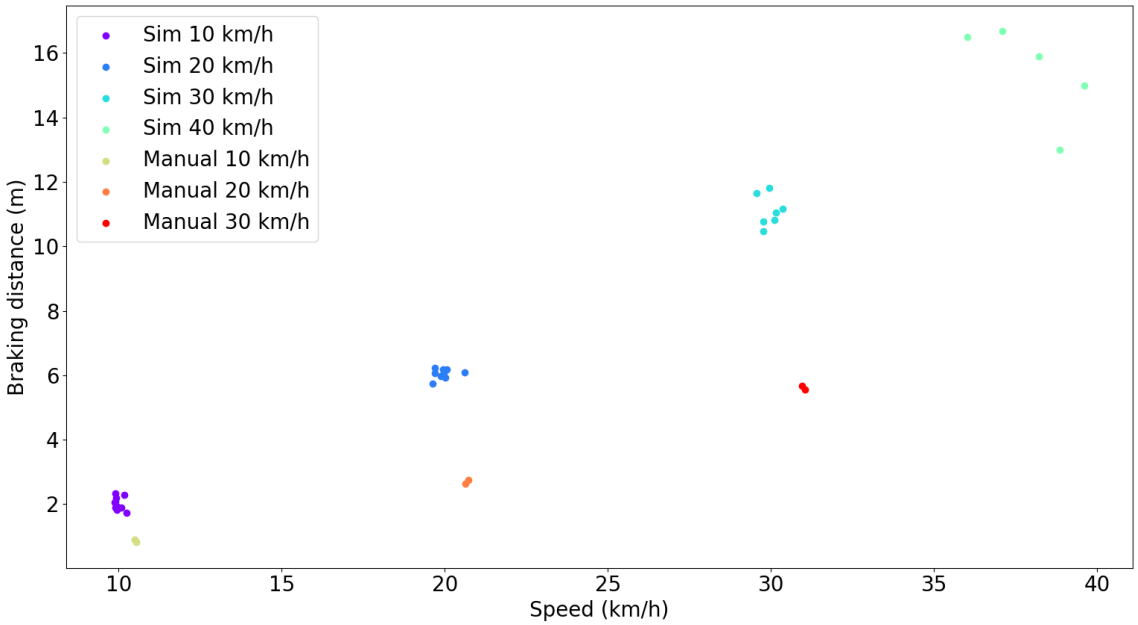

Figure 31. Experiment SimV2 results and empirical reference modes ERM_MB.

In Figure 31, results from experiment SimV2 are shown together with empirical reference mode ERF_MB. It can be observed that manual braking by a human driver in real life gives a lesser braking distance than ADS braking in a simulator.

The empirical reference mode ERF_MB was designed to validate the ADS. The ADS does not use the full braking capacity when we compare ERF_MB to the SimV2 results.

The empirical reference mode can be used to calculate the deviation of braking distance from a reference mode for each cluster of speeds can be calculated. The approach is described in Section 4.4.2. Figure 32 shows the fitted curve for ERF_MB data together with SimV2 results. It can be seen that the breaking distances in the simulation (SimV2) were consistently longer than what was measured during manual breaking (ERF_MB).

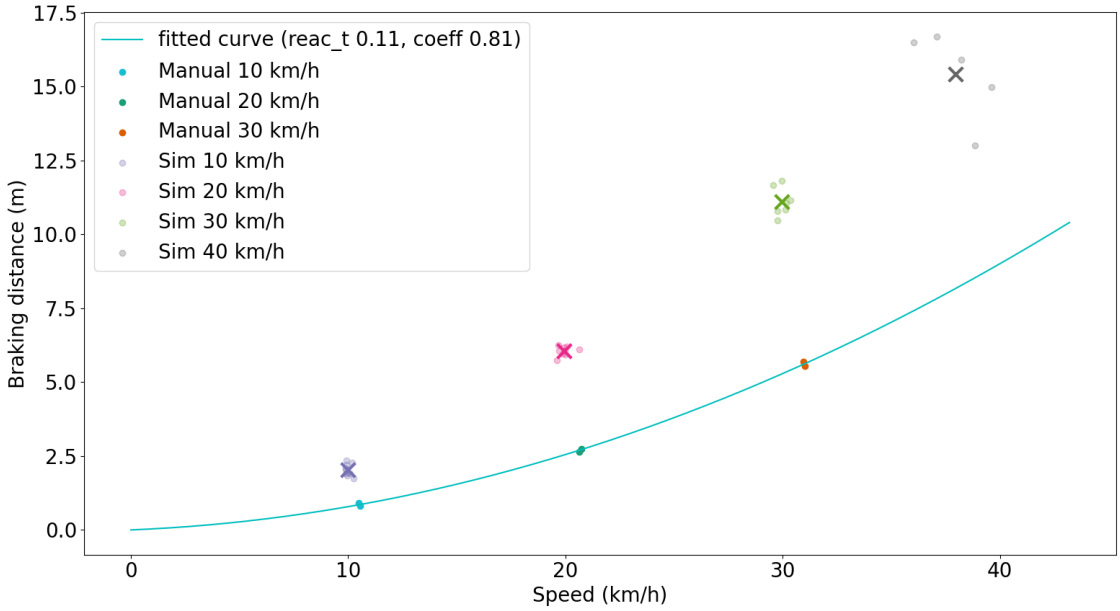

Figure 32. Fitted curve for ERF_MB data together with SimV2 data

4.4.5 SimV2 vs. ERF_VO

In Figure 33, results from the experiment SimV2 are shown together with empirical reference mode ERF_VO. It can be observed that the ADS braking distance, when a virtual object is placed in front of the vehicle in real life, is slightly shorter than the ADS braking in the simulator.

The empirical reference mode ERF_VO can also be used to test and compare the processing time of a specific component by removing it from the processing, in this case, the object detection and clustering of points using LiDAR.

The empirical reference mode can be used to calculate the deviation of braking distance from a reference mode for each cluster of speeds can be calculated. The approach is described in Section 4.4.2. Figure 34 shows the fitted curve for ERF_VO data together with SimV2 results. It can be seen that the breaking distances in the simulation (SimV2) were slightly longer than what was measured during automated braking with a virtual obstacle (ERF_VO), although it follows the curve.

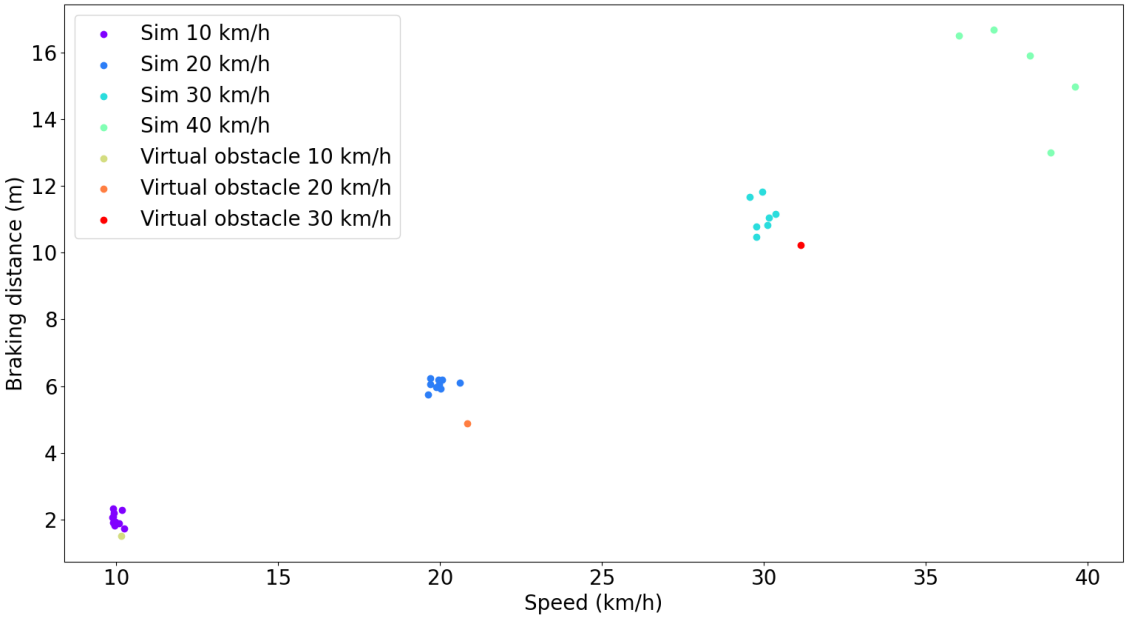

Figure 33. Experiment SimV2 results and empirical reference modes ERM_VO.

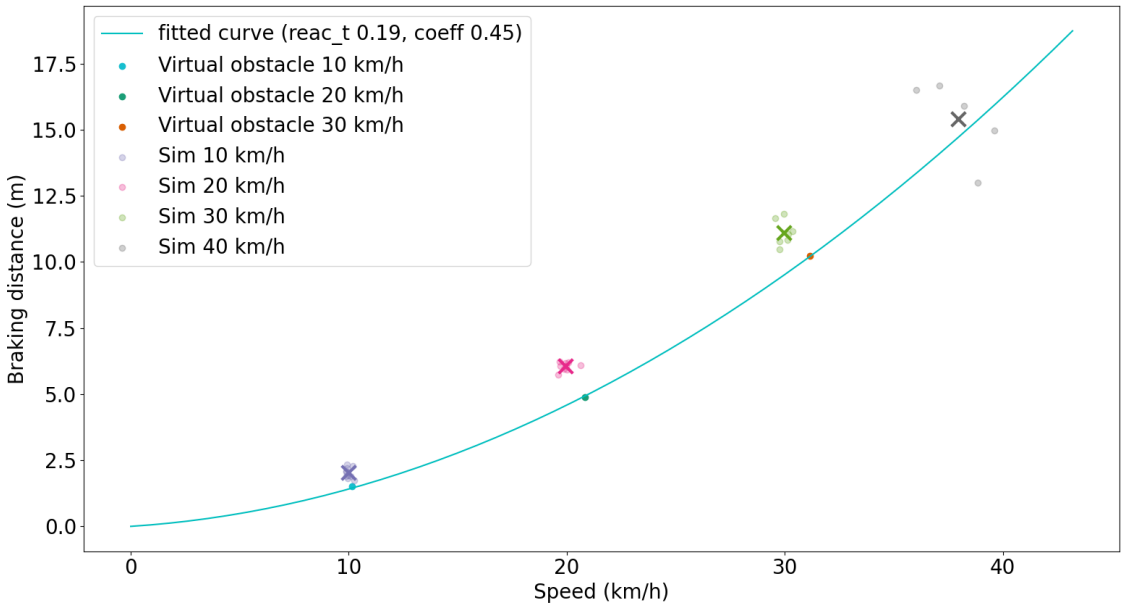

Figure 34. Fitted curve for ERF_VO data together with SimV2 data

4.4.6 SimV2 vs. ERF_TD

In Figure 35, results from experiment SimV2 are shown together with empirical reference mode ERF_TD. It can be observed that the ADS braking distance when a test dummy is pushed in front of the vehicle in real life is slightly shorter than the ADS braking in the simulator.

The empirical reference modes ERF_VO and ERF_TD were designed to validate the simulator. The ADS behaves slightly worse than in empirical reference modes ERF_VO and ERF_TD in real life. This could be explained by the differences in the environment or even the computer's capacity to simulate all sensors with an adequate frame rate.

When empirical reference mode ERF_VO is compared to empirical reference mode ERF_TD, they both perform similarly, although more data-gathering sessions have to be performed to have a conclusive result of how removing a component affected the processing time.

The empirical reference mode can be used to calculate the deviation of braking distance from a reference mode for each cluster of speeds can be calculated. The approach is described in Section 4.4.2. Figure 36 shows the fitted curve for ERF_TD data together with SimV2 results.

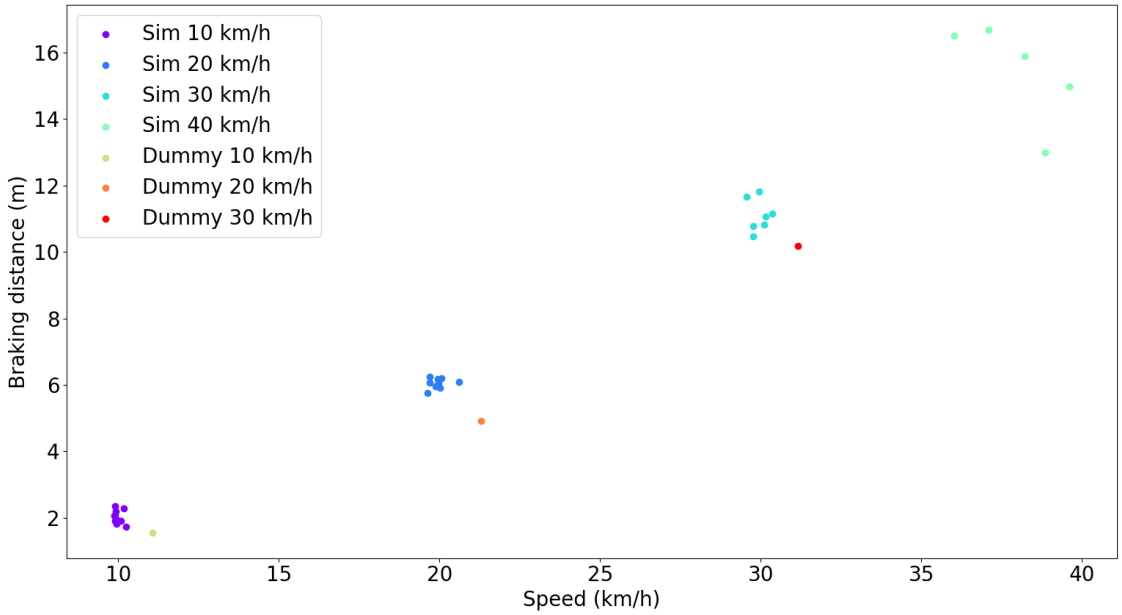

Figure 35. Experiment SimV2 results and empirical reference modes ERM_TD.

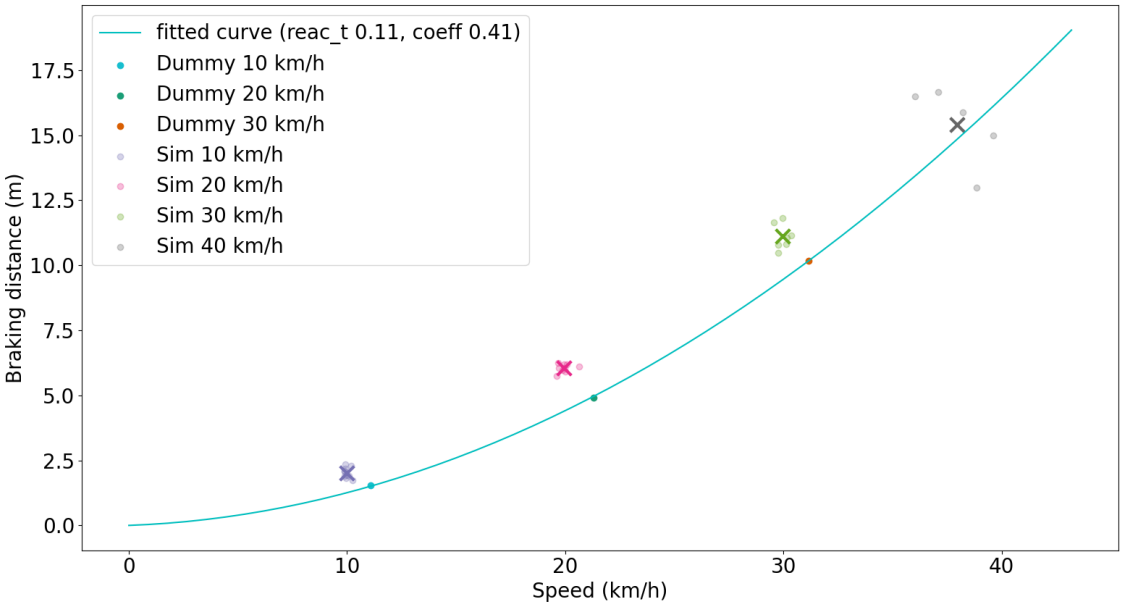

Figure 36. Fitted curve for ERF_TD data together with SimV2 data

Figure 37 shows how much the speed differs from the set target speed for the ERF_TD. The orange line shows the speed deviation for the SimV2 experiment when keeping the target speed of 30 km/h. When comparing Figure 37 to Figure 26, it can be observed that both differ, which shows that the controllers do not behave the same way.

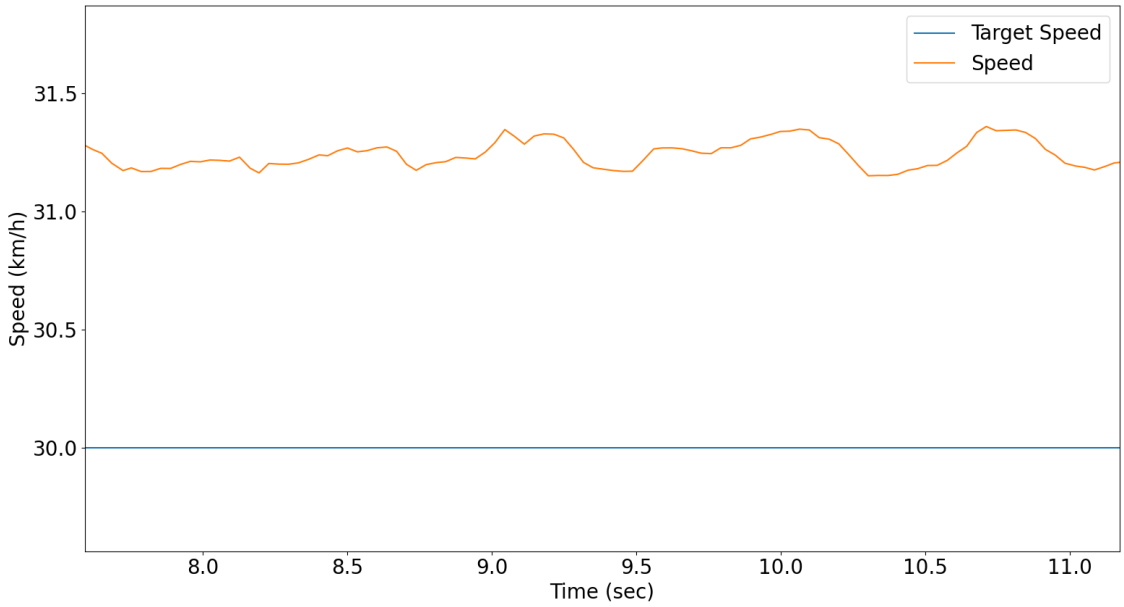

Figure 37. The deviation of speed for the ego vehicle in empirical reference mode ERF_TD while keeping a constant target speed of 30 km/h.

5 Discussion and Limitations

Based on the method, the experiments were designed and executed. The reference modes were designed based on the scenario selected, and data-gathering sessions were executed. The method was implemented successfully and was applicable.

Unfortunately, it was not successful to draw conclusions about safety due to unexpected issues and non-realistic behavior discovered related to one element in the simulation setup. The main issue stems from the controller in the simulator not being programmed the same as in real life. The simulator uses a PID controller, although the real-life vehicle uses a proprietary PACMod controller. It was seen that they do not behave the same where improvements to the PID controller in the simulator are required. Due to the simulator's non-realistic behavior, the simulation outcomes are not trustworthy. Even though the experiment results are inconclusive, once this issue is solved, the method can be repeated, and results can be drawn from there.

The comparison between the simulation results of SimV1 and SimV2 shows the improvement of the controller in keeping the target speed, although a proportional relationship can be seen with the braking distance, which increases in the results of SimV2. It can be improved by using a different controller, improving the current one, or even using a different simulator. From the results, if the simulator had behaved similarly as in the real world, with the experiments performed during the time frame of the thesis, it would have been possible to build some trust in the simulator and assert a judgment of the safety of the ADS.

When comparing the results from the empirical data-gathering sessions, it was seen that the ADS takes more time to brake than a human driver in manual braking reference mode. This is a decision of the developers to reduce the impact of phantom obstacles occurring during a drive. If a vehicle is behind and the ADS suddenly brakes, it becomes hazardous for everyone involved. This ADS is a research platform where unexpected situations may arise, and a safety driver is in the vehicle to take care of such situations.

The method presented in this thesis tests the full ADS with its sensors and autonomy software as it would be in real life. There is another way of testing separate components, such as the planner or perception modules. If a separate module needs to be tested, the method can and should be adjusted to fit the requirements.

Validation of the simulator is required with simple scenarios that are partly isolated and do not involve complicated actions. When the scenarios become more and more complicated, it gets harder to define reference modes for such situations. Before testing more complicated situations, the simulator's trustworthiness must be gained, as it is not easy to produce reference modes for complicated situations. At a later stage, scenarios based on the edge cases are developed without reference modes when the trustworthiness has been built.

A wide variety of scenarios can be implemented and tested using the proposed method. It is possible to test many smaller scenarios and combine them to test the combined functionality. Other factors, such as the weather, should be tested to ensure reliability and a wider ODD. The results of the experiments only describe one particular scenario. Emergency braking can be tested with different parameters, such as objects, vehicles, and weather conditions. To build trustworthiness, many more scenarios have to be tested similarly.

The method helps to analyze an ADS in-depth while looking at both aspects of the validity of the simulator and the ADS. This helps to understand the problems more in-depth, analyze them, and give valuable feedback to the developers of the autonomy software.

5.1 Limitations

A limitation of the approach is that it only works if all the elements are correctly implemented. This includes the simulation environment, the connection of the ADS autonomy software to the simulator, and the autonomy software itself. Even if implemented correctly, the simulator might not produce realistic results. One way of mitigating this issue is by using reference modes. However, a problem exists where the number of reference modes is limited. Even if the simulation is consistent with a reference mode, there might be scenarios where the behavior is inconsistent. This comes down to making the simulator trustworthy and proving that the simulator behaves the same as in the real world.

Another limitation comes from the difficulty of constructing reference modes. Once the scenarios become more and more complex, it is not easy to construct the reference mode, as many parameters come together at once. It takes a lot of effort and time to construct empirical reference modes. As the scenarios become more complex, it gets more challenging to construct the same scenario in real life. In addition, if the scenario tests critical safety systems, putting the safety driver and the real-life vehicle through such a situation would not be feasible. This is where artificially created scenarios in the simulation help build complex and safety-critical scenarios.

In this work, there are several threats to validity. Currently, the controller in the simulator and the real-world vehicle are different. This causes differences in the results. Additionally, the computer specifications can cause distinct differences in the results. Because the simulation has to simulate the sensors and the environment, it can impact how well the simulation is able to run, thus affecting the results. It can be solved by having a unified testing environment with high specifications that do not hinder the performance during testing.

Simulation of the real world is not perfect. The environment looks different; the sensor simulation can be different, and the hardware components, such as brakes, can be simulated differently. The brakes could be braking more. The data from scenarios from the smallest unit must be analyzed to be able to be compared. For this, real-world experiments are required to validate the simulator and its results.

Even if the simulation behaves well, is it possible to say it is trustworthy? It is possible to tell if there is a difference only if it behaves incompetently. In addition, the scenarios defined could be incorrect. This means that modules are being incorrectly tested or not tested at all.

In addition, the engineering metrics directly impact the results of the thesis. Depending on the development of the autonomy software, testing of the full ADS system can be affected. For example, it depends on how the perception module is trained and how well the model is able to detect objects and track them. It also depends on the data the model was trained and how much the data from the simulator differs from the data from the real-life sensors.

Due to the time frame of the thesis and the unpredictable weather conditions in Tartu up until the end of April 2024, it was challenging to perform the empirical data-gathering sessions more than once. Due to this, the braking performance might not be as precise as the braking distance was not averaged using several runs. Several data-gathering sessions have to be conducted for more precise results, considering the same conditions.

5.2 Future Work

After discussions with the ADL developer team, it was seen that the node controlling the ADS was not behaving as expected and tuning the controller parameters proved non-trivial. It is planned to improve it by tuning the parameters more or rewriting the node themselves.

In the future, the method proposed can be used as a base for progressing on scenario-based safety testing. Even if the technology stack differs, the method can be applied by replacing some building blocks in the simulation setup. The setup can be used to perform regression testing for different versions of the autonomy software.

Based on the results of this thesis, it is possible to repair the issues discovered and repeat the same work to provide conclusive statements for the safety of the ADS.

6 Conclusion

The contribution of the thesis is two-fold. Firstly, a method for simulation-based safety testing of an ADS has been developed. The method describes the steps for setting up the simulation-based safety testing environment, the experiments' design, and the analysis of the data gathered. It gives the possibility of quantifying the behavior of the ADS and how it is possible to draw conclusions to provide transparency for safety.

Secondly, the execution of experimental simulation runs, the definition of reference modes, and the analysis of the results were performed. This part of the thesis serves as a proof-of-concept for the method's applicability and provides examples of how to use the method for future users. In addition, a conclusion on how the simulation compares to the real-world behavior of the ADS is made.

The method directed the design and execution of the experiments. The reference modes and data collection sessions were developed in accordance with the experimental scenario. The method was applicable and successfully put into practice.

Unfortunately, the unexpected and non-realistic behavior linked to the controller implemented in the simulator prevented any conclusions about safety from being made. The simulation's results are unreliable because of the controller's unrealistic behavior. Experiments yield inconclusive findings, but once this problem is resolved, the process can be repeated, and conclusions can be made.

The results from SimV1 and SimV2 experiments show the inconsistent behavior of the controller where the PID controller is not able to keep the speed constant to the target speed set for the autonomy software. The comparison between the simulation results of SimV1 and SimV2 shows the improvement of the controller in keeping the target speed, although a proportional relationship can be seen with the braking distance, which increases in the results of SimV2.

In the future, the method can be used as a starting point for progressing on scenario-based safety testing. Even if the technology stack differs, the method can be applied by replacing some building blocks in the simulation setup. The setup can be used to perform regression testing for different versions of the autonomy software.

Based on the results of this thesis, it is possible to repair the issues discovered and repeat the same work to provide conclusive statements for the safety of the ADS.

In summary, the goal of defining the method for simulation-based safety testing was achieved. A set of experiments with reference modes was defined and performed to validate the method. The conclusion of the findings was mentioned, and the results and limitations were discussed.

Acknowledgment

I would like to thank my supervisor, Dr. Dietmar Pfahl, for his support throughout the thesis. The discussions, guidance, and feedback have helped me immensely. I would also like to thank the University of Tartu for the supportive study environment provided, i.e., the Autonomous Driving Lab, and Tambet Matiisen for his advice and support, Alan Mitt for introduction to the digital twin of Tartu city and the simulated vehicle, Mahir Gulzar for guidance on the Autoware Mini setup in CARLA, as well as Edgar Sepp and Kertu Toompea for help in designing and executing the empirical reference modes.

Special gratitude goes to my partner, Savvy, and my family for their support and advice. For always encouraging me to go forward to achieve my goals and always being there for me.

I would like to thank the University of Tartu and the IT Academy for supporting me financially throughout my studies.

This thesis was partly funded by the Autonomous Driving Lab, a collaboration project between the University of Tartu and Bolt.

References

[1] "Self-Driving Car Technology for a Reliable Ride - Waymo Driver," Waymo. Accessed: Oct. 11, 2023. [Online]. Available: https://waymo.com/waymo-driver/

[2] "Autopilot." Accessed: Oct. 07, 2023. [Online]. Available: https://www.tesla.com/autopilot

[3] S. D. Pendleton *et al.*, "Perception, Planning, Control, and Coordination for Autonomous Vehicles," *Machines*, vol. 5, no. 1, Art. no. 1, Mar. 2017, doi: 10.3390/machines5010006.

[4] A. S. Mueller, J. B. Cicchino, and D. S. Zuby, "What humanlike errors do autonomous vehicles need to avoid to maximize safety?," *Journal of Safety Research*, vol. 75, pp. 310–318, Dec. 2020, doi: 10.1016/j.jsr.2020.10.005.

[5] P. Koopman, *How Safe Is Safe Enough?: Measuring and Predicting Autonomous Vehicle Safety*. Pittsburgh, Pennsylvania: Independently published, 2022.

[6] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, Dec. 2016, doi: 10.1016/j.tra.2016.09.010.

[7] "J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International." Accessed: Nov. 01, 2023. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/

[8] "SAE Levels of Driving Automation™ Refined for Clarity and International Audience." Accessed: Nov. 01, 2023. [Online]. Available: https://www.sae.org/site/blog/sae-j3016-update

[9] S. Tang *et al.*, "A Survey on Automated Driving System Testing: Landscapes and Trends," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, p. 124:1-124:62, Jul. 2023, doi: 10.1145/3579642.

[10] Publication Office of the European Union, "Ethics of Connected and Automated Vehicles: recommendations on road safety, privacy, fairness, explainability and responsibility," 2020. [Online]. Available: https://doi.org/10.2777/035239

[11] N. Kauffmann, F. Fahrenkrog, L. Drees, and F. Raisch, "Positive risk balance: a comprehensive framework to ensure vehicle safety," *Ethics Inf Technol*, vol. 24, no. 1, p. 15, Feb. 2022, doi: 10.1007/s10676-022-09625-2.

[12] AAA Foundation for Traffic Safety, "Rates of Motor Vehicle Crashes, Injuries, and Deaths in Relation to Driver Age, United States, 2014 – 2015," 2017. [Online]. Available: https://aaafoundation.org/wp-content/uploads/2017/11/CrashesInjuriesDeathsInRelationToAge2014-2015Brief.pdf

[13] N. Kalra and D. G. Groves, "The Enemy of Good: Estimating the Cost of Waiting for Nearly Perfect Automated Vehicles," RAND Corporation, Nov. 2017. Accessed: Apr. 01, 2024. [Online]. Available: https://www.rand.org/pubs/research_reports/RR2150.html

[14] I. Hasuo, "Responsibility-Sensitive Safety: an Introduction with an Eye to Logical Foundations and Formalization." arXiv, Jun. 07, 2022. Accessed: Mar. 18, 2024. [Online]. Available: http://arxiv.org/abs/2206.03418

[15] A. K. Yadav and N. R. Velaga, "Modelling the relationship between different Blood Alcohol Concentrations and reaction time of young and mature drivers," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 64, pp. 227–245, Jul. 2019, doi: 10.1016/j.trf.2019.05.011.

[16] A. Pouliou, F. Kehagia, G. Poulios, M. Pitsiava-Latinopoulou, and E. Bekiaris, “Drivers’ Reaction Time and Mental Workload: A Driving Simulation Study,” *Transport and Telecommunication Journal*, vol. 24, pp. 397–408, Nov. 2023, doi: 10.2478/ttj-2023-0031.
[17] J. J. Rolison and S. Moutari, “Combinations of factors contribute to young driver crashes,” *Journal of Safety Research*, vol. 73, pp. 171–177, Jun. 2020, doi: 10.1016/j.jsr.2020.02.017.
[18] K. Čulík, A. Kalašová, and V. Štefancová, “Evaluation of Driver’s Reaction Time Measured in Driving Simulator,” *Sensors (Basel)*, vol. 22, no. 9, p. 3542, May 2022, doi: 10.3390/s22093542.
[19] Z. Zhang, Y. Asakawa, T. Imamura, and T. Miyake, “Experiment Design for Measuring Driver Reaction Time in Driving Situation,” in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2013, pp. 3699–3703. doi: 10.1109/SMC.2013.630.
[20] P. Droździel, S. Tarkowski, I. Rybicka, and R. Wrona, “Drivers ’reaction time research in the conditions in the real traffic,” *Open Engineering*, vol. 10, pp. 35–47, Jan. 2020, doi: 10.1515/eng-2020-0004.
[21] “Road vehicles – Functional safety (ISO 26262-1:2018).” [Online]. Available: https://www.iso.org/standard/68383.html
[22] “Road vehicles — Safety of the intended functionality (ISO 21448:2022).” Accessed: Mar. 04, 2024. [Online]. Available: https://www.iso.org/standard/77490.html
[23] S. Liu, “Council Post: Safety Of The Intended Functionality (SOTIF) For Autonomous Driving,” Forbes. Accessed: Mar. 05, 2024. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2022/09/23/safety-of-the-intended-functionality-sotif-for-autonomous-driving/
[24] “Presenting the Standard for Safety for the Evaluation of Autonomous Vehicles and Other Products | UL Standards & Engagement.” Accessed: Dec. 30, 2023. [Online]. Available: https://ulse.org/ul-standards-engagement/presenting-standard-safety-evaluation-autonomous-vehicles-and-other-1
[25] “Road vehicles — Test scenarios for automated driving systems — Scenario based safety evaluation framework (ISO 34502:2022).” [Online]. Available: https://www.iso.org/standard/78951.html
[26] F. Khan, M. Falco, H. Anwar, and D. Pfahl, “Safety Testing of Automated Driving Systems: A Literature Review,” *IEEE Access*, pp. 1–1, 2023, doi: 10.1109/ACCESS.2023.3327918.
[27] Z. Zhong, Y. Tang, Y. Zhou, V. de O. Neves, Y. Liu, and B. Ray, “A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation.” arXiv, Dec. 01, 2021. Accessed: Nov. 01, 2023. [Online]. Available: http://arxiv.org/abs/2112.00964
[28] K. Go and J. M. Carroll, “The blind men and the elephant: views of scenario-based system design,” *interactions*, vol. 11, no. 6, pp. 44–53, Nov. 2004, doi: 10.1145/1029036.1029037.
[29] T. Menzel, G. Bagschik, and M. Maurer, “Scenarios for Development, Test and Validation of Automated Vehicles.” arXiv, Apr. 27, 2018. doi: 10.48550/arXiv.1801.08598.
[30] Bureau of Transportation Statistics, “Pre-Crash Scenario Typology for Crash Avoidance Research.” 2007. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/6281/dot_6281_DS1.pdf

[31] Gelder et al., “Scenario Categories for the Assessment of Automated Vehicles.” 2020. [Online]. Available: https://wiki.unece.org/download/attachments/117509173/VMAD-SG1-11-03%20%28NL%29%20Scenario%20Categories_v1.7.pdf?api=v2
[32] “IEEE Standard for Assumptions in Safety-Related Models for Automated Driving Systems,” *IEEE Std 2846-2022*, pp. 1–59, Apr. 2022, doi: 10.1109/IEEESTD.2022.9761121.
[33] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, “A Survey on Simulators for Testing Self-Driving Cars,” in *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*, Apr. 2021, pp. 62–70. doi: 10.1109/MetroCAD51599.2021.00018.
[34] C. Team, “CARLA,” CARLA Simulator. Accessed: Oct. 12, 2023. [Online]. Available: http://carla.org//
[35] “CARLA ScenarioRunner.” Accessed: Nov. 03, 2023. [Online]. Available: https://carla-scenariorunner.readthedocs.io/en/latest/
[36] “ASAM OpenSCENARIO.” Accessed: Oct. 14, 2023. [Online]. Available: https://www.asam.net/standards/detail/openscenario/
[37] “Autoware Mini.” University of Tartu Autonomous Driving Lab, Oct. 07, 2023. Accessed: Oct. 14, 2023. [Online]. Available: https://github.com/UT-ADL/autoware_mini
[38] Toyota Motor Europe, “RX L, Owner’s Manual.” [Online]. Available: https://www.lexus.co.uk/owners/about-my-lexus/manuals
[39] Y. Li *et al.*, “Choose Your Simulator Wisely: A Review on Open-source Simulators for Autonomous Driving.” arXiv, Dec. 26, 2023. Accessed: Mar. 04, 2024. [Online]. Available: http://arxiv.org/abs/2311.11056
[40] “OpenSCENARIO support - CARLA ScenarioRunner.” Accessed: Nov. 03, 2023. [Online]. Available: https://carla-scenariorunner.readthedocs.io/en/latest/openscenario_support/
[41] “ASAM OpenSCENARIO: User Guide.” Accessed: Nov. 03, 2023. [Online]. Available: https://releases.asam.net/OpenSCENARIO/1.0.0/ASAM_OpenSCENARIO_BS-1-2_User-Guide_V1-0-0.html
[42] F. Khan, H. Anwar, and D. Pfahl, “A Process for Scenario Prioritization and Selection in Simulation-Based Safety Testing of Automated Driving Systems,” in *Product-Focused Software Process Improvement*, R. Kadgien, A. Jedlitschka, A. Janes, V. Lenarduzzi, and X. Li, Eds., in Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2024, pp. 89–99. doi: 10.1007/978-3-031-49266-2_6.
[43] Virginia Transportation Research Council, “An investigation of the utility and accuracy of the table of speed and stopping distances.” [Online]. Available: https://vtrc.virginia.gov/media/vtrc/vtrc-pdf/vtrc-pdf/01-r13.pdf
[44] “Euclidean distance | Definition, Formula, & Facts | Britannica.” Accessed: May 14, 2024. [Online]. Available: https://www.britannica.com/science/Euclidean-distance
[45] “Least Squares Method: What It Means, How to Use It, With Examples,” Investopedia. Accessed: May 12, 2024. [Online]. Available: https://www.investopedia.com/terms/l/least-squares-method.asp
[46] “Numeracy, Maths and Statistics - Academic Skills Kit.” Accessed: May 13, 2024. [Online]. Available: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html

License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, **Laima Anna Dalbina**,

(*author's name*)

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

Simulation-based Safety Testing of an Automated Driving System (ADS),

(*title of thesis*)

supervised by Dietmar Alfred Paul Kurt Pfahl.

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Laima Anna Dalbina

14/05/2024