TU Braunschweig, Carl Friedrich Gauß Faculty

Master's Thesis

Scenario Generation for Testing of Automated Driving Functions based on Real Data

Author:

Fin Malte Heuer

18. April 2022

Advisors:

Prof. Dr-Ing. Ina Schaefer

TU Braunschweig - Department of Software Engineering and Automotive Informatics

Prof. Dr. Frank Köster

Carl von Ossietzky Universität Oldenburg - Department of Informatics - Intelligent Transport Systems

Heuer, Fin Malte:
Scenario Generation for Testing of Automated Driving Functions based on Real Data
Master's Thesis, TU Braunschweig, 2022.

Abstract

Scenario-based testing is state-of-the-art for testing **A**dvanced **D**riving **A**ssistance **S**ystem / **A**utonomous **D**riving (ADAS/AD). The challenge in scenario-based testing is the generation and selection of the scenarios. To generate reproducible scenarios and to efficiently perform tests of ADAS/AD, simulation environments are used because the environment is under control. However, an open research question on this topic is the realism of the emerging scenarios within the simulation. Realism is a challenge because the ADAS/AD must eventually function in the real world.

To solve this challenge, we contribute a concept (1) to use a simulation environment to generate realistic synthetic scenarios and (2) to evaluate their realism. We focus our research on dynamic objects within the scenarios. We parameterize the microscopic traffic simulation environment SUMO and generate synthetic scenarios by simulation. We base the evaluation of realism on real scenarios observed by the testbed Lower Saxony. To measure realism, we define ten different characteristics in different aspects. With these characteristics, we measure realism by comparing the characteristics against the real data. As a prototype, we implement this concept and compare three different methods of parameterization concerning their realism: (a) expert-based, (b) optimization-based, and (c) clustering-based.

Based on our evaluation, we find that parameterization has a strong influence on the realism of criticality metrics such as the **T**ime **T**o **C**ollision (TTC). In contrast, we find that the influence of parameterization on other aspects is comparatively low. We observe that realism depends on the parameterization and the capabilities of the simulation model. We discover that expert-based parameterization generates the most realistic scenes compared to the other methods and about 2.5 times as many realistic scenes during the same period as without parameterization. Each parameterization has its own strengths concerning different aspects of realism. We conclude that SUMO generates realistic dynamic objects in scenarios in many aspects.

Contents

List of Figures

List of Tables

List of Code Listings

List of Acronyms

AD	**A**utonomous **D**riving
ADAS	**A**dvanced **D**riving **A**ssistance **S**ystem
ADAS/AD	**A**dvanced **D**riving **A**ssistance **S**ystem / **A**utonomous **D**riving
API	**A**pplication **P**rogramming **I**nterface
ASAM	**A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems
CCLCI	**C**ross-**C**orrelation **L**ane **C**hange **I**ndicator
CFM	**C**ar of **F**ollowing **M**odel
CLI	**C**ommand **L**ine **I**nterface
DLR	German Aerospace Center (**D**eutsches Zentrum für **L**uft- und **R**aumfahrt)
DRAC	**D**eceleration **R**ate to **A**void a **C**rash
EIDM	**E**xtended **I**ntelligent **D**river **M**odel
GA	**G**enetic **A**lgorithm
GUI	**G**raphical **U**ser **I**nterface
IDM	**I**ntelligent **D**river **M**odel
MTS	**M**icroscopic **T**raffic **S**imulation
PCA	**P**rincipal **C**omponent **A**nalysis
SPSA	**S**imultaneous **P**erturbation **S**tochastic **A**pproximation
SUMO	**S**imulation of **U**rban **Mo**bility
SVD	**S**ingular **V**alue **D**ecomposition
TH	**T**ime **H**eadway
TraCI	**Tra**ffic **C**ontrol **I**nterface
TTC	**T**ime **T**o **C**ollision
UTC	**C**oordinated **U**niversal **T**ime

UTM **U**niversal **T**ransverse **M**ercator

XML e**X**tensible **M**arkup **L**anguage

1. Introduction

Modern road vehicles are equipped with increasingly comprehensive and powerful **A**dvanced **D**riving **A**ssistance **S**ystems (ADASs) for the automation of road traffic [FKP+20]. They are designed to improve safety and comfort on the road [Tat15]. By delegating further driving functions from the driver to ADASs, the boundary to **A**utonomous **D**riving (AD) vanishes. An essential aspect in the development of ADAS/AD is the safeguarding of the driving function [HWLZ16]. In the course of increasingly complex interrelationships and larger systems, the testing of ADAS/AD is a challenge, as the requirements of the system cannot be fully formulated. The application of real-world tests is becoming increasingly difficult [WW15, Sch17] and is not suitable as the only test for release [WW16]. Therefore, we need methods for verification and validation that limit the number of real test kilometers driven.

The state-of-the-art is scenario-based testing, where real-world traffic is abstracted by scenarios [NWH+20]. The core idea is to use representative scenarios to test the ADAS in the real world, or the simulation [Sch17]. As a result, insignificant sections without action or event should be removed from the validation procedure [Tat15]. The selection of scenarios determines the tested behavior and the likelihood of uncovering defects. Therefore, an essential component in this testing process is generating or collecting the scenarios. Open questions on this topic are: Which scenarios are needed, and at what point can the driving function is classified as safe? To determine which scenarios to test, two approaches to scenario selection have evolved: data-based and knowledge-based [Tat15]. The knowledge-based approach uses knowledge about how roads are built to derive scenarios [BMKM18]. In contrast, data-driven approaches use observations to collect or derive scenarios.

Pütz et al. [PZBE17] distinguish between real-world data and traffic simulation data as a source for scenarios in the data-driven approaches. Real-world data is used to derive scenarios directly, e.g., collect takeovers. According to Pütz et al. [PZBE17], simulation environments can be efficiently used to identify critical scenarios, which are highly relevant for the verification and validation of ADAS/AD. This approach is cheaper compared to real test kilometers, easier to implement, and allows to explicitly enforce critical scenarios, which is dangerous in the real world. Yue et al.

[YSWL20] present a method to generate scenarios using a simulation environment. Within the development of ADAS/AD simulators are already an established tool [AWS14] therefore, scenario generation from simulators can be easily performed. However, it is not clear whether the simulation behavior reflects reality and the derived scenarios can be used to test real ADAS/AD.

The German Aerospace Center (**D**eutsches Zentrum für **L**uft- und **R**aumfahrt) (DLR) built the testbed Lower Saxony in Germany in 2020 [KMKL18]. One of the key components is the detection system on highway A39 between Brunswick and Wolfsburg. The detection system is based on stereo-video sensors and detects objects on a length of approximately 7.45 km around the clock. All objects driving on the highway are digitally recognized and stored in the form of trajectories. This infrastructure provides a comprehensive insight into the behavior of road users and the events happening on the section of the road. The data provided by the testbed are very well suited for data-driven approaches as they represent a ground truth for reality.

Goal of this Thesis

The detailed trajectory data provided by the testbed Lower Saxony describes the reality of the traffic on a highway. This data contains realistic scenarios since they are observed within the real world. We compare these real-world scenarios against scenarios generated by a simulation environment. By this comparison, we judge which aspects of the simulation-based scenarios are realistic. Thus, we investigate whether the simulation-based scenarios reflect reality and contribute to solving the problem if simulation environments generate realistic scenarios. Our key research question is: How can a simulation environment be parameterized to create realistic scenarios?

We contribute a concept that consists of two parts: (1) the generation of realistic scenarios and (2) the evaluation of their realism. First, we parameterize a simulation environment using three different parameterization methods. With these parameterizations, we generate synthetic scenarios with the simulation environment. Within a set of ten characteristics, we evaluate the realism of the scenarios and investigate the ability of these characteristics to judge realism.

Structure of the Thesis

First, we introduce background information in Chapter 2. In Chapter 3 we present a process chain for generating synthetic trajectories and evaluate their realism. This chapter introduces different methods for parameterizing the simulation environment and aspects for measuring realism. We present the implementation of a prototype of this process chain in Chapter 4. Afterward, we describe the experiments and discuss the results in Chapter 5. Chapter 6 provides an overview of related work and how this work differs from theirs. Finally, we summarize our findings in the Chapter 7 and identify potential areas for further research in Chapter 8.

2. Background

In this chapter, we discuss the fundamentals of this thesis. First, a short introduction to the current concepts for testing in the automotive context is given in Section 2.1. Within this section, the topic of scenario-based testing is explained. Since this thesis focus on the dynamic objects within scenarios, Section 2.2 gives a short introduction about trajectories, the datasets, and their characteristics. We use traffic simulation to generate synthetic trajectory data. Section 2.3 will explain the fundamentals of traffic simulation and the configuration.

2.1 Testing in Automotive Context

In this section, we will explain the basics of testing in the context of the automotive industry. Fist current concepts for testing and their limitations within the context of AD are shown. This will lead to an introduction of scenario-based testing and its characteristics.

Current test approach in automotive context

One of the main aspects of developing a new ADAS is the safeguarding of the driving function. Within the automotive context, different concepts have been established. Wachenfeld and Winner [WW16] present concepts for driverless, assistive, and semi-automated systems. The concepts and their targeted automation levels have in common that they access the driver as a backup level [WW16]. These concepts focus on ensuring controllability by the driver using a distance-based testing approach. Wachenfeld and Winner [WW16] argue that (within these levels of automation) it is sufficient to ensure controllability by the driver in the verification and validation process since the driver controls the behavior in case the system malfunctions. One major problem of the distance-based approach is the required traveled distance to ensure the safety of the ADAS. Wachenfeld and Winner [WW16] calculate that it is necessary to drive at least 2.1 bn test kilometers in order to reliably demonstrate that a ADAS reduces the number of fatal accidents by half. Since this enormous number of test kilometers is not realistic to achieve in reality, Schuldt [Sch17] proposes scenario-based testing as a solution.

Scenario-based testing

Scenario-based testing reduces the necessarily driven test kilometers into a relevant subset of scenarios. In order to test an ADAS/AD, a set of scenarios is required, and the ADAS/AD has to be tested within these scenarios. The tests are mainly performed in virtual environments [Sch17]. Tatar [Tat15] distinguishes the scenario generation into data-based and knowledge-based approaches. Knowledge-based approaches use knowledge about how roads are built to derive scenarios [BMKM18]. Data-driven approaches use observations to collect or derive scenarios [dGP17, YLW+14, ZdR17]. Ulbrich et al. [UMR+15] define the terms scenario, scene, and situation as follows:

- Scene: "A scene describes a snapshot of the environment including the scenery and dynamic elements, as well as all actors' and observers' self-representations, and the relationships among those entities. Only a scene representation in a simulated world can be all-encompassing (objective scene, ground truth). In the real world it is incomplete, incorrect, uncertain, and from one or several observers' points of view (subjective scene)." [UMR+15]

- Situation: "A situation is the entirety of circumstances, which are to be considered for the selection of an appropriate behavior pattern at a particular point of time. It entails all relevant conditions, options and determinants for behavior. A situation is derived from the scene by an information selection and augmentation process based on transient (e.g. mission-specific) as well as permanent goals and values. Hence, a situation is always subjective by representing an element's point of view" [UMR+15]

- Scenario: "A scenario describes the temporal development between several scenes in a sequence of scenes. Every scenario starts with an initial scene. Actions & events as well as goals & values may be specified to characterize this temporal development in a scenario. Other than a scene, a scenario spans a certain amount of time." [UMR+15]

Schuldt [SSL+13] introduces a model to describe the structure of scenarios in four different levels, which is adapted by Bagschik et al. [BMKM18] to a 5-level model. These levels are shown in Figure 2.1. Level 1 (L1) describes the static road model, including the geometry and the surface properties. Level 2 (L2) describes the structural boundaries and traffic signs. Level 3 (L3) introduces temporal manipulation of L1 and L2, for example, road works. The fourth level (L4) describes objects that behave dynamically or statically and their interactions. The fifth level (L5) describes environmental factors, for example, weather conditions.

2.2 Trajectories

In this section, we first present the available real-world reference datasets. Afterward, we explain the term trajectory.

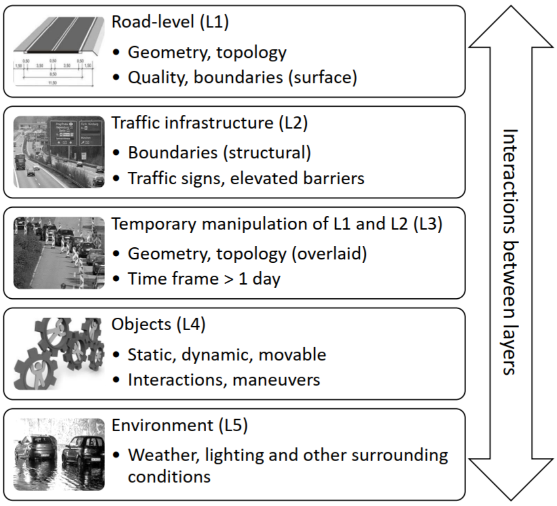
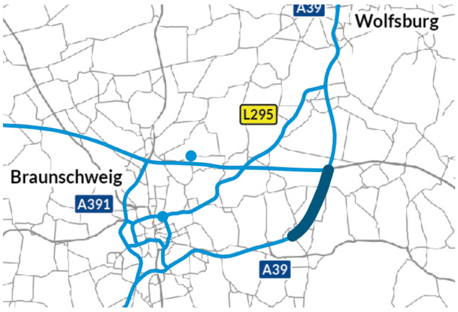


Figure 2.1: 5-level model scenario model of Bagschik et al. Source: [BMM18]



(a) Location of the testbed Lower Saxony. The section highlighted in blue marks the area of the detection system on the highway. Source: [DLR22]

(b) A mast with two stereo-camera sensors from the testbed Lower Saxony. Source: [DLR22]

Figure 2.2: Testbed Lower Saxony

Real data

Different trajectory datasets are publicly available. The most popular datasets are from the project NGSIM [FHW07] or the HighD [KBKE18] dataset. In this thesis, we use highway trajectory data from the testbed Lower Saxony [KMKL18]. The testbed was build in 2020 by the DLR. The data is captured by a static infrastructure comparable to the NGSIM infrastructure [FHW07]. It covers two lanes and the emergency lane in each direction on a length of 7.45 km and is captured by 142 stereo-camera systems. The testbed is located in Germany - Lower Saxony between Brunswick and Wolfsburg. The detection location is shown in Figure 2.2(a). The blue highlighted part marks the detection area on the A39 highway. Figure 2.2(b) shows one mast from 71 equipped with 2 stereo-camera sensors. The two boxes at the top of this figure contain two cameras combined into a stereo-camera system. The hardware and antennas in the lower part of the figure are V2X communication devices, which are not of further interest in this work. The detection system recognizes every object within the detection range and stores it digitally as trajectories. The trajectories are represented in a global coordinate system (Universal Transverse

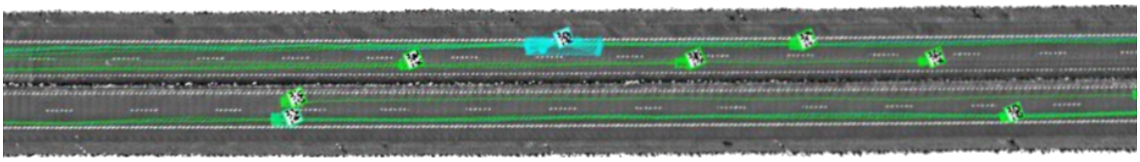


Figure 2.3: Trajectory data visualized from birds eye perspective. The boxes mark the detected object, the lines the followed trajectory. Above the objects, a small pixel map is a visualization of the objects' identifier.

Mercator (UTM)) and absolute time (Coordinated Universal Time (UTC)) with a sampling frequency of 20 Hz. Figure 2.3 shows an excerpt from the detected data visualized from a birds' eye view. The detected objects are visualized by boxes. The different types of vehicles are identified by their color. The distance traveled is represented by a line. While the data semantically represents a situation shown in Figure 2.3, the underlying dataset is presented in a database.

Trajectories

A trajectory in traffic describes the path that an object takes. A distinction is made between discrete-time and continuous-time trajectories. While functions represent continuous-time trajectories, discrete-time trajectories are described by sample points at specific time steps. In the context of this thesis, only discrete-time trajectories are of interest. Wagner et al. [WMR+13] define a trajectory T as an ordered list of spatiotemporal measurements $p_1, p_2, ...p_n$. Each point p_i consists of the spatial coordinates x_i, y_i and a timestep t_i: $p_i = (x_i, y_i, t_i)$. The trajectory $T = (p_1, p_2, ..., p_n)$ is ordered by time $t_1 < t_2 < ... < t_n$. In the real world, spatial coordinates often describe global coordinates, while relative coordinates are often used in the simulation context. The same applies to the temporal dimension. The individual points p_i may contain further information derived from their temporal progression, for example, velocity, acceleration, or heading. A trajectory may also contain general information about its classification or object dimensions.

2.3 Traffic Simulation

In this section, we first describe the fundamentals of traffic simulation. Afterward, we describe the traffic simulator SUMO in more detail. Since SUMO is a microscopic traffic simulator, we briefly describe how the microscopic traffic simulation is performed. In order to simulate traffic, the simulation parameters have to be determined. This process is called calibration. Finally, we give a short introduction to traffic simulation calibration.

Fundamentals of Traffic Simulation

Simulations allow modeling reality and provide an abstraction used to study the modeled characteristics. Traffic simulations model the traffic on different levels of abstraction. Within the context of scenario-based testing, traffic simulators are used to apply the testing [Sch17] and also to derive the scenarios from it [SBW+16]. Krauss [Kra98] distinguishes simulation environments into three different classes of abstraction:

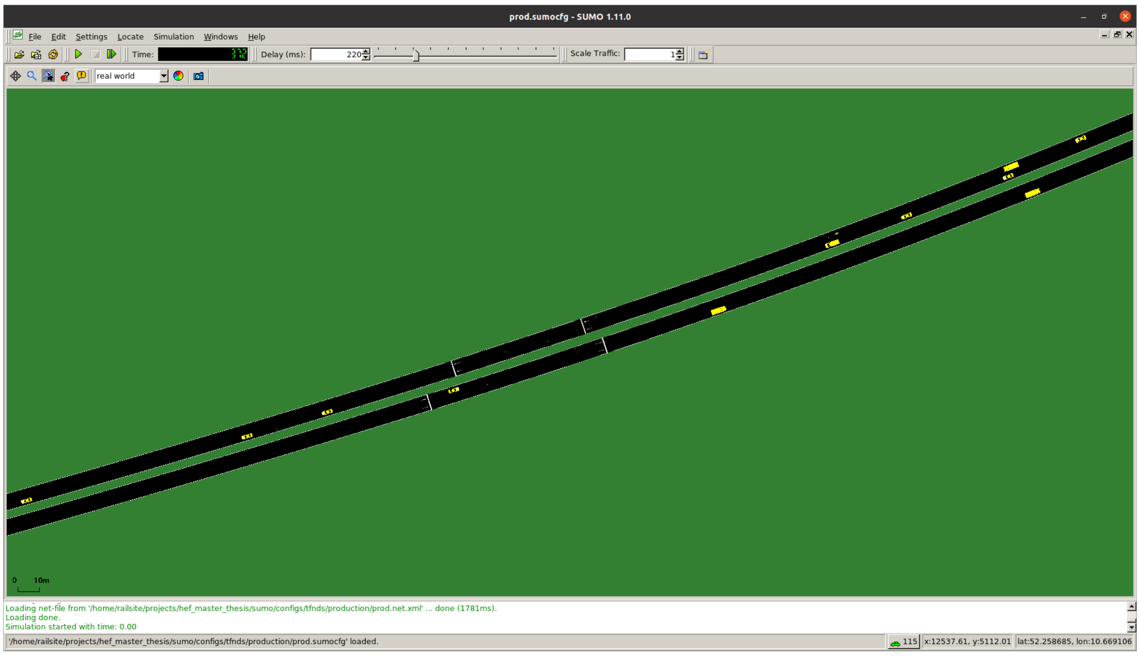


Figure 2.4: SUMO GUI

- Microscopic: models each vehicle on its own. Mainly by the behavior of how one vehicle follows another and when it performs a lane change.
- Macroscopic: in contrast, it does not model the dynamics of individual vehicles but the traffic flow, for example, by average speed and traffic density.
- Mesoscopic: a mixture of microscopic and macroscopic modeling.

Additionally, traffic simulations are also classified as submicroscopic [MBWvA+20]. These types extend the microscopic vehicle models, for example, by the yaw value (also called heading) and thus the lateral positioning of the vehicle. In contrast, a microscopic traffic simulation mainly models the longitudinal behavior of vehicles along a lane in combination with lane-change decisions [MBWvA+20]. For this work, microscopic simulators are of interest because they allow the individual observation of single road users.

SUMO

SUMO is an open-source microscopic traffic simulation [LWB+18]. It is developed by the DLR and provides a fully-featured traffic simulation suite for various use cases [BBEK11]. The main component: the simulator is either executed with a **G**raphical **U**ser **I**nterface (GUI) or with the **C**ommand **L**ine **I**nterface (CLI) with only the simulator. The GUI is shown in Figure 2.4.

In order to execute the simulation, SUMO provides a tool to generate networks (netgenerate), edit (netedit), or import road networks (netconvert). These road networks are stored in a SUMO specific e**X**tensible **M**arkup **L**anguage (XML)-format. Additionally, a XML route file is required to start the simulation. This file contains the information about the vehicle types, their parameters and the traffic flow. SUMO provides different ways to describe traffic flows. The Listing 2.1 shows the method used in this thesis.

Listing 2.1: SUMO route configuration file

```
1 <routes>
2     <vType id="pkw0" vClass="passenger" speedFactor="1.0" tau="1.5" probability="0.5" />
3     <vType id="pkw1" vClass="passenger" speedFactor="1.5" tau="0.66" probability="0.2" />
4     ...
5
6     <vTypeDistribution id="td0" vTypes="pkw0 pkw1 ..." />
7
8     <flow id="0" type="td0" begin="0" end="100" vehsPerHour="500" from="edge0" to="edge0" />
9 </routes>
```

First, two different vehicle types with the tag <vType> are created. These refer to a set of vehicles that have the same parameters in common. The attributes shown in Listing 2.1 Line 2 tells SUMO, the vehicles of type `pkw0` are from the class `passenger` and typically drive at the speed limit (`speedFactor="1.0"`). While vehicles of type `pkw1` (Line 3) will drive at 150 % of the speed limit. The parameter `tau` indicates the desired time headway for this vehicle type. Finally, the setting of the parameter `probability` results in 50 % of the inserted vehicles being of this type. The parameters shown describe only a subset of the available parameters. The <vTypeDistribution> (Line 6) tag tells SUMO to use a distribution to insert vehicles (instead of specifying individual vehicles and routes). Eventually, the traffic flow is defined by the tag <flow> (Line 8), it describes that between simulation times $0 - 100$ s, 500 vehicles travel between the edges `edge0` and `edge1`. These are generated according to the specified vehicle type distribution.

The output of the simulation is stored in an XML-file (with the possibility of different levels of detail). There are two **A**pplication **P**rogramming **I**nterfaces (APIs): **Traffic C**ontrol **I**nterface (TraCI)[1] and LibSUMO[2] which are designed to programmatically interact with SUMO. These APIs allow dynamic interaction with the simulation environment and bypass static file-based configuration. TraCI is a control interface for SUMO over network. LibSUMO is a library to directly interact with SUMO via C++ function calls. TraCI is the recommended[2] way to interact with SUMO since it is more flexible in terms of multiple platforms and languages. LibSUMO has, in comparison, a lower overhead which increases the performance drastically when used, for example, to collect datasets. With both APIs, it is possible to manipulate single vehicles and change their parameters. These APIs also allows the collection of the generated output directly into other data formats.

[1] https://sumo.dlr.de/docs/TraCI.html
[2] https://sumo.dlr.de/docs/Libsumo.html

SUMO simulates the vehicles time-discrete and space continuous [BBEK11]. This behavior leads to arbitrary positions of vehicles within the time-discrete domain. The global simulation parameter `--step-length` determines the time interval.

Microscopic Traffic Modelling

Microscopic traffic simulations break down the behavior of individual drivers in terms of how they follow another vehicle and when they make a lane change [Kra98]. The longitudinal behavior is explained by the **C**ar of **F**ollowing **M**odel (CFM). The CFM will determine the speed behavior through time. Vehicles are modeled to drive at the desired velocity v_{des}. It is assumed that they change their velocity only when they deviate from v_{des} [Kra98]. Given a relaxation time τ and the current velocity v, Krauss [Kra98] defines a basic CFM as:

$$\frac{dv(t)}{dt} = \frac{v_{des} - v}{\tau} \tag{2.1}$$

According to Equation 2.1 a velocity change is also introduced by a change of v_{des}. Car-following aims to model the interaction between vehicles. Therefore, the target is to model the velocity change introduced by other vehicles. The interaction is limited to the vehicle ahead in almost any CFM [Kra98]. Currently, SUMO support 16 CFMs[3]. In the context of SUMO, each CFM has two parameters in common. These are: τ (different from Equation 2.1) and *minGap*. τ denotes the desired time headway of a following vehicle and *minGap* the minimal gap between follower and leader at standstill. The concrete interpretation of these parameters may change by individual models. The speed of the following vehicle v is used to infer a desired distance by:

$$d = minGap + v \cdot \tau \tag{2.2}$$

Further parameters of the concrete follow behavior are dependent on the CFM.

According to Krauss [Kra98], lane-changing behavior is far less researched in comparison to CFMs. This behavior is based on a rule set that determines when the lane change should be performed [Kra98]. SUMO uses a lane change model from Erdmann [Erd15] by default. This will perform lane changes instantaneously. In order to model lane changes as smooth changes and accurate lateral movement, the sub-lane model has to be activated.

Calibration

Calibration is the process of modifying the simulation parameters to reduce the difference between simulation and reality [CLOR03b]. Therefore, it is necessary to have ground-truth data and measurements to compare it against simulation-generated data. This process is carried out until conformity between simulation and reality is achieved [CLOR03b]. Wagner and Antoniou [ABB+14] define the final goal as "minimizing the difference between reality [...] and the model results [...]" [ABB+14, p. 59]. Different guidelines are available on how to calibrate traffic simulations [ABB+14]. Wagner and Antoniou [ABB+14] summarize that the following four steps should be performed to use a traffic simulation:

[3] https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#car-following_models

1. Building / collecting the road network
2. Calibrate the traffic flow, demand, and capacity
3. Fine-tune parameters: for example, car-following, lane-changing
4. Validation with a different dataset used for calibration

3. Concept

This thesis aims to generate realistic scenarios within a simulation environment and assess their realism. To simulate, we need to determine the parameters of the simulation environment. The challenge is identifying methods to determine these parameters and defining what realistic means in scenario-based testing. We present in this chapter a generic concept that allows the generation of synthetic scenarios and compares these in terms of realism. Therefore, we use a simulation environment to generate the synthetic scenarios. We compare the synthetic scenarios against real scenarios by examining the trajectories within the scenarios. In Figure 3.1 we present the overall concept. The concept consists of trajectory generation (blue) and evaluation (orange). During parameterization, we use the real dataset to generate simulation parameters. We use these parameters to generate synthetic trajectories via the simulation environment. This process is described in Section 3.1. For evaluation, we use the generated synthetic trajectories for comparison with the real dataset. We use characteristics that describe the trajectories within these datasets for comparison. The evaluation process is described in Section 3.2.

3.1 Synthetic Trajectory Generation

Within this section, we present a concept to generate synthetic trajectories. We use a simulation environment to generate these trajectories. In order to simulate, simulation parameters must be determined. We use a real dataset in order to determine these parameters. We present a concept to generate synthetic scenarios consisting of four steps: dataset preparation, determining the simulation parameters, simulation, and OpenSCENARIO export. In Section 3.1.1 we introduce the concept to prepare the synthetic and real datasets (Figure 3.1 Step 1). The result is syntactically and semantically identical datasets that we use in the further steps. We show in Section 3.1.2 different methods for parameter determination and use them to generate sets of simulation parameters (Figure 3.1 Step 2). With the parameters found, we generate synthetic datasets by simulation in Section 3.1.3 (Figure 3.1 Step 3). Finally, in Section 3.1.4 we introduce an export to a generic data format for further use of the synthetic results (Figure 3.1 Step 4).

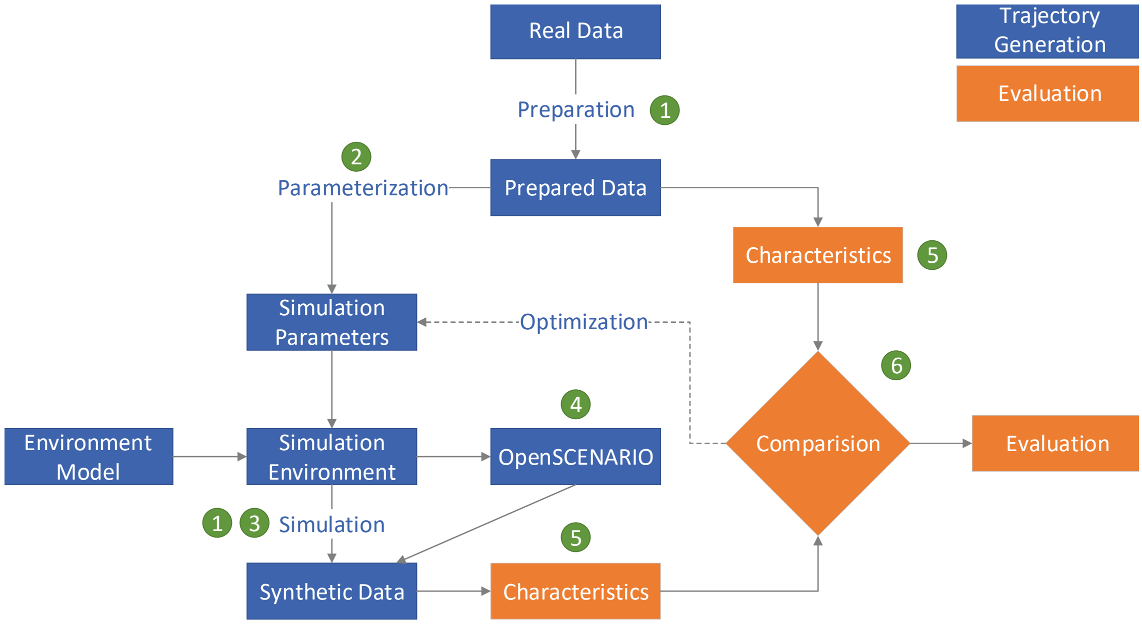


Figure 3.1: Overview of the entire concept

3.1.1 Dataset Preparation

First, we prepare the datasets to determine simulation parameters and efficiently compare synthetic and real data. To compare the datasets, both must be syntactically and semantically identical. However, the synthetic and real datasets differ in syntax and semantics. We present a procedure to harmonize these datasets and prepare them to be compared. In the following paragraphs, we first briefly describe the datasets.

We use ground truth data from the testbed Lower Saxony [KMKL18]. The testbed Lower Saxony recognizes these objects live and inserts them into a database at each detection step. The resulting database has a shape similar to the table shown in Table 3.1. The *Identifier* and *Time Step* columns form a unique combination for a specific vehicle at a specific time. A single trajectory is given by every row where the *Identifier* matches the vehicles' identification. The detection system is designed to detect objects live. The inserted data corresponds to the best currently available knowledge about the objects' past. When a vehicle is being tracked on the highway, the system collects further information every time step. After the vehicle has passed through the testbed, the most reliable measurement of object information, such as dimensions, is available. This leads to the fact that the initial inserted measurements do not contain the best information.

The columns of the simulation dataset are similar to that of the real dataset (Table 3.1). A difference is the representation of the data within the columns. For example, the simulation environment locates objects in a local coordinate system, and the real data is in the UTM coordinate system. Therefore, a coordinate transformation must be performed to compare these datasets.

Figure 3.2 shows an example scene with a car following a truck. We use this example to explain the preparation process. We calculate later in this thesis characteristics

Identifier	**Time Step**	**X**	**Y**	**Vel-X**	**Vel-Y**	**Class**	**Width**	**...**
0	0	200	400	10	30	Car	180	...
1	0	300	100	12	35	Car	185	...
0	1	205	405	-5	-20	Truck	240	...
1	1	295	90	15	25	Van	215	...
...	...	...	...	...	...	...	...	...

Table 3.1: Example representation of a trajectory dataset in tabular form.

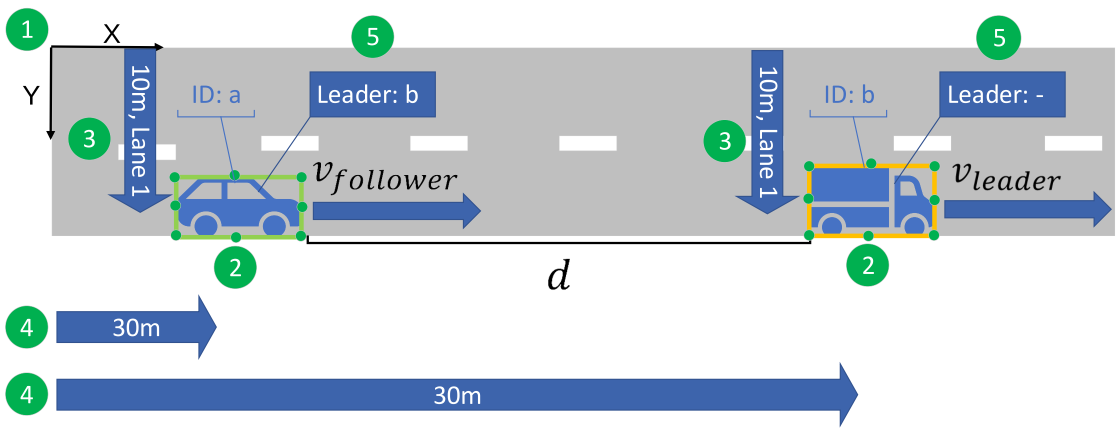


Figure 3.2: Visualization of the necessary preparatory steps for processing the datasets.

that depend on positions and velocity within the lane coordinate system. Therefore, we calculate these positions and velocities within this process. The process consists of five steps: data transformation, data enhancement, lateral position assignment, longitudinal position assignment, and leader assignment. In the following paragraphs, we describe these steps in detail.

First, we perform the data transformation (Figure 3.2 Step 1). After this step, we use the same process for synthetic and real datasets. This step consists of multiple sub-steps. First, we perform a coordinate transform of the simulation data to align the coordinate systems. The real dataset contains the velocity and acceleration as a vector in two directions within the global coordinate system (Table 3.1). In the further steps, we need these attributes as measurements in the direction of travel. We add the magnitude of the vector as an additional column. Finally, we inspect all real trajectories, calculate the most reliable measurements of dimensions and classification over the entire detection cycle, and apply those to the dataset. After this step, we have two datasets with the same columns and data representations. This process allows the use of the following process chain for both datasets.

For the following steps, we assume that we want to compute a TTC. The TTC measures the time if a collision is predicted until the collision occurs. To calculate the TTC, we need multiple input values that are not available. First, we need to know that the car follows the truck to predict a collision. Then, we need the distance and the velocity of both to calculate the TTC. We assume in Figure 3.2 that the car drives faster than the truck because only then a TTC is defined. To calculate the TTC, we need to extend the dataset with the required information. First, the outer points of the vehicle must be determined to calculate the correct distance between both (Figure 3.2 Step 2). The real dataset describes the vehicles with a reference position and additional extents in each direction along the direction of travel. The simulation data uses the center position of the front bumper as a reference and the length and width to represent the object boundaries, where the underlying lane implicitly gives the direction of travel. With this information, we add the outer points to both datasets. To calculate a leading vehicle, the information on which lane the vehicles are driving is needed. We calculate this information for each time step and vehicle and add it to the database (Figure 3.2 Step 3). Finally, we need the longitudinal position along the lane because we need to measure the distance between both vehicles. We calculate this position and add it to the database (Figure 3.2 Step 4). Based on the known longitudinal position within this lane, we identify the leading vehicle (Figure 3.2 Step 5). We calculate the distance between both vehicles using the previously calculated outer points of the vehicle and their position within the lane. We identify the velocity of the leading vehicle using the known leading vehicle and the distance. With these attributes (distance and velocity of the following and leading vehicle), we calculate the TTC. Finally, we have two datasets that differ only in the trajectories they contain, but the format is the same.

3.1.2 Determining Simulation Parameters

Input parameters determine the simulation behavior. These parameters must be determined before simulation [ABB+14]. We distinguish these parameters between static and behavior parameters. Behavioral parameters determine how vehicles behave and interact with other road users, such as the desired speed. The static parameters have no direct influence on the driver's behavior, for example, the road model. We keep the static parameters the same during these runs to obtain comparable results between the simulation runs and the real dataset. However, we need to determine the behavior parameters. These parameters are the parameters of the CFM. In this section, we present methods for determining the behavioral parameters. Therefore, methods for determining these parameters are required. We conducted a literature review to examine existing methods for determining traffic simulation parameters. The most prominent method in literature for determining the simulation parameters is optimization [KT08a, LXABA15, PBF+17, CLOR03a, HAB+15]. This method aims to optimize an objective using an algorithm. Similar use-cases use expert-based methods to determine simulation parameters [CJSM21]. An expert provides the necessary knowledge and experience and combines the literature results to determine the parameters. In the literature, there are several works with already determined parameters that the expert can use [LFAR19, LHP+21, KLY+21, SNB+20, SKvA12]. The last method is a clustering-based approach [MA07, HA14]. It aims to find different driving styles in real data and model them in simulation. Therefore, we use the following three methods to compare which method is best at generating realistic trajectories:

- Expert-based
- Optimization-based
- Clustering-based

Each method takes the dataset as input and outputs the simulation parameters. As an exception, in the expert-based method, an expert also takes literature and his knowledge as input. We use the three different methods: expert-, optimization-, and clustering-based, to generate three different sets of parameters for vehicle behavior. In the following paragraphs, we explain the parameterization methods in detail.

Expert-based parameterization

The expert-based method is a manual process in which an expert determines the parameters of the simulation environment. This method does not require any reference data. However, an expert is required who provides the required expertise. The expert must know how the simulation behavior is determined and what realistic parameterizations are. It is a process of direct inference rather than an iterative process. The expert uses his experience, knowledge, and literature to determine the parameters. These factors are decisive for the result of this method. In addition, the expert analyzes real data using statistical methods. For example, one parameter of all CFMs in SUMO is τ, the time headway. The expert examines the minimum or mean time headway for different vehicle types in the real dataset and sets observed values directly in the configuration of SUMO.

Other parameters such as *minGap* cannot be directly observed (in a highway setting) and easily determined. The *minGap* describes the desired minimal gap between two vehicles at a standstill. It is easy to determine this parameter in urban traffic, while it cannot be observed on the highway when all vehicles are in free flow and no traffic jam occurs. Since the desired gap gap_{des} in SUMO is modeled as the sum of the minimal gap $minGap$ and a function f that depends on the actual vehicles' velocity v, it is essential also to specify this parameter. In SUMO the desired gap increases with an increase of the vehicles' velocity. This behavior is formally defined as in Equation 3.1.

$$gap_{des} = minGap + f(v) \tag{3.1}$$

Since this parameter cannot be observed in the real dataset, the expert uses his knowledge about this behavior or the literature. For example, SUMO provides a list of default parameters for each vehicle type[1]. This list summarizes literature or publicly available information and sets these as default values.

Furthermore, the expert uses his knowledge and experience to evaluate the resulting parameters. He uses his knowledge about realistic bounds to judge if the results are realistic. If these are unrealistic, he chose them explicitly differently.

The strengths of this method lie in the direct derivation of parameters from the ground truth dataset and the human ability to judge plausibility. All publicly available (and privately accessible by him) information and literature are at the disposal of the expert. This information provides a massive basis for the determination of the parameters. This method is also applicable when little or no real data is available. It is possible to determine the parameters based on the literature, knowledge, and experience. In contrast, this method requires manual effort, and different experts have different opinions, leading to different results and plausibility assessments. It is unclear which opinions and literature are correct or ideal for this setting with all available information. The expert also needs experience and knowledge in this particular setting to achieve realistic results.

Optimization-based parameterization

We use optimization as the second method for parameterization. It is an iterative approach to determine an optimum given an objective. An objective function expresses the objective. In the context of this application, we specify an objective function that describes realism. Therefore, we minimize the discrepancy between simulation and reality.

In Figure 3.3, we present the abstract optimization approach. First, we chose a set of initial parameters. The parameters are based on previous work or are arbitrarily chosen. Within step 1 of Figure 3.3, we select a subset of these initial parameters. Miller [Mil09] shows that many input parameters from another simulation environment are eliminated with this step since they have little or no effect on the objective function. We assume that there are also parameters in this setting that can be eliminated. Ros-Roca et al. [RRMB17] find that increasing system complexity (many input and output parameters) also leads to an increase in optimization complexity. Therefore, we argue to limit the number of input parameters to reduce the system's complexity. In the second step of Figure 3.3, we simulate using the selected parameters. After the simulation is complete, we evaluate the run against the objective

[1] https://sumo.dlr.de/docs/Vehicle_Type_Parameter_Defaults.html

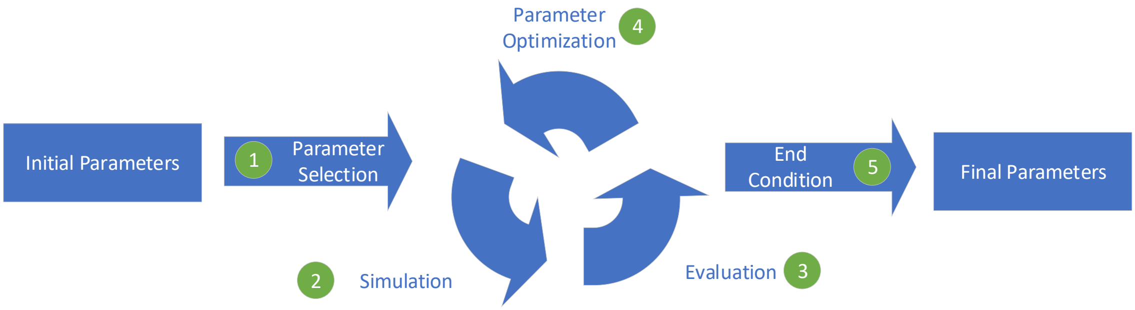


Figure 3.3: Parameter optimization for determining simulation parameters.

function in the third step of Figure 3.3. Based on this evaluation result, we perform an optimization of the input parameters in step 4. A specific optimization strategy determines how the optimization is performed. For example, a gradient of the objective function is calculated based on small variations of the input parameters. The gradient is then used to determine new parameters that are likely to be better evaluated based on the gradient. By the optimization, we generate a new set of parameters to repeat the whole process. In the fifth step of Figure 3.3, we repeat the process until one of two conditions is met: the maximum number of iterations or the change in evaluation is less than a threshold. Finally, we receive a parameter set that leads to a minimum or maximum of the objective function. Since the objective function can be arbitrarily shaped, there is no guarantee that the minimum or maximum is found in the global minimum or maximum. Therefore, the optimization method must deal with the problem of ending in local minima.

In the following paragraphs, we explain the details of the steps. First, we define our objective within optimization. Our goal is to generate realistic trajectories. Therefore, we must define the realism of the trajectories as our objective. Our objective function thus describes the discrepancy between simulation and reality. We compare the synthetic trajectories with the real trajectories to measure the discrepancy. However, a challenge is the different viewpoints. A trajectory is an individual path taken by an object. To compare two individual trajectories, they need the same environmental factors. For example, the same starting point and the same road users are necessary to compare trajectories directly. Small changes within these factors, such as a slightly faster crossing vehicle, could prevent an accident and lead to a different scenario. Therefore, the same environmental conditions within the simulation are necessary. However, our goal is not that the same trajectories occur but that these trajectories behave like real ones. We propose a method using characteristics of these trajectories to compare them. For example, a characteristic is the average velocity of the trajectories, and we compare whether this matches real data. We define multiple characteristics and compare them with reality. This comparison leads to an error between simulation and reality, which we express as an error function. This error function is the objective function that we want to minimize. Thus, the goal is to minimize the error and find a global minimum of an objective function f given a set of continuous β and discrete parameters γ. The objective function measures the result of a simulation run $\boldsymbol{M}^{sim}$. Using $\boldsymbol{M}^{obs}$ as the measurement within the observed data, Ciuffo and Punzo [CP14] formulate the minimization problem as follows:

$$\min_{\beta,\gamma} f(\boldsymbol{M}^{obs}, \boldsymbol{M}^{sim}) \tag{3.2}$$

For the purposes of this thesis, it is not the value of the minimum that is of interest, but the set of parameters β, γ that gives the minimum of f in the set of all possible parameters B, Γ. Therefore, we formulate the problem as follows:

$$\underset{\beta \in B, \gamma \in \Gamma}{arg\,min}\ f(\boldsymbol{M}^{obs}, \boldsymbol{M}^{sim}) \tag{3.3}$$

Moreover, Ciuffo and Punzo [CP14] extend the problem by introducing constraints. Let m be the number of classes within the simulation, $\boldsymbol{l}_{\beta,\boldsymbol{i}}, \boldsymbol{l}_{\gamma,\boldsymbol{i}}$ define the lower bound and $\boldsymbol{u}_{\beta,\boldsymbol{i}}, \boldsymbol{u}_{\gamma,\boldsymbol{i}}$ the upper bound for a given continuous or discrete parameter. Ciuffo and Punzo [CP14] formally describe these constraints as:

$$\boldsymbol{l}_{\beta,\boldsymbol{i}} \leq \boldsymbol{\beta_i} \leq \boldsymbol{u}_{\beta,\boldsymbol{i}},\ \ i = 1, ..., m \tag{3.4}$$

$$\boldsymbol{l}_{\gamma,\boldsymbol{i}} \leq \boldsymbol{\gamma_i} \leq \boldsymbol{u}_{\gamma,\boldsymbol{i}},\ \ i = 1, ..., m \tag{3.5}$$

In order to find the minimal parameters $\beta_{min}, \gamma_{min}$ it is necessary to define the objective function f. To achieve the best results in the final evaluation, f should be chosen to be the evaluation function. Therefore, this method aims to directly find the best possible parameter set for the aspects under study.

We design the objective function to minimize the discrepancy between simulation and reality. A critical part of this method is the runtime (since thousands of iterations are performed). Therefore, we choose the objective function so that it is computed within seconds. We use the discrepancy between simulation and reality in TTC, DRAC, and TH as the optimization objective. This choice introduces the challenge of multi-objective optimization. Different strategies have evolved to deal with this challenge. In this thesis, we use goal programming [Diw20] to address this problem because it is an easy-to-implement approach. In goal programming, a combined objective function $f(C_1, ..., C_n)$ is constructed from many objective functions for individual characteristics f_{C_1}. This choice introduces another challenge: the combined objective function must be defined. Therefore, we have to define weights for the three individual objectives (TTC, DRAC, TH). Since we currently have no insight into which objective influences realism the most, we weigh them all equally. Since the error between those characteristics differs in magnitude, we normalize each characteristic towards the error computed with the default parameters. Thus, at a value of one, the objective expresses equally realistic results as the default configuration below one, better than the default, and higher than one, worse than the default. With E as the error p_i the current optimization parameters and p_d the default parameters, we define the objective as follows:

$$Objective = \frac{E(p_i, TTC)}{E(p_d, TTC)} + \frac{E(p_i, DRAC)}{E(p_d, DRAC)} + \frac{E(p_i, TH)}{E(p_d, TH)} \tag{3.6}$$

To reduce optimization effort, we reduce the complexity of the individual optimization steps. Therefore, we limit the number of parameters that are optimized. Inspired by Henclewood et al. [HSR+17], we identify uninfluential parameters. These parameters are excluded from the optimization process.

When performing optimization, boundaries are used, on the one hand, to limit the exploration space (and reduce the effort) and, on the other hand, to eliminate implausible states [Adb13]. For example, the optimization will choose a very low desired time headway if this is a minimum of the objective. This value is the optimal result concerning the objective function, but it is not realistic that every vehicle will drive with a very low time headway. Therefore, we define boundaries for each optimizable parameter.

Theoretically, this method will always be the best available method to find the optimal parameters given the evaluation objectives. Some challenges make it difficult to achieve the best possible result in practice. First, the global optimum is not easy to find. Some methods employ a greedy strategy (for example, gradient descent), so they are likely to end up in local minima within spaces with many local minima. Each minimum has to be examined or analytically proven to tell if a minimum is the global minimum [Adb13]. In an infinite parameter space, this is often impossible in practice. Therefore, if the optimization ends with reaching a minimum, it is not easy to tell if this is the best solution. The optimization strategy must consider this and use methods to bypass local minima.

Without the final evaluation of this method in mind, this method benefits from an automated process and less manual interaction. In addition, it is possible to use any objective function. This method will produce an optimal solution concerning the objective function within practical limits. In contrast, practical problems must be addressed: for example, local minima or the constraints on the optimization parameters. Furthermore, the objective function must express the final evaluation target to achieve good results. This relationship leads to practical issues when the final evaluation is time expensive. Since this method uses an iterative approach and requires multiple executions, each iteration must be time-efficient. This consideration also applies to all steps in any optimization iteration, especially simulation. Therefore, additional effort must be spent to reduce this calculation time.

Clustering-based parameterization

In contrast to the methods shown so far, we distinguish driving style by this method (passive, aggressive) for a vehicle class (for example, car, van, truck). We identify different driving styles in reality and introduce them into the simulation. Therefore, we need to identify the driving styles. We use a clustering-based approach to determine driving styles within a vehicle class. As with any other method, we use the reference dataset as the basis. For example, to determine the desired time headway, we analyze the dataset from this aspect. Then, we cluster the results to find dense regions within the dataset with all desired time headway observations. We assume that different classes of drivers behave, for example, with a different desired time headway. We map these driver classes in the simulation with different behavioral parameters. This will lead to multiple SUMO-`<vType>` attributes (Section 2.3) for a single generic vehicle class. The general procedure is shown in Figure 3.4. First, we characterize the trajectories and represent them in a feature space. Then, we cluster the attributes and determine the simulation parameters from these clusters. The following paragraphs will explain the procedure in detail.

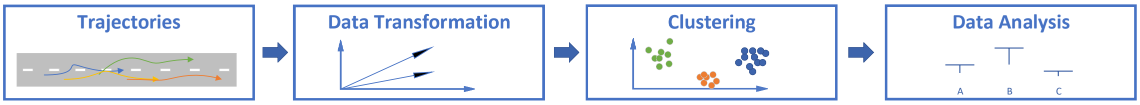


Figure 3.4: Clustering-based approach to determine driver behavior within the real dataset.

We assume that there are different driving styles in reality and that they are distinguishable from each other. In order to distinguish driving styles from each other, we need to identify the characteristics of these driving styles. Therefore, we define a set of characteristics, such as minimal time headway, minimal TTC or the number of lane changes. We calculate these characteristics for each trajectory observed in the real data. This calculation will result in a dataset containing a list of every trajectory's characteristics. This dataset is high-dimensional with important and unimportant characteristics. Since the unimportant characteristics do not provide additional information, we eliminate them. Therefore, we transform the dataset into a latent space using a dimension reduction strategy. This latent space is still high-dimensional but with fewer dimensions than the original dataset and less redundant information. We use a clustering strategy to find clusters of similar data points and assign each data point to a cluster. A cluster semantically represents a group of people who behave similarly in the observed traffic and, therefore, a driving style. The cluster describes the driving style within a latent space. This latent space has no direct connection to the parameters of the simulation environment. Therefore, we need to derive these parameters from the clusters we found. We link a cluster to the original characteristics by the original dataset. With these links, we describe the found clusters. We use the subsets of the characterized trajectory dataset associated with the same cluster to derive the parameters using statistical methods, for example, the mean time headway within a cluster. Each cluster found leads to its own set of parameters. We use these parameter sets to parameterize the simulation and have thus mapped the real driving styles in the simulation, assuming that the selected characteristics are representative of the driving style.

In comparison to the expert-based method, the clustering method is fully automated. The number of clusters found is a customizable parameter depending on the concrete cluster method. Thus, the granularity of the different driving styles is configurable. One option is to add or remove characteristics to improve the results. This extensibility makes this method flexible. In contrast, there are some challenges. For example, the minimal time headway as input characteristic has to be selected manually. It is not clear whether these characteristics reflect a driving style. Also, it is not known how the latent space is structured, and it is unclear which clustering strategy works best in this space. Since the clusters found have internal variance, the derivation of the simulation parameters (for example, mean) may not adequately map the driving style into the simulation. There is no guarantee that a dense cluster within the latent space with low variance will have low variance within the mapped parameter space of the simulation. Therefore, parameter determination from clusters is a critical point. Furthermore, the best choice for cluster granularity is not apparent, and it is not clear how to decide which number of clusters will produce the most realistic results.

3.1.3 Simulation

In this section, we present the concept to simulate traffic. We use the generated parameter sets by an expert, optimization, and clustering-based approach for the vehicle behavior. We specify the parameters that we need, besides behavioral parameters, to simulate traffic with SUMO. These parameters are step length, road model including signage, traffic flow, and simulation duration. The step length determines the simulated duration in each iteration by the simulation. The road model describes the street network and includes the signage, for example, speed limits. Traffic flow specifies how many vehicles drive on the routes between points within the road network. The simulation duration determines the time simulated by the simulation environment. We keep these parameters the same for all simulation runs to obtain comparable results.

The first parameter is the step length. It determines the simulated duration in each simulation step. This parameter influences the frequency of the output, position updates of the vehicles, and vehicles' behavior. For example, with a step length of 1 s, the vehicle's velocity change and position are updated each second according to the CFM. Compared to a shorter step length, the decision to accelerate or decelerate is delayed. Thus, the vehicle behaves differently. To compare the synthetic and real datasets, we chose the step length to match the sampling frequency of the real dataset. In order to model the delayed driver's behavior, we use the "action-step-length" parameter[2] within the parameterization process. This parameter decouples the CFM update from the simulation frequency.

In order to simulate traffic, we need to specify the road model. We use an existing high-resolution map of the testbed Lower Saxony as input. This map ensures comparability between the trajectories generated by the simulation and the real trajectories. We keep the road model constant during all simulation runs. The road model also contains the traffic signs. The speed is not restricted on the testbed Lower Saxony (except for a small part that is ignored because it restricts the speed on 1 km from 7.45 km on one lane and only at specific hours). Therefore, the simulation should not include speed limits. In SUMO, vehicles will always travel at the highest possible speed[3], which is limited by the maximum speed allowed on the road and the maximum speed of the vehicle class. When interactions with other road users occur, the speed is adjusted according to the CFM. It is not realistic that vehicles will always drive at the maximal possible speed. Therefore, this behavior needs to be adjusted. Instead, it is a common assumption that drivers have the desired speed [BJEZB13]. We introduce a speed limit of 100 kph within the road model to control this behavior. By setting this speed limit, vehicles will drive at a maximum of 100 kph but always try to reach this speed. Therefore, we introduced the desired speed of 100 kph. However, this desired speed is static for each vehicle. SUMO allows setting a speed factor of vehicles depending on the speed limit. For example, setting a speed factor of 1.2 with the given speed limit of 100 kph achieves the desired speed of 120 kph.

[2]https://sumo.dlr.de/docs/Simulation/Basic_Definition.html#defining_the_action_step_length

[3]https://www.eclipse.org/sumo/

The parameter traffic flow describes the number of vehicles traveling on a route between two points within the road network. With an increasing traffic flow, the number of interactions between road users also increases. The higher traffic flow results in different trajectories since vehicles have to change lanes or decelerate when vehicles are ahead. Nevertheless, this parameter does not describe the behavior of the driver itself. Therefore, we set the traffic flow to a constant value for each method.

We define a fixed simulation duration to achieve optimal comparability between simulation runs. We select this duration according to the duration of the real dataset. We parameterize the simulation environment with these environment settings and generate a synthetic trajectory dataset. Like the real dataset, we process the resulting dataset and add more information, such as leader assignment. The necessary steps are already shown in Section 3.1.1. With these static parameters and the behavioral parameter sets from the previous step (Section 3.1.2), we generate three synthetic trajectory datasets for each parameterization method using SUMO.

3.1.4 OpenSCENARIO Export

We export the trajectories into a standard format, which can be used for further research in scenario-based testing. SUMO provides its simulation output in a custom format that cannot be used directly in other simulators to apply scenario-based testing. In this section, we convert this output into a standard format.

In the context of scenario-based testing and simulation, "OpenX" standards[4] have evolved that describe the input required for scenario-based testing. The advantage of these standards is that they are publicly available and are understood by various simulators, for example, CARLA[5], CarMaker[6] or VTD[7]. They describe different parts (road network, road surface, and driving maneuvers) of scenarios in XML. The OpenX standards are developed by the **A**ssociation for **S**tandardization of **A**utomation and **M**easuring Systems (ASAM). In order to model the dynamic objects and the environment, the OpenSCENARIO standard was established. This standard resembles the L4 and L5 within the 5-level model of scenarios by Bagschik [BMKM18]. It is primarily designed to describe driving maneuvers with multiple road users and includes, for example, weather or environmental models.

Since the OpenSCENARIO standard is well established within scenario-based testing, we use the standard to store the scenarios generated by the simulation environment to use the scenarios in further research. Even though the standard has many features, for example, dynamic vehicle interaction and triggers, we use static trajectories for each participant. Thus, we ensure that the scenarios that are considered realistic are also realistically represented in other simulators. We export the dataset into multiple OpenSCENARIO files. Therefore, we split the dataset into multiple subsets of the same period. For each subset, we create a single OpenSCENARIO file

[4]https://www.asam.net/, 16.03.2022

[5]https://carla-scenariorunner.readthedocs.io/en/latest/openscenario_support/

[6]https://ipg-automotive.com/en/products-solutions/software/carmaker/, https://www.asam.net/members/product-directory/detail/carmaker/

[7]https://www.mscsoftware.com/product/virtual-test-drive, https://www.asam.net/members/product-directory/detail/virtual-test-drive-vtd/

containing all trajectories over the entire spatial domain of the simulation. Within the scenario files, we create entities for each participant. We assign the traveled trajectory observed within the simulation to this entity. Finally, we assign the starting times of the participants within the scenarios. The start point is necessary since not all participants start at the beginning of the scenario. We use relative time within these scenarios to compare them against each other. This export allows the usage within different simulators, for example, to apply scenario-based testing or to evaluate the simulation behavior of other simulation environments.

3.2 Dataset Comparison

In this section, we compare the realism of synthetic scenarios. Our goal is to evaluate whether a synthetic set of scenarios is similar to a real set. The challenge is to evaluate this similarity and, therefore, the realism. Our research focuses on evaluating the realism of the dynamic object, thus the trajectories within the scenarios. In Section 3.2.1 we define characteristics determined within both datasets. We use the characteristics as representatives for the trajectories and evaluate the similarity within these characteristics. Section 3.2.2 describes the comparison of these characteristics, which eventually leads to a measure of realism.

3.2.1 Dataset Characteristics

To evaluate the realism of trajectories, we define characteristics that represent individual aspects. We divide the notion of realism into different parts represented by these characteristics. For example, we evaluate the resulting trajectories to determine whether their behavior of following another vehicle is realistic. In this section, we introduce ten characteristics. We divide the characteristics into three subcategories within the highway setting: longitudinal, lateral, and mixed. Longitudinal characteristics examine behavior within a single lane, for example, interaction with a vehicle ahead. Lateral characteristics inspect the interaction between multiple lanes. The mixed characteristics examine interactions between longitudinal and lateral behavior.

Longitudinal

In the context of scenario-based testing, the testing of critical scenarios is of particular interest [JSW17]. Since critical scenarios occur less frequently, part of the research focuses on the targeted generation of critical scenarios [NKM19, XFX20]. This selective generation introduces a bias within the collected scenarios. Our goal is to generate realistic scenarios. Therefore, we match the criticality between simulation and reality. Junietz et al. [JSW17] present different characteristics to measure criticality. We use the (1) **T**ime **T**o **C**ollision (TTC) and (2) **D**eceleration **R**ate to **A**void a **C**rash (DRAC) as criticality characteristics since they are often used to measure criticality [JSW17]. Furthermore, we add the (3) **T**ime **H**eadway (TH) as a longitudinal characteristic. We use the TH since it is often used for parameterization [AB14], and it is independent of the TTC [Vog03]. Since the TH is often used for parameterization, we assume that it is important to reflect it correctly. In

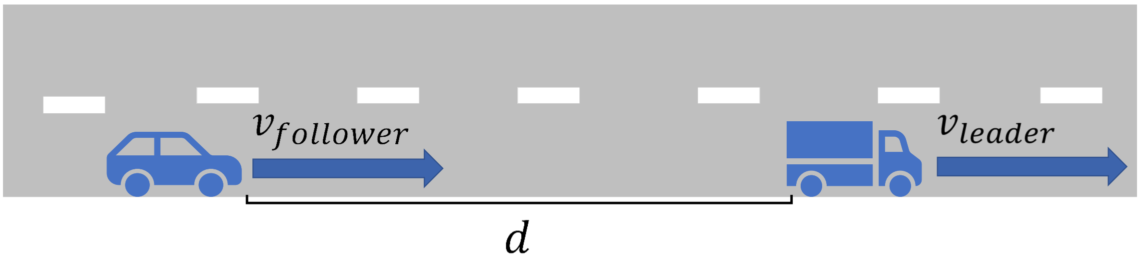


Figure 3.5: Example of a scene where two vehicles follow each other. The car is assumed as the ego vehicle and travels at speed $v_{follower}$. The leading truck drives at the velocity of v_{leader}. The distance between the two vehicles (the front bumper of the car and the rear bumper of the truck) is defined as d.

Figure 3.5, we show a sample situation in order to explain the longitudinal characteristics. Within this figure, we present a scene with a car driving at a speed of $v_{follower}$ following a truck driving at a speed of v_{leader}. The distance between both vehicles is given by d.

The TTC measures the time to collision when a collision is predicted to occur at a given velocity difference. A collision is predicted if the vehicles will collide in the future. Therefore, the future trajectories must be predicted. We assume that the vehicles will stay in their lane on a highway. Thus, we predict a collision if there is a vehicle in front of the ego vehicle in the same lane traveling slower than the ego vehicle. If no collision is predicted, the TTC is not defined. Balas and Balas [BB06] define the TTC according to the attributes of Figure 3.5 formally as:

$$TTC = \frac{d}{v_{follower} - v_{leader}} \tag{3.7}$$

The TTC examines the longitudinal behavior on a highway, and we use this as characteristic $\hat{C}_{TTC}$ to evaluate longitudinal realism.

We use the DRAC as second longitudinal characteristic $\hat{C}_{DRAC}$. This characteristic calculates the necessary deceleration to avoid a collision assuming that a collision is predicted and the vehicle ahead does not change its speed. Similar to the TTC, the value is defined when a collision is predicted, and the leading vehicle drives slower than the ego vehicle. We use the same collision prediction mechanism as for the TTC. Fazekas et al. [FHKO17] define the DRAC formally as:

$$DRAC = 0.5 \cdot \frac{(v_{follower} - v_{leader})^2}{d} \tag{3.8}$$

The TH is also a longitudinal characteristic since it defines in a follow-lead situation the time until the following vehicle is at the position of the leader. A smaller TH leads to less time for the driver to react or brake in an emergency situation. The TH is only defined in a follow-lead situation. Yan and Dianhai [YD12] define the time headway as:

$$TH = \frac{d}{v_{follower}} \tag{3.9}$$

The characteristics $\hat{C}_{TTC}$, $\hat{C}_{TH}$, $\hat{C}_{DRAC}$ can be determined in each time step for each vehicle if they are defined. Thus, a comparison of all values is required to compare these characteristics within two data sets. There are possibilities to compare these characteristics by a combination of all measures for a vehicle (for example, minimum or mean) [OCI15, LBSB13]. However, two trajectories can have the same mean but a different deviation, so the two trajectories are different but would be evaluated the same if we examine only the mean. Since we want to investigate the realism of the entire trajectory, we examine every single measurement. To compare each measurement, we propose a comparison based on the distribution of these characteristics. Therefore, we introduce the characteristics C_{TTC}, C_{TH}, C_{DRAC} by calculating the distribution function DF of each characteristic within the dataset. Finally, this yields the following longitudinal characteristics:

$$C_{TTC} = DF(\hat{C}_{TTC}) \tag{3.10}$$

$$C_{TH} = DF(\hat{C}_{TH}) \tag{3.11}$$

$$C_{DRAC} = DF(\hat{C}_{DRAC}) \tag{3.12}$$

Lateral

In the previous characteristics, we studied mainly longitudinal behavior in highway settings. Since there are also lateral movements, we add characteristics that examine them. First, we add a characteristic of the lane distribution: C_{LD} because it is easy to calculate. This characteristic indicates how many vehicles drive in the main and passing lanes. The results are given in percentages. For example, 30 % of vehicles travel in the passing lane, while 70 % travel in the main lane. A difference within these percentages can have various causes but indicates a difference in the vehicle's behavior. For example, in an overtaking maneuver, a narrower cut-out and cut-in (before and after the overtaken car) will result in less use of the passing lane. Therefore, a systematic shift within the overtaking behavior would result in a different lane distribution. Another cause is the lack of a specific driving style, such as a driver in the passing lane who has no following vehicle and does not turn into the main lane even though he could. This characteristic is directly computable based on the dataset preparation (lane assignment).

In the context of scenario-based testing, maneuvers are of particular interest [ZHP+17, HPS+19, EUA+19]. Therefore, we add characteristic that specifically targets maneuvers. As an example maneuver, we examine lane changing since it is less complex than an overtaking. We assigned a lane to each vehicle at each timestamp in the preparation phase. Thus, we recognize a lane change as a change in lane assignment. In order to allow for the observation of realism concerning further maneuvers, we want to introduce an extensible approach to maneuver detection. Therefore, we propose a method that uses cross-correlation to recognize maneuvers. A reference signal represents the maneuver. We use cross-correlation to recognize this signal within other trajectories with this reference signal. Currently, the trajectories are located within a global coordinate system. Cross-correlation within a global coordinate system detects only patterns at the same global coordinates. However, the detection should be location-independent. Thus, we introduce a location and street independent representation by locating the vehicle within a road

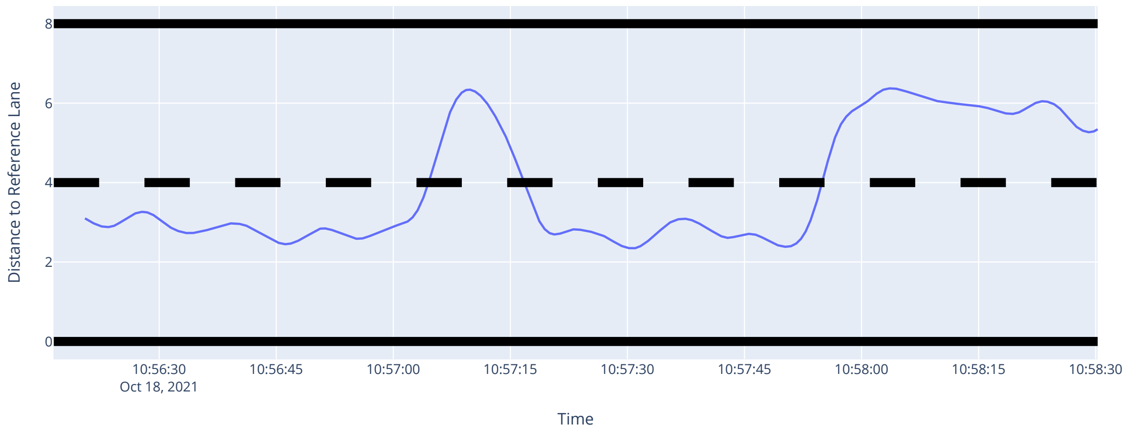


Figure 3.6: Trajectory representation within the local street and time coordinate system. There are two left lane changes (at ~10:57:05 and ~10:57:55) and one right lane change (~10:57:15) performed.

coordinate system. Trajectories are represented by lateral position within a road as a function of time. We assume that the lane changes occur at different speeds in the same amount of time. By using time as a dependency, the representation becomes independent of velocity. A slow-moving vehicle travels much less within the same time as a fast-moving vehicle, and therefore the signal would be compressed. Since the cross-correlation is not independent for compressions within the value range [CNvMH99], we use the time-domain representation. In Figure 3.6, we present a trajectory in the time-street coordinate system. We consider a maneuver as detected if the cross-correlation between the input and reference signals is greater than a certain threshold ϵ. An advantage of this method is that a signal inverse to the reference signal is found. A left lane change leads to an anti-correlation with a right lane change and is also detected with the same reference signal. Therefore, the cross-correlation indicator finds variations of symmetrical maneuvers within the lateral positioning with a single reference signal.

Our final goal is to evaluate realism. Therefore, we need to define a characteristic that evaluates realism based on the detected lane-change maneuvers. Thus, we first propose the occurrence of lane changes as a characteristic. We use the CCLCI to recognize lane changes within the datasets. We calculate the lane changes concerning the total track length within a given period to compare the occurrences. With a given total observation time T, a time unit Δt (for example, 1 hour), an indicator threshold ϵ, a set of trajectories S, and the length of road l the characteristic $C_{|LC|}$ is defined as follows:

$$C_{|LC|} = \frac{\sum_{traj \in S} CCLCI(traj, \epsilon)}{l \cdot \dfrac{T}{\Delta t}} \tag{3.13}$$

The indicator CCLCI returns the number of detected lane changes for a trajectory and a threshold. This characteristic indicates a difference in driving behavior and heterogeneity in vehicle speeds for the same traffic flow. A higher variance in the distribution of vehicle speeds decreases the likelihood for vehicles to travel at the same speed. Thus, lane changes increase because more vehicles overtake. Furthermore, if there is a difference in the decision process for a lane change, this characteristic responds if, for example, vehicles tend to make fewer lane changes.

We further investigate how a lane change is performed to examine the maneuver in detail. Therefore, we investigate the specific lane change behavior. We extract the lane changes with a specific time interval before and after the lane change from the dataset with the CCLCI. We use the dataset to calculate a mean shape for a lane change. The mean shape itself is a characteristic of the dataset C_{meanLC}. We investigate if the mean lane change is modeled correctly with this characteristic. Furthermore, we use the dataset to cluster the lane changes. The clusters represent different variations of the lane change maneuver. Thus, we introduce a new characteristic: the number of lane change variations $C_{|LCvariations|}$. We inspect if there is enough variability within this maneuver with this characteristic. Finally, we inspect the shape of all clusters as a characteristic $C_{LCclustershape}$. Therefore, we compare the typical manifestations of the lane change maneuver. With these characteristics, we study lateral movements in highway scenarios.

Mixed

In the previous sections, we have collected characteristics that primarily examine only one aspect: longitudinal or lateral. We examine characteristics that reflect both aspects to inspect interactions between both. First, we propose a characteristic that describes the contained scenes within a dataset C_{scenes}. Our goal is to inspect which constellations of traffic participants occur. Therefore, we analyze the dataset in terms of the scenes it contains. Since we want to compare the scenes later, we need a generic representation of the scenes that allows minor variations. For example, it is sufficient to know that a scene occurs with two vehicles following each other at a variable distance within meters. In order to define the scenes independent of the road, we use a local street coordinate system. We define a scene as the vehicles within a particular area at a specific time. We round the positions and speeds of the vehicles to specific intervals to allow minor variations. We determine the complete set of occurring scenes for a dataset by cutting out a local environment around each vehicle and time step and grouping the vehicles in it as a scene. We use the collection of all scenes as the characteristic C_{scenes}.

Finally, we introduce a characteristic of the parameterization method rather than the dataset itself. By this characteristic, we investigate the ability of the simulation to simulate individual trajectories correctly. Therefore, we use a random scene S_{real} of the real dataset. We select all vehicles on the whole road at a random time. We initialize the simulation with the real scene S_{real} and with the configuration parameters with the results from the parameterization method. In this step, we adjust the position, heading, speed, and acceleration of the vehicles according to S_{real}. We assign the vehicles in the simulation to a vehicle type in the configuration according to their vehicle class in S_{real}. By this initialization, we run the simulation step after step. In each step, we record the position of the initialized vehicles. For example, we simulate 30 s. thus, we collected a new set of trajectories. We do not change the initial state of the simulation. Thus, vehicles that drive into the testbed are not introduced in the simulation. We limit the simulation duration to reduce the possible effects of these vehicles on introduced vehicles. In order to reduce the

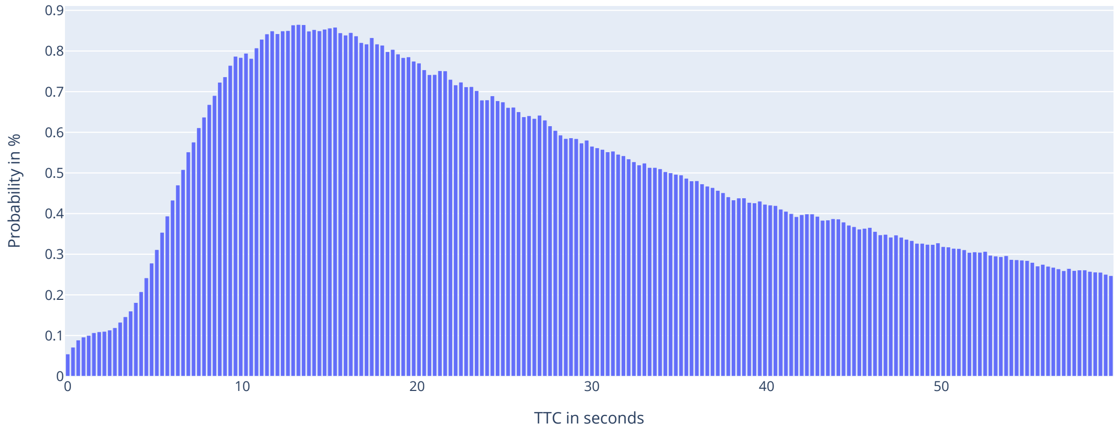


Figure 3.7: Sample real TTC probability distribution shown as histogram with 200 bins. On the abscissa, the TTC value is shown in seconds. The ordinate represents the probability of the occurring TTC value in percent. For example, the bin at 10 s represents that the TTC value of 10 s occurs at about 0.8 % within the dataset.

effect of random events, we repeat this process several times with different starting scenes. This repetition results in a set of trajectory datasets which we use as a characteristic C_{pred} for the configuration.

We collected and developed numerous characteristics that describe the underlying dataset. These characteristics describe different aspects of the trajectories.

3.2.2 Comparison - Realism Metric

Finally, we evaluate the realism of the resulting trajectories by comparing the previously introduced characteristics. The characteristics alone do not reflect reality. For example, we measure a certain TTC value in a dataset, but we are not able to determine realism based only on this characteristic because we currently do not know what a realistic value is. We introduce a comparison step of these characteristics that leads to measures of realism within these aspects. The following sections describe the comparison of the previously presented characteristics.

Longitudinal

We introduced three different longitudinal characteristics: C_{TTC}, C_{DRAC}, C_{TH}, now we compare them within different datasets. These characteristics are defined as the distribution of the occurring values of, for example, the TTC. Figure 3.7 shows a sample TTC distribution determined for a real dataset. The probability of TTC values is shown in a histogram as percentages. Statistical tests are available to compare distributions [Kan06]. These tests are based on a hypothesis that is tested using a test metric. With the test metric applied to the data, the hypothesis can be confirmed or refuted. Our goal is to compare different sets of, for example, TTC distributions and determine which dataset is closest to another. The tests only lead to a binary decision. Therefore, these tests cannot be used directly for our goal. However, the underlying test metric indicates whether the two distributions are similar. The Kolmogorov–Smirnov test [MJ51] is used to test if arbitrary samples follow a given distribution. It is independent of the distribution, for example, a normal or gamma distribution, because it uses the cumulative distribution function as the test metric. Massey [MJ51] defines the test metric d of the Kolmogorov–Smirnov test with a given cumulative distribution $F_0(x)$ and an observed cumulative step-function $S_N(x)$ (since this distribution is observed, the function is discrete and therefore a step-function) as follows:

$$d = maximum|F_0(x) - S_N(x)| \tag{3.14}$$

The maximum distance between the two cumulative distribution functions is used to determine whether the observed sample follows a particular distribution. We want the same generalizability to test if two distributions are similar and know if one distribution is more similar to a reference than others. Thus, we use the cumulative distribution function to compare two samples. In our case, both underlying distributions are observations, and therefore the cumulative distribution function is discrete. With $F_R(x)$ as reference and $F_O(x)$ as observed cumulative distribution step-function, a discrepancy between these two at a certain point x_0, we calculate an error as follows:

$$|F_R(x_0) - S_O(x_0)| \tag{3.15}$$

To compare the full distribution, we sum the discrepancy to an error E between these two distributions for a given set of discrete observations X of:

$$E = \sum_{x \in X} |F_R(x) - S_O(x)| \tag{3.16}$$

We compare two arbitrary distributions without knowing internals like a mean or the variation. However, there is a semantic problem with the definition of Equation 3.16. To explain this problem, we assume three distributions. The first one is the reference distribution. The second is similar but differs by small changes in each observed value. All values occur with the same frequency in the third distribution, except for two values. For example, the value 1 occurs very often, while the value 2 is very unlikely. By the definition of Equation 3.16, the error is equal if the error within these two observations is precisely equal to the sum of the many but minor errors of distribution two. For a random variable, it is more likely that the second distribution is more similar to the third distribution. The third distribution appears to have a huge bias within these two observations, while the variations in the second distribution can be explained by random sampling. Therefore, when comparing a random distribution, a large difference within a few observations should have a greater impact than many small variations. Thus, we introduce a quadratic error term that penalizes these exceptions and finally define the comparison of two distributions as follows:

$$E = \sqrt{\sum_{x \in X} (F_R(x) - S_O(x))^2} \tag{3.17}$$

Under an infinite development of the continuous distribution function, Equation 3.17 converges to Equation 3.14 ($\lim_{x\to\infty}(\sum_{x\in X}(F_R(x) - S_O(x)))^{1/2} = maximum|F_R(x) - S_O(x)|$). This supports our choice of the comparison function, due to the mathematical similarity. We use this error function as a comparison for the characteristics

C_{TTC}, C_{DRAC}, C_{TH}. Now we determine how realistic the simulation results are in terms of the distribution of TTC, DRAC, and TH. Thus, we compare longitudinal the behavior.

Lateral

Now we compare the introduced lateral characteristics (Section 3.2.1). These characteristics must be compared separately since not all of them are distributions. Therefore, we introduce methods for comparison for each of these characteristics. The first characteristic is C_{LD}, which describes the distribution of lane usage. Since it is a distribution, we use Equation 3.17 for comparison.

The characteristic $C_{|LC|}$ describes the total number of lane changes. Similarly, $C_{|LCvariations|}$ determines the number of manifestations of the lane change maneuver. Both characteristics are a single number. Therefore, we compare an observation C_O to a reference C_R of a characteristic by calculating the difference between both $E = |C_O - C_R|$.

C_{meanLC} and $C_{LCclustershape}$ describe a trajectory. These trajectories are represented by lateral position on the road at discrete time intervals. Both characteristics are defined within a certain time interval before and after an event. Therefore, we compare the trajectory by the lateral position within the relative time before and after the event. Similar to the reasoning of the outlier punishment of distributions, we penalize one large discrepancy more than many small ones. Thus, we use a quadratic error term to determine the discrepancy between an observation and the reference. With n as the number of discrete time steps, C_R as the reference characteristic (trajectory), and C_O as the observed characteristic, we formally define the discrepancy as follows:

$$E_{traj} = \sqrt{\sum_{i=0}^{n}(C_R(t_i) - C_O(t_i))^2} \tag{3.18}$$

We directly compare the characteristic C_{meanLC} of two observations since this characteristic describes a single trajectory. In contrast, $C_{LCclustershape}$ describes a set of trajectories for a single observation. Therefore, we need to compare two sets of trajectories against each other for two observations. One challenge is that the number of trajectories in one set may be different from the second. Furthermore, if we match each trajectory from one set directly with a trajectory from the other set by selecting the closest match, it is not guaranteed to evaluate all trajectories. For example, if a set contains a trajectory close to all within the second set, and all other trajectories within the first set are far from it, the matching error can still be small because all calculations are mapped to one trajectory. However, the error should be higher because the first set contains far-off trajectories. Additionally, the error should be independent of the number of contained samples within the set. Comparing two large sets should not automatically result in a higher error than comparing two small sets. We use different steps to solve these challenges. First, we perform a selective matching with the ground truth set. An element of both sets is only selected once. We perform the selection that the summed error of each trajectory match E_{traj} is minimized by testing every combination. After this step, either an empty set (both sets are the same size), a set with remaining trajectories from the ground truth set, or the observation set is left. We use the remaining set to match the closest element in the opposite set. We ensured that every element from both sets was used and matched with the other set. However, the size of the sets still influences the overall error. Therefore, we divide the error by the maximum cardinality of both sets: $max(|S_0|, |S_1|)$, equivalent to the number of matched lane-change maneuvers. Using this method, we compare the cluster shapes characteristic $C_{LCclustershape}$.

Mixed

We compare the introduced mixed characteristics (Section 3.2.1). We need new methods for comparison since they do not fall into any of the comparison methods presented so far (for example, distributions or trajectories). The C_{scenes} characteristic describes the local scenes around the vehicles contained within a dataset. Therefore, the characteristic describes a set of scenes. We use the Jaccard-Index to compare the C_{scenes} characteristic. Rahman et al. [RHB10] define the Jaccard-Index of two sets A, B as:

$$Jaccard(A, B) = \frac{A \bigcap B}{A \bigcup B} \tag{3.19}$$

It calculates the ratio between elements contained in both sets and at least one set. This comparison also considers the number of unrealistic scenes compared to an intersection. For example, the intersection is huge, but the second set introduces many unrealistic scenes. We assume this as beneficial since the unrealistic scenes are taken into account. Since the other characteristics express error, a low value indicates good results. Nevertheless, a low Jaccard-Index indicates poor results. Therefore, we use the reciprocal of the Jaccard-Index.

As the last characteristic, we introduced the C_{pred}. This characteristic describes the position of vehicles over a specific time interval for a given initial scene s_0. We compare this characteristic with the real position over the same time interval for these vehicles. We extract these positions from the reference dataset and define the ground truth characteristic as $C_{pred,\, s_0,\, gt}$. Let $C_{pred,\, s_0,\, sim}$ be an observation of this characteristic within the simulation for a sample simulation run, we compare a vehicle v_j at time t_i as follows:

$$E_{pred,\, s_0}(t_i, v_j) = ||C_{pred,\, s_0,\, gt}(t_i, v_j) - C_{pred,\, s_0,\, sim}(t_i, v_j)|| \tag{3.20}$$

Here $C_{pred,\, s_0,\, gt}(t_i, v_j)$ provides the position of the vehicle v_j at time t_i as does $C_{pred,\, s_0,\, sim}(t_i, v_j)$ for the simulation run. We use the euclidean norm as distance measure. Finally, we use the mean position error at the last time step of the observation t_n to compare the two characteristics. Given the set of all vehicles V, we define the error for a given initial scene s_0 as follows:

$$E_{pred}(s_0) = \frac{\sum_{v_i \in V} E_{pred,\, s_0}(t_n, v_i)}{|V|} \tag{3.21}$$

Therefore, we evaluate the mean error after the simulation interval. As described in Section 3.2.2, the process of simulation is repeated for a set of initial scenes: S. To compare the final characteristic (position accuracy after simulating different initial scenes), we define the final error as follows:

$$E_{pred} = \frac{\sum_{s \in S} E_{pred}(s)}{|S|} \tag{3.22}$$

With this error function, we compare a simulation run with reality and determine the degree of realism. We call this error the predictive error or the prediction ability.

4. Tool Support

In this chapter, we implement a prototype of the concept presented in Chapter 3. We describe the tool support and the implementation of the process chain. Similar to Chapter 3, this chapter is structured according to the process shown in Figure 3.1. In Section 4.1, we present the prototypically implemented process chain for generating synthetic trajectories. We describe the implementation for parameterizing the simulation environment and use the parameterization to generate trajectories. In Section 4.2, we present the implementation of the chosen realism characteristics, presented in Section 3.2, and the implementation for evaluating the realism of the generated trajectories by comparing the characteristics.

Running Example Within this chapter, we use a running example dataset to implement the process chain and argue our design decisions. We captured the dataset on 10/18/2021. The dataset covers a time of 02:42 hours and about 23.75 million rows with 4178 trajectories.

4.1 Trajectory Generation

Figure 4.1 shows an overview of the sub-process chain to generate synthetic trajectories. In the first step of Figure 4.1, we prepare the raw dataset. We use the real data as input, and a prepared dataset is output. We describe the process in Section 4.1.1. As the second step in Figure 4.1, we use the prepared dataset as input to generate parameters for the simulation as output. We use three parameterization methods to generate three sets of simulation parameters. We describe the implementation of the parameterization methods in Section 4.1.2. In step three of Figure 4.1, we use the three sets of simulation parameters as input to generate three synthetic trajectories datasets by simulation as output. We describe the implementation of this step in Section 4.1.3. Finally, in step four of Figure 4.1, we use the three trajectory datasets as input to generate scenarios in the OpenSCENARIO representation. We present the implementation in Section 4.1.4.

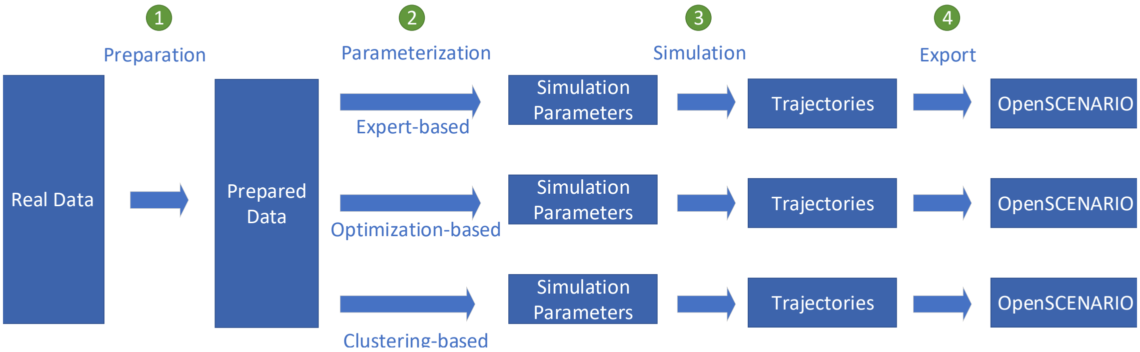


Figure 4.1: Abstract synthetic trajectory generation process.

4.1.1 Dataset Preparation

In this section, we present the implementation for the dataset preparation. First, we present the general tool support for our process chain (basis to all steps in Figure 4.1). Therefore, we select a programming language and the general tool support for the data processing. Afterward, we describe the implementation of the preparation steps using the general tool support.

For the prototypical implementation, we need a programming language that is easy to use, fast to program, provides a variety of libraries within the field of data science, is extensible, and is flexible. Especially scripting languages provide flexibility and fast implementation. Therefore, we focus the selection of the programming language on scripting languages. Many scripting languages are available that match the previously mentioned requirements. For example, Python[1], ECMAScript[2], Kotlin[3], or R[4]. Since Python provides various libraries and tool support, we implement our process chain in Python (version 3.8).

First, we set up the general tool support for our process chain. We import the existing dataset into a Python-efficient dataset format. The input to the process chain is a trajectory dataset that is spatial data. Pandas [pdt20] is a data analysis and modification library for Python. The library allows fast calculations on large datasets (millions of rows) through its native interface and also has extensions that allow spatial processing data through the extension GeoPandas [JdBF+20]. Therefore, we use it as the basis for our process chain. The input dataset is presented in an SQLite[5] database. In order to use Pandas, we convert the SQLite database into a Pandas dataframe. A dataframe resembles a temporary two-dimensional data table with columns and rows, like a table within a database, within the main memory. It is necessary to save the object to persist the results. Pandas provides different ways for serialization[6]. Python provides its own serialization library[7] called pickle, which pandas also support. Every Python environment understands this serialization. However, the size of the resulting export for the specific design of the dataframe

[1] https://www.python.org/
[2] https://www.ecma-international.org/publications-and-standards/standards/ecma-262/
[3] https://kotlinlang.org/
[4] https://www.r-project.org/
[5] SQLite is a file-based database format. https://www.sqlite.org/
[6] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
[7] https://docs.python.org/3/library/pickle.html

is huge. Using Apache Parquet [Voh16] reduces the exported size of the example dataset to 17% (5.1 GB compared to 905 MB). This export also reduces the time required for loading the dataset by half (with the same dataset the parquet export requires 11.9s and pickle: 22.3s). Therefore, we use the Apache Parquet export of Pandas dataframes for persistence with the help of the Python library fastparquet[8], which provides the implementation for the Apache Parquet export within Pandas. The setup of Python, Pandas, GeoPandas, and Apache Parquet builds the basis for our process chain.

We now perform the preparation step presented in Section 3.1.1. These are, performing coordinate transform, adding the outer positions of the bounding box, and performing a map matching. The map matching process assigns lateral and longitudinal positions within the road, lanes to vehicles, and a leading vehicle. The following sections briefly describe the implementation and the tool support of these steps.

Coordinate Transform The simulation data is described within a local coordinate system, while the reference data is located within the UTM coordinate system. In order to compare and prepare the data from both datasets with the same process chain, we transform the simulation data into the global coordinate system. We use the global coordinate system because we later perform map matching with data provided in the global coordinate system. We apply the coordinate transform by the UTM projection to the SUMO coordinate system with the GeoPandas extension.

Bounding Box The representation of vehicles within the simulation and reference dataset differs. Within the simulation, the vehicle is described by the center of the front bumper, constant width, length, height, and orientation. The real data uses a similar description but the center position as a reference. This center position is due to the detection system variable, and variable extents describe each direction's front and rear position. The heading in the real dataset is given in degrees, with 0° pointing east and moving counterclockwise. In the simulation, 0° points north and moves clockwise. We match the heading description in the simulation dataset to the real dataset by applying mathematical operations of Pandas. With these adjustments, we calculate the outer positions of the vehicles in simulation and reality by applying translation and rotations (according to the heading) using Pandas, and the NumPy [HMvdW+20] scientific computing library. We use NumPy [HMvdW+20] to speed up the calculations of mathematical operations by vectorized operations.

Map Matching We have two use-cases for the map matching procedure. The first is the precise alignment with a lane with high accuracy and low runtime requirements. The second use case has low accuracy requirements and high runtime requirements. Therefore, we implement two versions for calculating the lateral and longitudinal positions. The first is required for evaluation and calculating the correct characteristics from Section 3.2.1. For the analysis of a lane change maneuver,

[8] https://fastparquet.readthedocs.io/

for example, the lateral position must be calculated accurately. In the evaluation, it is acceptable if this procedure requires more computing time. The second use-case is required, for example, within optimization. We perform thousands of iterations during this procedure, and the runtime has to be reduced. For time efficiency, less accuracy in lateral mapping is acceptable.

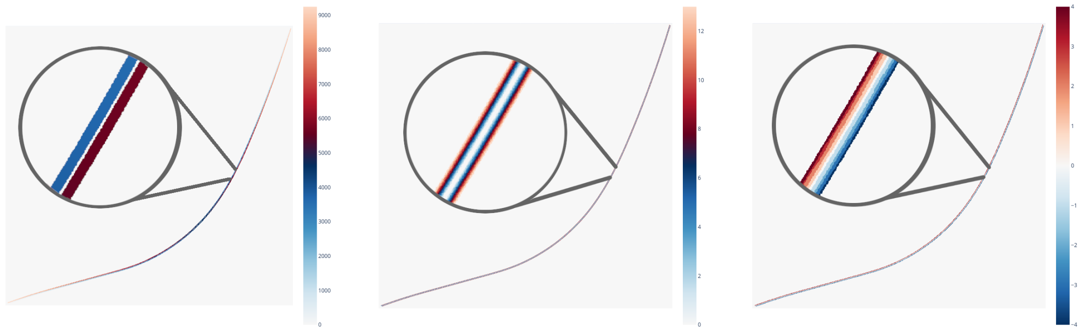
We call the first version ref-lane-method. This version is based on two reference lane markers from a high-resolution map. We use the left lane markings of the passing lane in both directions on the highway to calculate the distance of the vehicle to these lines. Based on the minimum distance, we determine the vehicle's direction of travel by `argmin(distance(veh,lane_marking_1), distance(veh,lane_marking_2))`. Thus, we implicitly calculate the lateral position since we calculate the vehicle's distance from the reference lane. Finally, to calculate the longitudinal position, we project the point of the vehicle along the reference line to determine the longitudinal position. As an initial implementation, we use GeoPandas in combination with Shapely[G+] and PyGEOS, a Python wrapper to the GEOS [GEO21] library, in order to calculate the distances and projections (from a point to a line). With the usage of PyGEOS, we accelerate the calculations. For the example dataset of three hours, the conversion of pure numerical information into spatial information (coordinate columns into GeoPandas objects) is accelerated from 86.2s to 2.7s using PyGEOS. The calculation of distances and projections is accelerated from 85.5s to 49.1s with this dataset by PyGEOS. As the final implementation, we remove the intermediate step of Pandas and Shapely and only use PyGEOS in combination with NumPy [HMvdW+20]. This implementation improves the calculation time for the entire process (calculation of the transverse and longitudinal position, including the construction of spatial objects) to 22.2 seconds (51.8 with GeoPandas). Without GeoPandas as an intermediate library, we reduce the memory requirement from 8 GB to only about 100 KB. By these calculations, we determine the longitudinal and lateral position within the lane coordinate system.
We assign the lane by dividing the lateral position into three areas: (1) passing lane, (2) main lane, and (3) side lane. The Pandas library provides the method `DataFrame.cut` to divide values into discrete bins. However, we use the NumPy function `digitize` because it is faster than the Pandas method `DataFrame.cut` (by a factor of 2 for the example dataset). We represent the lane as a discrete numerical value. To assign the leader we use the Pandas `DataFrame.group_by` operator. We group the dataset by the time step and the assigned lane, then sort the vehicles in descending order of their longitudinal position and assign a leader by using the previous vehicle. We accelerate the calculations by vectorized NumPy functions. The execution time for the example data set is 21.8 s for the assignment. Since the `DataFrame.group_by` operator is time-consuming, we accelerate the process using only on NumPy [HMvdW+20] operations. First, we sort the full dataset based on time step, lane, and longitudinal position. We then shift the columns Longitudinal Position, ID, and Speed one row up. Thus, we have the leader's position within the same row of the dataset and efficiently compute the distance to the leader. We introduced errors within the assignment due to a mismatch of time steps or lanes within the dataset by shifting. For example, one row contains a vehicle within the passing lane and the next row within the passing lane. We assigned the vehicle within the passing as the leader for the one in the main lane by shifting. This

Ground Res.	Error	Mean	Min	Max	False Leader
25 cm	Leader Gap	0.12 m	0.00 m	0.65 m	0.000%
	Lat. Position	0.08 m	0.00 m	0.42 m	
	Long. Position	0.09 m	0.00 m	0.43 m	
100 cm	Leader Gap	0.73 m	0.00 m	2452.75 m	0.098%
	Lat. Position	0.08 m	0.00 m	1.53 m	
	Long. Position	0.09 m	0.00 m	1.69 m	

Table 4.1: Error between ref-lane-method and map-based-method. The column *False Leader* describes the number of false leader assignments caused by wrong lateral or longitudinal position or lane assignment in comparison to the ref-lane-method. The error rate is based on a simulation dataset with 1319026 rows. The ground resolution of 25 cm produced no false assignment.

assignment is incorrect due to the different lanes. Therefore, we assign no leader if the time step or lane has changed. We calculate the difference between each row in the time step and the lane column and reset the leader assignment if the difference is not zero. We reduce the assignment duration from 21.8 s to 2.4 s for the example dataset with this improvement. This method is accurate but time-consuming (about 25 s for complete preparation).

We call the second version map-based-method. This method is based on two-dimensional maps with pre-computed information about the lateral and longitudinal position and lane. We use the previous method based on PyGEOS to calculate the positions and lanes for each element in the two-dimensional maps. We use a three-dimensional NumPy [HMvdW+20] array in order to represent these maps. The first two dimensions are the spatial coordinates, and the third dimension represents the map (lateral, longitudinal, and lane). A parameter of this method is the ground resolution. This parameter determines the size of each grid cell. A lower ground resolution produces more accurate results but requires more space to be stored in the main memory. Table 4.1 shows an error between the ref-lane-method and map-based-method. Each 25 cm ground resolution map requires 2.2 Gb, and the 100 cm maps require 0.14 Gb. As shown in Table 4.1, a ground resolution of 100 cm causes a higher error and leads to a few cases of incorrect leader assignments. We choose a ground resolution of 25 cm because it produces no false assignments and the memory footprint is not important. With the map-based-method a lookup of the full example dataset of three hours trajectory data takes 1.5 s in comparison to 21.8s. The total duration including leader assignment with this method takes 3.9 s. Compared to the ref-lane-method, the map-based-method takes only 15.6% of the time required. Figure 4.2 visualizes the three assignment maps. In 4.2(a) the longitudinal position is shown. The color within the highway indicates the longitudinal position (white 0 m and orange 9.2 km). Since the highway runs in two directions, the colors start with white at one end, and the opposite lane is red. 4.2(b) shows the lateral position with the same color mapping but a smaller value range. In 4.2(c) the lane assignment is shown.



(a) Visualization of the longitudinal position on the road.

(b) Visualization of the lateral position on the road.

(c) Visualization of the lane assignment map.

Figure 4.2: Visualization of the assignment maps. The visualization shows a color-coded assignment of longitudinal and lateral position as well as the lane. For example, the map shown in 4.2(a) shows a discrete position for each 1 x 1 m square. A ground resolution of 1 m was chosen to display the discrete squares and reduce the image size.

We use the ref-lane-method to prepare the real dataset. This resembles the first step within Figure 4.1. We perform the preparation of the real dataset only once. Therefore, a one-time higher processing time is acceptable. The following sections explicitly mention which method is used for the preparation.

4.1.2 Determining simulation parameters

In this section, we describe the implementation of the expert, optimization, and clustering-based parameterization. We provide the prepared real dataset (prepared by the high accuracy ref-lane-method) as input to these methods. The methods will output a SUMO configuration containing the simulation parameters.

Expert-based parameterization

We realize the expert-based method by manually setting the simulation parameters. To do this, we fill in a SUMO route configuration file using a text editor. Unlike the other methods, we use the GUI of SUMO to view the simulation and check our settings. We use this to detect anomalies and change our configuration accordingly.

Optimization-based parameterization

To implement the optimization-based method, we need to define (1) the objective function and (2) the optimization strategy. First, we implement the objective function. We implement Equation 3.6 with the error of Equation 3.17 using NumPy [HMvdW+20]. To obtain the characteristics of Equation 3.6, we prepare the results of the simulation. We use the performance-optimized map-based-method to reduce the time for each iteration within the optimization. We limit the optimized parameters to the following SUMO-CFM parameters since, they are available for each CFM: "maxSpeed", "speedFactor" (mean, deviation, min and max), "sigma", "tau" and "minGap".

Second, we implement the optimization strategy by first setting realistic bounds for each parameter shown in the appendix in Table A.1. We implemented five optimization strategies. First, we use SciPy [VGO+20] in order to optimize using the L-BFGS-B [ZBLN97] algorithm. This method performs a single-threaded optimization, which leads to an optimization time of weeks. We implemented a parallelized version using the Python optimparallel [Ger20] package. This package provides a parallelized version of L-BFGS-B using Python's multiprocessing API. Since Python's multiprocessing does not work well with shared memory and all variables are cloned between processes, the memory requirement is increased with multiple threads and limits the number of parallel optimization processes. To bypass this problem, we implement a custom optimization method that uses the Ray [MNW+18] framework for parallelization and is based on the gradient descent method. Pseudocode of this method is attached in Algorithm A.1. However, the gradient descent often ends up in local minima. Optimization methods used in literature [RRMB17] for optimizing parameters of simulations environments are, for example, SPSA [Spa92] or GA [CLHG10]. We implement both methods. The SPSA method requires only two calculations to determine the optimization step and accounts for noisy measurements. This behavior is beneficial for optimization because it reduces the number of simulations required, and the simulation is driven by random events (for example, inserted vehicles or desired speeds). We implement a SPSA-based optimization using the library Qiskit[9]. We implement the GA method based on the U-NSGA-III [SD15] algorithm using the pymoo [BD20] library. To accelerate this process, we parallelize this implementation by using Ray [MNW+18].

Clustering-based parameterization

Various libraries for clustering are available in Python, for example, sklearn [PVG+11], python-cluster[10], or pyclustering[11]. We implement the clustering-based method using the sklearn framework [PVG+11] since it provides various interchangeable cluster strategies, dimension reduction strategies, and maintained documentation with examples. First, we describe all trajectories contained within a dataset by the characteristics shown in Table A.6. We implement the characterization using statistical methods provided by Pandas [pdt20]. We normalize each characteristic using a min-max scaling also by using Pandas operations. Different methods to perform the dimension reduction are available, for example, the **P**rincipal **C**omponent **A**nalysis (PCA) or the truncated **S**ingular **V**alue **D**ecomposition (SVD). The sklearn framework [PVG+11] provides an implementation for both methods. We use the PCA [F.R01] since this algorithm works well on the given data. We employ the implementation of the sklearn framework [PVG+11] to transform the normalized characteristics into a latent space and reduce the number of dimensions. For clustering, again, a variety of methods are available. We use KMeans [M+67] clustering since the number of clusters is a configurable parameter. Within transformed space by the PCA, we cluster the data using the implementation of the sklearn framework [PVG+11] for KMeans. To determine the optimal number of clusters, we use the elbow method [Tho53]. This method uses the vertex of a performance measure to

[9] https://qiskit.org/
[10] https://github.com/exhuma/python-cluster
[11] https://pyclustering.github.io/

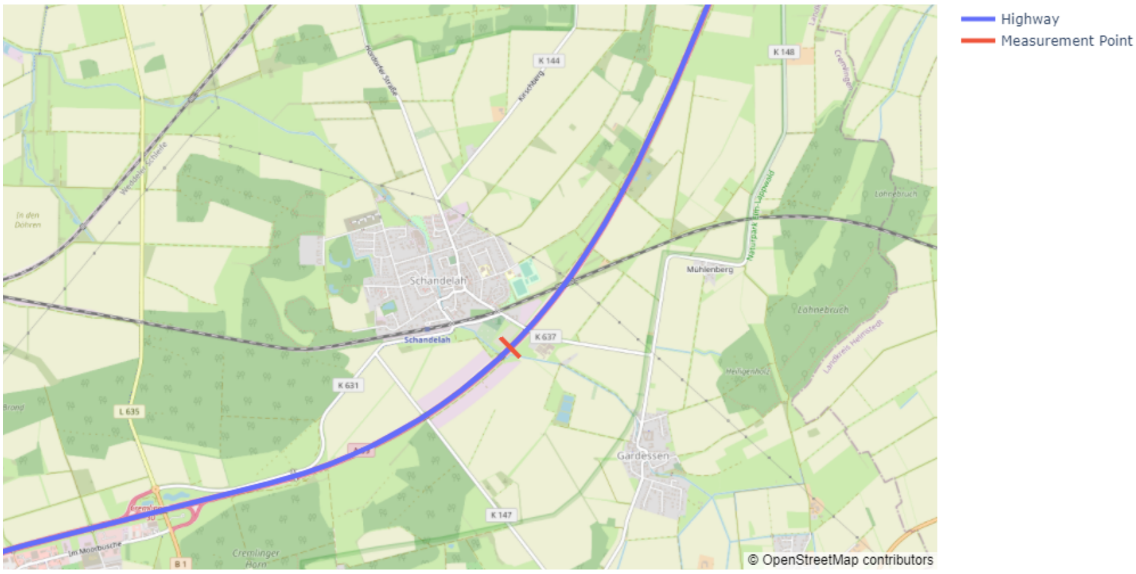


Figure 4.3: Virtual measure point used to determine traffic flow on the highway.

determine the optimal number of clusters. We use the intra-cluster similarity as the performance measure.

4.1.3 Simulation

In this section, we describe the implementation of our simulation process chain. A variety of microscopic traffic simulators is available, for example, SUMO [LWB+18], MITSIM [YK96], or AIMSUN[12]. We use SUMO since it is free, open-source, provides a comprehensive documentation[13], and Python libraries for interaction. We perform the simulations with version 1.11.0 of SUMO. First, we define the road model. The available high-resolution road model from the testbed Lower Saxony is described in OpenDRIVE. SUMO uses an own XML-based road model. We use the `netconvert` tool provided by SUMO to convert the OpenDRIVE file to the SUMO specific road model. To speed up simulation runs, we reduce the number of nodes and edges within this network by limiting the full map (approx. 120 km) to the 7.45 km of the detection system. As described in Section 3.1.3, we need to set a speed limit within the simulation. We create a script that changes the maximum speed allowed on each road to 100 kph to set the speed limit in this road model.

To set up the traffic flow as explained in Section 3.1.3, we determine the traffic flow within the real dataset using GeoPandas. Since traffic flow requires a static location, we introduce a location where the traffic is measured. Figure 4.3 shows the used virtual measurement point in the center of the testbed's detection system. We use GeoPandas in combination with PyGEOS to compute intersections between the trajectories and the virtual measurement point. Based on these results, we initialize the traffic flow within SUMO.

To analyze the simulation data, we access the results from SUMO using a file-based output. In Figure 4.4, we present the process of the collection of the simulation

[12]https://www.aimsun.com/
[13]https://sumo.dlr.de/docs/

Figure 4.4: File-based import for SUMO simulations.

data. We implement a connection to SUMO file-based output called FCDOutput. This implementation reads an FCDOutput-XML file and converts it to a Pandas dataframe, and stores it in the Apache Parquet format. We implement the converter as a Python script. However, this method incurs an overhead due to the self-describing properties of XML. This overhead leads to long loading times until the simulation results are available. Therefore, we implemented connectors to SUMO using the Python libraries TraCI and LibSUMO. LibSUMO performs overall the best regarding the time required to have the data available as Pandas dataframe (1000 s simulation, file-based: 47 s, TraCI: 184 s, LibSUMO: 19 s). Therefore, we use the implementation based on the LibSUMO to retrieve the simulation results.

For simulation, SUMO provides two methods for calculating simulation updates. The first version is called Euler and considers the speed of the vehicles to be constant during the update. This method is preset as default. The second method is called ballistic and considers the acceleration constant during the update. Treibar and Kanagaraj [TK15] show that the ballistic integration method better represents synthetic trajectories from simulation compared to real trajectories than the standard Euler version of SUMO. Therefore, we use the ballistic method as an update method.

4.1.4 OpenSCENARIO Export

Inspired by Asbach [Asb22], we first generate a Python interface for the OpenSCENARIO description with the generateDS[14] library. We write an interface that translates a Pandas dataframe into an OpenSCENARIO file to translate the simulation data. First, we create an `OpenScenario` entity with the general setup, for example, the road model, the vehicle catalog, or the storyboard. We extract a full trajectory from the dataset and initialize a `ScenarioObject` with matching dimensions. We set appropriate initial states (position, heading, and trigger action to start the trajectory at the given time it appears) for these vehicles and model their trajectory by the `Polyline` entity. Finally, we divide the entire dataset into subsets of equal length to generate scenarios. The length is parametrizable. We use the `pandas.Grouper` object to group the dataset into these intervals and export each subset as its own OpenSCENARIO file.

4.2 Dataset Comparison

Within this section, we describe the implementations for the dataset comparison as shown in Section 3.2. First, we describe the implementation and tool support for the dataset characteristics in Section 4.2.1. Within Section 4.2.2 we present the implemented comparison methods for the individual characteristics.

[14]http://www.davekuhlman.org/generateDS.html

4.2.1 Dataset Characteristics

In Section 3.2.1, we differentiate the characteristics into longitudinal, lateral, and mixed. The following paragraphs describe the implementation of these characteristics.

Longitudinal

The longitudinal characteristics reflect the driving behavior in the direction of travel. These are: C_{TTC}, C_{DRAC} and C_{TH}. A leading vehicle must be determined to calculate these characteristics. By the preparation step introduced in Section 4.1.1, we assigned a leader in the dataset. However, a leading vehicle is assigned, even if it is many kilometers ahead. When a TTC is calculated with vehicles far apart, the TTC value is technically correct. Nevertheless, it is unlikely that a collision will occur because there is enough time for the drivers to react. Therefore, we restrict the calculation of longitudinal characteristics to situations where the vehicle in front is below a certain threshold. According to Higgs and Abbas [HA14], we use a threshold of 120 m to study the following behavior. We implement the metrics using Pandas dataframe operations. Listing 4.1 shows the exemplary implementation for the TTC. This function assumes that the provided dataframe contains only vehicles with a leading vehicle below the 120 m threshold. The column `leader_gap` denotes the gap to the leading vehicle. The column `ego_speed` refers to the speed of the ego vehicle and `leader_speed` refers to the speed of the leading vehicle. Listing 4.1 line 4 sets the TTC to undefined if the leading vehicle drives faster than the ego vehicle. To compute the characteristics described in Section 3.2.1 (C_{TTC}, C_{DRAC}, and C_{TH}), we determine the distribution. We use NumPy [HMvdW+20] in order to calculate the distributions.

Listing 4.1: Function that calculates the TTC based on a Pandas dataframe

```
1 def calc_ttc(df: pandas.DataFrame) -> pandas.DataFrame:
2     df['TTC'] = df['leader_gap'] / \
3         (df['ego_speed'] - df['leader_speed'])
4     df.loc[df['TTC'] < 0, 'TTC'] = numpy.NaN
5     return df
```

Lateral

The lateral characteristics denote the interaction between lanes orthogonal to the direction of travel. We defined the following lateral characteristics: C_{LD}, $C_{|LC|}$, C_{meanLC}, $C_{|LCvariations|}$ and $C_{|LCclustershape|}$ in Section 3.2.1. We implement the lane distribution characteristic (C_{LD}) by counting the occurrences of each lane within the dataset divided by the dataset size by Pandas operations. The other characteristics require the CCLCI. We use SciPy's signal package[15] in order to implement the cross-correlation for the CCLCI. To detect lane changes by cross-correlation, we first establish a ground truth for lane changes by computing the difference between

[15] https://docs.scipy.org/doc/scipy/reference/signal.html

lane assignments for each given trajectory. We define a reference trajectory used for cross-correlation by randomly selecting one lane change. We use only one sample to determine if this detector has good generalization ability and works with a minimum number of observations. We apply the `scipy.signal.correlate` function to a given trajectory and the reference trajectory. In order to detect the peaks within the correlated signals we use the function `scipy.signal.find_peaks`. We use two parameters of this method: the threshold, which indicates the minimum height of a peak to be classified as a peak, and the prominence, which indicates how much a peak stands out from the surrounding signal. We found experimentally that a threshold of 0.5 and prominence of 0.5 work the best with our data with the `scipy.signal.find_peaks` method. The experimental results are shown in the appendix in Table A.7.

To compute characteristics based on lane changes, we first apply the CCLCI to the entire dataset and store the locations where it was triggered. We calculate the characteristic $C_{|LC|}$ by the frequency of the trigger from the CCLCI using a Pandas operation. To calculate C_{meanLC}, we choose a window (experimentally, 5 s has been shown to work best) around the lane change event and calculate the mean trajectory using Pandas. For determining $C_{|LCvariations|}$ and $C_{|LCclusterhape|}$, we cluster the lane changes. Similar to Section 4.1.2, we first transform the trajectory (as a vector of lateral position in relation to time) using a PCA into the latent space and reduce the number of dimensions. Within this space, we cluster the trajectories by KMeans using the sklearn framework [PVG+11]. We calculate the characteristic $C_{|LCvariations|}$ using the elbow method [Tho53]. Finally, to determine the characteristic $C_{|LCclusterhape|}$, we select all trajectories associated with the same cluster and compute a mean shape using Pandas.

Mixed

In this paragraph, we present the implementation for the characteristics C_{scenes} and C_{pred}. First, we implement the C_{scenes} characteristic. To extract all the scenes in a dataset, we group the data by the time step and the direction in which the vehicles are traveling. This operation results in one scene per time step and direction for the entire spatial observation area. As described in Section 3.2.1 we inspect the local area around each vehicle. Therefore, we divide the scene containing all vehicles into many small scenes containing only a subset of these vehicles. We choose a range of 120 m as the local environment around the vehicles, with the same considerations as for the TTC characteristic. In order to improve performance we implement a parallelization of this process using Ray [MNW+18]. A final scene is described as a list of tuples containing the vehicles. Listing 4.2 shows a sample scene. A tuple

Listing 4.2: Scene description in Python.

```
>>> example_scene = [(30, 1, 5, 25), (120, 2, 4, 30)]
```

is formed from four values. The first value of the tuple describes the longitudinal position within the local environment (from 0 m to 240 m, with the local environment set to 120 m). The second value of the tuple indicates the lane, the third the class, and the last the speed (given in m/s). We round the position and speed to allow for minor variations described in Section 3.2.1. We choose the longitudinal position rounded to 5 m and a velocity rounded to 1 m/s. A problem with this representation is the slow comparison ability. To determine if a given scene is in the real dataset, we need to compare each scene and iterate over each tuple within that scene. This leads to long calculation times and is not efficient. Therefore, each individual tuple is hashed using Python's built-in hash method, shown in Listing 4.3. To detect if

Listing 4.3: Hashed scene participants in Python.

```
>>> hashed_example_scene = [hash(vehicle) for vehicle in
    example_scene]
>>> hashed_example_scene
[6785173475036912956, 5185509306378155339]
```

a scene is contained in another dataset, we still need to iterate over all contained scenes and vehicle representations inside the scenes. We compute a total scene hash as the sum of all individual hashes, shown in Listing 4.4. We use the sum of the

Listing 4.4: Combined hased scene description in Python.

```
>>> scene_hash = sum(hashed_example_scene)
>>> scene_hash
11970682781415068295
```

hashes because a different order within the vehicles would still result in the same final hash value for the scene. This consideration is necessary as the vehicles are unsorted within the time steps and direction group. We store this hash in a Python `set` data structure and efficiently search for hashes in a vast dataset.

As the last characteristic, we implement the prediction ability C_{pred}. First, we choose 1000 random timestamps evenly from the set of unique timestamps. We collect the scenes from the set of timestamps by extracting all vehicles from the dataset that match the timestamp. With these scenes, we initialize SUMO with the initial settings of vehicles by LibSUMO. We choose the vehicle's parameters (for example, velocity or acceleration) as in the initial scene. In order to simulate, we assign the inserted vehicle to one of the vehicle types (SUMO-`<vType/>`) defined by the configuration. The expert- and optimization-based methods generate one vehicle type for each real vehicle class (for example, passenger car or truck). However, the cluster-based method generates many vehicle types for one real class. We chose the vehicle type by a weighted sampling from the available vehicle types for a single class. We derive the weights from the probability of occurrence of the respective vehicle type, normalized to the overall probability of the vehicle class. Using LibSUMO, we simulate and save the results as a Pandas dataframe. We parallelize the process of running the 1000 simulations with Ray [MNW+18].

4.2.2 Comparison - Realism Metric

The longitudinal characteristics C_{TTC}, C_{DRAC}, C_{TH} and the lateral characteristic C_{LD} are distributions. We implement the comparison according to Equation 3.17 using NumPy [HMvdW+20]. Since $C_{|LC|}$ and $C_{|LCvariations|}$ are single numbers, the implementation is just a subtraction. To implement the comparison of C_{meanLC} and $C_{LCclusterhape}$ according to Equation 3.18, we use NumPy [HMvdW+20].

In Section 4.2.1 we store the scenes contained in a dataset inside a Python `set`. This representation makes it easy to calculate the Jaccard-Index using Python operators, shown in Listing 4.5. The operator `&` forms the intersection and the operator `|` forms the union of two set objects. Given a set of real scenes `set1` and a set of simulation scenes `set2` (hashes of these scenes), we efficiently compare millions of scenes.

Listing 4.5: Implementation of the Jaccard-Index using Python-set's.

```python
def jaccard(set1: set, set2: set) -> float:
    return len(set1 & set2) / len(set1 | set2)
```

To compare the prediction ability C_{pred} we introduced Equation 3.20. We implement this equation using NumPy [HMvdW+20]. To finally compute the overall error simulation runs according to Equation 3.22 we use Pandas.

5. Evaluation

Our goal is to investigate which methods are well suited for parameterizing a simulation environment to generate realistic scenarios for testing ADAS/AD. Hence, we implement a process chain to generate synthetic datasets based on real data and subsequently assess them. We use a real trajectory data set from the Lower Saxony testbed to determine simulation parameters. We use an expert-based, optimization-based, and clustering-based method to determine different parameter sets as parameterization methods. With these parameter sets, we generate synthetic trajectories using the simulation environment SUMO and export these as OpenSCENARIO files. Finally, to compare these methods, we introduced several characteristics that examine different aspects of the trajectories and introduced comparison methods for these characteristics. We present in this chapter the experiments, the system under study, and the experimental results. Furthermore, we discuss our results and their validity.

5.1 Research Questions

In this study, we focus our research on the realism of the dynamic objects within the generated scenarios (L4 according to Bagschik et al. [BMKM18]). We defined a concept in Chapter 3 to generate synthetic trajectories with a simulation environment and evaluate these in terms of realism. This concept is based on a parameterization method for the simulation environment. We identify the chosen method and the input parameters of the simulation environment as a possible influence on the realism of the trajectories. To evaluate realism, we divide the notion of realism into several aspects. These aspects are represented by characteristics measured within the synthetic trajectories. We want to understand how these characteristics relate in the simulation compared to reality. We identified three different methods for parameterization (expert, optimization, and clustering-based). As a result, we assume that the choice of the parameterization method influences realism. Furthermore, we expect the selection of realism characteristics will influence the evaluation of overall realism. We define the following research questions to guide our research:

RQ 1: What influence does parameterization have on the realism of synthetically generated trajectories?

RQ 2: How realistic are the synthetically generated trajectories by the simulation?

RQ 3: How can a simulation environment be parameterized to create realistic scenarios?

RQ 4: What influence do the evaluation characteristics have on the assessment of the realism of the trajectories?

The objective of **RQ 1** is to determine whether parameterization is necessary to create realistic scenarios in terms of dynamic objects. We investigate whether the parameterization has an influence, how strong the influence is, and which parameters influence the realism of the resulting trajectories. With **RQ 2**, we examine the realism of the resulting trajectories in the different aspects introduced by the characteristics. We identify in which aspects the synthetic trajectories are similar to reality and how they differ. Our goal is to gain insights into the situations in which simulation environments are useful for creating realistic scenarios. **RQ 3** aims to answer which method generates the most realistic trajectories. Our goal is to determine whether a single method generates overall or whether individual methods generate realistic results in specific aspects. With **RQ 4** we examine if all characteristics are suitable to evaluate the realism of the synthetic trajectories. We also want to investigate whether a single characteristic is suitable for evaluating realism as a whole.

5.2 Experiment Design

To answer the research questions, we conduct two experiments. One experiment is designed to answer **RQ 1**. The other one shall answer **RQ 2**, **RQ 3**, and **RQ 4**.

5.2.1 Experiment 1

Inspired by Henclewood et al. [HSR+17] we set up an experiment to determine the influence. The abstract experiment design is shown in Figure 5.1. In the first step of Figure 5.1 we generate a set of parameters. Within each parameter generation, we change only one parameter within selected boundaries. As the second step in Figure 5.1, we use the previously generated parameter set to generate a synthetic trajectory dataset. After simulation, in the third step of Figure 5.1, we determine the error within the characteristics under study. We use the error to evaluate the effect on the realism of the input parameters. In the fourth step of Figure 5.1, we store the error. We repeat this process with all selected parameters until a maximum number of iterations is reached. After all simulations are complete, we determine the influence for a parameter p, a characteristic c, the according error function E, and the simulation results R as follows:

$$I_{p,c} = max(E_{p,c}(R)) - min(E_{p,c}(R)) \tag{5.1}$$

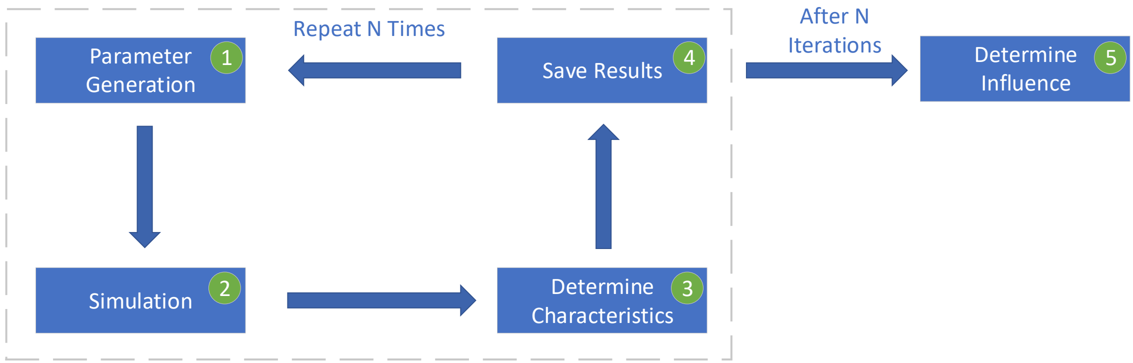


Figure 5.1: Experimental design for the first experiment in order to determine the influence of the parameterization and individual parameters.

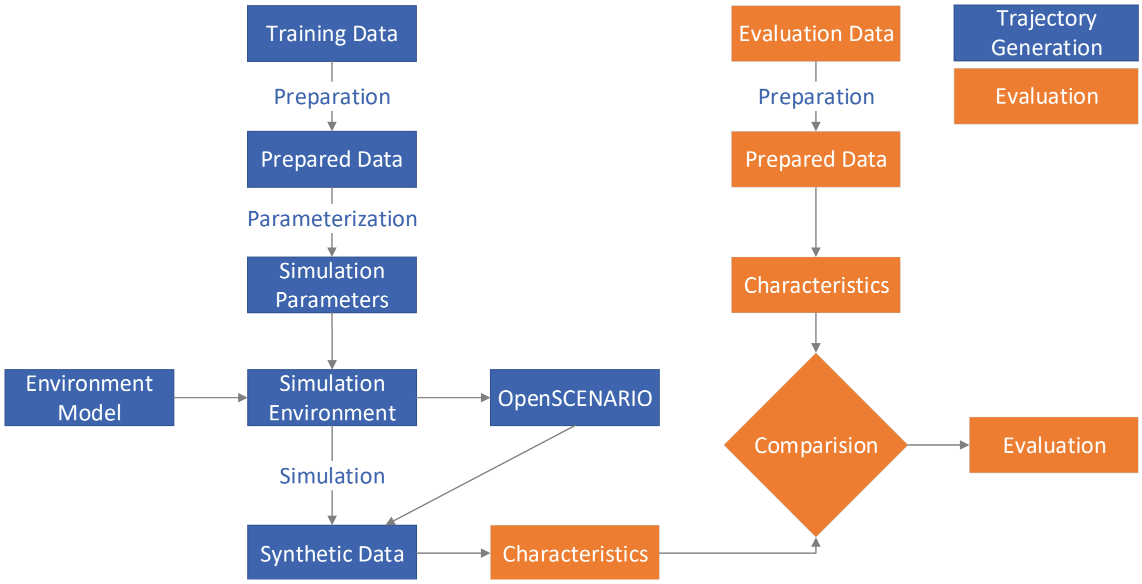


Figure 5.2: Experimental design for the second experiment to determine the realism of the synthetically generated trajectories.

Using this equation, we calculate an influence for each parameter and characteristic for the simulations performed where this parameter changed in the fifth step of Figure 5.1. We use only the results where the parameter is changed to study the isolated influence of this parameter.

5.2.2 Experiment 2

As second experiment we set up the process shown in Figure 5.2. In contrast to the process chain from Figure 3.1 we use different datasets for parameterization (training) and evaluation. In this way, we ensure an independent evaluation and assess the performance of the methods. We split the full dataset into a training dataset and an evaluation dataset. We use the first 70% of the whole dataset for training and the remaining 30% for evaluation. One possibility for dividing the dataset into two parts is using many small samples, for example, dividing each hour into ten-minute datasets. This distribution is beneficial since environmental factors, for example, changed weather or temperature, are contained in training and evaluation.

However, trajectories that are supposed to be connected will split among several datasets. Therefore, a trajectory is cut into multiple parts. With more truncated trajectories, the reliability of the statements about a driver decreases because less information is available. Based on the training dataset, we determine five SUMO configurations with different parameterization methods. With these SUMO configurations, we generate five sets of synthetic trajectory data with the same length as the evaluation dataset. We chose this length to achieve optimal comparability with the evaluation dataset. Within the synthetic dataset, we calculate characteristics. We calculate the same characteristics for the real dataset. Using the calculated values of both datasets, we compute an error: E. For example, we compare the TTC characteristic C_{TTC} for a measurement within the real dataset $C_{TTC,Real}$ and a simulation measurement $C_{TTC,Sim}$ according to Equation 3.17. To answer **RQ 2**, we examine each characteristic and evaluate the realism of the resulting trajectories within this aspect. We examine all longitudinal, lateral, and mixed characteristics regarding their properties and error. For example, we inspect the error and anomalies within the TTC distribution in order to determine in which aspects the TTC distribution is realistic. With **RQ 3**, we want to compare these methods against each other. Therefore, we introduce an overall characteristic summarizing each aspect and compare this characteristic between the methods. For **RQ 4**, we examine the detailed results of the individual characteristics. We inspect, for example, whether one method generates the most realistic results in all characteristics. Thus, we inspect whether realism is dependent on individual characteristics or whether the characteristics agree with their assessment of realism.

Experiment 2.1 As part of the second experiment, we set up an additional experiment to investigate the characteristic of scene consistency in more detail. Therefore, we reuse the previously determined simulation parameters by each parameterization method. With these parameters, we generate an extended synthetic dataset. We use this synthetic and the evaluation dataset to inspect the scene consistency characteristic over a more extended period. We investigate how many scenes the simulation environment can replicate if the simulation time is extended.

5.3 Subject System

In this section, we describe the system under test. We structure this section according to the two experimental setups.

5.3.1 Experiment 1

Within the first experiment, we identified the following influences: input parameters, simulation setup, number of iterations, and evaluation characteristics. We select the parameters and their according boundaries shown in Table 5.1. We use this selection because these parameters are available for all CFMs in SUMO. The bounds are selected based on literature values and expert opinion. As a base simulation setup, we use the default configuration from SUMO. We simulate 15 min with each configuration. We choose this length as a compromise between the reliability of the measurements and computation time. If the simulation time is chosen too short,

Parameter Name	Minimum	Maximum
tau	0.1	5
minGap	0.1	10
accel	0.1	10
length	1	15
maxSpeed	15	125
sigma	0	1
height	1	4.5
width	0.7	2.8
speedFactorMue	0.5	4.5
speedFactorSigma	0.0	2.0
vehsPerHour	100	5000
actionStepLength	0.05	1

Table 5.1: Used parameters and their boundaries for the first experiment in order to determine their influence.

random events influence the measured characteristics. In contrast, the computation time increases. We set the step length to 0.05 s (20 Hz) to match the frequency of the real dataset to ensure comparability. We measure the influence of the characteristics C_{TTC}, C_{DRAC}, C_{TH}, C_{LD}, and $C_{|LC|}$. To reduce calculation effort, we focus on the five selected characteristics. In comparison to the remaining characteristics, these can be computed within seconds. We generate 100 configurations for each input parameter. We select randomly by a uniform distribution within the bounds. Additionally, we chose one configuration on each side of the boundary. Thus, with 12 parameters and $100+2$ configurations per parameter, we perform 1224 iterations.

5.3.2 Experiment 2

According to Figure 5.2, we identify the real dataset (training and evaluation), the simulation environment, the parameterization methods, and the evaluation characteristics as influences of the second experiment.

Real Dataset As a real dataset, we use a trajectory dataset from the testbed Lower Saxony. We collected the data on Monday, February 21, 2022 with a time span from 07 am to 12 pm on the testbed Lower Saxony. The full dataset consists of approx. 32.7 million rows. A total of 7335 vehicles (3584 southbound and 3751 northbound) traveled on the testbed during the observation period. Thus, the average traffic flow is approx. 716 veh/h southbound and approx. 750 veh/h northbound. We split the five-hour data set into a training dataset of 3:30 hours and an evaluation dataset of 1:30 hours.

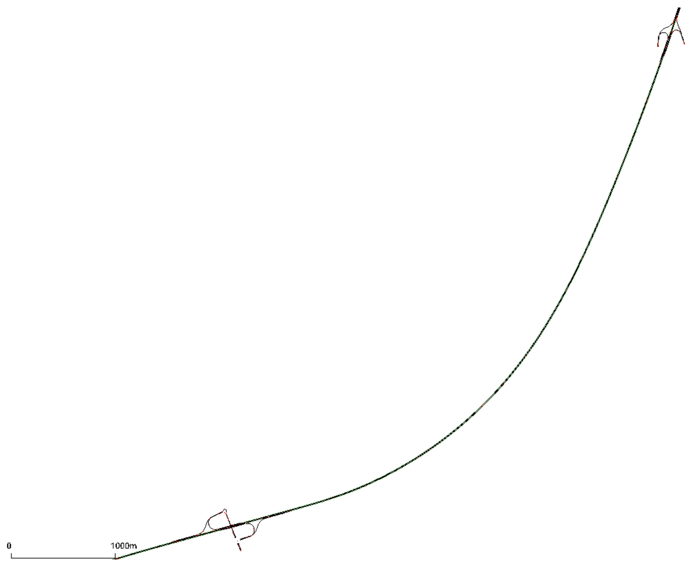


Figure 5.3: Screenshot of simulated track within SUMO.

Simulation Environment We use SUMO as the simulation environment to generate the synthetic trajectories. As described in Section 4.1.3, we use a high-resolution OpenDRIVE map for the simulation. The map originates from the project testbed Lower Saxony[1]. Figure 5.3 shows a snapshot of the converted OpenDRIVE map within the netedit tool from SUMO. We use the speed restriction of 100 kph as described in Section 3.1.3 on the highway. As for traffic flow, we set up the real measured values within the training dataset for training and the evaluation of traffic flow within the evaluation. Within the second experiment, we simulate 01:30 hours as it matches the length of the evaluation dataset. For experiment 2.1, we use a length of 10:00 hours since this is the technical limit for the prototypical process chain.

Parameterization Methods In Section 3.1.2 we presented three different methods for parameterization, namely: expert, optimization, and clustering-based. As part of our implementation in Section 4.1.2, we have found two methods of optimization commonly used in the literature, where it is not clear which works better. These methods are: using a **G**enetic **A**lgorithm (GA) and using the SPSA algorithm. Since both seem appropriate for this approach, we use both. We also add a method that uses only the default parameters of SUMO. This eventually leads to the following methods: SUMO default parameters, expert-based, optimization-based by SPSA, which we call SPSA from now on, optimization-based by GA, which we call GA from now on, and clustering-based.
Since the expert-based method does not have an implementation, we briefly describe our derivation for the expert-based method. First, we determine the CFM since the other parameters depend on this choice. We conduct a literature search to determine the best fitted CFM. Salles et al. [SKR20] compare the **I**ntelligent **D**river **M**odel (IDM), Krauss, and the **E**xtended **I**ntelligent **D**river **M**odel (EIDM) with each other, their results show that the EIDM best represents the real data.

[1]www.testfeld-niedersachsen.de

Therefore, we use the EIDM as CFM.
Literature provides different natural driving studies and research about human behavior [EW83, WBM03, PD05]. Within these analyses, the human behavior regarding time headway (CFM parameter tau) is researched. Parameters from humans differ from those determined to be optimal for CFM within simulation [PBF+17, KBG+16, VHSK+15]. Therefore, we use parameters determined for simulation environments. Salles et al. [SKR20] provide a parameter set for the EIDM. We use these parameters by reference because they explicitly provide parameters for the EIDM within SUMO.

Evaluation Characteristics To compare the synthetic and real datasets, we examine all ten introduced characteristics in Section 3.2.1. These are, C_{TTC}, C_{DRAC}, C_{TGAP}, C_{LD}, $C_{|LC|}$, C_{meanLC}, $C_{|LCvariations|}$, $C_{LCclustershape}$, C_{Scenes}, and C_{Pred}. To compare the methods against each other, we introduce a characteristic that combines all ten characteristics. We have no insight into which characteristics are well suited to determine realistic results. Therefore, we weigh them all equally. We introduce a total error based on the set of all characteristics C and the trajectories generated by a method $traj_m$ as follows:

$$E_{traj_m} = \sum_{c \in C} E_c(traj_m) \tag{5.2}$$

One challenge with this representation is that the range of values of some errors is larger or smaller than others. This would lead to undesirable weighting. Therefore, we have to normalize each error. One way is that we normalize the errors within their mathematical bounds. However, this would also introduce undesirable weighting since, for example, it is unlikely that the maximum error will be reached by Equation 3.17, but it is more likely that the limits of Equation 3.19 will be reached. Thus, we introduce normalization based on the default parameters. We finally evaluate the methods using the following equation:

$$E_{traj_m} = \frac{\sum_{c \in C} \frac{E_c(traj_m)}{E_c(traj_{default})}}{|C|} \tag{5.3}$$

We divide the error by the cardinality because the range of values of the error becomes independent of the number of characteristics.
Within experiment 2.1, we examine the scene consistency characteristic C_{scene} in more detail. Due to hardware limitations, we only simulate 10:00 hours. However, we want to know how this characteristic evolves over longer periods. Therefore, we fit the results to a function f. Motivated by the known absolute limit of $max(f) = 1$ (the simualtion reproduces all real scenes) and an assumed converging behavior, we use the following function with the constants a and b to approximate the data.

$$f(t) = \sqrt{t} \cdot a \cdot (1 - e^{-t \cdot b}) \tag{5.4}$$

We determine the constants a and b for each method by regression.

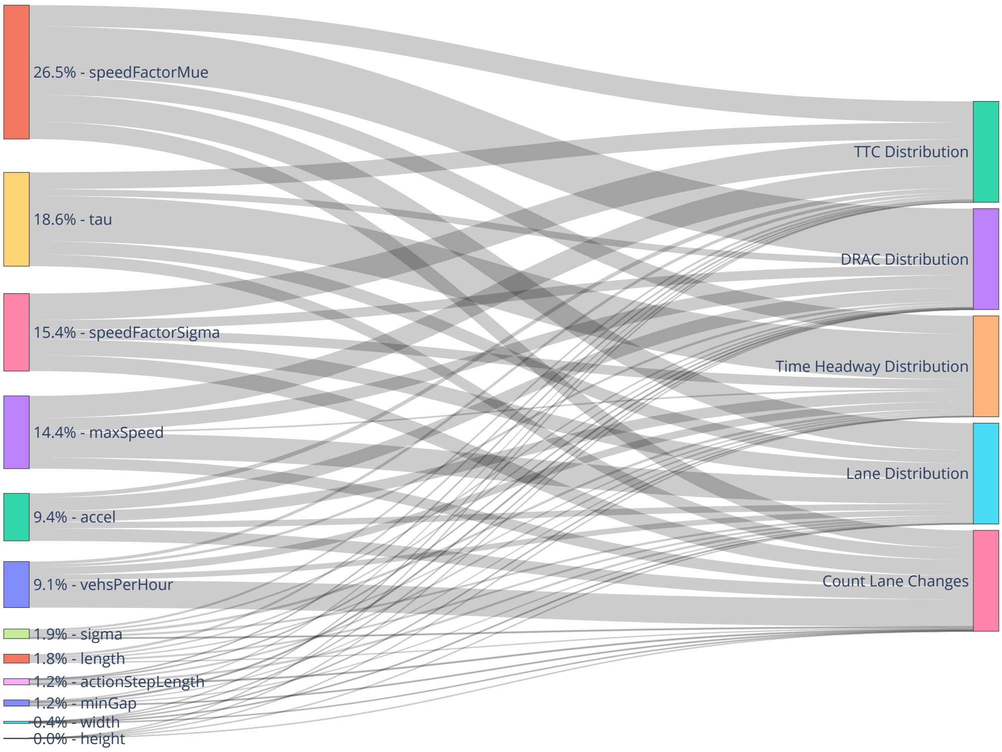


Figure 5.4: Overall parameter influence analysis.

5.4 Experiment Results

In this section, we present our experimental results. First, we present the results of experiment 1, in which we examine the effects of individual parameters on the realism of the chosen characteristics. In the second part, we present the results of the second experiment, in which we investigate the realism of the trajectories. Within this section, the observed real data is called ground truth.

5.4.1 Experiment 1

We set up the experiment according to Section 5.2.1. We use Equation 5.1 to determine the influence of each input parameterization on each characteristic. We use a Sankey diagram to illustrate the influence. The influence of each input parameter on the output characteristics is illustrated by the size of the links between them. The results are shown in Figure 5.4. The left side displays each input parameter of Table 5.1. On the right side, the normalized error (according to the error functions shown in Section 3.2.2) on the output characteristics is displayed. We sort the parameters on the left according to their combined influence on all characteristics. We compute an aggregated influence of a parameter on all characteristics, expressed as a percentage on the left-hand side. Considering these results, we identify the speedFactorMue parameter as the most influential among the selected parameters with 26.5 % and the height parameter as the least influential with approx. 0.0 %. We observe a significant gap in the range of influence between the parameters vehsPerHour and sigma. The six parameters below vehsPerHour have in sum (6.5 %) less influence than vehsPerHour with 9.1 %. We conclude that the influence on

Method	Execution Duration
Default	00:00 h
Expert	03:00 h
Optimization (SPSA)	13:39 h
Optimization (GA)	14:02 h
Clustering	00:12 h

Table 5.2: Execution duration for each individual method.

the output characteristics of individual parameters varies. For example, the speedFactorMue is most influential for the DRAC distribution. Within the distribution of the TH the parameter tau is most influential. Individual visualizations for the TTC distribution are provided in the appendix in Figure A.1, for the DRAC in Figure A.2, for the TH in Figure A.3, for the lane distribution in Figure A.4, and for the number of lane changes in Figure A.5.

5.4.2 Experiment 2

In this section, we first describe the execution times and findings within the experiment execution. Then the results for each characteristic, Finally, the combined results that summarize all characteristics.

Experiment Execution

We execute the experiment shown in Section 5.2.2. We use the five parameterization methods (default, expert, SPSA, GA, and clustering-based) to generate five different configurations for SUMO. The final parameterization determined by the SPSA method (optimization) are shown in the appendix in Table A.2. For the method using a GA (optimization), the parameters are presented in the appendix in Table A.3. The cluster-based parameterization is also given in the appendix in Table A.4. The full set of parameters for the expert-based method is shown in the appendix in Table A.5. In Table 5.2 the execution times for each individual method are shown. We use a machine with 72 logical cores, 196 Gb RAM, and the operating system Ubuntu 20.04 to determine the results. During the execution of the clustering-based method, we find that the traffic density is about twice as high. The higher traffic density is due to congested traffic. We discover that the congestion depends on the simulation parameter sigma. Sigma indicates the driver's imperfection in terms of how strict he acts according to the CFM[2] (value between 0 and 1). Since sigma cannot be observed directly in real traffic, we do not implement this parameter within the clustering-based method. We discovered that values below about 0.6 (more accurate driving) lead to congestion. Therefore, we set the sigma value to 0.75 to generate trajectories using the clustering-based method.

[2]https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#car-following_model_parameters

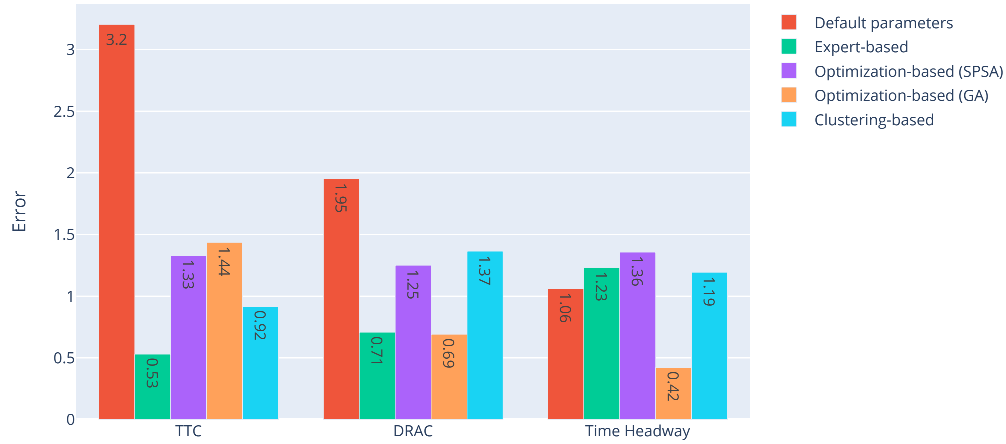


Figure 5.5: Results of the longitudinal characteristics. Lower values are more realistic.

Experiment Results

We divide this paragraph into the subcategories of our notion of realism. These are longitudinal, lateral, and mixed characteristics. Finally, we summarize all characteristics to a combined result for each method.

Longitudinal Characteristics We inspect results of the longitudinal characteristics: TTC distribution (C_{TTC}), DRAC distribution (C_{DRAC}), and TH distribution (C_{TH}). We use Equation 3.17 to calculate the error for each parameterization method. The results are shown in Figure 5.5. The abscissa shows the different longitudinal characteristics, while the ordinate shows the error according to Equation 3.17. The lower the values on the ordinate, the more realistic the method is concerning this characteristic compared to the ground truth evaluation dataset. Figure A.6 in the appendix presents the underlying data as histograms for each longitudinal characteristic and the result produced by the method. We observe within the TTC distribution that the expert-based method yields the lowest error of 0.53, while the default configuration results in the highest error of 3.2 when using Equation 3.17. For the DRAC distribution, we find similar rankings, but the GA method generates slightly more realistic trajectories than the expert-based method. In comparison to the TTC distribution, we observe a lower discrepancy between default parameters and the other methods. The results of the TH distribution differ from those of the prior distributions, as the GA method results in the lowest error of 0.42, and the default parameters result in the second-lowest error of 1.06. The SPSA method leads to the highest error of 1.36. We conclude that all methods except the GA method (which results in significantly lower error) lead to comparable results within the TH distribution. We observe within all longitudinal characteristics the most variation in error between the highest and lowest in the TTC distribution. The expert and the GA method both yield the lowest error in two of the three characteristics, with a small lead of the GA method within the DRAC distribution.

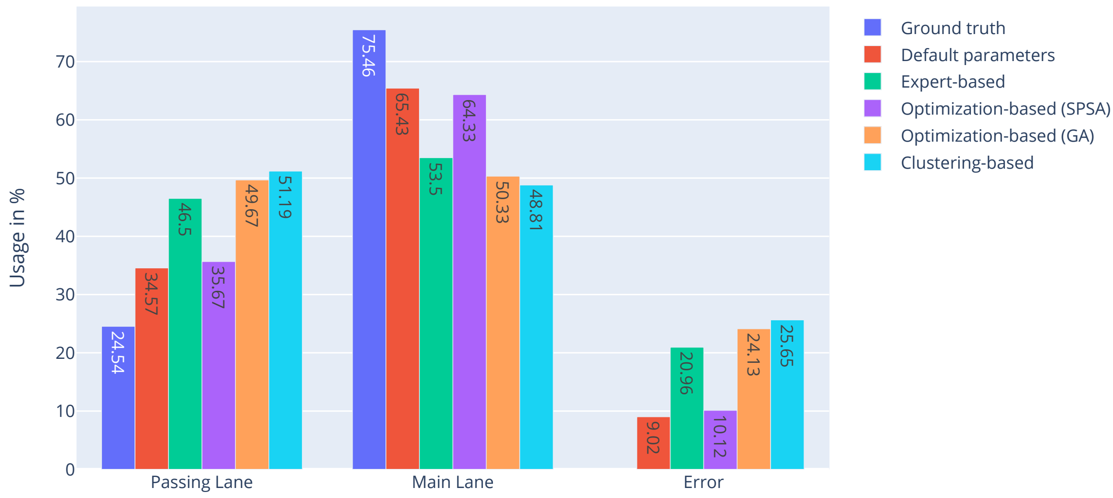


Figure 5.6: Distribution of lanes on the main and passing lanes in both directions of the road. The error is shown on the right side (lower values are more realistic).

Lateral Characteristics The lateral characteristics are: lane distribution (C_{LD}), number of lane changes ($C_{|LC|}$), mean lane change maneuver (C_{meanLC}), number of variations for the lane change maneuver ($C_{|LCvariations|}$), and the variation of the lane change maneuver itself ($C_{LCclusterhape}$). We investigate the characteristics separately. In Figure 5.6 the distribution of lanes C_{LD} for each parameterization method is shown. The abscissa shows the parameterization method, the passing lane on the left and the main lane in the middle, and the error compared to reality on the right. The ordinate represents the distribution of the lane (main or passing lane) and the total error in percent. We divide the usage and the error by each parameterization method. We detect the lowest error for the default parameters, closely followed by the SPSA method. The clustering-based method results in the highest error. We observe a comparable error within the expert, SPSA, and clustering-based method.

As second lateral characteristic, we examine the results of the $C_{|LC|}$ characteristic. In Figure 5.7 the lane changes per minute and track kilometer are shown. The abscissa shows the parameterization method, the passing lane on the left, the main lane in the middle, and the error to the right. The ordinate displays the number of lane changes performed per minute and track kilometer. We observe the lowest error using the SPSA method. Using the default parameters of SUMO we obtain the highest error, closely followed by the expert-based method. We discover an equal number of lane changes to the left ($C_{|LC|,gt,left} = 4.58$) and right ($C_{|LC|,gt,right} = 4.59$) within the ground truth data. We perceive that a left lane change is performed more frequently than a right lane change within all simulation runs. For example, with the SPSA method, the left lane is changed 3.36 times, while the right lane is changed 2.15 times per minute and track kilometer.

Within this paragraph, we present the results concerning the C_{meanLC} characteristic. With this characteristic, we inspect the mean lane change. Figure 5.8 shows the results of the mean lane change. The abscissa displays the time in seconds before

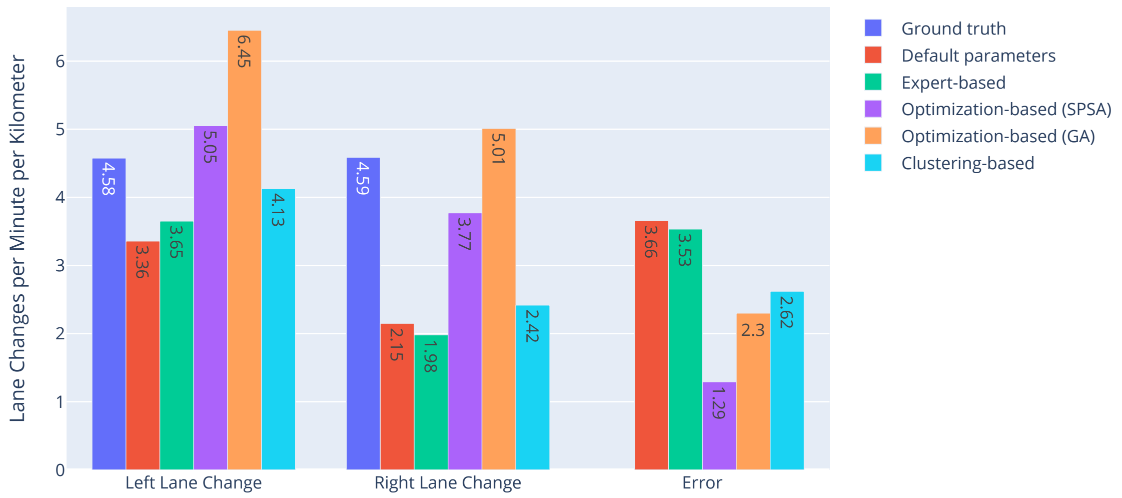


Figure 5.7: Lane changes per minute and track kilometer in reality and simulation. The error is shown on the right side (lower values are more realistic).

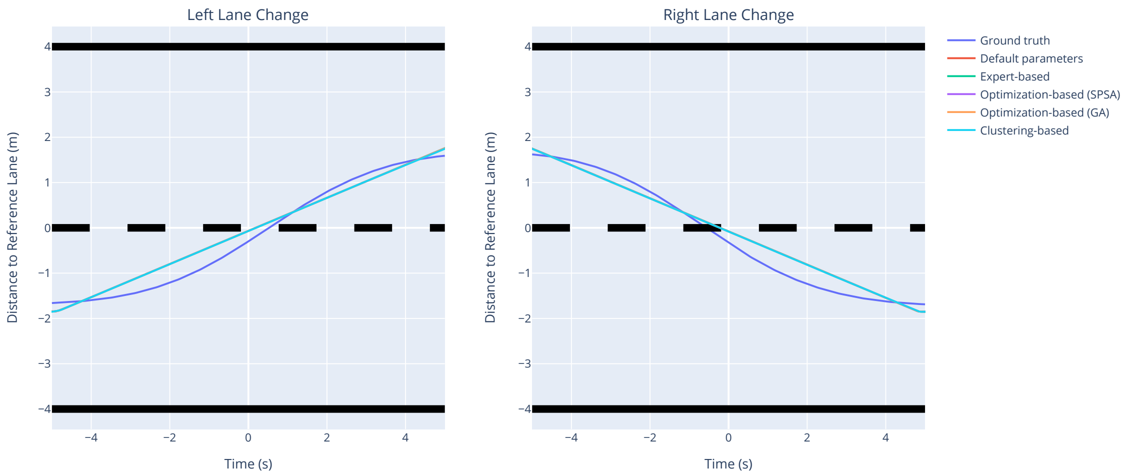


Figure 5.8: Mean lane change behavior. Note: All lane changes within the simulation are on the same line.

Dataset / Method	Number of clusters
Ground truth	5
Default parameters	1
Expert-based	1
Optimization-based SPSA	1
Optimization-based GA	1
Clustering-based	1

Table 5.3: Number of variations of the lane change maneuver detected in reality and by the parameterization method.

and after the lane change event. On the ordinate, the lateral position in the road coordinate system is shown. The left figure shows the left lane change, and the right figure the right lane change. The black bars at the top and bottom indicate the side lane markings. The middle road marking is shown as a dashed line. The upper part of this figure shows the passing lane, and the lower part shows the main lane. The time window ranges from -5 to 5 s before and after the lane change. We chose this interval because the lateral position at -5 and 5 s are the extreme values. The mean lane change of the ground truth dataset is shown in dark blue. We observe a non-linear progression of this lane change. While in the simulation, we detect the same progression for each parameterization which is linear. Therefore, only one lane change for the clustering-based method is visible since the stack over each other. Thus, we calculate the same error: $E \approx 4.1$ for each parameterization method according to Equation 3.18.

As final lateral characteristics, we analyze $C_{|LCvariations|}$ and $C_{LCclusterhape}$. First, we analyze the number of detected clusters by the parameterization method using the elbow method as described in Section 4.2.1. The results are shown in Table 5.3. The simulation performs only one variant of the lane change maneuver. In reality, there are five different variants. Thus, we calculate the same error: $E = 4.0$ for all parameterization methods. Since the simulation always performs the same variant of the lane change maneuver, this leads to the same cluster shape. This behavior results in the fact that the error within all cluster shapes $C_{LCclusterhape}$ is also the same for each method. Therefore, in Figure 5.9 we only show the variants of the lane change maneuver for the ground truth data, since all parameterization methods would only show a linear maneuver. The abscissa and ordinate show the same values as in Figure 5.8. We observe that four of the five identified clusters follow logistic growth. We identify a different start and return position after the lane change. For example, the left lane change shown in purple in Figure 5.9 starts in the middle of the main lane (-2.0 m) and also ends in the middle of the passing lane ($+2.0$ m). The orange left lane change starts near the middle lane marking (approx. -1.3 m) and ends near the middle lane marking (approx. $+1.2$ m). We discover that the red left and green right lane change deviates from the classical logistic growth due to its curvature and amplitude. The lane change takes place near the middle lane marking. Compared to the other variants, we perceive the red variant in the

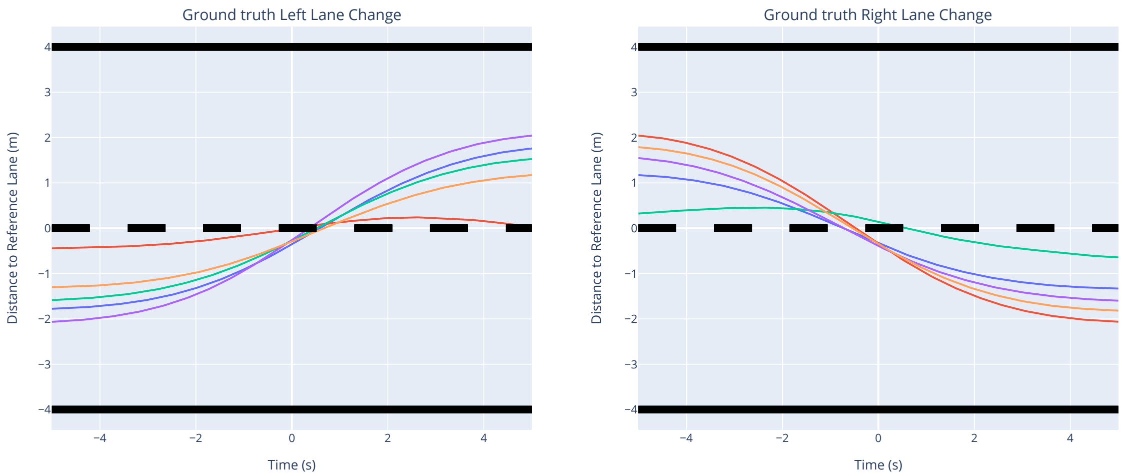


Figure 5.9: Cluster variants for the observed lane changes within the real data.

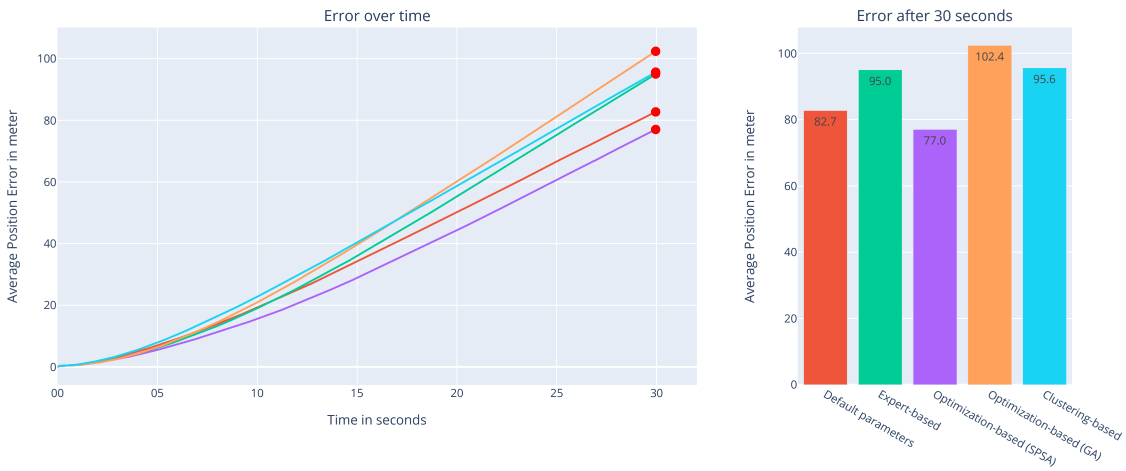


Figure 5.10: Prediction ability shown by parameterization method. The left diagram shows the positional error in meter evolving over time. The right diagram displays the last position error after 30 s. Lower values are more realistic.

context of the left lane change returning to the middle lane marking after about $t = 3$ s. The other varieties are monotonically increasing within the left lane change and do not return to the middle lane marking. The green variant in the right lane change also reflects this phenomenon, but this variant starts with this phenomenon of antagonistic progression compared to the left lane change.

Mixed Characteristics Within this paragraph, we present the results of the mixed characteristics: prediction ability (C_{pred}) and scene consistency (C_{scenes}). In Figure 5.10 we present the results of the prediction ability C_{pred}. The left plot indicates the time in seconds after the first scene on the abscissa. The ordinate shows the position error in meters compared to the ground truth data. The right diagram shows the final position error after 30 s, differentiated according to the parameterization method. We observe the lowest position error (77.8 m) after 30 seconds using the SPSA method, while the GA method has the highest error (102.4

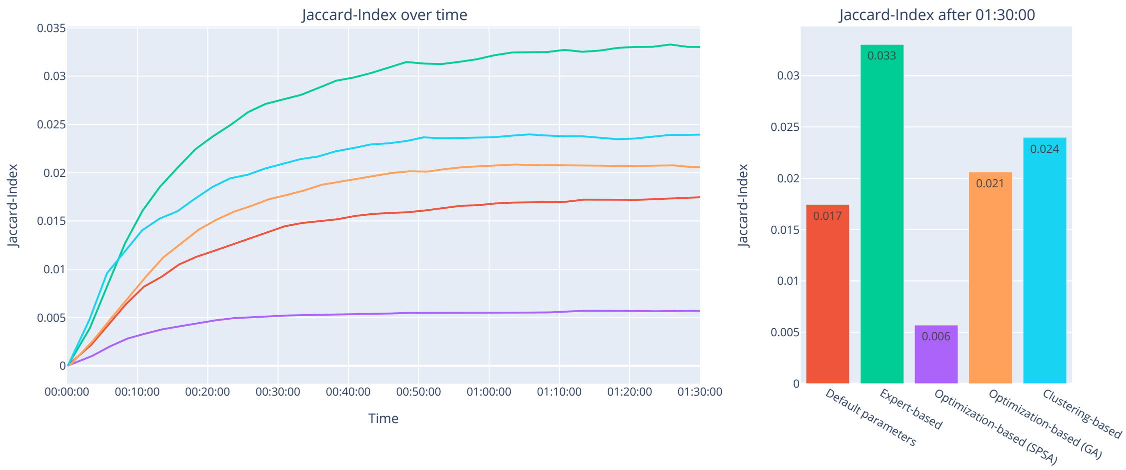


Figure 5.11: Jaccard-Index of scenes occurring in reality and simulation. The time is specified in hours. The left figure shows the Jaccard-Index over the simulation time, and the right figure shows the Jaccard-Index after 01:30:00h. Higher values are more realistic.

m). We calculate an error margin of 24.6 m between the lowest and the highest error. Within the time evolution, the SPSA method always results in the smallest error. While the GA method initially yields a lower error than the clustering-based method, this changes after about 18 s. At the last observation point of 30 s, we note a lower slope of the clustering-based method compared to the expert-based method.

In Figure 5.11, we present the results of the C_{scenes} characteristic. The left side presents the Jaccard-Index over the simulation time in hours. The right side of the figure shows the final Jaccard-Index after 01:30 hours of simulation time. We observe that the expert-based method achieves the highest final Jaccard-Index with 0.033, and the SPSA method the lowest with 0.006. Compared to these two methods, we note that the default, clustering, and GA methods achieve similar values (0.007 difference) but still have a clear ranking with the clustering-based method as the leader within this group. The expert-based method has a 550% higher Jaccard-Index compared to the lowest Jaccard-Index. The SPSA method results in a 65% lower Jaccard-Index compared to the second-lowest Jaccard-Index (compared to the highest Jaccard-Index 82%). As the Jaccard-Index evolves over time, the ranking is the same at any time except for the starting point, where the clustering-based method leads to a slightly higher Jaccard-Index. Within the Jaccard-Index, we perceive a convergence towards a maximum for all methods. We note that the SPSA method reaches this maximum after about 25 minutes. In contrast, we remark that all other methods reach this maximum value after about 50 minutes. In addition to the Jaccard-Index, we show the proportion of found scenes in Figure 5.12. This graph shows the percentage of found scenes from the ground truth dataset as a function of time. We observe the same ranking between the parameterization methods as for the Jaccard-Index. Compared to the Jaccard-Index, the given percentages do not converge within the given time interval.

In Figure 5.13, we show the proportion within the experiment 2.1. This experiment uses a simulation time of 10:00 hours. We discover the same ranking as

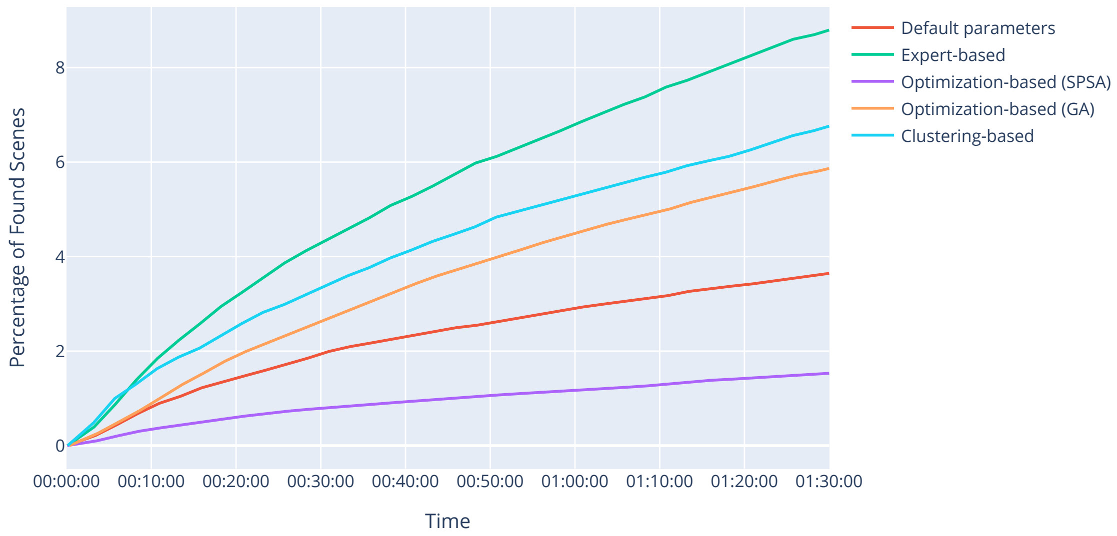


Figure 5.12: Proportion of found scenes from the ground truth dataset after 01:30 hours of simulation. Higher values are more realistic.

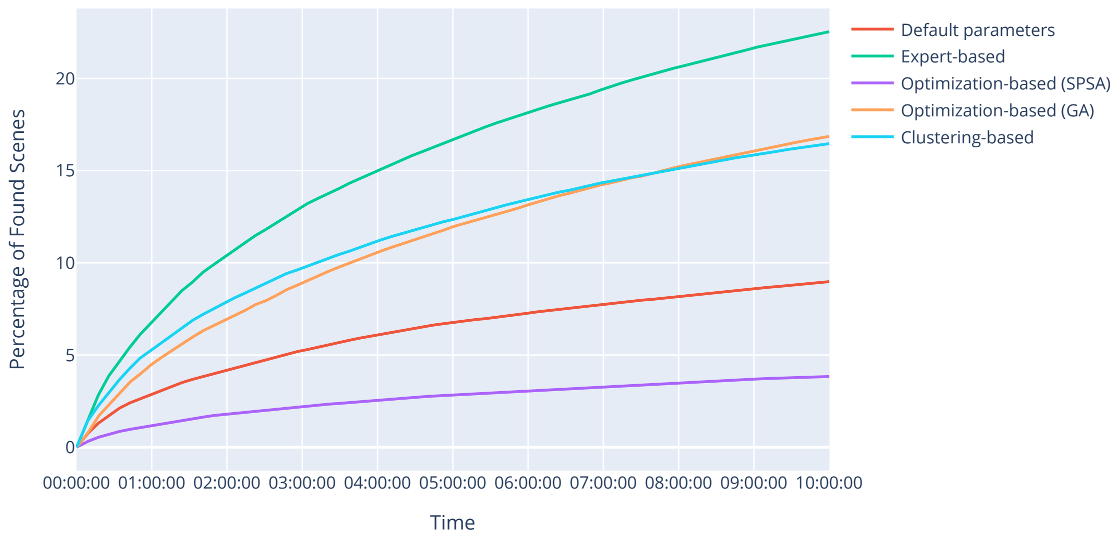


Figure 5.13: Proportion of found scenes from the ground truth dataset after 10:00 hours of simulation. Higher values are more realistic.

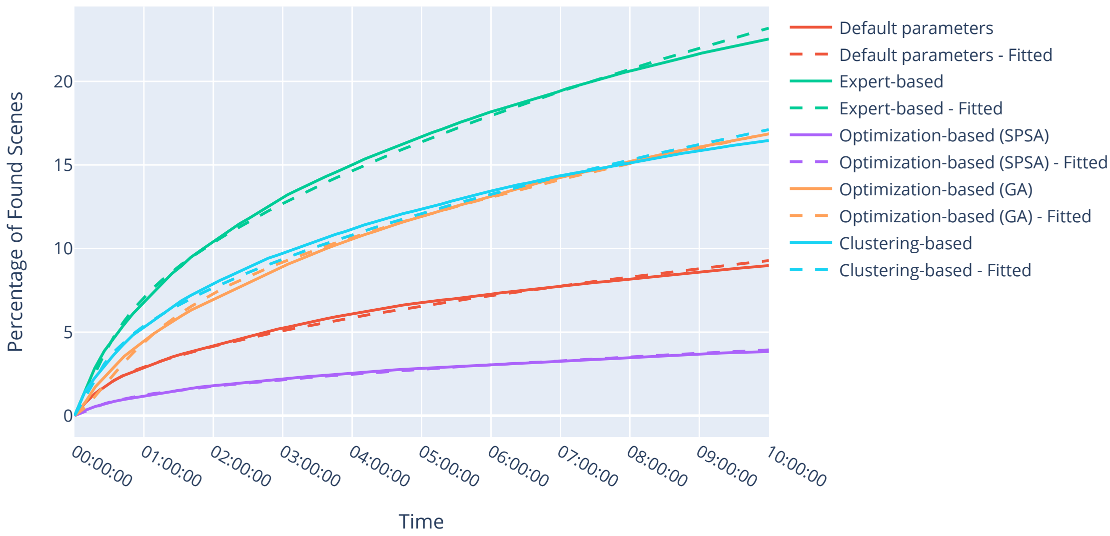


Figure 5.14: Percentage of found scenes from Figure 5.13 with Equation 5.4 fitted to the data. Higher values are more realistic.

Method	a	b	**Days until threshold reached**
Default	0.00048	0.00161	39.2
Expert	0.00122	0.00094	6.3
Optimization (SPSA)	0.00021	0.00135	217.8
Optimization (GA)	0.00089	0.00047	11.8
Clustering	0.00090	0.00134	11.5

Table 5.4: The days of simulation necessary to reproduce 90 % of the scenes contained in the real evaluation dataset (01:30 hours) according to approximation using Equation 5.4. Parameter a and b according to Equation 5.4.

within the simulation of 01:30 hours. In contrast to the 01:30 hours dataset, the clustering-based method has a lower slope in the long term than the GA method. The clustering-based and GA method intersect at about 07:30 hours, and the GA method has a higher percentage.
We use Equation 5.4 to model the time progression of the scene consistency characteristic. In Figure 5.14 we present the result of Figure 5.13 with the fitted constants of Equation 5.4. We use this fit to predict future values for the percentage of found scenes. We define threshold of 90 % of scenes found by the simulation. Based on the fitted functions we predict when this threshold is reached. The results are shown in Table 5.4 distinguished by the parameterization method. The table shows the parameters a and b according to Equation 5.4 and the days needed to generate 90 % scenes of the ground truth dataset by simulation with the given parameterization method. The days are given in simulation time. For example, it is necessary to simulate 6.3 days within the simulation using the expert-based method to generate 90 % of the scenes of the ground truth dataset according to the fit. We consider the

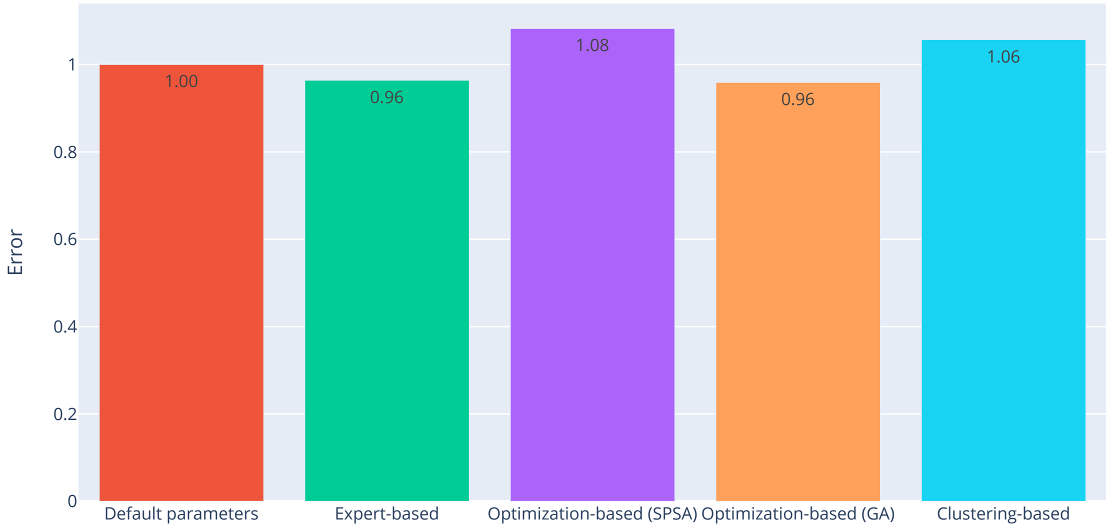


Figure 5.15: Combined results of all characteristics according to Equation 5.3. Lower values are more realistic.

SPSA method as an outlier because it requires ~34 times the simulation time compared to the expert-based method. Except for the GA and cluster-based methods, the results differ significantly from each other.
As an additional visualization, we show the total number of found scenes (compared to the percentages) in the appendix in Figure A.7. We present the total number of unique scenes generated by the simulation in the appendix in Figure A.8. We observe that the number of unique scenes within the simulation depends on parameterization. The results of the Jaccard-Index with the 10:00 hours dataset (with the same visualization as the 01:30 hours dataset in Figure 5.9) are shown in the appendix in Figure A.9. Compared to the 01:30h data set, we perceive that the Jaccard-Index does not converge to a maximum but slowly decreases after reaching the maximum.

Combined Results Finally, we combine the prior results of each characteristic. We use Equation 5.3 in order to calculate an overall error for the parameterization methods. The results are shown in Figure 5.15. The expert-based and GA method results in the lowest error overall: $E_{overall} = 0.96$. The default parameters provided by SUMO result in a slightly higher error of $E_{overall} = 1.00$. The cluster-based approach results in a slightly higher error of 1.06, closely followed by the SPSA method with 1.08. Compared to the results of the previous characteristics, the total error varies only slightly between the methods. In Figure 5.16 the error of all previous characteristics is summarized in one diagram.

5.5 Discussion

In this section, we discuss the experiment results concerning our research questions. We divide this section into four parts according to the research questions. In Section 5.5.1, we discuss the influence of parameterization and individual parameters

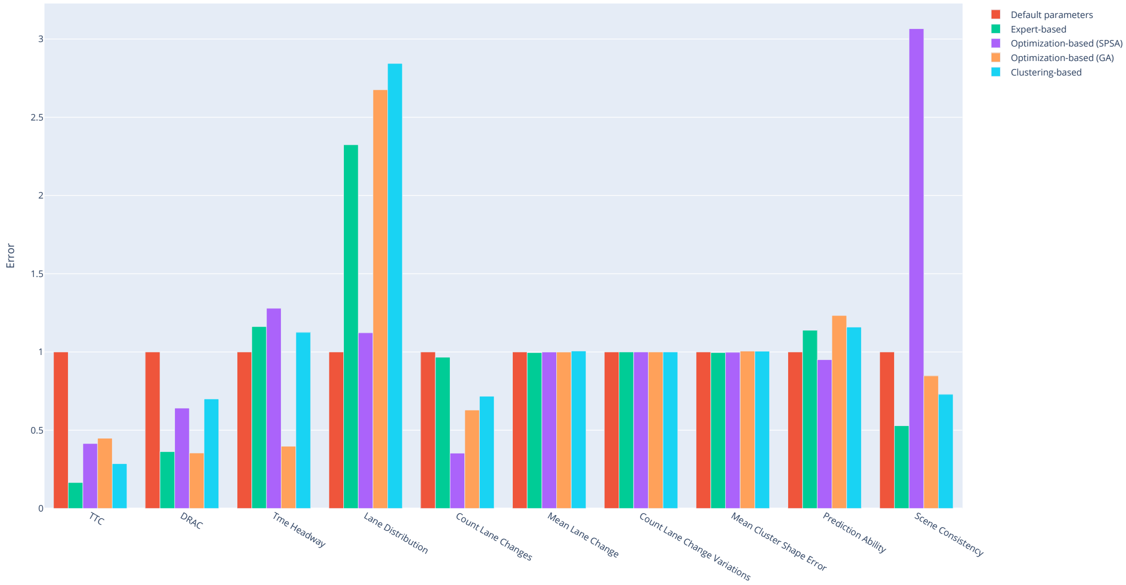


Figure 5.16: Final error within all characteristics, normalized to the error with default parameters.

thus **RQ 1**. We discuss the realism of synthetic trajectories in Section 5.5.2 thus **RQ 2**. In Section 5.5.3, we discuss which parameterization method generates the most realistic synthetic trajectories thus **RQ 3**. Finally, we discuss the impact of the evaluation characteristics in Section 5.5.4 thus **RQ 4**.

5.5.1 RQ 1: Influence of Parameterization

We expect that different parameterizations and methods will lead to a difference in the realism of the trajectories. Furthermore, we expect that individual parameters are more influential than others.
We have shown that the parameters speedFactorMue, tau, speedFactorSigma, maxSpeed, accel, and vehsPerHour are more influential than sigma, length, actionStepLength, minGap, width, and height. By the results of Figure 5.5, we conclude that if we set the speedFactorMue parameter correctly, the trajectories are closer to reality than if we adjust the eight least influential parameters correct. This statement applies only to realism concerning the five characteristics under study. We conclude that we can reduce the parameterization effort by omitting unimportant parameters. In Figure 5.4, we identify a logical connection between the parameter tau and the time headway within SUMO, as it is the most influential parameter on the time headway. This relationship is an intuitive result. In comparison, the speedFactorMue parameter is most influential on the lane distribution, which is not directly intuitive. We explain this relationship due faster vehicles do not use small gaps to change lanes but stay in the passing lane, and therefore a higher usage of the passing lane occurs. We detect direct relationships between parameters and realism aspects in the first experiment. Looking at the results of the second experiment, we observe a significant difference in the realism of the resulting parameterizations. The results of Table 5.4 show that 97 % of the required simulation time to reproduce 90 % of the scenes is reduced by choosing a different parameterization. The

dependence on parameterization does not apply to all characteristics. For example, the characteristic for analyzing the lane change maneuver does not change using different parameterizations. We explain this finding by the fact that SUMO only implement one variant of the lane change maneuver. With default settings, SUMO performs a lane change immediately. In our baseline configuration, we used linear interpolation for this lane change maneuver, which results in the same maneuver and linear progression of the lane change for all parameterizations. The implementation within SUMO explains the same results for all parameterizations. Except for the lane-change characteristics, we observe that each characteristic has a distinct method with the lowest and highest error. Interestingly, the results of all the characteristics together (Figure 5.15) suggest that there is no difference in the realism of the resulting trajectories in a broader context. We have two interpretations of this finding. First, evaluating realism in a broader context is not dependent on parameterization. Second, the choice of characteristics and the weighting cause this finding. We assume that the second interpretation is more likely because we observed that realism is heavily dependent on parameterization. Since we only inspect a small set of characteristics, we expect this finding will dissolve by other or more characteristics.

We observed during the execution of the experiments in Section 5.4.2 a dependence between the parameter sigma and traffic density. We perceive an influence of only 1.9 % of the parameter sigma on all measured characteristics within the first experiment. The low value does not resemble the strong influence and threshold of sigma to generate traffic jams. We assume that more hidden dependencies and influences exist. This finding supports that parameterization has a strong influence on realism.

RQ 1: Conclusion

Summarizing the findings discussed before, it is clear that specific input parameters are more influential than others on the realism of the resulting trajectories. By parameterization, the resulting trajectories get more realistic in specific characteristics. However, when all characteristics are combined and weighted equally, there is no significant difference between the different parameterizations.

5.5.2 RQ 2: Realism of Synthetic Trajectories

With this research question, we investigate how realistic the resulting trajectories are. We expect the simulation environment to represent specific characteristics well. Using the longitudinal characteristics shown in Figure 5.5 and the corresponding histograms shown in Figure A.6, we conclude that the simulation is able to produce realistic trajectories within these characteristics. For example, looking at the TTC distribution of the SPSA method, the distribution is subjectively very similar to the ground truth. However, the mode is at a higher TTC value compared to the ground truth, and the ground truth has a higher skewness. In addition, very low TTC values (below 2 s) rarely occur, but in reality, these values are more common. Other parameterizations reproduce the low TTC values, for example, the SPSA method. However, the course of these parameterizations is different, and they cannot realistically reproduce other TTC values. Considering the time headway, we observe that the GA method produces the most realistic results. In detail, however, we also perceive deviations from reality, for example, a lower variance.

Considering the lateral characteristics, we conclude that the simulation fails to produce realistic results. In terms of the real lane distribution, the ratio between the main and passing lanes is 75:25. The closest simulation run achieves 35:65. This discrepancy is a significant difference even with the most realistic simulation. Three of the five simulations have a ratio of about 50:50. We assume this behavior is a finding since we use the same environment model and traffic flow. We anticipate that the simulation model causes the discrepancy. Considering all simulation runs, we conclude that the simulation does not accurately represent the lane distribution. We discover that the simulation reflects the number of lane changes well using the GA method. However, we found that the number of left and right changes is different. In reality, they are about the same. We explain this finding as vehicles are placed preferentially in the main lane. We set the departure lane as "first". Thus, vehicles are placed in the right-most lane if it is free. This behavior increases the likelihood that vehicles will change onto the passing lane since they start on the main lane. Thus, a bias within the ratio of left and right lane changes is introduced. Therefore, we conclude that the lane distribution is adequately represented by the GA method as we can explain the minor deviations.
In the previous sections, we discussed that the simulation environment does not represent the lane change maneuver well. However, in reality, we observed two findings in Figure 5.9. First, we observed different reversal points within the lane (amplitude). Second, we note close lane changes to the middle lane marking in one cluster. We explain the first result with the different driving behavior of the drivers. Some drivers drive closer to the center lane, others further out. To explain the second finding, we analyze the samples associated with this cluster by hand. We discover two cases that apply to this cluster. First, we discovered slow lane changes in that cluster. We still interpret this behavior as lane changes. Second, we found trajectories that travel directly on the middle lane with minor deviations to both lanes. Both findings are recognized and mapped to this cluster.
We suggest implementing a more sophisticated lane change maneuver to evaluate the realism of the parameterization. Instead of a simple linear lane change, a logistic maneuver with the amplitude and the stretch over time as parameters represents our findings. This model represents the observed shape of the lane change and the variation within this maneuver.

Within the scene consistency, we observe that the simulation generates realistic scenes. However, we note that the percentage of scenes found is low (maximum 9 %) after 01:30 hours of simulation time. We identified different causes for this. First, the occurring road users and their behavior are random. The sheer number of possibilities for scenes on a highway makes it unlikely that the same scenes will occur. Second, the reference dataset is only 01:30 hours long. Therefore, we assume that many scenes generated by the simulation could be real but are not considered because they do not occur in the section of reality. We find within the scene characteristics that the maximum of the Jaccard-Index is around the time of the reference dataset. We explain this in terms of the simulation dataset being disproportionately large after 01:30 compared to the real dataset, resulting in a high denominator in the Jaccard-Index calculation. Further research is needed to determine if the maximum Jaccard-Index found is based on the reference dataset or is a constant for the realism of the simulation. Based on the results of Table 5.4, we assume simulation environ-

ments are suited to create realistic scenes. We estimated that for a simulation of about 6.3 days, 90% of the scenes from the ground truth dataset occur at least once within the simulation. In scenario-based testing, we can test the ADAS/AD within the simulation over a long period and therefore ensure that real scenes occur. We observed how many unique scenes are generated by different parameterizations. We find that the default parameters' total number of unique scenes differs significantly from the other methods. Unlike the other methods, this parameterization uses the Krauss CFM. Although fewer unique scenes are generated, we observe a higher percentage of real scenes found than for the GA method (Figure 5.12). We conclude that the default parameters have a higher proportion of realistic scenes.
Using the prediction characteristic, we observed that the trajectories in the simulation deviate by a positional error of more than 77 m after 30 s. We assume many influences for this result. For example, we initialize the vehicles with the parameters that are most likely at the initialization. Over time, however, the real driver may change his desired speed or be distracted, creating a discrepancy between simulation and reality. This behavior is not adequately represented in the simulation and can cause a positional error. We suggest using a test setup that controls all but one vehicle to reduce this effect.

RQ 2: Conclusion

As a conclusion of the differentiated discussion of the individual characteristics mentioned above, we note that the simulation environment is capable of generating realistic trajectories in certain aspects. It is challenging to define what precisely realistic is. We cannot identify thresholds to judge whether the results are realistic. However, we find that specific methods' results are more realistic than other methods. For longitudinal characteristics, realism depends on parameterization. In terms of lateral characteristics, the simulation environment has deficits but also manages to realistically represent, for example, the number of lane changes. Based on the developed characteristics of scene consistency, we assume that the simulation environment is suitable for generating realistic scenes for scenario-based testing.

5.5.3 RQ 3: Comparison of Parameterization Methods

We expect that the results of the parameterization methods differ in realism. Furthermore, we expect one method to generate the most realistic results.
Against our expectations, we conclude that there is not a single method that generates the most realistic trajectories by Figure 5.15. We identify that many settings within the parameterization methods are a challenge in comparing these methods. An example of this is the optimization-based method. This method has many optimization strategies available (we tested two), whose hyperparameters must be adjusted, the objective function, and the input parameters must be chosen. These settings make it difficult to determine whether a single method produces the most realistic results because the methods resemble an abstract concept with many implementations.
By Figure 5.16, we observe that specific methods perform the most realistic results in certain aspects. For example, the expert-based method generates the most realistic scenes. For each parameterization method, we find one characteristic with the

most realistic and one characteristic with the least realistic results, except for the cluster-based method. The cluster-based method does not provide the most realistic results in any characteristic. However, we remark that this method does not generate the least realistic results within the combined results. We note that the results for some characteristics are consistently less realistic with parameterization than with the default parameters. This finding applies, for example, to the lane distribution. We identify two reasons for this behavior. First, the default parameters are already realistic. Second, parameterization cannot handle these aspects well. We assume that a combination of both applies. First, because SUMO uses parameters from literature and own research as default parameters. Second, because, for example, the optimization-based methods do not optimize against all characteristics due to technical limitations. This limitation will lead to the fact that the optimization will not consider the aspects represented by these characteristics. This reasoning also explains why the optimization-based methods do not provide the optimal solution.

RQ 3: Conclusion

We note that the optimization-based methods have the highest execution times (Table 5.2). Comparing the execution time with the results of Figure 5.15, we observe that an expert-based configuration is overall the most efficient and the most effective. However, the expert-based method does not generate the most realistic results in all aspects. We still expect a high potential in the systematic methods. For example, the GA-based method achieves the same low overall error. We conclude that the optimization-based methods are particularly applicable for generating realistic trajectories in well-describable characteristics. In summary, we did not find any single method that generates the most realistic results overall. We have found that specific methods are well suited to generate realistic trajectories under certain aspects. The chosen characteristics strongly influence which method is the most appropriate.

5.5.4 RQ 4: Influence of Evaluation Characteristics

With this research question, we want to investigate whether the results of all characteristics are consistent with each other and whether these characteristics are suitable to evaluate realism. We expect some characteristics to reflect several others, meaning that they are interdependent. Moreover, we assume that some characteristics are not well suited to evaluate the realism of trajectories.

Considering the differentiated results of Figure 5.16, we argue that the selected evaluation characteristics have a significant impact on the final assessment of which method generates the most realistic results. We base this statement on the fact that the ranking within the characteristics differs. We conclude that the lane change characteristics are unsuitable for comparing realism in the given setting. Since SUMO implements only one lane change variant, all parameterizations perform this variant. This behavior results in the same error for all parameterizations and does not differentiate the parameterizations. We assume that this characteristic is still suited to evaluate realism between parameterizations within this setting. Against our expectations, we did not find a single characteristic that reflected several other characteristics. Each characteristic has an individual ranking and scales of error.

We observe that single characteristics are highly influential within Figure 5.16. Due to the differences within the scene consistency, the error for the SPSA method is high. If we evaluate only the longitudinal and lateral characteristics, the combined error of the SPSA method with 0.79 is significantly lower than the second-lowest method (GA) with an overall error of 0.96. This consideration would lead to a distinct method with the most realistic results. However, by doing so, we exclude the scene consistency characteristic. This characteristic is the only characteristic that allows a direct statement about realism. Testing an ADAS/AD within the simulation, a higher rate of realistic scenes ensures that the ADAS/AD will be tested in more realistic situations. Therefore, we conclude that the characteristic of scene consistency is an essential characteristic.

RQ 4: Conclusion

We conclude that the chosen characteristics are crucial for the assessment of realism. We observe a strong dependence on the evaluation result by the chosen characteristics and their weighting. To define abstract realism, we assume that many individual characteristics need to be examined.

5.6 Threats to Validity

In this section, we examine the validity of our study and its results. We discuss possible threats and our approaches to mitigate them. First, we examine which factors influence our results and thus internal validity. Second, we consider external validity, in which we discuss the generalizability of our results. Finally, we investigate the construct validity by examining our within-concept biases.

5.6.1 Internal Validity

Evaluation Bias Influencing the results by an invalid evaluation is possible, for example, by an error in the evaluation scripts. We use reliable and well-tested software frameworks to calculate the results. Thus, we prevent implementation errors. Furthermore, we use different visualizations and data representations for the results to identify possible errors. We formulate our expectations before the experiments, discuss them and compare them with our results. In this way, we prevent a biased evaluation.

Experimenter Bias The implemented process chain automates the experiments and the result calculation to minimize the possible influence of the experimenter. However, the selection and configuration of the parameterization methods is a manual process. We select the methods according to frequently used methods found in the literature. We also use configurations from the literature and use them as predefined configurations to limit the influence of the experimenters. To reduce the impact of manual implementations, configurations, and errors, we use existing frameworks and have implemented different versions of the parameterization methods and tested various settings. In this way, we control the experimenter's bias.

Bias of the Reference Data We use trajectory data provided by testbed Lower Saxony as a reference. The accuracy and validity of the trajectories provided are necessary to produce reliable results. Regular measurements on various reference vehicles with high-precision positioning systems validate the accuracy of the trajectories. Thus, we ensure reliable and accurate reference data. The used reference data represent only five hours of the real world. We explicitly chose a data set on a weekday and in the morning to examine rush hour traffic as an example. Due to time constraints, we do not examine the results over more extended periods and other conditions. Within the limits of our technical capabilities, we have opted for the largest amount of reference data to achieve the largest possible representation of reality. Thus, we control the independent variables.

Bias of the Simulation Environment We use a simulation environment to generate synthetic trajectories. A simulation environment has several parameters that influence the trajectories. Some, like the CFM parameters, directly influence the trajectories since they describe the drivers' behavior. There are also indirect influences due to, for example, the road model or the simulation step-length. These parameters influence the dependent variables (realism aspects). We address this problem by setting these parameters according to the reference data. For example, we use the sampling rate of the real data as the step length and a high-resolution road model based on reality. Furthermore, we use the same settings for these non-driving parameters across all simulation runs. Thus, we ensure that we measure the influence of the independent variables (drivers' behavior) on the dependent variables (realism aspects).

5.6.2 External Validity

Generalizability to other Traffic Settings The developed concept applies to different traffic settings. We paid explicit attention to the modularity and interchangeability of individual components during the concept development, such as the input data or parameterization methods. We used the maximum technically possible size for the reference dataset to maximize the generalizability. Therefore, we expect good generalizability across similar highway settings regarding the results of our experiments. We expect the results to differ from those presented in this thesis on urban traffic. Furthermore, we expect the results to be different under other environmental conditions. Especially with different days of the week, times, or weather conditions. Further research is necessary to investigate the behavior under different environmental conditions. However, the developed concept is also applicable to urban areas and other environmental conditions.

Generalizability to other Simulation Environments We implemented our concept using SUMO as a simulation environment. We build the process chain with a generic interface to accept trajectory data from different sources. After loading the data into the process chain format, the source of the trajectory data becomes insignificant, as all further steps use this same format. Other Microscopic Traffic Simulations (MTSs) work similarly to SUMO, for example, they use the same CFMs or road models. Therefore, we expect good generalizability across other simulation environments.

Generalizability to other Evaluation Measures In our concept, we introduced ten different characteristics to measure realism. In the context of our experiments in Section 5.4, we have shown that it is possible to extend our concept with additional evaluation characteristics. Therefore, we expect good generalizability of our approach when using different characteristics for evaluation. Considering the results, we expect that the evaluation of the parameterization methods will lead to different results when other characteristics are chosen, and the generalizability will suffer. In our discussion in Section 5.5 we have shown that the selected characteristics are mainly responsible for the outcome of the realism evaluation. We use a variety of characteristics in order to cover a broad spectrum to evaluate realism and increase generalizability. However, future work is necessary for urban roads and cities since many new situations like traffic lights or roundabouts are introduced.

5.6.3 Construct Validity

Bias in Experimental Design We evaluate the results of the different parameterization methods with multiple characteristics. Thus, we reduce the probability of making false statements that could result from limiting the observation. To measure realism, we calculate a reference using the real data and evaluate the deviation from it. In this way, we ensure that we measure realism. We always followed existing and well-established methods to calculate and assess the deviation. If no method is available, we justify mathematically and logically why the evaluation with our chosen method is appropriate. Thus, we establish construct validity.

6. Related Work

In this chapter, we present related work. We introduce related works and show in which aspects ours differs from them. We distinguish between scenario-based testing, simulation-based scenario generation, and calibration of simulation environments within these works.

Scenario-based Testing

Scenario-based testing is a suitable approach to address the issues associated with verification and validation of ADAS/AD. Several research papers in this field can be found in the literature. Tatar [Tat15] distinguishes the process of scenario generation into data-based and knowledge-based approaches. Bagschik et al. [BMKM18] proposes a knowledge-based approach using an ontology for generating scenarios. They use the model of scenario layers introduced by Schuldt [SSL+13] and knowledge about each layer to vary the parameters of these layers and generate scenarios from them. In order to generate each layer (L1-L5) they use guidelines and catalogs to implement the ontology. From an initial scene, they derive possible end scenes and appropriate transitions. Bagschik et al. [BMKM18] export the resulting scenarios as a scenario graph and a custom visualization. Similar to our approach, they focus on generating particularly typical scenarios for German highways. In comparison, they use a knowledge-based approach and do not evaluate their results on real data, only regarding the correctness of the ontology.
Zhou and Re [ZdR17] propose a data-driven approach to collect scenarios from real data. They define a metric used to identify critical scenarios. With this indicator, they collect a scenario catalog. They evaluate their method using a performance measure that aims to find a safety boundary. In comparison to Bagschik et al. [BMKM18] and our method, they explicitly focus on critical scenarios.

Simulation-based Scenario Generation

Sippl et al. [SBW+16] present a general concept to collect situations from a simulation environment in order to derive test cases for scenario based-testing. They use maneuver spaces (logically separated regions) around a subject vehicle to derive

the test cases. To describe the test cases, they use a domain-specific language. We differentiate by extracting complete scenarios and not generating test cases from these.
Yue et al. [YSWL20] present an approach using SUMO to extract scenarios. They model the urban environment of Shenzhen. Yue et al. [YSWL20] calibrate SUMO on a complete road network and focus the calibration process on the traffic density. They use a Scenario Risk Index based on the TTC and a loss to extract scenarios from the simulated traffic. As a result, they collect different scenario types and analyze them. In comparison, we focus on calibrating the realism of trajectories and evaluating them against reality. Additionally, we explicitly extract all scenarios, not just the critical ones, and we set up SUMO on a highway area with a limited road network.

Calibration of Simulation Environments

In traffic simulation calibration, there are several studies on the process and additional guidance. A part of the research focuses on traffic flow calibration [BABA+07, FKBN18, TKD+03]. This research aims to find the correct traffic flow on road networks and optimize the results using real traffic flow data. Vehicle detector loops often provide the source for the real data. In comparison, we focus on calibrating the realism of the trajectories rather than the overall traffic flow. Therefore, we use real trajectory data instead of traffic flow data for calibration.
Another part focuses on calibrating the CFM within the simulation environment [KT08b, CLHG10, VNS+14]. Kesting and Treiber [KT08b] calibrate the IDM using a GA for optimization. They use the error within the gap behavior between simulation and reality as objective. As a reference, they use three empirical trajectories. We also use a GA for optimization but formulate our objective from a variety of characteristics and use a dataset containing thousands of trajectories. Chen et al. [CLHG10] also calibrate the IDM-CFM using a GA. As reference dataset they use the NGSIM [FHW07] dataset. They also use the measured gap between simulation and reality as the objective function. In comparison, we use a different objective function and focus our research on the interaction with scenario-based testing. We aim to determine which calibration method generates the most realistic scenarios.

7. Conclusion

In scenario-based testing, generating and acquiring the scenarios needed for testing is a challenge. Simulation environments are used to generate scenarios reproducible and test the ADAS/AD. However, it is not clear whether simulation environments sufficiently resemble reality to use the generated scenarios for verification and validation of ADAS/AD. To investigate whether simulation environments are suitable to generate realistic scenarios, we use real trajectory data from the testbed Lower Saxony. We focus our research on the dynamic objects within the generated scenarios.

We contribute a concept to generate realistic scenarios within the simulation and evaluate the realism within the dynamic objects of scenarios. The concept consists of two components: (1) the generation of realistic dynamic objects and (2) their realism evaluation. We developed a generic process chain that generates synthetic trajectories and compares them through characteristics observed in real and synthetic traffic.

We implement this process chain prototypically with the simulation environment SUMO and real trajectory data from the testbed Lower Saxony. We consider three methods for parameterization: (1) expert, (2) optimization, and (3) clustering-based method. With the three different methods, we parameterize SUMO. Based on the simulated trajectories by SUMO, we compare them with reality and the default configuration of SUMO. We use ten different characteristics to compare the dynamic objects, for example, the distribution of the TTC, the number of lane changes, or the scenes occurring in simulation and reality. Finally, we assess realism by comparing the characteristics measured within synthetic and real trajectory data.

We conduct two experiments: the first one investigates the influence of parameterization, and the second one the realism of the trajectories. Based on the first experiment, we conclude that parameterization significantly impacts the realism of the trajectories. For example, we find that the expert-based parameterization generates about 150 % more realistic scenes than no parameterization. Furthermore, we discovered that individual simulation parameters are highly influential for certain aspects of realism. For example, the velocity deviation of the vehicles in the simulation influences the realism of the TTC distribution about five times more than the acceleration ability. In order to generate realistic scenarios, we suggest first examining which parameters influence the measures of realism. Based on these results, we suggest fine-tuning these parameters to generate realistic scenarios efficiently.

Based on the second experiment, we find that SUMO generates trajectories that are realistic, for example, within the TTC distribution. In contrast, we discover that the generated trajectories by the simulation are not realistic in all aspects. For example, SUMO does not represent the variation of the lane change maneuver observed within reality. Within this experiment, we identify that the evaluation of realism is heavily dependent on the evaluation measures. For example, the expert-based method generates an unrealistic time headway distribution but the most realistic scenes. We suggest using multiple aspects to measure the realism within scenario-based testing. Depending on the domain, we suggest using a weighting of these aspects, for example, a higher weighting of TTC distribution to generate a realistic level of criticality.

Within this thesis, we successfully build a process chain that generates realistic trajectories within the simulation. We evaluate the realism based on real trajectory data from the testbed Lower Saxony. We successfully demonstrated experimentally that the resulting trajectories are realistic, for example, in terms of their TTC distribution.

8. Future Work

In this chapter, we present future work and possible future research based on the results of this thesis. In our experiments, we used a dataset of 05:00 hours in length to evaluate and study the driving behavior. With the possibility of 24/7 detection and traffic analysis by the testbed Lower Saxony, we propose a detailed investigation of a broader database. We expect environmental factors to impact the driver's behavior, such as weather conditions or time dependencies. In particular, the fifth layer of the scenario model from Bagschik et al. [BMM18] can be studied. With an enlarged database, our findings can also be substantiated and confirmed.

Considering the approach of the cluster-based method, we propose to study the drivers' behavior using different representations. We used a representation based on manually created properties of the trajectories transformed by a dimensionality reduction strategy. We suggest a representation of the driver's behavior using other methods, for example, artificial intelligence methods such as the autoencoder shown by Rakos et al. [RABS20]. In contrast to the manually selected characteristics, we expect the high-dimensional representations generated by artificial intelligence algorithms to perform better since they can uncover hidden dependencies.

We suggest further research for the scene consistency characteristic since we expect a high relevance of this characteristic within scenario-based testing. We propose to study this characteristic based on a broader data basis. In particular, how the thresholds for the Jaccard-Index and the percentage of scenes found evolve when there are more scenes to match. Furthermore, we suggest investigating the scenes which do not occur in reality since they can challenge an ADAS/AD.
Finally, we propose to extend scene consistency in the direction of examining the degree of realistic scenarios. Therefore, we suggest adding a temporal component to the current representation. With such a characteristic, the statements about the realism of scenarios become more reliable since it directly resembles if a scenario occurs in reality.

A. Appendix

Parameter	Minimum	Maximum
maxSpeed	20	100
speedFactor mean	0.5	3
speedFactor deviation	0.01	2.5
sigma	0	1
minGap	0.2	100
tau	0.05	5
actionStepLength	0.05	1

Table A.1: Parameter bounds for optimization-based parameterization method.

Parameter	Value
carFollowModel	EIDM
maxSpeed	23.07
speedFactor mean	0.90
speedFactor deviation	0.1
sigma	0.77
minGap	0.23
tau	1.12

Table A.2: Final simulation parameters determined by the optimization-based (SPSA) method.

Parameter	Value
carFollowModel	EIDM
maxSpeed	95.98
speedFactor mean	1.27
speedFactor deviation	0.14
sigma	0.44
minGap	2.38
tau	0.60

Table A.3: Final simulation parameters determined by the optimization-based (GA) method.

Parameter	CFM	speedFactor	tau	Probability
Passenger Car 0	EIDM	normc(1.17, 0.08, 0.79, 1.60)	1.80	0.32
Passenger Car 1	EIDM	normc(1.33, 0.06, 0.88, 1.85)	1.99	0.24
Passenger Car 2	EIDM	normc(0.68, 0.06, 0.45, 0.91)	4.05	0.05
Passenger Car 3	EIDM	normc(1.02, 0.03, 0.75, 1.35)	3.56	0.12
Truck 0	EIDM	normc(0.85, 0.02, 0.74, 0.95)	3.17	0.10
Truck 1	EIDM	normc(1.07, 0.05, 0.79, 1.40)	1.72	0.02
Truck 2	EIDM	normc(1.02, 0.02, 0.76, 1.52)	2.32	0.01
Truck 3	EIDM	normc(0.87, 0.03, 0.73, 1.09)	3.56	0.02
Truck 4	EIDM	normc(0.72, 0.05, 0.38, 0.92)	5.57	0.01
Truck 5	EIDM	normc(0.87, 0.03, 0.74, 1.06)	2.09	0.03
Van 0	EIDM	normc(1.14, 0.07, 0.79, 1.48)	1.73	0.04
Van 1	EIDM	normc(0.64, 0.07, 0.42, 0.87)	4.70	0.01
Van 2	EIDM	normc(1.25, 0.05, 0.85, 1.69)	2.14	0.02
Van 3	EIDM	normc(1.00, 0.03, 0.73, 1.27)	3.05	0.02
Motorcycle 0	EIDM	normc(0.85, 0.02, 0.62, 1.56)	1.32	0.01

Table A.4: Final simulation parameters determined by the clustering-based method. The representation of speedFactor refers to the representation of the SUMO normal distribution: normc(mean, deviation, min, max).

Parameter	**Value**
carFollowModel	EIDM
speedFactor mean	1.14
speedFactor deviation	0.16
speedFactor min	0.92
speedFactor min	1.51
sigma	0.75
minGap	2.0
tau	1.1
delta	2.0
stepping	0.25
tpreview	4.0
tPersDrive	3.0
tPersEstimate	10
treaction	0.5
ccoolness	0.99
sigmaleader	0.02
sigmagap	0.1
sigmaerror	0.1
jerkmax	3.0
epsilonacc	1.0
taccmax	1.2
Mflatness	2.0
Mbegin	0.7
maxvehpreview	0.0
vehdynamics	0.0

Table A.5: Final simulation parameters determined by the expert-based method.

Algorithm 1 Parallel Optimization

Algorithm A.1

$N \leftarrow$ *Number of minimas to inspect*
$\theta \leftarrow$ *Threshold of minima distance*
$\gamma \leftarrow$ *Step width*
$results \leftarrow$ *Current results*
$lp \leftarrow$ *Queue of last N parameter sets*
$p \leftarrow$ *Propability of randomly mutating optimal parameters*
Sort $results$ in ascending order of their function value
$ps_{opt} \leftarrow None$
for $ps_n = ps1, ps2, \ldots$ *in results* **do**
    if ps_n is not within range of θ regarding lp **then**:
        $ps_{opt} \leftarrow ps_n$
        break
    end if
end for
if ps_{opt} is $None$ **then**:
    $ps_{opt} \leftarrow$ *current global optimum (first element in results)*
end if
$choice \leftarrow$ *random choice with probalility p*
if $choice$ is 1 **then**:
    for $p_n = p1, p2, \ldots$ *in list of optimzable parameters* **do**
        if enough information is available to determine a gradient **then**:
            $p_{opt} \leftarrow p_{opt}$ *with p_n changed γ in gradient direction*
        else:
            $ps_{opt} \leftarrow$ *random mutation of* ps_{opt}
        end if
    end for
else:
    $ps_{opt} \leftarrow$ *random mutation of* ps_{opt}
end if
$ps_{opt} \leftarrow$ *Check bound of* ps_{opt}
Append ps_{opt} to list of next executed simulation
Repeat until convergence

Characteristic	Description
Tau	Time headway between two following vehicles
Length	Length of the vehicle
Width	Width of the vehicle
Height	Height of the vehicle
Minimal Gap	Minimal gap to a leading vehicle as 5 % percentile
Mean Gap	Mean gap to a leading vehicle
Deviation Gap	Standard deviation of the gap for a trajectory
Minimal TTC	Minimal TTC as 5 % percentile
Mean TTC	Mean TTC for a trajectory
Minimal velocity	Minimal velocity as 5 % percentile
Maximal velocity	Maximal velocity as 95 % percentile
Mean velocity	Mean velocity for a trajectory
Deviation velocity	Standard deviation of the velocity for a trajectory
Maximal acceleration	Maximum acceleration as 95 % percentile
Desried speed	Desired velocity (only observabel if no vehicle ahead)
Minimal desried speed	Minimum desired speed as 5 % percentile
Maximal desried speed	Maximum desired speed as 95 % percentile
Count lane changes	Number of lane changes for this trajectory
Maximal DRAC	Maximum DRAC as 95 % percentile
Comfort	Standard deviation of acceleration vs. mean velocity $\frac{\sigma_a}{\bar{v}}$ [KGH06]
Lane percentages	the used lane percentages (main or passing lane)

Table A.6: Characteristics used for trajectory clustering. A leading vehicle is assumend when the gap is less than 120 m [HA14].

Threshold	Prominence	Precision	Recall
0.25	0.25	32.14%	93.95%
0.5	0.25	75.32%	89.19%
0.75	0.25	96.78%	66.32%
0.25	0.5	49.28%	92.47%
0.5	0.5	**93.81%**	**88.84%**
0.75	0.5	97.11%	63.67%
0.25	0.75	51.86%	70.59%
0.5	0.75	94.53%	68.17%
0.75	0.75	97.16%	61.43%

Table A.7: Experimental results for the CCLCI with the given thresholds.

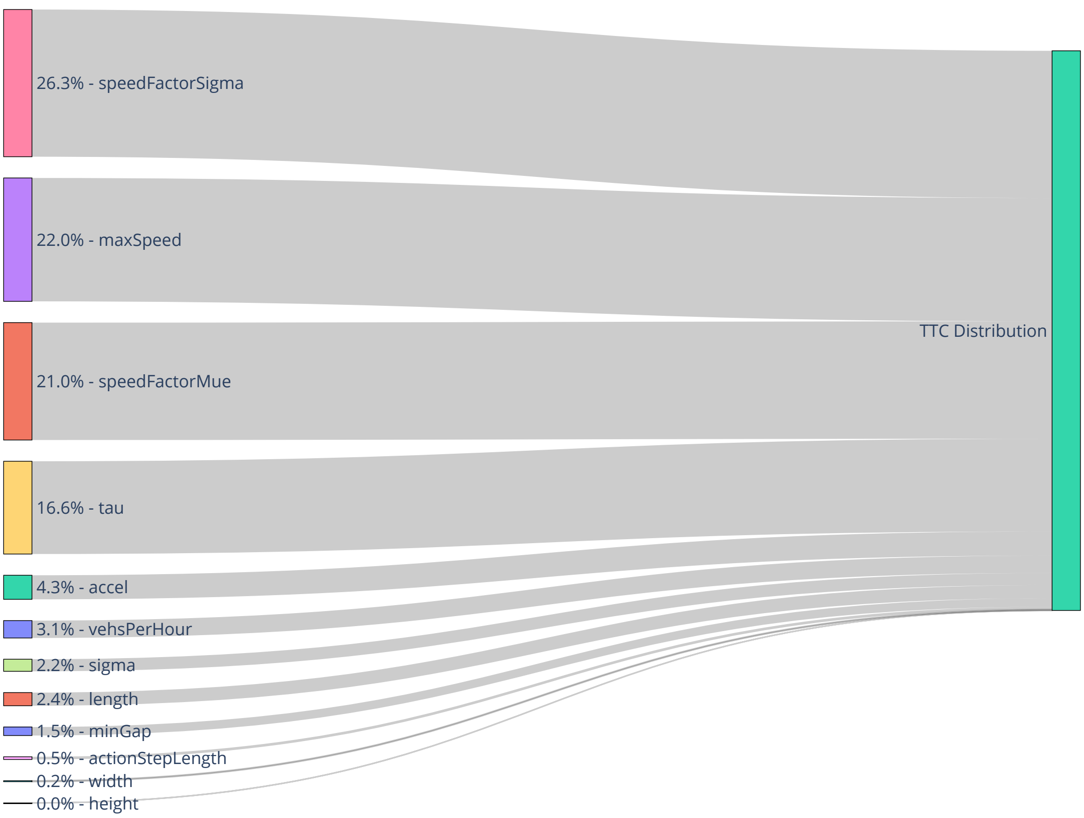


Figure A.1: Parameter impact on the TTC distribution C_{TTC}.

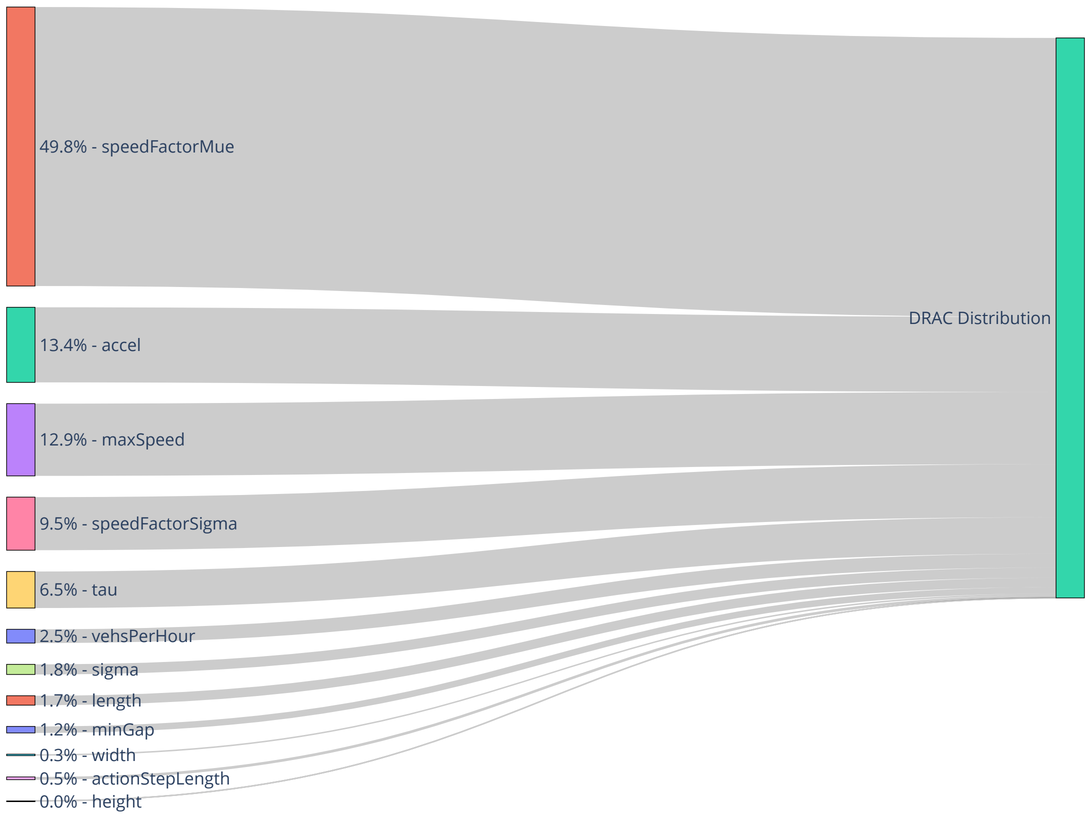


Figure A.2: Parameter impact on the DRAC distribution C_{DRAC}.

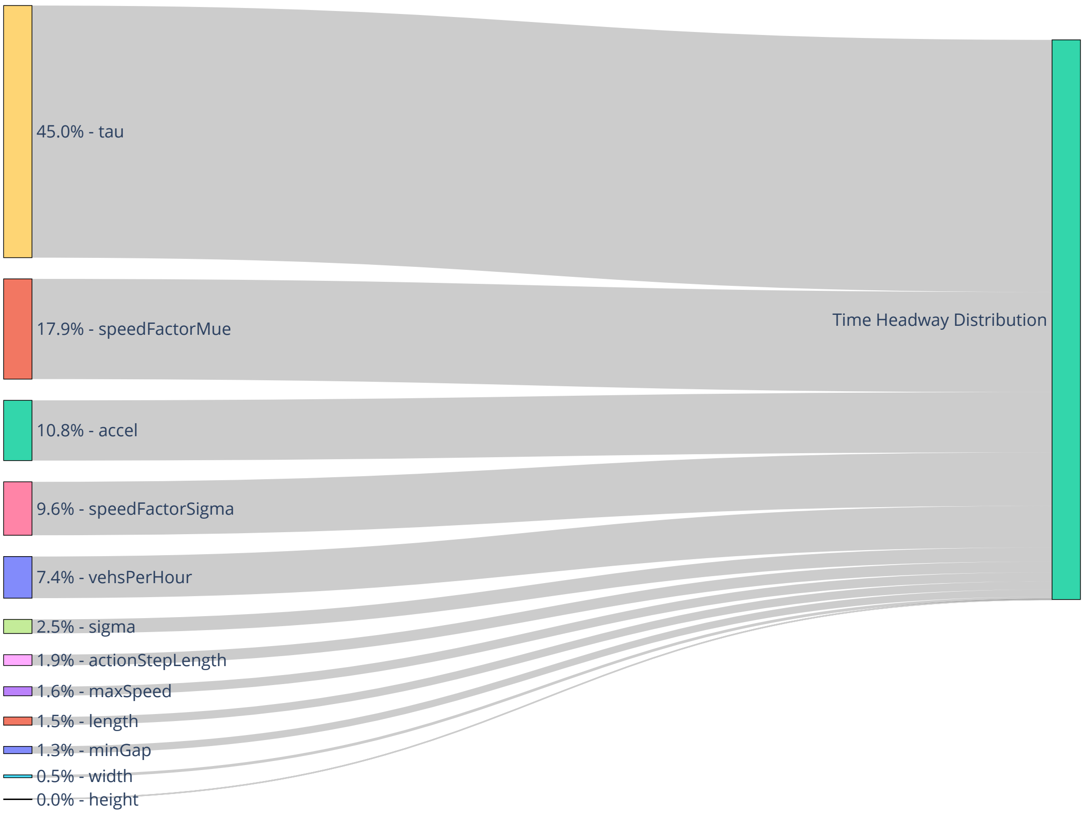


Figure A.3: Parameter impact on the TH distribution C_{TH}.

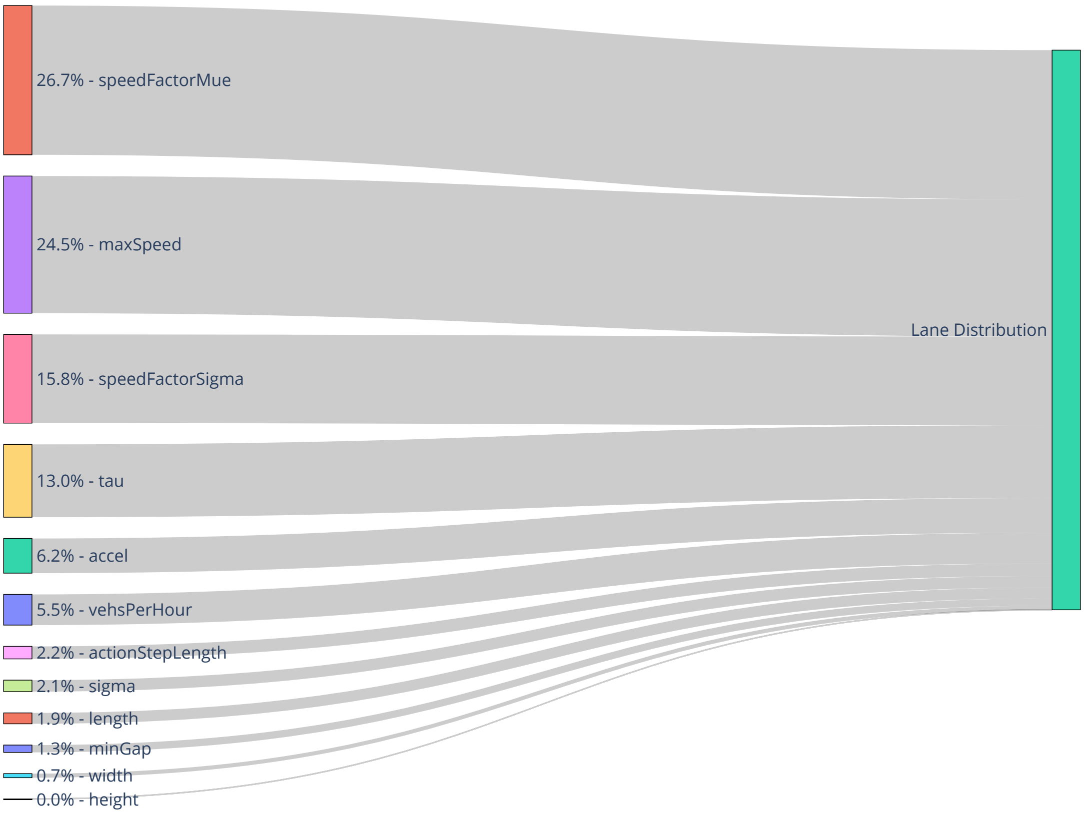


Figure A.4: Parameter impact on the lane distribution C_{LD}.

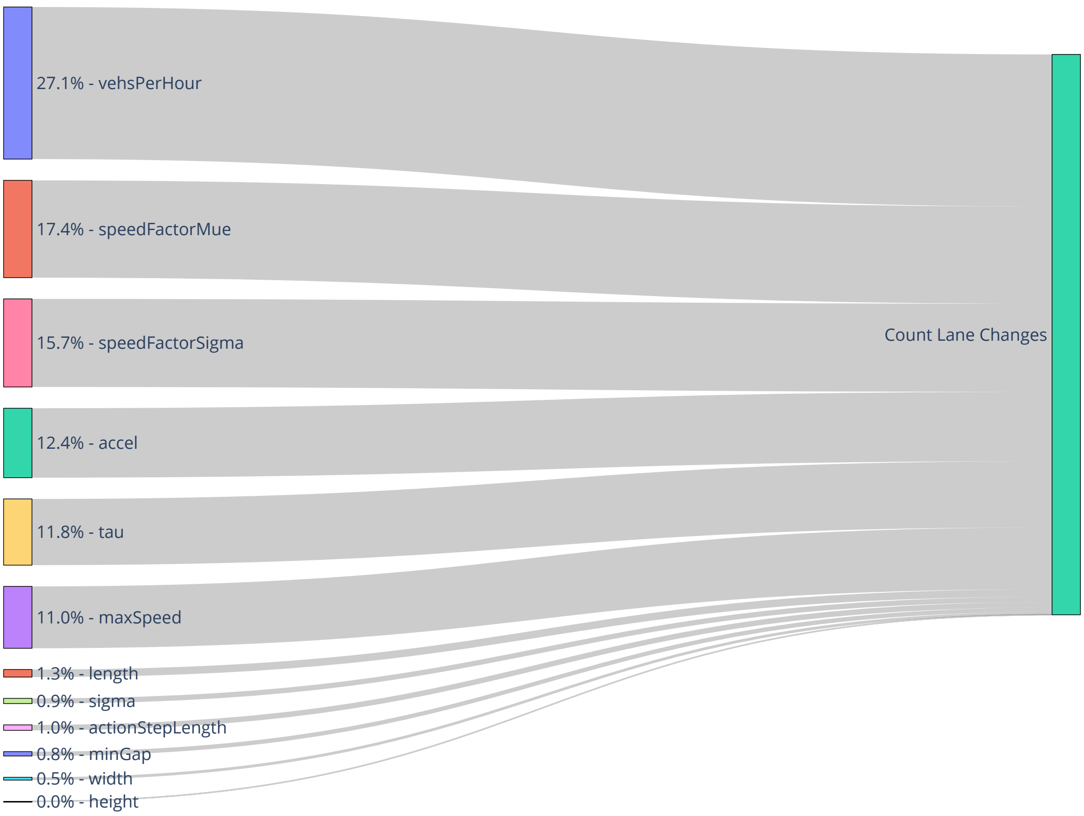


Figure A.5: Parameter impact on the number of lane changes $C_{|LC|}$.

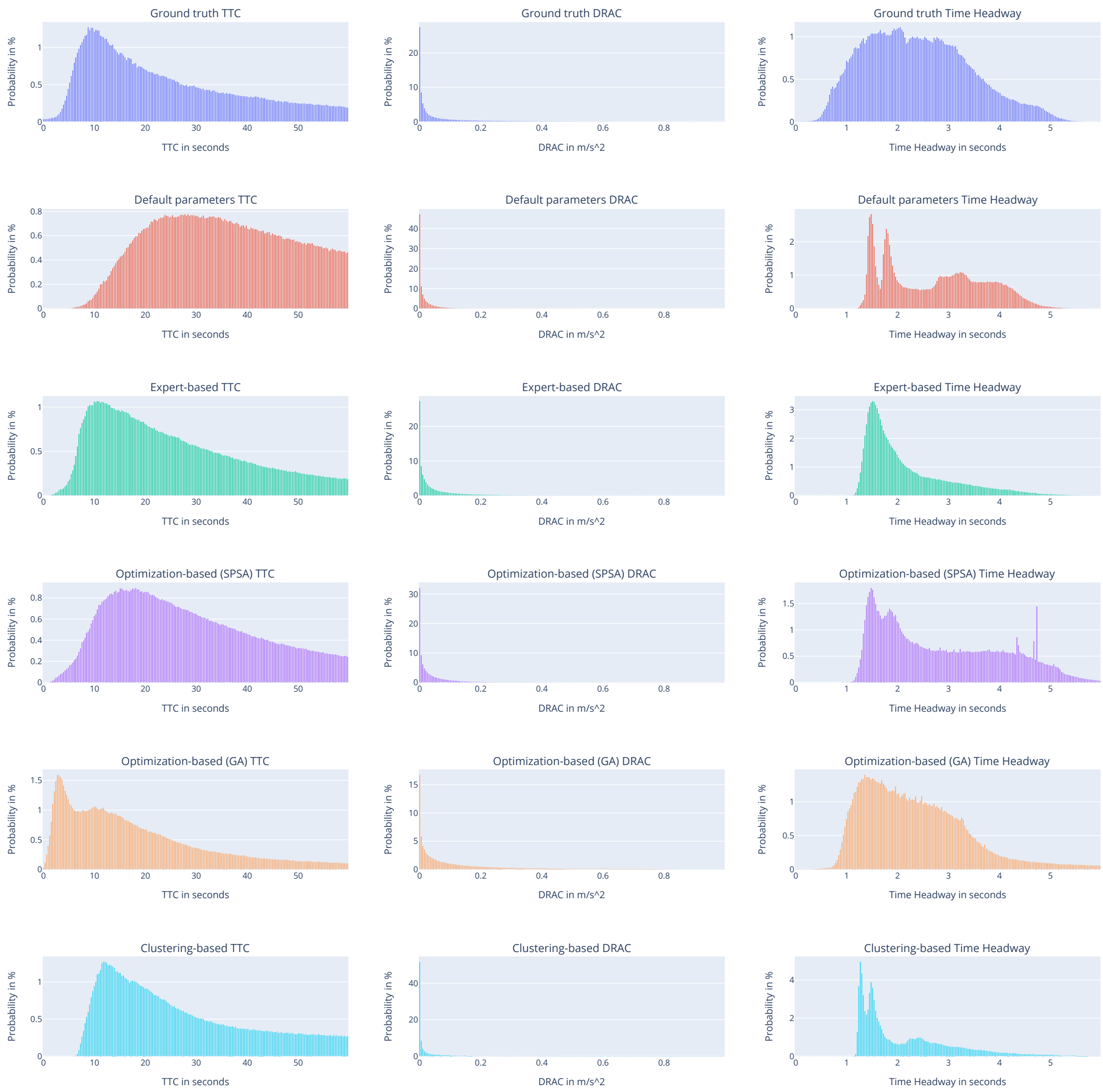


Figure A.6: Visualization of each characteristic (TTC, DRAC, and TH) as a histogram from each parameterization method.

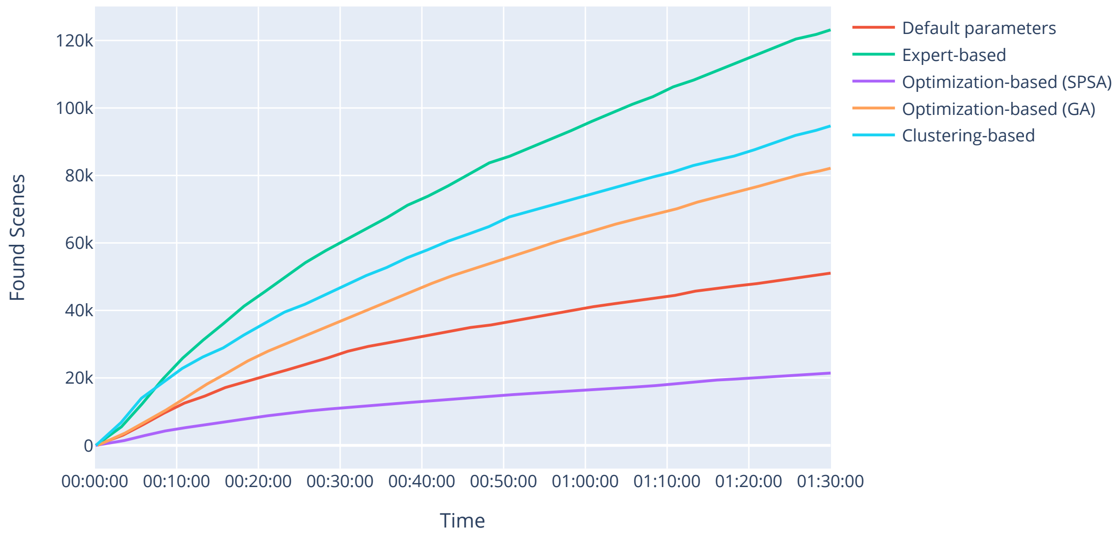


Figure A.7: The total number of found scenes from the simulation.

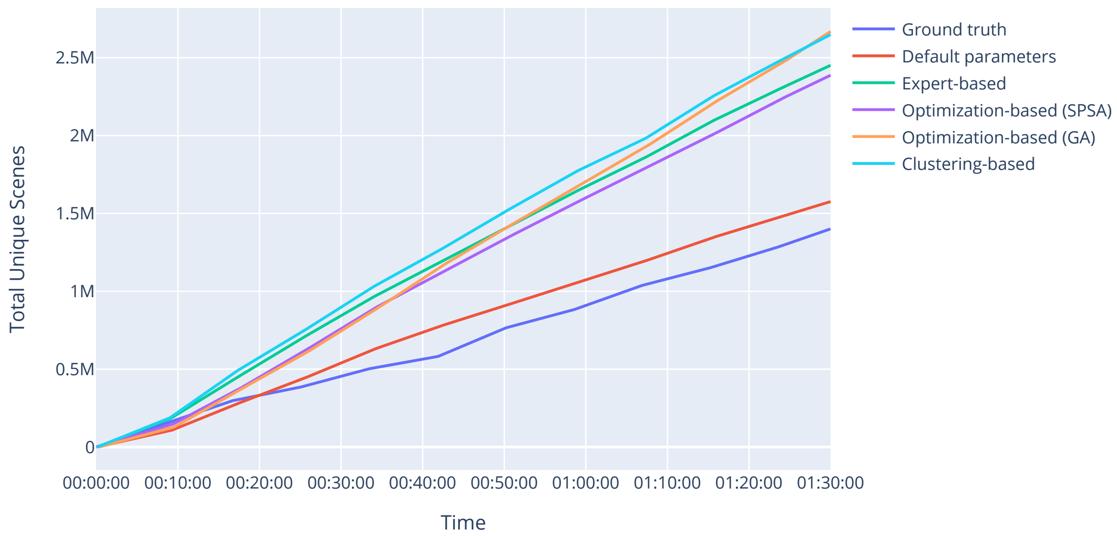


Figure A.8: The total number of generated scenes from the simulation.

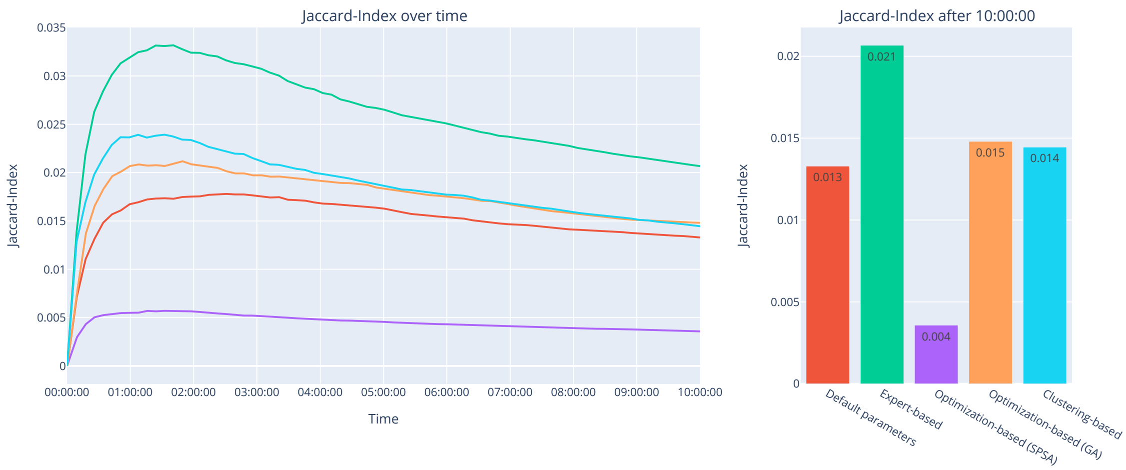


Figure A.9: Jaccard-Index of scenes occurring in reality and simulation. The time is specified in hours. The left plot shows the Jaccard-Index over the simulation time and the right plot shows the Jaccard-Index after 10:00:00 hours.

Bibliography

[AB14] Baker Kh Abdalhaq and Maher I Abu Baker. Using meta heuristic algorithms to improve traffic simulation. *Journal of Algorithms*, 2(4):110–128, 2014. (cited on Page 23)

[ABB+14] Constantinos Antoniou, Jaume Barcelo, Mark Brackstone, Hilmi Celikoglu, Biagio Ciuffo, Vincenzo Punzo, Pete Sykes, Tomer Toledo, Peter Vortisch, and Peter Wagner. Traffic simulation: Case for guidelines. JRC885, 02 2014. (cited on Page 9 and 14)

[Adb13] Peter Adby. *Introduction to optimization methods*. Springer Science & Business Media, 2013. (cited on Page 19)

[Asb22] Lennart Asbach. *Valide Testfallerzeugung aus Simulationsdaten*. PhD thesis, 2022. (unpublished). (cited on Page 41)

[AWS14] Harald Altinger, Franz Wotawa, and Markus Schurius. Testing methods used in the automotive industry: Results from a survey. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, pages 1–6, 2014. (cited on Page 2)

[BABA+07] Ramachandran Balakrishna, Constantinos Antoniou, Moshe Ben-Akiva, Haris N. Koutsopoulos, and Yang Wen. Calibration of microscopic traffic simulation models: Methods and application. *Transportation Research Record: Journal of the Transportation Research Board*, 1999(1):198–207, 2007. (cited on Page 74)

[BB06] Valentina E Balas and Marius M Balas. Driver assisting by inverse time to collision. In *2006 World Automation Congress*, pages 1–6. IEEE, 2006. (cited on Page 24)

[BBEK11] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011. (cited on Page 7 and 9)

[BD20] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020. (cited on Page 39)

[BJEZB13] K Bentaleb, K Jetto, H Ez-Zahraouy, and A Benyoussef. A cellular automata traffic flow modeling of desired speed variability. *Chinese Physics B*, 22(1):018902, 2013. (cited on Page 21)

[BMKM18] G. Bagschik, T. Menzel, C. Körner, and M. Maurer. Wissensbasierte szenariengenerierung für betriebsszenarien auf deutschen autobahnen. In *Workshop Fahrerassistenzsysteme und automatisiertes Fahren. Bd*, page 12, 2018. (cited on Page 1, 4, 22, 47, and 73)

[BMM18] Gerrit Bagschik, Till Menzel, and Markus Maurer. Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820, June 2018. (cited on Page vii, 5, and 77)

[CJSM21] Hao Cheng, Fatema T. Johora, Monika Sester, and Jörg P. Müller. Trajectory modelling in shared spaces: Expert-based vs. deep learning approach? In Samarth Swarup and Bastin Tony Roy Savarimuthu, editors, *Multi-Agent-Based Simulation XXI*, pages 13–27, Cham, 2021. Springer International Publishing. (cited on Page 15)

[CLHG10] Chenyi Chen, Li Li, Jianming Hu, and Chenyao Geng. Calibration of mitsim and idm car-following model based on ngsim trajectory datasets. In *Proceedings of 2010 IEEE International Conference on Vehicular Electronics and Safety*, pages 48–53, 2010. (cited on Page 39 and 74)

[CLOR03a] Lianyu Chu, H. X. Liu, Jun-Seok Oh, and W. Recker. A calibration procedure for microscopic traffic simulation. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, pages 1574–1579. IEEE, 12-15 Oct. 2003. (cited on Page 15)

[CLOR03b] Lianyu Chu, H.X. Liu, Jun-Seok Oh, and W. Recker. A calibration procedure for microscopic traffic simulation. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, volume 2, pages 1574–1579 vol.2, 2003. (cited on Page 9)

[CNvMH99] BA Carreras, DE Newman, B Ph van Milligen, and C Hidalgo. Long-range time dependence in the cross-correlation function. *Physics of Plasmas*, 6(2):485–494, 1999. (cited on Page 26)

[CP14] Biagio Ciuffo and Vincenzo Punzo. “no free lunch” theorems applied to the calibration of traffic simulation models. *IEEE Transactions on Intelligent Transportation Systems*, 15(2):553–562, April 2014. (cited on Page 18)

[dGP17] Erwin de Gelder and Jan-Pieter Paardekooper. Assessment of automated driving systems using real-life scenarios. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 589–594, 2017. (cited on Page 4)

[Diw20] Urmila M Diwekar. *Introduction to applied optimization*, volume 22. Springer Nature, 2020. (cited on Page 18)

[DLR22] DLR. www.testfeld-niedersachsen.de. Unpublished., 2022. (cited on Page 5)

[Erd15] Jakob Erdmann. Sumo's lane-changing model. In *Modeling Mobility with Open Data*, pages 105–123. Springer, 2015. (cited on Page 9)

[EUA+19] Ahmetcan Erdogan, Burak Ugranli, Erkan Adali, Ali Sentas, Eren Mungan, Emre Kaplan, and Andrea Leitner. Real-world maneuver extraction for autonomous vehicle validation: A comparative study. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 267–272. IEEE, 2019. (cited on Page 25)

[EW83] Leonard Evans and Paul Wasielewski. Risky driving related to driver and vehicle characteristics. *Accident Analysis & Prevention*, 15(2):121–136, 1983. (cited on Page 53)

[FHKO17] Adrian Fazekas, Friederike Hennecke, Eszter Kalló, and Markus Oeser. A novel surrogate safety indicator based on constant initial acceleration and reaction time assumption. *Journal of advanced transportation*, 2017, 2017. (cited on Page 24)

[FHW07] Department of Transportation. FHWA, US. Ngsim - next generation simulation, 2007. (cited on Page 5 and 74)

[FKBN18] Christina Flitsch, Karl-Heinz Kastner, Károly Bósa, and Matthias Neubauer. Calibrating traffic simulation models in sumo based upon diverse historical real-time traffic data – lessons learned in its upper austria. EPiC Series in Engineering, pages 25–26. EasyChair, 2018. (cited on Page 74)

[FKP+20] Daniel J. Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A. Seshia, Atul Acharya, Xantha Bruso, Paul Wells, Steve Lemke, Qiang Lu, and Shalin Mehta. Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 20.09.2020 - 23.09.2020. (cited on Page 1)

[F.R01] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. (cited on Page 39)

[G+] Sean Gillies et al. Shapely: manipulation and analysis of geometric objects, 2007–. (cited on Page 36)

[GEO21] GEOS contributors. *GEOS coordinate transformation software library*. Open Source Geospatial Foundation, 2021. (cited on Page 36)

[Ger20] Florian Gerber. florafauna/optimparallel-python v0.1.1, June 2020. (cited on Page 39)

[HA14] Bryan Higgs and Montasir Abbas. Segmentation and clustering of car-following behavior: Recognition of driving patterns. *IEEE Transactions on Intelligent Transportation Systems*, 16(1):81–90, 2014. (cited on Page xi, 15, 42, and 83)

[HAB+15] David K Hale, Constantinos Antoniou, Mark Brackstone, Dimitra Michalaka, Ana T Moreno, and Kavita Parikh. Optimization-based assisted calibration of traffic simulation models. *Transportation Research Part C: Emerging Technologies*, 55:100–115, 2015. (cited on Page 15)

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. (cited on Page 35, 36, 37, 38, 42, and 45)

[HPS+19] Lukas Hartjen, Robin Philipp, Fabian Schuldt, Bernhard Friedrich, and Falk Howar. Classification of driving maneuvers in urban traffic for parametrization of test scenarios. In *9. Tagung Automatisiertes Fahren*, 2019. (cited on Page 25)

[HSR+17] Dwayne Henclewood, Wonho Suh, Michael O Rodgers, Richard Fujimoto, and Michael P Hunter. A calibration procedure for increasing the accuracy of microscopic traffic simulation models. *SIMULATION*, 93(1):35–47, 2017. (cited on Page 18 and 48)

[HWLZ16] WuLing Huang, Kunfeng Wang, Yisheng Lv, and FengHua Zhu. Autonomous vehicles testing methods review. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 163–168. IEEE, 01.11.2016 - 04.11.2016. (cited on Page 1)

[JdBF+20] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Adrian Garcia Badaracco, Carson Farmer, Geir Arne Hjelle, Alan D. Snow, Micah Cochran, Sean Gillies, Lucas Culbertson, Matt Bartos, Nick Eubank, maxalbert, Aleksey Bilogur, Sergio Rey, Christopher Ren, Dani Arribas-Bel, Leah Wasser, Levi John Wolf, Martin Journois, Joshua Wilson, Adam Greenhall, Chris Holdgraf, Filipe, and François Leblanc. geopandas/geopandas: v0.8.1, July 2020. (cited on Page 34)

[JSW17] Philipp Junietz, Jan Schneider, and Hermann Winner. Metrik zur bewertung der kritikalität von verkehrssituationen und-szenarien. In *Workshop Fahrerassistenz und automatisiertes Fahren*, 2017. (cited on Page 23)

[Kan06] Gopal K Kanji. *100 statistical tests*. Sage, 2006. (cited on Page 28)

[KBG+16] Tamás Kovács, Kálmán Bolla, Rafael Alvarez Gil, Edit Csizmás, Csaba Fábián, Lóránt Kovács, Krisztián Medgyes, József Osztényi, and Attila Végh. Parameters of the intelligent driver model in signalized intersections. *Tehnicki vjesnik/Technical Gazette*, 23(5), 2016. (cited on Page 53)

[KBKE18] Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2118–2125, Nov 2018. (cited on Page 5)

[KGH06] Joonho Ko, Randall Guensler, and Michael Hunter. Variability in traffic flow quality experienced by drivers: Evidence from instrumented vehicles. *Transportation Research Record*, 1988:1–9, 2006. (cited on Page 83)

[KLY+21] Arpan Kusari, Pei Li, Hanzhi Yang, Nikhil Punshi, Mich Rasulis, Scott Bogard, and David J LeBlanc. Enhancing sumo simulator for simulation based testing and validation of autonomous vehicles. *arXiv preprint arXiv:2109.11620*, 2021. (cited on Page 15)

[KMKL18] Frank Köster, Jens Mazzega, and Sascha Knake-Langhorst. Automatisierte und vernetzte systeme effizient erprobt und evaluiert. *ATZextra*, 23(5):26–29, 2018. (cited on Page 2, 5, and 12)

[Kra98] Stefan Krauss. Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics, Apr 1998. (cited on Page 6 and 9)

[KT08a] Arne Kesting and Martin Treiber. Calibrating car-following models by using trajectory data: Methodological study. *Transportation Research Record*, 2088(1):148–156, 2008. (cited on Page 15)

[KT08b] Arne Kesting and Martin Treiber. Calibrating car-following models by using trajectory data methodological study. *Transportation Research Record Journal of the Transportation Research Board*, 2088:148–156, 03 2008. (cited on Page 74)

[LBSB13] David J LeBlanc, Shan Bao, James R Sayer, and Scott Bogard. Longitudinal driving behavior with integrated crash-warning system: Evaluation from naturalistic driving data. *Transportation research record*, 2365(1):17–21, 2013. (cited on Page 25)

[LFAR19] Gonçalo Leao, Joao Ferreira, Pedro Amaro, and Rosaldo JF Rossetti. Using simulation games for traffic model calibration. *ISC'2019*, page 131, 2019. (cited on Page 15)

[LHP+21] Marcel Langer, Michael Harth, Lena Preitschaft, Ronald Kates, and Klaus Bogenberger. Calibration and assessment of urban microscopic traffic simulation as an environment for testing of automated driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3210–3216. IEEE, 2021. (cited on Page 15)

[LWB+18] Pablo Alvarez Lopez, Evamarie Wiessner, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flotterod, Robert Hilbrich, Leonhard Lucken, Johannes Rummel, and Peter Wagner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582. IEEE, 04.11.2018 - 07.11.2018. (cited on Page 7 and 40)

[LXABA15] Lu Lu, Yan Xu, Constantinos Antoniou, and Moshe Ben-Akiva. An enhanced spsa algorithm for the calibration of dynamic traffic assignment models. *Transportation Research Part C: Emerging Technologies*, 51:149–166, 2015. (cited on Page 15)

[M+67] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967. (cited on Page 39)

[MA07] Xiaoliang Ma and Ingmar Andreasson. Behavior measurement, analysis, and regime classification in car following. *IEEE transactions on intelligent transportation systems*, 8(1):144–156, 2007. (cited on Page 15)

[MBWvA+20] Freddy Antony Mullakkal-Babu, Meng Wang, Bart van Arem, Barys Shyrokau, and Riender Happee. A hybrid submicroscopic-microscopic traffic flow simulation framework. *IEEE Transactions on Intelligent Transportation Systems*, 2020. (cited on Page 7)

[Mil09] David Michael Miller. *Developing a procedure to identify parameters for calibration of a VISSIM model.* PhD thesis, Georgia Institute of Technology, 2009. (cited on Page 16)

[MJ51] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951. (cited on Page 28 and 29)

[MNW+18] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX Symposium* *on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, 2018. (cited on Page 39, 43, and 44)

[NKM19] Andreas Nonnengart, Matthias Klusch, and Christian Müller. Crisgen: Constraint-based generation of critical scenarios for autonomous vehicles. In *International Symposium on Formal Methods*, pages 233–248. Springer, 2019. (cited on Page 23)

[NWH+20] Christian Neurohr, Lukas Westhofen, Tabea Henning, Thies de Graaff, Eike Mohlmann, and Eckard Bode. Fundamental considerations around scenario-based testing for automated driving. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 121–127. IEEE, 19.10.2020 - 13.11.2020. (cited on Page 1)

[OCI15] Osama A Osman, Julius Codjoe, and Sherif Ishak. Impact of time-to-collision information on driving behavior in connected vehicle environments using a driving simulator test bed. *Traffic Logist. Eng*, 3(1), 2015. (cited on Page 25)

[PBF+17] Mitra Pourabdollah, Eric Bjärkvik, Florian Fürer, Björn Lindenberg, and Klaas Burgdorf. Calibration and evaluation of car following models using real-world driving data. In *2017 IEEE 20th International conference on intelligent transportation systems (ITSC)*, pages 1–6. IEEE, 2017. (cited on Page 15 and 53)

[PD05] Sakda Panwai and Hussein Dia. Comparative evaluation of microscopic car-following behavior. *IEEE Transactions on intelligent transportation systems*, 6(3):314–325, 2005. (cited on Page 53)

[pdt20] The pandas development team. pandas-dev/pandas: Pandas, February 2020. (cited on Page 34 and 39)

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (cited on Page 39 and 43)

[PZBE17] Andreas Pütz, Adrian Zlocki, Julian Bock, and Lutz Eckstein. System validation of highly automated vehicles with a database of relevant traffic scenarios. *situations*, 1:E5, 2017. (cited on Page 1)

[RABS20] Olivér Rákos, Szilárd Aradi, Tamás Bécsi, and Zsolt Szalay. Compression of vehicle trajectories with a variational autoencoder. *Applied Sciences*, 10(19):6739, 2020. (cited on Page 77)

[RHB10] Mustafizur Rahman, Md. Rafiul Hassan, and Rajkumar Buyya. Jaccard index based availability prediction in enterprise grids. *Procedia Computer Science*, 1(1):2707–2716, 2010. ICCS 2010. (cited on Page 31)

[RRMB17] Xavier Ros-Roca, Lídia Montero, and Jaume Barceló. Notes on using simulation-optimization techniques in traffic simulation. *Transportation Research Procedia*, 27:881–888, 2017. (cited on Page 16 and 39)

[SBW+16] Christoph Sippl, Florian Bock, David Wittmann, Harald Altinger, and Reinhard German. From simulation data to test cases for fully automated driving and adas. In *IFIP International Conference on Testing Software and Systems*, pages 191–206. Springer, 2016. (cited on Page 6 and 73)

[Sch17] Fabian Schuldt. *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen*. PhD thesis, 2017. (cited on Page 1, 3, 4, and 6)

[SD15] Haitham Seada and Kalyanmoy Deb. U-nsga-iii: a unified evolutionary optimization procedure for single, multiple, and many objectives: proof-of-principle results. In *International conference on evolutionary multi-criterion optimization*, pages 34–49. Springer, 2015. (cited on Page 39)

[SKR20] Dominik Salles, Stefan Kaufmann, and Hans-Christian Reuss. Extending the intelligent driver model in sumo and verifying the drive off trajectories with aerial measurements. 2020. (cited on Page 52 and 53)

[SKvA12] Wouter J Schakel, Victor L Knoop, and Bart van Arem. Integrated lane change model with relaxation and synchronization. *Transportation Research Record*, 2316(1):47–57, 2012. (cited on Page 15)

[SNB+20] Yadavilli Sashank, Nitin A Navali, Arjuna Bhanuprakash, B Anil Kumar, and Lelitha Vanajakshi. Calibration of sumo for indian heterogeneous traffic conditions. In *Recent Advances in Traffic Engineering*, pages 199–214. Springer, 2020. (cited on Page 15)

[Spa92] J.C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992. (cited on Page 39)

[SSL+13] Fabian Schuldt, Falko Saust, Bernd Lichte, Markus Maurer, and Stephan Scholz. Effiziente systematische testgenerierung für fahrerassistenzsysteme in virtuellen umgebungen. *Automatisierungssysteme, Assistenzsysteme und Eingebettete Systeme Für Transportmittel*, 2013. (cited on Page 4 and 73)

[Tat15] Mugur Tatar. Enhancing adas test and validation with automated search for critical situations. In *Driving Simulation Conference (DSC)*, 2015. (cited on Page 1, 4, and 73)

[Tho53] Robert L Thorndike. Who belongs in the family. In *Psychometrika*. Citeseer, 1953. (cited on Page 39 and 43)

[TK15] Martin Treiber and Venkatesan Kanagaraj. Comparing numerical integration schemes for time-continuous car-following models. *Physica A: Statistical Mechanics and its Applications*, 419:183–195, 2015. (cited on Page 41)

[TKD+03] Tomer Toledo, Haris N. Koutsopoulos, Angus Davol, Moshe E. Ben-Akiva, Wilco Burghout, Ingmar Andréasson, Tobias Johansson, and Christen Lundin. Calibration and validation of microscopic traffic simulation tools: Stockholm case study. *Transportation Research Record: Journal of the Transportation Research Board*, 1831(1):65–75, 2003. (cited on Page 74)

[UMR+15] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, Sep. 2015. (cited on Page 4)

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. (cited on Page 39)

[VHSK+15] CPIJ Van Hinsbergen, WJ Schakel, VL Knoop, JWC van Lint, and SP Hoogendoorn. A general framework for calibrating and comparing car-following models. *Transportmetrica A: Transport Science*, 11(5):420–440, 2015. (cited on Page 53)

[VNS+14] Luís Vasconcelos, Luís Neto, Sílvia Santos, Ana Bastos Silva, and Álvaro Seco. Calibration of the gipps car-following model using trajectory data. *Transportation Research Procedia*, 3:952–961, 2014. (cited on Page 74)

[Vog03] Katja Vogel. A comparison of headway and time to collision as safety indicators. *Accident analysis & prevention*, 35(3):427–433, 2003. (cited on Page 23)

[Voh16] Deepak Vohra. Apache parquet. In *Practical Hadoop Ecosystem*, pages 325–335. Springer, 2016. (cited on Page 35)

[WBM03] Jingxian Wu, M Brackstone, and M McDonald. The validation of a microscopic simulation model: a methodological case study. *Trans-*

portation Research Part C: Emerging Technologies, 11(6):463–479, 2003. (cited on Page 53)

[WMR+13] Ricardo Wagner, José Antonio Fernandes de Macedo, Alessandra Raffaetà, Chiara Renso, Alessandro Roncato, and Roberto Trasarti. Mob-warehouse: A semantic approach for mobility analysis with a trajectory data warehouse. In *International Conference on Conceptual Modeling*, pages 127–136. Springer, 2013. (cited on Page 6)

[WW15] Walther Wachenfeld and Hermann Winner. Die freigabe des autonomen fahrens. In *Autonomes Fahren*, pages 439–464. Springer Vieweg, Berlin, Heidelberg, 2015. (cited on Page 1)

[WW16] Walther Wachenfeld and Hermann Winner. The release of autonomous vehicles. In *Autonomous driving*, pages 425–449. Springer, 2016. (cited on Page 1 and 3)

[XFX20] Zhang Xinxin, Li Fei, and Wu Xiangbin. Csg: Critical scenario generation from real traffic accidents. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1330–1336. IEEE, 2020. (cited on Page 23)

[YD12] Liu Yan and Wang Dianhai. Minimum time headway model by using safety space headway. In *World Automation Congress 2012*, pages 1–4. IEEE, 2012. (cited on Page 24)

[YK96] QI Yang and Haris N. Koutsopoulos. A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research Part C: Emerging Technologies*, 4(3):113–129, 1996. (cited on Page 40)

[YLW+14] Wen Yao, Yubin Lin, Chao Wang, Huijing Zhao, and Hongbin Zha. Automatic lane change data extraction from car data sequence. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1894–1895, 2014. (cited on Page 4)

[YSWL20] Bingjian Yue, Shuming Shi, Shuo Wang, and Nan Lin. Low-cost urban test scenario generation using microscopic traffic simulation. *IEEE Access*, 8:123398–123407, 2020. (cited on Page 2 and 74)

[ZBLN97] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, dec 1997. (cited on Page 39)

[ZdR17] Jinwei Zhou and Luigi del Re. Reduced complexity safety testing for adas & adf. *IFAC-PapersOnLine*, 50(1):5985–5990, 2017. (cited on Page 4 and 73)

[ZHP+17] Ding Zhao, Xianan Huang, Huei Peng, Henry Lam, and David J LeBlanc. Accelerated evaluation of automated vehicles in car-following maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):733–744, 2017. (cited on Page 25)

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

Braunschweig, den 18. April 2022