HAW HAMBURG

BACHELORTHESIS
Benedikt Buhk

Development of a method for energy efficiency optimization of an autonomous, electric vehicle on the test track for automated and connected driving in Hamburg

FAKULTY OF COMPUTER SCIENCE AND ENGINEERING
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Benedikt Buhk

Development of a method for energy efficiency optimization of an autonomous, electric vehicle on the test track for automated and connected driving in Hamburg

Bachelorthesis based on the study regulations for the study programme *Bachelor of Science Elektro- und Informationstechnik* at the Department of Information and Electrical Engineering of the Faculty of Computer Science and Engineering of the Hamburg University of Applied Sciences

Supervising examiner: Prof. Dr. Rasmus Rettig
Second examiner: Prof. Dr. Pawel Buczek

Day of delivery: 10. Juni 2021

Benedikt Buhk

Title of Thesis

Development of a method for energy efficiency optimization of an autonomous, electric vehicle on the test track for automated and connected driving in Hamburg

Keywords

energy efficiency, driving algorithms, electric vehicle, traffic simulation, forward-facing energy model

Abstract

This study introduces a simulation model that couples a forward facing energy model with a microscopic traffic simulator for the analysis of the energy performance of connected and autonomous vehicles (CAVs). The developed model is validated against measurement data from driving cycles of an Electric Vehicle (EV). The applicapability of the model is demonstrated by optimizing a driving algorithm on energy efficiency.

Benedikt Buhk

Thema der Arbeit

Entwicklung einer Methode zur Optimierung der Energieeffizienz eines autonomen Elektrofahrzeugs auf der Teststrecke für automatisiertes und vernetztes Fahren in Hamburg

Stichworte

Energieeffizienz, Fahralgorithmen, Elektrofahrzeug, Vehrkehrssimulation, vorwärts-gerichtetes Energie Model

Kurzzusammenfassung

Diese Studie schlägt ein Simulationsmodel zur Analyse der Energieeffizienz von autonomen Fahrzeugen vor, welches ein vorwärts gerichtetes Energiemodel mit einer Vehrkehrssimulationssoftware koppelt. Das entwickelte Model wird anhand von Messdaten mehrerer Fahrzyklen eines Elektrofahrzeuges validiert. Die Anwendbarkeit des Models wird mit der Optimierung eines Fahralgorithmus demonstriert.

Contents

List of Figures

List of Tables

Acronyms

AC air conditioning.

ADVISOR Advanced Vehicle Simulator.

API application programming interface.

AWD all-wheel-drive.

CAN controller area network.

CAV connected and autonomous vehicle.

CSV comma seperated value.

DLR Deutsches Zentrum für Luft- und Raumfahrt.

EPA Environmental Protection Agency.

EPL Eclipse Public License.

EV Electric Vehicle.

EVRA Electric Vehicle Reference Application.

GEH Geoffrey E. Havers.

GHG greenhouse gas.

GUI graphical user interface.

HAW Hochschule für Angewandte Wissenschaften.

HIL Hardware In the Loop.

ITS intelligent transport systems.

LSA Lichtsignalanlage.

NEDC New European Driving Cycle.

OSM OpenStreetMap.

RNG random number generator.

SoC State of Charge.

SUMO Simulation of Urban MObility.

TAVF Teststrecke für automatisiertes und vernetztes Fahren.

TCP Transmission Control Protocol.

TraCI Traffic Control Interface.

TRANSIMS TRansportation ANalysis SIMulation System.

UML Urban Mobility Lab.

VISSIM Verkehr In Städten - SImulationsModell.

VT-CPEM Virginia Tech Comprehensive Power-based EV Energy consumption Model.

1 Introduction

1.1 Publication note

Parts of this study have been submitted to the SUMO User Conference 2021 [9] to be published in [2].

1.2 Motivation

Today, the transport sector causes about 27 % of greenhouse gas (GHG) emissions in the EU and has not improved its sustainability in recent years [5]. Therefore, solutions for sustainable mobility are of compelling importance to meet the Paris agreement and prevent global warming to rise above 1.5 degrees compared to pre-industrial temperatures.

In the controversial debate about such solutions, automated and connected vehicles (CAVs) in combination with an electrical drivetrain are often presented as a promising technology to increase the energy efficiency of the transport sector. However, such an increase in energy efficiency under urban traffic conditions is very difficult to estimate. Therefore, a tool is needed that can analyze the energy consumption of CAVs under realistic traffic conditions. Such a tool could also be used to optimize a given driving algorithm in its energy efficiency.

1.3 Literature Review

In literature, some studies that analyze and optimize the energy consumption of CAVs with simulation tools can be found. This section summarizes these studies and commonly

used simulation tools to point out shortcomings for the evaluation of a driving algorithm's energy performance.

To generate realistic traffic scenarios and analyze them down to a single vehicle, microscopic traffic simulation tools are already widely used today. These tools such as VISSIM [27], TRANSIMS [28], aimsun [1] or SUMO [21], simulate complex vehicle interactions on a microscopic level. They are used in the area of transportation planning and traffic engineering for a large range of purposes from demand and supply analysis over the test of vehicle routing methods to detailed vehicle interaction simulations. To calculate the energy consumption of a vehicle, microscopic traffic simulators are usually based on backward-facing energy models. Those backward-facing energy models such as the VT-CPEM proposed by Fiori et al. [6] or the one of T. Kurczveil et al. [19] implemented in SUMO can deliver instantaneous energy consumption estimations requiring only a few vehicle-specific parameters. They remain computationally efficient by calculating the consumed energy based on simple equations from the current speed "backward" to the energy source. However, forward-facing models such as ADVISOR [13] can deliver more accurate instantaneous energy consumption estimations due to more accurate vehicle parametrization for the cost of being computationally more complex. They also take the speed-over-time data of a driving cycle as input. This speed is then used as a reference for a driver model which controls the torque request to the engine. From this torque request, the drive train is calculated "forward" to the wheels resulting in an actual speed of the vehicle. Since all of the vehicle aggregates and the drivetrain components can be simulated, the consumed energy to achieve the actual speed is calculated in a causal way. Therefore, forward-facing models are the better choice when it comes to a detailed analysis of a vehicle's energy consumption.

R. Galvin [7] analyzed the influence of different speed and acceleration values on the energy consumption over a short driving scenario without traffic for eight commonly-used electric vehicles (EVs). One of the key findings was the significant reduction of energy efficiency with modest to high acceleration. To reduce the energy consumption of CAVs different methods are proposed in the literature. They can mainly be divided into two groups: eco-routing approaches, and eco-driving approaches. Eco-routing focuses on the energy-efficient routing of one or more vehicles. For example, the traffic light control can be optimized on the energy consumption of all vehicles on the road as Luin et al. [22] demonstrated. In their study, they used SUMO together with a backward-facing energy model to verify their results. In this case, a more accurate, forward-facing model would be unfavorable since its computational complexity would result in very long simulation

times when calculating the energy consumption for a whole vehicle fleet. Eco-driving approaches focus on the optimal speed for a single vehicle. For the evaluation of the energy performance of a driving algorithm for CAVs, J. Han et al. [13] proposed an eco-driving control system for energy-optimal acceleration and deceleration and simulated the performance. Their simulation was based on the assumption, that the preceding vehicle stays the same and there is no other vehicle joining the gap between the controlled vehicle and its leader over a whole driving cycle. However, this assumption is usually not fulfilled in real urban traffic.

To meet the shortcomings of existing simulation methods and analyze the energy performance of CAVs under realistic traffic scenarios, the coupling of a microscopic traffic simulator with a forward-facing energy model could be a promising solution.

1.4 Solution Approach

This study is introducing a simulation-based method for the analysis of the energy efficiency of a driving algorithm for a certain EV. The presented method is coupling the open-source microscopic traffic simulator SUMO (Simulator for Urban Mobility) [21] with a highly accurate, forward-facing energy model implemented in Simulink [23]. SUMO comes with an application programming interface called Traffic Control Interface (TraCI) [32] which can be used to control certain vehicles inside the simulation with an external driving algorithm. Therefore, a detailed analysis of the energy performance of CAVs becomes possible. With the coupled model lots of realistic traffic scenarios are generated with SUMO and the performance of different driving algorithms and parameters can be analyzed and optimized. With the forward-facing energy model also power-train optimizations of the considered vehicle would be possible for the cycles generated with the driving algorithm.

To validate the Simulation results, this study also presents the speed and State of Charge (SoC) data collected from a test vehicle in real urban traffic. Therefore, a 2017 Tesla Model S [36] is driven three cycles around the test track for automated and connected driving (TAVF) in Hamburg [11]. A SUMO model of the same road network, the TAVF, is used together with route files representing realistic traffic demand on those streets. With this model, driving cycles for the test vehicle following the same route as in the measurements are simulated. Thus, the test vehicle is controlled by a driving algorithm

and the resulting cycles are analyzed for the vehicle's energy consumption and compared to the consumption of the real cycles measured in this study.

The simulation model developed in this study can be used in a very flexible way to analyze and optimize the energy efficiency of CAVs. Various driving algorithms can be implemented, the energy model can be parameterized for any vehicle, and the traffic scenarios, in which the behavior and energy consumption of a certain vehicle with a certain driving algorithm is analyzed, can be changed to any scenario(s) of interest.

2 Principles of the used Technologies

In this chapter, the main principles of the technologies used for the data collection and the developed simulation model in this study are presented. First, the CAN-bus is explained briefly, which was logged in a test vehicle to collect relevant data for this study. The main specifications and features of the test vehicle, a Tesla Model S, are explained as well as the test track for automated and connected driving in Hamburg (TAVF), that was driven during the data collection.

For the coupled simulation model both the used energy model, an extended version of the EVRA, and the used microscopic traffic simulator (SUMO), are presented.

2.1 CAN-bus

A controller area network (CAN) is a communication network that was designed to interconnect vehicle components and to save wiring. Instead of using two wires for every signal as it was common in vehicles before the CAN-bus was developed, it allows multiple systems to exchange messages over the same two wires (CAN HI and CAN LO in Figure 2.1). The difference between the voltage on CAN HI and CAN LO is interpreted as a '1' if it is greater than 2 V. Otherwise the current bit is interpreted as a '0'. The voltage on both bus wires is changed to switch between ones and zeros. Therefore, a short signal edge is achieved allowing speeds up to 1 Mbps. CAN is a field bus and is widely used in vehicles nowadays. [20]

The messages that are exchanged on the bus have a data frame shown in Figure 2.1. They start with a unique 11-bit identifier (indicated in green) which represents the priority. In the extended frame format, the identifier has 29-bit. The messages can include up to 8 Bytes (64-bit) of data (indicated in red). The length of the data in Bytes is stored in the 4 bits (indicated in yellow).

The data is generally broadcasted on the bus so that every system can filter out the relevant information from the bus using the identifier.

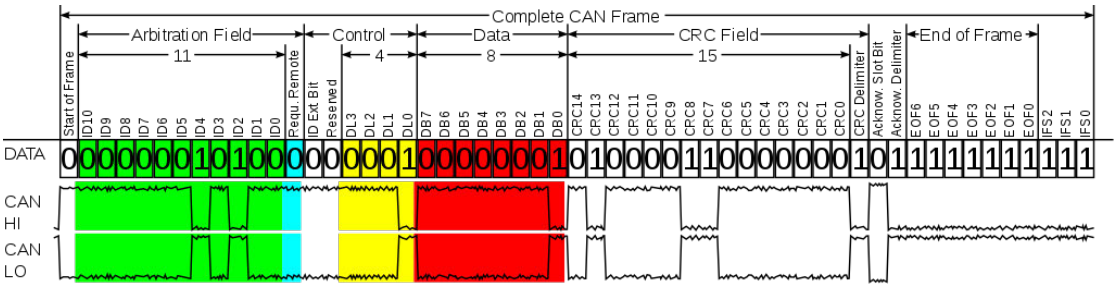


Figure 2.1: Data frame of a CAN message [33]

The subsystems of the Tesla Model S exchange information over four different highspeed CAN-busses with a data rate of 500 kbps and use the extended frame format. On CAN3 information about the powertrain is transmitted. The current SoC of the battery pack is published under the identifier 0x302 with a frequency of 1 Hz and the current speed under the identifier 0x256 with a frequency of 10 Hz. In this study, the SoC and speed was logged from CAN3 as described in chapter 3.

2.2 Tesla Model S 75D 2017

The 2017 Model S [36] in the 75D version is an EV produced by the car manufacturing company Tesla. It has a 75 kWh battery. In this study, the Model S 75D of the Urban Mobility Lab (UML) of the Hamburg University of Applied Sciences (HAW Hamburg) serves as a test vehicle for the experimental data collection (see chapter 3). The main characteristics of the Model are presented in Table 2.1.

Battery Capacity	75 kWh
Power	386 kW (518 hp)
Torque	441 Nm
Drivetrain	AWD
0 - 100 km/h	4,4 s
Range (EPA [34])	417 km
Range (NEDC [35])	490 km
Top Speed	225 km/h
Year	2016 - 2019

Table 2.1: Tesla Model S 75D vehicle characteristics [36]

2.3 Test track for automated and connected driving in Hamburg (TAVF)

The test track for automated and connected driving (Teststrecke für automatisiertes und vernetztes Fahren (TAVF)) [11] is a route in the middle of the city Hamburg in Germany. It consists of about 8.5 km street distance with a total of 37 traffic lights [11]. The test track is not separated from the regular city traffic. Its streets represent real urban traffic in a speed limit zone of 50 km/h and they all have between two or three lanes. Some traffic lights on the track are equipped with IEEE802.11p Road Side Units (RSUs) [24]. The TAVF is a project that aims to test intelligent transport systems (ITS) in an open collaboration of various users and contributors under real traffic conditions.

2.4 Used Energy Model: EVRA

The used energy model in this study is an extended version of the Electric Vehicle Reference Application (EVRA) [29] that was adapted in a previous student work [25]. The EVRA is a forward-facing energy model. It represents a full EV model including motor-generator, battery, direct-drive transmission as well as associated powertrain control algorithms and can serve for powertrain optimizations, component selection, diagnostic algorithm design, and Hardware In the Loop (HIL) testing [29].

The model takes a driving cycle in the form of a two-dimensional vector as an input. In the first column, the time in seconds is passed and the second column contains the corresponding speed in km/h. This driving cycle with one speed value per second is

seen as the target speed cycle. The car model calculates a torque request based on the actual speed and the target speed. From this torque request, the whole drive train is calculated in a forward and causal way, resulting in the actual speed of the vehicle. Also, the atmospheric temperature is considered by the model and taken as an input of the battery model since it has a significant influence on the battery performance.

In [25] the model was extended to take the energy consumption of the auxiliary systems air conditioning (AC), ventilation, headlights, infotainment (board computer with touch-screen and speakers), and seat heating into account. For these systems, the extended model takes the percentage of use of the auxiliary systems over time as an input. Those percentages result in factors that are multiplied with the maximum current measured for the respective system in the test vehicle. The model was also parameterized to the 2017 Model S 75D of the UML of the (HAW) in [25]. The maximum current of the auxiliary systems was measured to be for the AC 6 A, the ventilation 2 A, the seat heating 0.6 A, the headlights 0.4 A, and the infotainment systems 0.2 A. The AC and the ventilation are supplied from a 400 V system meanwhile the rest of the considered systems run with a 12 V power supply ([25]).

In this study, this extended EVRA model which was adapted and parameterized for the test vehicle (2017 Model S 75D) in [25] is used to calculate the continuous SoC value for driving cycles of the test vehicle that were generated in SUMO.

2.5 Microscopic Traffic Simulator SUMO

There are multiple microscopic traffic simulators in the market. In the commercial sector there are VISSIM [27] and aimsun [1]. TRANSIMS [28] and SUMO [21] are open source simulators. In this study, SUMO was chosen to serve as a tool to generat realistic driving for a CAV due to its open availability, its high portability and its application programming interface (API) TraCI that allows on-line connection to other applications during a simulation.

SUMO was mainly developed by the German Aerospace Center (DLR) [8] under the EPL 2.0 [4] license. The open source software comes with an extensive documentation that can be accessed online [10].

The simulation package can run continuous traffic scenarios on large networks with vehicles from different classes and also considers pedestrians. During a simulation, the

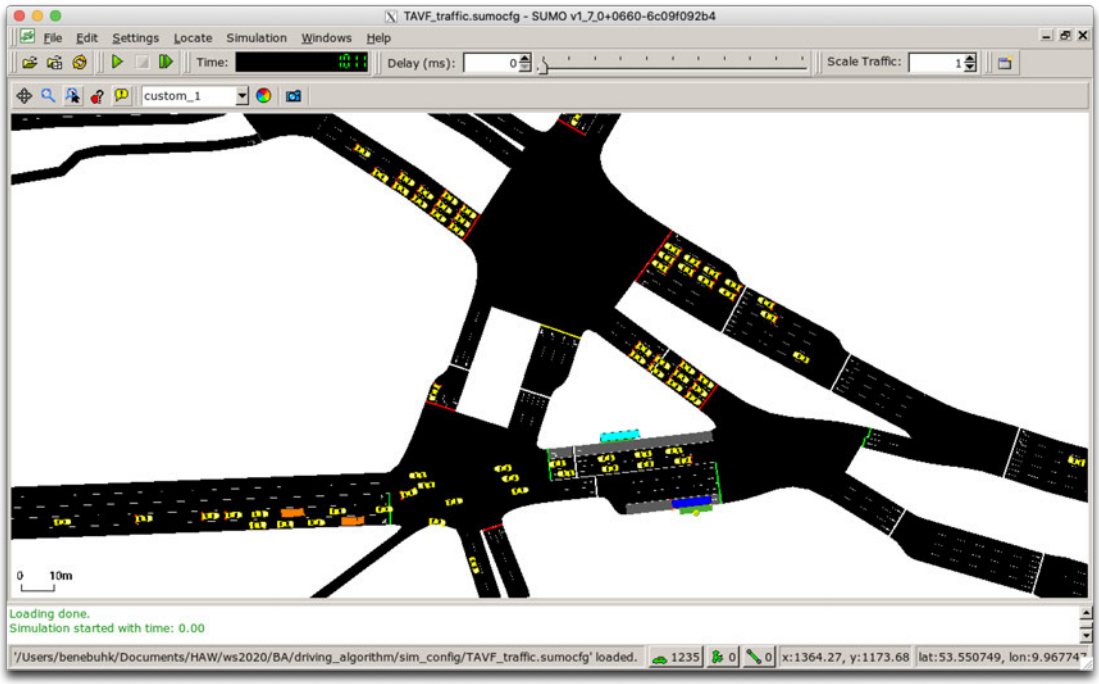


Figure 2.2: Screenshot of a traffic simulation with the SUMO-GUI [21]

network state including the position of every vehicle, their speed, and more are calculated for every simulation step. It is open to a large range of input data and comes with many tools that extend its connectivity to other software and data formats. To interact with the simulation during a scenario the Traffic Control Interface (TraCI) [32] is implemented which can start SUMO as a server and connect to it as a client with a TCP connection.

A SUMO simulation is based on a network file, which represents the street network with positions of the streets, lane numbers, and traffic lights. It can handle very large networks and comes with different tools that can create networks automatically or import them from other traffic simulation tools or OpenStreetMap (OSM) [12]. To simulate different vehicles on those streets, they have to be defined in route files containing the type of each vehicle and information about its route in the street network. The vehicle type defines features of the vehicle such as its identifier, class, color, length, maximum speed, maximum acceleration and deceleration, speed factor, driving imperfection value sigma, and others. The speed factor is multiplied by the speed limit of the street the vehicle is driving on. The driver imperfection value sigma is between 0 and 1.0. It indicates random fluctuations of the speed of a vehicle. With a sigma of 0, the vehicle will drive

continuously at the speed limit of a street if no preceding vehicles or traffic lights are forcing it to brake.

To run a simulation a SUMO configuration file has to be written or generated. This .sumocfg-file defines all network, route, and additional files as well as simulation parameters such as simulation step length, begin and end time, the integration method, or the seed value for the random number generator (RNG). Also, the outputs that the simulation will generate are defined in the configuration file. A .sumocfg-file can be executed in the SUMO-GUI or in a terminal. In the GUI, the street network with all the vehicles can be seen during the simulation. The speed of the simulation can be adjusted and the simulation can be paused.

The used network files of the TAVF and the route files for the traffic used in this study are described in more detail in section 4.3.

3 Experimental Data Collection

For this study, the speed and SoC data of three real driving cycles were collected from a test vehicle. This chapter describes the setup and conditions under which the measurements were taken and presents the collected data.

First, the driven route is presented and the external influences as well as the auxiliary system use is explained for the three cycles, respectively. Then, the setup for the measurements in the vehicle is described. Finally, the collected speed and SoC data from the three cycles are presented.

3.1 Experimental Setup

3.1.1 Route

For the data collection, three driving cycles were measured: Cycle 1, Cycle 2, and Cycle 3. These measured cycles will be referred to with a capital initial letter in the further course of this thesis. All of them had the same route. The test vehicle was driven one circle on the TAVF in Hamburg [11]. In Cycle 1 and Cycle 2, the starting (and ending) point was in front of the train station Hamburg Dammtor and the Cycle 3 started (and ended) at Holstenwall (next to the park Planten un Blomen). The TAVF is shown in Figure 3.2. Also, the starting points of the cycles are marked. In all three cycles, the big, counterclockwise round on the TAVF was driven over Stephansplatz and Dammtor. At the Elphilharmonie, a turn was performed to continue the route. The driver of the vehicle was only performing lane changes when they were necessary to follow the route.

The TAVF is located in the center of Hamburg and represents real urban traffic. On the route there are 37 traffic lights, the streets have between 2 and 3 lanes and the speed limit is 50 km/h.

Figure 3.1: The TAVF with the starting points of Cycle 1, 2, and 3 [30]

3.1.2 External Influences

The energy consumption for a cycle on the TAVF has many external influences that do not depend on the driving behavior of the car such as traffic, traffic lights, and temperature. The traffic depends highly on the weekday and time. The experimental data of Cycle 1 and Cycle 2 were taken consecutively on the 16th of December 2020 between 14:00 and 15:00. The ambient temperature was 8 degrees Celsius. During those measurements, the cabin of the car was already well-tempered to around 18 degrees Celsius. Cycle 3 was measured on the 13th of January 2021 between 15:00 and 16:00 at an ambient temperature of 4 degrees Celsius. At the beginning of this cycle, the cabin was still cold and the air conditioning systems were used to a higher extent to warm up the cabin, as shown in the next subsection 3.1.3.

3.1.3 Auxiliary Systems

The auxiliary systems with the most significant energy consumption of the test vehicle were analyzed in a previous study [25]. Those values are also taken into account in the energy model (section 4.4) with a factor representing the percentage of use of each

of those systems. Table 3.1 shows those factors for each system in the three cycles approximately.

Note, that the AC and ventilation were set to a higher power during the measurements of Cycle 3 compared to the measurements of Cycle 1 and 2.

Measurement	AC	Ventilation	Seat Heater	Headlights	Infotainment
Cycle 1	0.3	0.3	0	1.0	0.1
Cycle 2	0.3	0.3	0	1.0	0.1
Cycle 3	0.8	0.8	0	1.0	0.1

Table 3.1: Use factor of auxiliary systems during the measurements of Cycle 1 to 3

3.1.4 In-Vehicle Setup

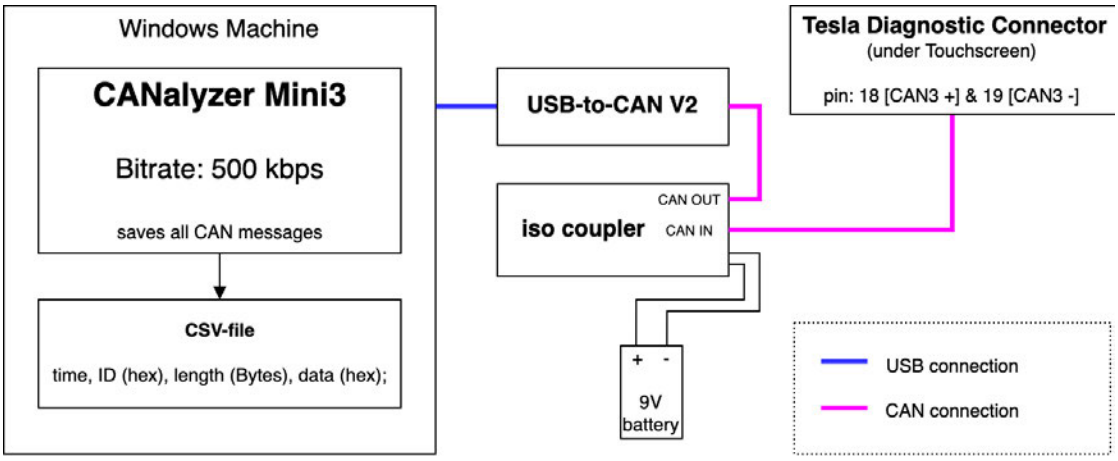


Figure 3.2: Setup inside the vehicle during the measurements on the TAVF

The used test vehicle is a 2017 Tesla Model S 75D [36]. The internal systems of the car communicate over a Controler Area Network (CAN) bus. On the CAN3 bus, data regarding the powertrain is shared. This CAN3 bus can be accessed over pin 18 and 19 of the Tesla Diagnostic Connector under the Touchscreen of the middle console. During the driving cycles on the TAVF, the bus messages are read and logged over the USB-CAN interface 'USB-to-CAN V2' of the brand Ixxat [15] using the software CANalyzer Mini3 [14]. To ensure that no signals are disturbed and nothing can be sent on the bus, a certified CAN-bus iso coupler, the SAM-CAN-ISO011 [17], is used for the connection. It connects the CAN interface with galvanic isolation to the vehicle bus network.

All messages from the CAN3 are logged in a CSV file. The speed data is published on the bus with the identifier 0x256 with a frequency of 10Hz. The SoC-value has the identifier 0x302 and is published with a frequency of 1 Hz. After the measurements, the logged CAN messages are exported in a CSV file containing the time, the identifier, the length of the data in bytes, and the data bits in hexadecimal for each message.

3.2 Data Processing

For each of the three measured cycles, one CSV file is obtained from the measurements. These files contain all messages that were published on the vehicles CAN3 bus during the measurements. Since there are usually more than 2000 messages per second published on CAN, the files of the cycle measurements have more than 2 million lines.

An existing python script that was developed in the UML was executed on the files to create one file for each identifier containing all the messages that were published under this identifier. For the identifier 0x0256 and 0x0302 the speed and SoC values were encoded after [16] automatically by the provided script.

In Cycle 1 and Cycle 2 the CAN3 log was started on the way to the TAVF and continued after the route was finished. The times of the route start and end were noted and the corresponding period was cut out of the measurement data. For Cycle 3 the log was started in the starting position of the route (a parking lot at the side of the TAVF) and stopped when passing that same point at the end of the route. Therefore, the whole data of the log file represents the whole cycle.

The obtained data contains for each cycle speed values with a frequency of 10 Hz and SoC values with a frequency of 1 Hz. The distance of each cycle was calculated by calculating the distance traveled in 0.1 s with every speed value and accumulating these values to get the distance of the whole cycle. For the input of the energy model, a two-dimensional array was created containing integer values for the time of the cycle in seconds in the first column and the corresponding speed at this second in the second column. For the speed value, the average over the 10 measured speed values for each second was taken.

3.3 Measurement Results

The collected speed and SoC data for the three cycles are shown in Figure 3.3. The duration in seconds, the distance in meter, the average speed in km/h and the difference in the SoC in percent of the battery charging state over the whole cycle are given in Table 3.2 for each cylce respectively:

Measurement	duration	distance	average speed	Δ SoC
Cycle 1	1442 s	8491.4 m	21.2 km/h	2.2 %
Cycle 2	1406 s	8605.2 m	22.03 km/h	2.3 %
Cycle 3	1462 s	8240.0 m	20.29 km/h	2.7 %

Table 3.2: Key values of the measurements of three TAVF cycles with human driver

The battery of the test vehicle has a capacity of 75 kWh. The energy consumption of the three cycles is shown in Table 3.3.

Measurement	Δ SoC	consumption full cycle	consumption per km
Cycle 1	2.2 %	1650 Wh	194,31 Wh/km
Cycle 2	2.3 %	1725 Wh	200,46 Wh/km
Cycle 3	2.7 %	2025 Wh	245,75 Wh/km

Table 3.3: Energy consumption of the three measurement cycles

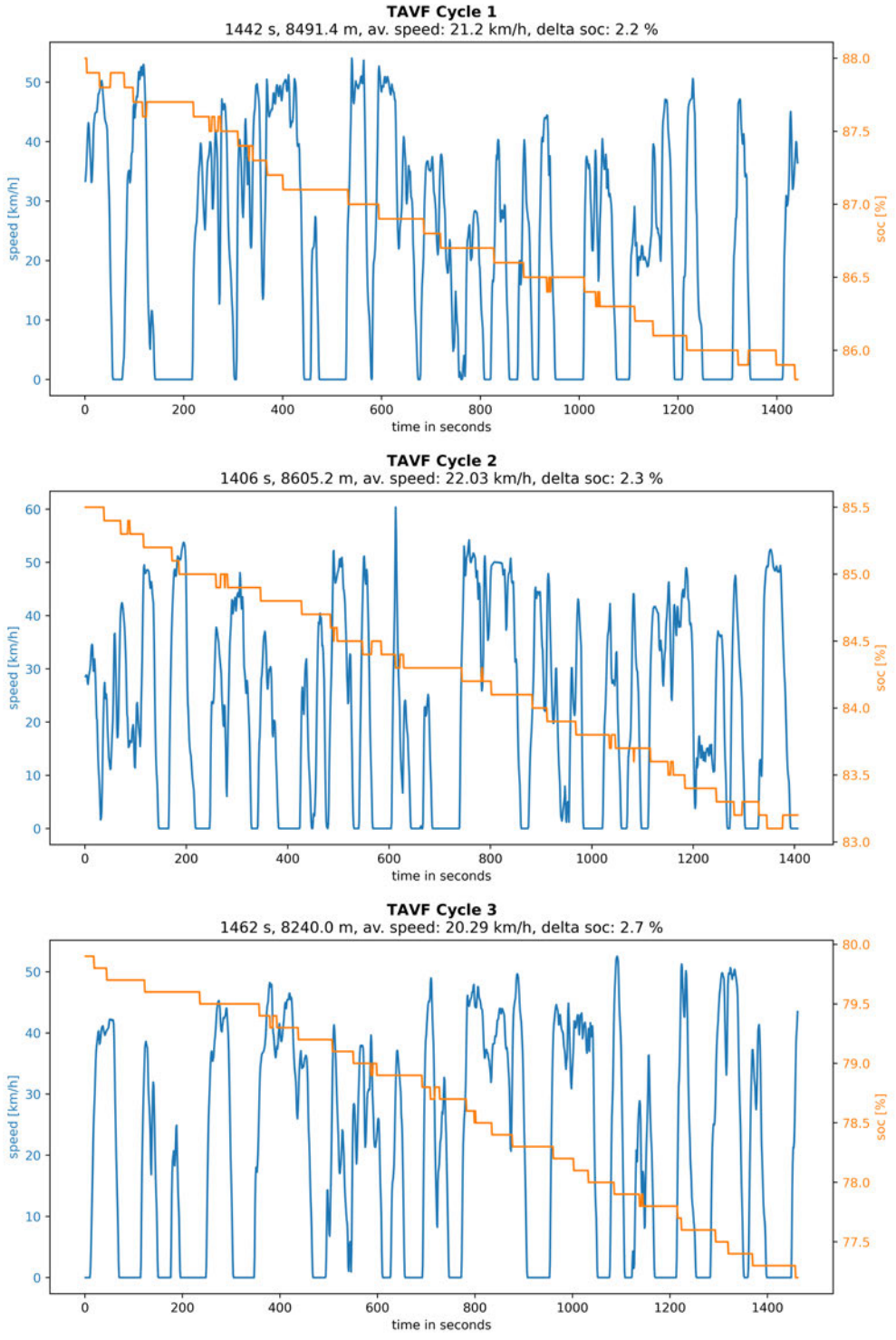


Figure 3.3: Measurement data from Cycle 1, Cycle 2 and Cycle 3

4 Simulation Model

This chapter describes the model implemented to run the energy simulations. It is described how the top-layer python script couples the traffic simulator and the energy model to generate driving cycles for CAVs and calculate their instantaneous energy consumption. The required software to implement the model is shown. Furthermore, this chapter is describing the used simulation files of the traffic simulator SUMO [21] and the setup for the forward-facing energy model, which is implemented in Matlab [23]. Finally, it is shown, how a driving algorithm can be implemented in the model to control vehicles in the simulation and what requirements such algorithms have to meet.

4.1 Simulation Toolchain

The developed model is based on a top layer python script that couples the traffic simulation tool SUMO and the forward-facing energy model implemented in Simulink.

With SUMO, CAVs can be simulated under various traffic conditions. The driving algorithm controlling the CAVs can be implemented in python (see section 4.5). The surrounding streets and vehicles are defined in network and route files in SUMO. The driving cycles for the analyzed CAVs are saved by the python script and forwarded to the energy model after a traffic simulation is done. From the energy model, the corresponding SoC-over-time-data for a driving cycle of a vehicle is obtained. Therefore, the model has to be parameterized for a given vehicle.

In this study, the developed model was used to analyze the energy performance of the test vehicle controlled by the driving algorithm implemented in SUMOs Kraus car-following model [18] on the TAVF. Therefore, multiple pseudo-random scenarios of the test vehicle driving a round on the TAVF are simulated for different parameters for acceleration and deceleration of the driving algorithm. The setup of the SUMO simulation can be seen in section 4.3.

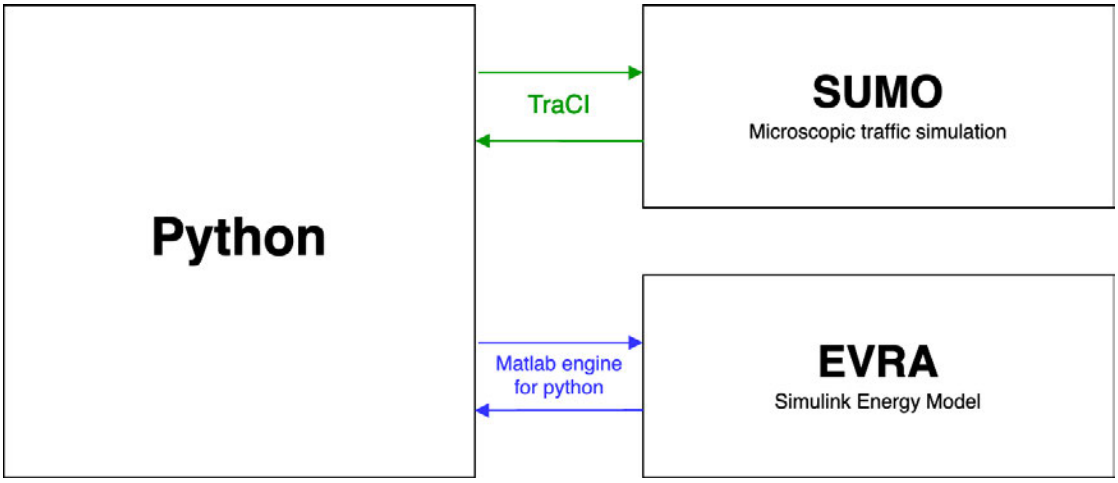


Figure 4.1: The toolchain of the coupled simulation model

For the results of this study (chapter 6), the python script 'automatic_simulation.py' was developed. In Figure 4.2 a flow chart of this script is shown. The whole code can be seen in the appendix (section A.1). The script initializes the simulation setup and creates the chosen vectors accel and decel with the values that are intended to be analyzed. The value of n is defining the number of random scenarios that are simulated per parameter set. In the simulations of this study, five values for the accel parameter and six for decel were analyzed. This gives 30 possible parameter combinations. For each of those parameter combinations, one simulation-run of n = 20 random simulations is executed. The randomness of the simulations is achieved by setting different seed values for the SUMOs random number generator (RNG). In 'automatic_simulation.py', the seed value for a simulation-run is increased for each simulation. Therefore, the first simulation in a simulation-run has a seed value of 1, and the n-th simulation a seed value of n.

For every simulation in a simulation-run a sumo config file is generated including the TAVF route and network files and the seed value. The communication between the python script and sumo is realized over TraCI (indicated green in Figure 4.1). With the generated .sumocfg-file SUMO is started as a server in the simulation_run()-function (line 184 of automatic_simulation.py).

```
184        traci.start([sumoBinary, "-c", cfg_file])
```

code 4.1: Starting SUMO as a server with TraCI from automatic_simulation.py

Afterward, the run()-function is executed with the accel and decel parameters of the current simulation-run. The function is defined in lines 63 to 92 of automatic_simulation.py and is shown in code 4.2. When the simulated test vehicle 'Tesla HAW' first appears in the simulation, which is set to 100 seconds plus a random departure offset, the values for accel and decel are set (line 76 and 77) for the driving algorithm in SUMO. After every simulation step that is executed the current speed of the test vehicle is appended together with the current simulation time to the cycle array (line 82). When the test vehicle is not in the simulation anymore the simulation is closed in line 86. The run()-function returns the cycle array with the speed over time values of the test vehicle in the simulation.

```
63 # runs sumo simulation with TraCi and returns cycle (speed over
       time) data for the test vehicle (TeslaHAW)
64 def run(accel, decel):
65     """execute the TraCI control loop"""
66     step = 0     # simulation step
67     time = 0     # simulation time
68     cycle = np.empty((0,2)) # array for the cycle data: collumn
           0 = time in seconds, collumn 1 = speed in m/s (later
           converted to km/h)
69
70     while(step < end/step_length):
71         traci.simulationStep() # execute one SUMO simulation
               step
72         if ('TeslaHAW' in traci.vehicle.getIDList()):
73             # initialiye tesla vehicle settings before first
                   step
74             if(time == 0):
75                 traci.vehicle.setImperfection('TeslaHAW', sigma)
76                 traci.vehicle.setAccel('TeslaHAW', accel)
77                 traci.vehicle.setDecel('TeslaHAW', decel)
78                 traci.vehicle.setSpeedFactor('TeslaHAW',
                       speedFactor)
79
80             time += 1
```

```
81              cycle = np.append(cycle, np.array([[time, traci.
                    vehicle.getSpeed('TeslaHAW')]]), axis=0) #
                    appending current time (s) and speed of tesla
82
83          # if no tesla after 500 seconds the route was completed
84          elif (step > 500):
85              traci.close()   # close SUMO simulation
86              sys.stdout.flush()
87              return cycle    # return driving cycle of tesla (
                    time in s, corresponding speed)
88
89          step += 1
90      traci.close()
91      sys.stdout.flush()
92      return cycle
```

code 4.2: run()-function of automatic_simulation.py

The cycle array is the input of the energy model to calculate the corresponding energy consumption for the test vehicle. The Matlab script 'run_EVRA.m' (section A.2) executes the forward-facing energy model (see section 4.4). The coupling to the python script is done through the Matlab engine for python, which is started in line 178 of automatic_simulation.py.

```
178     eng = matlab.engine.start_matlab() # starts Matlab engine
```

code 4.3: Starting a Matlab engine from automatic_simulation.py

The cycle array is converted to a Matlab double array in line 189. Then the Matlab script run_EVRA.m is executed and in line 191 the resulting SoC data is converted to a numpy array.

```
189         cycle_m = matlab.double(cycle.tolist())
190         soc_m = eng.run_EVRA(cycle_m, nargout=1)
191         soc = np.array(soc_m).astype(float)
```

code 4.4: Running the energy model from automatic_simulation.py

After the model has calculated the energy consumption for the cycle, the SoC over time array is returned to the python script. The SoC and speed data for the cycle are saved as a file and as a plot figure.

The script runs until the chosen number of scenarios is generated. Each of those scenarios is different since each of the SUMO simulations is initialized with another seed value for the pseudo-RNG.

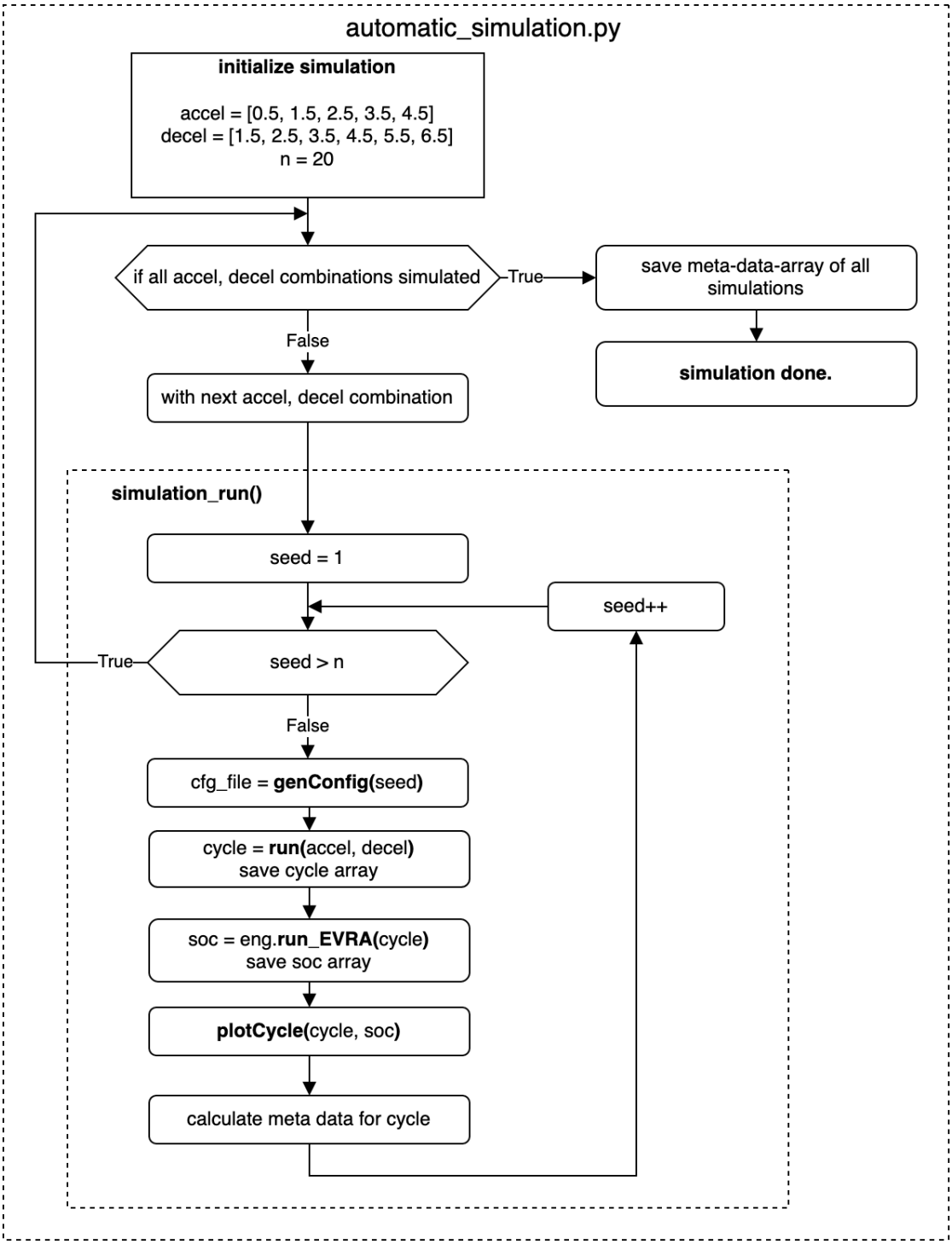


Figure 4.2: Flowchart of the top-layer python script automatic_simulation.py (section A.1)

4.2 Required Software

The top layer python script is implemented in python version 3.6.8 [31]. This is the latest version that is still compatible with the Matlab engine for python [23], which is also used. The energy model used in this study requires Simulink, the Powertrain Toolbox, and the Parallel Computing Toolbox of MathWorks from version 2020b or later. In the python code, the cycle data is analyzed and saved using the NumPy [26] library. For the traffic simulation the SUMO version 1.7.0 is used [21].

4.3 SUMO Traffic Simulation Setup

SUMO is used to generate random scenarios for the route of the test vehicle. The chosen driving algorithm can be analyzed under a big range of different traffic models for example on highways, in urban traffic, or certain cities or districts. The only requirements are accurate network and routing files for the whished traffic situations and enough computer power and time to run the simulations. The driving algorithm can be used to control one or more vehicles in the simulation. Note that especially the analysis of the energy consumption with the forward-facing energy model is computationally expensive, which increases the run time of this toolchain significantly if the energy consumption is calculated for multiple vehicles of the same scenario. Also, the energy model has to be parameterized for the exact vehicle it is representing (see section 4.4).

In this study, all generated cycles are simulated on the same road network, which is a model of the TAVF and only one vehicle in the simulated scenarios is controlled by the driving algorithm and analyzed with the energy model.

This vehicle is representing the test vehicle of the measurements. It virtually drives the same route on the TAVF as the real test vehicle in the measurements (see chapter 3) with the starting point at Hamburg Dammtor (as in Cycle 1 and 2 of the measurements) in every simulation scenario. Also, the other vehicles in the simulation - the traffic demand - are the same for every simulation. SUMO allows pseudo-random changes in the driving behavior of the other vehicles and a random departure offset of every vehicle created with a RNG based on an integer seed value. In one simulation run, for every SUMO simulation, the seed value is increased by one. This way, many different cycles are generated with the same route files.

The used network file and route files of the TAVF were provided by the German Aerospace Center (DLR). The route files are based on induction loop count data of the streets of the TAVF from the authorities of the state of Hamburg. Cars and trucks were generated from this data using a GEH statistic so that they represent weekday traffic between 15:00 and 16:00 with an accuracy between 95 and 99 %. The cars and trucks have a sigma value of 0.5, which means their driving behavior includes random deviations from a perfect one with a factor of 0.5. Additionally, buses are considered in the route files. The bus routes are based on the schedules from the Hamburger Verkehrsverbund (HVV, the public transport provider in Hamburg) from autumn 2019.

With the network file also the traffic lights are defined. In the provided network file of the TAVF, three of the 37 traffic lights of the TAVF are implemented based on the real phases of their counterparts, namely LSA 34, 560, and 94. The rest of the traffic lights were generated automatically by SUMO and partly optimized to guarantee realistic traffic flow during the simulations.

The different cycles simulated with SUMO from the python script all have the same configuration file except for the seed value, which is increased with every simulation by one. Therefore, the simulated scenarios are repeatable, but the behavior of the cars and their starting time varies randomly in every single scenario. The seed value for the RNG of SUMO is influencing the random behavior of other vehicles and the random departure offset of each vehicle, which is set in the simulation configuration to a maximum of 30 s. Therefore, every vehicle departs with a random offset between 0 and 30 s to their programmed departure time in every simulation.

4.4 Energy Simulation Setup

The used energy model is based on the Electric Vehicle Reference Application (EVRA) from MathWorks [23] which was parameterized for the 2017 Tesla Model S and extended for the most energy-consuming auxiliary systems of the Tesla in a previous study [25]. The model was rebuilt and saved in the Simulink model 'EVRA_tesla_EM.slx' for this study.

The diagram in Figure 4.3 shows how this model is used in this study. A driving cycle, a use factor for each of the listed auxiliary systems, and the atmospheric temperature are passed as input and after the energy simulation has finished the SoC data for the driving

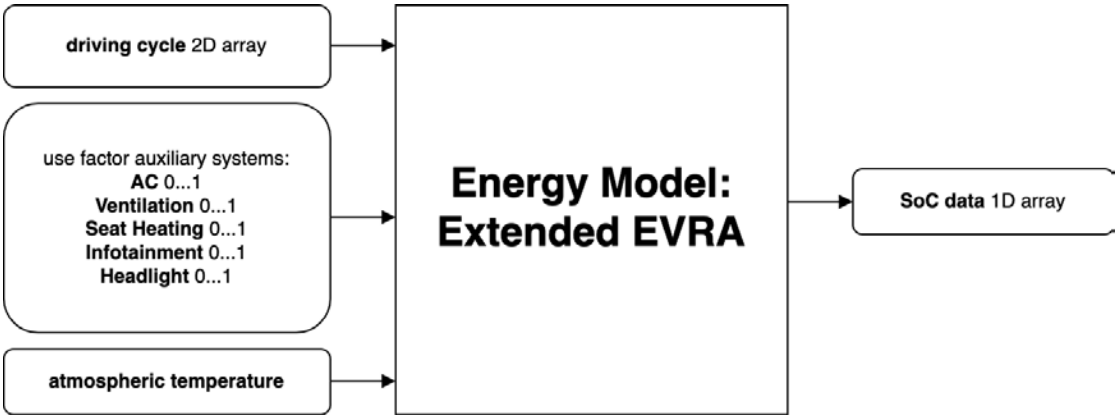


Figure 4.3: Inputs and outputs of the energy model

cycle is obtained. The driving cycle is passed as a two-dimensional array where the first column holds the time of the cycle in seconds and the second one the corresponding speed in km/h. The auxiliary systems and the atmospheric temperature are assumed to remain at a chosen value over the whole cycle.

To use the energy model in that way, the Matlab function 'run_EVRA.m' was written (section A.2). It takes the cycle array as an input, executes the simulation of the energy model, and retrieves the SoC data. The energy model saves the SoC values over the cycle with a resolution of 0.1 s. To assure the same length of the cycle input array and the corresponding SoC output array, the output vector is shortened to every 10th value before passed. Therefore, the SoC values in the output correspond to the time and the speed of the input array at the same index. The use factor for the auxiliary systems, the atmospheric temperature as well as the SoC value of the battery at the beginning of the cycle is set in the Simulink model before the simulation.

4.5 Driving Algorithms

With the method presented in this study, any longitudinal driving algorithm that is based on input data available in the SUMO simulation can be implemented in the python script to control the speed of one or multiple vehicles in the simulation. The performance of an implemented algorithm can then be tested over lots of random scenarios and under different traffic conditions. Furthermore, with the method presented in this study,

the algorithm can be analyzed in its energy consumption for a certain vehicle and its parameters can be optimized for the chosen traffic conditions.

Over the Traffic Control Interface (TraCI) of SUMO, the algorithm can access data of the vehicle's surroundings from the SUMO simulation in between each simulation step. A selection of the information that can be accessed with TraCI is given below.

- The current longitudinal speed of the vehicle
- The current lateral speed of the vehicle
- The current position of the vehicle
- A list of traffic lights on the vehicle's route with their current state and distance
- The leading vehicle of the car and its distance
- The current speed of any vehicle in the simulation

Based on this input data, the algorithm can control the vehicle's speed over the whole simulation. The route of the controlled vehicle can be programmed before in the route file so that the algorithm has to take care only of the longitudinal movement. The vehicle can be controlled by the algorithm by setting either its speed or its position for the next simulation step. It is also possible to consider lane changes in the driving algorithm. In that case, it can be implemented as a car-following model in SUMO.

To show the applicability of the presented method, the energy efficiency of the driving algorithm of the Kraus car-following model [18] is analyzed. This algorithm is already implemented in SUMO and calculates the speed of a vehicle for the next simulation step based on the parameters of its acceleration, deceleration, emergency deceleration, the minimal gap it should have to a leading vehicle, a value sigma representing the imperfection in the driving behavior and a value tau representing the reaction time a driver needs to react on changing conditions. The values sigma and tau are implemented to simulate non-perfect, human driving behavior in SUMO. Since in this study, the algorithm is used to simulate an autonomous vehicle, sigma is chosen to be 0 (no random variation in speed) and tau to the shortest possible value of the simulation step length, which is set to one second for the simulations in this study.

The algorithm decelerates the vehicle with a deceleration corresponding to the deceleration parameter value d to stop if a red or yellow traffic light is coming up or to respect

the minimal gap to any leading vehicle. Otherwise, the vehicle is accelerated depending on the acceleration parameter a.

4.6 Model Setup

With existing network- and route-files, a SUMO simulation can be run. SUMO simulates all vehicles over a given range of simulation steps which corresponds to a time period. The time of the beginning of the simulation, the time of its end and the simulation step-length has to be defined before the simulation is started in the .sumocfg-file. For example, to simulate the scenario for the first 10 minutes, the begin variable has to be chosen to 0 (s) and the end variable to 600 (s). The simulation parameter step-length determines how many steps are calculated for the scenario. It is set to 1 (s) as default. Therefore, in the given example, 600 simulation steps would be calculated representing the state of the scenario after every full second. This allows to access information about the scenario and its vehicles such as speed, position, state of the next traffic light, etc. after every full second.

The SUMO simulation can calculate the state of the network using one of the two integration methods Euler or Ballistic. The method determines how the speed between two simulation steps is assumed for the calculation of the positions of the vehicles. Using the Euler method, the vehicles are assumed to drive with the speed of the next simulation step over the whole time between two steps. Therefore, the position for a time step $p_{step=0}$ is calculated by adding the product of the speed for that time step $v_{step=0}$ and the simulation step-length $t_{step-length}$ to the position of the previous simulation step $p_{step=-1}$ (Equation 4.1). If the speed of a vehicle was 5 (m/s) in the last simulation step and is set to 0 for the next one, its position in the next step will be the same as in the previous one.

The Ballistic integration method assumes the vehicles to travel with the average speed of the previous and the next simulation step (see Equation 4.2). Therefore, using the ballistic method, in the given example with a simulation step-length of 1 (s), the next position would be the position of the previous step plus $\frac{5+0}{2} = 2.5m$.

$$Euler : p_{step=0} = p_{step=-1} + v_{step=0} \cdot t_{step-length} \tag{4.1}$$

$$Ballistic : p_{step=0} = p_{step=-1} + \frac{v_{step=0} + v_{step=-1}}{2} \cdot t_{step-length} \tag{4.2}$$

To find useful parameters for the simulation-step-length and the integration method, simulations with different parameters were run and analyzed. In all the simulations, one test vehicle was simulated to drive one circle on the TAVF. Therefore, the same route on the same road network as in all simulations from this study was simulated. A time period of 2000 seconds was simulated (begin = 0, end = 2000). The simulation was run for different step-length parameters and for scenarios with traffic and with the test vehicle as the only vehicle in the simulation by using the Euler and the Ballistic integration method. The simulations were analyzed on their simulation time, the real-time factor (the factor of the simulation time to equal the simulated period of 2000 s), the number of warnings, and the number of them referring to a vehicle in the simulation being teleported because of a crash or another problem and the route duration of the test vehicle to complete one round on the TAVF.

The simulation results for the Euler method can be seen in Table 4.2 and the ones for the Ballistic are shown in Table 4.1. The high number of warnings of simulations with a step-length greater than 2 s can be explained with a parameter for the action-step-length of a driver in SUMO, which default is 1 s. Therefore, in those simulations, the vehicles drive for a longer time period with an unchanged speed, than they are expected to need to react to their environment. Lots of crashes in the simulation are the following of those settings. One can see from the results, that the time the test vehicle needs to complete its route depends on the parameters of the step length and the integration method. The simulation time increases for both methods with decreasing step-length. This is expected, since the number of simulation steps that are calculated also increased with a decreasing step length. One can see that in simulations using the Ballistic integration method more warnings occur compared to the one using the Euler integration method except for a step-length of 0.01. Since the warnings that happened in these simulations all indicate either an emergency braking of a vehicle or the teleportation of a vehicle, a realistic traffic scenario of the TAVF over 2000 s should not include many warnings. Therefore, for the further simulations in this study, the integration method is chosen to be Euler and the simulation step-length to be 1 s. With these parameters, driving cycles with the highest

possible resolution expected by the energy model of one speed value per second can be generated in a short simulation time.

step length	route duration	simulation time	real time factor	number warnings (teleportations)
without traffic (test vehicle only)				
10 s	930 s	0.05 s	38461	0
2 s	906 s	0.1 s	21052	0
1 s	908 s	0.13 s	14925	0
0.1 s	908.8 s	0.69 s	2881	0
0.5 s	908.5 s	0.21 s	9708	0
0.01 s	908.98 s	6.41 s	312	0
0.001 s	908.990 s	62.513 s	31	0
with traffic				
2 s	884 s	23.67 s	84	>32000
1 s	1023 s	48.11 s	41	73 (24)
0.1	1004.4 s	519.28 s	3.85	179
0.5	1238.0 s	74.92 s	26	56
0.01	1039.84 s	2643.59 s	0.76	871 (3)

Table 4.1: Results for SUMO Ballistic simulation

step length	route duration	simulation time	real time factor	number warnings (teleportations)
without traffic (test vehicle only)				
10 s	960 s	0.06 s	36363	0
2 s	906 s	0.09 s	22727	0
1 s	907 s	0.12 s	16666	0
0.1 s	908.8 s	0.71 s	2816	0
0.5 s	908.0 s	0.21 s	9569	0
0.01 s	908.97 s	6.38 s	313	0
0.001 s	908.989 s	62.293 s	32	0
with traffic				
2 s	906 s	23.22 s	86	>35000
1 s	1158 s	41.60 s	48	1
0.1	1275.1 s	298.75 s	6.69	165 (1)
0.5	1099.5 s	70.5 s	28.4	24
0.01	1009.06 s	2583.99 s	0.77	929

Table 4.2: Results for SUMO Euler simulation

5 Model Validation

This chapter is dedicated to validate the two simulation models against the measurement data. For the Energy Model, the measured driving cycles of the TAVF are passed as inputs. The resulting SoC from the energy model is shown to be highly accurate when compared to the measured SoC.

The SUMO model of the TAVF is analyzed in the accuracy for the generation of realistic driving cycles. Therefore, a set of generated cycles is compared to the cycles measured in duration, distance, average speed, and the number of stops and time standing.

5.1 Energy Model Validation

The energy model was run for each measurement cycle with the required inputs: atmospheric temperature, the percentage of use of the considered auxiliary systems, and the speed cycle. Those values for the Cycles 1 to 3 are shown in section 3.1.

In Figure 5.2 the resulting SoC calculations for Cycle 1, 2, and 3 are plotted in red on top of the corresponding measurements.

One can see that the SoC values calculated with the energy model are roughly following the measured ones. Note that the measured SoC was logged with a precision of 0.1 % SoC and the calculated values were rounded to a precision of 0.001 % SoC . For Cycle 1, the Δ SoC, so the difference of SoC at the beginning of the cycle to the end of the cycle, is measured to be 2.2 % and calculated to be 2.219 %.

The absolute and relative differences between measured and calculated Δ SoC % consumed by each cycle are listed in Figure 5.1.

Cycle 3 shows the biggest gap with an absolute difference of 0.027 % SoC. In all three Cycles, the energy model comes to a Δ SoC with a relative deviation of 1 % or less.

TAVF Cycle	measured [%]	calculated [%]	difference abs. [%]	difference rel.
Cycle 1	2.2	2.219	0.019	+0.86 %
Cycle 2	2.3	2.291	-0.009	-0.39 %
Cycle 3	2.7	2.673	-0.027	-1.00 %

Table 5.1: Δ SoC [%] measured and calculated for each TAVF cycle

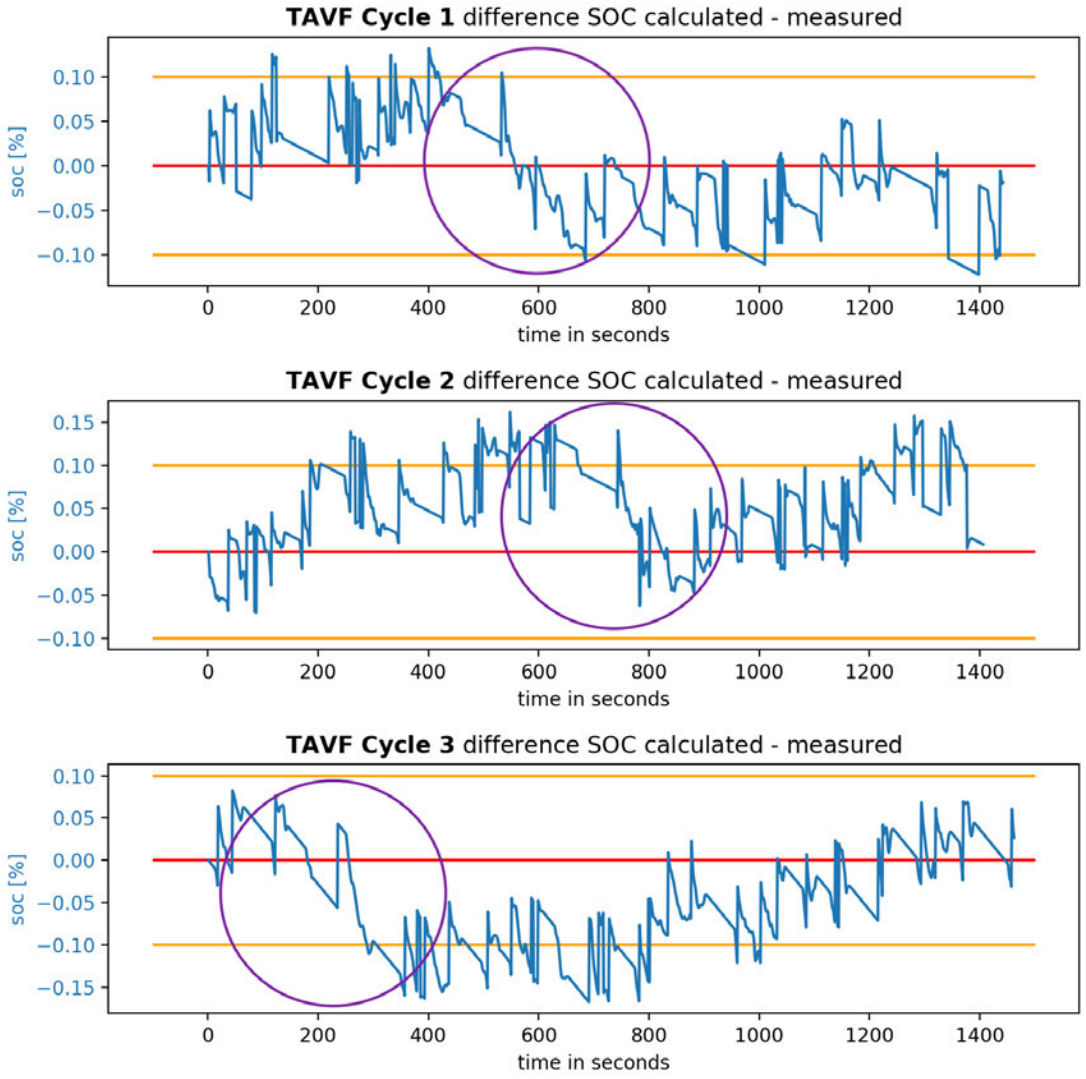


Figure 5.1: Difference from calculated SoC to measured SoC

In Figure 5.1 the difference in the SoC value in % from the energy model (calculated) to the measured one is shown over the whole cycle for TAVF Cycle 1, 2, and 3. In the plots, the highest resolution of the measurement values of 0.1 is marked in yellow.

One sees that even though the calculated SoC value is quite close to the measured one at the end of the cycles, during the cycles the difference is significantly higher. It can also be seen that this difference follows a bit of a pattern, where it is slightly increasing over more or less the whole cycle and then at some point dropping at a comparably high rate. In Cycle 1 and Cycle 2, this drop happens at about the same period of around 500 to 700 and 600 to 800 seconds meanwhile in Cycle 3 the drop can be seen between 150 and 350 seconds approximately.

This phenomenon can be explained by the road grade of the route. The TAVF is comparably flat over most of its length but has a higher negative road grade between U St. Pauli and U Landungsbrücken. Since the road grade is not considered in the energy model, in this part of the cycle the test vehicle consumes less energy than calculated. This results in higher calculated energy consumption over that period than measured, and consequently, the difference between calculated and measured SoC is decreasing. This explanation also agrees with the results of the study of S.C. Yang et al. [37], in which the influences of different road grades on an EVs energy consumption was analyzed.

The road grade of the rest of the TAVF does not have such significance at any other part of its length. Evidently, it does have a positive road grade at other parts to come back to the same height after a full round. Therefore, since the measured cycles were all going over a full round on the TAVF, during the rest of the cycle this effect is canceled out by the model calculating a lower energy consumption at the positive road grade parts.

The time period of the drop in the difference of calculated and measured SoC also meets the assumptions for the three cycles. Cycle 1 and 2 both have the same starting point at Dammtor. And they both show the drop event in about the same time period between 500 to 800 seconds from the cycle start.

Cycle 3 on the other hand started at Holstenwall, which is closer to the negative road grade part. Therefore, this cycle also shows the drop event earlier.

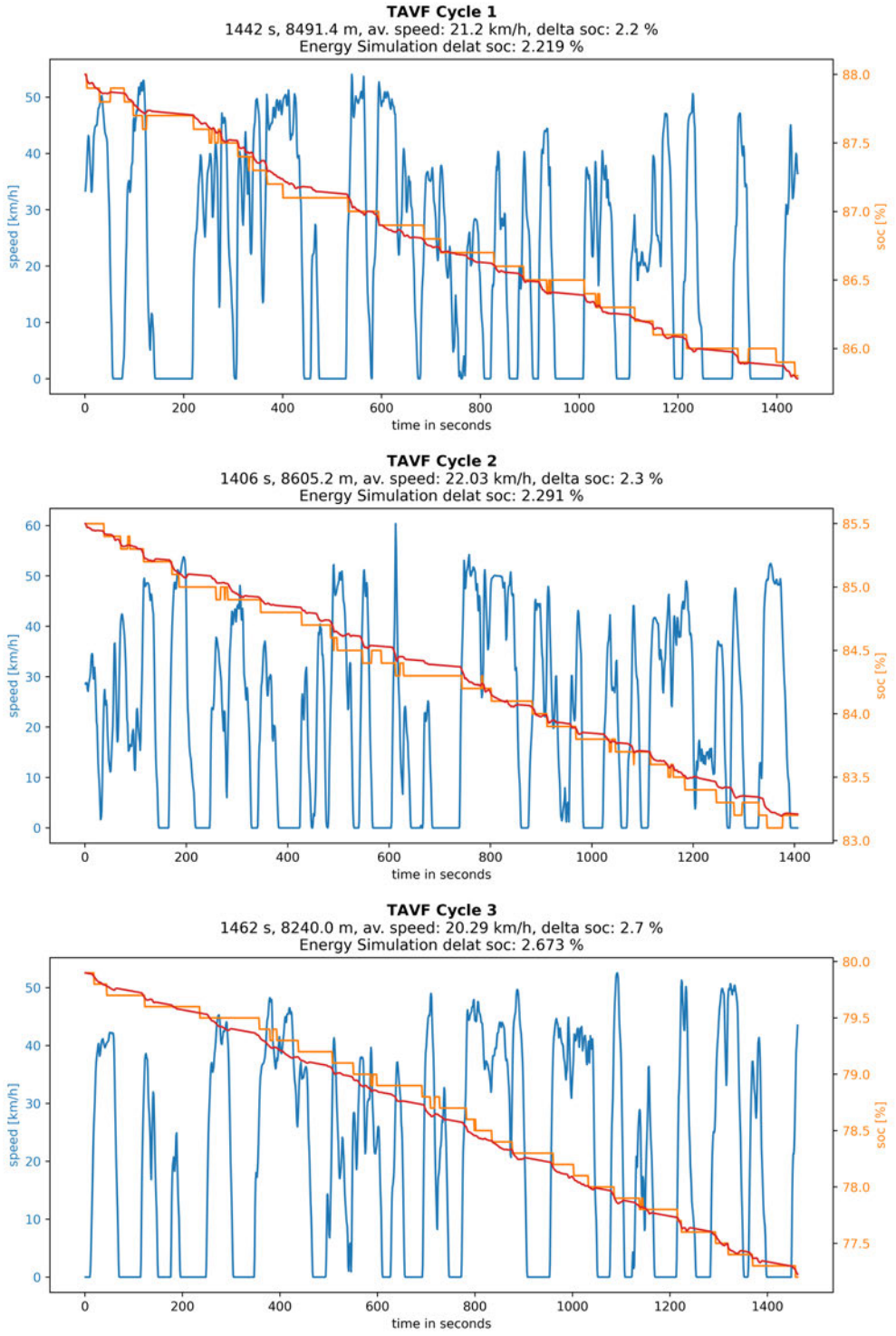


Figure 5.2: Cycle 1, Cycle 2 and Cycle 3 measurements and the SoC of the energy model (red)

5.2 TAVF Driving Cycle Generation with SUMO Model

For the method presented in this study, SUMO is used to generate realistic driving scenarios for a vehicle under fixed traffic conditions. With the used SUMO model for the TAVF and the traffic presented in section 4.3, the resulting driving cycles of the test vehicle of 100 pseudo-random simulations were analyzed and compared to the measured driving cycles of the TAVF.

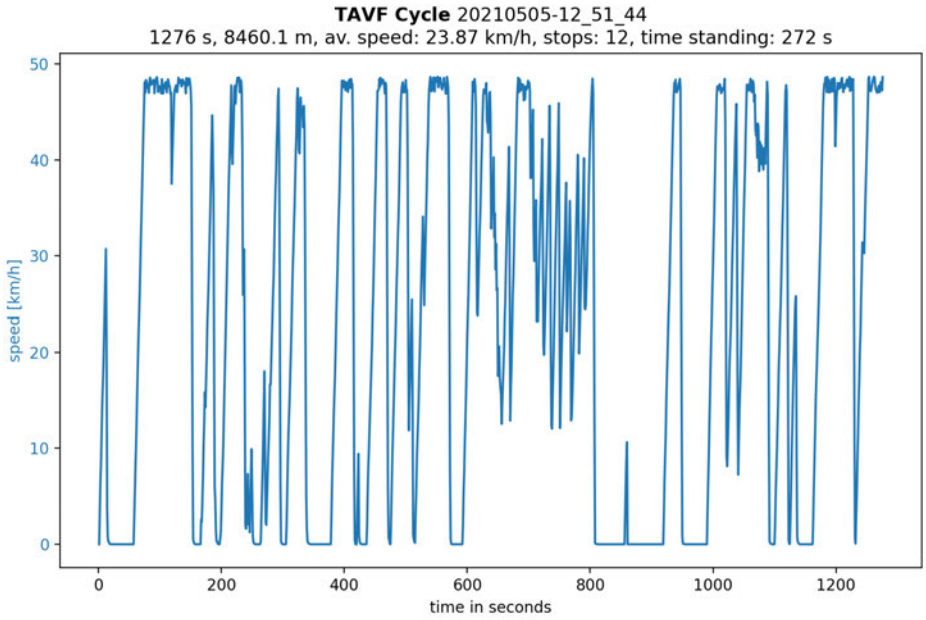


Figure 5.3: One of 100 random TAVF cycles generated with SUMO

For the 100 cycles, the test vehicle was simulated with the Kraus car-following model in SUMO. Its acceleration was set to 1.0 m/s^2, the deceleration to 5 m/s^2 and the value for sigma - which represents a driving imperfection in SUMO - was chosen to be 0.5. In Figure 5.3 one of those 100 cycles is shown. When compared to the measured driving cycles from Figure 5.2, one can see in the example plot that the driving behavior simulated in SUMO does not match the real human driving behavior from those cycles very closely. In the simulation, the vehicle increases speed to its speed limit (50 km/h multiplied by a random factor) with steady acceleration, and the random speed variation around the limit caused by the sigma-value looks like white noise. Also, the deceleration is steady when braking. The measured cycles in contrast have varying acceleration and deceleration values and the speed variations do not have the characteristics of white

TAVF Cycle	duration	distance	average speed	stops	time standing
Cycle 1	1442 s	8491.4 m	21.20 km/h	15	428 s
Cycle 2	1406 s	8605.2 m	22.03 km/h	16	340 s
Cycle 3	1462 s	8240.0 m	20.29 km/h	15	489 s
100 generated cycles:					
average	1394 s	8464.2 m	22.08 km/h	15.95	384 s
max	1772 s	8472.1 m	28.57 km/h	23	662 s
min	1067 s	8450.8 m	17.19 km/h	9	127 s
std. dev.	137.11 s	4.35 m	2.22 km/h	3,1	98,47 s

Table 5.2: Key values of random SUMO cycle generation compared with measurements

noise. Nevertheless, the generated cycles do deliver some clues about the suitability of the TAVF model in SUMO. Therefore, the values of duration, distance, average speed, the total time standing, and the number of stops are observed for the generated cycles and compared to the measured cycles.

All of these values are approaching a normal distribution over the 100 random cycles. Their average, maximum, minimum, and standard deviation for the 100 simulations are shown in Table 5.2 together with the data of the measurements. The time standing is the number of seconds where the vehicle had a speed of 0 km/h and for the number of stops, the occurrences of standing situations (consecutive speed of 0 km/h) are counted where short movements of less than 5 seconds (up to four consecutive seconds with a speed other than 0 in between two standing situations) are respected as one single stop. This prevents a stop at a single traffic light where the vehicle moves forward again after stopping to close the gap to the leading vehicle in the same red phase to be counted as two stops.

The statistics of the simulated cycles show that the model matches the values from the measured Cycle 1, 2, and 3 in the chosen categories. The values for duration, average speed, and stops of all three measured cycles are inside one standard deviation from the mean value over the 100 simulated cycles. For the standing time, only Cycle 3 is outside this boundary with 489 seconds, being 1.35 % higher than the one standard deviation over the mean. Also, this Cycle is well below the maximum standing time of all 100 generated cycles which is 662 seconds. The only category in which the value range of the simulated cycles does not cover the values of the measurements is the distance of the cycle. Here, the standard deviation over the values of 100 simulated cycles is very little with 4.35 m. The difference in the distance of the simulated cycles is, therefore, much smaller than the difference between the distance of the measured cycles, which have a

maximum difference of 365.3 m. It is also striking that the distance of Cycle 3, which was measured about one month later than Cycle 1 and 2, is significantly smaller than the distance of Cycle 1 and 2.

This could possibly be due to measurement inaccuracies. The distance of the cycles from Table 5.2 were calculated from the speed cycle. For the measurement cycles, a speed value for every 0.1 s was measured and used to calculate the distance over that time period. Afterward, the distances were accumulated for each cycle. The measured speed from the CAN-bus is calculated by the internal Tesla systems from the rotation speed of the wheels multiplied by the wheel perimeter. Even though all three cycles were driven with the same wheels, their perimeter can vary slightly due to differences in tire pressure and tread. If the tire pressure of the test vehicle was higher during the measurement of Cycle 3 compared to the one during Cycle 1 and 2, the wheel perimeter was larger as well. And if the wheel perimeter was larger than the one assumed by the car to calculate the current speed, the actual speed values were higher, than the ones measured. Therefore, also the real distance traveled during the cycle would be larger. Between Cycle 1 and 2 a difference in tire pressure is very unlikely since they were taken consecutively.

Another influence on the distance is the selection of the period of the measurement cycles. During the measurements of Cycle 1 and 2, a log-book was written with the time of the start and end of each cycle to identify the period of the TAVF cycle from the measurement data. Inaccuracies in the start and end time of the cycle also increase or decrease the distance and duration of the cycle.

For Cycle 3, these inaccuracies were avoided by starting the measurements from a parking position at the side of the TAVF and finishing in at the same parking.

The fact that Cycle 3 is 210.8 m shorter than the lowest simulated distance might also be caused by the route of the used TAVF model in SUMO being about this distance longer than the real route on the TAVF (that would be 2,56 %). In that case, the TAVF model would also show slightly higher average speed values since it includes all the traffic lights but has longer streets in between them where the vehicle is driving faster than the average speed. Also, the duration of a cycle would be slightly higher as if it would be simulated with a TAVF model that has the real distance of the route.

Overall, the randomly generated cycles show that the TAVF model used in this study matches the expectations of the three measured cycles of the real TAVF. Excepts for the distance where the lowest simulated one is 2.56 % larger than the lowest measured one,

the values of the 100 simulated cycles are all distributed over a range that contains the measured ones. If one assumes that the used model was representing the TAVF perfectly, the measured values of Cycle 1 to 3 for the duration, average speed, stops, and standing time do not only represent possible but also very probable quantities when compared to the distribution of the values from the simulations. Vice versa, the presented results can be seen as evidence that the network and routing files used in the simulation are an accurate model of the real TAVF with realistic traffic.

6 Simulation Results

In this section, the simulation results for the energy consumption of the driving algorithm of the Kraus car-following model are presented. With the method described in chapter 4, the test vehicle is simulated to drive always the same route on the TAVF. During the simulation, its speed is controlled from the algorithm of the Kraus car-following model described in section 4.5 with different parameters for acceleration and deceleration. Next, the performance of the algorithm is analyzed for the energy consumption and the duration of one TAVF cycle.

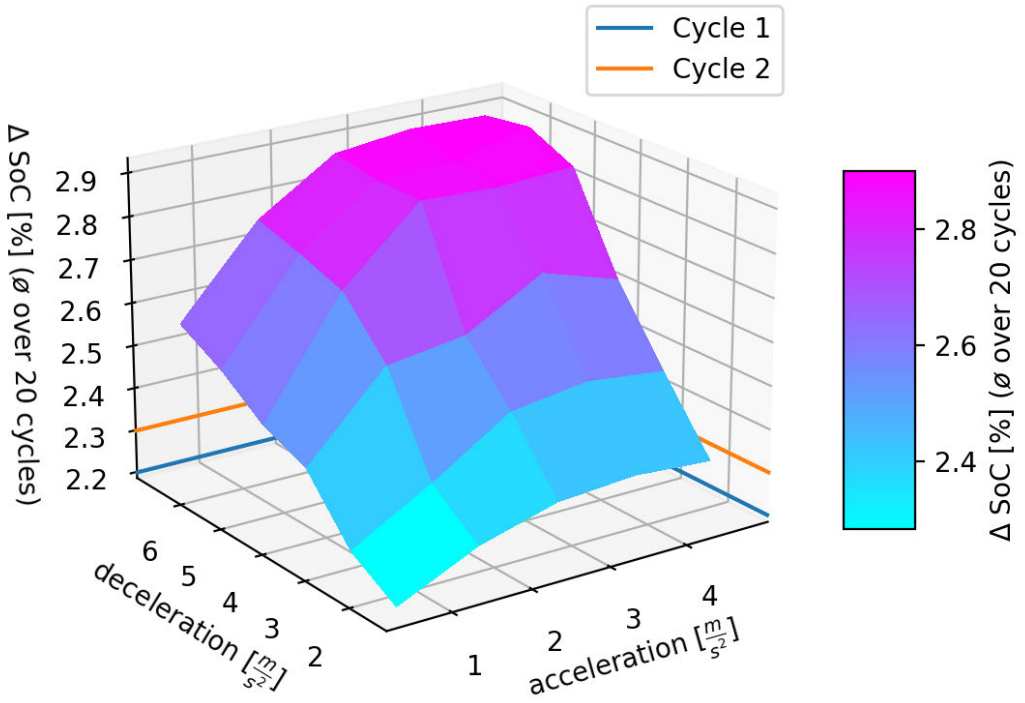


Figure 6.1: Average Δ SoC over random 20 TAVF cycles for different acceleration and decelaration parameters with low auxiliary system use

The acceleration parameter a was set to 0.5, 1.5, 2.5, 3.5 and 4.5 m/s^2 and the decelartion parameter d to 1.5, 2.5, 3.5, 4.5, 5.5 and 6.5 m/s^2. For each value combination of a and d in the driving algorithm, the test vehicle was simulated over 20 random cycles and its energy consumption has been calculated.

In Figure 6.1 the average difference in SoC per cycle over 20 cycles is plotted for different combinations of the driving algorithm parameters of a and d. The energy model was set to the lower auxiliary system use of Cycle 1 and 2 of the measurements for those results (see subsection 3.1.3). The difference in SoC of these two cycles measured is also indicated in the plot. One can see a trend for higher energy consumption if the algorithm is set to higher a and d values. Only in three simulation-runs the average Δ SoC stays below the 2.3 % of the measurement Cycle 2. This is in the runs with the parameter combinations $a = 0.5$ and $d = 0.5$ m/s^2 (2.204 %), $a = 1.5$ and $d = 1.5$ m/s^2 (2.288 %), and $a = 0.5$ and $d = 2.5$ m/s^2 (2.268 %). None of the simulation-runs achieved a lower average energy consumption per cycle than the 2.2 % Δ SoC of Cycle 1.

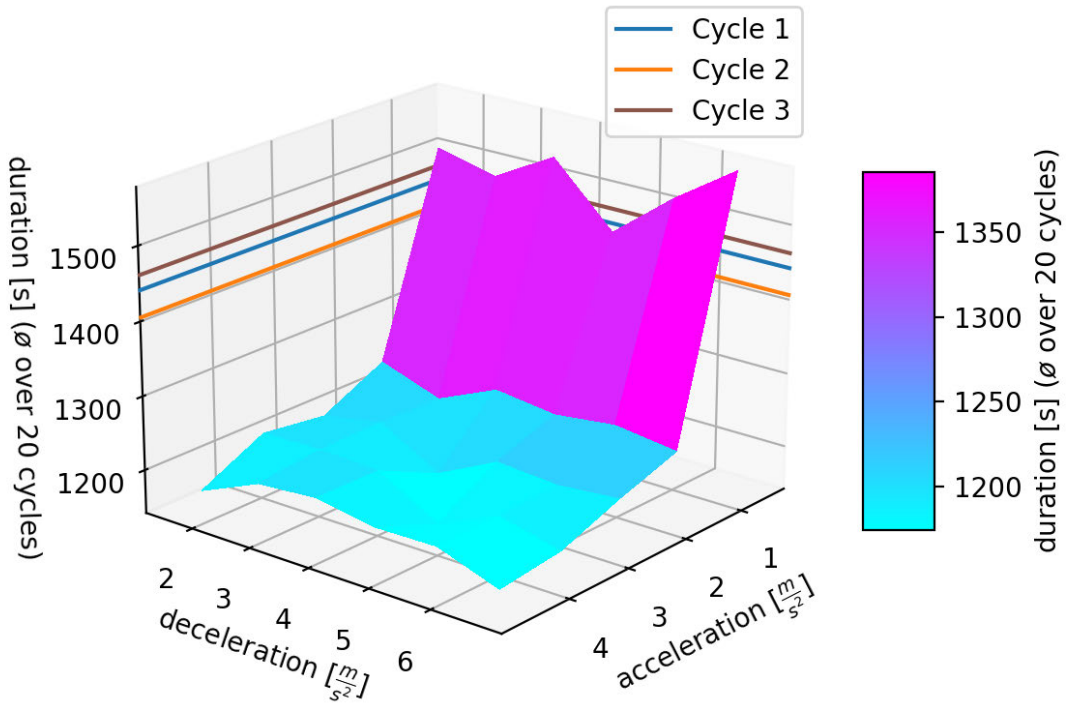


Figure 6.2: Average duration over random 20 TAVF cycles for different acceleration and decelaration parameters

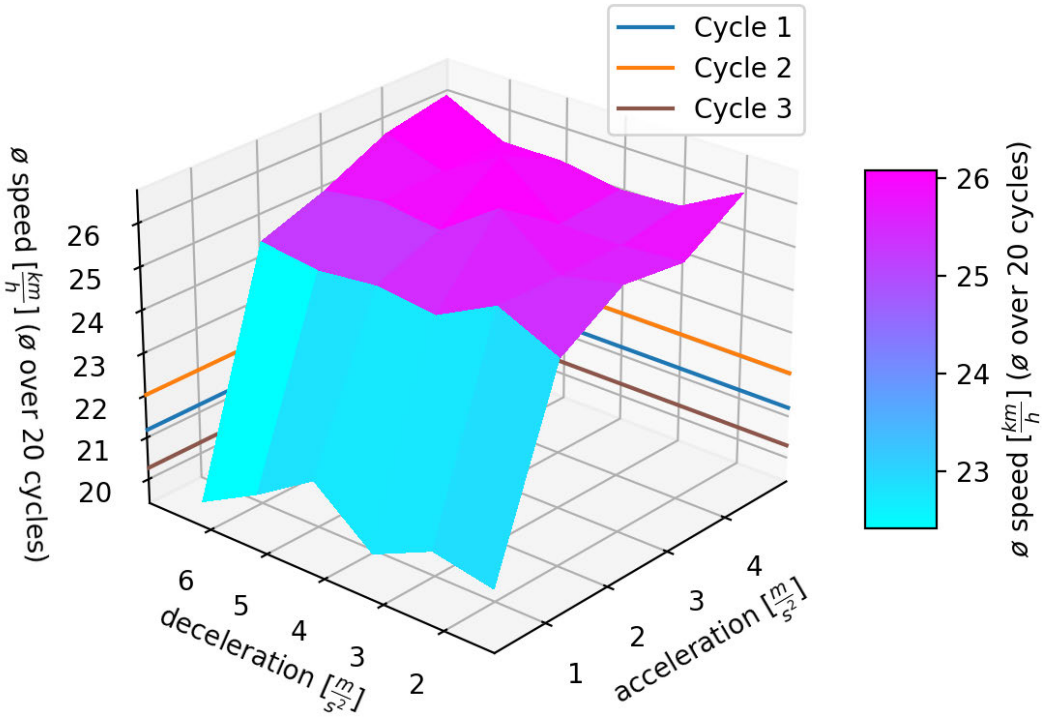


Figure 6.3: Average over the average speed of random 20 TAVF cycles for different acceleration and decelaration parameters

Figure 6.2 and Figure 6.3 show the average duration and average speed per TAVF cycle over 20 cycles for the same a and d values. Also, the corresponding values of the three measurement cycles are marked in the figures. Since the average cycle distance does not vary more that 10 m between the simulated parameters (the minimum is 8457 and the maximum 8466 m), duration and average speed data show the same trend. One can see that for a values of 1.5 m/s^2 and higher the average cycle duration remains very similar around 1200 seconds. For those parameters, the maximum duration is 1237 s (at $a = 1.5$ m/s^2 and $d = 3.5$ m/s^2).

Meanwhile a values above 1.5 m/s^2 do not result in significant changes in the cycle duration, the simulation results show significantly higher values for a lower a of 0.5 m/s^2. Here, the duration is between 1444 and 1568 s, which is between 16.7 % and 26.8 % higher, than the highest result for other parameters (1237 s). The parameter d does not influence the cycle duration considerably over the whole range of simulated values.

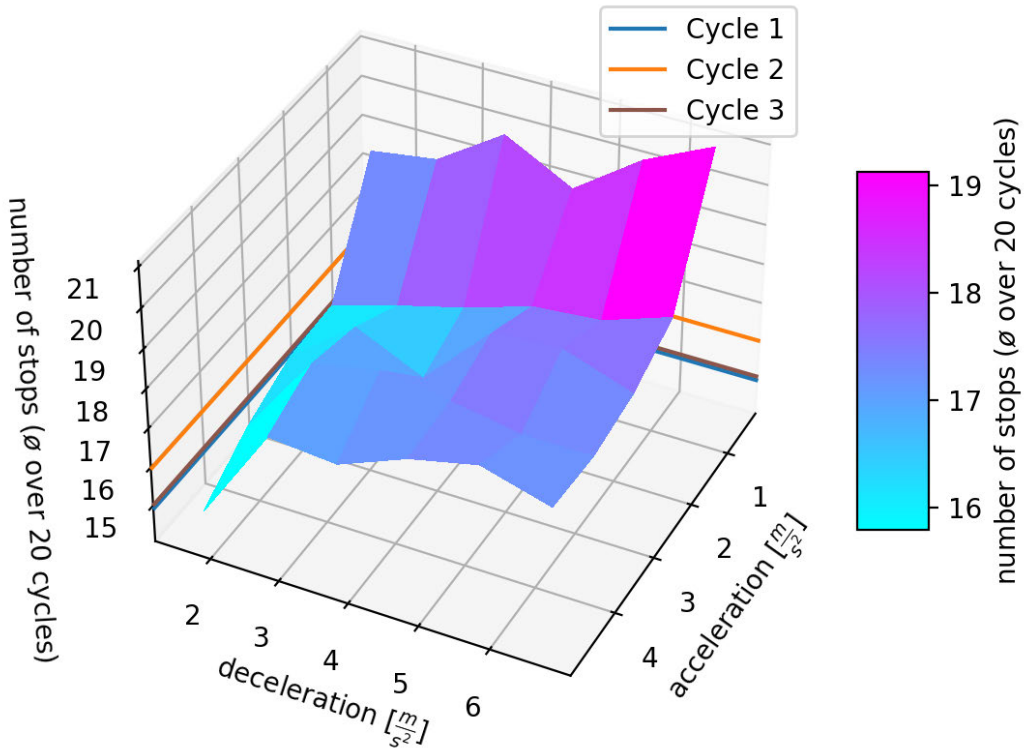


Figure 6.4: Average number of stops per cycle over random 20 TAVF cycles for different acceleration and decelaration parameters

The average number of stops and the average stopping time per cycle are shown in Figure 6.4 and Figure 6.5 respectively. The plots indicate that generally for the simulated scenarios and overall the acceleration and deceleration parameters, more stops lead to longer stopping time. Also, they show that for a low a of 0.5 m/s^2 the stopping time is significantly higher than for a values of 1.5 m/s^2 and above. This can explain the longer cycle duration for those simulations and the lower average speed. The results for the average number of stops and stopping time in the simulated TAVF cycles for different acceleration and deceleration parameters of the driving algorithm also show a light trend of lower values for a lower d meanwhile, the average cycle speed does not show this trend. Therefore, for the lower deceleration values, lower stopping time and fewer stops do not result in a higher average speed or lower cycle duration.

Regarding the average energy consumption per cycle for lower auxiliary system use in Figure 6.1, the cycles with an a value of 0.5 m/s^2, which have a longer cycle duration,

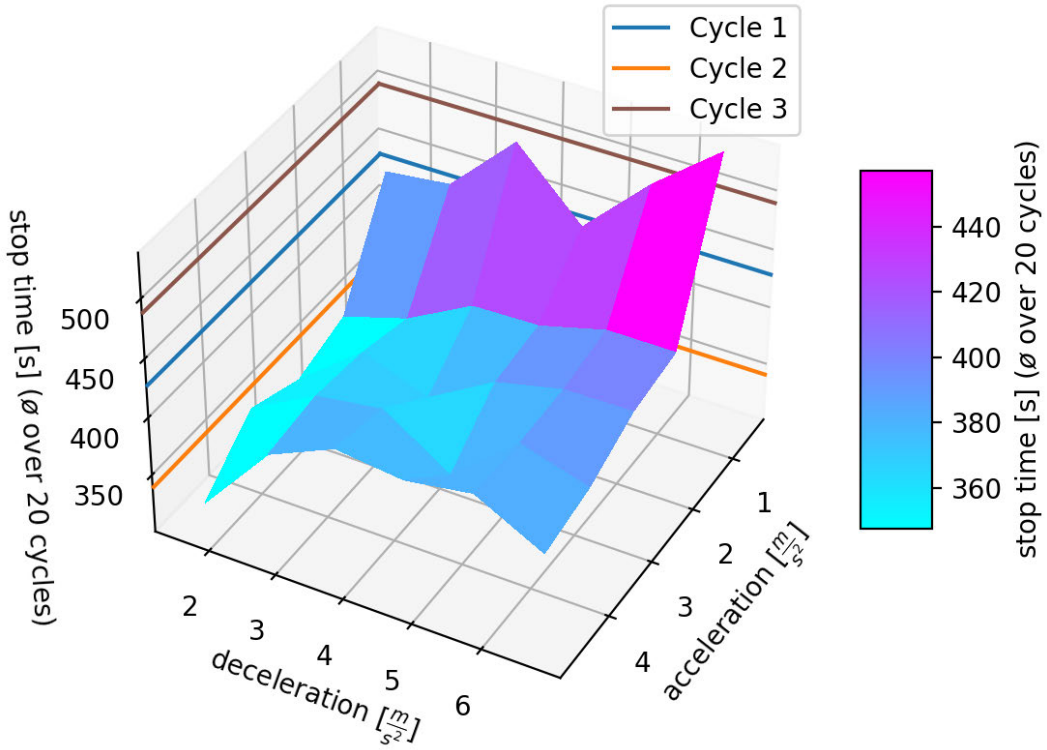


Figure 6.5: Average stoping time per cycle over random 20 TAVF cycles for different acceleration and decelaration parameters

still show a lower energy consumption than the cycles with higher a values that have shorter trip duration.

In Figure 6.6, the energy consumption for the same cycles is shown with higher auxiliary system use. Therefore, the energy model is set to the parameters for auxiliary systems and atmospheric temperature of the measurements Cycle 3. The Δ SoC of Cycle 3 is also plotted in the figure. One can see that with this auxiliary system use, the energy consumed in all cycles is higher. The average Δ SoC per cycle over all the cycles is here 3.196 %. For the energy calculations with lower use of auxiliaries in Figure 6.1, the average over all the cycles is 2.610 %. Therefore, the energy consumption per TAVF cycle increased by 0.586 % SoC on average when changing from a low auxiliary system use as in measurement Cycles 1 and 2 (AC and Ventilation to 30 %, atmospheric temperature of 8 degrees Celsius) to a higher one as in the measurements of Cycle 3 (AC and Ventilation to 80 %, the atmospheric temperature of 4 degrees Celsius, see subsection 3.1.3). This is an increase in energy consumption of 22.4 %.

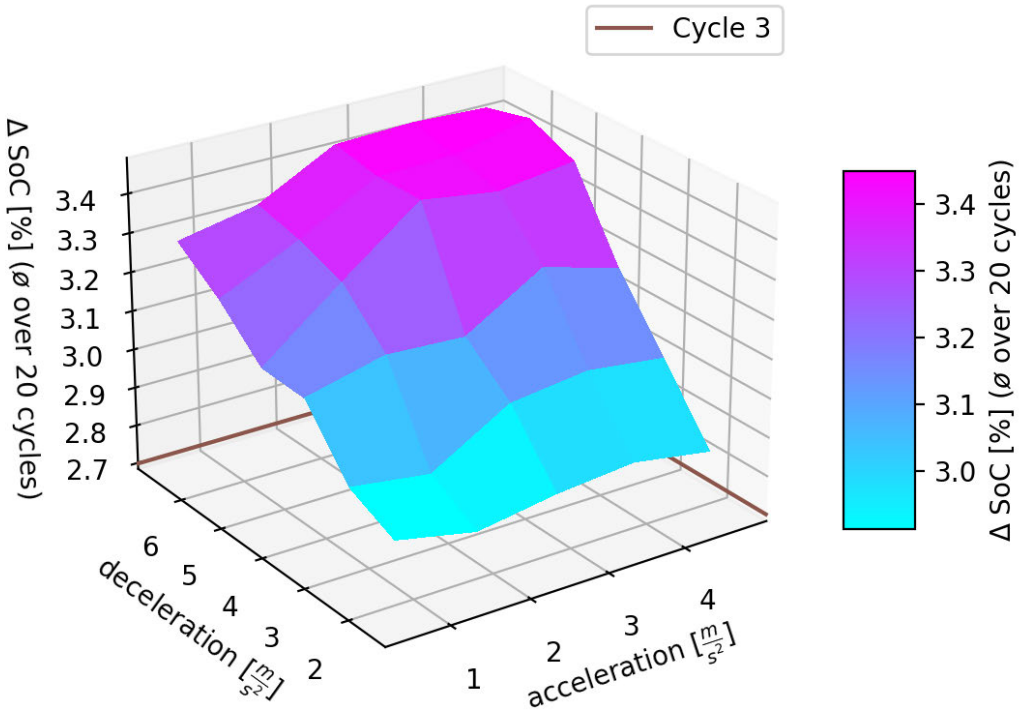


Figure 6.6: Average Δ SoC over random 20 TAVF cycles for different acceleration and decelaration parameters with high auxiliary system use

One can also see when comparing Figure 6.1 and Figure 6.6, that the longer cycle duration for simulations with an a value of 0.5 m/s^2 leads to a larger increase in energy consumption for the high auxiliary system use than for other a values, that correspond to lower cycle durations. In the cases for $d = 1.5$ m/s^2 and $d = 2.5$ m/s^2, one can see that with the high auxiliary system use, the energy consumption is even larger for an a of 0.5 m/s^2 than for one of 1.5 m/s^2, due to the longer duration of those cycles. In fact, in the simulations with $d = 1.5$ m/s^2, the average Δ SoC value over 20 cycles for different acceleration was even the second highest for an accelarion of 0.5 m/s^2 with 2.905 % (only with $a = 3.5$ m/s^2 the resulting energy consumption was slightly higher with a Δ SoC of 2.912 %).

The lowest average energy consumption per cycle achived by the algorithm is 2.862 % SoC for high auxiliary system use with the parameters $a = 1.5$ m/s^2 and $d = 1.5$ m/s^2. For the simulation with low auxiliary system use, the best value is 2.204 % with the parameters $a = 0.5$ m/s^2 and $d = 1.5$ m/s^2.

7 Conclusion

A method to evaluate the energy consumption of an EV controlled by a driving algorithm in realistic urban traffic was developed in this thesis. The simulation model couples the microscopic traffic simulator SUMO with a forward-facing energy model in Simulink. The energy model was validated with speed and SoC data collected under real urban traffic on the TAVF in Hamburg, Germany with a 2017 Tesla Model S 75D. The energy model with the parametrization of [25] was shown to calculate an energy consumption with a relative difference of 0.86 %, -0.39 % and -1.00 % to the measured one for the three measured TAVF cycles.

The used network and route files of the TAVF in SUMO were shown to deliver realistic cycle statistics when simulating driving cycles with the same virtual route on the TAVF in SUMO, that was driven for the measurements.

With the introduced method, a driving algorithm for a vehicle can be analyzed on the vehicle's energy consumption and also on other statistics such as trip duration, trip distance, average speed, number of stops, and stopping time. The compatibility for algorithms is so far limited to input variables available in SUMO and the vehicle can not be controlled by a torque request, but by setting its speed directly. However, in this direction, the model can possibly be extended (see chapter 8).

To demonstrate the applicability of the simulation model, the driving algorithm of the Kraus car-following model was analyzed. Therefore, multiple, random scenarios for different values of the acceleration and deceleration algorithm parameters were simulated. In each of the simulations, the driving algorithm controlled the speed of the test vehicle during one cycle on the TAVF. It was shown that the introduced simulation model can serve as an energy consumption analysis and optimization tool for a driving algorithm. With the model, a driving algorithm can be used to control one or multiple cars in any traffic scenario implemented in SUMO. The energy consumption for a simulated driving

cycle can be calculated for any given vehicle, that the energy model is parameterized for.

The simulation results show that, for the Kraus driving algorithm on the TAVF with low use of auxiliary systems, the cycles with the lowest values for the acceleration and deceleration parameters of 0.5 m/s^2 and 1.5 m/s^2 respectively have the lowest energy consumption with a Δ SoC of 2.204 %. This is very close to the energy consumption of the measured, human-driven cycles with the same auxiliary system use, namely Cycle 1 and Cycle 2. Here, the consumed energy was measured to be 2.2 % and 2.3 %, respectively. It was also shown that, for the Kraus driving algorithm, the trip duration for one cycle on the TAVF does not depend on the deceleration parameter. And the acceleration parameter does not influence the duration significantly with a value of 1.5 m/s^2 or above (not more than 6.68 %) but with a value of 0.5 m/s^2 the duration is 26.8 % longer than for higher values.

Furthermore, the energy simulations with higher use of the auxiliary systems AC and ventilation show that, for the same cycles, higher steady energy use results in a larger reduction of energy efficiency for cycles with a higher duration. Therefore, in this simulation with the higher auxiliary system use, the parameters deceleration = 1.5 m/s^2 and acceleration = 1.5 m/s^2 resulted in the lowest average energy consumption due to their lower cycle duration. With these parameters, the driving algorithm achieved an average Δ SoC of 2.862 % for a TAVF cycle. This is relatively 6 % more than the measured Δ SoC of 2.7 % for Cycle 3, which had the same, higher auxiliary system use.

The energy simulations demonstrate the significant effect of auxiliary systems on a vehicle's energy consumption. Overall the 600 TAVF cycle simulations of this study, the consumed energy per cycle increased by 22.4 % when changing the use factor for the AC and ventilation from 30 % to 80 %. This finding also overlaps with the results of [6].

8 Outlook

This thesis demonstrates the applicability of the coupled simulation model. It can now be used to study the energy performance of other driving algorithms and under different traffic conditions and optimize many parameters. Also, the effects of multiple CAVs controlled by a driving algorithm on energy efficiency and traffic flow can be analyzed. Therefore, insight on the potential of CAVs in the increase of the energy efficiency of the transport sector can be obtained.

Furthermore, different vehicle types can be analyzed by parameterizing the energy model for other vehicles. For the analyzed vehicles, also drive-train optimizations for defined routes and traffic conditions become possible with the model. And new vehicle-mobility concepts such as small city busses, car sharing, or car renting can be analyzed for their energy efficiency.

Despite using the model for further energy performance studies, the model itself can also be extended to increase its capabilities. Most of the driving algorithms developed for CAVs are based on camera data. In the developed version of the model, implemented driving algorithms have to meet the requirements described in section 4.5. They can only rely on inputs available in SUMO. A possible way to enable the implementation and analysis of a larger range of driving algorithms could be the coupling of the model with a three-dimensional graphical simulation tool such as webots [3]. With a webots simulation of the SUMO traffic simulation, sensors such as lidar or camera sensors can be simulated. Based on this simulated sensor data many new algorithms could be implemented to control the CAV in SUMO.

Bibliography

[1] Aimsun: *Aimsun Next*. https://www.aimsun.com/aimsun-next/. – [Accessed: 28.05.2021]

[2] Buhk, Benedikt ; Rettig, Rasmus: Simulation based method for the analysis of energy-efficient driving algorithms using SUMO, 2021

[3] Cyberbotics Ltd.: *Webots*. https://www.cyberbotics.com/. – [Accessed: 28.05.2021]

[4] Eclipse: *Eclipse Public License - v 2.0*. https://www.eclipse.org/org/documents/epl-2.0/EPL-2.0.html. – [Accessed: 29.05.2021]

[5] EEA: *Greenhouse gas emissions from transport in Europe*. https://www.eea.europa.eu/data-and-maps/indicators/transport-emissions-of-greenhouse-gases/transport-emissions-of-greenhouse-gases-12. December 2019. – [Accessed: 14.05.2021]

[6] Fiori, Chiara ; Ahn, Kyoungho ; Rakha, Hesham A.: Power-based electric vehicle energy consumption model: Model development and validation. In: *Applied Energy* 168 (2016), S. 257–268

[7] Galvin, Ray: Energy consumption effects of speed and acceleration in electric vehicles: Laboratory case studies and implications for drivers and policymakers. In: *Transportation Research Part D: Transport and Environment* 53 (2017), S. 234–248

[8] German Aerospace Center: *Institute of Transportation Systems*. https://www.dlr.de/ts. – [Accessed: 29.05.2021]

[9] German Aerospace Center: *SUMO User Conference 2021*. https://www.eclipse.org/sumo/conference/. – [Accessed: 16.05.2021]

[10] German Aerospace Center: *SUMO User Documentation*. https://sumo.dlr.de/docs/index.html. – [Accessed: 29.05.2021]

[11] Geschäftsstelle der Teststrecke für automatisiertes und vernetztes Fahren Hamburg c/o ITS mobility e. V.: *Teststrecke für automatisiertes und vernetztes Fahren in Hamburg*. https://tavf.hamburg. – [Accessed: 14.05.2021]

[12] Haklay, Mordechai ; Weber, Patrick: Openstreetmap: User-generated street maps. In: *IEEE Pervasive computing* 7 (2008), Nr. 4, S. 12–18

[13] Han, Jihun ; Sciarretta, Antonio ; Ojeda, Luis L. ; De Nunzio, Giovanni ; Thibault, Laurent: Safe-and eco-driving control for connected and automated electric vehicles using analytical state-constrained optimal solution. In: *IEEE Transactions on Intelligent Vehicles* 3 (2018), Nr. 2, S. 163–172

[14] HMS Industrial Networks: *canAnalyzer mini*. – URL https://www.ixxat.com/de/produkte/industrie-produkte/pc-can-interfaces-uebersicht/softwareunterstuetzung. – [Accessed: 05.05.2021]

[15] HMS Industrial Networks: *USB-to-CAN V2*. https://www.ixxat.com/de/produkte/industrie-produkte/pc-can-interfaces-uebersicht/usb/usb-to-can-v2-professional?ordercode=1.01.0281.12001. – [Accessed: 06.05.2021]

[16] Hughes, Jason: *Tesla Model S CAN Bus Deciphering*. http://skie.net/uploads/TeslaCAN/. – [Accessed: 07.05.2021]

[17] IPETRONIK GmbH & Co. KG (Veranst.): *SAM-CAN-ISO011*. – URL https://www.ipetronik.com/zubehoer-detail/sam-can-iso011.html. – [Accessed: 05.05.2021]

[18] Krauss, Stefan: Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. (1998)

[19] Kurczveil, Tamás ; López, Pablo A. ; Schnieder, Eckehard: Implementation of an Energy Model and a Charging Infrastructure in SUMO. In: *Simulation of Urban MObility User Conference* Springer (Veranst.), 2013, S. 33–43

[20] Lawrenz, Wolfhard: *CAN system engineering*. Bd. 121. Springer, 1997

[21] Lopez, Pablo A. ; Behrisch, Michael ; Bieker-Walz, Laura ; Erdmann, Jakob ; Flötteröd, Yun-Pang ; Hilbrich, Robert ; Lücken, Leonhard ; Rummel, Johannes ; Wagner, Peter ; Wiessner, Evamarie: Microscopic traffic simulation

using sumo. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* IEEE (Veranst.), 2018, S. 2575–2582

[22] Luin, Blaž ; Petelin, Stojan ; Al-Mansour, Fouad: Microsimulation of electric vehicle energy consumption. In: *Energy* 174 (2019), S. 24–32

[23] The Mathworks, Inc. (Veranst.): *MATLAB version 9.9.0.1467703 (R2020b)*. 2020

[24] Nourmofidi, Omidreza: *Entwicklung eines Systems zur automatisierten Erkennung und Meldung vulnerabler Verkehrsteilnehmer mittels C2X-Kommunikation*. 2021

[25] Oellerich, Christopher ; Rettig, Rasmus: *Entwicklung eines Modells zur Simulation der Energiebilanz beim Betrieb eines Elektrofahrzeugs*. 2020. – unpublished

[26] Oliphant, Travis E.: *A guide to NumPy*. Bd. 1. Trelgol Publishing USA, 2006

[27] PTV Planung Transport Verkehr AG: *VISSIM*. https://www.ptvgroup.com/de/loesungen/produkte/ptv-vissim/. – [Accessed: 28.05.2021]

[28] Smith, Laron ; Beckman, Richard ; Baggerly, Keith: TRANSIMS: Transportation analysis and simulation system / Los Alamos National Lab., NM (United States). 1995. – Forschungsbericht

[29] The Mathworks, Inc.: *EV Reference Application*. https://de.mathworks.com/help/autoblks/ug/electric-vehicle-reference-application.html#d123e19478. – [Accessed: 25.03.2021]

[30] V., Geschäftsstelle der Teststrecke für automatisiertes und ver-netztes Fahren Hamburg c/o ITS mobility e.: *TAVF Streckenkarte*. https://tavf.hamburg/fileadmin/user_upload/Bilder/TAVF_Streckenkarte_quer_de_200108.jpg. – [Accessed: 16.05.2021]

[31] Van Rossum, Guido u. a.: *Python Programming Language*. 2007

[32] Wegener, Axel ; Piórkowski, Michał ; Raya, Maxim ; Hellbrück, Horst ; Fischer, Stefan ; Hubaux, Jean-Pierre: TraCI: an interface for coupling road traffic and network simulators. In: *Proceedings of the 11th communications and networking simulation symposium*, 2008, S. 155–163

[33] Wikipedia: *CAN bus*. https://en.wikipedia.org/wiki/CAN_bus. – [Accessed: 06.04.2021]

[34] Wikipedia: *FTP-75*. https://en.wikipedia.org/wiki/FTP-75. – [Accessed: 16.04.2021]

[35] Wikipedia: *New European Driving Cycle*. https://en.wikipedia.org/wiki/New_European_Driving_Cycle. – [Accessed: 16.04.2021]

[36] Wikipedia: *Tesla model S characteristics*. https://en.wikipedia.org/wiki/Tesla_Model_S. 2017

[37] Yang, SC ; Li, M ; Lin, Y ; Tang, TQ: Electric vehicle's electricity consumption on a road with different slope. In: *Physica A: Statistical Mechanics and its Applications* 402 (2014), S. 41–48

A Appendix

A.1 automatic_simulation.py

```
1  #!/usr/bin/env python
2
3  # automatic_simulation_traci.py
4  # author: Benedikt Buhk
5  # date: 07.04.21
6  # description: this scrip runs a traffic simulation with sumo to
       generate a driving cycle for a test vehicle (tesla) and
       calculates the corresponding
7  # energy consumption using a matlab simulink model for a range
       random scenarios and varying acceleration and deceleration
       parameters.
8  # For the traffic simulation sumo is started as a server and the
       script connects as a TraCI-client. The energy simulation is
       done with the model
9  # EVRA/EVRA_tesla_EM.slx which is executed over the matlab
       script run_EVRA.m. The generated .sumocfg-files are saved in
       sim_config. The simulated cycles
10 # and their corresponding energy consumption (soc = state of
       charge) over time are saved in sim_cycles as numpy arrays
       and in sim_plots as png plots.
11 # the average metadata [seed, timestamp, duration, distance, av.
       speed(km/h), delat-soc, stopSeconds, nStops] over n cycles
       is saved as a numpy array in
12 # sim_cycles and as png plots in sim_plots_stat_analysis. in
       sim_outputs some outputs from the sumo simulations are saved
       and in sim_log one .txt-logfile
```

```
13 # is saved per execution of this script.
14
15 import os
16 import sys
17 from datetime import datetime
18 import numpy as np
19 import matlab
20 import matlab.engine
21 import matplotlib.pyplot as plt
22 from sumolib import checkBinary
23 import traci
24
25 # function declaration
26
27 #  generating a sumo-config file in the subfolder with the
       passed seed value (for randomness)
28 def genConfig(seed):
29     # generating tempTAVF.sumocfg
30     begin = 0        # time in second when simulation begins
31     fcd_tesla = 1    # enable sumo fcd-output for the tesla (test
           vehicle) in sumo
32     traffic = 1      # include traffic route files in the
           simulation config (0 = tesla as only vehicle in
           simulation)
33
34     ballistic = 0    # 0 = ballistic integration method, 1 =
           euler integration method
35     log = 1          # enable sumo simulation log (the log file
           will have the same timestamp as the generated config
           file)
36     random_depart_offset = 30    # vehicles in the simulation
           start with an random offset between 0 and 30 seconds
37
38     # timestamp (YYYYMMDD-HH_MM_SS) at time when config-file is
           generated
```

```
39      timestamp = datetime.now().strftime("%Y%m%d-%H_%M_%S")  # timestamp indicating when file was generated (to identify simulation )
40      cfg_file = 'sim_config/tempTAVF' + timestamp + '.sumocfg'  # path to the config file
41
42      # the cfg command executed in a terminal will generate the wished .sumocfg-file
43      cfg_cmd = 'sumo -C ' + cfg_file + ' -n ../sumo_files/tavf.net.xml -b ' + str(begin) + ' -e ' + str(end) + ' --seed ' + str(seed) + ' --step-length ' + str(step_length) + ' --random-depart-offset ' + str(random_depart_offset)
44
45      if (ballistic):
46          cfg_cmd += ' --step-method.ballistic'
47
48      if (log):
49          cfg_cmd += ' -l ../sim_outputs/simulation_' +timestamp + '.log.txt'
50
51      if (fcd_tesla):
52          cfg_cmd += ' --fcd-output=../sim_outputs/tesla_fcd_output_' +timestamp + '.xml --device.fcd.explicit=TeslaHAW'
53
54      if (traffic):
55          cfg_cmd += ' -a ../sumo_files/busStops.add.xml' #../static_files/detectors.add.xml,
56          cfg_cmd += ' -r ../sumo_files/tesla_tavf_start_dammtor.rou.xml,../sumo_files/vType.xml,../sumo_files/LKW_automatic.rou.xml,../sumo_files/PKW_automatic.rou.xml,../sumo_files/bus_route.rou.xml'
57      else:
58          cfg_cmd += ' -r ../tesla_tavf_start_dammtor.rou.xml'
59
```

```
60      os.system(cfg_cmd)  # execute the command (genereates the .
            sumocfg-file)
61      return cfg_file, timestamp  # return the path to the file
            and the timestap indicating when the file was generated
            (to identify simulation results)
62
63  # runs sumo simulation with TraCi and returns cycle (speed over
        time) data for the test vehicle (TeslaHAW)
64  def run(accel, decel):
65      """execute the TraCI control loop"""
66      step = 0     # simulation step
67      time = 0     # simulation time
68      cycle = np.empty((0,2)) # array for the cycle data: collumn
            0 = time in seconds, collumn 1 = speed in m/s (later
            converted to km/h)
69
70      while(step < end/step_length):
71          traci.simulationStep() # execute one SUMO simulation
                step
72          if ('TeslaHAW' in traci.vehicle.getIDList()):
73              # initialiye tesla vehicle settings before first
                    step
74              if(time == 0):
75                  traci.vehicle.setImperfection('TeslaHAW', sigma)
76                  traci.vehicle.setAccel('TeslaHAW', accel)
77                  traci.vehicle.setDecel('TeslaHAW', decel)
78                  traci.vehicle.setSpeedFactor('TeslaHAW',
                        speedFactor)
79
80              time += 1
81              cycle = np.append(cycle, np.array([[time, traci.
                    vehicle.getSpeed('TeslaHAW')]]), axis=0) #
                    appending current time (s) and speed of tesla
82
83          # if no tesla after 500 seconds the route was completed
84          elif (step > 500):
```

```
85                 traci.close()    # close SUMO simulation
86                 sys.stdout.flush()
87                 return cycle     # return driving cycle of tesla (
                       time in s, corresponding speed)
88
89             step += 1
90         traci.close()
91         sys.stdout.flush()
92         return cycle
93
94 # shows or saves plot of cycle (takes 2D cycle, 1D soc, and
       string(number or timestamp of cycle) as inputs)
95 def plotCycle(cycle, soc, number):
96         fig, ax1 = plt.subplots(figsize=(10, 6))
97         plt.title(r'$\bf{TAVF\ Cycle\ }$' + number + '\n' + str(len(
               cycle[:,1])) + '␣s,␣' + \
98             str(getDist(cycle)) + '␣m,␣av.␣speed:␣' + str(getAvSpeed
                   (cycle)) + \
99             '␣km/h,␣delta␣soc:␣' + str(np.round(soc[0, 0] - soc[-1,
                   0], 3)) + '␣%\n')
100        color = 'tab:blue'
101        ax1.set_xlabel('time␣in␣seconds')
102        ax1.set_ylabel('speed␣[km/h]', color=color)
103        ax1.plot(cycle[:,0], cycle[:,1], color=color)
104        ax1.tick_params(axis='y', labelcolor=color) #color of y-
               values
105
106        ax2 = ax1.twinx()
107        color = 'tab:red'
108        ax2.set_ylabel('soc␣[%]', color=color)
109        ax2.plot(cycle[:,0], soc[:], color=color)
110        ax2.tick_params(axis='y', labelcolor=color) #color of y-
               values
111        #plt.show()
112        plt.savefig('sim_plots/cycle' + number + '.png', dpi=200)
113        plt.close()
```

```
114
115 # takes meta_data array of simulation run and plots statistics
        of 6 categories
116 def plot_meta_analysis(meta_data):
117     run_first_timestamp = meta_data[0,1]
118     run_n_simus = len(meta_data[:,1])
119
120     run_av_data = np.empty((0))
121     run_av_data = np.append(run_av_data, run_first_timestamp)
122     run_av_data = np.append(run_av_data, run_n_simus)
123
124     for c in range(2,8):
125         data = meta_data[:,c].astype(np.float)
126         categories = np.array(['seed', 'timestamp', 'duration',
              'distance', 'av-speed', 'delta-soc', 'stop_time', '
              stops'])
127         categories_unit = np.array(['seed', 'timestamp', '
              duration(s)', 'distance(m)', 'av.speed(km/h)', '
              delta-soc(%)', 'stop time(s)', 'number of stops'])
128         print('analysing cycles in their ', categories[c])
129         print('first: ', data[0], '; last: ', data[-1])
130
131         run_mean = np.mean(data)
132         run_max = np.max(data)
133         run_min = np.min(data)
134         run_var = np.var(data)
135         run_std_dev = np.sqrt(run_var)
136
137         # append mean
138         run_av_data = np.append(run_av_data, run_mean)
139
140         plt.title(r'$\bf{Histogram\ of\ }$' + categories[c] + ':
              ' + str(run_n_simus) + ' cycles (run ' +
              run_first_timestamp + ') \n' \
141              + 'mean: ' + str(np.round(run_mean, 3)) + ' min: ' +
                   str(np.round(run_min, 3)) + ' max: ' \
```

```
142                     + str(np.round(run_max, 3)) + ' var: ' + str(np
                            .round(run_var, 5)) + ' std. dev.: ' + str(
                            np.round(run_std_dev, 3)))
143
144         plt.xlabel(categories_unit[c])
145         plt.ylabel('frequency')
146         plt.hist(data)
147         plt.axvline(x=np.mean(data), linewidth=2.0, color='red')
148         plt.axvline(x=np.mean(data)+np.sqrt(np.var(data)),
                linewidth=2.0, color='aqua')
149         plt.axvline(x=np.mean(data)-np.sqrt(np.var(data)),
                linewidth=2.0, color='aqua')
150         #plt.show()
151         plt.savefig('sim_plots_stat_analysis/simRun' + meta_data
                [0,1] + '_' + categories[c] + '.png', dpi=200)
152         plt.close()
153     print('run_av_data shape: ', np.shape(run_av_data), '; 
            run_av_data: ', run_av_data)
154     print('run_av soc: ', run_av_data[5], 'run av stops: ',
            run_av_data[7])
155     return run_av_data
156
157 # returns distance of cycle in m rounded to 1 digit after comma
158 def getDist(cycle):
159     return np.round(np.sum(1000*cycle[:,1]/3600), 1)
160
161 # average speed of cycle in km/h rounded to 2 digits after comma
162 def getAvSpeed(cycle):
163     return np.round(np.sum(cycle[:,1])/len(cycle[:,1]), 2)
164
165 # counts number of stops in the cycle. (2 stops less than 5
      seconds away from each other are counted as 1)
166 def countStops(cycle):
167     idx = np.where(cycle[:,1]==0)
168     nStops = 0
169     for i in range(1,len(idx[0])):
```

```
170             if idx[0][i] > (idx[0][i-1] + 5):
171                 nStops += 1
172         return nStops
173 
174 # runs n random simulations for given accel and decel values.
        also calculates soc with matlab simulink energy model, plots
        and saves cycle and meta data
175 def simulation_run(n, accel, decel):
176         print('starting simulation run of ', n, ' different
            scenarios')
177         sim_meta_data = np.empty((0,8)) # meat data ([seed,
            timestamp, duration, distance, av.speed(km/h), delat-soc
            , stopSeconds, nStops]) for each simulation
178         eng = matlab.engine.start_matlab() # starts Matlab engine
179         for seed in range(1,n + 1):
180             # generate config file
181             [cfg_file, timestamp] = genConfig(seed)
182             print('starting simulation of scenario', seed)
183             # sumo is started as a subprocess and then the python
                script connects and runs
184             traci.start([sumoBinary, "-c", cfg_file])
185             cycle = run(accel, decel) # runs sumo simulation (
                generates tesla cycle from tavf scenario)
186             cycle[:,1] = 3600*cycle[:,1]/1000    # converts from m/s
                to km/h
187             np.save('sim_cycles/cycle' + timestamp, cycle) # save
                cycle data
188             # simulate energy consumption of cycle (with matlab
                simulink EVRA model)
189             cycle_m = matlab.double(cycle.tolist())
190             soc_m = eng.run_EVRA(cycle_m, nargout=1)
191             soc = np.array(soc_m).astype(float)
192             np.save('sim_cycles/cycle' + timestamp + 'soc', soc)  #
                save calculated soc data
193             #plot cycle with soc
194             plotCycle(cycle, soc, timestamp)
```

```
195             # save key cycle data to metadata of current simulation
                    run ([seed, timestamp, duration, distance, av.speed(
                    km/h), delat-soc, stopSeconds, nStops])
196             sim_meta_data = np.append(sim_meta_data, np.array([[seed
                    , timestamp, len(cycle[:,1]), getDist(cycle),
                    getAvSpeed(cycle), soc[0, 0] - soc[-1, 0], np.
                    count_nonzero(cycle[:,1]==0), countStops(cycle)]]),
                    axis=0)
197             print('simulation␣of␣scenario␣', seed, '␣is␣done!')
198         # save metadata from all cycles of this simulation run
199         np.save('sim_cycles/metadata_cycle_' + sim_meta_data[0,1] +
                '_to_' + sim_meta_data[-1 ,1] + '_' + str(n) + 'cycles',
                 sim_meta_data)
200         # plot metadata analyses of this simulation run
201         return plot_meta_analysis(sim_meta_data) # return array
                run_av_data (8,) that contains first timestamp, number
                of cycles and average values of run for duration,
                distance, av.speed(km/h), delat-soc, stopSeconds, nStops
202
203 ### script starts here:
204 t_start = datetime.now()     # for simulation timer
205 timestamp_start = t_start.strftime("%Y%m%d-%H_%M_%S")
206
207 # we need to import python modules from the $SUMO_HOME/tools
        directory
208 if 'SUMO_HOME' in os.environ:
209     tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
210     sys.path.append(tools)
211 else:
212     sys.exit("please␣declare␣environment␣variable␣'SUMO_HOME'")
213
214 # tesla vehicle settings
215 sigma = 0.0          # sumo value for driver imperfection (random
        value between 1 and sigma is factor for speed factor)
216 speedFactor = 1.0    # factor for speed limit that is respected
        by vehicles
```

```
217 #speedDev = 0.1 # not adjustable with traci
218
219 end = 2000       # end of sumo simulation (globally declared to
        guarante access for run() function)
220 step_length = 1 # step length of sumo simulation (globally
        declared to guarante access for run() function)
221
222 # simulate with sumo gui or in sumo commandline tool (gui chosen
        when this script is called with 'gui' as an argument)
223 if (len(sys.argv) > 1 and 'gui' in sys.argv[1:]):
224     sumoBinary = checkBinary('sumo-gui') # path to sumo-gui
225 else:
226     sumoBinary = checkBinary('sumo') # path to sumo
227
228 # this is the main entry point of this script
229 if __name__ == "__main__":
230
231     ########################## parameters
            ##############################
232     n = 20  # number of simulations per accel-decel-pair
233     range_accel = np.arange(0.5, 4.6, 1.0)  # vector with all
            deceleration values to simulate (of the tesla)
234     range_decel = np.arange(1.5, 6.6, 1.0)  # vector with all
            acceleration values to simulate (of the tesla)
235     #
            ##########################################################################


236
237     # name of log file for whole simulation run (BigRunYYYYMMDD-
            HH_MM_SS_(n-accel-values)x(n-decel-values)cycles)
238     log_filename = 'sim_logs/BigRun' + timestamp_start + '_' +
            str(len(range_accel)*len(range_decel)) + 'x' + str(n)+ '
            cycles.txt'
239
240     # print simulation information and add it to the log file
```

```
241     message = '\
            n----------------------------------------------------------------------
            \nTOP SIMULATION started now (' + timestamp_start \
242     + '). \ntotal simulations: ' + str(len(range_accel)*len(
            range_decel)*n) + ' ( ' + str(len(range_accel)*len(
            range_decel)) + ' simulation runs of ' \
243          + str(n) + ' simulations each.) \n----------------------\
                nrange of parameters:\n' + str(len(range_accel)) \
244     + ' acceleration values ( ' + str(range_accel) + ') and\n' +
            str(len(range_decel)) + ' deceleration values ( ' \
245          + str(range_decel) + ')\
                n----------------------------------------------------------------------
                '
246     print(message)
247     os.system('echo \'' + message + '\' >> ' + log_filename)
248
249     # start the simulation run (n simulation for every parameter
            pair (accel and decel))
250     bigRun_meta_data_av = np.empty((0,8)) # array that saves the
            average meta data over n cycles for each parameter pair
251     # for every parameter pair
252     for accel in range_accel:
253         for decel in range_decel:
254             # print simulation information and add it to the log
                    file
255             message = 'starting simulation run with accel = ' +
                    str(accel) + ' and decel = ' + str(decel) + ' at
                    ' + datetime.now().strftime("%Y%m%d-%H_%M_%S")
256             print(message)
257             os.system('echo ' + message + ' >> ' + log_filename)
258
259             # runs n simulation for given value pair and saves
                    resulting average meta data
260             bigRun_meta_data_av = np.append(bigRun_meta_data_av,
                    np.array([simulation_run(n, accel, decel)]),
                    axis=0)
```

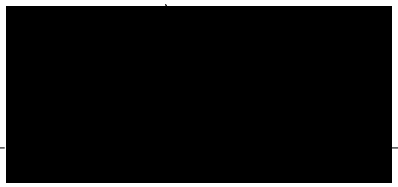
```
261
262     # save meta data array to file
263     np.save('sim_cycles/metadata_bigRun' + timestamp_start + '_'
            + str(len(range_accel)*len(range_decel)) + 'runs_a_' +
            str(n) + 'cycles', bigRun_meta_data_av)
264
265     t_finish = datetime.now()    # get time when simulation
            finishes for simulation timer
266     # print simulation information and add it to the log file
267     message = 'Simulation ended at ' + str(t_finish) + '(' + str
            (len(range_accel)*len(range_decel)) + ' x ' + str(n) + '
            = ' + str(len(range_accel)*len(range_decel)*n) + '
            scenarious)! (simulation started at: ' \
268     + str(t_start) + '; simulation duration: ' + str(t_finish -
            t_start) + ')\
            n_____\n'
269     print(message)
270     os.system('echo \'' + message + '\' >> ' + log_filename)
```

A.2 run_EVRA.m

```
1 %% setup model from python
2 % this script is called by a python script. it automatically
      starts the
3 % energy simulation and returns the results to python.#
4
5 function soc = run_EVRA(cycle)
6     cd('./EVRA')
7     input_cycle = cycle;
8     save('input_cycle', 'input_cycle')
9     dur = length(input_cycle);
10
11    out = sim('EVRA_tesla_EM.slx', 'StopTime', num2str(dur))
12    cd('..')
13    soc = out.logsout{2}.Values.Data(1:10:10*dur);
14    delta_soc = soc(1)-soc(end);
15 end
```

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.



Ort Datum