HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Mario Franke

# Interactive Driving Simulator for Car2X Scenarios

Bachelorarbeit im Fach Computer Engineering

12. Juli 2018

Please cite as:
Mario Franke, “Interactive Driving Simulator for Car2X Scenarios,” Bachelor Thesis (Bachelorarbeit), Heinz Nixdorf Institute, Paderborn University, Germany, July 2018.


Distributed Embedded Systems (CCS Labs)
Heinz Nixdorf Institute, Paderborn University, Germany

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.ccs-labs.org/

# Interactive Driving Simulator for Car2X Scenarios

Bachelorarbeit im Fach Computer Engineering

vorgelegt von

**Mario Franke**

geb. am 09. February 1996
in Paderborn

angefertigt in der Fachgruppe

**Distributed Embedded Systems**
**(CCS Labs)**

**Heinz Nixdorf Institut**
**Universität Paderborn**

| | |
|---|---|
| Betreuer: | **Dominik S. Buse** |
| | **Sven Henning** |
| Gutachter: | **Falko Dressler** |
| | **Holger Karl** |
| Abgabe der Arbeit: | **12. Juli 2018** |

## Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

## Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Mario Franke)
Paderborn, 12. Juli 2018

# Abstract

Due to the rapidly increasing number of vehicles participating in road traffic, scientists draw interest in increasing safety in road traffic. Technologies enabling Inter-Vehicle Communication (IVC) are a promising approach to make important information available for a larger number of vehicles in order to enhance collision avoidance. Today's vehicles already gather lots of data with several sensors. Research has shown that sharing the data among neighboring vehicles may have a positive effect on safety. Unfortunately, most research results are completely based on vehicular network simulators since real field tests are expensive and not well scalable. Furthermore, it is hard to provide reproducible results due to uncontrollable events. Vehicular network simulators provide precise simulation results even for large-scale road traffic scenarios but they only incorporate human imperfection based on some theoretical models. Driving simulators offer controllable road traffic scenarios and incorporate human imperfection. But most driving simulators lack the ability to simulate IVC or have some negative characteristics. Bridging both simulation domains promises a simulation tool capable of incorporating human imperfection as well as precise simulation of IVC. To combine advantages of both simulation domains, a network coupling the vehicular network simulator Vehicles in Network Simulation (Veins) and a driving simulator based on a vehicle dynamics model and a visualization framework is proposed. Evaluation results show the feasibility of such a coupled simulation framework as well as problems occurring when both simulation domains differ too much in the time domain. Furthermore, it is evaluated how well the system can recover from slight real-time deadline violations. Strategies avoiding such time differences are essential to provide a realistic driving experience and representable simulation results of real-world environments.

# Kurzfassung

Hinsichtlich des dauerhaften Bestrebens den Straßenverkehr sicher zu gestalten, bietet Kommunikation zwischen Fahrzeugen einen vielversprechenden Ansatz. Informationen, die zur Unfallvermeidung beitragen können, werden möglichst vielen Verkehrsteilnehmern zur Verfügung gestellt. Heutige Fahrzeuge erfassen schon viele solcher Informationen. Wissenschaftliche Ausarbeitungen haben gezeigt, dass das Verbreiten entsprechender Informationen einen positiven Einfluss auf die Sicherheit im Straßenverkehr haben kann. Leider basieren die meisten Ergebnisse auf Verkehrssimulationen, da echte Messungen teuer und schlecht skalierbar sind. Verkehrssimulationen liefern genaue Ergebnisse bezüglich der Kommunikation zwischen Fahrzeugen, sogar für Szenarien, die ganze Städte umfassen. Jedoch beziehen solche Simulatoren menschliche Ungenauigkeiten nur basierend auf theoretischen Modellen ein. Fahrsimulatoren bieten kontrollierbare Verkehrsszenarien und beziehen das Verhalten des Fahrers mit ein. Die meisten Fahrsimulatoren sind aber nicht in der Lage, Kommunikation zwischen Fahrzeugen zu simulieren oder weisen Nachteile in ihrem Konzept auf. Wenn man beide Simulatortypen verbindet, erhält man einen Simulator, der Kommunikation zwischen Fahrzeugen wie auch menschliche Ungenauigkeiten berücksichtigen kann. Um die Vorteile aus beiden Simulatortypen zu vereinen, ist in dieser Ausarbeitung ein Netzwerk aus dem Verkehrssimulator Veins und einem Fahrsimulator präsentiert. Die Auswertung zeigt, dass es grundsätzlich möglich ist, beide Simulatortypen zu verbinden, jedoch Probleme auftreten, wenn sich beide Simulatortypen zu sehr in der simulierten Zeit unterscheiden. Des Weiteren ist untersucht, wie gut sich ein solcher Simulator von Echtzeitverletzungen erholt. Strategien, um zu große Zeitunterschiede zwischen beiden Simulatortypen zu vermeiden, sind notwendig für ein realistisches Fahrverhalten.

# Contents

# Chapter 1

# Introduction

Car manufactures continuously try to improve the safety of their cars. With the help of several sensors modern vehicles already assist the driver in keeping track of the complex road traffic conditions. Such systems are called Advanced Driver Assistant System (ADAS). Popular examples are blind spot assistants and emergency brake systems. Both systems currently rely on vision-based or radar sensors which can measure the distance of the vehicle to objects in vicinity [1]. Such sensors can only detect dangerous situations in close vicinity due to their regional restricted sensing area. Moreover, only the driver of the vehicle detecting the dangerous situation is informed. Vehicles in vicinity have to detect the dangerous situation on their own.

A promising upcoming technology tries to enable communication in-between vehicles, especially cars, or between vehicles and infrastructure based on Radio Frequency (RF) technologies. In the research field, these technologies are called Inter-Vehicle Communication (IVC) or more general Car2X (C2X) communication [2]. With the help of IVC, vehicles can make other vehicles aware of themselves by regularly sending their position and possible trajectories. Based on this information cooperative ADAS can detect dangerous situation earlier and even without a direct line of sight. Additionally, all vehicles in the neighborhood are informed. Joerer et al. [3] show that IVC is able to prevent two vehicles from colliding.

But since ADAS can currently only warn drivers, the drivers have to avoid the dangerous situation. In the experiments by Joerer et al. [3] all vehicles were controlled by a simulator. Consequently, the impact of human drivers was neglected. In order to evaluate the impact of ADAS including human imperfection one could think of conducting field tests. On the one hand field tests provide the most representative results. But on the other hand field tests do not provide reproducible results due to constantly changing road traffic conditions. Moreover, field tests are expensive and can endanger test persons.

Another approach are simulations. Simulations can provide reproducible simulation results and do not endanger test persons. Further, simulations are better scalable than field tests because scenarios with many vehicles are expensive. Currently, driving simulators are commonly used tools to investigate human driving behavior [4], [5] and vehicular network simulators [6] are capable of simulating large-scale road traffic using IVC. Vehicular network simulators are able to simulate precise radio propagation models for IVC and accurate behavior of the road traffic.

To combine the advantages of both domains this thesis tries to couple a vehicular network simulator with a driving simulator. The resulting driving simulator is able to integrate human inputs to which the vehicular network simulator has to react. IVC-based ADASs assist the driver. Information and warnings generated by ADASs are displayed inside the visualization of the driving simulator. Humans controlling the driving simulator can react accordingly to the displayed information and can experience driving a vehicle capable of ADASs relying on IVC. This driving simulator can be used as tool for investigating the interaction between cooperative ADASs and humans who are still responsible for driving vehicles.

To get an overview of the state of the art of driving simulators and vehicular networking, a literature review is done. Based on this knowledge, a concept regarding the information exchange between the necessary components is developed and implemented. The implemented driving simulator is evaluated concerning subjective characteristics and quantitative metrics.

# Chapter 2

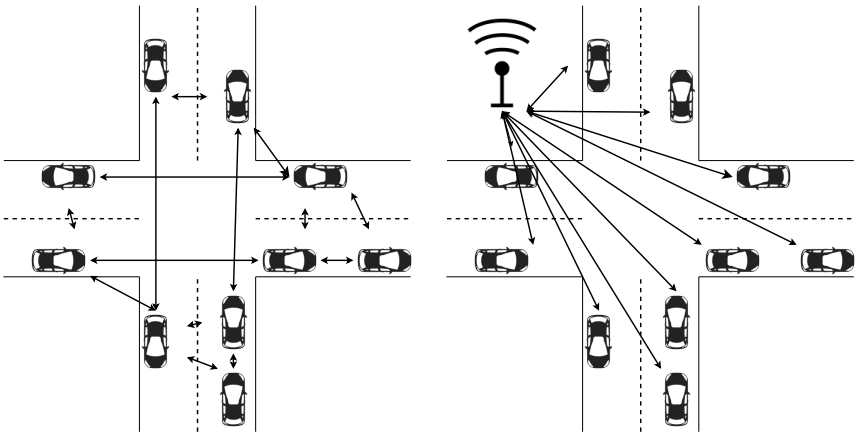
# Fundamentals

The proposed driving simulator needs to simulate IVC and ADASs. Both technologies are closely related. General information about IVC and ADAS and how to simulate both technologies are described in the next two sections. Basic concepts regarding driving simulators and coupling simulator platforms are presented in the following sections.

## 2.1 Traffic Information Systems

Traffic Information System (TIS) [7] is a generic term for systems which aim at distributing information in-between vehicles and roadside units. As shown by Joerer et al. [3], cooperative ADASs are individual TISs which aim at improving safety in road traffic in certain situations like collision avoiding at crossings. To achieve this goal, each vehicle shares its current state. By broadcasting basic messages, which are called *beacon messages*, all vehicles in vicinity become aware of the broadcasting vehicle. According to the European Telecommunications Standards Institute (ETSI) [8], a basic message describing the current state of a vehicle contains data fields like vehicle type, vehicle role and its size. Moreover, some geographical information like the vehicle's position, orientation, velocity and acceleration are added in order to make other vehicles aware of the sending vehicle. Based on such messages, the vehicle trajectories can be estimated and probable collisions can be avoided. These messages are distributed among all neighboring vehicles with the help of RF technologies enabling IVC. Several vehicles capable of IVC may form a Vehicular Ad Hoc Network (VANET) [2]. Communication inside VANETs can be accomplished by either Dedicated Short Range Communication (DSRC) or cellular-based approaches like Long Term Evolution (LTE) [9].

DSRC enables direct communication between vehicles which are in vicinity. A common DSRC protocol stack is IEEE 802.11p. It specifies the physical layer and



**(a)** DSRC **(b)** LTE

**Figure 2.1** – Different communication patterns of DSRC and LTE. Figure inspired by Rémy et al. [9].

the Medium Access Control (MAC). On top of this protocol stack, there can be several other protocols which provide services for higher layer applications. The most common ones are the American IEEE 1609.4/Wireless Access in Vehicular Environments (WAVE), the European ETSI ITS-G5 and the Japanese ARIB STD-T109.

In contrast to DSRC, cellular based communication requires a base station. The base station aggregates vehicle states and therefore has centralized knowledge about the traffic conditions in its area.

The described technologies have to be simulated precisely in order to provide simulation results representing real-world behavior of VANETs.

## 2.2 Vehicular Network Simulations

According to Sommer, German, and Dressler [6], simulating VANETs involves two main domains which are closely related. On the one hand a realistic driving behavior and on the other hand the IVC needs to be simulated. Using two separate simulators, one for each domain, ensures representative simulation results regarding realistic driving behavior and realistic IVC. There is a need for realistic road traffic simulators and computer network simulators which are capable to simulate RF technologies for communication. In addition, both simulators have to be able to provide their simulation state at regular intervals. The reason for this requirement is the close relation of both domains. Realistic simulation of IVC requires exact positions of senders and receivers. Thus, realistic driving behavior affects IVC. But IVC can affect the driving behavior of vehicles, too. For example, with the help of ADASs, vehicles

can avoid traffic jams if they receive the knowledge of a traffic jam early enough to find another route. Concluding, IVC affects the driving behavior of vehicles and vice versa during runtime. Consequently, VANETs simulators need to be coupled bidirectionally. For exchanging the simulation states of road traffic simulators and computer network simulators, there needs to be an interface. This interface has to ensure the synchronization between both simulators.

Most computer network simulators are event-based [10]–[12] and therefore the synchronization could be done in-between of processing two events. An event-based simulator processes only one event at a time, starting with the event which is nearest in time. One processed event can schedule several new events in the future which are going to be processed one after another. A timeline on which scheduled events are marked can be seen in Figure 2.2. The leftmost event in Figure 2.2 is processed next. Tasks which have to be executed when this type of event occurs are executed and if the event has to be repeated it is going to be scheduled again in the future. Further, the computer network simulator does not wait for the time an event is scheduled at. After finishing the processing of a single event the computer network simulator immediately processes the event which is next in time. Processing one event can require high computing power depending on the number of simulated vehicles. Consequently, the computation time of one event varies.

However, such a network of a road traffic simulator and a computer network simulator cooperating together provide realistic simulation results in the field of VANETs. The interactive driving simulator described in this thesis uses the vehicular network simulator Vehicles in Network Simulation (Veins). Veins [6] is an open source vehicular network simulation framework which couples the discrete event simulator Objective Modular Network Testbed in C++ (OMNeT++) [10] and the road traffic simulator Simulation of Urban MObility (SUMO) [13]. Veins provides realistic radio propagation models and common protocol stacks for IVC. SUMO is a large-scale road traffic simulator using validated vehicle behavior models. Furthermore, Veins and SUMO, both implement the Traffic Control Interface (TraCI) [14] protocol for exchanging simulation states via Transmission Control Protocol (TCP). Additionally, with the help of TraCI, it is possible to take over control of a vehicle of the simulation. There are commands provided by TraCI to set the position of a vehicle manually. These commands enable integrating a vehicle which is not controlled by SUMO. Hence, all SUMO-controlled vehicles have to react to the ego vehicle.

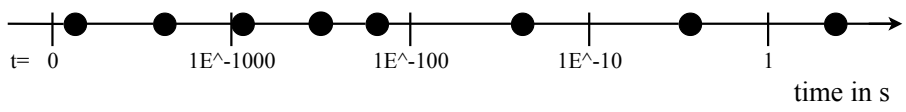



**Figure 2.2** – Timeline with scheduled simulation events.

The basic architecture of Veins, bidirectionally coupled with SUMO via TraCI, is illustrated in Figure 2.3. For synchronization purposes, Veins and SUMO are in a master-slave relationship in which Veins is the master and SUMO is the slave. Veins controls the time domain in both simulations. In the first part of a single simulation step, Veins simulates the whole network traffic in the simulation scenario. Commands influencing the behavior of vehicles are sent to SUMO which buffers all incoming commands. When Veins finishes its own simulation step, Veins invokes SUMO to simulate one simulation step. Figure 2.4 illustrates the described message exchange between Veins and SUMO. SUMO simulates a new simulation step and sends the new simulation state back to Veins. With the new simulation state, Veins simulates another simulation step before it invokes SUMO again. In such a synchronized manner, Veins and SUMO provide realistic simulation results for VANETs regarding driving behavior and radio propagation models.

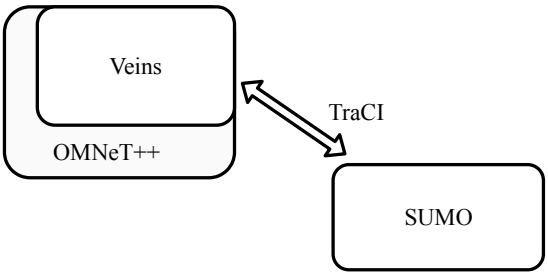



**Figure 2.3** – Architecture of the bidirectionally coupled simulators Veins and SUMO.

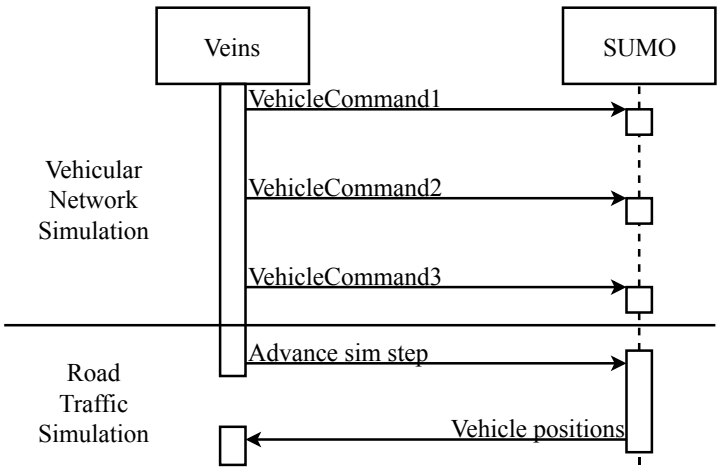



**Figure 2.4** – Message sequence diagram of one time step of the Veins framework. Figure inspired by Wegener et al. [14].

## 2.3 Driving Simulators

In contrast to vehicular network simulators, interactive driving simulators take human interactions into account in order to simulate a human-controlled vehicle and its interactions with other road traffic. In the following, the human-controlled vehicle is called ego vehicle and all other vehicles are called fellow vehicles. As shown by Kuhl et al. [15], a driving simulator gets inputs from a human in order to simulate the behavior of a single vehicle, the ego vehicle, and to visualizes it. To provide an immersive simulation, the researchers state that driving simulators have to accomplish a high degree of fidelity regarding the interface between human and simulator, the vehicle dynamics model and the visualization environment.

Common input devices and visualization techniques for driving simulators can vary from commercially available gaming steering wheels and simple monitors to complete vehicle cabins which provide a wide field of view and all driving controls of usual vehicles. Blana [16] contributes a survey of different driving simulators regarding costs and complexity. Examples of different scaled driving simulators can be seen in the publication of Abdelgawad et al. [17].

According to Kuhl et al. [15], the visualization is a complex aspect of driving simulators. Because the driver continuously interacts with the simulated environment, in terms of controlling a vehicle in it, the environment has to be a representative image of reality. Road traffic, buildings, traffic signs and traffic light systems have to be visualized in a realistic manner.

Vehicle dynamics [18] rely on physical laws in order to have realistic vehicle movements. Abdelgawad et al. [19] list several vehicle dynamics submodels. Their vehicle dynamics model consist of a horizontal dynamics model, a vertical dynamics model, a steering model, an engine model, a gearbox model, a differential model, a tire and wheel model and a brake model. Physical values which are important for the visualization are position, orientation and speed of the simulated vehicle.

Since the vehicle dynamics model depends on the interactive input, the driving simulator has to process its simulation steps including calculations for the vehicle dynamics model and visualization in real-time.

The term real-time is explained in the book written by Kopetz [20] in the following way: Computer programs process their tasks sometimes slower and sometimes faster depending on the complexity of the task and the computation load. If a computer program has to finish its tasks in real-time, it should finish the task to an exact point in time regarding the wall clock time. Real-time can be considered in two graduations. When a computer slightly misses a real-time deadline for a task but the computation result is still valid, the deadline is called a soft real-time deadline. But when the real-time deadline has to be held and otherwise the computation result is invalid, the deadline is called a hard real-time deadline. Violating real-time deadlines in

interactive driving simulators has a negative impact on the entire driving simulator because the inputs by the human cannot be displayed immediately. As a result, there is no realistic immersion driving a vehicle. A System which includes real-time tasks is called a real-time system.

A driving simulator is a real-time system which allows investigating human behavior in certain situations. Several research studies on human behavior in automotive domains were conducted with the help of driving simulators. Tateiwa, Nakamura, and Yamada [4] investigated the driver awareness of pedestrians at intersections. The researchers collected data on situations in which drivers were unaware of pedestrians crossing a street at an intersection with the help of driving simulators. They aim at estimating the awareness of the driver depending on the vehicle's behavior. In case the driver was indicated as unaware although there is a pedestrian on the street, the driver was warned. They claim to have a positive effect on the driver using the warning system.

Another study by Sonoda and Wada [5] investigated the driver trust in vehicular self-driving systems. The researchers investigated a scenario in which a vehicle equipped with a vehicular self-driving system has to overtake a motorbike. Three different overtaking manoeuvres were compared. The driver was equipped with vibrotactile displays to provide spatial information on the situation awareness of the vehicular self-driving system. The study showed the vibrotactile displays and the manoeuvre affects the trust of the driver in the vehicular self-driving system.

Both studies demonstrate the capabilities of driving simulators to provide scientific findings in experiments involving humans. Moreover, it is possible to investigate dangerous situations without endangering test persons. But most driving simulators neither have large-scale road traffic nor support IVC for enabling cooperative ADASs. In order to support IVC and ADASs driving simulators can be coupled with vehicular network simulators.

## 2.4 Coupling Simulators

Different simulators are coupled in order to benefit from precise simulation results in several domains. For example, Sommer, German, and Dressler [6] coupled the network simulator OMNeT++ and SUMO to simulate precise large-scale road traffic and realistic RF propagation models. Currently, a highly investigated research field of coupling simulators is Hardware-in-the-Loop (HIL) systems. A HIL system aims at integrating a real hardware component, often a subsystem like an engine of a vehicle [21], into a computer-based simulation emulating the subsystem's environment, for example, the whole vehicle and its environment. The proposed interactive driving simulator could be described as a Human-in-the-Loop (HuIL) system because

the human's behavior of driving a vehicle can be investigated inside a simulation environment. Both, HIL and HuIL systems, have quite similar requirements in order to provide valid simulation results because in both cases the simulation tools have to incorporate input data generated by physical input quantities.

According to Swanson et al. [22], there are some requirements for building HIL or HuIL systems, especially for driving simulators. The in-the-Loop system has to have an interface for gathering sensor data. The sensor can either be a camera-detection system, a steering wheel or any other device operated by a human for HuIL systems. The data gathered by the sensor has to be incorporated in the simulation with the help of an interface building the bridge between the physical subsystem and the simulation. Due to the sensors constantly generating data, in-the-Loop systems have to provide milliseconds level latencies in-between components responsible for processing sensor data. The low latency level is necessary for providing a realistic feedback for the tested subsystem or the human.

Moreover, the latencies in-between all components are not the only time critical aspect of coupling simulators. As stated by Buse et al. [23], the simulation step length of the used components is a key factor for the granularity of the simulation results. Further, the synchronization between all components has to be ensured even if the simulation components have different simulation step lengths.

Several other researchers have tried coupling different simulators and integrating interactive inputs. Thus, there are some publications which try to couple vehicular network simulators with driving simulators or other human input devices. Their approaches are presented in the next section.

## 2.5 Related Work

The investigated research field covers vehicular networking, ADASs and coupling simulators. The need for coupling different simulators in order to benefit from each of them was recognized by several researchers. For example, Sommer, German, and Dressler [6] bidirectionally coupled the discrete event simulator OMNeT++ and the road traffic simulator SUMO. They investigated the impact of IVC on road traffic and vice versa. As a result, they proclaimed to simulate vehicular networks with the help of bidirectionally coupled simulators due to the influence of IVC on road traffic and vice versa. Veins is a state of the art vehicular network simulator and is not designed to visualize its simulation in 3D environments or to interactively integrate a vehicle based on human decisions.

The approach by Guan, De Grande, and Boukerche [24] synchronizes a road traffic simulator with a 3D visualization. Their chosen visualization framework is Unity3D, since it incorporates a simplistic mobility model which controls the

simulated vehicles. Both components are coupled to exchange systems states in real-time, but their system does not simulate IVC and the system is not able to incorporate real-time inputs.

To incorporate real-time inputs and to test ADAS, Abdelgawad et al. [17] and Gruyer et al. [25] followed quite similar approaches. Both research groups implemented complex vehicle dynamic models. Additionally, both research groups used immersive visualization techniques. Gruyer et al. [25] used a head-mounted dual screen and the controls of a real vehicle. Abdelgawad et al. [17] used a driving simulator platform on which whole vehicle cabins are mounted. Due to the complex vehicle dynamic models and the immersive visualization techniques both approaches provide a quite realistic driving experience. But both approaches are lacking of large-scale road traffic and TISs. They only investigate ADASs which do not rely on IVC. Regarding road traffic, Abdelgawad et al. [17] try to simulate road traffic but their approach is not able to simulate large road networks and Gruyer et al. [25] do not simulate road traffic at all.

Griggs et al. [26] developed a platform in which a real vehicle is integrated in SUMO. Consequently, simulating large road networks is possible thanks to SUMO. The data exchange between the vehicle and SUMO is achieved by an Android smartphone. The smartphone sends the vehicle state via a 3G UMTS network to a computer running SUMO. Additionally, the smartphone acts as an interface for the driver as it displays the simulated state of the road traffic around the real vehicle or feedback from ADASs. But the system is lacking realistic or simulated VANET characteristics. There is no IVC. Since the researchers only used SUMO as a single simulator, they did not have to cope with challenges regarding coupling real-time based simulators with discrete time-based simulators.

Solving these challenges is the aim of the concept developed by Buse et al. [23]. The presented Ego Vehicle Interface (EVI), acting as a broker, synchronizes a real-time based HIL system with Veins. The real-time based system sends clock tick messages in order to invoke Veins to simulate another time step. As they propose, it is possible to exchange the HIL system with a driving simulator.

Michaeler and Olaverri-Monreal [27] try to incorporate road traffic, vehicular networking and a user-controlled vehicle. The developed driving simulator is based on the Unity3D visualization framework. They proclaim combining Unity3D with simulators like OMNeT++ or SUMO is not suitable because the vehicular network simulations steps have to be finished before the driving simulator can advance. Therefore, all calculations regarding road traffic and vehicular networking are processed within Unity3D. Thus, their approach is lacking state of the art IVC lower layer network protocol stacks like IEEE 802.11p and exact radio propagation models.

In order to provide more precise simulation results, Prendinger et al. [28] developed a multi-user driving simulator named Distributed Virtual Environment (DiVE).

DiVE is based on a server-client architecture. Each client is an independent driving simulator controlled with the help of steering wheels and pedals or different input devices. The used visualization framework is Unity3D. On the server's side, there is a road traffic simulator and a vehicular network simulator based on OMNeT++. The clients and the server are synchronized. But on the client's side, there is no vehicle dynamics model. The only physics model is the one provided by Unity3D. Hence, the immersion might be affected negatively.

For maximum immersion, Hou et al. [29] build a three-in-one simulator. It consists of the road traffic simulator Paramics, the Network Simulator 2 (NS2) and the driving simulator developed at the University at Buffalo. Paramics implements car-following and lane-changing modes and NS2 is able to simulate IVC. The driving simulator provides a six-degree-of-freedom motion platform and four displays to visualize the 3D environment. But since the driving simulator is only developed at the University at Buffalo, it might be difficult to exchange the driving simulator in order to run the three-in-one simulator at different places.

Table 2.1 provides an overview of the capabilities of all presented systems. It shows most of the systems do not fulfill the claimed requirements or have disadvantages over the proposed system. Figure 2.5 illustrates the current state of the art in a more intuitive way. As showed before, there are already systems which are capable of a subset of the claimed functionality. These systems are represented by the intersection of two circles. The grey point in the middle represents the technology gap which this thesis tries to fil. By building the proposed interactive driving simulator for C2X scenarios all circles should intersect.

Furthermore, after investigating the related work, most publications do not evaluate the actual simulator or system. In the presented evaluation chapters, there

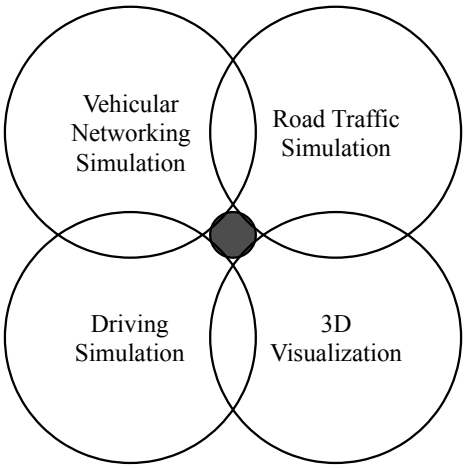



**Figure 2.5** – Illustration of already intersecting research domains. The grey point illustrates the technology gap this thesis tries to fill.

| Publication | 3D Visualization | Road Traffic Simulator | Vehicular Network Simulator | Driving Simulator |
|---|---|---|---|---|
| Sommer, German, and Dressler [6] | No | SUMO | OMNeT++/Veins | No |
| Guan, De Grande, and Boukerche [24] | Unity3D | Own traffic model | No | No |
| Abdelgawad et al. [17] | Unity3D | No | No | MATLAB/Simulink |
| Gruyer et al. [25] | Yes | No | No | Own physics model |
| Griggs et al. [26] | Real vehicle | SUMO | No | Real vehicle |
| Buse et al. [23] | Extendible | SUMO | Veins | Extendible |
| Michaeler and Olaverri-Monreal [27] | Unity3D | Built in Unity3D | Built in Unity3D | Built in Unity3D |
| Prendinger et al. [28] | Unity3D | Own traffic model | Based on OMNeT++ | Unity3D |
| Hou et al. [29] | Yes | Paramics | NS2 | Platform based |

**Table 2.1** – Overview of different simulator platforms.

are applications described and evaluated which run within the system, but there are no evaluation results of the simulation system itself regarding scalability or performance.

To cope with all the presented problems and challenges, this thesis proposes an architecture for an interactive driving simulator for C2X scenarios built on different simulation tools and a visualization framework. The concept of this thesis and its implementation are described in the following two chapters and the performance of the implemented interactive driving simulator is presented in the evaluation.

# Chapter 3

# Concept

The general idea in this thesis is to couple simulators of different research fields in order to benefit of the advantages each simulator type provides. As described earlier, the vehicular network simulator Veins follows this idea, too [6]. Veins is coupled bidirectionally with SUMO. A driving simulator capable of IVC can be build by following this idea, too.

The necessary components are a vehicle dynamics model and a 3D visualization building the driving simulator part and simulators responsible for IVC and large-scale road traffic. The vehicle dynamics model is responsible for providing a realistic ego vehicle behavior based on the inputs by humans and an accurate physics model. An immersive 3D visualization is needed in order to provide a good feedback for humans. To have realistic road traffic, a road traffic simulator has to be able to integrate the ego vehicle and has to incorporate the behavior of the ego vehicle for simulating reactions of the other road traffic. The IVC has to be simulated by a vehicular network simulator which has to take precise radio propagation models and the standardized protocol stacks into account. Given that, each component needs data provided by another component, each component needs to be able to communicate bidirectionally with each other as implemented by Buse et al. [23].

Figure 3.1 illustrates the basic information flow between all components. The 3D visualization and the vehicular network simulator need the positions of all vehicles provided by the vehicle dynamics model and the road traffic simulator. Furthermore, the road traffic simulator has to be able to integrate the ego vehicle controlled by the vehicle dynamics model, so that all fellow vehicles can react to the ego vehicle. The ego vehicle is controlled with the help of a steering wheel or another appropriate input device which is connected to the vehicle dynamics model. The vehicle positions are important for displaying all vehicles at the right place inside the visualization and for correctly simulating IVC. Based on the IVC a cooperative ADAS can warn or inform the driver of the ego vehicle about certain traffic situations. Since an implementation

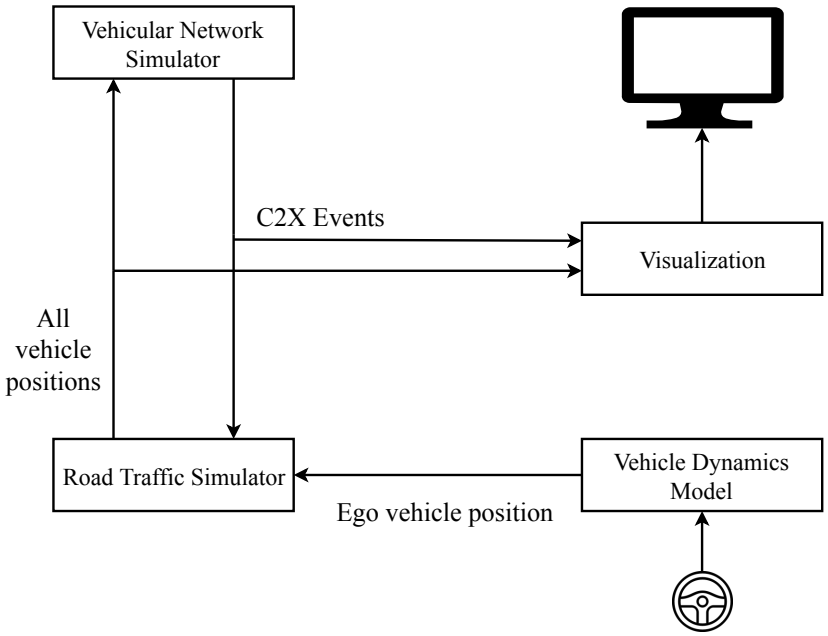



**Figure 3.1** – Basic information shared between all components.

of a cooperative ADAS has to be done inside the vehicular network simulator, the decision to notify the driver is made by the vehicular network simulator. Consequently, the visualization needs the knowledge of when to visualize a notification by the vehicular networking simulator. For synchronizing all components and managing the information sharing, Buse et al. [23] present the EVI. The implementation of the interface synchronizing all components for the interactive driving simulator is described in the next chapter.

# Chapter 4

# Implementation

The proposed concept of the developed interactive driving simulator consists of a modular network enabling information sharing among all components. For simulating IVC and large-scale road traffic, this approach uses the open source simulators Veins 4.6 which already couples OMNeT++ 5.1.1 and SUMO 0.30.0. The vehicle dynamics of the ego vehicle is simulated by a MATLAB/Simulink R2015b simulation and the driving simulator environment is visualized with the help of the free visualization framework Unity3D 5.6.0f3. According to the approach proposed by Buse et al. [23], there needs to be an interface which orchestrates all components. The proposed interface in this thesis is called Yet Another Ego Vehicle Interface (YAEVI).

YAEVI couples Veins and SUMO with Unity3D because there is already a bidirectionally connection between Veins and SUMO [6]. A steering wheel to control the position and orientation of the ego vehicle is connected to the MATLAB/Simulink simulation. The MATLAB/Simulink simulation sends updates of the ego vehicle state to Unity3D in real-time. The ego vehicle update is forwarded to SUMO in order to incorporate it into the large-scale road traffic simulation. Since the vehicle dynamics model can only send ego vehicle updates and cannot get information regarding the fellow vehicles, it is not possible to simulate collisions between the ego vehicle and fellow vehicles. Figure 4.1 illustrates the basic network architecture. In contrast to Figure 2.3, the direct connection between OMNeT++/Veins is interrupted by the YAEVI. The YAEVI is responsible for exchanging all network messages in-between OMNeT++/Veins, SUMO and Unity3D and its synchronization.

## 4.1 Networking Tools

For exchanging simulation states, there needs to be protocols which handle the communication in-between all components and which specify the structure of the network messages. The network protocols, responsible for sending and receiving

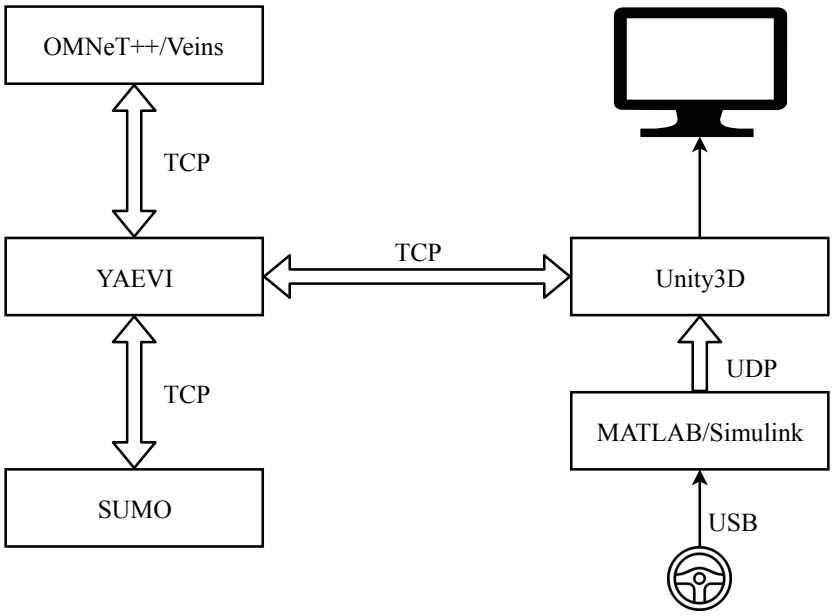



**Figure 4.1** – Basic architecture of the implemented interactive driving simulator.

data packets in-between all simulators, can be either reliable (TCP) or unreliable (User Datagram Protocol (UDP)). The network which handles the message exchange is realized with the help of ZeroMQ [30]. ZeroMQ is an open source meta transport layer providing several standard transport layer protocols in combination with predefined communication patterns. The different components are connected via TCP using a publish-subscribe (PubSub) communication pattern provided by ZeroMQ. TCP may cause problems because of the reordering of messages or retransmissions causing violating real-time deadlines. But, TCP ensures the receiving of all network messages in the order as they were sent. Therefore, no simulator has to implement strategies for dropped network messages. The PubSub cummunication pattern provides publisher and subscriber sockets. Publisher sockets can only send network messages and subscriber sockets can only receive network messages, but both socket types work without any synchronization of each other. Consequently, their advantage is the ability to continuously send and receive network messages. Figure 4.2 illustrates the PubSub communication pattern.

The structure of the network messages exchanged by OMNeT++/Veins and SUMO is specified by the TraCI protocol, see Figure 2.3 and Figure 2.4. The TraCI protocol specifies message structures for exchanging simulation states of vehicle, traffic lights and commands affecting the simulation. But, TraCI lacks commands needed for the 3D visualization in order to visualize driver notifications because it was not build for synchronizing simulators with 3D visualizations. In this approach,

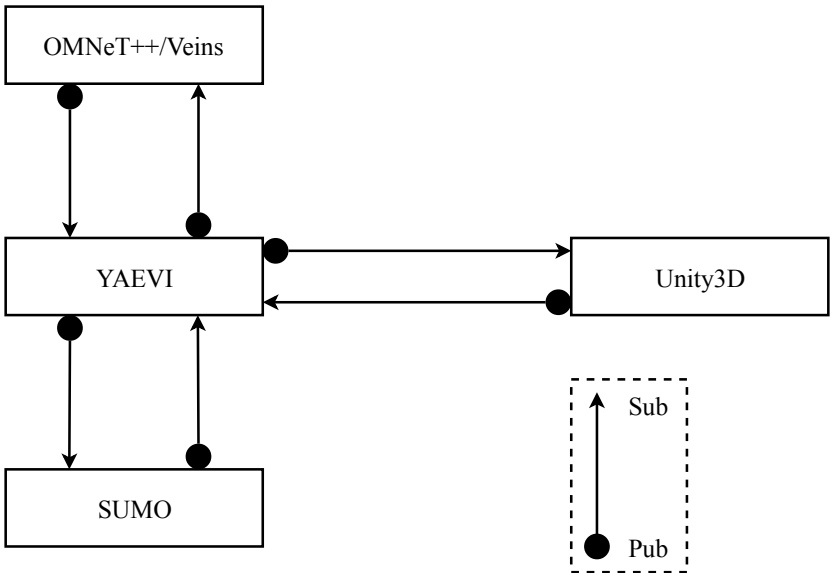



**Figure 4.2** – Architecture with PubSub communication pattern. Points illustrate publisher sockets. Arrowheads illustrate subscriber sockets.

the structure of the network messages exchanged between Veins, YAEVI and SUMO is based on TraCI because Veins and SUMO already implement TraCI. To compensate the lack of 3D visualization commands, TraCI needs to be extended. Since TraCI commands are encoded as bytes, TraCI is hard to extend and maintain. Furthermore, the number of different commands is limited. A tool which provides high extendibility and maintainability is Google Protocol Buffers[1]. Google Protocol Buffers is an open source programming language independent and extendible tool for serializing data. Google Protocol Buffers is able to generate an Application Programming Interface (API) for most common programming languages out of .proto files which specify Google Protocol Buffers messages and their data fields. The proposed solution is to tunnel whole TraCI commands as a single data field inside a Google Protocol Buffers message, as illustrated by Figure 4.3. Additional custom commands can be easily extended by adding new data fields to the Google Protocol Buffer message. Currently, there are two custom commands added. First, there is a *generic information command* in order to display text based information inside the visualization. Second, there is a *generic warning command* which has a text data field and a data field indicating the importance of the warning.

The structure of the network messages exchanged between YAEVI and Unity3D is completely based on Google Protocol Buffers messages. There are four different message types which are developed by Buse et al. [23].

- *Session messages* are used to initialize and shut down the visualization.

[1] https://developers.google.com/protocol-buffers

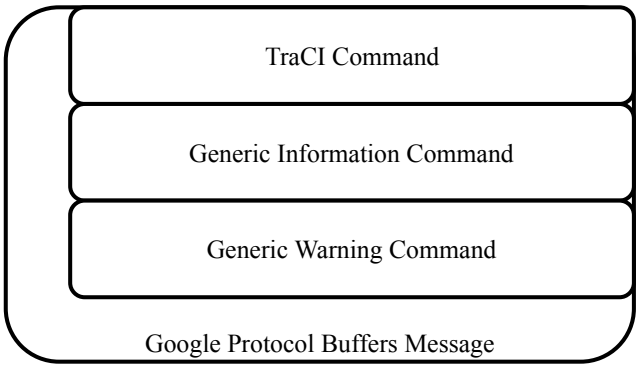



**Figure 4.3** – TraCI tunneled with Google Protocol Buffers.

- *Vehicle messages* provide states of all vehicles regarding vehicle id, position, orientation, vehicle speed and road id.
- *Traffic light messages* provide states of all traffic light systems regarding traffic light signals and the junction where they are at.
- *Visualization messages* are used to visualize notifications and warnings generated by Veins

By using their message protocol the EVI and the YAEVI can be exchanged easily and the visualization can be used by both interfaces. For ensuring to send and receive the right message type to the right point in time all components have to be synchronized.

## 4.2 Synchronization of Components

Since all network messages in the network have to be forwarded by the YAEVI, it is responsible for synchronizing the different simulators. The behavior of the ego vehicle is simulated by a MATLAB/Simulink simulation in a real-time domain. The fellow vehicles, representing other road traffic, are simulated by SUMO. Veins simulates the IVC. Both, Veins and SUMO simulate events in a discrete time domain without any real-time guaranties. The real-time duration of a single simulation step of Veins or SUMO may differ depending on the complexity of the current simulation step. As a result, one simulated time step can take longer or shorter than the duration of the time step in real-time. According to Buse et al. [23], the duration of one simulated time step is a key factor for the granularity of the simulation results. They propose that 100 ms is a good trade-off between simulation granularity and computational effort. In this approach one simulation step of Veins and SUMO takes 100 ms, too, in order to update the fellow vehicle state in the visualization fast enough. With a larger synchronization interval the difference of the positions of the fellow vehicles between SUMO and the visualization would be too big.

Coupling Veins and SUMO with the real-time based visualization requires some kind of synchronization because the MATLAB/Simulink simulation has to run in parallel. It should not be stopped due to its real-time calculations. When the vehicle dynamics model would be stopped the immersion of the driving simulator would be affected negatively because the inputs provided by the human would be displayed with a noticeable delay. The synchronization has to ensure that Veins and SUMO do not simulate faster than the real-time based simulation. Due to the MATLAB/Simulink simulation is directly connected to Unity3D, the clock provided by Unity3D is assumed to be the real-time clock all other simulators have to be synchronized on. By Unity3D sending clock tick messages, Veins and SUMO can be synchronized with Unity3D. The discrete time-based simulators invoke one time step every time a clock tick messages arrives at the YAEVI. If a simulation step of the discrete time-based simulators takes longer than real-time, a mechanism, which enables catching back up to real-time, can mitigate slight real-time violations.

As illustrated in Figure 4.4, a single simulation step of the system invokes a predefined sequence of message exchanges. First, the YAEVI waits for Veins sending a command requesting to advance the system-wide simulation step. Next, the YAEVI waits for Unity3D sending a clock tick message. Unity3D generates clock tick messages everytime when the wall clock, all components are synchronized on, exceeds 100 ms. A generated clock tick message is inserted in a list. In order to not desynchronize, clock tick messages are only sent when the YAEVI provided the fellow vehicle states of the last simulation step. When the YAEVI is too slow to provide fellow vehicle states the list managing the clock tick messages grows. Consequently, when the driving simulator holds the real-time deadlines the clock tick

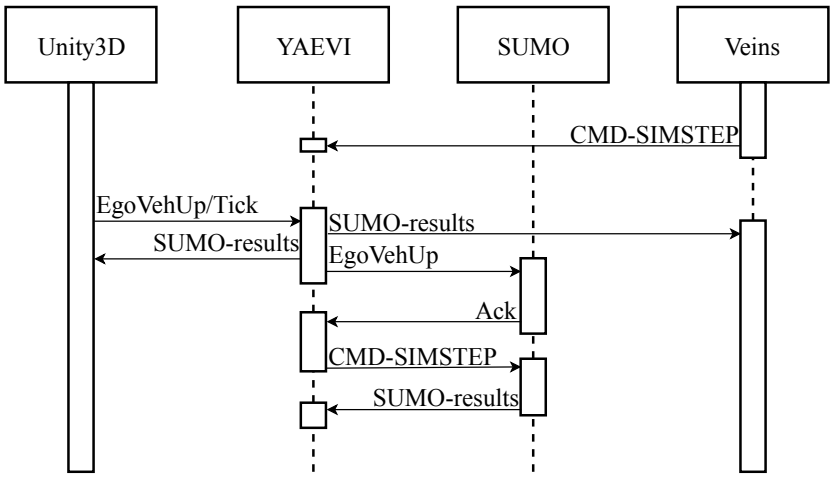



**Figure 4.4** – Message sequence diagram of synchronized messages of a single simulation step.

message list in Unity3D is not larger than one because the clock tick messages are sent immediately. The clock tick messages from Unity3D also include an ego vehicle update for SUMO. After the YAEVI received the clock tick message, it immediately sends the simulation state of SUMO to Veins, a fellow vehicle update to Unity3D and an ego vehicle update message to SUMO. SUMO acknowledges the receiving of the ego vehicle update message. To conclude a single simulation step, the YAEVI sends a message to make SUMO advance its simulation by 100 ms and to provide new states of the fellow vehicles.

Additionally, all components are coupled synchronously. This design decision has a disadvantage. The whole system misses a real-time deadline when either Veins or SUMO misses a real-time deadline. This can happen because all components have to wait for the component missing the real-time deadline. If Veins misses the real-time deadline, but SUMO meets it, the fellow vehicles will not be displayed in a real-time manner anyway because the whole driving simulator is slowed down by waiting for Veins finishing its current simulation step. When the described case frequently occurs it can lead to slowed down or jumping fellow vehicles in the visualization.

Messages, which have to be sent due to IVC events, are sent asynchronously from Veins to SUMO or to Unity3D. This means, corresponding messages are sent immediately when the event is processed by Veins. The messages do not have to wait to be sent until the simulation step of Veins is advanced. Such message types are, for example, the visualization messages for Unity3D or messages instructing SUMO to reroute a fellow vehicle.

In order to send Veins the simulation state of SUMO immediately, SUMO's simulation state is one simulation step ahead of all other components. SUMO is instructed to simulate two simulation steps at once in the first system-wide simulation step. The first simulation state from SUMO is forwarded to Veins. The simulation state of the second simulation step is cached in the YAEVI. In the next system-wide simulation step YAEVI can immediately send the cached simulation state from SUMO to Veins. This trick shortens the time Veins has to wait for the new simulation state provided by SUMO.

The overall network structure of the developed interactive driving simulator is now presented. In an ideal case the human steering the ego vehicle does not recognize any network specific events. The human controls the ego vehicle with the help of a steering wheel and gets feedback by the visualization. Both aspects are explained in the next two sections.

## 4.3 Vehicle Dynamics Model

As input device for the driving simulator described in this thesis, a Logitech Driving Force GT from 2007 is used. It supports a 900° steering wheel angle which means it takes 2.5 turns from lock to lock. Furthermore, it supports force feedback, has throttle and brake pedals and a gearshifter. It is plugged in via USB. The Logitech Driving Force GT is shown in Figure 4.5.

Nevertheless, the best devices for interfacing between human and driving simulator do not provide a realistic immersion if the driving simulator itself does not take physics into account. Realistic movements and orientation of the ego vehicle based on physical laws enhance the immersion of driving a real vehicle. The parameters for the physical calculations rely on the human controlling the steering wheel. For affording realistic physics for the interactive driving simulator, the output signals of the steering wheel are used by a MATLAB/Simulink simulation implemented by Abdelgawad et al. [19]. Based on the steering wheel angle, the pedal states and the gear the MATLAB/Simulink simulation provides position, rotation and speed of the ego vehicle. These values are sent via UDP to Unity3D. In contrast to all other network connections, see Figure 4.1, this connection is realized with UDP because it is not that critical if a single UDP message is dropped. The MATLAB/Simulink simulation sends ego vehicle updates fast enough that a single dropped update is not recognizable by humans.

The ego vehicle is visualized only based on the mentioned values provided by the MATLAB/Simulink simulation. But the ego vehicle itself is a quiet small part of

**(a)** Steering wheel **(b)** Pedals

**Figure 4.5** – Steering wheel and pedals used for the interactive driving simulator.

the whole visualization. The visualized ego vehicle is only a single vehicle inside a complete environment which has to be visualized.

## 4.4 Scenario Generation

According to Kuhl et al. [15], driving environments have to be as realistic as possible. With the help of the used components, it is possible to visualize 3D environments based on OpenStreetMap (OSM) [31] data. Consequently, the visualization can illustrate exact 3D environments imaging real-world cities. Currently, it possible to visualize real-world street networks including exact positions of buildings. A tool called *netconvert* provided by SUMO is able to generate SUMO network and SUMO polygon files out of OSM files. The generated SUMO polygon files are needed by Veins, too, because buildings have a significant influence on radio propagation models. SUMO network files save all information regarding the street network including traffic light systems. Unity3D can parse these SUMO network and SUMO polygon files and builds streets, buildings and traffic light systems in the 3D environment. Figure 4.6 shows a street network of the north of Paderborn[2] provided by OSM, the corresponding 3D environment generated by Unity3D and the environment integrated in SUMO. Since the described 3D environment only visualizes the street network and buildings it lacks dynamic objects like fellow vehicle traffic and traffic light systems.

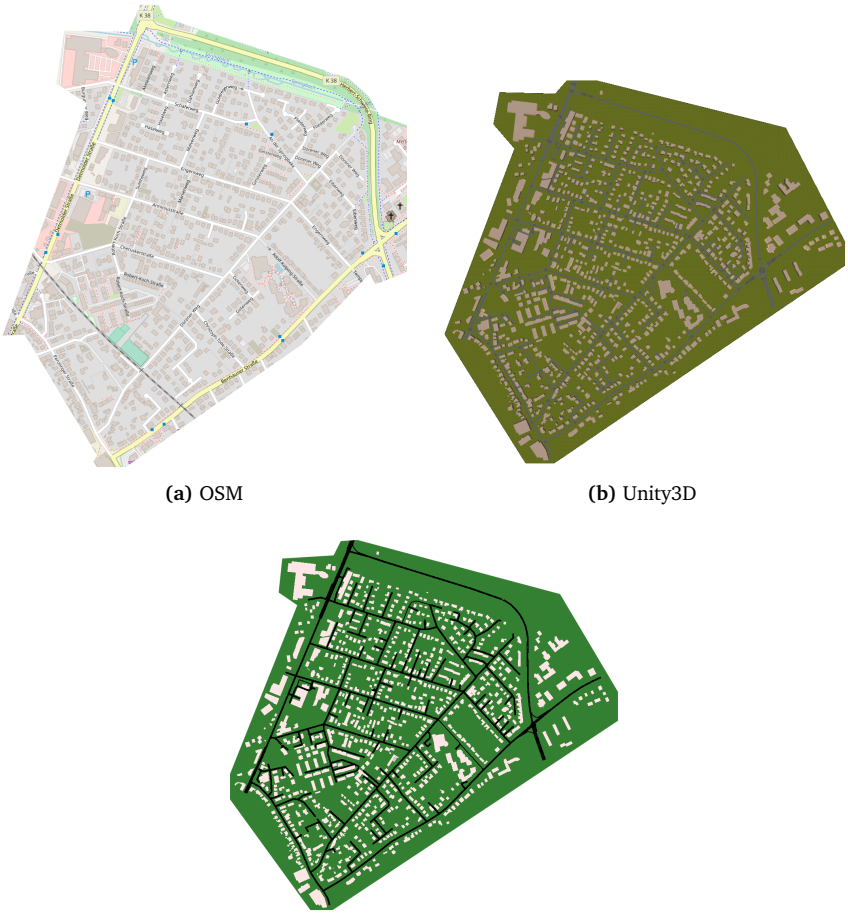
## 4.5 Visualization

Moreover, Kuhl et al. [15] state, the 3D rendering quality should be as high as possible because the displayed state of the ego vehicle is the main feedback for the human giving the inputs. The widely-used visualization framework Unity3D is responsible for visualizing driving environments of the interactive driving simulator developed in this thesis.

Traffic light systems are visualized by Unity3D, too, but they are simulated by SUMO. Every simulation step Unity3D gets a message which describes the signal state of all simulated traffic light systems. With the information provided by SUMO, Unity3D only has to set the traffic lights according to the received message. Because SUMO is simulating the traffic light system states it is easily possible to change the signaling to the exact behavior of the traffic light system of the real-world. But traffic light systems without fellow vehicle traffic are useless.

The fellow vehicle traffic is implemented quite similar to the traffic light systems. Every simulation step Unity3D gets a vehicle update message containing the state of

[2] https://www.openstreetmap.org/#map=16/51.7270/8.7674



**(a)** OSM **(b)** Unity3D

**(c)** SUMO

**Figure 4.6** – Top down view of a neighborhood of the north of Paderborn provided by OSM and the corresponding environment visualized by Unity3D and SUMO.

all simulated vehicles. This means, Unity3D gets every 100 ms an update about all fellow vehicles and their positions. If Unity3D only sets all fellow vehicles to their corresponding position, all fellow vehicles would jump through the environment because there are only 10 updates per second. To compensate the low update rate Unity3D interpolates between two fellow vehicle updates. Unity3D already provides functions for interpolation. Such a function gets the current position of a fellow vehicle, its destination, a speed vector and the time to arrive at the destination. Since the update interval is 100 ms, the fellow vehicles should arrive after 100 ms at their destination. When fellow vehicle updates arrive at Unity3D it calls the interpolation function for all fellow vehicles to provide smooth fellow vehicle movements.

The last part of the visualization is to display the generic warning and generic information messages sent by Veins. Figure 4.7 shows a warning message visualized


**Figure 4.7** – Warning message visualized in the lower right corner.

in Unity3D. If an ADAS notices a situation of which the driver has to be informed or warned the visualization has to notify the driver. When Unity3D receives a generic information message a canvas in the right lower corner of the display pops up and displays the text information. After a few seconds the canvas disappears. A generic warning message triggers the canvas to pop up, too, and a warning sound rings and a red warning light is activated. After a while the warning indications disappear.

The described visualization builds in combination with the vehicle dynamics model a simplistic driving simulator but with the addition of simulated large-scale road traffic and precise simulation of radio propagation models for IVC. Moreover, it is possible to notify the driver about road traffic conditions based on IVC events. The performance and characteristics of the interactive driving simulator and its network architecture are evaluated in the next chapter.

# Chapter 5

# Evaluation

To show the feasibility of the proposed concept and its implementation, this chapter evaluates the interactive driving simulator. The evaluation consists of two parts. The first part highlights a few subjective impressions recognized during the evaluation. This means, there are characteristics described which cannot be properly measured because these characteristics depend on the perception of the human controlling the driving simulator. For example, the feeling of the velocity of the controlled vehicle or the behavior of the fellow vehicles belong to these characteristics. The second and main part of the evaluation is about the performance of the developed network architecture regarding scalability and real-time criteria. To better understand the evaluation results, the chapter starts with a description of the measurement setup and tools.

## 5.1 Measurement Setup and Tools

As described in the fundamentals chapter, Kuhl et al. [15] state having an as accurate as possible driving environment improves the immersion. Therefore, a real-world driving environment fulfills the requirement best. The OSM provides the required data for real-world driving environments. Any OSM-based scenarios can be generated as it is described in the scenario generation section. In this evaluation, a 3.5 km round course in a neighborhood in the north of Paderborn is used as test course. The test course is visualized in Figure 5.1 with the help of the blue line. The ego vehicle route starts at the most southern point of the blue line and follows the blue line counterclockwise back to the starting point. To provide reproducible and comparable simulation results for this evaluation, the ego vehicle route is recorded. These recordings are called ego vehicles traces. There are ego vehicle traces for 5 to 25 simulated vehicles with a stepwidth of 5 and for 25 to 200 simulated vehicles with a stepwidth of 25. Additionally, for each number of simulated vehicles there are 3 ego

**Figure 5.1** – Test course used for the evaluation. The red dot indicates the start position of the ego vehicle.

vehicle traces recorded with different SUMO seeds in order to differently initialize the random number generator of SUMO. Consequently, there are 36 different ego vehicles traces. The ego vehicle traces build the base for this evaluation because it is not possible to drive the ego vehicle exactly in the same way for each experiment. The different number of simulated vehicles affects the driving behavior of the ego vehicle as well. With the help of the ego vehicle traces, each experiment can be conducted several times with exactly the same behavior of the ego vehicle and the corresponding reaction of the fellow vehicles. A single ego vehicle trace takes between 8 min to 10 min. Consequently, the evaluation is based on ego vehicle traces lasting about a total of five and a half hours. The total driven distance is 126 km.

The recording of the ego vehicle traces is done with the tool *Wireshark*. Wireshark is able to record UDP streams and can save them in .pcap files. Since the ego vehicle states are updated by the vehicle dynamics model which is directly connected to the visualization via UDP, Wireshark only has to save the UDP packets sent by the vehicle dynamics model. In order to replay a recorded ego vehicle trace, the tool *tcpreplay* is used. Tcpreplay is able to replay the recorded .pcap files. The visualization notices no difference whether the ego vehicle states are updated by the ego vehicle's dynamics model or by a saved ego vehicle trace.

Due to the network architecture, single components of the network can run on different computers. In the evaluated architecture, the 3D visualization is running on a Windows 7 computer. Veins, SUMO, YAEVI and tcpreplay are running on a Ubuntu 16.04 LTS computer. The system specifications are highlighted in Table 5.1. As shown by Buse et al. [23], SUMO needs around 11 ms to provide results of a new simulation step in their scenario as worst case approximation. This is far less than

| Component | Windows 7 PC | Ubuntu 16.04 LTS PC |
|---|---|---|
| CPU | Intel Core i7-3770K @ 3.5 GHz | Intel Core i7-7700K @ 4.2 GHz |
| RAM | 16 GB | 16 GB |
| GPU | Nvidia GTX 650 Ti | no dedicated GPU |

**Table 5.1** – System specification of the computers used for the evaluation.

the synchronization interval of 100 ms. According to Obermaier and Facchi [32] and Buse et al. [23], OMNeT++ will probably be the bottleneck in the developed driving simulator architecture. The more interesting question is in which manner the driving simulator network behaves with different computation times of a single time step and whether it is possible to compensate real-time deadline violations.

To investigate the performance of the driving simulator network with different OMNeT++/Veins time step computation times, a dummy application is implemented in Veins. This dummy application is called *dummy-app*. With the help of dummy-app it is possible to control the computation time of a single time step of Veins. The dummy-app does nothing except of waiting for the adjusted Veins time step duration to exceed. By setting the duration of a single time step of Veins, it can be investigated how the driving simulator's behavior depends on the duration that Veins takes for simulating a single time step. Furthermore, the simulation results measured with dummy-app are independent of a specific Veins implementation or scenario because the duration of a single time step does not depend on functions responsible for processing radio propagation models. In this thesis, the duration of a single time step of Veins is called Veins load. The system-wide synchronization interval is 100 ms. When the duration of a single Veins simulation time step is adjusted to 40 ms, the Veins load is set to 40% of the synchronization interval of 100 ms.

In this thesis three different experiments are conducted to investigate the driving simulator's performance.

- *Static Veins Load:* The Veins load is set to values from 10% to 120% for each different number of fellow vehicles. The Veins load is constant during the complete simulation time. With the help this experiment, the general performance of the driving simulator's behavior depending on different Veins loads is investigated.

- *Uniformly Distributed Veins Load:* The Veins load is varied between a minimum value and a maximum value of an uniform distribution. The average Veins load is consequently the average of the minimum and the maximum value. Everytime Veins advances its simulation state, a new random value corresponding to the adjusted uniform distribution is taken as duration for the new time step of Veins. The uniform distribution enables investigations of

the behavior of the driving simulator's performance when a simulation step varies in its duration.

- *10 Hz Beaconing:* In this scenario each vehicle runs a concrete application in Veins which counts the fellow vehicles in communication range. Therefore, each vehicle sends beacon messages at a rate of 10 Hz. This experiment shows how many vehicles can be simulated in real-time in this particular scenario with the described hardware.

## 5.2 Subjective Observations

As mentioned before there are 36 ego vehicles traces. Each trace is 3.5 km long. Consequently I drove 36 times 3.5 km which are 126 km altogether, only for evaluation purposes. During driving there were several effects impacting the immersion driving a real vehicle which I recognized.

First of all, the current field of view is too small. Since the visualization is currently presented on a single monitor, only the cockpit of the ego vehicles is visualized. Hence, it is hard to see vehicles on intersecting roads.

The second effect reducing the immersion is the sense for velocity inside visualization. It is hard to estimate the current velocity of the ego vehicle without a speedometer. This effect is mostly noticeable at taking turns. If the speed of the ego vehicle is too high, sharp turns are not possible and the ego vehicle will leave the road.

A positive aspect of the visualization is that the human does not recognize whether there are two or three missed real-time deadlines because the ego vehicle is always displayed in a real-time manner due to the direct connection between the vehicle dynamics model and the visualization. Thus, the immersion of the driving simulator does not break entirely when there are a few missed real-time deadlines.

However, the biggest effect which affects the immersion negatively is caused by SUMO. SUMO is responsible for simulating the fellow vehicles. Inside SUMO, vehicles are simulated as single points in space and there is no vehicle dynamics model for the fellow vehicles. Especially in turns, the lack of a vehicle dynamics model is noticeable because fellow vehicles can almost turn on the point where they are without reducing their velocity.

Furthermore, fellow vehicles behave strange compared to real-world road traffic with the default SUMO configuration. For example, the fellow vehicles sometimes ignore traffic regulations at crowded intersections without traffic light system. This behavior can be explained by the aim of SUMO to involve effects caused by humans to some degree. One parameter impacting the behaviour is called *impatience*. If a vehicle has to wait too long at an intersection its impatience factor increases.

After exceeding a certain threshold, the vehicle enters the intersection even if the vehicle breaks traffic regualtions or other vehicles have to brake in order to not collide. For humans using the driving simulator, it is hard to interact with this behavior because the fellow vehicles act as there were no traffic regulations and it is hard to enter the intersection without provoking a collision. Moreover, the way this parameter is implemented does not represent the behavior of real humans at crowded intersections. This behavior may be improved with a better SUMO configuration.

Concluding, the driving simulator provides an immersion good enough to show the feasibility of the concept. Since a few missed real-time deadlines are not noticeable, the immersion does not break entirely and the simulation is still representable when the driving simulator is able to catch up the missed real-time deadlines. The next section highlights under which circumstances the driving simulator holds the real-time deadlines and in which manner the driving simulator is able to recover from a few missed real-time deadlines.

## 5.3 Quantitative Performance Analysis

In order to get a basic insight in the driving simulator's performance, results of the Static Veins Load experiment are evaluated. Before evaluating the network part of the driving simulator, it is important to know whether the visualization is able to update the 3D environment fast enough for a fluent visualization. Richard et al. [33] show that humans can interact properly with visualizations when 14 to 28 Frames Per Second (FPS) are displayed. Without a fluent visualization the driving simulator loses its immersion. Therefore, the FPS of the visualization are measured with different numbers of synchronized vehicles.

The number of frames per synchronization interval is counted and converted in FPS. In Figure 5.2 the average FPS is plotted against the number of synchronized vehicles. First it can be seen that the visualization is fast enough even for 200 synchronized vehicles on average with the given setup and scenario. For 200 synchronized vehicles there are 36 FPS on average which leads to a quiet fluent visualization. The error bars indicate the 95% confidence interval. This means 95% of all measured data points lay between the errorbars. 2.5% off all values lay below the lower error bar and 2.5% lay above the higher error bar. Hence, even for 200 synchronized vehicles there are at least 20 FPS in 97.5% of all synchronization intervals. In synchronization intervals in which the FPS are below 20, the visualization might lag but it is still possible to control the ego vehicle.

The more interesting fact is that the FPS are decreasing linearly with an increasing number of synchronized vehicles. This effect occurs because the visualization has the same amount of computational effort for updating the synchronized states for

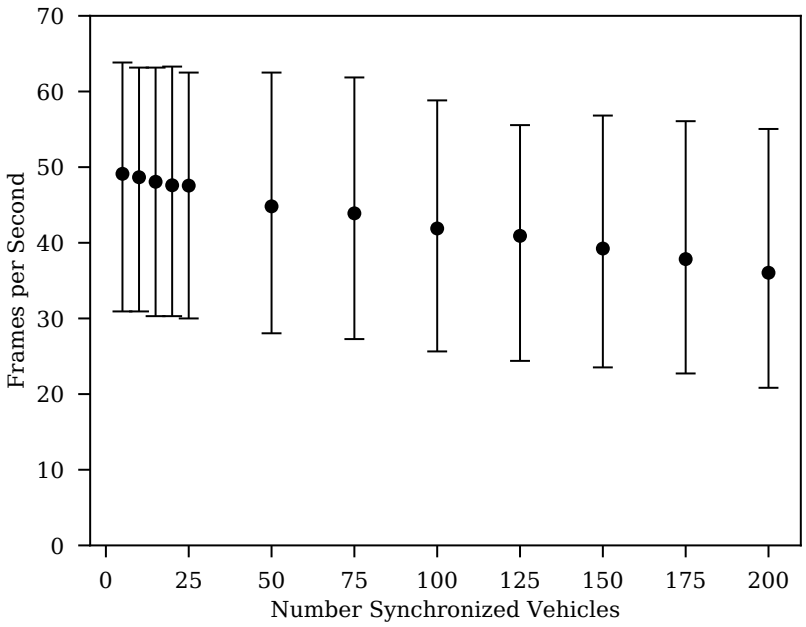



**Figure 5.2** – Average FPS depending on number of synchronized vehicles. Error bars indicate the 95% confidence interval.

each vehicle. The data points of 20, 50 and 100 synchronized vehicles lay a little bit lower than the lineal tendency. The reason for that may be that Figure 5.2 shows the FPS depending on the number of synchronized vehicles, not the number of actual displayed vehicles. When there is more computational effort for actually displaying the fellow vehicles the FPS will consequently decrease.

Since it is shown that the visualization is able to fluently display the driving environment, the remainder of this chapter is about the discrete time-based simulators performing under real-time conditions. As stated before, SUMO is responsible for the fellow vehicle behavior and for including the ego vehicle. Hence, SUMO has a large influence on the driving simulator without considering the IVC aspect. Because the ego vehicle is controlled by human the ego vehicle state inside SUMO has to be updated. Figure 5.3 shows the distribution of the duration for updating the ego vehicle state in SUMO. 99.99% of all ego vehicle updates are done in less than 0.5 ms of the 100 ms synchronization interval. This small value is reasonable since SUMO only has to update the ego vehicle state and send back an acknowledge message. SUMO does not advance its simulation state when it receives an ego vehicle update. Accordingly, the measured time is more or less the round trip time between YAEVI and SUMO. This may also explain the outlier for an ego vehicle update duration of over 4 ms since the network could be exposed to fluctuation.

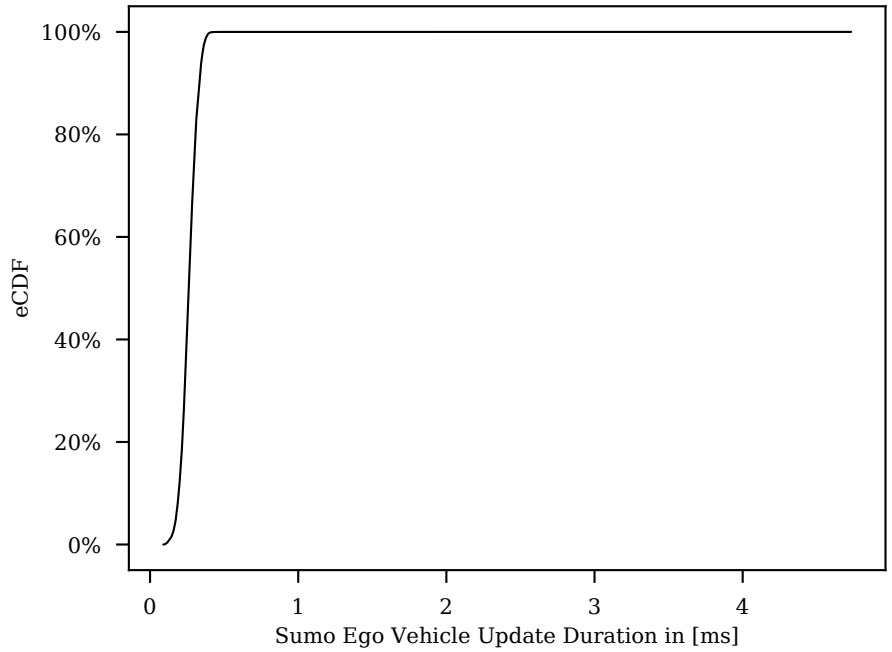



**Figure 5.3** – eCDF of the duration for updating the ego vehicle state in SUMO.

In order to investigate the duration SUMO needs to advance its simulation state, the time points before sending the command to advance the simulation by one simulation step and after receiving the new vehicle state from SUMO are measured. The difference between these two points is considered as the duration SUMO takes for simulating a single time step. This duration has to be lower than the 100 ms synchronization interval. Figure 5.4 shows that the duration for simulating a single time step in SUMO is always lower than 100 ms. On average SUMO needs about 8 ms to 15 ms for providing new simulation states of the fellow vehicles. The longest duration SUMO takes to provide new simulation step results is about 35 ms. Thus, SUMO never missed a real-time deadline.

But unfortunately, it can be seen in Figure 5.4 that the duration increases quite linearly with no upper bound in the investigated scenario. This could lead SUMO to constantly miss real-time deadlines after a certain time. Consequently, the driving simulator components would desynchronize. Moreover, Figure 5.4 shows the appearance of some outliers. These two observations could be based on two aspects. First, during the recordings of the ego vehicle traces it was observed that the computational effort of SUMO increased when the ego vehicle left the street. SUMO is not used to vehicles not driving on the street and therefore there might be some behavior inside SUMO causing the increasing SUMO simulation step duration. Furthermore, in order to have always the same number of fellow vehicles inside the simulation,

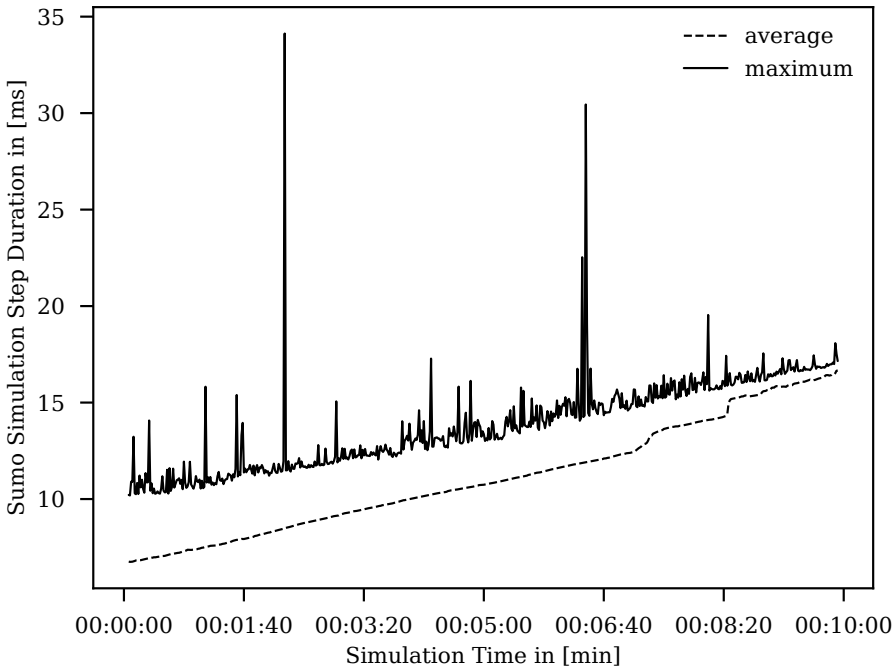



**Figure 5.4** – Average and maximum duration SUMO takes to simulate single time step over time.

the maximum number of fellow vehicles is bounded to a given number. When there are more fellow vehicles scheduled than allowed in SUMO, SUMO will load the scheduled fellow vehicles anyway but will not integrate them in the simulation. In the evaluated SUMO configuration there are permanently generated fellow vehicles in order to always have enough fellow vehicles inside the scenario. The list of fellow vehicles which are loaded but cannot be integrated in the simulation permanently grows. The growing list might be a problem for SUMO simulating a new simulation step causing the duration to grow as well. Due to time limitations of this thesis there is no further investigation on this behavior of SUMO. But Buse et al. [23] already showed that SUMO is able to perform better under similar conditions regarding real-time criteria. Hence, SUMO might perform better or in best case have an upper bound of the simulation step duration with another parametrization.

Investigations regarding Veins influencing the driving simulator are presented next because Veins will probably be the bottleneck in order to meet the 100 ms synchronization interval [32]. To provide evaluation results regarding Veins independent of a specific Veins implementation or scenario, the driving simulator is investigated with different Veins loads. As illustrated by Figure 5.5, Veins can use up to 95% of the synchronization interval for IVC simulation and the driving simulator

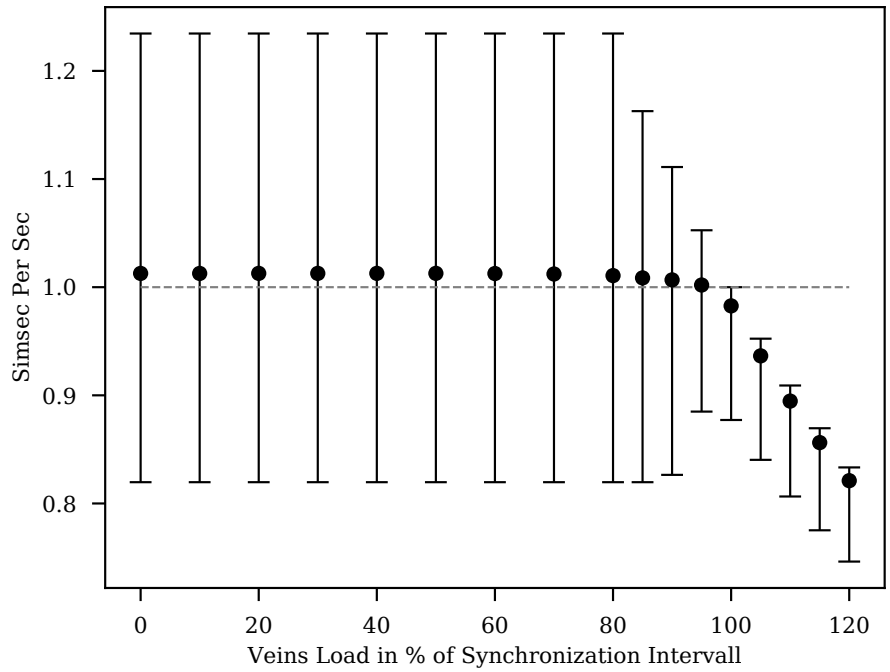



**Figure 5.5** – Average simsec per second depending on different static Veins loads. Error bars indicate the 95% confidence interval.

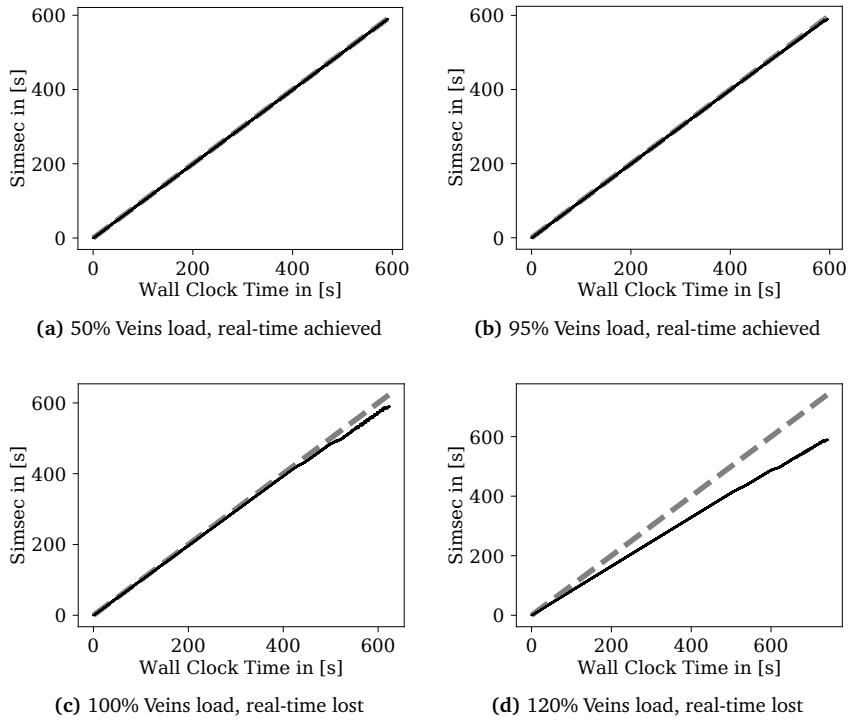
has an average simulation second per second of 1. Means that, on average, the driving simulator meets the real-time deadline up to a Veins load of 95%.

The error bars indicate the 95% confidence interval and show that the simulation second per second values vary around 1. The reason for this is the varying duration between two clock tick messages which arrive at the YAEVI. Sometimes the duration between these messages is shorter or longer than the synchronization interval because the wall clock of the visualization does not always provide exact 100 ms intervals. Therefore, the corresponding simulation second per second varies because the discrete time-based simulators Veins and SUMO advance their simulation states in exactly 100 ms steps.

When the Veins load is adjusted to 100% or higher the average simulation second per second value is lower than 1. In some cases the simulation second per second value is 1 when the duration between two clock tick messages is longer than 100 ms. Consequently, with such an adjusted Veins load the driving simulator does not meet any real-time deadline. When the Veins load is adjusted to 100%, a single Veins simulation step takes as long as the synchronization interval. Since the YAEVI also needs a few milliseconds for synchronizing all simulators, a Veins load of 100% will cause missing real-time deadlines. For even higher Veins loads this effect will increase. Figure 5.6 illustrates the simulation second per second ratio in a more

intuitive way. The grey dashed line represents the wall clock time. When the black line of the plots lays exactly on top of the grey line like in Figure 5.6a and Figure 5.6b the driving simulator performs in a real-time manner. Figure 5.6c and Figure 5.6d show the driving simulator not performing in a real-time manner because the black line is not as steep as the grey line indicating the wall clock time.

When Veins always takes longer than the synchronization interval for simulating a single time step, as shown in Figure 5.6d, it is obvious that the driving simulator will not perform in a real-time manner. A more interesting question is how the driving simulator behaves when there is not always a static load but rather a fluctuation on the Veins load. Is the driving simulator able to catch up time and recover to real-time when there are some simulation steps taking longer than the synchronization interval and some simulation steps taking shorter than the synchronization interval? To investigate the driving simulator's behavior with fluctuating Veins load the *Uniformly Distributed Veins Load* experiment was conducted. The fluctuation models the behavior of Veins in a more realistic way because the duration of a time step of a concrete application in Veins deviates, too, depending on the actual computational effort needed for processing it. Four different uniform distributions are highlighted in




**(a)** 50% Veins load, real-time achieved

**(b)** 95% Veins load, real-time achieved

**(c)** 100% Veins load, real-time lost

**(d)** 120% Veins load, real-time lost

**Figure 5.6** – Simulation second per second relation depending on different static Veins loads.

Table 5.2. The four different average Veins loads shall represent three cases. First, 80% average Veins load represents a case in which it is possible to meet the real-time deadlines when there would be a static load. 95% and 100% average Veins load represents the corner case in which it can be critical to meet the real-time deadlines and 110% represents a case in which it should not be possible to meet the real-time deadlines. Figure 5.7 illustrates the highlighted uniformly distributed Veins loads. Unfortunately, there are some Veins simulation steps that are even longer than the maximum of the adjusted uniform distribution recognizable by the not perfectly straight lines. The even longer simulation steps could be caused by some overhead of Veins managing the states of all vehicles. Normally, an empirical Cumulative Distribution Function (eCDF) of an uniform distribution is a perfect straight line.

| average | uniform minimum value | uniform maximum value |
|---|---|---|
| 80% | 30% | 130% |
| 95% | 65% | 125% |
| 100% | 70% | 130% |
| 110% | 80% | 140% |

**Table 5.2** – Input parameter of the dummy-app for uniformly distributed Veins loads.

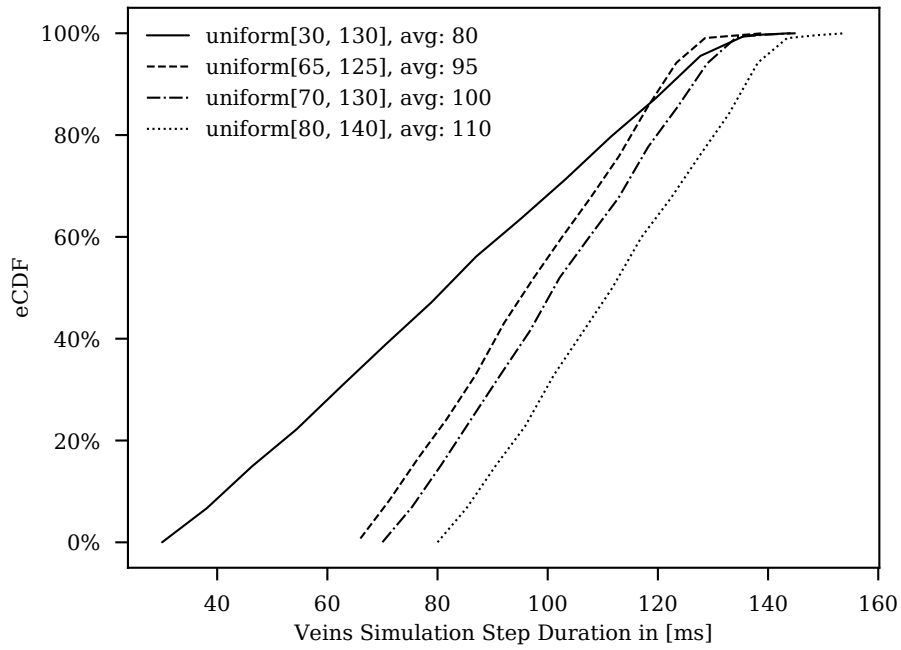
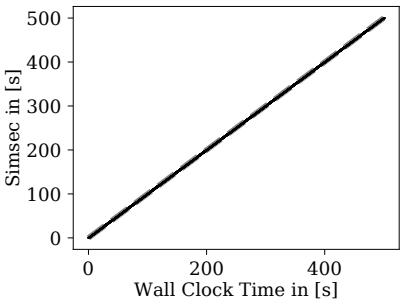


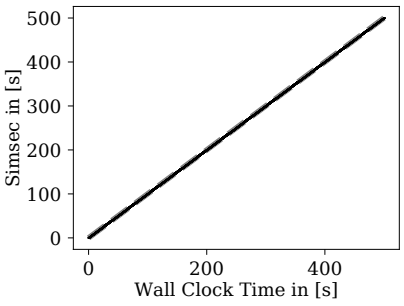

**Figure 5.7** – Simulation step duration of investigated uniformly distributed Veins loads.

Important to recognize is that in all distributions there are Veins simulation steps which take longer than 100 ms.

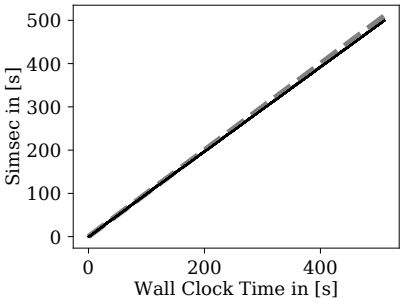
As stated in the section about subjective observations, the ability to compensate missed real-time deadlines prevents the driving simulator to lose the immersion. In order to keep synchronized, the driving simulator has to compensate such too long Veins simulation steps and has to catch up time when there are Veins simulation steps shorter than the synchronization interval of 100 ms. Figure 5.8 illustrates the relation of simulation second to second. As shown by Figure 5.8a and Figure 5.8b, the driving simulator is able to compensate slight real-time deadline misses. This behavior is caused because the YAEVI always tries to synchronize all driving simulator components as fast as possible and afterwards waits for the new clock tick message provided by the visualization. When the driving simulator missed a deadline, the YAEVI does not have to wait for the corresponding clock tick message and can therefore immediately synchronize all driving simulator components in order to catch up time. But as shown by Figure 5.8c and Figure 5.8d, when the average Veins load is too high it is not possible to perform in a real-time manner.



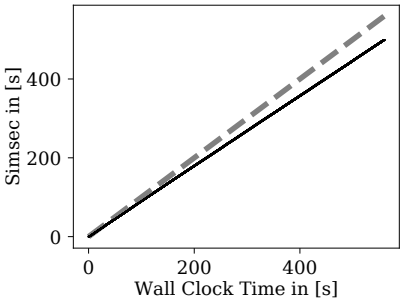

**(a)** Uniform distribution 30 to 130, avg: 80, real-time achieved




**(b)** Uniform distribution 65 to 125, avg: 95, real-time achieved




**(c)** Uniform distribution 70 to 130, avg: 100, real-time lost




**(d)** Uniform distribution 80 to 140, avg: 110, real-time lost

**Figure 5.8** – Simsec to second relation depending on uniformly distributed Veins simulation step duration

The maximum difference between the simulation time of Veins and SUMO and the wall clock time confirms the results and is showed in Table 5.3. In Figure 5.8a the maximum difference is −175 ms between the simulation time of the discrete time-based simulators and the wall clock time provided by the visualization. This means Veins and SUMO never lag more than two simulation steps behind the visualization. But in Figure 5.8d the maximum difference is −59 873 ms which means that Veins and SUMO almost lag a whole minute behind the visualization. When the components are desynchronized so much, the driving simulator does no longer work properly because the simulation states of the vehicles does not match with the visualized environment.

All recent results base on different Veins loads generated by the dummy-app, but these results do not show how many vehicles can be simulated in the actual scenario, given the used software versions and the available hardware. In order to investigate the capabilities of the driving simulator with real simulated IVC, the *10 Hz Beaconing* experiment is conducted. This experiment is an actual application in Veins using radio propagation models and obstacle shadowing. In this scenario all simulated vehicles, including the ego vehicle, send beacon messages with a sending rate of 10 Hz in order build up a minimalistic neighbor table. The sending rate of 10 Hz is used because this rate is the highest rate in which beacon messages are generated according to the ETSI ITS-G5 protocol stack [8].

Figure 5.9 shows the average simulation second per second of the driving simulator depending on the number of simulated vehicles. Up to 20 vehicles can be simulated in order to perform under real-time conditions in this particular scenario. When there are 25 simulated vehicles the average simulation second per second is already below 1 and therefore the real-time deadlines are not met on average. The error bars show the 95% confidence interval in the same way as in Figure 5.5. Figure 5.10 shows the distribution of the Veins simulation step duration and helps to explains why the average simulation second per second value is about 1 when there are 20 simulated vehicles and why the average simulation second per second value is below 1 when there are 25 simulated vehicles.

| uniform distribution | maximum time difference |
|---|---|
| Figure 5.8a | −175 ms |
| Figure 5.8b | −396 ms |
| Figure 5.8c | −10 006 ms |
| Figure 5.8d | −59 873 ms |

**Table 5.3** – Maximum difference between simulation time and wall clock time of the discrete time-based simulators Veins and SUMO and the visualization.

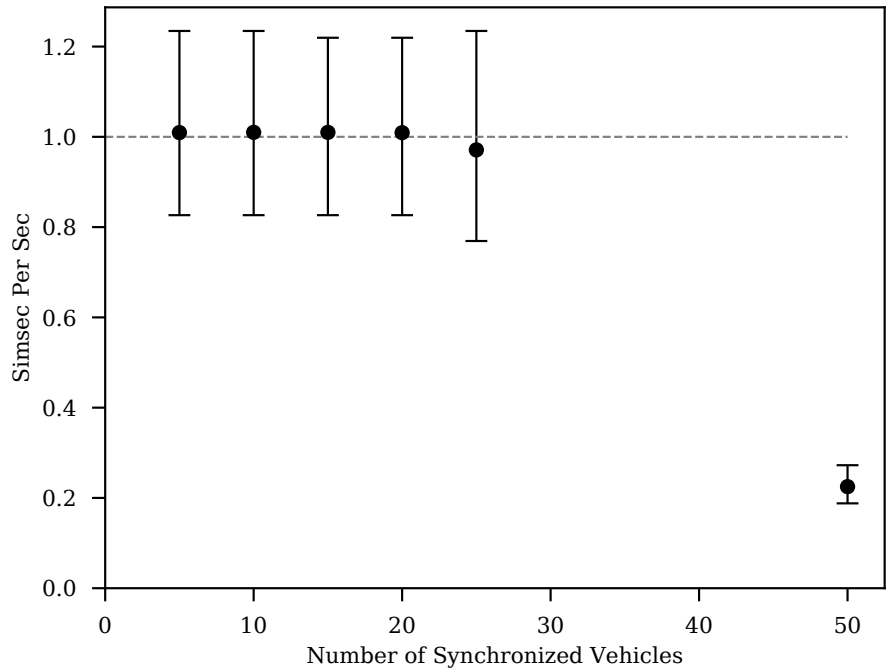



**Figure 5.9** – Average simsec per second depending on number of simulated vehicles [5, 10, 15, 20, 25, 50].

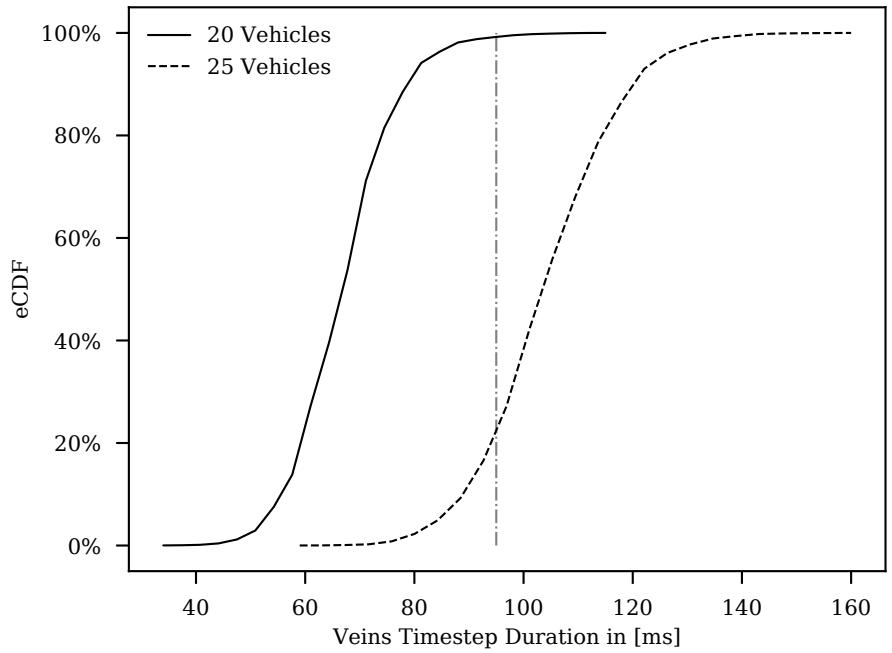



**Figure 5.10** – Distribution of the duration Veins needs to simulate a single simulation step when there are 20 and 25 simulated vehicles.

The static load experiment showed that Veins can use up to 95% of the synchronization interval for processing a single simulation step. Consequently, a Veins simulation step can take about up to 95 ms in this scenario without missing real-time deadlines. When there are only 20 simulated vehicles 99% of all Veins simulation steps take less than 95 ms. As showed earlier, the driving simulator is able to recover from simulation steps taking longer than the synchronization interval. Thus, the 1% of Veins simulation steps taking longer than 95 ms can be compensated by the driving simulator. But when there are 25 simulated vehicles, only 24% of all Veins simulation steps take less than 95 ms. The rest of the Veins simulation steps take longer and therefore cause missing real-time deadlines. Furthermore, the driving simulator cannot compensate the larger amount of simulation steps longer than the synchronization interval because there is too much load generated by Veins on average. As a result, the maximum number of simulated vehicles which allows simulating in a real-time manner lays between 20 and 25 vehicles.

These results complete the evaluation. The evaluation shows the feasibility of the proposed concept and implementation. The driving simulator performs in a real-time manner when all components meet the real-time deadlines. The evaluation shows that Veins can use up to 95% of the synchronization interval and the components keep synchronized. Further, the driving simulator is able to catch up time when Veins misses a few real-time deadlines. The recovering from missed deadlines works when the average Veins load is equal or below 95%. Finally, the evaluation results show that the driving simulator supports about 20 vehicles sending beacon messages at a sending rate of 10 Hz.

# Chapter 6

# Conclusion

This thesis proposed a concept and a concrete implementation of an interactive driving simulator for C2X scenarios. The driving simulator consists of four major components and an interface orchestrating the components. The discrete time-based simulators Veins and SUMO are connected with a real-time based vehicle dynamics model and a 3D visualization. The orchestration is done with the help of the YAEVI. The YAEVI is responsible for synchronizing Veins, SUMO and the 3D visualization. Moreover, the YAEVI handles the information exchange among Veins, SUMO and the 3D visualization. Evaluation results show the capability of all components meeting the real-time deadlines. Further, it is illustrated that Veins can use to up 95% of the time of the synchronization interval for simulating IVC. Even too long Veins simulation steps due to uniformly distributed Veins loads can be compensated when the average Veins load is not higher than 95%. Consequently, the immersion provided by the driving simulator does not break entirely when there are only a few missed real-time deadlines. With the developed driving simulator, humans can control a vehicle inside C2X scenarios with currently about 20 supported vehicles. Cooperative ADASs and human behaviors of driving vehicles can be evaluated together in small scenarios. With a more efficient Veins implementation more vehicles could be supported because Veins is currently the bottleneck of the driving simulator regarding real-time deadlines. As future work, an approach which couples the components asynchronously avoids the driving simulator from entirely desynchronizing when Veins is not able to meet the real-time deadlines at all. Moreover, an subjective study to investigate the human perception based aspects of the driving simulator in order to improve them is a reasonable future work.

# List of Abbreviations

| | |
|---|---|
| **ADAS** | Advanced Driver Assistant System |
| **API** | Application Programming Interface |
| **C2X** | Car2X |
| **DiVE** | Distributed Virtual Environment |
| **DSRC** | Dedicated Short Range Communication |
| **eCDF** | empirical Cumulative Distribution Function |
| **ETSI** | European Telecommunications Standards Institute |
| **EVI** | Ego Vehicle Interface |
| **FPS** | Frames Per Second |
| **HIL** | Hardware-in-the-Loop |
| **HuIL** | Human-in-the-Loop |
| **IVC** | Inter-Vehicle Communication |
| **LTE** | Long Term Evolution |
| **MAC** | Medium Access Control |
| **NS2** | Network Simulator 2 |
| **OMNeT++** | Objective Modular Network Testbed in C++ |
| **OSM** | OpenStreetMap |
| **PubSub** | publish-subscribe |
| **RF** | Radio Frequency |
| **SUMO** | Simulation of Urban MObility |
| **TCP** | Transmission Control Protocol |
| **TIS** | Traffic Information System |
| **TraCI** | Traffic Control Interface |
| **UDP** | User Datagram Protocol |
| **VANET** | Vehicular Ad Hoc Network |
| **Veins** | Vehicles in Network Simulation |
| **WAVE** | Wireless Access in Vehicular Environments |
| **YAEVI** | Yet Another Ego Vehicle Interface |

# List of Figures

# List of Tables

# Bibliography


[1] A. Gern, U. Franke, and P. Levi, "Robust Vehicle Tracking Fusing Radar and Vision," in *International Conference on Multisensor Fusion and Integration for Intelligent Systems MFI 2001*, Baden-Baden, Germany: IEEE, Aug. 2001, pp. 323–328. DOI: 10.1109/MFI.2001.1013555.

[2] C. Sommer and F. Dressler, Vehicular Networking. Cambridge University Press, Nov. 2014. DOI: 10.1017/CBO9781107110649.

[3] S. Joerer, B. Bloessl, M. Huber, A. Jamalipour, and F. Dressler, "Assessing the Impact of Inter-Vehicle Communication Protocols on Road Traffic Safety," in *20th ACM International Conference on Mobile Computing and Networking (MobiCom 2014), 6th Wireless of the Students, by the Students, for the Students Workshop (S3 2014)*, Maui, HI: ACM, Sep. 2014, pp. 21–23. DOI: 10.1145/2645884.2645885.

[4] K. Tateiwa, A. Nakamura, and K. Yamada, "Study on Estimating Driver Awareness of Pedestrians While Turning Right at Intersection Based on Vehicle Behavior Utilizing Driving Simulator," in *IEEE Intelligent Vehicles Symposium (IV)*, Gothenburg, Sweden: IEEE, Jun. 2016, pp. 388–393. DOI: 10.1109/IVS.2016.7535415.

[5] K. Sonoda and T. Wada, "Displaying System Situation Awareness Increases Driver Trust in Automated Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 185–193, Sep. 2017. DOI: 10.1109/TIV.2017.2749178.

[6] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011. DOI: 10.1109/TMC.2010.133.

[7] L. Wischhof, A. Ebner, H. Rohling, M. Lott, and R. Halfmann, "SOTIS - A Self-Organizing Traffic Information System," in *57th IEEE Vehicular Technology Conference (VTC2003-Spring)*, Jeju, South Korea: IEEE, Apr. 2003, pp. 2442–2446. DOI: 10.1109/VETECS.2003.1208829.

[8] ETSI, “Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service,” ETSI, EN 302 637-2 V1.3.2, Nov. 2014.

[9] G. Rémy, S.-M. Senouci, F. Jan, and Y. Gourhant, “LTE4V2X: LTE for a Centralized VANET Organization,” in *IEEE Global Telecommunications Conference (GLOBECOM 2011)*, Houston, TX: IEEE, Dec. 2011. DOI: 10.1109/GLOCOM.2011.6133884.

[10] A. Varga, “The OMNeT++ Discrete Event Simulation System,” in *European Simulation Multiconference (ESM 2001)*, Prague, Czech Republic, Jun. 2001.

[11] M. Lacage and T. R. Henderson, “Yet Another Network Simulator,” in *2006 Workshop on Ns-2: The IP Network Simulator*, Pisa, Italy: ACM, Oct. 2006. DOI: 10.1145/1190455.1190467.

[12] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, 1st ed. Boston, MA: Springer US, 2009. DOI: 10.1007/978-0-387-71760-9_2.

[13] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, “SUMO (Simulation of Urban MObility); An Open-source Traffic Simulation,” in *4th Middle East Symposium on Simulation and Modelling (MESM 2002)*, Sharjah, United Arab Emirates, Sep. 2002, pp. 183–187.

[14] A. Wegener, M. Piorkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, “TraCI: An Interface for Coupling Road Traffic and Network Simulators,” in *11th Communications and Networking Simulation Symposium (CNS 2008)*, Ottawa, Canada: ACM, Apr. 2008, pp. 155–163. DOI: 10.1145/1400713.1400740.

[15] J. Kuhl, D. Evans, Y. Papelis, R. Romani, and G. Watson, “The Iowa Driving Simulator: An Immersive Research Environment,” *IEEE Computer*, vol. 28, no. 7, pp. 35–41, Jul. 1995. DOI: 10.1109/2.391039.

[16] E. Blana, “A Survey of Driving Research Simulators Around the World,” Institute for Transport Studies, University of Leeds, Working Paper 481, Dec. 1996.

[17] K. Abdelgawad, B. Hassan, J. Berssenbrügge, J. Stöcklein, and M. Grafe, “A Modular Architecture of an Interactive Simulation and Training Environment for Advanced Driver Assistance Systems,” *International Journal On Advances in Software*, vol. 8, no. 1 & 2, pp. 247–261, Jun. 2015.

[18] Y. Boukadida, A. Masmoudi, G. M. Casolino, and F. Marignetti, “A Simple Assessment of the Dynamics of the Road Vehicles,” in *18th International Conference on Ecological Vehicles and Renewable Energies (EVER)*, Monte-Carlo, Monaco: IEEE, Apr. 2018. DOI: 10.1109/EVER.2018.8362416.

[19] K. Abdelgawad, M. Abdelkarim, B. Hassan, M. Grafe, and I. Gräßler, "A modular architecture of a PC-based driving simulator for advanced driver assistance systems development," in *15th International Workshop on Research and Education in Mechatronics (REM)*, El Gouna, Egypt: IEEE, Sep. 2014. DOI: 10.1109/REM.2014.6920237.

[20] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, 2nd ed. Springer US, 2011. DOI: 10.1007/978-1-4419-8237-7.

[21] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, May 1999. DOI: 10.1016/S0967-0661(98)00205-6.

[22] K. S. Swanson, A. A. Brown, S. N. Brennan, and C. M. LaJambe, "Extending Driving Simulator Capabilities Toward Hardware-in-the-Loop Testbeds and Remote Vehicle Interfaces," in *IEEE Intelligent Vehicles Symposium (IV'13)*, Gold Coast, QLD, Australia: IEEE, Jun. 2013, pp. 122–127. DOI: 10.1109/IVS.2013.6629458.

[23] D. S. Buse, M. Schettler, N. Kothe, P. Reinold, C. Sommer, and F. Dressler, "Bridging Worlds: Integrating Hardware-in-the-Loop Testing with Large-Scale VANET Simulation," in *14th IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS 2018)*, Isola 2000, France: IEEE, Feb. 2018. DOI: 10.23919/WONS.2018.8311659.

[24] S. Guan, R. E. De Grande, and A. Boukerche, "Real-time 3D Visualization for Distributed Simulations of VANets," in *18th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, Toulouse, France: IEEE, Oct. 2014, pp. 138–146. DOI: 10.1109/DS-RT.2014.25.

[25] D. Gruyer, O. Orfila, V. Judalet, S. Pechberti, B. Lusetti, and S. Glaser, "Proposal of a Virtual and Immersive 3D Architecture dedicated for Prototyping, Test and Evaluation of Eco-Driving Applications," in *IEEE Intelligent Vehicles Symposium (IV'13)*, Gold Coast, QLD, Australia: IEEE, Jun. 2013, pp. 511–518. DOI: 10.1109/IVS.2013.6629519.

[26] W. M. Griggs, R. H. Ordóñez-Hurtado, E. Crisostomi, F. Häusler, K. Massow, and R. N. Shorten, "A Large-Scale SUMO-Based Emulation Platform," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3050–3059, Dec. 2015. DOI: 10.1109/TITS.2015.2426056.

[27] F. Michaeler and C. Olaverri-Monreal, "3D Driving Simulator with VANET Capabilities to Assess Cooperative Systems: 3DSimVanet," in *IEEE Intelligent Vehicles Symposium (IV'17)*, Redondo Beach, CA, USA: IEEE, Jun. 2017, pp. 999–1004. DOI: 10.1109/IVS.2017.7995845.

[28] H. Prendinger, M. Miska, K. Gajananan, and A. Nantes, "A Cyber-Physical System Simulator for Risk-Free Transport Studies," *Computer-Aided Civil and Infrastructure Engineering*, vol. 29, no. 7, pp. 480–495, Aug. 2014. DOI: 10.1111/mice.12068.

[29] Y. Hou, Y. Zhao, A. Wagh, L. Zhang, C. Qiao, K. F. Hulme, C. Wu, A. W. Sadek, and X. Liu, "Simulation-Based Testing and Evaluation Tools for Transportation Cyber-Physical Systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 3, pp. 1098–1108, Mar. 2016. DOI: 10.1109/TVT.2015.2407614.

[30] P. Hintjens, ZeroMQ Messaging for Many Applications. O'Reilly Media, 2013.

[31] M. Haklay and P. Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008. DOI: 10.1109/MPRV.2008.80.

[32] C. Obermaier and C. Facchi, "Observations on OMNeT++ Real-Time Behaviour," in *4th OMNeT++ Community Summit (OMNeT++ 2017)*, Bremen, Germany: arXiv, Sep. 2017.

[33] P. Richard, G. Birebent, P. Coiffet, and G. Burdea, "Effect of Frame Rate and Force Feedback on Virtual Object Manipulation," *Presence: Teleoperators and Virtual Environments*, vol. 5, no. 1, pp. 95–108, Winter 1996. DOI: 10.1162/pres.1996.5.1.95.