

Höchstleistungsrechenzentrum
Universität Stuttgart
Prof. Dr.-Ing. Dr. h.c. Dr. h.c. Prof. E.h. Michael M. Resch
Nobelstraße 19
70569 Stuttgart
Institut für Höchstleistungsrechnen

Simulationsgestützte Absicherung von Fahrerassistenzsystemen

Von der Fakultät Energie-, Verfahrens- und Biotechnik der Universität
Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

Vorgelegt von
Marius Feilhauer
aus Schorndorf

Hauptberichter:	Prof. Dr.-Ing. Dr. h.c. Dr. h.c. Prof. E.h. Michael M. Resch
Mitberichter:	Prof. Dr. Hans-Christian Reuss
Tag der Einreichung:	4. Juni 2018
Tag der mündlichen Prüfung:	20. Dezember 2018

Institut für Höchstleistungsrechnen der Universität Stuttgart
2018

Zusammenfassung

„Es kommt der Tag, an dem wir sagen: Jetzt musst du nicht mehr auf den Verkehr achten“, sagt Erik Coelingh, Entwicklungsleiter bei Volvo, im Interview mit Zeit Online [Assendorpf 2016]. Wann genau dieser Tag kommen wird, lässt sich schwer vorhersagen. Doch die Aussage verdeutlicht ein großes Arbeitsfeld aktueller Automobilentwicklung: Die Realisierung autonomer Fahrzeuge. Neben traditionellen Autobauern wie Volvo, GM, Ford, Honda, Toyota, BMW, Daimler, Bosch oder Continental befassen sich auch Unternehmen wie Tesla, Google, Nvidia, Intel oder Mobileye mit den technologischen Herausforderungen selbstfahrender Automobile. Die Umsetzung von hoch- und vollautomatisierten Fahrzeugen wird zu Beginn des kommenden Jahrzehnts erwartet. [Vgl. Guhlich 2017; Walker 2017; Segner 2015].

Neben der Bewältigung technischer Fragestellungen stellt auch die gesellschaftliche Akzeptanz der neuen Technologie eine Herausforderung dar: Bereits während überwachter Testfahrten mit selbstfahrenden Autos ereigneten sich tödliche Zusammenstöße zwischen Mensch und Maschine. Mit einem Anteil von 94% sind momentan fast alle Unfälle auf menschliches Verschulden zurückzuführen. Dies lässt den großen Vorteil vermuten, den zuverlässige Technik bringen könnte. [Vgl. Fasse 2018; Reidl 2018]

Eine passende Metrik für ethische Fragestellungen des autonomen Fahrens ist eine gesellschaftliche Aufgabe. Schnell lassen sich bei dieser Diskussion ethische Dilemmata skizzieren. Solche Unglücksszenarien sind bei der Umsetzung einer so vielversprechenden Technologie jedoch wenig zielführend. Peter Dabrock, Vorsitzender des deutschen Ethikrats, rät sich von diesen inszenierten Fragestellungen nicht einnehmen zu lassen und „pragmatischer an die ethische Bewertung autonomer Fahrzeuge“ heranzugehen [Dabrock 2017]. Jedoch entbindet ein pragmatisches Vorgehen Hersteller nicht von der Pflicht, die technische Funktionalität der Systeme möglichst umfassend sicherzustellen. Dies führt zum Fokus dieser Arbeit: Wie lassen sich Assistenzsysteme praxistauglich und nachhaltig absichern?

Als *Funktionale Unzulänglichkeit* kann der Umstand beschrieben werden, weshalb sich die Absicherung von Assistenzsystemen von etablierten Testpraktiken im automobilen Umfeld (beispielsweise basierend auf [ISO 26262]) unterscheidet. Zur vollständigen Absicherung der Funktionalität (teil-)autonomer Fahrzeuge müssten alle möglichen Verkehrssituationen beschrieben und das Verhalten anderer Verkehrsteilnehmer vorhergesagt werden können [Wilhelm 2015]. Dies ist nicht machbar und führt dazu, dass sich in einer Spezifikation nicht alle

Eventualitäten des realen Straßenverkehrs vorab definieren lassen.

Ein Lösungsansatz könnte die rein statistische Absicherung bieten, die ohne Systemwissen auskommt. Die Überlegungen von [Winner 2015a] zeigen, dass dieser Ansatz die wirtschaftlich und zeitlich akzeptablen Grenzen einer funktionalen Absicherung sprengt. Der Nachweis, dass ein Fahrzeug mit einem Autobahnassistenten bei 5% Irrtumswahrscheinlichkeit auf Autobahnen weniger tödliche Unfälle verursacht als ein Fahrzeug ohne das Assistenzsystem, bedarf einer Fahrstrecke von mehreren Millionen Kilometern. Hinzu kommt, dass diese Absicherung nur für exakt eine Fahrzeugkonfiguration und einen Softwarestand gültig wäre. Bei einer Änderung der Einbauposition von Sensoren oder einem Update der Software ist die Absicherung hinfällig und müsste wiederholt werden. [Winner 2015a]

Diese Arbeit stellt einen anderen Absicherungsansatz vor: Ein iterativ erweiterbarer Katalog zu testender Verkehrsszenarien.

Hierfür wird zunächst eine Äquivalenzklassen-motivierte Beschreibung von Verkehrsszenarien entworfen. Dies stellt den Grundstein des iterativ erweiterbaren Katalogs dar. Die Beschreibungen dienen als Testgrundlage und stellen Verkehrsszenarien zur funktionalen Absicherung von Assistenzsystemen dar. Sie basieren auf den momentan als relevant identifizierten Parametern. Der Szenarienkatalog stellt das zentrale Element einer Absicherungsstrategie dar, deren Ablauf in Abbildung 1 schematisch visualisiert ist.

Gespeist wird der Szenarienkatalog aus Black-, Gray- und White-Box-Quellen, die sich anhand des verwendeten Systemwissens unterscheiden.

Black-Box-Ansätze benötigen kein Wissen über das zu testende System, wie beispielsweise Dauerläufe oder Flottentests. Zufällige Verkehrsszenarien führen eventuell zu einer unerwarteten Systemreaktion und können in den Testkatalog aufgenommen werden. Im Rahmen der Arbeit wird die probabilistische Testfallgenerierung näher betrachtet. Dieses Vorgehen bedarf nur wenige Einschränkungen und eignet sich zur automatisierten Erzeugung unterschiedlicher Szenarien. Wird der Zielwert *Time-To-Collision* als Metrik für *interessante* Testfälle gewertet, so zeigt die dargestellte Betrachtung, dass der Anteil *interessanter* Szenarien gering ausfällt.

Gray-Box-Ansätze verwenden abstrahiertes Systemwissen. Experten können anhand ihrer Erfahrungswerte hinsichtlich des Assistenzsystems wertvolle Tests ableiten, um das System an dessen Grenzen zu bringen. Ein Beispiel ist die von Radarsensoren aufgrund der hohen Reflexivität häufig falsch klassifizierte Cola-Dose. Zudem lassen sich Szenarien anhand von Systemmodellen systematisch generieren. Hierfür wird ein modellbasierter Ansatz betrachtet. Die Erstellung des benötigten Systemmodells ist aufwändig oder führt bei starker Vereinfachung zur Vernachlässigung von Testfällen. Gleichzeitig kann die Ab-

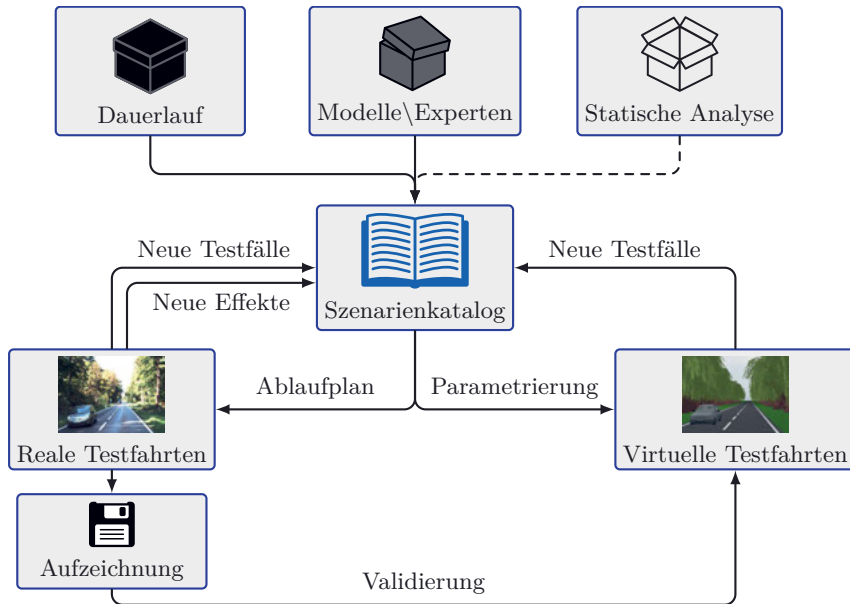


Abb. 1: Sequenzdiagramm der Absicherungsstrategie

leitung minimal notwendiger Testfälle für die Stimulation einzelner Zielwerte gelingen, wie das in der Arbeit dargestellte Beispiel verdeutlicht.

White-Box-Methoden nutzen den Quellcode des Systems zur Ableitung von Testvektoren. Wie sich zeigt ist dieser Ansatz nur für sehr einfache Funktionen praktikabel. Die Komplexität praxisnaher Anwendungen erlaubt keine sinnvolle Anwendung dieses Vorgehens.

Die Testabläufe des Szenarienkatalogs eignen sich zur Beschreibung realer Testfahrten. Im Rahmen der Arbeit werden anhand der vorgestellten ontologischen Beschreibung von Verkehrsszenarien reale Testfahrten dokumentiert. Aufgrund der Eignung der Ontologie zur Parametrierung virtueller Testfahrten gelingt der Transfer realer in virtuelle Fahrscenarien. Ein exemplarischer Einparkvorgang stellt zudem die Beschreibung komplexer dynamische Abläufe dar.

Werden bei realen Testfahrten neue Effekte oder Parameterkombinationen entdeckt, so müssen diese durch eine Anpassung oder Erweiterung der Szenarien in den Katalog aufgenommen werden. Durch die Möglichkeit der effizienten Parametervariation in Simulationsumgebungen können auch hier neue Verkehrskonstellationen identifiziert werden, deren Aufnahme sich in den Szenarienkatalog lohnt.

Eine Möglichkeit zur Reduktion kostenintensiver Prototypen und Testfahrten

ist deren Transfer in virtuelle Welten. Um dies zu realisieren, wird eine Simulationsarchitektur anhand dreier Ebenen erarbeitet. Die abstrakte *Anwendersicht* veranschaulicht relevante Entitäten zum Transfer in eine virtuelle Welt. Die *Kosimulationsstruktur* fokussiert nachfolgend die Trennung von Verantwortlichkeiten unterschiedlicher Simulationsmodule sowie die Wiederverwendbarkeit vorhandener Modelle. Das *Umgebungsmodell* spielt hierbei eine zentrale Rolle. Zur dreidimensionalen Repräsentation des Fahrzeugumfelds wird eine Klassenstruktur entworfen, die Ansätze der Spieleindustrie aufnimmt und sich zur Integration in die vorgestellte Kosimulationsstruktur eignet. Die Realisierung eines echtzeitfähigen Hardware-in-the-Loop Prüfstands zum Test eines realen Notbremsassistenten demonstriert die Wiederverwendbarkeit vorhandener Modelle, die Anwendbarkeit der präsentierten Architektur sowie die Umsetzung des Umgebungsmodells.

Abschließend werden zwei synthetische Sensormodelle entworfen. Basierend auf einer abstrakten Wirkkette von Fahrerassistenzsystemen findet die Modellbildung des *Idealen* und des *Multi-Sensor-Modells* statt. Das Ideale Modell liefert hoch performante Ground-Truth Informationen aus der virtuellen Umgebung. Das auf einem Raytracing-Ansatz basierende, physikalisch motivierte Multi-Sensor-Modell eignet sich zur Virtualisierung von Ultraschall-, Radar-, Video- und Lidar-Sensoren. Das für alle Sensortypen einheitliche Modellierungsprinzip sowie die Integration in die Simulationsarchitektur ermöglichen die Umsetzung eines konsistenten Umgebungsmodells. Neben Performance-Betrachtungen der Modelle zeigen Validierungen die Möglichkeiten der Virtualisierung von Messfahrten am Beispiel eines Ultraschallsensors. Zudem wird ein auf neuronalen Netzen basierender Objekterkennungsalgorithmus sowohl mit realen als auch mit synthetischen Videobildern stimuliert. Hierbei wird deutlich, dass eine Stimulation und qualitative Bewertung möglich ist. Allerdings ergeben sich sowohl False-Positive als auch False-Negative-Abweichungen, weshalb von einer quantitativen Performance-Bewertung der Algorithmen auf rein synthetisch basierten Daten abzuraten ist. Gleichzeitig erlaubt die vorgestellte Bewertungsmethodik anhand eines Vergleichs von Receiver-Operating-Characteristic-Kurven eine qualitative Einschätzung.

Gelingt es zukünftigen Modellentwicklungen die Abbildung von Sensorfehlern mit idealen Modellen zu realisieren, so ergeben sich hieraus entscheidende Vorteile: Die Performance der idealen Modelle ist ein Vielfaches höher als bei physikalisch motivierten Ansätzen. Somit können die Modelle problemlos für Model-, Software- und Hardware-in-the-Loop Tests verwendet werden. Gleichzeitig lassen sich die Komponenten von Assistenzsystemen ab der Objektlisten-Schnittstelle mit idealen Modellen testen. Durch die kurzen Simulationszeiten in Verbindung mit der Parallelisierung von Fahrscenarien lässt

sich so innerhalb von Minuten ein Szenarienkatalog mit mehreren tausend Szenarien in der virtuellen Welt abfahren. Da die Stimulation von Objekterkennungsalgorithmen mittels synthetischer Daten nicht robust gelingt, ist es zielführender im Rahmen einer Absicherungsstrategie diese Algorithmen Open-Loop anhand aufgezeichneter Sensordaten zu testen.

Durch die vorgestellte Absicherungsstrategie sowie die detaillierte Umsetzung virtueller Testfahrten leistet diese Arbeit einen Beitrag zur praxisnahen Absicherung von Assistenzsystemen und unterstützt so die Realisierung (teil-)autonomer Fahrzeuge.

Englische Zusammenfassung (Abstract)

Eric Coelingh, development manager at Volvo, states in an interview with Zeit Online that there will come the day when we can say: You don't have to be aware of the traffic any longer [Assendorpf 2016]. It's difficult to predict when exactly this day will come. But this statement illustrates a major field of work in current automotive development: The realization of autonomous vehicles. Besides traditional car manufactures like Volvo, GM, Ford, Honda, Toyota, BMW, Daimler, Bosch or Continental, companies like Tesla, Google, Nvidia, Intel or Mobileye are also focusing on the technological challenges to achieve autonomous vehicles. The realization of highly and fully automated cars is expected in the beginning of the upcoming decade [See Gühlich 2017; Walker 2017; Segner 2015].

Besides mastering the involved technical tasks, the social acceptance of self-driving cars is challenging: Even during monitored test drives of autonomous vehicles there have been deadly accidents between humans and machines. With a fraction of 95% currently almost all accidents are based on human failure. This suggests the enormous benefit of a reliable technical solution. [See Fasse 2018; Reidl 2018]

Finding an accepted metric for the ethical questions of autonomous vehicles is a social task. It's easy to come up with ethical dilemmas. But such disastrous situations are not expedient when trying to realize this promising technology. Peter Dabrock, chairmen of the German Ethics Council, advises not to get occupied by such produced scenarios but to approach the ethical assessment of autonomous vehicles more pragmatic [Dabrock 2017]. But such an approach does not discharge manufacturers of their duties when ensuring the technical functionality of the systems. This leads to the focus of this thesis: How to validate driver assistance systems in a practical and sustainable way?

The term *functional insufficiency*¹ expresses the fact, why current test strategies (e.g. based on [ISO 26262]) cannot be applied directly to validate Advanced Driver Assistance Systems (ADAS). To completely validate the functionality of (partly) autonomous vehicles, all possible traffic situations and the behavior of traffic participants would have to be predicted [Wilhelm 2015]. This is not possible and leads to the fact that not all eventualities can be defined within a system's specification.

¹Based on the German expression *Funktionale Unzulänglichkeit*

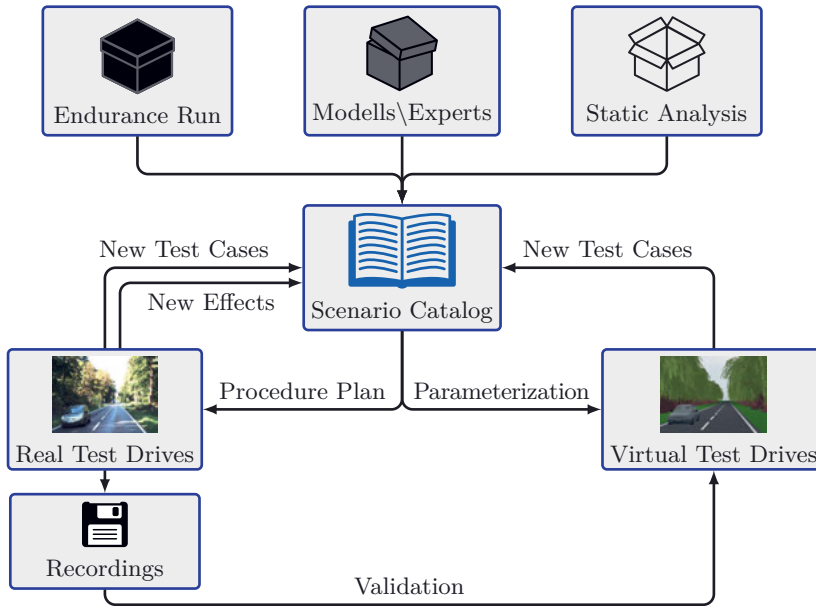


Figure 1: Sequence diagram for the validation strategy

A pure statistical validation approach could be a possible solution to this problem. The considerations in [Winner 2015a] show that such an approach is not within the acceptable financial and time boundaries for a functional validation. To proof that a vehicle, equipped with a highway pilot, performs better than a vehicle without a highway pilot, requires a multi-million-kilometer test drive, when accepting a significance level of 5%. Furthermore, this validation is only valid for a single vehicle configuration and a single software status. When changing a sensor’s installation position or updating the software, the validation is invalid and must be repeated. [Winner 2015a]

This work presents an alternative validation approach: An iteratively expandable test catalog.

As a cornerstone for this catalog an equivalence-class based description of driving scenarios is designed. These descriptions are the basis for tests and represent driving scenarios for the functional validation of driver assistance systems. They are based on the currently identified parameters relevant for the description of driving scenarios. Figure 1 illustrates the sequence diagram for the proposed validation strategy, including the catalog as its central element.

There are Black-Box, Gray-Box and White-Box approaches to set up the scenario catalog.

Black-Box approaches are not requiring any system knowledge, like endurance

runs or fleet tests. Random situations in endurance runs may lead to an unexpected reaction of the system. Such situations can then be included into the catalog. Within this work a closer look is taken at a probabilistic approach for test case generation. This procedure requires only few restrictions and can be used to automatically generate a huge number of different scenarios. When observing the target value *Time-To-Collision* it becomes clear that the proportion of *interesting* test cases is low.

Gray-Box approaches are based on abstract system knowledge. Experts can derive meaningful test cases based on their experience to stress the assistance system. An example is the Cola can, which often leads to incorrect classifications due to its strong radar reflectivity. Furthermore, scenarios can systematically be derived from system models. Such scenario creation is shown by a model based approach. The required establishment of a system model can be complex and might lead to neglected test cases when being oversimplified. Nevertheless, it is demonstrated that it is possible to create a minimum set of scenarios to test certain target values by using this approach.

White-Box methods use the system's source code to derive test cases. It turns out that this approach is only practical for very simple functions. Due to the complexity of real world applications this approach cannot be deployed reasonably.

The catalog's test cases can be used to describe real test drives. Within this work real test drives are documented using the proposed ontological description of driving scenarios. Additionally, the transfer of real to virtual test drives is realized by applying the scenario description as a parametrization for a simulation environment. Furthermore, the specification of complex dynamic maneuvers is illustrated by describing a parking scenario.

When exploring new effects or parameter combinations within real test drives, this information must be transferred into the catalog by adapting or extending the scenarios. Due to the possibility of varying parameters efficiently within a simulation environment, interesting constellations might be explored in virtual test drives, too.

A possibility to reduce expensive prototypes and test runs is their transfer into virtual worlds. To enable such a transfer, a simulation architecture is presented by three different layers. The abstract *user's view* illustrates relevant entities which have to be transferred into a virtual world. The *co-simulation structure* focuses on the separation of responsibilities and on the reuse of available simulation models. The *environment model* plays a central role within the overall simulation setup. To create a three-dimensional representation of the vehicle's environment and to enable its integration into the proposed software architecture, a class structure is designed using approaches from the

game industry. The realization of a real-time capable Hardware-in-the-Loop test setup including a real emergency brake assist demonstrates the reuse of available simulation models, the application of the presented architecture as well as the implementation of the environment model.

This work concludes by the design of two sensor models. Based on an abstract functional pipeline of Advanced Driver Assistance Systems, an *Ideal* and a *Multi-Sensor-Model* is designed. The Ideal Model provides high performance Ground-Truth information of the virtual environment. The ray-tracing-based, physically motivated Multi-Sensor-Model can be used to virtualize ultrasonic-, radar-, video and lidar-sensors. The consistent modeling approach along all sensor types and the possibility to integrate the model into the proposed software architecture, enables the realization of a consistent environment model. Besides performance tests of the models, validations show the possibility to virtualize ultrasonic measurements. Additionally, an object recognition algorithm, based on neural networks, is stimulated with real and synthetic data. This shows that the stimulation and qualitative assessment of such algorithms is generally possible. However, as there are false-positive and false-negative-discrepancies, the quantitative assessment of such algorithms should not be based on synthetic data only. The proposed evaluation approach using a comparison of Receiver-Operating-Characteristic curves enables to qualitatively judge the performance.

There are decisive advantages when future model developments succeed in enabling the mapping of sensor errors onto ideal models: The performance of ideal models is way higher compared to physically motivated approaches. Therefore, such models can easily be integrated within Model-, Software- and Hardware-in-the-Loop test setups. Additionally, these models can be used to test Advanced Driver Assistance Systems from the object-list-interface onwards. Due to their short simulation times and the possibility to parallelize the execution of virtual driving scenarios, a catalog including several thousand scenarios can be simulated within minutes. As the stimulation of object detection algorithms with synthetically generated data is not robust, it is expedient to test such algorithms open-loop based on recorded sensor data.

Due to the proposed validation strategy and the detailed realization of virtual test drives, this work contributes in the validation of Advanced Driver Assistance Systems and therewith supports the realization of (partly) autonomous vehicles.

Vorwort

Die vorliegende Arbeit entstand im Rahmen einer Industriepromotion in Kooperation mit dem Höchstleistungsrechenzentrum der Universität Stuttgart und der ETAS GmbH. Zu dieser Promotion haben viele Menschen auf unterschiedlichste Art und Weise beigetragen.

Bei der ETAS lernte ich in zahlreichen Diskussionen und bei regem Austausch interessante Kollegen kennen, die stets neue Aspekte und Sichtweisen zum Thema beigetragen haben. Allen voran möchte ich Dr. Jürgen Häring danken, von dessen fachlicher Erfahrung und motivierender Persönlichkeit ich viel Neues lernen durfte, wovon ich weit über die Zeit meiner Promotion profitieren werde.

Unterstützt wurde ich zudem von den Studenten Satheesh Raveendralingam, Benjamin Schnarr, Michael Ohlscher, Siyu Wang und Damian Boborzi. Vielen Dank für die stets freundliche, motivierte und produktive Zusammenarbeit mit Euch.

Ein weiterer Dank gilt den zahlreichen Kollegen innerhalb ETAS und Bosch, für deren fachliche Expertise und Unterstützung. Hier möchte ich Prathyusha Duvvuru, Dr. Jens Buchner, Lars Reimer, Paul Weber, Jürgen Crepin, Dr. Holger Ulmer und Dr. Andrej Junginger sowie meine Kollegen und Mitstreiter Raphael Hans und Marc Habiger nennen.

Vielen herzlichen Dank an meinen Doktorvater Professor Michael Resch für das sehr lehrreiche und professionelle Feedback, den stets freundlichen Austausch und die Möglichkeit zu dieser Promotion. Zudem hat es mich gefreut erste Erfahrungen in der Lehre an der Universität Stuttgart sammeln zu können. Des Weiteren möchte ich mich bei Herrn Professor Hans-Christian Reuss für die entgegenkommende Übernahme des Korreferats bedanken. Für die tolle Begleitung während meines Studiums sowie im Rahmen meiner Promotion einen herzlichen Dank an Michael Gienger, Wolfgang Schotte, Dr. Uwe Wössner und Dr. Bastian Koller vom HLRS.

Außerdem ein großes Dankeschön für das hilfreiche Feedback an die fleißigen Korrekturleser Dr. Jürgen Häring, Marc Habiger, Viola, Justinus und Helmut Feilhauer. Ganz besonders möchte ich mich abschließend bei meiner Familie und meiner Frau Viola bedanken, die mich ununterbrochen mit viel Verständnis während meiner gesamten Ausbildung unterstützt und motiviert haben. Erst hierdurch wurde diese Arbeit möglich. Vielen Dank.

Für meine Mutter.
Ursula Feilhauer
(* 1958, † 2010)

Inhaltsverzeichnis

Zusammenfassung	i
Englische Zusammenfassung (Abstract)	vii
Abkürzungsverzeichnis	xvii
Abbildungsverzeichnis	xxi
Tabellenverzeichnis	xxiii
1 Einleitung	1
1.1 Herausforderungen auf dem Weg zu autonomen Fahrzeugen . .	1
1.2 Stand der Technik	4
1.3 Zielsetzung der Arbeit	12
1.4 Gliederung der Arbeit	13
2 Grundlagen	15
2.1 Fahrerassistenzsysteme	15
2.2 Umfeldsensorik	21
2.3 Test und Freigabe von Fahrerassistenzsystemen	30
3 Absicherungsstrategie	33
3.1 Funktionale Unzulänglichkeit	33
3.2 Ontologische Beschreibung von Verkehrsszenarien	49
3.3 Testfallgenerierung	62
4 Simulation von Fahrerassistenzsystemen	71
4.1 Simulationsarchitektur	72
4.2 ADAS Pipeline	84
4.3 Ideales Sensormodell	89
4.4 Multi-Sensormodell	92
5 Anwendung und Validierung	107
5.1 Anwendung und Nutzen der Absicherungsstrategie	107
5.2 Simulationsarchitektur und Sensormodelle	118
6 Zusammenfassung und Ausblick	135
6.1 Zusammenfassung	135
6.2 Ausblick	136

Anhang	141
A Exemplarische Objektliste	141
B XML-Schema der Szenariobeschreibung	145
C Exemplarische Szenariobeschreibung eines HiL Setups	155
D Szenariobeschreibungen der modellbasierten Testfallgenerierung	157
E Szenariobeschreibungen der Transfersituationen	163
F Systeminformationen	167
Literaturverzeichnis	169

Abkürzungsverzeichnis

ACC	Adaptive Cruise Control
ADAC	Allgemeiner Deutscher Automobil-Club e.V.
ADAS	Advanced Driver Assistance System
ADTF	Automotive Data and Time-Triggered Framework
AI	Artificial Intelligence
BASt	Bundesanstalt für Straßenwesen
BRDF	Bidirectional Reflectance Distribution Function
Car2X	Car-to-X Communication
CFD	Computational Fluid Dynamics
DSP	Digital Signal Processing
EBA	Emergency Brake Assist
EURO NCAP	European New Car Assessment Programme
ESP	Elektronisches Stabilitätsprogramm
FVCWS	Forward Vehicle Collision Warning System
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FMCW	Frequency Modulated Continuous Wave
HiL	Hardware-in-the-Loop
HMI	Human-Machine Interface
ISO	Internationale Organisation für Normung
JSON	JavaScript Object Notation
MBT	Modellbasiertes Testen

MiL	Model-in-the-Loop
NCAP	New Car Assessment Programme
NHTSA	National Highway Traffic Safety Administration
RDE	Real-Driving-Emissions
RCS	Radar Cross Section
ROC	Receiver Operating Characteristics
SAE	Society of Automotive Engineers
SiL	Software-in-the-Loop
SCU	Sensor Control Unit
UUT	Unit Under Test
XiL	X-in-the-Loop
XML	Extensible Markup Language

Abbildungsverzeichnis

1.1	Test zur Funktionsfähigkeit eines Notbremsassistenten	5
1.2	Schematischer Vergleich zwischen Open- und Closed-Loop-Testing	7
2.1	Prinzipskizze eines Notbremsassistenten	18
2.2	Prinzipskizze eines Einparkassistenten	19
2.3	Prinzipskizze eines Abstandsregelautomaten	20
2.4	Prinzipskizze eines Spurhalte-Assistenten	20
2.5	Frequenzspektren unterschiedlicher Sensoren	21
2.6	Trilaterationsprinzip an einem Punkthindernis	23
2.7	Sende- und Empfangssignal eines frequenzmodulierten Dauerstrichradars	24
2.8	Schematische Visualisierung von Wellenfronten an einem Antennen-Array	25
2.9	Idealisierte Lidar Impulsantwort mit drei Objekten im Messkanal	26
2.10	Unterschiedliche Strahlsensoriken	27
2.11	Einfluss unterschiedlicher Kameraparameter	29
2.12	Runde Teststrecke des Zielunterscheidungstests	32
3.1	Poissonverteilungen für $\lambda = 3$	36
3.2	Routenplanung von Stuttgart nach München	39
3.3	Symbol des Szenarienkatalogs	43
3.4	Kontrollflussgraph für Quellcode 3.1	45
3.5	Quellen zur Generierung von Szenarien	47
3.6	Sequenzdiagramm der Absicherungsstrategie	49
3.7	Beispiel einer Bildmodifikation mit dem Ziel ein neuronales Netz zu täuschen	50
3.8	Oberste Hierarchieebene der Szenariobeschreibung	52
3.9	Exemplarische Parameter einer Straße	54
3.10	Wesentliche Elemente von OpenDRIVE	54
3.11	Kontrollflussgraph zur Ausführung dynamischer Vorgänge	62
3.12	Prinzip des modellbasierten Testens	64
3.13	Schematische Visualisierung eines Closed-Loop Tests	66

3.14	Auswahl von Repräsentanten einer Äquivalenzklasse	68
3.15	Vorgehen zur systematischen Testfallgenerierung	68
4.1	Schematische Visualisierung einer Verkehrsszene	72
4.2	Wirkkette von Fahrerassistenzsystemen	73
4.3	Modellbildungsvarianten	74
4.4	Exemplarische Kosimulationsstruktur eines Notbremsassistenten .	77
4.5	Vergleich variabler und fester Berechnungsschrittweiten am Beispiel eines Notbremsassistenten	78
4.6	Aufbau der abstrakten Klasse SimulationObject	81
4.7	Anwendung der SimulationObject Klasse in einer virtuellen Umgebung	83
4.8	Ablaufdiagramm für verteilte Sensorsimulationen	84
4.9	Abstrahierte, funktionale Wirkkette von Fahrerassistenzsystemen	87
4.10	Modellierung einer Fehlerkennung	91
4.11	Klassenstruktur der Typen Geometrie und Material	95
4.12	Schematische Darstellung der bidirektionalen Reflektanzverteilungsfunktion	98
4.13	Prinzipskizze des Backward-Raytracings	100
4.14	Vereinfachte Visualisierung der Ausbreitung eines Primärstrahls sowie daraus resultierende Anteile der Impulsantwort	103
5.1	Transfer realer Verkehrssituationen in die Simulation	108
5.2	Visualisierung des dynamischen Verlaufs eines Einparkszenarios .	110
5.3	Systemaufbau für das Anwendungsbeispiel der Testfallgenerierung	112
5.4	Trajektorien zweier Fahrzeugmodelle für neun generierte Szenarien	114
5.5	Vergleich zwischen Vorhersage und Simulation des Zielzustandes TTC	115
5.6	Visualisierungen probabilistisch generierter Verkehrsszenarien . . .	116
5.7	Minimale TTC-Werte aus probabilistisch erstellten Szenarien . . .	117
5.8	Systematischer Aufbau des HiL Prüfstandes	119
5.9	Trajektorien zweier Fahrzeuge eines HiL Testlaufs	120
5.10	Simulationszeiten involvierter Modelle	121
5.11	Ablaufdiagramm für verteilte Sensorsimulationen	123
5.12	Simulationsdauer in Abhängigkeit der Polygonanzahl	124
5.13	Visualisierungen der Messsituation	126
5.14	Validierung des Ultraschall-Sensormodells (Zylinder)	127
5.15	Impulsantworten des Ultraschall-Sensormodells (Zylinder)	127
5.16	Validierung des Ultraschall-Sensormodells (Quader)	128
5.17	Impulsantworten des Ultraschall-Sensormodells (Quader)	128
5.18	Exemplarische Objekterkennung auf transferierten Szenarien . . .	131

5.19	Receiver Operating Characteristic eines Klassifikators für synthetische und reale Daten	132
6.1	Mit Blender synthetisch erzeugte Bilder	138
6.2	Anhand eines Stiltransfers augmentierte synthetische Daten	139
A.1	Visualisierung der exemplarischen Objektliste	143

Tabellenverzeichnis

2.1	Klassifikationen von Assistenzsystemen	16
2.2	Auflistung von Normen mit Testbeschreibungen für Assistenzsysteme	31
3.1	Mögliche Kombinationen der drei Vergleichsgruppen	38
3.2	Drei exemplarische Einträge eines Objektkatalogs	57
4.1	Modelle einer exemplarischen ADAS Simulation sowie dazugehörige Simulationsdomänen	75
4.2	Sensorrohdaten und verarbeitete Daten unterschiedlicher Sensoren	93
4.3	Vergleich dreier Modellierungsansätze	93
4.4	Kategorisierung der Modellanforderungen	94
4.5	Exemplarisches Fahrzeugmodell für unterschiedliche Sensortypen .	95
4.6	Merkmale der unterschiedlichen Sensortypen	97
A.1	Exemplarische Objektliste mit vier Einträgen	142
A.2	Systeminformationen	167

1. Einleitung

1.1. Herausforderungen auf dem Weg zu autonomen Fahrzeugen

Wird die Anzahl eingereicherter Patentanmeldungen beim deutschen Patent- und Markenamt 2016 als Grundlage zur Bewertung von Innovationsfähigkeit herangezogen, so kann man die Automobilindustrie als eine der innovativsten Branchen Deutschlands bezeichnen. Zehn Unternehmen der Automobil- und Zulieferindustrie führen die Liste der Patenteinreichungen an [DPMA 2016]. Zentrale Forschungsthemen dieser Unternehmen sind neue Geschäftsmodelle für sich ändernde Mobilitätskonzepte², die Einhaltung strengerer Abgasnormen³, die Vernetzung des Fahrzeugs mit seiner Umgebung und die Realisierung autonomer Fahrzeuge [VDA 2017; Guhlich 2017]. Das Themengebiet dieser Dissertation ist in letzterem Forschungszweig angesiedelt.

Vereinfacht lässt sich die Funktionsweise selbstfahrender Automobile wie folgt darstellen: Autonome Fahrzeuge übernehmen die longitudinale und laterale Steuerung des Fahrzeugs. Hierfür wird die Fahrzeugumgebung durch mehrere Sensoren erfasst und deren Informationen verarbeitet, um schließlich als Entscheidungsgrundlage für die Fahrzeugführung dienen zu können. In der Übergangszeit zwischen rein manuellem und vollautomatisiertem Fahren existiert eine Vielzahl an Mischformen, bei denen die Fahrzeugführung zwischen Mensch und Maschine aufgeteilt ist. Beispiele für solche Mischformen sind Notbremsassistenten oder Spurhalteassistenten. Die Umsetzung vollständig selbst-fahrender Fahrzeuge wird innerhalb des nächsten Jahrzehnts anvisiert [Walker 2017; Bhuiyan 2016]. Hierbei übernimmt das autonome System vollständig die Längs- und Querführung und ist in der Lage das Fahrzeug in einen risikominimalen Systemzustand zu überführen [BASt 2012].

Bei der Umsetzung selbst-fahrender Automobile ergeben sich Fragestellungen in unterschiedlichsten Bereichen: Welche rechtlichen und ethischen Rahmenbedingungen gelten für die autonomen Kraftfahrzeuge? Unter welchen Voraussetzungen darf ein Algorithmus eines autonomen Systems über Men-

²Die Firma *Uber* ist beispielsweise „einer der am höchsten bewerteten Mobilitätsanbieter, ohne ein einziges Auto zu produzieren und zu besitzen oder auch nur einen Taxifahrer anzustellen“ [Gassmann 2017, S. 5]. Dieses Geschäftsmodell stellt eine neue Konkurrenzsituation für die traditionellen Automobilunternehmen dar

³Hier ist vor allem der Ansatz der Real-Driving-Emissions (RDE) zu nennen. Bei diesen Abgasmessverfahren werden Emissionen nicht wie bisher unter Laborbedingungen, sondern im realen Straßenverkehr ermittelt. [Gerstenberg 2015]

schenleben entscheiden, diese eventuell gegeneinander abwägen? Wer haftet im Falle eines Unfalls? Es wird intensiv an gesellschaftlich akzeptablen Konzepten gearbeitet, um einen rechtlichen Rahmen für den technischen Fortschritt zu schaffen. Die bisher hierfür relevanten Gesetze basieren auf dem *Wiener Übereinkommen über den Straßenverkehr* aus dem Jahr 1968. Eine Ergänzung des Übereinkommens, die 2016 in deutsches Recht umgesetzt wurde, erlaubt es autonome Fahrzeuge auch auf deutschen Straßen einzusetzen, sofern diese jederzeit übersteuert oder abgeschaltet werden können [Breitinger 2016; United Nations 2014].

Neben rechtlichen und gesellschaftlichen Aspekten befassen sich Automobilhersteller und Zulieferer mit der **technischen Realisierung** und **Absicherung** der Assistenzsysteme. Durch die Art der softwarebasierten Informationsverarbeitung, die ein Fahrzeug zur autonomen Fortbewegung benötigt, stellt die Entwicklung der Assistenzsysteme die Automobilindustrie vor neue Herausforderungen [Segner 2015]. Objekterkennungsalgorithmen prozessieren die Daten der Umfellsensorik, um eine Einschätzung der aktuellen Verkehrssituation zu ermöglichen. Für ein einziges Bild einer Videokamera beispielsweise, gibt es bei einer Auflösung von 800×600 Pixel rund 8 Billionen theoretisch mögliche Farbkombinationen⁴. Ein autonomes System muss den Inhalt solcher Bilder zuverlässig interpretieren. Betrachtet man, dass die Umfellsensorik von Fahrzeugen mehrere Videokameras, Ultraschall-, Radar- und zukünftig auch Lidar-Sensoren beinhaltet, wird deutlich, dass ein vollständiger Test aller möglichen Eingangsparameter nicht realisierbar ist.

Um die (teil-)autonomen Fahrzeuge dennoch sicher zu gestalten und deren gesellschaftliche Akzeptanz zu gewährleisten, ist eine Absicherungsstrategie notwendig. Somit stellt sich die Frage, wie diese Systeme mit vertretbaren zeitlichen und finanziellen Mitteln getestet werden können. [Winner 2015a, 1173 ff.] berechnet exemplarisch den Aufwand zur statistischen Black-Box-Absicherung eines Assistenzsystems. Diese Überlegung veranschaulicht die „Freigabefälle des autonomen Fahrens“: Das simple Beispiel zeigt, dass die Nachweisstrecke für ein einzelnes System schnell eine Größenordnung von 240 Millionen notwendiger Fahrkilometer erreicht. Diese Strecke wäre für den statistischen Nachweis, dass ein autonomes System zuverlässiger als das menschliche Pendant ist, nötig, übersteigt jedoch „die technischen, personellen und wirtschaftlichen Möglichkeiten heutiger Unternehmen“.

Diese Argumentation führt zum Schluss, „dass mit den bekannten Testverfahren [...] keine ökonomisch vertretbare Entwicklung bzw. Zulassung von

⁴Bei 8 Bit Farbtiefe je Kanal ergibt sich die Anzahl der Möglichkeiten zu $2^{(3 \cdot 8)} \cdot 800 \times 600 \approx 8,1 \cdot 10^{12}$.

autonomen Fahrzeugen möglich ist“ [Winner 2015a, S. 1177]. Dies wiederum führt zur Frage, wie die Entwicklung autonomer Systeme effizienter gestaltet und Fahrerassistenzsysteme ökonomisch vertretbar und zuverlässig abgesichert werden können. Diese Herausforderung stellt die Motivation dieser Arbeit dar.

Betrachtet man ethische Aspekte dieser Herausforderung, so lassen sich schnell Dilemmata der Fahrroboter skizzieren, wie beispielsweise das Weichensteller-Problem⁵: Ein außer Kontrolle geratener Zug wird eine Gruppe von Personen überfahren, sofern nicht ein Zeuge einen Weichenhebel bedient, um den Zug auf ein Nebengleis zu steuern, wo *nur* eine Person überfahren werden würde [Vgl. Simanowski 2017]. Menschliche Fahrer müssten in einer entsprechenden Fahrsituationen spontan reagieren: In einer nachgelagerten Analyse kann einem Menschen in einer solchen Notsituation im Allgemeinen kein Vorwurf gemacht werden. Anders sieht es zukünftig jedoch bei autonomen Fahrzeugen aus: Hier wird ein vorab programmiertes System Anweisungen ausführen, um die Fahrbahn des Fahrzeugs zu bestimmen. Bei der Programmierung dieses Systems realisieren Entwickler diese Anweisungen mittels deterministischer Software. Wie in [Birnbacher 2016] formuliert, besteht der „grundlegende Vorzug einer maschinellen Entscheidung gegenüber einer menschlichen Entscheidung ‚vor Ort‘ [...] darin, dass derartige Entscheidungen statt unter Zeit- und Entscheidungsdruck und unter hochgradiger Beteiligung von Emotionen auf einer von unmittelbarem Handlungsdruck entlasteten, rational kontrollierten und allgemeineren Ebene getroffen werden können“. Während heutzutage Menschen in Fahrsituationen spontan reagieren müssen, wird der Ausgang kritischer Szenarien zukünftig durch andere Instanzen festzulegen sein.

Unabhängig von der ethischen Entscheidungsgrundlage ist ein Ziel der Entwicklung von Assistenzsystemen, deren Performance und Robustheit bestmöglich auszulegen. Peter Dabrock, Vorsitzender des deutschen Ethikrats, plädiert bei der Umsetzung autonomer Fahrzeuge für ein pragmatisches Herangehen. Es sei nicht zielführend „immer wieder ethische Dilemmata [zu] inszenier[en], die die Akzeptanz der neuen Fahrtechnik unterlaufen.“ Stattdessen sei es ratsam, einige Grundsätze zu beachten. Unter anderem sei der Transformationsprozess hin zum automatisierten Fahren anhand aktualisierter Dokumentationen immer wieder aufs Neue zu prüfen und zu bewerten. So komme man „schon einige Jahre weiter, [könne] den transformierenden Prozess in beobachtender Teilnahme miterleben sowie bewerten und müsse [sich] nicht von dramatisierenden Gedankenexperimenten gefangen nehmen lassen.“ [Dabrock 2017]

⁵Das Weichensteller-Problem wird häufig auch als Trolley-Problem bezeichnet.

1.2. Stand der Technik

Der Notbremsassistent [Vgl. AUDI AG 2017] und der aktive Spurhalteassistent [Vgl. Daimler-AG 2018] sind zwei Beispiele für Assistenzsysteme, die bereits heute in Serienfahrzeugen verfügbar sind. Nach der Nomenklatur der Bundesanstalt für Straßenwesen gehören diese Systeme zur Kategorie *Assistenz* [Vgl. BASt 2012]. Charakteristisch für diese Systeme ist, dass die Quer- oder Längsführung des Fahrzeugs automatisiert wird, der Fahrer aber jederzeit zur Übernahme der Steuerung bereit sein muss. Rechtlich betrachtet trägt somit momentan stets der Fahrer die Verantwortung über die Fahrzeugführung, auch wenn das Assistenzsystem die Verkehrssituation falsch einschätzen sollte.

Bereits diese Systeme, bei denen der Fahrer stets als Rückfallebene dient, stellen ein potentiell Sicherheitsrisiko im Straßenverkehr dar. Deshalb müssen sie schon heute so ausgelegt werden, dass sie zuverlässig arbeiten und Fehlverhalten so weit wie möglich ausgeschlossen werden kann. Bei einem Notbremsassistenten beispielsweise ist das Ziel einer solchen Auslegung die Vermeidung von Falschauslösungen. Eine sogenannte False-Positive Notbremsung⁶ führt schnell zu einem Auffahrunfall.

Um eine grundsätzliche Funktionalität dieser Systeme zu gewährleisten gibt es mehrere Verbraucherschutztests. Ein bekanntes Testverfahren ist das European New Car Assessment Programme (EURO NCAP). Hierbei werden in den für Automobilhersteller freiwilligen Fahrzeugtests seit 2014 auch Fahrerassistenzsysteme einbezogen. Die Testergebnisse können anschließend als *Sterne-Bewertung* von den Herstellern werbewirksam genutzt werden. Im Rahmen des EURO NCAP gibt es Testverfahren für Notbremsassistenten und Spurhalteassistenten [Euro NCAP 2017]. Qualitativ ähnliche Tests sind auch in den technischen Vorschriften unterschiedlicher Normen zu finden. Beispielsweise definiert die [ISO 22839] Testfälle für Notbremsassistenten. Hierbei werden unterschiedliche Systemfunktionen aufgelistet, die ein System als *Forward Collision Mitigation System* (FCMS) klassifizieren. So ist die Detektion eines vorausfahrenden Fahrzeugs, dessen Abstand und Relativgeschwindigkeit nötig. Zusätzlich muss das System in der Lage sein, die Bremsen des Fahrzeugs zu steuern.

Zentraler Teil dieser Norm ist die Definition konkreter Testfälle. Hierfür werden Begrifflichkeiten definiert, die zur Beschreibung eines Fahr Szenarios notwendig sind. Dies erleichtert und vereinheitlicht das Verständnis innerhalb des Dokuments. Gleichzeitig werden hierdurch die für den Test relevanten

⁶Eine False-Positive Notbremsung ist eine fälschlicherweise als gefährlich klassifizierte Verkehrssituation, die zur Auslösung einer automatischen Notbremsung führt.

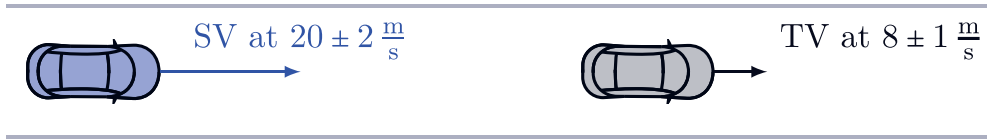


Abb. 1.1: Test zur Funktionsfähigkeit eines Notbremsassistenten nach [ISO 22839]

Elemente benannt. Als *target vehicle*⁷ wird beispielsweise das Fahrzeug verstanden, welches sich auf demselben Pfad des zu testenden Fahrzeugs und in dessen Sensor-Sichtbereich befindet.

Für Testfälle werden die Startbedingungen und erwarteten Reaktionen des Prüflings definiert. Die Beschreibungen lassen Spielraum in der konkreten Umsetzung des Fahr Szenarios, so dass eine reale Testfahrt mit hoher Wahrscheinlichkeit die Testvorgaben erfüllt. Dies erlaubt eine Testdurchführung auch bei nicht beeinflussbaren Randbedingungen. Beispielsweise gibt es keine Einschränkungen für die während des Tests vorherrschenden Lichtbedingungen. Abbildung 1.1 zeigt die schematische Visualisierung eines Tests des Notbremsassistenten, wobei das zu testende Fahrzeug (engl. Subject Vehicle, SV) mit $72 \frac{km}{h}$ auf ein mit $29 \frac{km}{h}$ fahrendes Zielfahrzeug (engl. Target Vehicle, TV) auffährt. Der Test gilt als bestanden, wenn eine Kollisionswarnung und Geschwindigkeitsreduktion erfolgte.

Die Tests des NCAP und der ISO-Normen prüfen die Funktionsfähigkeit von Assistenzsystemen an wenigen Arbeitspunkten. Da keine produktspezifischen Annahmen getroffen werden, eignen sich diese Tests für eine Vielzahl möglicher Systeme. Gleichzeitig handelt es sich hier jedoch um sehr punktuelle Messungen komplexer Systeme. Eine direkte Übertragbarkeit der Leistungsfähigkeit des Systems auf andere Arbeitspunkte ist in der Regel nicht möglich [Vgl. Seiniger 2015, S. 181]. Deshalb bieten diese punktuellen Testverfahren keine breite Testabdeckung der Assistenzsysteme.

Um das System an vielen Arbeitspunkten zu testen und eine breite Testabdeckung zu erhalten, setzen Automobilhersteller auf reale Testfahrten. Mit Sonderfahrgenehmigungen ausgestattete Prototypen fahren im öffentlichen Straßenverkehr [Vgl. Volvo Cars 2017; Ohnsman 2017]. Bei großangelegten Flottentests ergeben sich zufallsbedingt unterschiedliche Fahrsituationen, innerhalb derer die Funktionalität des Systems beobachtet werden kann. Neben eventueller Gefährdung von Verkehrsteilnehmern sowie hohem Kostenaufwand für teure Prototypen stellt die fehlende Systematik und mangelnde Reproduzier-

⁷ „forward vehicle that is within the effective range of the subject vehicle (SV)’s forward ranging sensor“ [ISO 22839, S. 11]

barkeit der Fahrscenarien einen Nachteil dieses Ansatzes dar. Verwendet man die Klassifizierung von Softwaretesttechniken als Grundlage, so lassen sich die Testfahrten als Zufallstest klassifizieren: „Der Zufallstest (random test) ist eine Testtechnik, die aus den Wertebereichen der Eingabedaten zufällig Testfälle erzeugt. [Ihm] liegt [...] keine deterministische Strategie zugrunde.“ [Liggesmeyer 2002, S. 208] Teilweise wird eine grobe Gruppierung der Testfahrten vorgenommen, etwa in *Überland*, *Autobahn* und *Innerorts*. Hier wird der Eingabebereich des Systems in Teilbereiche zerlegt, jedoch ist diese Klassifizierung sehr grob und unterscheidet sich deshalb im Hinblick auf die Systemkomplexität kaum vom reinen Zufallstest.

Um der mangelnden Reproduzierbarkeit von Testfahrten zu begegnen, werden Sensorsignale während der Fahrt aufgezeichnet. Dies erlaubt die exakte Wiedergabe einer Testfahrt im Nachgang. Wird ein Fehlverhalten in einer bestimmten Fahr-situation entdeckt, so können die aufgenommenen Sensorinformationen weiterhin als Eingabeparameter des zu testenden Algorithmus dienen. Wird dessen Logik angepasst, so ist die Verwendung der aufgenommenen Sequenzen limitiert: Auf Grund der fehlenden Rückkopplung (Open-Loop Testing) kann der modifizierte Algorithmus, sofern er die Aktorik des Fahrzeugs beeinflusst, nicht mehr mit den vorab aufgezeichneten Sensorsignalen getestet werden.

Folgendes Beispiel illustriert diese Eigenschaft: Im Rahmen einer Testfahrt wird erkannt, dass ein automatischer Spurwechsel nicht bei einem Abstand von 12 Metern zum Vorderfahrzeug, sondern erst bei 10 Metern eingeleitet werden sollte. Nach Anpassung des Algorithmus können die während der Testfahrt aufgezeichneten Sensordaten nicht weiterhin als Testdaten verwendet werden. Die Daten spiegeln die Änderungen des Algorithmus auf Grund der mangelnden Rückkopplung nicht wider. Für solche Entwicklungssituationen ist der Ansatz des Open-Loop-Replay von Sensordaten deshalb nicht zielführend. Abbildung 1.2 visualisiert schematisch den Vergleich zwischen dem Vorgehen bei Open- und Closed-Loop-Tests.

Im Open-Loop Fall werden reale Sensordaten aufgezeichnet und anschließend für Regressionstests der Unit Under Test (UUT) wiederverwendet. Dadurch stehen Testdaten für die iterative Anpassung eines Algorithmus zur Verfügung, sofern dieser keine Rückkopplung zur Fahrzeugbewegung besitzt. Demgegenüber wird beim Closed-Loop Aufbau die UUT mittels synthetisch erzeugter Daten stimuliert. Die Reaktion kann wiederum die Generierung der Simulationsdaten beeinflussen.

Um gleichzeitig eine breite Testabdeckung sowie die Möglichkeit einer Rückkopplung zu erhalten, werden Testfahrten mittels Simulation in virtuelle Umgebungen transferiert. Dieser Transfer realer Verkehrssituationen in virtuelle Umgebungen bedarf einer Vielzahl unterschiedlicher Modelle. Da Fahrerassis-

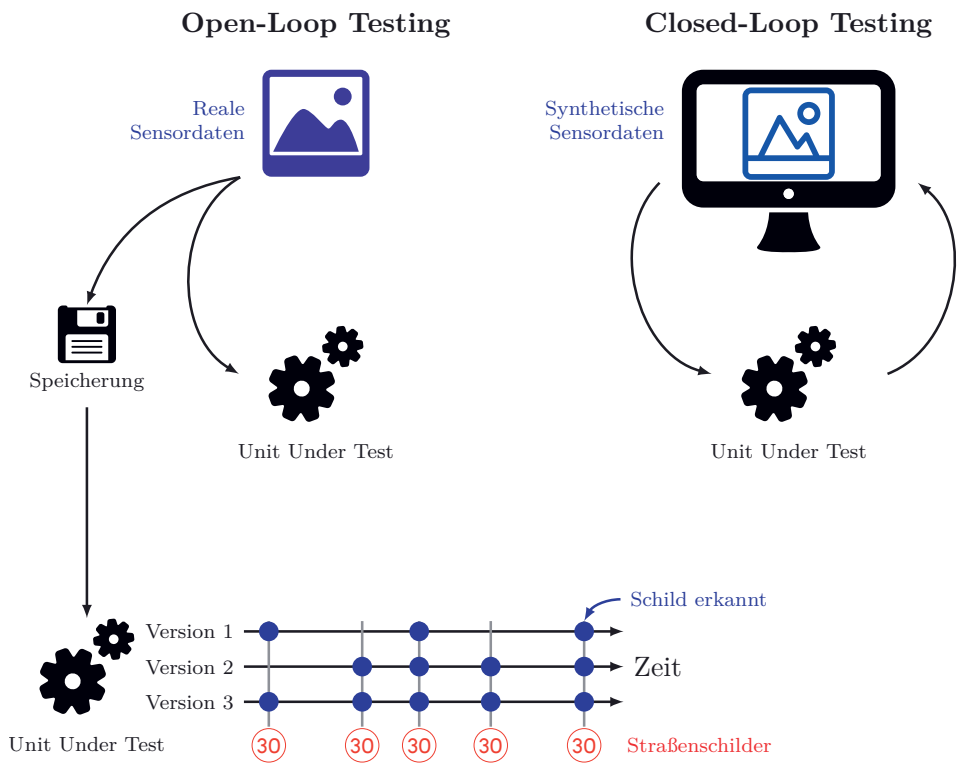


Abb. 1.2: Schematischer Vergleich zwischen Open- und Closed-Loop-Testing

tenzsysteme die Umgebung durch Sensoren erfassen, ist es auch notwendig Umgebungsobjekte in die Simulation zu überführen. So sind neben einem Modell des Eigenfahrzeugs mindestens auch Fahrermodelle, Sensormodelle und Straßenmodelle notwendig. Diese Aufzählung lässt sich beliebig fortführen, wenn weitere Elemente, wie Wettereinflüsse oder zusätzliche Verkehrsteilnehmer, mit in die Simulation aufgenommen werden sollen. Die Architektur der dafür notwendigen Gesamtsimulation unterliegt unterschiedlichsten Randbedingungen: Bisherige Simulationsmodelle sollen möglichst wiederverwendbar sein; für einen durchgängigen Entwicklungsprozess ist eine hohe Modularität notwendig, um je nach Arbeitszwischenstand einzelne Module erweitern und austauschen zu können; für die Einbindung realer Komponenten in die Testumgebung ist eine hohe Performance der Gesamtsimulation notwendig. Da sich die Spieleindustrie seit Jahren ausgiebig mit der performanten Darstellung virtueller Welten beschäftigt, scheint es sinnvoll, hierbei entstandene Ansätze zu verwenden.

In [Neumann-Cosel 2014, S. 75 ff.] wird der Aufbau eines Simulationswerkzeugs dargestellt und auf die Verwendung der Module *Fahrzeugumfeld*, *Verkehrssimulation*, *Visualisierung* sowie *Sensormodelle* eingegangen. Außerdem wird der Abgleich zwischen virtuellen und realen Sensorsignalen exemplarisch durchgeführt. [Hanke 2015] beschreibt eine mögliche Softwarearchitektur mit dem Fokus auf die Signalverarbeitung der Sensorik. Durch den Vorschlag einer standardisierten Schnittstelle auf Objektlisten-Ebene erlaubt dieser Ansatz die Ankopplung unterschiedlicher Fahrdynamiksimulationen. Für am Markt verfügbare Werkzeuge, wie beispielsweise [CarMaker] oder [PreScan], gibt es relativ wenig Einblick in die verwendete Softwarearchitektur und die konkrete Umsetzung der Umgebungssimulation. Dem Autor sind keine weiterführenden Quellen bekannt, die eine Softwarearchitektur für den Simulationsaufbau von Fahrerassistenzsystemen adressieren. Somit fehlt eine Beschreibung unterschiedlicher Abstraktionsebenen einer solchen Simulationsumgebung. Zur nachhaltigen Realisierung ist eine modulare Struktur der Umgebungssimulation ein zentraler Aspekt. Deshalb ist eine Beschreibung der Simulationsarchitektur mit Fokus auf das Umgebungsmodell notwendig. Eine solche Architektur wird im Rahmen dieser Arbeit entwickelt.

Demgegenüber sind in der Literatur mehrere Ansätze zur Modellierung virtueller Sensoren zu finden. Hierbei stehen Radar-, Video-, Ultraschall- sowie Lidar-Sensoren im Fokus der Modellbildung, da vorrangig diese vier Umgebungssensoren bei der Entwicklung von Assistenzsystemen Verwendung finden [Vgl. Niehsen 2005]. [Schubert 2014] teilt die Modellierungsansätze in drei qualitativ unterschiedliche Kategorien ein: *Ground Truth Modelle* stellen idealisierte Sensoren dar, die Objekte innerhalb eines Sichtbereichs detektieren

und Referenzinformationen liefern. Das mögliche Ergebnis eines solch idealisierten Modells stellen der exakte Abstand und die Relativgeschwindigkeit eines detektierten Fahrzeugs dar. Diese Modelle bieten aufgrund der einfachen Modellierung eine hohe Performance, bilden jedoch keine charakteristischen Eigenschaften eines speziellen Sensortyps ab. *Physikalische Modelle* versuchen einzelne Sensortypen sehr detailliert abzubilden, indem das physikalische Prinzip der Strahlenausbreitung und Detektion modelliert wird. Diese Ansätze modellieren teilweise zusätzlich auch Elemente der Signalverarbeitung. Modelle dieser Kategorie haben meist einen hohen Rechenaufwand und können nur schwierig an andere Sensortypen angepasst werden. Das physikalisch motivierte Modell eines Videosensors ist beispielsweise nur unter hohem Aufwand in ein Lidar-Sensormodell zu transferieren. *Probabilistische Modelle* reichern idealisierte Modelle um spezifische Sensorcharakteristiken an, wobei die jeweiligen Sensorfehler unter Verwendung von statistischen Daten abgebildet werden. So kann beispielsweise die Position eines vom idealen Sensormodell erkannten Objektes statistisch variiert werden, um den realen Sensorfehler, wie er bei einer Radar-basierten Objekterkennung auftreten würde, zu approximieren. Diese Modelle liefern bei hoher Performance realistischere Sensorcharakteristiken als Ground Truth Modelle. Hierfür müssen jedoch die jeweils passenden statistischen Daten zur Verfügung stehen.

In [Bernsteiner 2015] wird ein Radarsensormodell vorgestellt, das sich durch eine Kombination aus phänomenologischer und realer Modellierung von Sensoreigenschaften für den Einsatz in Echtzeitsimulationen eignet. Dieses Kriterium ist entscheidend dafür, dass die Simulationsmodelle durchgängig im Entwicklungsprozess eingesetzt werden können, was auch die Hardware-in-the-Loop (HiL) Anwendungen einschließt. Das vorgestellte Modell operiert auf mittels Quadern abstrahierten Modellen der Umgebungsfahrzeuge. Zusätzlich werden Verdeckungseffekte modelliert, indem die Strecke zwischen Sender und Umgebungsfahrzeugen entlang ausgewählter Verbindungspfade auf Hindernisse⁸ überprüft wird. Dies erlaubt eine bessere Annäherung der Distanzinformationen zwischen Sensor und Hindernis, als es rein phänomenologische Modelle leisten könnten. Die Limitierung des Modells auf die Aussendung einzelner Strahlen erlaubt einerseits schnelle Rechenzeiten, vereinfacht dennoch sehr die reale Strahlenausbreitung eines Radarsensors.

Die vorgestellten Modelle abstrahieren das reale Sensorverhalten stark (Ground Truth Modelle), benötigen Statistiken zur Modellierung des Fehlverhaltens (Probabilistische Modelle) oder sind rechenintensiv und schwierig auf andere Modelle übertragbar (Physikalische Modelle). Deshalb ist ein Modellierungsansatz nötig, der bei akzeptablem Rechenaufwand die physikalischen

⁸Dies stellt eine Anwendung von vereinfachtem Raytracing (Strahlenverfolgung) dar.

Wirkprinzipien der vier gängigen Sensortypen für Fahrerassistenzsysteme abbilden kann. Gleichzeitig sollte sich ein solches Modell in die gleiche Simulationsarchitektur wie vorhandene Ansätze eingliedern lassen, um bestehende Werkzeuge weiterverwenden zu können. Im Rahmen dieser Arbeit wird deshalb ein Modellierungsansatz diskutiert, um die von autonomen Fahrzeugen eingesetzten Umgebungssensoren zu virtualisieren.

Damit die virtuellen Testfahrten innerhalb des Entwicklungsprozesses belastbare Informationen liefern ist die Validierung der Simulation notwendig. In [Neumann-Cosel 2014, S.91 ff.] werden ein ideales Sensormodell und ein Kameramodell anhand einzelner Fahrszenarien mit realen Sensorinformationen verglichen und die Modelle somit für diese Szenarien validiert. Dieses Vorgehen zeigt die Güte der spezifischen Sensormodelle für einzelne Fahrszenarien. Allerdings wird keine Aussage über die Inter- oder Extrapolationsfähigkeit der Modelle getroffen. Was geschieht bei leichter Variation der virtuellen Verkehrsszenarien? Ist das Sensormodell beispielsweise auch aussagefähig, wenn es nicht das virtuelle Fahrzeugmodell A, sondern das etwas kleinere Fahrzeugmodell B detektiert? Eine solche Einschätzung ist notwendig, wenn Assistenzsysteme in großem Umfang mittels virtueller Umgebungen getestet werden. Eine Steigerung der Effektivität durch den Einsatz von Simulation ist nur möglich, wenn nicht jedes virtuelle Fahrszenario ein reales Validierungspendant benötigt.

Um die Effektivität der Absicherung entlang des gesamten Entwicklungsprozesses zu erhöhen, ist es notwendig, dass ein einheitlicher Zugang entlang aller Entwicklungsphasen zur Virtualisierung vorhanden ist. Müssen Simulationsmodelle beispielsweise zwischen der ersten Visualisierung von Kundenanforderungen und dem Systemdesign auf unterschiedliche Simulationsframeworks transferiert werden, so stellt dies einen erheblichen unnötigen Zeitaufwand dar. Dies gilt es unbedingt zu vermeiden und mittels einer durchgängigen Teststrategie entlang der Model-, Software- und Hardware-in-the-Loop Testphasen des V-Entwicklungsmodells zu gewährleisten.

[Hakuli 2015] beschreibt, wie unter Verwendung des FMI/FMU-Mechanismus eine standardisierte Möglichkeit besteht, um Simulationsmodelle geschützt zwischen unterschiedlichen Herstellern auszutauschen. Die Modelle können unabhängig voneinander in spezialisierten Tools für die jeweilige Problemstellung implementiert werden. Ein standardkonformer Export erlaubt anschließend die Komposition der einzelnen Modelle zu einer Gesamtsimulation. Während der Entwicklung des FMI-Standards lag der Fokus primär auf dem Austausch physikalischer Simulationsmodelle des Antriebsstrangs. Daraus resultiert, dass sich die Interfaces für diesen Anwendungsfall eignen. So kann beispielsweise der Datenaustausch zwischen den unterschiedlichen Simulationsmodellen anhand einer vorab definierten Anzahl reellwertiger Zahlen erfolgen. Dynamische

Längen der Schnittstellen oder eine höhere Abstraktionsebene, beispielsweise der Austausch von komplexeren Strukturen, sind nicht vorgesehen. Für die Kopplung von Sensormodellen, die beispielsweise synthetische Videobilder generieren, sind komplexere Schnittstellen notwendig.

[Hanke 2017] präsentiert den Entwurf eines Interfaces, das sich besser für den Austausch von Daten in Advanced Driver Assistance System (ADAS) spezifischen Anwendungsfällen eignet. Hierbei wird die Objektebene als Schnittstelle vorgeschlagen. Dieses Interface ermöglicht einen Informationstransfer von einer virtuellen Welt an generische Sensormodelle. Betrachtet man jedoch die Wirkkette von Assistenzsystemen, so stellt diese nur eine von mehreren möglichen Schnittebenen dar. Analog dazu sind Anwendungsfälle denkbar, bei denen die Übertragung von Sensorrohdaten notwendig ist. Für Tests eines zentralen Steuergeräts für Assistenzsysteme könnte die Übertragung mehrerer Sensorrohdaten notwendig sein, wie beispielsweise der Transfer von Videobildern. Diese Art des Datenaustauschs ist mit bisher vorgeschlagenen standardisierten Interfaces nicht möglich.

Neben dem Modellaustausch ist zudem der Austausch einer Beschreibung des zu simulierenden Verkehrsszenarios nötig. Dies ermöglicht die Erstellung eines herstellerunabhängigen Szenarienkatalogs. Hierfür wird beispielsweise im Rahmen des Projekts „PEGASUS“ [Mazzega 2017] ein Vorschlag erarbeitet, wie Fahrscenarien maschinenlesbar beschrieben werden können. Fokus der dabei entstehenden Ontologie ist die Realisierung eines Austauschs von Fahrscenarien zwischen unterschiedlichen Simulationswerkzeugen. Aufgrund der Komplexität der Gesamtsimulation ist das Ziel einer solchen einheitlichen Szenariobeschreibung nicht, dass die Simulatoren beim Austausch der Szenarien die exakt gleichen Ergebnisse liefern. Hierfür wären der Austausch aller involvierten Parameter sowie eine direkte Vergleichbarkeit der verwendeten Modelle nötig. Dennoch ist die Möglichkeit einen Katalog von Fahrscenarien an unterschiedlichen Simulationsumgebungen zu verwenden sinnvoll, um einmalig beschriebene Testfälle auf mehreren Systemen ausführen zu können. Während der Entwicklung von Assistenzsystemen sind die relevanten Einflussparameter leider nicht immer vorab bekannt. Aufgrund der Komplexität einer computerbasierten Umgebungswahrnehmung ist beispielsweise der Einfluss der Umgebungsbeleuchtung nur schwer zu spezifizieren. So ist es möglich, dass sich gewisse Parameter während der Entwicklung als relevant herausstellen. Eine Szenariobeschreibung muss diesem Zustand Rechnung tragen und solche Parameter zusätzlich mit aufnehmen. Diese iterative Sichtweise unterscheidet sich von der Herangehensweise der Parameterauswahl, wie sie in [Mazzega 2017] gezeigt wird. Zur Generierung eines konsistenten Testkatalogs ist eine iterative Erweiterung der Testbeschreibung bei neuem Erkenntnisgewinn essentiell.

1.3. Zielsetzung der Arbeit

Bei der Umsetzung von autonomen Fahrzeugen gibt es in unterschiedlichen Disziplinen noch große Herausforderungen. Neben ethischen und gesellschaftlichen Aspekten stellt sich im technischen Umfeld die Frage, wie die Freigabe von Assistenzsystemen effizient und zuverlässig gestaltet werden kann.

Die vorgestellten Freigabenormen beschreiben Testverfahren, um einzelne Arbeitspunkte von Assistenzsystemen zu prüfen. Um eine breite Testabdeckung zu erzielen, werden Prototypen mit Sondererlaubnis in Flottentests auf der Straße erprobt. Virtuelle Testfahrten wiederum liefern mit eigenen Testbeschreibungen die Möglichkeit, Szenarien gefahrlos und gut parallelisierbar zu testen. Unabhängig von der Durchführung von Testfahrten ist eine Beschreibung der Verkehrsszenarien notwendig, mittels derer sich reale Testfahrten oder virtuelle Erprobungen definieren lassen. Die Umfeldsensorik zur Erfassung des Verkehrsgeschehens bedingt eine komplexe Datenverarbeitung. Die zu Beginn einer Entwicklung erstellte Spezifikation für ein Assistenzsystem kann nicht alle möglichen Verkehrsszenarien und relevante Parameter beinhalten. Eine Szenariobeschreibung, die sich als Grundlage einer durchgängigen Absicherungsstrategie eignet, muss deshalb kontinuierlich erweiterbar sein. Ein Ziel dieser Arbeit ist die Vorstellung eines Absicherungsansatzes, der frühzeitig in einer Systementwicklung eingesetzt werden kann. Basis dieses Ansatzes soll eine maschinenlesbare und erweiterbare Beschreibung von Fahrscenarien sein. Dadurch wird der Einsatz realer und virtueller Testfahrten im Rahmen der Absicherung ermöglicht.

Eine solche Szenariobeschreibung, welche die als relevant identifizierten Einflussparameter beinhaltet, kann zudem zur Beschreibung konkreter Testfälle genutzt werden. Um eine möglichst hohe Testabdeckung des Assistenzsystems nicht dem Zufall zu überlassen, ist eine systematische Generierung von Testfällen sinnvoll. Im Rahmen dieser Arbeit soll ein solcher Ansatz zur automatisierten Generierung erarbeitet werden.

Etabliert sich ein umfangreicher Katalog von Testszenarien, so gilt es diese Szenarien möglichst effektiv abzufahren. Zur Beschleunigung der Testfahrten stellt deren Simulation eine mögliche Lösung dar. Für den Transfer realer Verkehrsszenarien in virtuelle Welten sind Simulationsmodelle notwendig. Ein zentrales Element der Simulation von Assistenzsystemen ist die Modellierung der Umfeldsensorik. Hier werden sehr vereinfachende ideale Sensormodelle oder probabilistische Modelle eingesetzt, deren Performance auch im HiL Kontext ausreichend ist. Physikalische Sensormodelle erscheinen bisher zu spezialisiert auf einen bestimmten Sensortyp sowie zu rechenintensiv. Deshalb stellt die Umsetzung eines physikalisch-motivierten Sensormodells ein weiteres

Ziel dieser Arbeit dar. Dieses Modell soll sich entlang aller X-in-the-Loop (XiL)⁹ Stufen einsetzen lassen und nativ die Abbildung unterschiedlicher Sensortypen ermöglichen.

1.4. Gliederung der Arbeit

Im Anschluss an diese Einleitung werden in **Kapitel 2** einige **Grundlagen** vorgestellt, die dem Verständnis dieser Arbeit dienen. Hierbei werden mehrere Assistenzsysteme sowie deren Klassifizierung erläutert. Daran schließt sich eine Übersicht der bei Assistenzsystemen verwendeten Sensorik an. Mit einem Überblick zu Testmethoden für Fahrerassistenzsysteme schließt das Grundlagenkapitel ab.

Motiviert durch das Ziel der Aufwandsreduktion im Vergleich zum statistisch basierten Black-Box-Ansatz wird in **Kapitel 3** eine **Absicherungsstrategie** für Fahrerassistenzsysteme entworfen. Basierend auf einer Ausführung zur sogenannten *Funktionalen Unzulänglichkeit* wird ein Testkonzept entworfen, das eine praxisnahe und transparente Absicherung teilautonomer Fahrzeuge ermöglichen soll. Eine zentrale Rolle hierbei spielt die semantische Beschreibung des Verkehrsgeschehens. Deshalb wird eine Möglichkeit zur Szenariobeschreibung präsentiert. Darauf aufbauend schließt sich ein Ansatz zur systematischen Generierung von Testfällen an.

Um eine Absicherung durch Anwendung von Simulation zu beschleunigen, wird in **Kapitel 4** ausführlich auf die **Simulation von Fahrerassistenzsystemen** eingegangen. Zu Beginn wird eine Simulationsarchitektur entworfen, die sich aus der Darstellung dreier unterschiedlicher Sichtweisen auf das Verkehrsszenario ableitet. Damit einhergehend werden numerische Aspekte bei der Kopplung unterschiedlicher Simulationsmodule diskutiert. Im Anschluss wird die Wirkkette von Assistenzsystemen abstrahiert, um die anschließende Modellbildung zu unterstützen. Dieser Verallgemeinerung schließt sich die Modellbildung eines einfachen idealen Sensormodells sowie eines physikalisch-motivierten Sensormodells an.

Zur Bewertung des vorgestellten Absicherungsansatzes und der Modelle befasst sich **Kapitel 5** mit deren **Validierung**. Hierbei wird auf die Vor- und Nachteile der Absicherungsstrategie eingegangen sowie Anwendungen der Sensormodelle detailliert betrachtet. Dies ermöglicht eine Bewertung der präsentierten Ansätze.

Abschließend folgt in **Kapitel 6** eine **Zusammenfassung** der Ergebnisse dieser Arbeit sowie ein **Ausblick** auf weiterführende Forschungsfragen.

⁹X-in-the-Loop steht zusammenfassend für Model-, Software- und Hardware-in-the-Loop.

2. Grundlagen

In diesem Kapitel wird die Funktionsweise von Assistenzsystemen präsentiert, vorhandene Klassifikationen der Systeme aufgelistet sowie einige Beispiele von Assistenzsystemen in Serienfahrzeugen dargestellt. Zudem werden vier Sensoren vorgestellt, die zur Detektion des Fahrzeugumfelds verwendet werden. Abschließend werden derzeitige Test- und Freigabeverfahren der Systeme präsentiert.

2.1. Fahrerassistenzsysteme

Systeme zur Unterstützung des Fahrers gibt es bereits in der frühen Automobilgeschichte. So kam beispielsweise der elektrische Anlasser erstmalig im Jahre 1912 in einem Cadillac zum Einsatz. Er diente als Ersatz für die manuelle Anlasserkurbel. Der Anlasser und Systeme wie der Bremskraftverstärker, das Automatikgetriebe oder die Servolenkung sind inzwischen technischer Standard in Automobilen [Reif 2010, S. 209].

Aktuelle Fahrerassistenzsysteme unterstützen den Fahrer bei der Fahraufgabe. Hierfür greifen sie, abhängig vom Automatisierungsgrad, in die Längs- und Querführung der Fahrzeugbewegung ein. Zur Kategorisierung unterschiedlicher Automatisierungsgrade gibt es diverse Vorschläge. Die Bundesanstalt für Straßenwesen (BASt) definiert zur rechtlichen Einordnung der Systeme fünf Automatisierungsgrade. Auch die US-Bundesbehörde National Highway Traffic Safety Administration (NHTSA) sowie die Society of Automotive Engineers (SAE) haben eine Kategorisierung der Assistenzsysteme vorgeschlagen. Tabelle 2.1 führt die Kategorien nach [SAE 2014] auf. Hierfür sind neben der Beschreibung einzelner Kategorien die Kriterien der Zuordnungen aufgelistet. Dabei wird stets die Aufteilung der Fahraufgabe zwischen Fahrer und System betrachtet. Diese Aufgabe teilt sich in Fahrzeugführung (**Aktion**), Überwachung der Fahrzeugumgebung (**Überwachung**) und Überführung des Fahrzeugs in einen sicheren Zustand (**Fallback**). Zudem kann der freigegebene Einsatzort limitiert sein (**Szenarien**). Wird das Fahrzeug vollständig von einem menschlichen Fahrer gesteuert, wie es bisher in der Regel der Fall ist, so entspricht dies der SAE Kategorie *Keine Automatisierung*.

Ein Abstandsregelautomat (engl. Adaptive Cruise Control, ACC) kann automatisch den Abstand zu einem Vorderfahrzeug anpassen. Das System greift durch Brems- und Beschleunigungsvorgänge selbständig in die Längsdynamik ein. In heutigen Serienprodukten ist dies auf bestimmte Geschwindigkeitsberei-

Beschreibung	Aktion	Überwachung	Fallback	Szenarien	BAST	NHTSA	SAE
Volle Kontrolle der Fahraufgabe durch den Fahrer	Fahrer	Fahrer	Fahrer	-	Driver Only	0	Keine Automatisierung
Quer- oder Längsführung teilweise automatisiert	Fahrer & System	Fahrer	Fahrer	einzelne	Assistiert	1	Fahrerassistenz
Quer- und Längsführung teilweise automatisiert Fahrer als Rückfallebene	System	Fahrer	Fahrer	einzelne	Teilautomatisiert	2	Teilautomatisiert
Quer- und Längsführung teilweise automatisiert Übernahme nach Aufforderung	System	System	Fahrer	einzelne	Hochautomatisiert	3	Bedingte Automatisierung
Quer- und Längsführung teilweise automatisiert Risikominimaler Zustand möglich	System	System	System	einzelne	Vollautomatisiert	3/4	Hochautomatisiert
Quer- und Längsführung immer automatisiert	System	System	System	alle	-	3/4	Vollautomatisiert

Tabelle 2.1: Klassifikationen von Assistenzsystemen nach [SAE 2014]

che und Situationen beschränkt, beispielsweise auf Kolonnenverkehr auf der Autobahn. Somit stellt ein ACC-System einen Repräsentanten der Kategorie *Fahrerassistenz*¹⁰ dar.

Ein Spurhalteassistent hält das Fahrzeug auf dem vorgesehenen Fahrstreifen. Wird ein unerwünschtes Verlassen des Fahrstreifens diagnostiziert, so greift das System korrigierend in die Querdynamik des Fahrzeugs ein. Die Kombination der Systeme Spurhalteassistent und Abstandsregelautomat in einem Fahrzeug kann als *Teilautomatisiert*¹⁰ klassifiziert werden.

Die bisherigen Beispiele unterstützen den Fahrer bei der Fahraufgabe. Dieser muss jedoch stets in der Lage sein das Fahrzeug selbst kontrollieren zu können. Deshalb überprüfen aktuell verfügbare Assistenzsysteme teilweise den Aufmerksamkeitszustand des Fahrers, um möglichen Missbrauch der Systeme zu reduzieren. Wird mangelnde Achtsamkeit des Fahrers diagnostiziert, beispielsweise über eine Sensorik im Lenkrad zur Detektion der Handinnenflächen, so werden akustische oder haptische Warnungen emittiert. Reagiert der Fahrer auf das beispielsweise vibrierende Lenkrad nicht, so schaltet sich das Assistenzsystem ab. [Vgl. Urhahne 2013]

Für Systeme der Kategorie *Bedingte Automatisierung*¹⁰ und *Hochautomatisiert*¹⁰ reduziert sich diese direkte Aufsichtspflicht des Fahrers. Ein solches System darf sich erst dann abschalten, wenn der Fahrer die Fahraufgabe übernimmt. In kritischen Situationen kann das System Warnhinweise geben und den Fahrer zur Übernahme auffordern, muss bei System ab der Kategorie *Hochautomatisiert*¹⁰ aber selbständig in der Lage sein das Fahrzeug in einen risiko-minimalen Zustand zu überführen. Ein solcher Zustand könnte beispielsweise das kontrollierte Abstellen des Fahrzeugs auf dem Seitenstreifen darstellen. Diese Fallback-Fähigkeit des Systems kann sich auf einzelne Einsatzorte beschränken.

Ein System, welches die Fahrdynamik und den Übergang in einen risiko-minimalen Zustand für alle Fahrscenarien realisiert, wird als *Vollautomatisiert*¹⁰ kategorisiert.

Zur Darstellung der Funktionalität aktueller Fahrerassistenzsysteme werden nachfolgend vier Systeme vorgestellt. Hierzu wird jeweils die Funktionalität beschreiben, die verwendete Sensorik aufgelistet sowie eine Prinzipskizze präsentiert.

¹⁰nach SAE Nomenklatur

Notbremsassistent

Als Beispiel wird das System *Audi pre sense plus* verwendet [AUDI AG 2017]. Der Notbremsassistent kann im Falle eines bevorstehenden Unfalls die Fahrzeuggeschwindigkeit verringern, um so den Unfall zu vermeiden oder den Unfallschaden zu verringern. Das System unterstützt den Fahrer bei einer drohenden Frontalkollision in mehreren Stufen. Bei nahender Kollision wird zuerst eine akustische Warnung an den Fahrer gesendet. Anschließend wird mittels eines kurzen Bremsrucks versucht die Aufmerksamkeit des Fahrers auf das Unfallszenario zu lenken. Löst der Fahrer daraufhin selbst einen Bremsvorgang aus, so wird dieser vom System unterstützt. Reagiert der Fahrer nicht, so wird die Warnblinkanlage des Fahrzeugs aktiviert, die Fenster geschlossen und der Gurt gestrafft. Außerdem leitet das System im Falle eines unvermeidbaren Unfalls selbständig eine Vollbremsung ein. Dies kann zu einer Reduktion der Aufprallgeschwindigkeit um bis zu $40 \frac{\text{km}}{\text{h}}$ führen. Als Umfeld-Sensoren werden zwei Front-Radarsensoren sowie eine Videokamera verwendet. Abbildung 2.1 visualisiert eine Prinzipskizze des beschriebenen Notbremsassistenten.

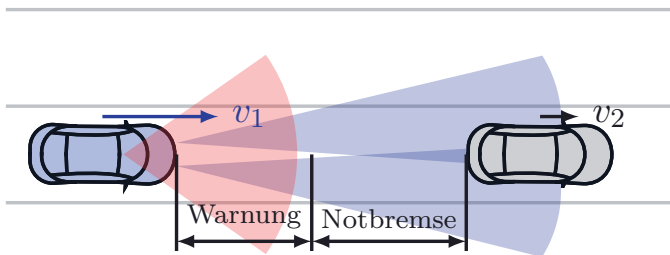


Abb. 2.1: Prinzipskizze eines Notbremsassistenten

Einparkassistent

Als Beispiel wird das System *Volkswagen Park Assist* verwendet [Volkswagen 2016].

Das System übernimmt die Lenkbewegung des Fahrzeugs während eines Einparkvorgangs. Zuerst aktiviert der Fahrer hierfür das System und fährt mit maximal $40 \frac{\text{km}}{\text{h}}$ an einer passenden Parklücke vorbei. Anschließend ist das System in der Lage die Querdynamik des Fahrzeugs zu übernehmen. Der Fahrer legt den Rückwärtsgang ein und ist für die Steuerung der Längsdynamik verantwortlich.

Falls die Parkposition nicht direkt erreicht werden kann, so wird der Fahrer aufgefordert vorwärts und gegebenenfalls erneut rückwärts zu fahren. Das System lässt sich jederzeit übersteuern. Als Umfeld-Sensoren werden mehrere Ultraschallsensoren verwendet. Abbildung 2.2 visualisiert eine Prinzipskizze des beschriebenen Einparkassistenten.

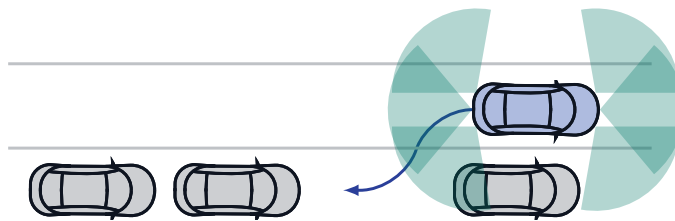


Abb. 2.2: Prinzipskizze eines Einparkassistenten

Abstandsregelautomat

Als Beispiel wird das System *Aktive Geschwindigkeitsregelung* der BMW AG verwendet [BMW 2013].

Mit Hilfe der aktiven Geschwindigkeitsregelung kann die Fahrzeuggeschwindigkeit an ein vorausfahrendes Fahrzeug angepasst werden. Das für Autobahnen und Schnellstraßen konzipierte System erfasst Verkehrsteilnehmer in einem Bereich von 150 Metern in Fahrtrichtung. Die Abstandsregelung erfolgt über Eingriffe in die Motorsteuerung oder Bremsen. Die Regelung basiert auf der Messung des zeitlichen Abstands zum Vorderfahrzeug. Dadurch kann das Fahrzeug im Falle eines Staus zum Stillstand gebracht werden. Sofern möglich beschleunigt das Fahrzeug anschließend wieder selbständig, wenn die Standzeit kürzer als drei Sekunden war. Das System lässt sich jederzeit übersteuern. Erkennt das Assistenzsystem eine kritische Situation, so wird der Fahrer akustisch und optisch zur Übernahme aufgefordert. Als Umfeld-Sensoren werden drei Radarsensoren in Fahrtrichtung verwendet. Abbildung 2.3 visualisiert eine

Prinzipskizze des beschriebenen Abstandsregelautomaten.

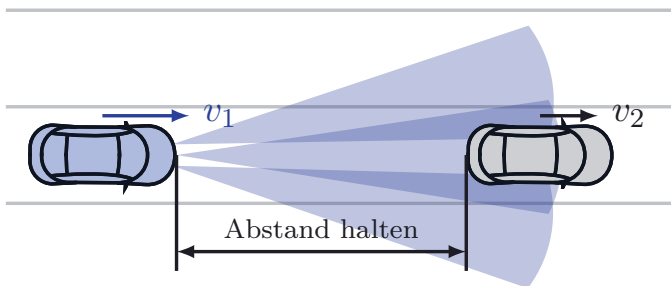


Abb. 2.3: Prinzipskizze eines Abstandsregelautomaten

Spurhalte-Assistent

Als Beispiel wird das System *Aktiver Spurhalte-Assistent* der Daimler AG verwendet [Daimler-AG 2018].

Das System unterstützt den Fahrer beim Einhalten der Fahrspur. Hierzu wird beim ungewollten Verlassen der Fahrspur eingegriffen. Auf das vermutlich unbeabsichtigte Überfahren einer unterbrochenen Fahrbahnmarkierung wird der Fahrer durch ein pulsierendes Lenkrad aufmerksam gemacht. Ist die Nachbarspur zudem durch andere Verkehrsteilnehmer belegt oder wird eine durchgängige Fahrbahnmarkierung unbeabsichtigt überfahren, so wird die Querdynamik des Fahrzeugs durch einseitige Bremsengriffe korrigiert. In einem Geschwindigkeitsbereich von 60 bis $120 \frac{\text{km}}{\text{h}}$ ist der Spurhalte-Assistent aktiv. Das System lässt sich jederzeit übersteuern und wird automatisch inaktiv, sobald eine Fahreraktivität beim Spurwechsel erkannt wird. Als Umfeld-Sensoren werden eine Stereokamera sowie vier Radarsensoren verwendet. Hiervon befindet sich ein Radarsensor am Heck, ein Fern- sowie zwei Nahbereich-Radarsensoren sind an der Fahrzeugfront angebracht. Abbildung 2.4 visualisiert eine Prinzipskizze des beschriebenen Spurhalte-Assistenten.

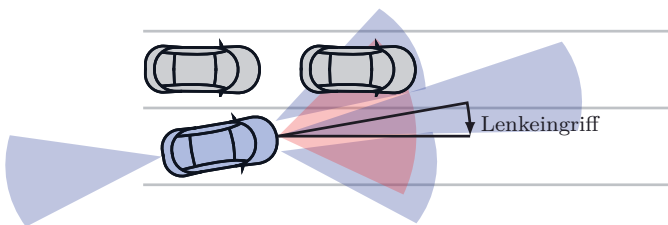


Abb. 2.4: Prinzipskizze eines Spurhalte-Assistenten

2.2. Umfeldsensorik

Wie an den exemplarisch aufgeführten Assistenzsystemen erkenntlich wird, werden unterschiedliche Umfeldsensoren verwendet, um das Verkehrsgeschehen für Algorithmen zugänglich zu machen. Neben intrinsischen Sensoren für die Umfelderkennung kommen auch extrinsische Systeme zum Einsatz. Hierzu zählen Positionierungssysteme wie GPS, oder die Kommunikation im Rahmen von Car-To-Car sowie Car-To-Infrastructure. Gemein haben diese extrinsischen Systeme, dass sie eventuell nicht dauerhaft verfügbar sind. So kann in Tunneln beispielsweise das GPS Signal verloren gehen oder eine Car-To-Infrastructure Kommunikation je nach Region nicht vorhanden sein. Aufgrund der unsicheren Verfügbarkeit können diese Systeme sicherheitskritische Anwendungen zwar unterstützen, jedoch nicht allein als primäre Informationsquelle dienen.

Die intrinsische Umfeldsensorik führt das Fahrzeug demgegenüber selbst mit. In Serienfahrzeugen werden momentan vorrangig Ultraschall-, Radar- und Videosensoren zur Erfassung der Fahrsituation eingesetzt. Zur dreidimensionalen Rekonstruktion der Umgebung werden zudem Lidar-Sensoren Anwendung in (teil-)autonomen Fahrzeugen finden. Diese sind momentan aufgrund hoher Kosten noch wenig verbreitet. [Vgl. Köllner 2017]

Abbildung 2.5 visualisiert Teile des elektromagnetischen und des Schalldruck-Spektrums. Zur Kategorisierung sind die vier Sensorsysteme in diese Spektren eingeordnet. Videosensoren arbeiten hauptsächlich im sichtbaren Lichtspektrum. Wellenlängen von Lidar-Sensoren für den automobilen Einsatz arbeiten im Wellenlängenbereich von 850 nm bis etwa 1 μm [Gotzig 2015, S. 328]. Ultraschall-Sensoren verwenden Frequenzen oberhalb des hörbaren Spektrums, beispielsweise 50 kHz [Noll 2015, S. 252]. Automobile Radarsensoren verwenden momentan vorrangig die Frequenzen 76,5 GHz und 24 GHz, was Wellenlängen

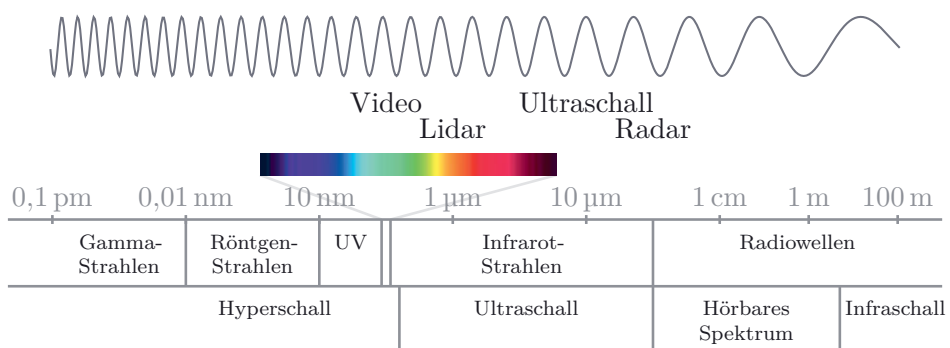


Abb. 2.5: Frequenzspektren unterschiedlicher Sensoren nach [Lagandré 2017]

von rund 4 mm und 12 mm entspricht [Winner 2015b, S. 260]. Die folgenden Abschnitte liefern einen Überblick über die Wirkprinzipien und den Einsatz der vier Sensorsysteme.

a. Ultraschallsensorik

Ultraschallsensoren für den Einsatz in Fahrzeugen nutzen die Laufzeitmessung eines Luftdruckimpulses zur Abstandbestimmung von Hindernissen. Hierbei kommt der Piezoelektrische Effekt zum Einsatz: Das Anlegen einer elektrischen Spannung führt zur Deformation eines Kristalls. Dieser Effekt wird im Fall der Ultraschallsensorik zur Erzeugung des Druckimpulses verwendet. Gleichzeitig führt eine von außen auf den Kristall wirkende mechanische Deformation zu einer proportional messbaren Spannung. Dieser direkte piezoelektrische Effekt wird zur Detektion der Impulsantwort verwendet, so dass ein Ultraschallwandler gleichzeitig als Emitter und Empfänger fungiert. [Noll 2015]

Der Öffnungswinkel von Ultraschallsensoren bewegt sich horizontal im Bereich von etwa 120° bis 140° , vertikal etwa 60° bis 70° und die Abstandmessung ist in einem Bereich zwischen 15 cm bis 6 m möglich [Vgl. Noll 2015; Robert Bosch GmbH 2018]. Der reduzierte vertikale Sichtbereich resultiert aus der Notwendigkeit unerwünschte Reflexe des Straßenbodens zu reduzieren. Neben Straßenreflexionen stellen akustische Fremdstrahler, wie metallische Reibgeräusche von Schienenfahrzeugen und Schmutzschichten auf der Sensormembran die hauptsächlichsten Störquellen dar.

Die Ultraschallsensoren werden vorrangig für Einparksysteme eingesetzt. Um trotz des großen Öffnungswinkels eines Ultraschallsensors Umgebungsobjekte örtlich zuzuordnen, werden mehrere Sensoren in einer horizontalen Ebene im Fahrzeug verbaut und sich des Trilaterationsprinzips zur Objektdetektion bedient. Aufgrund von Überlappungen der Sensorerfassungsbereiche kann ein gesendeter Impuls von mehreren Ultraschallwandlern detektiert werden. Wegen des bekannten Abstands der Sensoren zueinander kann anhand der Direkt- und Kreuzechos der Abstand zu einem Punkthindernis bestimmt werden. Zusätzlich kann die Differenz zwischen Direkt- und Kreuzecho auch zur Unterscheidung von Punkt- und Wand-förmigen Hindernissen herangezogen werden. Abbildung 2.6 veranschaulicht die Trilateration an einem Punkthindernis [Vgl. Noll 2015].

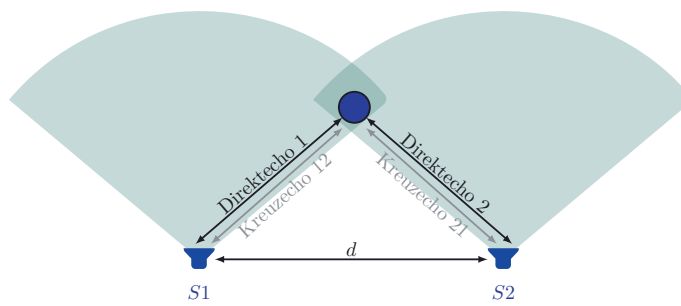


Abb. 2.6: Trilaterationsprinzip an einem Punkthindernis nach [Noll 2015]

b. Radarsensorik

Ähnlich wie Ultraschallsensoren basiert die Radarsensorik auf der Laufzeitmessung eines ausgesandten Signals. Für automobiler Anwendungen werden derzeit Radarsysteme mit elektromagnetischen Signalen, vorrangig im Frequenzbereich 76,5 GHz und 24 GHz verwendet [Winner 2015b, S.260]. Die Puls-Modulation von Radarsignalen wäre zur Distanzermittlung grundsätzlich möglich. Da hierbei die Bestimmung der genauen Laufzeit aufwändig und das Verhältnis von Spitzenleistung zur mittleren Sendeleistung ungünstig ist, werden stattdessen frequenzmodulierte Dauerstrichradare (engl. Frequency Modulated Continuous Wave, FMCW) eingesetzt [Vgl. Weihard 2010]. Abbildung 2.7 zeigt relevante Größen exemplarischer Sende- und Empfangsrampen. Betrachtet man die

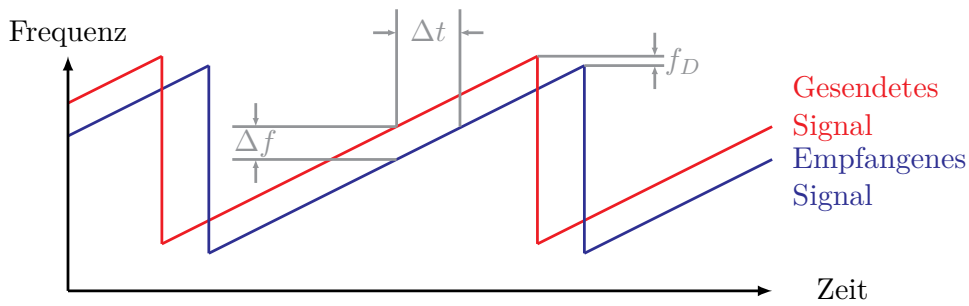


Abb. 2.7: Sende- und Empfangssignal eines frequenzmodulierten Dauerstrichradars nach [Weihard 2010]

dadurch erhaltene Impulsantwort im Frequenzspektrum, so setzen sich die zu einem Objekt gehörigen Maxima aus dem Einfluss der Objektdistanz und Relativgeschwindigkeit zusammen. Ändert sich die Momentanfrequenz rampenförmig zu

$$\omega(t) = \omega_0 + m_\omega(t - t_0),$$

so lässt sich durch die Verwendung zweier Frequenzrampen mit unterschiedlichen Steigungen $m_i = \frac{\delta f_i}{\delta t}$ ein lineares Gleichungssystem aufstellen. Daraus ergeben sich nach [Winner 2015b] der radiale Abstand r und die radiale Relativgeschwindigkeit \dot{r} eines Zielobjektes zu

$$r = \frac{c_L}{2} \cdot \frac{\omega_1 - \omega_2}{m_1 - m_2}$$

$$\dot{r} = \frac{c_L}{2\omega_0} \cdot \frac{m_1\omega_2 - m_2\omega_1}{m_1 - m_2}.$$

c_L beschreibt dabei die Lichtgeschwindigkeit, mit der sich die elektromagnetischen Radarwellen fortbewegen. Bei einer Vielzahl an Objekten stellt die

korrekte Zuordnung der Frequenzmaxima eine große Herausforderung dar, die im Rahmen der Signalverarbeitung und des nachgelagerten Trackings adressiert wird. [Vgl. Yang 2006; Winner 2015b]

Zur räumlichen Lokalisierung eines Objekts wird zudem der Azimutwinkel zwischen Sensor und Hindernis bestimmt. Um mechanisch rotierende Elemente im Fahrzeug zu vermeiden, kommen zur Winkelbestimmung keine mechanisch schwenkbaren Systeme zum Einsatz. Stattdessen wird ein Antennen-Array verwendet, wobei eine Phasenverschiebung zur Strahlschwenkung und Anpassung der Antennencharakteristik verwendet wird. Abbildung 2.8 visualisiert Wellenfronten an einem Antennen-Array. Der Schwenkwinkel Φ ergibt sich

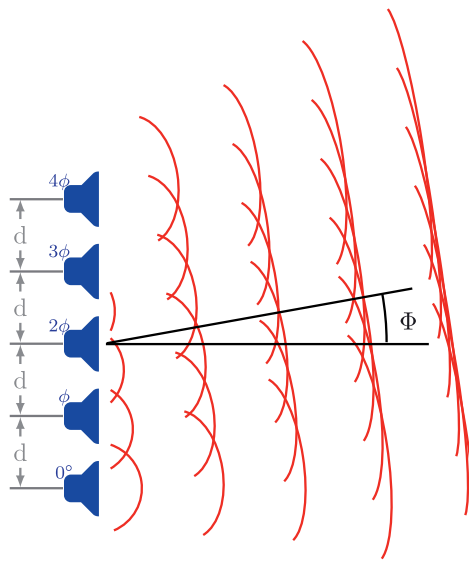


Abb. 2.8: Schematische Visualisierung von Wellenfronten an einem Antennen-Array nach [Olscher 2016]

nach [Weihard 2010] aus der Phasenverschiebung ϕ und dem Antennenabstand d zu

$$\phi = \frac{2\pi}{\lambda} d \cdot \sin(\Phi).$$

Das Radar-Reflexionsvermögen eines Zielobjekts wird durch dessen Radarquerschnitt σ (engl. Radar Cross Section, RCS) beschrieben. Relevante RCS-Werte für Ziele in automobilen Anwendungen, wie beispielsweise Fußgänger, Fahrzeuge oder Schachtdeckel, streuen sehr stark und liegen im Bereich von $\sigma = 0,01$ bis 10000 m^2 . In [ISO 15622] wird im Rahmen eines Funktionstests der

Radarquerschnitt für ein Fahrzeughindernis mit $10 \pm 3 \text{ m}^2$ angegeben. Neben den Materialeigenschaften des Zielobjekts hat die Objektgeometrie und dessen Orientierung großen Einfluss auf die reflektierte Signalintensität. [Winner 2015b]

c. Lidar-Sensorik

Die Abkürzung Lidar steht für Light Detection and Ranging und ist ein Messverfahren, das auf einer elektromagnetischen Impulslaufzeitmessung basiert und dessen Wellenlänge (850 nm bis $1 \mu\text{m}$) sich nahe dem sichtbaren Lichtspektrum befindet. Ein gebündelter Lichtimpuls wird emittiert und an einem Umgebungsobjekt reflektiert. Die Impulsantwort kann schließlich vom Sensor detektiert werden und aus der Signallaufzeit lässt sich auf den Abstand zwischen Sensor und Objekt schließen. Die Distanz d zum Objekt ergibt sich anhand der Lichtgeschwindigkeit c_0 und der Impulslaufzeit t zu

$$d = \frac{c_0 \cdot t}{2}.$$

Die Impulsantwort eines einzelnen, stehenden Zielobjekts ähnelt im Idealfall einer Gaußkurve. Durch entsprechende Signalverarbeitung gelingt es mehrere Objekte innerhalb eines Messkanals zu detektieren (Mehrzielfähigkeit). Störende Umwelteinflüsse, wie beispielsweise Nebel, reflektieren ebenfalls den vom Emitter ausgesandten Impuls. Die Impulsantwort eines solchen *weichen* Objekts unterscheidet sich jedoch von einem harten Reflektor. Das zurückgestrahlte Intensitätsmaximum fällt für weiche Objekte geringer aus und besitzt gleichzeitig eine ausgedehntere Impulsantwort.

Abbildung 2.9 visualisiert die Impulsantworten dreier Objekte. Hierbei han-

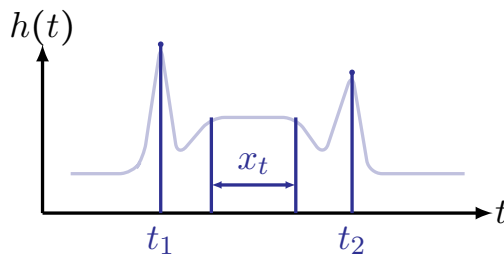


Abb. 2.9: Idealisierte Lidar Impulsantwort mit drei Objekten im Messkanal

delt es sich mit zunehmendem Abstand vom Emitter um ein hartes, ein weiches und ein hartes Objekt. Die Maxima der Impulsantworten bei t_1 und t_2 lassen sich den harten Objekten zuordnen. Nebel beispielsweise, der zwischen den

beiden Objekten liegt, würde zu der im Bereich x_t dargestellten Impulsantwort führen. [Vgl. Gotzig 2015]

Für die technische Umsetzung von Lidar Sensoren im Automobilbereich gibt es mehrere Varianten hinsichtlich der Strahlsensorik. Abbildung 2.10 visualisiert vier unterschiedliche Strahlsensoriken. Diese unterscheiden sich durch die



Abb. 2.10: Unterschiedliche Strahlsensoriken nach [Gotzig 2015]

Anzahl und Öffnungswinkel der einzelnen Strahlen sowie durch die mechanische Beweglichkeit der Sensoren. So werden beispielsweise rotierende Laserscanner von [Velodyne Lidar] für Entwicklungsfahrzeuge oder als Referenzsensorik verwendet. Das 360° horizontale und 40° vertikale Sichtfeld, eine Reichweite von 120 m und eine Azimut-Winkelauflösung von $0,08^\circ$ erlauben eine sehr hochauflösende Datenakquise des Fahrzeugumfelds. Aufgrund mechanisch rotierender Teile sowie der hohen Kosten dieser Sensorik werden aktuell robustere und kostengünstigere Sensoren für den Einsatz in Serienfahrzeugen entwickelt [Cameron 2017].

Bei einem rotierenden Single Beam Lidar Scanner werden die Maxima der Impulsantwort identifiziert, um daraus die Distanz zu ermitteln. Aufgrund der hohen Strahlbündelung lässt sich die Distanz mittels der aktuellen Sensorposition einem Punkt im Raum zuordnen.

d. Video-Sensorik

In der Fahrzeugtechnik werden Videokameras für unterschiedliche Zwecke eingesetzt: Eine Rückfahrkamera liefert dem Fahrer ein Echtzeitvideo zur Kontrolle des Bereichs hinter dem Fahrzeug. Kameras zur Fahrer- und Innenraumüberwachung werden dazu eingesetzt, um beispielsweise den Müdigkeitszustand des Fahrers zu ermitteln oder eine Handgestensteuerung des Infotainmentsystems zu ermöglichen. Neben diesen Komfortfunktionen werden Videosensoren auch zur Umfelderkennung in (teil-)autonomen Systemen verwendet, wobei die Auswertung der Daten anhand bildverarbeitender Algorithmen erfolgt. So werden Verkehrszeichen, Autos oder Fußgänger anhand der Bildinformation automatisch klassifiziert, um anschließend softwaregesteuert darauf zu reagieren. Mit Hilfe eines Fernlichtassistenten kann so das Fernlicht unter Einbeziehung der Anwesenheit anderer Verkehrsteilnehmer eingestellt werden. Eine Verkehrszeichenerkennung ermöglicht die Warnung des Fahrers bei Überschreitung der

zulässigen Geschwindigkeit und eine Fahrstreifenerkennung ist Voraussetzung zur Umsetzung eines automatischen Spurhalteassistenten. [Vgl. Stiller 2015; Schiele 2015]

Diese Aufgaben stellen mehrere Anforderungen an die Kamerahardware: Zur robusten Detektion von Verkehrszeichen ist eine hohe Auflösung von mehr als 15 Pixel/° sowie eine kurze Belichtungszeit nötig, damit die Bewegungunschärfe bei hohen Geschwindigkeiten gering bleibt. Kurze Belichtungszeiten erfordern eine hohe Lichtempfindlichkeit des Bildsensors. Um weiße von gelben Fahrbahnmarkierungen im Baustellenbereich unterscheiden zu können, muss zusätzlich die Farbinformation erfasst und ausgewertet werden. Bei tiefstehender Sonne oder Tunnel Ein- und Ausfahrten muss ein großer Helligkeitsbereich erfasst werden, um Details sowohl in hellen als auch in dunklen Bildbereichen darstellen zu können. Hierfür ist ein hoher Dynamikumfang des Bildsensors nötig. Welcher Bereich um das Fahrzeug je Kamera erfasst werden kann, hängt vom horizontalen und vertikalen Blickfeld der Kamera ab, die durch den Sensor sowie die verwendete Optik definiert werden. [Punke 2015]

Abbildung 2.11 veranschaulicht anhand mehrerer Bildreihen den Einfluss des Blickfelds, des Dynamikumfangs und der Auflösung auf die resultierenden Bilddaten. Die für das Ausgangsbild verwendete Kamera nutzt einen *Four Thirds Sensor* (17,3 mm × 13 mm), eine Brennweite von 42 mm, eine Blende von F/6,3, eine Auflösung von 4592 × 2919 Pixeln und eine Lichtempfindlichkeit von ISO-160. Hieraus resultiert ein horizontales Blickfeld von rund 23° und ein vertikales Blickfeld von rund 18°. Die Winkelauflösung beträgt somit 197 Pixel/°.

Die dreidimensionale Rekonstruktion der Fahrzeugumgebung mittels einzelner Videosensoren stößt wegen der zweidimensionalen Information, die ein Bild zur Verfügung stellt, schnell an die Grenzen. Deshalb werden häufig Stereokamerasysteme genutzt, wobei anhand des definierten Abstands der Sensoren zueinander und eines nachgelagerten Bildvergleichs zusätzliche Tiefeninformation extrahiert wird.

Aufgrund der Ähnlichkeit von Kameras mit der menschlichen Wahrnehmung sind die Störfaktoren einfach nachzuvollziehen: Hierzu gehören vorrangig die Verschmutzung der Optik, weshalb der Einbauort von Front-View-Kameras häufig im Fahrzeuginnenraum liegt. Zusätzlich ist die Adaption der Sensorlichtempfindlichkeit bei schnell wechselnden Lichtbedingungen nötig. Einflussfaktoren wie Linsenverzerrung und Ausrichtung der Kamera werden im Rahmen von Kalibrierungen adressiert und können teilweise in der nachgelagerten Signalverarbeitung korrigiert werden. [Punke 2015]

2.3. Test und Freigabe von Fahrerassistenzsystemen



Abb. 2.11: Einfluss unterschiedlicher Kameraparameter, basierend auf [Xan Griffin 2018]

2.3. Test und Freigabe von Fahrerassistenzsystemen

Wie an den Beispielen von Assistenzsystemen zu erkennen ist, sind unterschiedliche Produkte bereits in Serienfahrzeugen verfügbar. Zur Zulassung dieser Systeme existieren mehrere Normen. Diese Dokumente definieren unterschiedliche Begrifflichkeiten, die zur Beschreibung von Testabläufen nötig sind. Ein Beispiel hierfür ist der Begriff *stationary object*, der in [ISO 15622] ein stehendes Objekt vor dem Testfahrzeug beschreibt. [Schnarr 2016] liefert einen umfassenden Überblick von Parametern vorhandener Testfallbeschreibungen. Tabelle 2.2 listet fünf ISO-Normen auf, die sich mit dem Test von Assistenzsystemen befassen.

Norm	Beschreibung
[ISO 15622] Intelligent transport systems — Adaptive Cruise Control systems — Performance requirements and test procedures	Diese Norm betrachtet Abstandsregelautomaten (engl. Adaptive Cruise Control, ACC), wobei sich deren Einsatzort auf Autobahnen mit fließendem Verkehr beschränkt. Hierbei werden funktionale Anforderungen an das System, dessen Interaktion mit dem Benutzerinterface sowie Tests zur Sicherstellung der grundlegenden Abstandshaltfunktion definiert. Die Tests beschreiben Situationen mit einzelnen und mehreren Zielfahrzeugen auf geraden und kurvigen Straßen. Die Umgebungsbedingungen definieren trockene Straßenverhältnisse, einen Temperaturbereich von -20°C bis 40°C und eine Sichtweite von mehr als einem Kilometer.
[ISO 15623] Intelligent transport systems — Forward vehicle collision warning systems — Performance requirements and test procedures	Diese Norm betrachtet Systeme, die den Fahrer vor einer nahenden Kollision mit einem vorausfahrenden Fahrzeug warnen (engl. Forward Vehicle Collision Warning System, FVCWS). Die Systeme greifen nicht aktiv in die Fahrzeugführung ein. Auch dieses Dokument definiert mehrere Begrifflichkeiten zur Beschreibung von Anforderungen und Testfällen. Zudem werden Tests zur Überprüfung der grundlegenden Funktionalität definiert. Hierzu gehört das frontale Auffahren auf ein Hindernis sowie die Unterscheidung mehrerer Zielfahrzeuge auf geraden und kurvigen Straßen. Die Umgebungsbedingungen definieren trockene Straßenverhältnisse, einen Temperaturbereich von -20°C bis 40°C und eine Sichtweite von mehr als einem Kilometer.

Norm	Beschreibung
[ISO 17361] Intelligent transport systems — Lane departure warning systems — Performance requirements and test procedures	Diese Norm betrachtet Systeme, die den Fahrer beim vermutlich ungewollten Verlassen der Spur warnen. Die Systeme greifen nicht aktiv in die Fahrzeugführung ein. Im Rahmen der aufgeführten Tests werden Warnungen provoziert, indem das Fahrzeug langsam von geraden sowie gekrümmten Fahrbahnen abkommt. Die Umgebungsbedingungen definieren trockene Straßenverhältnisse, einen Temperaturbereich von -20°C bis 40°C und eine Sichtweite von mehr als einem Kilometer.
[ISO 17387] Intelligent transport systems — Lane change decision aid systems (LCDAS) — Performance requirements and test procedures	Diese Norm betrachtet Systeme, die den Fahrer Hinweise geben, ob ein Spurwechsel momentan möglich ist. Dies erfolgt meist durch ein Hinweislicht an den Außenspiegeln. Die Systeme greifen nicht aktiv in die Fahrzeugführung ein. Die Tests beschreiben mehrere Verkehrssituationen, bei denen sich unterschiedliche Verkehrsfahrzeuge in oder neben der Spur des Testfahrzeugs bewegen. Hierbei sind jeweils diejenigen Bereiche definiert, in denen ein Spurwechsel als gefährlich zu klassifizieren ist. Die Umgebungsbedingungen definieren trockene Straßenverhältnisse, einen Temperaturbereich von -20°C bis 40°C und eine Sichtweite von mehr als einem Kilometer.
[ISO 22839] Intelligent transport systems — Forward vehicle collision mitigation systems — Operation, performance, and verification requirements	Im Vergleich zur [ISO 15623] betrachtet diese Norm Systeme, die aktiv in die Längsdynamik des Fahrzeugs eingreifen um einen drohenden Unfall zu vermeiden oder die Aufprallenergie zu reduzieren. Die Tests überprüfen den Erfassungsbereich des Systems, die Funktionalität für frontale Hindernisse, die Unterscheidung mehrerer Zielfahrzeuge in kurvigen Straßen sowie Objekte oberhalb des Fahrzeugs, wie etwa Brücken. Die Umgebungsbedingungen definieren trockene Straßenverhältnisse mit hohen Reibwerten, einen Temperaturbereich von -20°C bis 40°C und eine Sichtweite von mehr als einem Kilometer.

Tabelle 2.2: Auflistung von Normen mit Testbeschreibungen für Assistenzsysteme, basierend auf [Schnarr 2016]

Nachfolgend wird ein Test der [ISO 22839] detaillierter beschrieben. Der Zielunterscheidungstest überprüft, ob das System in der Lage ist auf einer gekrümmten Straße die Trajektorien der Fahrzeuge richtig zu prognostizieren. Hierfür fährt das Testfahrzeug, welches mit einem Notbremsassistenten ausgestattet ist, auf einer kreisrunden Strecke. In der selben Spur fährt ein Zielfahrzeug, welches einige Sekunden nach Testbeginn bremst. In einer benachbarten Spur fährt ein weiteres Verkehrsfahrzeug, dass keine Gefahr für das Testfahrzeug darstellt. Während der Durchführung wird untersucht, ob der Notbremsassistent richtigerweise das Zielfahrzeug als Gefahr erkennt. Abbildung 2.12 visualisiert den beschriebenen Testaufbau.

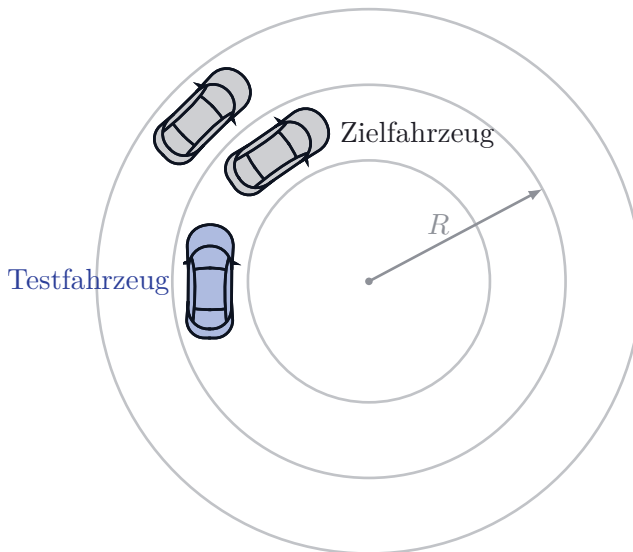


Abb. 2.12: Runde Teststrecke des Zielunterscheidungstests nach [ISO 22839]

Zusätzlich zu den Normen gibt es im Rahmen des EURO NCAP oder vom Allgemeinen Deutschen Automobil-Club (ADAC) Verbrauchertests, deren Durchführung sich stark an den Testfallbeschreibungen der Normen orientiert. Deutlich wird, dass die beschriebenen Testfälle die Funktionalität der Systeme an einzelnen Arbeitspunkten bei idealen Umgebungsbedingungen überprüfen.

3. Absicherungsstrategie

Fahrerassistenzsysteme bieten die Möglichkeit, Teile der Fahraufgabe zu automatisieren. Nach [Donges 1982] kann die Fahraufgabe in die drei hierarchischen Bereiche Navigation, Bahnführung und Stabilisierung gegliedert werden. Die Ausführung der Fahraufgabe unterteilt sich in die vier Stufen Wahrnehmung der Umgebung (Perzeption), Einschätzung der aktuellen Situation (Kognition), Verhaltenssteuerung sowie anschließender Trajektorienregelung.

Assistenzsysteme müssen diese vier Stufen teilweise selbständig übernehmen. Diese Aufgaben algorithmisch zu lösen ist eine Herausforderung der aktuellen Entwicklung autonomer Fahrzeuge. Nach [Segner 2015] ist für die Datenanalyse der Umfeldsensorik „noch Grundlagenforschung gefordert“, um beispielsweise mittels künstlicher Intelligenz und lernfähiger Systeme eine Kognition für selbst-fahrende Automobile zu ermöglichen. Sobald diese Systeme auf öffentlichen Straßen eingesetzt werden, stellen sie ein potentielles Risiko dar. Technische Fehler, wie eine fälschliche Kognition, sind unbedingt zu vermeiden, da hierdurch Menschenleben gefährdet werden. Deshalb ist eine Absicherungsstrategie für Assistenzsysteme notwendig, um das Auftreten von Fehlern soweit wie möglich zu reduzieren. Ein transparentes und nachvollziehbares Vorgehen für das Testen von Assistenzsystemen während des Entwicklungsprozesses wird hierfür im folgenden Kapitel entwickelt. Teile dieser Methodik wurden in [Feilhauer 2016a] und [Feilhauer 2017b] publiziert.

3.1. Funktionale Unzulänglichkeit

Die Absicherung automatisierter Fahrfunktionen unterscheidet sich von bisherigen Testansätzen in der Automobilindustrie. Zusätzlich zur Absicherung des Systemversagens auf funktionaler Ebene, die umfassend in [ISO 26262] beschrieben wird, kommt bei Fahrerassistenzsystemen durch die Umfeldwahrnehmung die Möglichkeit einer fehlerbehafteten Kognition hinzu. Eine komplexe Verkehrssituation beispielsweise, die in der Entwicklung nicht berücksichtigt und deren entsprechende Kognition nicht spezifiziert wurde, könnte vom System *falsch*¹¹ eingeschätzt werden und so zu einem Unfall führen [Vgl. Weitzel 2014]. Die Problematik der grundsätzlich unvollständigen Spezifikation stellt den zentralen Aspekt dar, weshalb bisherige Absicherungsansätze, bei denen man annimmt den Problemraum vollständig erfasst zu haben, nicht ausreichend sind.

¹¹Der Begriff *falsch* impliziert, dass es auch eine *richtige* Bewertung der Verkehrssituation gibt. Diese wurde aber in diesem Fall vorab nicht spezifiziert.

Dieser Umstand wird auch als *funktionale Unzulänglichkeit* beschrieben [Ebel 2009]. Die Tatsache, dass nicht alle möglichen Verkehrssituationen spezifiziert werden können, ist in einer Absicherungsstrategie zu berücksichtigen.

Die Herausforderungen im Umgang mit unvollständigen Spezifikationen sind auch im Rahmen der allgemeinen Absicherung von Software bekannt. Der Begriff *Robustheit* beschreibt die „Eigenschaft einer Betrachtungseinheit [...] auch in ungewöhnlichen Situationen definiert zu arbeiten und sinnvolle Reaktionen durchzuführen“ [Liggesmeyer 2002, S. 10]. Wird *Vollständigkeit* als die Realisierung aller Funktionen der Spezifikation verstanden, so kann auch ein vollständiges System bei nicht erfassten Spezifikationen ausfallen. Aufgrund der Komplexität moderner Software, beziehungsweise der unüberschaubaren Kombinatorik möglicher Umgebungsbedingungen bei Fahrerassistenzsystemen, lässt sich „aber im Allgemeinen nicht jede[r] denkbare[] Fall beim Erstellen einer Spezifikation berücksichtigen[, weshalb] Robustheit einen graduellen Charakter“ besitzt [Liggesmeyer 2002, S. 10]. Dies entspricht grundsätzlich dem Umstand funktionaler Unzulänglichkeit. Da nicht alle Möglichkeiten vorab spezifiziert werden können, muss sowohl die Spezifikation als auch die Absicherung durch neue Erkenntnisse, die während der Testphase oder des Betriebs entstehen, erweiterbar sein.

Einen möglichen Absicherungsansatz stellen **formale Methoden** dar. Zur Generierung einer Aussage, dass ein Fahrerassistenzsystem vollständig fehlerfrei sei, wäre ein Beweis wünschenswert. In [Mitsch 2013] wird die Anwendung formaler Methoden zur Absicherung der kollisionsfreien Trajektoriengenerierung eines bewegten Roboters präsentiert. Zusammenfassend wird beschrieben: „We [...] prove that collisions can never occur (as long as the robot system fits to the model)“ [Mitsch 2013, S. 2]. Dies zeigt die Notwendigkeit einer validierten Modellbildung für die Anwendung formaler Methoden. Für einzelne Problembereiche, wie der Trajektoriengenerierung, ist die Erstellung eines validen, mathematischen Modells eventuell möglich. Deshalb ist die Anwendung formaler Methoden für Teile der Assistenzsysteme auch sinnvoll. Im Bereich der Sensorik stellt die Formalisierung eine besondere, bisher ungelöste Herausforderung dar [Vgl. Wachenfeld 2015, S. 461]. Im Rahmen dieser Arbeit werden formale Methoden deshalb nicht als Absicherungskonzept für das Gesamtsystem fokussiert, sondern als unterstützender Ansatz für einzelne, klar abgegrenzte Bereiche betrachtet.

Einen alternativen Absicherungsansatz stellt ein **statistischer Nachweis** über die Einhaltung einer Obergrenze hinsichtlich des Systemversagens dar. Eine solche Vorgehensweise wird auch zur Absicherung von mechanischen

Bauelementen verwendet: Hier besteht die Problematik, dass beispielsweise Alterungserscheinungen nicht ausreichend genau beschrieben werden können oder die Umgebungseinflüsse während des Betriebs zum Entwicklungszeitpunkt unbekannt sind. Deshalb kann Bauteilversagen während des Betriebs meist nicht vollständig ausgeschlossen werden. Hier werden statistische Ansätze verwendet, mit denen unter repräsentativen Umgebungsbedingungen eine Versagenswahrscheinlichkeit für einzelne Bauteile ermittelt wird. Für komplexere Baugruppen wird anschließend eine Gesamtversagenswahrscheinlichkeit errechnet. Dieses Vorgehen ist für elektronische Bauteile weit verbreitet. Hier wird für einzelne Komponenten die Wahrscheinlichkeit des Ausfalls innerhalb eines bestimmten Zeitintervalls angegeben ($\lambda_{\text{Komponente}} = \frac{\text{Ausfälle}}{\text{Zeit}}$). Bei redundanzfreien Baugruppen ergibt sich die Gesamtausfallrate aus $\lambda_{\text{Baugruppe}} = \sum (\lambda_{\text{Komponente}})$ [Vgl. DIN EN 61709, S. 13].

Wählt man diesen statistischen Ansatz für Funktionen eines Fahrerassistenzsystems, so besteht eine Herausforderung in der Festlegung repräsentativer Umgebungsbedingungen: Der Parameterraum der Einflussgrößen beinhaltet bei Fahrerassistenzsystemen unterschiedliche Fahrprofile, Wetterbedingungen, Verhaltenseigenschaften von Verkehrsteilnehmern oder Materialeigenschaften umgebender Objekte. Die große Vielzahl unterschiedlicher Parameter erschwert eine Aussage über die Repräsentativität von Testbedingungen erheblich [Vgl. Wilhelm 2015, S.100 ff.]. Zusätzlich ist die erforderliche Fahrstrecke bei einem statistisch motivierten Absicherungsansatz sehr hoch, was folgende Überlegungen zeigen.

Die Leistungsfähigkeit eines Systems kann anhand einer Metrik M bewertet werden, die unterschiedliche Gütemaße ψ_i einbezieht: $M = f(\psi_1, \dots, \psi_n)$. Wie eine solche Metrik im Fall von sicherheitskritischen Assistenzsystemen auszusehen hat, ist eine gesellschaftliche Diskussion, die nicht im Fokus dieser Arbeit steht. Hier seien zwei Möglichkeiten aufgeführt: Für einen Notbremsassistenten kann eine solche Metrik lauten, dass eine Notbremsung nie fälschlicherweise ausgelöst werden darf. Die Anzahl der False-Positive Auslösungen muss somit für alle Verkehrssituationen gleich null sein.

Alternativ kann die Metrik lauten, dass die Anzahl der tödlichen Unfälle mit einem Notbremsassistenten n_{mit} kleiner sein muss als die Anzahl der tödlichen Unfälle ohne Notbremsassistenten n_{ohne} . Somit ergibt sich die Metrik zu

$$M(\psi_{\text{Unfälle}}) := \frac{n_{\text{mit}}}{n_{\text{ohne}}} < 1.$$

Eine Absicherungsstrategie muss einen belastbaren Nachweis über die Erfüllung einer solchen Metrik liefern.

Wie in [Winner 2015a] dargestellt, würde ein statistisch motivierter Nachweis zur Reduktion von Unfällen durch automatisierte Fahrfunktionen einen enormen Fahraufwand erfordern. Basierend auf k Ereignissen, wie beispielsweise *Unfälle mit Personenschaden*, wird ein autonomes Fahrzeug mit einem Referenzsystem¹² verglichen. Ziel ist die Länge der Teststrecke zu ermitteln, die bei einer Irrtumswahrscheinlichkeit ϵ ausreicht, um nachzuweisen, dass ein autonomes Fahrzeug weniger negative Ereignisse k als ein Referenzsystem verursacht. Grundlage der Überlegung stellt die Poisson-Verteilung

$$P_\lambda(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

dar. Diese Wahrscheinlichkeitsverteilung repräsentiert einen Grenzfall der Binomialverteilung. Sie beschreibt seltene, voneinander unabhängig auftretende Ereignisse (univariate, diskrete Wahrscheinlichkeitsverteilung, [Vgl. Christoph 2002, S 59.ff]). Mit Hilfe der Poisson-Verteilung lässt sich die Wahrscheinlichkeit ermitteln, mit der k Ereignisse bei einem Erwartungswert λ auftreten. Abbildung 3.1 zeigt die Wahrscheinlichkeitsverteilung sowie die akkumulierte Auftretshäufigkeit unterschiedlicher Ereigniszahlen für einen Erwartungswert $\lambda = 3$.

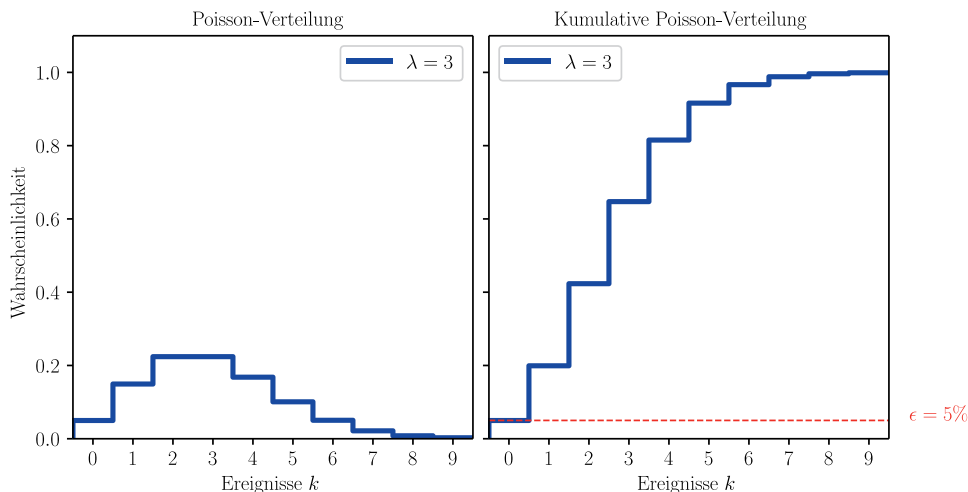


Abb. 3.1: Poissonverteilungen für $\lambda = 3$

Zusätzlich ist horizontal die Wahrscheinlichkeit von $\epsilon = 5\%$ markiert. Wird eine Irrtumswahrscheinlichkeit von 5% akzeptiert, so stellt dies die obere

¹²Als Referenzsystem wird hier ein System ohne das Assistenzsystem bezeichnet, das als Vergleich (Referenz) dient

Wahrscheinlichkeitsgrenze dar, mit der die akkumulierten Ereignisse auftreten dürfen. Wird auf einer Referenzstrecke bei einem Erwartungswert von $\lambda = 3$ Unfällen kein Unfall durch das autonome Fahrzeug verursacht, so kann man mit 95% Sicherheit sagen, dass das autonome System besser ist als das Referenzsystem. Hier stellt sich die Frage nach der Wahl der Referenzstrecke.

[Winner 2015a] führt eine Beispielrechnung zur statistischen Absicherung eines Autobahnautomaten an: Als Referenzstrecke wird die mittlere Strecke zwischen zwei Unfällen mit Personenschäden auf Autobahnen verwendet $s_{\text{ref}} = 12 \cdot 10^6$ km. Betrachtet man nur den Anteil der Unfälle, bei denen der Fahrzeugführer Hauptverursacher ist (60%), sowie einen Verbesserungswert¹³ moderner Fahrzeuge von 1,2, so erhöht sich die Referenzstrecke auf $12 \cdot 10^6 \text{ km} \cdot 1,2/0,6 = 24 \cdot 10^6$ km. Für einen Erwartungswert von $\lambda = 3$ Unfällen ergibt sich somit die Strecke $s = 3 \cdot 24 \cdot 10^6 \text{ km} = 72 \cdot 10^6$ km. Würde man die Erfüllung der Metrik $M(\psi_{\text{Unfälle}})$ basierend auf diesem statistischen Nachweis mit einer durchschnittlichen Fahrtgeschwindigkeit von $100 \frac{\text{km}}{\text{h}}$ absichern, so ergibt sich eine Fahrzeit auf Autobahnen von rund 82 Jahren. Diese Beispielrechnung veranschaulicht die Größenordnung, würde man die Absicherung statistisch unter bisherigen Black-Box-Freigabeansätzen betrachten [Vgl. Winner 2015a, S.1174 ff.]. Die Testfahrt müsste zudem bei jeder Funktionsänderung erneut durchgeführt werden, was gerade bei komplexeren autonomen Systemen wirtschaftlich unmöglich ist. Außerdem ist ein Nachweis bezüglich der Repräsentativität der gefahrenen Vergleichsstrecke notwendig: Würden die 72 Millionen Kilometer ausschließlich auf einem neuen, sehr breiten Autobahnabschnitt gefahren werden, wäre hier die Unfallrate vermutlich ohnehin geringer.

Das Vorgehen zeigt, wie man ohne Systemwissen eine statistisch belegte Aussage bezüglich der Erfüllung einer Metrik unter Randbedingungen erhält. Im Beispiel des Autobahnautomaten ergibt sich, dass das teilautonome Fahrzeug mit 95% Wahrscheinlichkeit weniger Unfälle als ein Vergleichsfahrzeug verursacht. Es ergibt sich somit die Wahrscheinlichkeit P , mit der das System die Metrik M erfüllt.

$$P(M)$$

Auf Grund der im Beispiel verwendeten Vergleichsgruppen (Einsatzbereich V_{Einsatz} , Unfallverursacher $V_{\text{Verursacher}}$, Verbesserungsfaktor $V_{\text{Verbesserung}}$) handelt es sich bei der Aussage hinsichtlich der Metrikerfüllung um die bedingte Wahr-

¹³Der Verbesserungswert ist ein Schätzwert der angibt, wie viel größer die Referenzstrecke (der Abstand zwischen zwei Unfällen mit Personenschaden) durch moderne Fahrzeuge auch ohne das Fahrerassistenzsystem gewesen wäre. Dieses Notwendigkeit beruht auf der Tatsache, dass angenommen wird die Daten der Referenzstrecke seien zu einem früheren Zeitpunkt ermittelt worden.

Szenario	Einsatzbereich	Unfallverursacher	Verbesserungs- faktor
S_i	V_{Einsatz}	$V_{\text{Verursacher}}$	$V_{\text{Verbesserung}}$
1	Autobahn	Hauptverursacher	$\leq 1,2$
2	Autobahn	Hauptverursacher	$> 1,2$
3	Autobahn	Nicht Hauptverursacher	$\leq 1,2$
4	Autobahn	Nicht Hauptverursacher	$> 1,2$
5	Nicht Autobahn	Hauptverursacher	$\leq 1,2$
6	Nicht Autobahn	Hauptverursacher	$> 1,2$
7	Nicht Autobahn	Nicht Hauptverursacher	$\leq 1,2$
8	Nicht Autobahn	Nicht Hauptverursacher	$> 1,2$

Tabelle 3.1: Mögliche Kombinationen der drei Vergleichsgruppen

scheinlichkeit

$$P(M \mid \text{Autobahn, Hauptverursacher, } V_{\text{Verbesserung}} = 1,2).$$

Klassifiziert man den vollständigen Zustandsraum der drei eingeführten Vergleichsgruppen vereinfacht in je zwei Teile, so ergeben sich die in Tabelle 3.1 aufgelisteten Kombinationen. Diese ganzheitliche Betrachtung des eingeführten Parameterraums liefert einen Überblick über die abgesicherten Teile des Systems. Dies ist für eine belastbare und transparente Absicherung des Systems essentiell. Als ein Szenario wird hier die Kombination der disjunkten¹⁴ Vergleichsgruppen verstanden. Somit listet Tabelle 3.1 acht unterschiedliche Szenarien auf. Für die Gesamtwahrscheinlichkeit zur Erfüllung der Metrik ergibt sich für die Szenarien S_i :

$$P_{\text{ges}}(M) = \sum_i P(M \mid S_i) \cdot P(S_i) \tag{3.1}$$

¹⁴Zustände, die nicht gleichzeitig eintreten können beziehungsweise sich gegenseitig ausschließen.

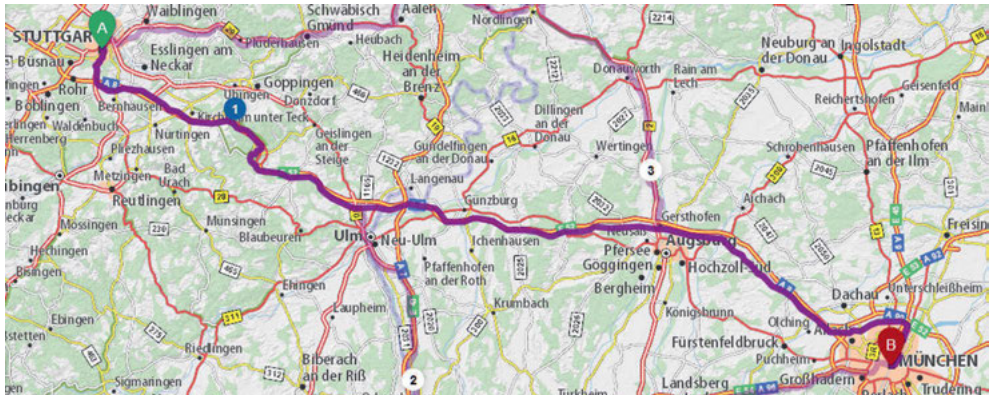


Abb. 3.2: Routenplanung von Stuttgart nach München [Michelin 2017]

$P(M | S_i)$ steht hierbei für die bedingte Wahrscheinlichkeit, mit der die Metrik für ein Szenario i erfüllt wird. $P(S_i)$ stellt die Auftrittswahrscheinlichkeit des Szenarios S_i dar. Gleichung 3.1 impliziert folgende Fragestellungen zur Aufwandsreduktion der Absicherung:

- Für welche Szenarien lässt sich die Erfüllung der Metrik über andere Wege als reine Black-Box-Vergleichsfahrten herleiten?
- Welche Szenarien haben die höchste beziehungsweise die geringste Relevanz $P(S_i)$?

Abbildung 3.2 visualisiert eine mögliche Route von Stuttgart nach München. Nach [Michelin 2017] beträgt die Distanz dieser Strecke 233 km, davon 221 km (95%) auf der Autobahn. Dies zeigt, dass selbst eine Testfahrt zwischen Stuttgart und München nur teilweise auf der Autobahn stattfindet. Im alltäglichen Einsatz befinden sich Fahrzeuge ebenfalls nicht ausschließlich auf der Autobahn. Dies unterstreicht die Notwendigkeit Vergleichsgruppen vollständig zu definieren, wenn Tests die Absicherung des vollständigen Anwendungsbereichs von Fahrzeugen fokussieren.

Für die exemplarische Testfahrt zwischen Stuttgart und München könnte Frage a) folgendermaßen beantwortet werden: Betrachtet werden die in Tabelle 3.1 aufgelisteten Szenarien 5...8, deren Einsatzbereich *Nicht Autobahn* ist. Könnte durch zusätzliches Systemwissen sichergestellt werden, dass der Autobahnautomat nur auf Autobahnen aktiviert ist, so kann die Wahrscheinlichkeit zur Erfüllung der Metrik (*Nicht mehr Unfälle als ohne System*) für die Szenarien 5.. 8 mit $P_i = 1$ angenommen werden. Ob sich das Fahrzeug auf der Autobahn befindet könnte beispielsweise durch ein Positionierungssystem überprüft werden. Nun müsste nicht der Autobahnautomat außerhalb von Autobahnen abgesichert werden, sondern die Zuverlässigkeit der GPS-basierten

Standortermittlung.

Frage b) bezüglich der Relevanz von Vergleichsgruppen lässt sich exemplarisch am Beispiel des Verbesserungsfaktors klären: Der Verbesserungsfaktor resultiert aus Betrachtungen der Unfallzahlen im Hinblick auf den stetig steigenden technischen Fortschritt von Fahrzeugen. Auch bei Fahrzeugen ohne Assistenzsystem hat sich in den letzten Jahren eine kontinuierliche Abnahme der Verkehrstoten gezeigt. Da sich diese Abnahme voraussichtlich nicht schlagartig vergrößern wird, kann man argumentieren, dass die Relevanz der Vergleichsgruppe $V_{\text{Verbesserung}} > 1,2$ als gering einzustufen ist. Bei einer Priorisierung der Absicherungsaufgaben kann dies als Entscheidungsgrundlage nützlich sein.

Neben dieser qualitativen Formulierung der Relevanz eines Szenarios können auch Fahrstatistiken verwendet werden. Sind Statistiken zu einer Vergleichsgruppe vorhanden, so kann daraus die Relevanz der dazugehörigen Szenarien abgeleitet werden. In etwa 60% der Unfälle auf Autobahnen ist der Fahrzeugführer *Hauptverursacher* [Vgl. Destatis 2017].

Die Anzahl unterschiedlicher Szenarien n ergibt sich aus der Mächtigkeit der disjunkten Vergleichsgruppen V_i zu

$$n = \prod |V_i| \quad (3.2)$$

Für das simple Beispiel des Autobahnautomaten ergeben sich acht unterschiedliche Szenarien anhand der bisherigen Vergleichsgruppen:

$$\begin{aligned} n &= |V_{\text{Einsatz}}| \cdot |V_{\text{Verursacher}}| \cdot |V_{\text{Verbesserung}}| \\ &= 2 \cdot 2 \cdot 2 \\ &= 8 \end{aligned}$$

Wie Gleichung 3.2 zeigt hängt der Absicherungsaufwand von der Anzahl der Elemente jeder Vergleichsgruppe ab.

Zur Unterscheidung der Fahrgeschwindigkeiten könnte zusätzlich die Vergleichsgruppe $V_{\text{Geschwindigkeit}}$ eingeführt werden. Ohne weitere Klassifizierung hat diese Vergleichsgruppe eine unendliche Anzahl möglicher Fahrgeschwindigkeiten von beispielsweise $v = [0 \dots 300] \frac{\text{km}}{\text{h}} \in \mathbb{R}$. Damit wäre auch die Anzahl der zu testenden Szenarien unendlich. Dies ist für eine praxisrelevante Absicherung unmöglich. Somit ist für diesen Parameterbereich eine Klassifizierung nötig. Das Vorgehen bei einer solchen Klassifizierung lässt sich durch den Ansatz der Äquivalenzklassenbildung für funktionsorientierte Tests begründen.

In [Liggesmeyer 2002, S. 53] wird die Notwendigkeit zur Einführung von Äquivalenzklassen wie folgt motiviert: „Die Durchführung eines erschöpfenden

Tests ist für reale Programme in der Regel nicht möglich.“ Folgende triviale Funktionsdeklaration veranschaulicht diese Tatsache:

```
unsigned int addOne(unsigned int v);
```

Für einen vollständigen Test dieser Methode müssten bei einer 32-bit Darstellung eines `unsigned int` Datentyps alle 2^{32} Eingabewerte überprüft werden.

Um dem zu begegnen werden Klassen äquivalenter Eingabebereiche (sogenannte Äquivalenzklassen) definiert. Ein Parameterbereich stellt eine Äquivalenzklasse dar, sofern sich die zu testende Funktion für jeden Repräsentanten der Klasse äquivalent verhält. Betrachtet man den Überlauf des Datentyps `unsigned int`, so wären die Bereiche $c_0 = [0 \dots 2^{32} - 2]$ und $c_1 = [2^{32} - 1]$ zwei mögliche Äquivalenzklassen. Für folgende Implementierung decken diese zwei Äquivalenzklassen den Eingabebereich der Methode vollständig ab:

```
unsigned int addOne(unsigned int v) {
    return v + 1;
}
```

Ohne Wissen über die Implementierung der `addOne` Methode kann dies jedoch nur eine Vermutung sein. Die Funktionsdefinition könnte (aus welchem Grunde auch immer) auch folgendermaßen aussehen:

```
unsigned int addOne(unsigned int v) {
    if (v == 100) {
        return v;
    } else {
        return v + 1;
    }
}
```

Für diese Implementierung wären die oben aufgeführten Äquivalenzklassen falsch. Für den Eingabeparameter $v = 100$ würde sich die Methode anders verhalten als für die restlichen Werte der Äquivalenzklasse $c_0 = [0 \dots 2^{32} - 2]$. Ohne Wissen über die Implementierung ist es im Allgemeinen nicht möglich die Äquivalenzklassen korrekt und vollständig zu wählen.

Für Tests, die auf Äquivalenzklassen basieren, werden Repräsentanten¹⁵ der Klassen als Eingabeparameter verwendet. Dieses Vorgehen hat zur Folge, dass einige wenige Tests ausgewählt werden, die idealerweise repräsentativ für ihre Parameterbereiche stehen. Gleichzeitig werden dadurch nicht alle möglichen Eingabeparameter getestet. [Vgl. Liggesmeyer 2002, S.39 ff.]

¹⁵Häufig werden auch Werte aus Randbereichen zur Analyse verwendet

Aufgrund der praktischen Notwendigkeit zur Durchführung von Tests wird der Ansatz der Äquivalenzklassenbildung auf Parametern von Testszenarien weiterverfolgt. Angenommen man wähle einen naiven Startpunkt und behauptete, für das abzusichernde Assistenzsystem sei es irrelevant, ob der Test auf einer trockenen Straße S_t oder einer nassen Straße S_n erfolge. Folglich gilt für die Wahrscheinlichkeit zur Erfüllung einer Metrik M im Hinblick auf die Straßenzustände: $P(M | S_t) = P(M | S_n)$. Diese Hypothese könnte aufgrund nicht vollständig erfasster Einflussfaktoren entstanden sein. Ohne falsifizierende Beobachtung der Hypothese würden nun Testfahrten zum Nachweis der Erfüllung der Metrik M gestartet. Während der Tests zeigt sich, dass die Metrik bei Fahrten auf trockenen Fahrbahnen problemlos erfüllt wird ($P(M | S_t) = 1$), nicht aber auf nassen Straßen ($P(M | S_n) < 1$). Diese während der Absicherung gewonnene Erkenntnis legt den Schluss nahe, eine weitere Vergleichsgruppe für unterschiedliche Straßenoberflächen einzuführen. Dies erlaubt einerseits eine unabhängige Absicherung der unterschiedlichen Bereiche (trocken/nass) und fokussiert gleichzeitig den Lösungsraum zur Behebung des Problems auf einen Bereich.

Führt man die Idee der Vergleichsgruppen weiter, so sollte eine Beschreibung von Fahrerszenarien alle als relevant identifizierten Parameter beinhalten. Eine solche *semantische Szenariobeschreibung* dient der systematischen Beschreibung von Einflussgrößen in Verkehrsszenarien. Gleichzeitig kann sie als iterativ-erweiterbare Dokumentationsgrundlage für Fahrversuche dienen. Auf eine mögliche Darstellung einer solchen Beschreibung für Verkehrssituationen wird in Kapitel 3.2 näher eingegangen. Grundsätzlich besteht diese aus Klassen von Parametern, die basierend auf aktuellem Erkenntnisstand sinnvoll für die Beschreibung von Testfällen für Fahrerassistenzsysteme erscheinen.

Analog zur Entdeckung neuer, relevanter Parameter während der Testdurchführung können auch Kombinationen bereits bekannter Parameter Verkehrssituationen zur Folge haben, die bisher nicht speziell bedacht wurden. Dieses Phänomen resultiert aus dem oben geschilderten Vorgehen der Äquivalenzklassenbildung ohne Kenntnis über die Funktionsimplementierung. Wie der im Folgenden beschriebene Unfall zwischen einem teilautonomen Fahrzeug und einem LKW zeigt, waren die relevanten Parameter bekannt. Allerdings war die zum Unfall führende Parameterkonstellation nicht getestet und wurde im Rahmen der Spezifikation auch nicht gesondert betrachtet (*Funktionale Unzulänglichkeit*).

Im Szenario, welches zum Unfall führte, bewegte sich ein weißer Lastwagen an einer Kreuzung orthogonal zum teilautonomen Fahrzeug. Dieses Fahrzeug war mit Video- und Radarsensoren ausgestattet. Der Lastwagen wurde dabei von der Sensorik des Assistenzsystems fälschlicherweise als Straßenschild kogniziert.

„Bei diesem Unfall führte die hohe weiße Seitenwand des Anhängers zusammen mit einer Radar-Signatur, die der eines hochhängenden Straßenschildes sehr ähnlich war, dazu, dass keine automatische Bremsung ausgelöst wurde“ [Zeit Online 2016]. Die *Geometrie*, die optische *Farbe* sowie die *Radarsignatur* sind Parameter, die im Rahmen der Entwicklung sicherlich beachtet wurden. Die zum Unfallzeitpunkt vorherrschende Konstellation dieser drei Parameter war jedoch ungetestet. Diese Verkehrssituation kann als *kritisches Szenario* eingestuft werden. Eine solche Situation, die bei dem zu testenden System eine unerwünschte Reaktion verursacht, muss als *kritisches Szenario* reproduzierbar in einen Testkatalog aufgenommen werden. Diese Sammlung von Verkehrsszenarien stellt ein zentrales Element der Absicherungsstrategie dar. Dadurch wird es möglich der *Funktionalen Unzulänglichkeit* dahingehend zu begegnen, dass die im Vorfeld unvollständige Spezifikation des Systems während einer Testphase kontinuierlich erweitert werden kann. Abbildung 3.3 stellt das Symbol dar, welches im weiteren Verlauf dieser Arbeit für den Szenarienkatalog verwendet wird.



Abb. 3.3: Symbol des Szenarienkatalogs

Damit Testszenarien nicht ausschließlich aus tragischen Unfällen im realen Straßenverkehr resultieren, sind zusätzliche Quellen für den Szenarienkatalog notwendig. Diese Quellen werden hier in drei qualitativ unterschiedliche Kategorien gegliedert:

a. **Black-Box**

Black-Box-Ansätze setzen kein spezielles Wissen über das zu testende Assistenzsystem voraus. Der Name *Black-Box* kann daher als „Nicht einsehbares System“ verstanden werden. Dass kein Systemwissen notwendig ist, stellt einerseits einen großen Vorteil dar, weil Black-Box-Ansätze somit vielseitig für unterschiedliche Systeme anwendbar sind. Gleichzeitig werden keine falschen Annahmen über das System getroffen, da keine Analyse notwendig ist. Andererseits gibt es kaum Einschränkungen hinsichtlich der Testdurchführung. Dies führt, wie am Beispiel Zählerstatistischer Absicherung des Autobahnautomaten dargestellt, zu einem enormen Testaufwand. Basiert eine Absicherung von Assistenzsystemen ausschließlich auf dieser Kategorie, so übersteigt der Testaufwand schnell finanzielle und zeitliche Ressourcen der entwickelnden Unternehmen.

Der statistische Absicherungsansatz gehört zu dieser Kategorie. Dauerläufe mit Prototypen lassen sich ebenfalls als Black-Box-Ansatz kategorisieren. Verkehrsszenarien werden dabei nicht (oder nur sehr grob) eingeschränkt und interessante Konstellationen ergeben sich zufällig während des Dauerlaufs. Die beschriebene Unfallsituation ist ein Beispiel dieser Kategorie. In Abschnitt 3.3-b wird ein probabilistischer Ansatz zur Generierung von Szenarien diskutiert. Dieses Vorgehen erfordert kein spezielles Wissen über das zu testende Assistenzsystem.

b. Gray-Box

Beim Gray-Box-Ansatz basieren die generierten Szenarien auf einer abstrahierten Darstellung des Systems. Der Name *Gray-Box* steht deshalb für den Bereich zwischen nicht-vorhandenem Systemwissen (black) und vollständiger Systemanalyse (white). Vorteilhaft hierbei ist die Möglichkeit, dass ein systematisches Vorgehen zur Testfallgenerierung möglich ist. Anhand einer Systembeschreibung kann die Ableitung von Fahrscenarien erfolgen. Die Notwendigkeit dieser Systembeschreibung stellt gleichzeitig eine mögliche Fehlerquelle dar, da hierfür eine Abstrahierung des realen Systems notwendig ist. Beinhaltet diese Abstrahierung Fehler, so sind auch die darauf basierenden Ableitungen fehlerbehaftet.

Spezifikationen oder Modelle von Assistenzsystemen, die als Grundlage für die Testfallgenerierung dienen, gehören in die Gruppe der Gray-Box-Ansätze. In Abschnitt 3.3-a wird ein Ansatz zur systematischen Generierung von Testfällen vorgestellt. Ausgehend von einer Analyse des Assistenzsystems werden hierbei mit Hilfe eines Modells des Gesamtsystems Szenarien generiert. Diese Szenarien testen schließlich Repräsentanten aller Parameterbereiche, die sich aus der Analyse des Assistenzsystems ergeben.

Auch Experteneinschätzungen, die beispielsweise auf der Kenntnis Systeminterner Abläufe basieren, können dem Gray-Box-Ansatz zugeordnet werden.

c. White-Box

Bei White-Box-Analysen wird der Quellcode des zu testenden Systems betrachtet. Im Bereich der allgemeinen Softwaretests stellt die statische Quellcodeanalyse einen solchen Ansatz dar. Die im Quellcodebeispiel 3.1 dargestellte Funktionsdefinition soll exemplarisch mittels eines Pfadüberdeckungstests getestet werden.

Quellcode 3.1: Funktionsdefinition `getNearObjects` in C++

```

list<Object> getNearObjects(list<Object> allObjects)
{
    auto nearObjects = list<Object>();

    for (auto object : allObjects) {
        if (object.distance() < 5.0) {
            nearObjects.push_back(object);
        }
    }

    return nearObjects;
}

```

Durch die Analyse des Quellcodes kann der in Abbildung 3.4 dargestellte Kontrollflussgraph erstellt werden. Dieses Beispiel veranschaulicht die Her-

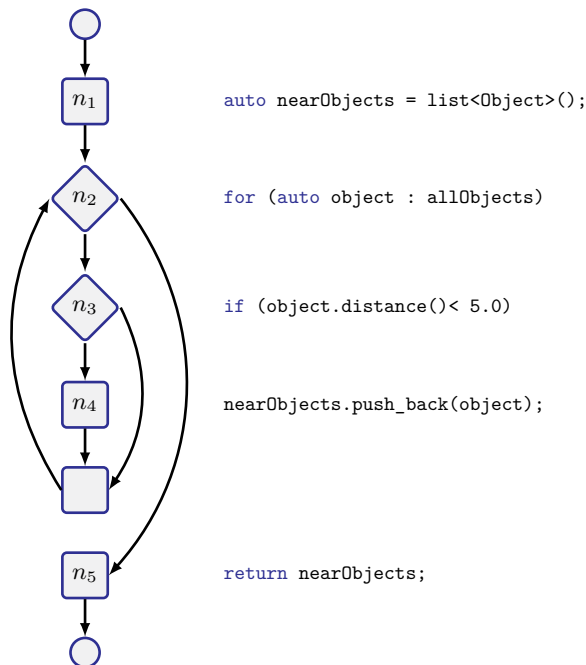


Abb. 3.4: Kontrollflussgraph für Quellcode 3.1

ausforderungen bei der Erstellung von Pfadüberdeckungstests. Die Länge der `for`-Schleife ist erst zur Laufzeit bekannt. Um alle Möglichkeiten der Funktionsausführung zu testen, müssten somit alle möglichen Eingabelängen getestet

werden. Da sich innerhalb der `for`-Schleife zudem eine `if`-Bedingung befindet, multipliziert sich der Testaufwand. Das gleiche gilt für ineinander verschachtelte Schleifen.

Angenommen der Eingangsparameter `list<Object> allObjects` bestünde aus maximal 4 Objekten, so ergeben sich mit der leeren Liste daraus 5 mögliche Listenlängen. Für jedes Element der Liste müssen je beide Zweige der `if`-Bedingung durchlaufen werden. Für die simple Funktion ergeben sich somit

$$\begin{aligned} n &= 2^0 + 2^1 + 2^2 + 2^3 + 2^4 \\ &= \sum_{i=0}^4 2^i = 2^5 - 1 \\ &= 31 \end{aligned}$$

Testfälle, um alle Pfade zu durchlaufen. Um konkrete Eingangswerte zu erhalten, muss die Logik des Programms invertiert werden. Diese Invertierung ist für das vorliegende Beispiel simpel. Um beide Pfade der `if`-Bedingung zu durchlaufen, muss `object.distance()` je einen Wert kleiner und größer als 5,0 zurückgeben. Aus der Kombination dieser `object`-Elemente für alle 5 Listenlängen ergeben sich schließlich die 31 Testfälle.

Dieses Prinzip lässt sich grundsätzlich auch auf die Generierung von Testfällen für Assistenzsysteme übertragen. Wie sich jedoch zeigt, steigt die Anzahl der Testfälle sehr schnell. Deshalb ist eine direkte White-Box-Analyse auf dem Quellcode der Assistenzfunktionen nicht praxistauglich. Neben der hohen Testanzahl kommt erschwerend hinzu, dass eine Invertierung der Funktion bei komplexen Datentypen nicht trivial ist. Für die Objekterkennung in einem Bild beispielsweise müssten zur Invertierung Bilddateien generiert werden, die sich für speziell eine Pfadausführung der Methode eignen.

In Abschnitt 3.3-a wird ein ähnlicher Ansatz näher beschrieben. Auf einem vereinfachten Modell der Assistenzfunktion kann eine solche Analyse zur Testfallerstellung auch praxistauglich angewandt werden. Dieser Ansatz verwendet zwar das Vorgehen von White-Box-Analysen, arbeitet dabei aber auf einer Abstraktion und ist deshalb als Gray-Box-Ansatz einzuordnen.

Die dargestellten Quellen zur Erstellung des Szenarienkatalogs bieten drei qualitativ unterschiedliche Ansätze zur Generierung von Szenarien. Sie können parallel und unabhängig voneinander verwendet werden, um so eine breite Testabdeckung zu erhalten und die jeweiligen Nachteile der Ansätze zu reduzieren. Abbildung 3.5 erweitert die Darstellung des Szenarienkatalogs um die beschriebenen Ansätze zur Szenariengenerierung.

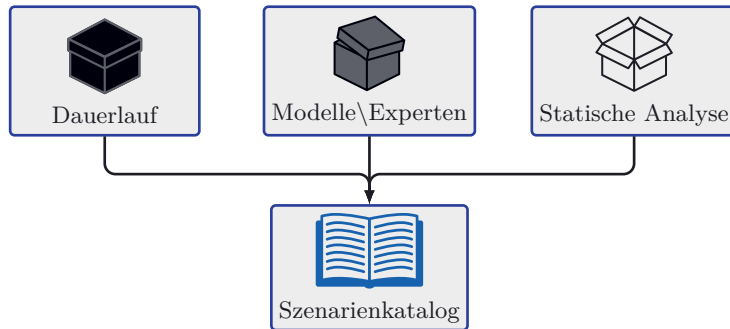


Abb. 3.5: Quellen zur Generierung von Szenarien

Ist ein initialer Szenarienkatalog vorhanden, so gilt es diese Fahrscenarien als Grundlage für Testfälle zu verwenden und auszuführen. Für reale Testfahrten können diese Szenarien als Ablaufplan aufbereitet werden, die einem Testfahrer Hinweise geben, welche Fahrmanöver auszuführen sind. Dadurch lassen sich Testfahrten im Vergleich zum zufälligen Dauerlauf strukturieren.

Ein kritisches Fahrscenario für einen Notbremsassistenten kann beispielsweise die Parameter *Autobahn*, *Nasse Fahrbahn*, *Tunneleinfahrt* und *Geschwindigkeit* $> 80 \frac{\text{km}}{\text{h}}$ beinhalten. Für die Aufbereitung des Szenarios als Ablaufplan einer realen Testfahrt wird eine Strecke mit einem Autobahntunnel vorgeschlagen. Gleichzeitig ist der Hinweis notwendig, dass die Testfahrt während oder nach Niederschlag durchzuführen ist.

Bei einer hohen Anzahl an Szenarien des Katalogs lassen sich die daraus resultierenden Ablaufpläne zudem nach Kriterien sortieren, was eine effiziente Abarbeitung der Testfahrten erlaubt. Das Softwareprogramm [INCA-FLOW] stellt unter anderem die Funktionalität einer geführten Applikation am Fahrzeug zur Verfügung. Unter Verwendung eines im Fahrzeug montierten Tablet-PCs kann ein Applikateur oder Testfahrer die im Tool beschriebenen Schritte ausführen. Gleichzeitig können Werte aus dem Fahrzeug aufgezeichnet oder Haltepunkte für eine spätere Analyse gesetzt werden. Mit Hilfe dieser Anwendung kann eine Szenariobeschreibung somit auch praxistauglich als Ablaufplan aufbereitet werden.

Bei der Verwendung der Szenarien als Ablaufplan ergeben sich im realen Einsatz stets Unsicherheiten und zufällige Ereignisse beeinflussen die Durchführung der beschriebenen Verkehrsszenarien. Somit kann sich die korrekte Umsetzung eines Szenarios als aufwändig oder unmöglich herausstellen. Gleichzeitig bieten die unerwarteten Ereignisse eine Variation analog der Zufallsereignisse beim Dauerlauf. Ergeben sich dadurch interessante Parameterkonstellationen oder die Erkenntnis neuer Einflussparameter, so lassen sich diese in den Szenari-

enkatalog rückführen. Bleibt diese Transition aus, so sind neue Erkenntnisse nicht nachhaltig dokumentiert und erkannte Probleme wertlos.

Neben der Verwendung als Ablaufplan dient die maschinenlesbare Beschreibung der Szenarien auch zur Parametrierung eines virtuellen Fahrversuchs. Dies erlaubt die nahtlose Einbindung der Simulation in die Absicherungsstrategie. Sowohl reale als auch virtuelle Testfahrten basieren somit auf einer gemeinsamen Beschreibung von Fahrscenarien.

Ein Vorteil der Simulation ist die reproduzierbare Ausführung der Testfahrten. Dabei stellt jedoch die Modellierung der Umgebung eine Herausforderung dar. Je mehr Informationen durch die Umfeldsensorik wahrgenommen werden, desto aufwändiger ist die Erschaffung virtueller Welten zur Stimulation der Sensorik. Um sicherzustellen, dass eine virtuelle Umgebung zum Test von Assistenzsystemen verwendet werden kann, ist die Validierung von Szenarien notwendig. Hierfür ist ein Abgleich realer Messungen mit dem virtuellen Pendant notwendig. In Kapitel 5.2 wird eine solche Validierung vorgestellt.

Hinsichtlich der Kosten ist die Simulation von Verkehrsszenarien im Vergleich zu realen Testfahrten mit Prototypen in der Regel deutlich günstiger. Die parallele Ausführung der Szenarien kann zu einer Zeitersparnis führen.

Im Vergleich mit realen Testfahrten werden in der virtuellen Welt keine zufälligen, neuen Einflussgrößen auftauchen. Jedoch eignet sich die Simulationsumgebung zur Variation von Parametern. Ergebnis einer solchen Parametervariation kann die Entdeckung kritischer Verkehrssituationen sein, bei denen die Unit Under Test nicht wie erwartet reagiert. Ist ein solches Szenario gefunden, so lohnt sich dessen Rückführung in den Szenarienkatalog.

Aufgezeichnete Sensordaten können zur Optimierung der Algorithmen oder für Regressionstests verwendet werden. So muss nach einer Anpassung der Algorithmen nicht jede Testfahrt wiederholt werden, sofern die Tests ohne Rückkopplung auf die Sensordaten (Open-Loop) realisierbar sind. Die aufgezeichneten Daten können zudem zur Validierung virtueller Testfahrten verwendet werden.

Mit der Verwendung des Szenarienkatalogs zur Beschreibung virtueller und realer Testfahrten sind alle Elemente der hier vorgestellten Absicherungsstrategie eingeführt. Deren Kombination zu einem ganzheitlichen Ansatz ist in Abbildung 3.6 visualisiert. Es ergibt sich eine iterativ wachsende Anzahl an Szenarien, die als praxistaugliche und transparente Grundlage zur Beschreibung von Testfällen zur Freigabe von Fahrerassistenzsystemen dienen können. Der folgende Abschnitt präsentiert eine mögliche ontologische Beschreibung von Verkehrsszenarien. Diese wird anschließend zur Generierung von Testfällen verwendet.

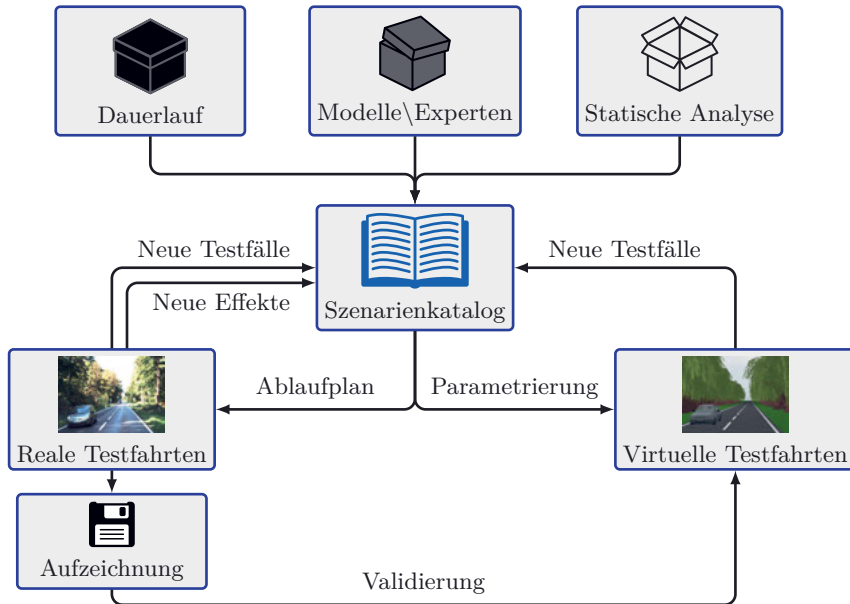


Abb. 3.6: Sequenzdiagramm der Absicherungsstrategie

3.2. Ontologische Beschreibung von Verkehrsszenarien

Bei der Beschreibung von Verkehrsszenarien zum Testen von Fahrerassistenzsystemen stellt sich die Herausforderung, die Komplexität der realen Fahrzeugumgebung auf möglichst wenig beschreibende Parameter abzubilden. Dabei müssen diejenigen Parameter, die einen Einfluss auf das Verhalten des zu testenden Systems darstellen, erhalten bleiben.

Aufgrund der komplexen Schnittstellen zwischen Assistenzsystemen und der Umgebung ist die Ableitung einer solchen Beschreibung nicht trivial. Welche abstrakt beschriebenen Größen beispielsweise Einfluss auf die Objekterkennung innerhalb eines Videobilds haben, kann nicht immer vorhergesagt werden. Bei der Verwendung von neuronalen Netzen beispielsweise können minimale Manipulationen einzelner Pixel das darauf abgebildete Objekt für den Klassifikator völlig entstellen. Für einen Menschen ist währenddessen der Unterschied zwischen dem Original und der Manipulation nicht erkennbar. In [Kurakin 2017] werden solche *adversarial images*¹⁶ präsentiert. Abbildung 3.7 zeigt ein Beispiel einer derartigen Bildmanipulation. Das linke Foto wird von dem verwendeten neuronalen Netz korrekt klassifiziert. Im mittleren Bild ist die (zehnfach verstärkte) Differenz zwischen dem modifizierten und dem originalen

¹⁶Ins Deutsche frei mit „feindseligen Bildern“ zu übersetzen.

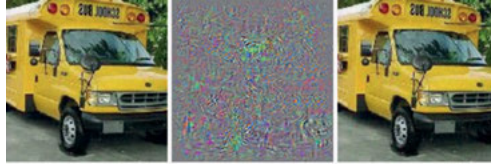


Abb. 3.7: Beispiel einer Bildmodifikation mit dem Ziel ein neuronales Netz zu täuschen [Szegedy 2014]

Foto visualisiert. Der rechte Teil zeigt schließlich das modifizierte (adversarial) Bild. Dieses wird als ein Vogel (afrikanischer Strauß) klassifiziert. Wie sensitiv eine Objekterkennung auf derartige Bildmanipulationen reagiert, hängt vom gewählten Algorithmus ab. Deshalb kann diese Sensitivität ohne zusätzliche Kenntnis der UUT auch nicht im Vorfeld von Systemtests spezifiziert werden.

Dieses Beispiel verdeutlicht, dass es nicht das Ziel einer Beschreibung von Verkehrssituationen sein kann, jede Eingangsgröße der Umfeldsensorik exakt zu spezifizieren. Dies würde für einen Videosensor zu einer pixel-basierten Testfalldefinition führen, was einerseits eine enorme Parametervielfalt bedeuten würde (Vgl. Kapitel 1.1) und andererseits keine zielführende Testfallbeschreibung wäre, da hierbei jegliche Abstraktion fehlt.

Dies wiederum bedeutet, dass eine Szenariobeschreibung, die auf der Ebene des Verkehrsgeschehens arbeitet, Abstraktionen verwendet. Eine solche Modellierung geht mit der Vernachlässigung von Effekten der Realität einher. Dies geschieht entweder wissend, wenn ein Effekt für die betrachtete Aussage irrelevant erscheint, oder unwissend, wenn die Auswirkung eines Effekts auf das betrachtete System nicht bekannt ist. Diese Eigenschaft bildet sich auch auf Testfälle ab, denen eine solche Szenariobeschreibung zugrunde liegt. Als Testinput lassen sich nur Verkehrsszenarien verwenden, die sich mittels der Beschreibung darstellen lassen. Um diesen Nachteil zu adressieren, stellt die iterative Absicherung eine Gegenmaßnahme dar, indem die Beschreibung kontinuierlich verbessert und angepasst werden kann.

Für eine ontologische Beschreibung von Verkehrsszenarien, die sich zur Anwendung in der vorgestellten Absicherung eignet, ergeben sich die nachfolgend beschriebenen Randbedingungen:

Maschinenlesbarkeit

Diese Eigenschaft ist notwendig, damit Simulationsumgebungen mittels der Beschreibung parametrisiert werden können. Die Transformation der Ontologie in einen Ablaufplan für einen Testfahrer lässt sich durch die Maschinenlesbarkeit automatisieren.

Erweiterbarkeit

Damit sich eine Beschreibung für die iterative Absicherung eignet, muss eine Erweiterung durch neue Parameter problemlos möglich sein.

Objektvielfalt

Da sich eine Vielzahl unterschiedlicher Objekte in der Fahrzeugumgebung befindet, ist es notwendig, diese Variabilität auch konsistent in einer Szenariobeschreibung abbilden zu können.

Instanziierung

Damit Umgebungsobjekte einmalig beschrieben und mehrfach verwendet werden können, ist die Möglichkeit zur Instanziierung notwendig.

Spezifikation dynamischer Vorgänge

Im Rahmen von Assistenzsystemen handelt es sich stets um Verkehrsszenarien, innerhalb derer sich Akteure bewegen. Eine Möglichkeit zur Darstellung dieser dynamischen Vorgänge ist in einer Szenariobeschreibung ebenfalls notwendig.

Systemunspezifisch

Die Beschreibung der virtuellen Testfahrt sollte möglichst unabhängig vom zu testenden System sein. Analog zu realen Testfahrten sollte die Definition eines Szenarios nicht davon abhängig sein, ob dabei ein Fahrzeug mit Assistenzsystem A oder B verwendet wird.

Unabhängigkeit hinsichtlich eines Simulationstools

Wird im Rahmen der Absicherungsstrategie ein Szenarienkatalog entwickelt, so sollte dieser unabhängig von dessen Ausführungsmöglichkeit sein. Dadurch ist der Szenarienkatalog nicht ausschließlich auf ein Simulationswerkzeug beschränkt, sondern kann vielfältig eingesetzt werden.

Wie in Kapitel 2.3 dargestellt, gibt es bereits unterschiedliche Tests für Assistenzsysteme. Die Abschlussarbeit [Schnarr 2016] listet ausführlich Parameter verschiedener Testbeschreibungen auf.

Basierend auf dieser Auflistung von Parametern realer Fahrversuche dienen die folgenden drei Gruppen als Grundlage zur Beschreibung einer Verkehrssituation. Zunächst ist eine Beschreibung des **Ortes** notwendig, an dem sich das Szenario abspielt. Zusätzlich müssen die am Ort vorhandenen **Akteure** spezifiziert werden, die an einer Verkehrssituation teilnehmen. Letztlich ist eine Beschreibung der **Handlung** notwendig, die von den am Ort vorhandenen Akteuren ausgeführt wird.

Die Anforderungen an den Ort einer Verkehrssituation sind vielfältig. Die Beschreibung des Ortes beinhaltet Informationen über den Straßenverlauf,

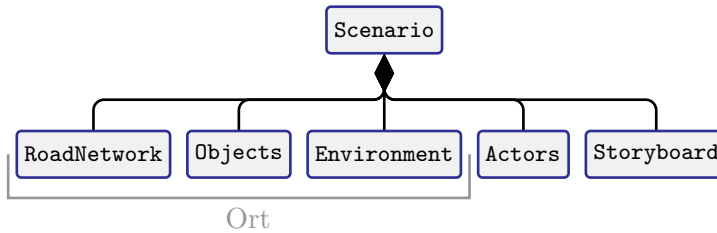


Abb. 3.8: Oberste Hierarchieebene der Szenariobeschreibung

dessen Randbebauung, vorhandene Gebäude, Brücken oder Vegetation. Gemeinsam ist den Elementen dieser Gruppe, dass sie statisch sind und sich ihre Eigenschaften, wie Position oder Farbe, während des Verkehrsszenarios nicht ändern. Da sich Verkehrsszenarios vorrangig im Umfeld einer Straße abspielen, scheint es sinnvoll die Beschreibung des Straßenverlaufs separat aufzuführen. Neben den Gebäuden und der Randbebauung gibt es zudem Umgebungseigenschaften, die eher globalen Charakter besitzen. Die vorhandene Beleuchtungssituation aufgrund der Sonneneinstrahlung oder die Sichtverhältnisse stellen solche globalen Umgebungseigenschaften dar. In der ontologischen Darstellung des Verkehrsszenarios gliedert sich die Ortsbeschreibung somit in drei Teile: a) Informationen zum Straßenverlauf (**RoadNetwork**), b) Details zu statischen Objekten des Ortes (**Objects**) sowie c) der Beschreibung globaler Umgebungseigenschaften (**Environment**).

Die Akteure sind ebenfalls am Ort vorhandene Objekte, deren Eigenschaften sich allerdings im Laufe der Verkehrssituation ändern. Als Beispiele dieser Gruppe sind PKWs, LKWs, Fußgänger, Radfahrer sowie Wechselverkehrszeichen oder Lichtsignalanlagen zu nennen. In der ontologischen Beschreibung des Verkehrsszenarios werden diese dynamischen Elemente zur Gruppe **Actors** zusammengefasst.

Neben der Beschreibung des Vorhandenseins unterschiedlicher Elemente innerhalb des Verkehrsszenarios ist zusätzlich die Beschreibung des Ablaufs notwendig. Ziel dieser Ablaufbeschreibung ist die Definition, wie sich Akteure während eines Verkehrsszenarios an dem beschriebenen Ort verhalten. Informationen hierzu werden in der Gruppe **Storyboard** zusammengefasst.

Auf oberster Hierarchieebene enthält der Vorschlag zur ontologischen Beschreibung eines Verkehrsszenarios somit die in Abbildung 3.8 dargestellten Elemente. Bevor die einzelnen Elemente detaillierter erörtert werden, wird das technische Format zur Speicherung des Szenarios diskutiert. Aus den Anforderungen leitet sich die Notwendigkeit eines maschinenlesbaren Formats ab. Damit die Beschreibung zudem in mehreren Anwendungen Verwendung finden kann,

sollte sie Betriebssystem-unabhängig sein und idealerweise ein möglichst weit verbreitetes Format haben. Gleichzeitig ist die Unterstützung hierarchischer Strukturen sinnvoll, um die Objektvielfalt sowie Instanziierung abbilden zu können. Zusätzlich sollte eine komfortable Definition der dynamischen Vorgänge möglich sein, ähnlich einer Skriptsprache.

Zur Beschreibung strukturierter Daten gibt es mehrere verbreitete Optionen. Dazu gehört die Erweiterbare Auszeichnungssprache (engl. Extensible Markup Language, XML), die JavaScript Object Notation (JSON) oder Protocol Buffers¹⁷. Da sich die Formate ineinander überführen lassen, könnte jede Option zur Beschreibung der Szenarien gewählt werden. Der Performance Vorteil von Protocol Buffers würde für deren Verwendung sprechen. Demgegenüber steht die nicht sehr fortgeschrittene Toolunterstützung und Verbreitung des Formats. Zudem wird die Entwicklung der Bibliotheken, die benötigt werden um mit Protocol Buffers zu arbeiten, hauptsächlich durch die Firma Google LLC geführt. Aufgrund der weit verbreiteten Anwendung von XML und dessen guter Unterstützung in Entwicklungsumgebungen, wie beispielsweise Microsoft Visual Studio, wird das XML-Format im Rahmen dieser Arbeit zur Beschreibung des Verkehrsgeschehens gewählt.

Die folgenden Abschnitte erläutern den Aufbau der fünf vorgestellten Elemente der Szenariobeschreibung. Ein dazugehöriges XML-Schema ist in Anhang B zu finden. Grundsätzlich ist hierbei zu beachten, dass es sich bei diesem Vorschlag um einen Startpunkt der Beschreibung handelt, die im Laufe einer Absicherung erweitert oder modifiziert werden kann.

a. Das Straßennetzwerk (RoadNetwork)

Ein zentrales Element bei der Beschreibung einer Straße ist der Straßenverlauf. Dieser kann anhand einer dreidimensionalen Referenzlinie der Straßenmitte definiert werden. Auf den Seiten dieser Referenzlinie verlaufen die einzelnen Fahrspuren. Abbildung 3.9 überlagert diese Eigenschaften dem Bild einer realen Straße. Zusätzlich ist die Beschreibung von Fahrstreifen und Straßenmarkierungen notwendig. Bei der Kombination mehrerer Straßenzüge kommt es zu Kreuzungen und zu- sowie abführenden Straßenelementen. Da es viele Objekte im Umfeld der Straße gibt, ist es sinnvoll deren Definition in Relation zum Straßenverlauf zu ermöglichen. Hierfür ist ein Straßenkoordinatensystem notwendig.

Für diese Art der Straßenbeschreibung gibt es, basierend auf Fahrdynamiksimulationen, bereits unterschiedliche Beschreibungen. Hierzu zählen proprietäre

¹⁷Ein Gegenentwurf von Google zur XML Beschreibung mit dem Fokus auf hohe Performance und geringe Datengröße [Vgl. Google LLC 2018a]



Abb. 3.9: Exemplarische Parameter einer Straße, basierend auf [Straße 2015]

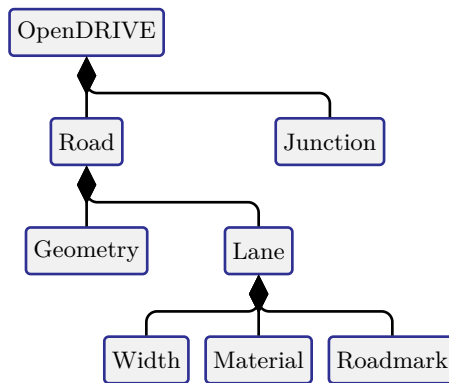


Abb. 3.10: Wesentliche Elemente von OpenDRIVE

Straßenformate (wie PEX der Firma Tass oder ROAD5 der Firma IPG), sowie das im Rahmen eines Konsortiums entstandene Format [OpenDRIVE]. Dieses Format eignet sich für die Beschreibung einer Vielzahl von Straßenverläufen, einschließlich Kreuzungen sowie zu- und abführenden Straßenelementen. Eine optional hochauflösende Beschreibung der Straßenoberfläche kann durch den Verweis auf ein entsprechendes Gitter (im OpenCRG Format) hinzugefügt werden. Zudem eignet sich die XML Struktur des OpenDRIVE Formats für dessen Referenzierung in der Szenariobeschreibung. Abbildung 3.10 visualisiert vereinfacht die wesentlichen Elemente dieses Standards zur Straßenbeschreibung. Details zur Spezifikation aller Elemente sind in [OpenDRIVE] zu finden. Road-Elemente beschreiben einzelne Straßenabschnitte. Deren Geometrie wird

durch die Aneinanderreihung von Liniensegmenten, Bogenabschnitten, Spiralen und Polynomen beschrieben. Damit kann der in Abbildung 3.9 visualisierte Straßenverlauf dargestellt werden. Die Mittellinie definiert gleichzeitig das Straßenkoordinatensystem $\text{COS}_{\text{Road}}(s, t)$. In Relation hierzu wird der Verlauf einzelner Spuren (**Lanes**) beschrieben. Die Spurbreite kann hierbei als Funktion des Straßenverlaufs angegeben werden. Dies ermöglicht die Umsetzung zu- und abführender Spuren, wie beispielsweise bei Ein- und Ausfahrten auf Autobahnen. Innerhalb des **Material** Elements können simple Materialeigenschaften der Straße, wie Reibwert oder ein anwenderspezifischer *Materialcode*, angegeben werden. Diese Materialparameter lassen sich je Spur in Relation zur s -Straßenordinate definieren. Die Angabe von Spurbegrenzungen erfolgt mittels vordefinierter Elemente innerhalb des **Roadmark** Elements.

Kreuzungen werden im Element **Junction** definiert. Dies geschieht mittels der Referenzierung von Straßenelementen sowie der Beschreibung derer Verbindungen untereinander.

Zusätzlich zu den Möglichkeiten, die OpenDRIVE zur Beschreibung der Straße bietet, ergeben sich im Rahmen der Absicherung eventuell weitere Parameter hinsichtlich der Straßenbeschreibung. Die Weiterentwicklung eines Standards, der im Rahmen eines Konsortiums verabschiedet wird, ist ein relativ träger Prozess. Zusätzliche Parameter werden deshalb nicht kurzfristig Einzug in den Standard erhalten, sondern eventuell erst nach einem mehrmonatigen Freigabeprozess. Um dennoch zusätzliche Parameter für die Absicherung verwenden zu können und gleichzeitig die Standardkonformität zu wahren, wird folgender Ansatz gewählt: Innerhalb der Szenariobeschreibung verweist das Element **RoadNetwork** mittels eines Attributs auf ein OpenDRIVE konformes Dokument. Zusätzliche Elemente, wie beispielsweise im Standard nicht unterstützte Straßenmarkierungen, können **RoadNetwork** als Unterelemente zugefügt werden.

Neben dem Know-how, das in die Erstellung von OpenDRIVE eingeflossen ist, stellt das inzwischen vorhandene Tooling einen weiteren Vorteil bei der Verwendung dieses Formats dar. So lassen sich beispielsweise Kartenausschnitte in eine OpenDRIVE konforme Beschreibung umwandeln. Dies vereinfacht den Transfer realer in virtuelle Umgebungen.

b. Statische Objekte (Objects)

Zur Beschreibung der Örtlichkeit eines Verkehrsszenarios gehören neben der Straße auch die statischen Objekte. Die Variabilität möglicher Objekte ist sehr hoch. Gemeinsam haben all diese Elemente jedoch, dass sich deren Lage und äußere Erscheinung beschreiben lässt. Diese zwei Eigenschaften werden

im weiteren Verlauf dieser Arbeit sehr häufig anzutreffen sein und stellen elementare Informationen zur Beschreibung von Objekten dar.

Eine Transformation kann mittels einer Abbildungsmatrix A (häufig eine Rotationsmatrix) sowie eines Translationsvektors \vec{t} dargestellt werden.

$$\vec{v}_{\text{transformiert}} = \vec{v}_{\text{original}} \cdot A + \vec{t} \quad (3.3)$$

Zur Beschreibung von Objekten im Raum kann ein homogenes affines Koordinatensystem verwendet werden. Zusätzliche homogene affine Koordinatensysteme können beispielsweise zur lokalen Beschreibung von Objekten eingeführt werden. Die Transformation, wie in Gleichung 3.3 beschrieben, kann alternativ auch mittels einer Transformationsmatrix M_{affine} dargestellt werden [Vgl. Gregory 2014, 187 ff.]:

$$\begin{aligned} \begin{bmatrix} \vec{v}_{\text{transformiert}} & 1 \end{bmatrix} &= \begin{bmatrix} \vec{v}_{\text{original}} & 1 \end{bmatrix} \cdot M_{\text{affine}} \\ &= \begin{bmatrix} \vec{v}_{\text{original}} & 1 \end{bmatrix} \cdot \begin{bmatrix} A_{3 \times 3} & \vec{0}_{3 \times 1} \\ \vec{t}_{1 \times 3} & 1 \end{bmatrix} \end{aligned}$$

Diese Schreibweise vereinfacht die Darstellung aufeinanderfolgender Transformationen M_i , die sich durch eine Multiplikation $\prod_i M_i$ zusammenfassen lassen. Anstatt der Angabe einer Transformation im globalen Koordinatensystem ist die Platzierung von Objekten relativ zur Straße sinnvoll. Deshalb wird neben der Möglichkeit einer globalen Transformation des Objekts zusätzlich die Option eingeführt, die Translation im Straßenkoordinatensystem $\text{COS}_{\text{Road}}(s, t)$ zu beschreiben. Bei Objekten wie Straßenschildern oder Leitpfosten ist die Positionsbeschreibung relativ zum Straßenverlauf deutlich einfacher.

Zusätzlich zur Lage eines Objekts ist die Beschreibung der Erscheinung notwendig. Für reale Testfahrten kann dies mit Hilfe eines möglichst eindeutigen Objektname definiert werden. So kann ein Straßenschild zur Anzeige einer zulässigen Höchstgeschwindigkeit von $60 \frac{\text{km}}{\text{h}}$ nach [StVO 2013] beispielsweise durch *Straßenschild Z 274-60* beschrieben werden. Für solche klar definierten Gegenstände ist diese Beschreibung einfach. Straßenlaternen sind demgegenüber weniger eindeutig, da es hier eine Vielzahl unterschiedlicher Hersteller und Bauweisen gibt. Hierfür ist ein Objektkatalog notwendig, der einem eindeutigen Objektname zusätzliche Informationen zuordnet, damit der Gegenstand in Tests reproduzierbar abgebildet werden kann.

Die Beschreibung der Vegetation eines Verkehrsszenarios stellt hierbei eine besondere Herausforderung dar. Befinden sich Pflanzen wie Bäume oder Sträucher in der näheren Umgebung des Fahrzeugs, so ist es sinnvoll, diese in die Beschreibung mit aufzunehmen. Die Reproduzierbarkeit im realen Fahrversuch







Name	Informationen	Beispiel	3D Modell
Z 206	Stop-Schild StVO Z 206		 Z_206.obj
Fahrzeug 1	Roter PKW		 Fahrzeug1.obj
Baum 1	Baum Höhe ≈ 10 Meter		 Baum1.obj

Tabelle 3.2: Drei exemplarische Einträge eines Objektkatalogs [Porada 2019; PKW 2018; Knezevic 2018]

ist hierbei nur in groben Grenzen gegeben. So kann beispielsweise die Form, Größe und Art eines Baumes beschrieben werden. Bei mehreren Fahrversuchen, eventuell an unterschiedlichen Orten und Zeiten, ist die Reproduktion der Vegetation mit vertretbarem Aufwand sehr limitiert. Tabelle 3.2 zeigt exemplarische Einträge eines Objektkatalogs. Neben Informationen zur Form kann zusätzlich ein Foto sowie ein dreidimensionales Computermodell referenziert werden, um diesen Gegenstand auch in virtuellen Welten abbilden zu können.

c. Globale Umgebungselemente (Environment)

Einige Eigenschaften der Umgebung beziehen sich nicht direkt auf einzelne Objekte, sondern haben *globalen* Charakter. Solche Elemente werden der Gruppe **Environment** zugeordnet. Die Beschreibung der Umgebungsbeleuchtung gehört zu dieser Kategorie. Grundlegende Parameter zur Beschreibung der Sonnenstrahlung stellen die Lichtfarbe, -intensität und -einfallrichtung dar. Herrscht während eines Testszenarios beispielsweise Nebel vor, so kann dieser auch innerhalb dieses Elements parametrisiert werden.

Auch eine Option zur vereinfachten Beschreibung eines Höhenprofils des Terrains, welches den Straßenverlauf umgibt, wird unter Verwendung abschnittsweise definierter Polynome in dieser Gruppe vorgeschlagen. Zur vereinfachten Definition einer virtuellen Umgebung kann zusätzlich eine Hintergrundgrafik angegeben werden. Dadurch müssen nicht alle Elemente im Sichtbereich eines virtuellen Sensors aufwändig mittels dreidimensionaler Objekte abgebildet werden, sondern lassen sich vereinfacht auf die Darstellung einer Hintergrundgrafik reduzieren.

d. Akteure (Actors)

Sich bewegende Elemente stellen den interessanten Teil eines Verkehrsszenarios dar. Objekte, die während der betrachteten Situation ihre Eigenschaften ändern können, werden in der Kategorie **Actors** zusammengefasst. Um eine eindeutige Definition der Bewegungsabläufe zu erlauben, wird jeder Akteur mit einer eindeutigen Identifikationsnummer versehen. Im Hinblick auf Verkehrsszenarien handelt es sich bei den zu beschreibenden Akteuren vorrangig um Fahrzeuge und Fußgänger. Hierbei ist, wie bereits bei den statischen Objekten, die Definition der Lage und äußeren Erscheinung des Akteurs notwendig. Die Beschreibung der Lage erfolgt global oder relativ zum Straßenverlauf. Zur Beschreibung der äußeren Erscheinung können Informationen des eingeführten Objektkatalogs herangezogen werden.

Ist eine Hierarchisierung sich bewegendere Elemente notwendig, so können diese ineinander verschachtelt werden. Dies eignet sich beispielsweise zur Beschreibung eines Fahrrads mit separat aufgeführtem Fahrradfahrer.

Zur Definition der im Rahmen einer Testfahrt verwendeten Sensorik können **Actor**-Elemente zudem Sensoren beinhalten. Die Angabe der Transformation des Sensors erfolgt relativ zum lokalen Koordinatensystem des zugehörigen Akteurs. Somit kann die Platzierung und Ausrichtung unterschiedlicher Sensoren an einem Fahrzeug im Rahmen der Szenariobeschreibung festgehalten werden. Die konkrete Bezeichnung der Sensorelemente ist frei wählbar und die Definition vorhandener Sensoren kann im Laufe der Entwicklung erweitert werden.

Die *KITTI Vision Benchmark Suite* stellt frei zugängliche Sensordaten realer Testfahrten zur Verfügung. Für einen Transfer dieser Daten in eine virtuelle Umgebung ist unter anderem die genaue Positionsbeschreibung der Sensorik am Fahrzeug notwendig. Diese Informationen sind in [KITTI Setup] für mehrere reale Fahrten aufgelistet. Mit Hilfe der vorgestellten Beschreibung von Akteuren kann dieser Sensoraufbau wiedergeben und in eine Simulationsumgebung transferiert werden.

e. Dynamische Vorgänge (Storyboard)

Die Beschreibung der dynamischen Vorgänge eines Verkehrsszenarios schließt die Definition eines Szenarios ab. Im Hinblick auf reale Testfahrten und Closed-Loop Simulationen kann der Ablauf der Bewegungen innerhalb eines Verkehrsszenarios durch die UUT oder andere Akteure beeinflusst werden. Bei der Beschreibung der Szenarien auf Verkehrsebene sind solche Einflussmöglichkeiten während der Ausführung beabsichtigt. Dies hat zur Folge, dass eine Beschreibung von Trajektorien anhand fester Ortskurven nicht zielführend ist. Im Rahmen von Tests, bei denen der Fokus auf der Fahrdynamik eines einzelnen Fahrzeugs liegt, ist die Vorgabe fixer Trajektorien oder enger Korridore üblich. In [ISO 3888] wird beispielsweise ein doppelter Spurwechsel definiert, der innerhalb eines engen, klar definierten Korridors stattfinden muss. Die Breite b dieses Korridors ist an der engsten Stelle definiert als $b = \text{Fahrzeugbreite} + 1 \text{ Meter}$. [Donges 1982] klassifiziert die Fahraufgabe in die Bereiche Navigation, Bahnführung und Stabilisierung. Die Angabe eines solch engen Korridors definiert eine fixe Trajektorie und ist deshalb für Tests aus dem Fahraufgabenbereich Stabilisierung sinnvoll. Dies zeigt sich auch daran, dass die [ISO 3888] für Tests von elektrischen Stabilitätssystemen vorgesehen ist. Assistenzsysteme und autonome Fahrzeuge übernehmen Aufgaben aus den Bereichen Bahnführung und Navigation. Bei Tests dieser Systeme steht die Kognition und Perzeption der Umgebung mit eventuell bewegten Elementen im Vordergrund. Zur Beschreibung der dynamischen Vorgänge der Verkehrssituation sind deshalb andere Ansätze nötig.

In [Ulbrich 2015] wird der Begriff *Szenario* „als zeitliche Abfolge von Aktionen [...] und Szenen“ verstanden. Eine Szene stellt hierbei „eine Momentaufnahme des Umfelds“ dar. Nach diesem Verständnis werden mehrere Szenen mittels Aktionen zu einem Szenario komponiert.

Aufbauend hierauf wird nun eine Definition dynamischer Vorgänge vorgeschlagen, die sich für eine formale Beschreibung realer und virtueller Fahrversuche eignet. Teile dieser Definition wurden im Rahmen der Abschlussarbeiten [Schnarr 2016] sowie [Wang 2017] entwickelt und prototypisch in einem HiL Prüfstand umgesetzt.

Gliedert man eine textuelle Testfallbeschreibung des *Warning generation tests* der [ISO 17361] zur Absicherung eines Spurhalteassistenten, so ergibt sich folgende Sequenz:

1. Die Wetterbedingungen, die Teststrecke und die Testfahrzeuge müssen den Vorgaben entsprechen.
2. Das Fahrzeug beschleunigt auf $v_{\text{longitudinal}} \in [20 \dots 22] \frac{\text{m}}{\text{s}}$ während es der Fahrspur der Teststrecke folgt.

3. Befindet sich das Fahrzeug in der Linkskurve der Teststrecke, so soll es mit einer Lateralgeschwindigkeit von $v_{\text{lateral}} \in (0 \dots 0,4] \frac{\text{m}}{\text{s}}$ die Kurve nach rechts außen verlassen.

Zusätzliche Randbedingungen definieren als Teststrecke eine flache, trockene und ausreichend lange Straße, deren Krümmung im Rahmen der vorgegebenen Kurvenradien liegt. Die Fahrbahnmarkierungen müssen in gutem Zustand sein und den nationalen Vorgaben entsprechen. Die Temperatur muss sich im Bereich von -20°C bis 40°C befinden und die Sichtweite mindestens einen Kilometer betragen.

Der statische Teil dieses Tests lässt sich mit Hilfe der bisher eingeführten Elemente der Szenariobeschreibung darstellen. Für die dynamischen Anteile ist ein allgemeines Schema notwendig. Als Basis-Element der Beschreibung wird das *Manöver* definiert. Ein Manöver beschreibt die kleinste Aktion eines dynamischen Akteurs innerhalb der Verkehrssituation. Im realen Fahrversuch entspricht dies einer Anweisung im Ablaufplan. Mögliche Manöver sind:

- Auf $v_{\text{longitudinal}} = 40 \frac{\text{km}}{\text{h}}$ beschleunigen
- Auf die linke Nachbarspur wechseln
- Zur Position (*s/t*) fahren
- Die Straße überqueren
- Auf ein Zielfahrzeug bis zum Abstand 10 m auffahren

Untersuchungen der Testfälle aus sechs ISO Normen in [Schnarr 2016, 43 ff.] zeigen, dass sich diese Tests mit zwei Manövern zur Beeinflussung der Längs- und Querrichtung des Fahrzeugs beschreiben lassen. Wie in Kapitel 2.3 dargestellt, werden nur wenige Arbeitspunkte durch die Testfälle der ISO Normen überprüft. Sollen mehrere Akteure, wie beispielsweise Fußgänger oder Fahrradfahrer, und komplexere Situationen unter Verkehrseinfluss betrachtet werden, so sind mehr als zwei Manöver notwendig. Neue Manöver lassen sich hinzufügen, sofern deren Instruktionen von realen Menschen und virtuellen Simulationsmodellen interpretiert werden können.

Bei den exemplarisch dargestellten Manövern wird deutlich, dass Größen teilweise undefiniert bleiben. So wird beispielsweise nicht näher definiert, mit welcher Beschleunigung die Zielgeschwindigkeit $v_{\text{longitudinal}}$ erreicht werden soll. Ist diese Angabe im Rahmen eines Tests dennoch notwendig, so kann das Manöver durch die zusätzliche Angabe einer Beschleunigung spezialisiert werden. Grundsätzlich sollten jedoch nur diejenigen Parameter Einzug in die Beschreibung finden, deren Definition das zu testende System beeinflusst.

Die sequentielle Abfolge von Manövern wird in *Steps* organisiert. Dabei kann ein *Step* mehrere parallelläufige Manöver beinhalten. Die Beschreibung der dynamischen Vorgänge eines Szenarios ist schließlich die Abfolge unterschiedlicher *Steps* und wird innerhalb des XML-Elements **Storyboard** beschrieben. Der dynamische Anteil der textuellen Testfallbeschreibung des oben aufgeführten Beispiels der [ISO 17361] kann mit Hilfe dieser Struktur wie folgt beschrieben werden:

```

<storyboard>
  <step actor="1">
    <maneuver type="accelerate" value="20" />
    <maneuver type="driveToLaneCenter" value="0" />
  </step>
  <step actor="1">
    <maneuver type="driveToPosition" s="120" />
  </step>
  <step actor="1">
    <maneuver type="accelerateLateral" value="0.2"
      ↪ />
  </step>
</storyboard>

```

Diese Struktur ermöglicht bereits eine Vielzahl unterschiedlicher Darstellungen, wobei die einzelnen Manöver nach Bedarf erweiterbar sind. Dennoch ist die Einführung eines Mechanismus zur Handhabung von Events notwendig. Der Versuch, folgendes Verkehrsszenario systematisch zu beschreiben, verdeutlicht diese Notwendigkeit: Für den Test eines Notbremsassistenten soll ein Ball auf die Straße geworfen werden, sobald das Testfahrzeug die Straßenkoordinate s_0 überfahren hat. Für die Beschreibung dieses dynamischen Vorgangs wäre es sinnvoll, die Ausführung des Manövers „Ball auf die Straße werfen“ so zu spezifizieren, dass es gestartet wird, *wenn* ein „Akteur s_0 überfahren hat“. Zur Realisierung eines Eventmechanismus wird eine Substruktur, die wiederum aus den Elementen Step und Manöver besteht, eingesetzt. Diese Substruktur wird im Folgenden als *Teilszenario* bezeichnet. Beim Eintritt eines vordefinierten Events wird ein Teilszenario ausgeführt.

Die praktische Umsetzung des Event-Mechanismus beinhaltet einige Details, auf die nachfolgend eingegangen wird. Einerseits stellt sich die Frage, wie mit einem momentan ablaufenden Step fortgefahren werden soll, wenn ein Teilszenario durch ein Event gestartet wurde. Ein Step kann abgebrochen oder fortgesetzt werden. Zudem ist die Definition einer Aufrufhäufigkeit notwendig,

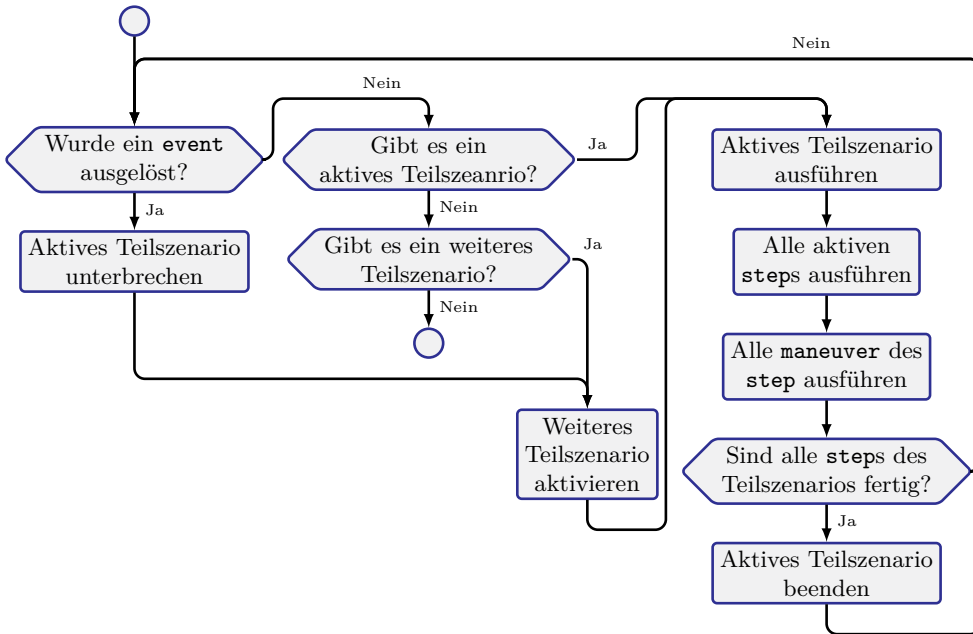


Abb. 3.11: Kontrollflussgraph zur Ausführung dynamischer Vorgänge

die beschreibt, ob ein Event einmalig oder mehrmalig eintritt. Wird beispielsweise ein Event durch das Überfahren der Straßenkoordinate s_0 ausgelöst, so beschreibt die Aufrufhäufigkeit, ob dieses Event nochmals ausgelöst wird, wenn s_0 erneut passiert wird.

Der Kontrollflussplan in Abbildung 3.11 visualisiert die Ausführung der dynamischen Vorgänge. In Kapitel 4.1 wird dieser Kontrollfluss für die Umsetzung der dynamischen Vorgänge im Rahmen der Simulationsumgebung verwendet. Das XML-Schema der beschriebenen Szenariobeschreibung ist im Anhang B zu finden. Ein exemplarisches Szenario-XML, das für ein Closed-Loop Testsystem eines Radar-basierten Notbremsassistenten verwendet wurde, ist in Anhang C aufgelistet.

3.3. Testfallgenerierung

Die vorgestellte Ontologie eignet sich als Grundlage für Testfälle, um Fahr-szenarien zu definieren. Hierfür ist die Wahl von Parameterwerten notwendig, um aus dem präsentierten Schema die Beschreibung konkreter Verkehrssituationen zu erhalten. Dies führt zu der Frage, wie sich die Parameter möglichst sinnvoll und geschickt wählen lassen. Ziel der Parameterwahl ist eine **hohe**

Testabdeckung bei geringer Anzahl an Testfällen.

Es gibt unterschiedliche Definitionen, was eine hohe Testabdeckung bedeutet. Gemein ist den Definitionen, dass eine Metrik zur Bewertung der Abdeckung gegeben sein muss. In [Liggesmeyer 2002, S.41 ff] wird das Vollständigkeitsverständnis der dynamischen Testverfahren von *strukturorientierten* und *funktionsorientierten* Tests verglichen. Bei strukturorientierten Verfahren kann eine Aussage zur Vollständigkeit hinsichtlich der zugrundeliegenden Softwarestruktur, wie beispielsweise Anweisungen oder Pfade, ermittelt werden. Bei funktionsorientierten Verfahren hingegen ist die Softwarestruktur unbekannt. Sie liefern eine Aussage zur Testabdeckung hinsichtlich der Spezifikation, garantieren aber keine vollständige Abdeckung der Programmstruktur.

Im Rahmen dieser Arbeit wird nicht der Quellcode, sondern ein Modell eines Assistenzsystems analysiert, so dass es sich nach [Liggesmeyer 2002, S. 40] dabei nicht um ein strukturorientiertes Verfahren¹⁸ im engeren Sinne handelt. Dennoch liegen der Analyse die gleichen Ansätze hinsichtlich der Testabdeckung zugrunde.

Die vorgestellte Ontologie beschreibt ein Modell realer Verkehrsszenarien. Bei der Formulierung von Anforderungen kann die Ontologie zur Beschreibung des gewünschten Verhaltens verwendet werden. Wird zusätzlich ein Modell des Assistenzsystems eingeführt, so steht eine abstrakte Beschreibung des Gesamtsystems Fahrzeug und Umgebung zur Verfügung. Dies erlaubt eine Testfallerstellung anhand des Gesamtmodells und entspricht dem Ansatz des Modellbasierten Testens (MBT).

Nach [Liggesmeyer 2002] stellen modellbasierte Tests keine eigene Klasse der Softwaretests dar, sondern bedienen sich grundlegenden Testtechniken und wenden diese auf ein Modell an. Basierend auf den Systemanforderungen wird ein Modell erstellt, um damit Testfälle auszuwählen oder zu generieren. Für die Tests wird einerseits das zu erwartende Systemverhalten y_{erw} (Testorakel) erstellt. Andererseits werden durch eine Konkretisierung der Testfälle die Testskripte abgeleitet. Die Testskripte werden schließlich auf dem Testobjekt ausgeführt. Aus dem Vergleich zwischen Systemausgaben und erwartetem Systemverhalten ergibt sich das Testergebnis. Abbildung 3.12 visualisiert den beschriebenen Ablauf des modellbasierten Testens. Dieses Schema lässt sich auf die Elemente der Absicherungsstrategie übertragen. Der Szenarienkatalog beinhaltet dabei die Testfälle. Aufgrund der eingeführten formalen Beschreibung der Verkehrssituationen ist keine spezielle Konkretisierung oder Implementierung in Testskripte notwendig. Die Szenarien dienen selbst als Testskripte. Die Testausführung auf dem Testobjekt erfolgt in realen oder virtuellen Testfahrten.

¹⁸[Liggesmeyer 2002, S. 83] definiert strukturorientierte Testtechniken als Ansätze, die „die Vollständigkeit des Tests anhand der Abdeckung des Software-Quellcodes beurteilen“.

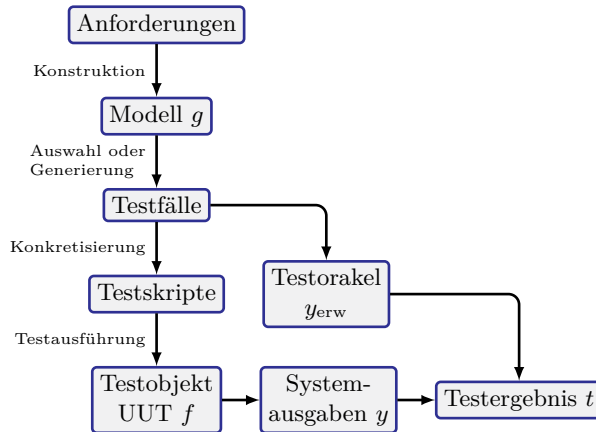


Abb. 3.12: Prinzip des modellbasierten Testens basierend auf [Liggesmeyer 2002, S. 216]

Das Testergebnis ergibt sich ebenfalls aus dem Vergleich einer zu erwartenden Systemreaktion und den Systemausgaben während der Testausführung. Werden die Szenarien anhand von Anforderungen oder einem Modell erstellt, so fällt dies in die Kategorie *Gray-Box* (Vgl. Abschnitt 3.1-b).

Zusätzlich zu dem in Abbildung 3.12 dargestellten Ablauf ist es bei der Testgenerierung für Assistenzsysteme notwendig, dass ein Gesamtmodell der UUT sowie der Umgebung vorhanden ist. Grund hierfür sind die gegenseitigen Wechselwirkungen zwischen Fahrzeug und Umgebung.

Nachfolgend wird eine Formalisierung der Beschreibung des Vorgehens zur Testfallerstellung im Hinblick auf Assistenzsysteme eingeführt. Darauf aufbauend wird die Testauswertung näher beleuchtet. Zusätzlich wird die Verwendung von Äquivalenzklassen bei der Modellbildung motiviert, was auf ähnliche Umstände wie in Kapitel 3.1 zurückzuführen ist. Teile hieraus wurden im Rahmen der Beiträge [Feilhauer 2016c] und [Habiger 2018] publiziert.

Ein zu testendes System f (die Unit Under Test) wird durch seine Eingänge x und Ausgänge $y = f(x)$ charakterisiert. Ein Test $t(x, y, y_{erw}) \in \{\text{erfolgreich, fehlgeschlagen}\}$ ist eine Funktion, die bei gegebenen Eingängen x die Systemausgaben y mit der erwarteten Reaktion y_{erw} vergleicht und dem Ergebnis erfolgreich oder fehlgeschlagen zuordnet.

$$t(x, y, y_{erw}) = \begin{cases} \text{erfolgreich} & \text{wenn } y = y_{erw} \\ \text{fehlgeschlagen} & \text{wenn } y \neq y_{erw} \end{cases}$$

Das System f kann zu komplex sein, um darauf direkt Testfälle abzuleiten. Hierfür wird deshalb ein Gray-Box-Modell g eingeführt, welches f soweit

abstrahiert, dass eine Ableitung von Testfällen möglich wird. Im Grenzfall gilt $g = f$. Ziel des Gray-Box-Modells ist die Generierung von Testparametern x_{ziel} , die sich als Eingangswerte zum Test von f eignen $x = x_{\text{ziel}}$. Hierfür ist die Invertierung des Modells nötig:

$$x_{\text{ziel}} = g^{-1}(y_{\text{erw}})$$

Da es sich bei g nur um ein Modell von f handelt, muss sich das System f und das Modell g bei gegebenen x_{ziel} nicht exakt gleich verhalten. Dies führt zur Unsicherheit, ob f sich während eines Tests verhält wie vom Modell g prognostiziert, und muss bei der Testdurchführung beachtet werden.

Im Hinblick auf die Testfallgenerierung von Assistenzsystemen ist der Einfluss der Umgebung zu berücksichtigen. Zudem kann es sein, dass eine direkte Stimulation des Systems f nicht möglich ist. Das System f ist also in eine Umgebung e eingebettet.

$$\begin{aligned} x &= e(x') \\ y &= f(e(x')) \end{aligned}$$

Nun müssen die Eingangswerte x' gefunden werden, mit denen das Gesamtsystem, bestehend aus f und e , getestet werden kann. Auch hierfür kann ebenfalls eine Abstraktion verwendet werden. Es wird das Umgebungsmodell h eingeführt, um eine Invertierung von e zu ermöglichen.

$$\begin{aligned} x &= h(x') \\ y &= g(h(x')) \end{aligned}$$

Durch die Invertierung des Umgebungsmodells ergeben sich hierfür die Testeingangsparameter.

$$\begin{aligned} x'_{\text{ziel}} &= h^{-1}(x_{\text{ziel}}) \\ &= h^{-1}(g^{-1}(y_{\text{erw}})) \end{aligned}$$

Zusätzlich ist das dynamische Verhalten der Umgebung zu beachten. Können die Eingänge x' zum Zeitpunkt $t = 0$ angegeben werden, so ergibt sich die zeitliche Abhängigkeit der Ausgangsparameter zu

$$x(t) = e(x', t)$$

Diese Ausgangsparameter des Umgebungsmodells sind die Eingänge der UUT

$$\begin{aligned} y(t) &= f(x(t)) \\ &= f(e(x', t)) \end{aligned}$$

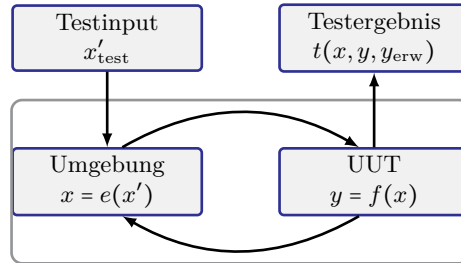


Abb. 3.13: Schematische Visualisierung eines Closed-Loop Tests

Zur Testparametergenerierung muss das zeitabhängige Modell der Umgebung invertiert werden. Sollen die Eingänge der UUT f zu einem Zielzeitpunkt t_{ziel} erreicht werden, so ergeben sich die Eingangsparameter

$$x'_{\text{test}} = h^{-1}(x_{\text{ziel}}, t_{\text{ziel}})$$

Werden die so generierten Testparameter x'_{ziel} verwendet, so sollte die erwartete Systemreaktion y_{erw} etwa zum Zeitpunkt t_{ziel} eintreten.

$$\begin{aligned} y &= f(e(x')) \\ y(t) &= f(e(x'_{\text{test}}, t)) \\ y(t_{\text{ziel}}) &= f(e(x'_{\text{test}}, t_{\text{ziel}})) \end{aligned}$$

Dies beschreibt die Anwendung der modellbasierten Testfallgenerierung für Closed-Loop Testsysteme. Abbildung 3.13 visualisiert schematisch die Verwendung der generierten Testdaten für einen Closed-Loop Testaufbau, sowie die anschließende Testauswertung. Bei der Testauswertung ist zu beachten, dass ein Test nur als *bestanden* oder *fehlgeschlagen* eingestuft werden darf, wenn auch die Eingangsparameter x in die UUT f betrachtet werden. Ansonsten ist nicht sichergestellt, ob die gewünschten Eingangsparameter x überhaupt erreicht wurden. Damit ist auch die Auswertung der Systemreaktion $y = f(x)$ hinfällig. Folgendes simples Beispiel veranschaulicht diese Notwendigkeit: Die Testeingangsparameter x' zum Test eines Notbremsassistenten würden ein Szenario beschreiben, bei dem ein Fahrzeug mit $v = 10 \frac{\text{m}}{\text{s}}$ auf ein 100 Meter entfernt stehendes Fahrzeug auffährt. Die zu erwartende Systemreaktion wäre, dass nach etwa 10 Sekunden eine Notbremsung ausgelöst wurde. Das Ergebnis einer Simulation sei, dass keine Notbremsung ausgelöst wurde. Ohne x zu betrachten kann nicht gesagt werden, ob der Test *bestanden* wurde. Würde sich herausstellen, dass das Zielfahrzeug mit $v = 15 \frac{\text{m}}{\text{s}}$ vom Testfahrzeug weggefahren ist, so erfolgte richtigerweise keine Notbremsung. Seien alternativ die Umgebungsparameter korrekt gewesen und es ereignete sich ein Unfall, so wäre der Test *fehlgeschlagen*.

a. Vorgehen der Modellbildung und Invertierung

In der praktischen Anwendung besteht das Umgebungsmodell meist aus mehreren Teilen. Während die UUT eventuell als einzelnes Modul betrachtet werden kann, so ist eine sequentielle Invertierung aller Teile des Umgebungsmodells nötig. Folgender Ablauf beschreibt das Vorgehen zur systematischen Generierung von Szenarien zum Test eines Systems f :

1. Das System f muss dahingehend analysiert werden, dass eine Beschreibung der relevanten Eingangsdimensionen d vorliegt.
2. Die identifizierten Dimensionen d sind in Äquivalenzklassen c_i^d zu unterteilen. Eine Äquivalenzklasse repräsentiert denjenigen Eingabebereich, innerhalb dessen sich das System für alle Repräsentanten der Äquivalenzklasse gleich verhält. Somit steht anschließend eine Abstraktion g des Systems f zur Verfügung, das aus der Beschreibung der Eingangsdimensionen d und den Äquivalenzklassen c_i^d besteht.
3. Die zu erreichenden Zielzustände x_{ziel} zum Test von f ergeben sich aus der Kombination von Repräsentanten der Äquivalenzklassen. Zur Wahl der Repräsentanten stehen unterschiedliche Optionen zur Verfügung: Ein einzelner Repräsentant kann aus der Mitte der Äquivalenzklasse gewählt werden. Eine im Rahmen des Softwaretests häufig angewandte Auswahl ist die Verwendung von Randwerten. Steht eine Statistik zur Verfügung, beispielsweise typische Fahrgeschwindigkeiten auf Autobahnen, so kann diese zur Wahl der Repräsentanten herangezogen werden. Abbildung 3.14 visualisiert unterschiedliche Auswahlmöglichkeiten. Werden m Repräsentanten je Äquivalenzklasse gewählt, so ergibt sich die Anzahl der Testfälle n_{test} für d Dimensionen aus der Mächtigkeit der Äquivalenzklassen $|c^d|$ zu

$$n_{\text{test}} = \prod_d (m \cdot |c^d|)$$

4. Abschließend ist die Transformation der Zielzustände x_{ziel} in Parameter der Szenariobeschreibung x'_{test} notwendig. Eine Option hierfür ist die Invertierung des Umgebungsmodells. Eine solche Invertierung kann sich je nach Modell aufwändig gestalten und lässt sich nicht immer analytisch lösen. Alternativ können Parameter der Szenariobeschreibung x'_{test} mit Hilfe eines Optimierungsverfahrens gesucht werden. In [Sattler 2015] wird eine solche Parameterbestimmung mittels evolutionärer Algorithmen realisiert. Dabei werden die Parameter des Umgebungsmodells solange

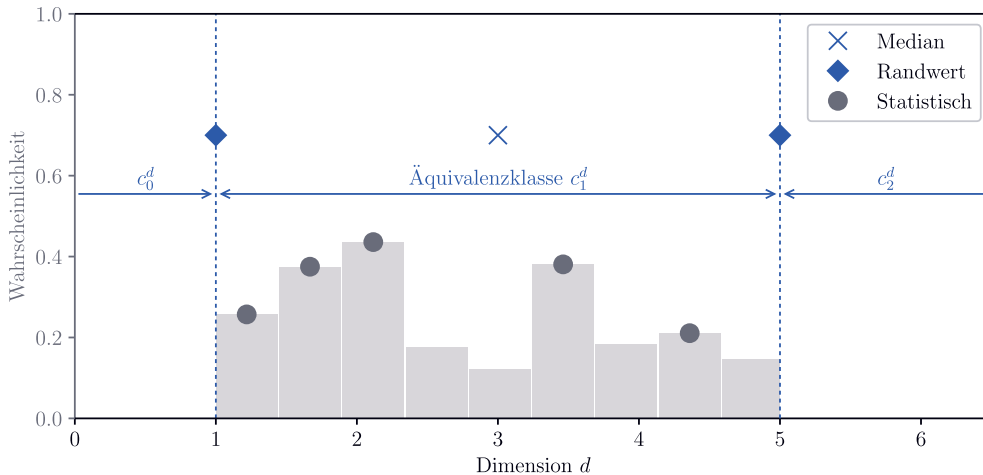


Abb. 3.14: Auswahl von Repräsentanten einer Äquivalenzklasse

variiert, bis sich der gewünschte Zielzustand während der Simulation ausführt. Hierfür sind mehrere Simulationen durchzuführen, um die Parameter zu ermitteln.

Das Ergebnis dieses Vorgehens sind n_{test} Testfälle, die sich zum Erreichen aller Zielzustände eignen, welche sich aus der Äquivalenzklassen-basierten Analyse von f ergeben haben. Abbildung 3.15 visualisiert das beschriebene Vorgehen. Eine exemplarische Anwendung sowie eine darauf basierende Bewertung des modellbasierten Ansatzes sind in Kapitel 5.1 aufgeführt.

Zentraler Bestandteil der präsentierten modellbasierten Testfallgenerierung stellt die notwendige Modellbildung dar. Dies birgt das Risiko, dass Abstraktionen lückenhaft sind und deshalb Testfälle für das reale System nicht generiert werden. Zufallsbasierte Ansätze erfordern aufgrund des fehlenden Systemwissens einen hohen Testaufwand, reduzieren aber das Risiko Testfälle wegen

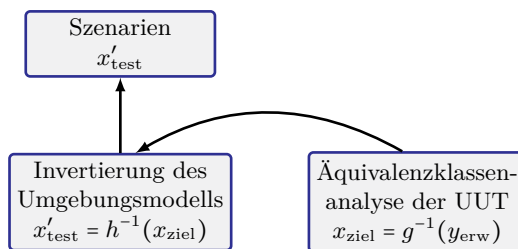


Abb. 3.15: Vorgehen zur systematischen Testfallgenerierung

falscher Modellbildung zu vernachlässigen. Folgender Abschnitt skizziert die probabilistische Testfallgenerierung für Assistenzsysteme.

b. Probabilistische Testfallgenerierung

Um die Gefahr fehlender Testfälle aufgrund unzulässiger Modellvereinfachungen zu reduzieren, kann eine probabilistische Testfallgenerierung in Betracht gezogen werden. Hierbei werden Testparameter unter Verwendung möglichst weniger Annahmen über das zu testende System zufällig ausgewählt. So ergeben sich eventuell Parameterkonstellationen, an die man nicht direkt gedacht hätte, beziehungsweise die sich anhand eines systematischen Vorgehens nicht ergeben hätten.

Ohne jegliche Abstraktion realer Verkehrsszenarien steht lediglich die Option realer Dauerläufe zur Verfügung. Bereits die vorgestellte Szenariobeschreibung abstrahiert das reale Verkehrsgeschehen. Da diese Beschreibung jedoch als Testinput für virtuelle Closed-Loop Tests von Assistenzsystemen dient, wird sie im Folgenden als Minimalanforderung zur Testausführung verstanden.

Eine erste Methodik der zufallsbasierten Testfallgenerierung könnte die Variation aller Parameter der Szenariobeschreibung darstellen. Bei der Umsetzung dieser Methodik zeigt sich, dass der Großteil generierter Szenarien unplausibel ist: Fahrzeuge sind in der Luft platziert, überschneiden sich stark mit anderen Objekten und würden beim Abfahren zufälliger Trajektorien nur sehr selten anderen Fahrzeugen begegnen. Dieses Vorgehen ist zwar grundsätzlich möglich, der Anteil *unsinniger* Szenarien ist dabei jedoch sehr hoch. Deshalb sind Einschränkungen bei der zufälligen Variation der Parameter nötig. Diese Limitierungen müssen jedoch so gewählt werden, dass dabei getroffene Annahmen möglichst nur unplausible Szenarien aussortieren. Der Vorteil des probabilistischen gegenüber dem modellbasierten Ansatz bleibt solange erhalten, wie die dabei angewandten Einschränkungen zur Parametervariation weniger limitierend sind als die Annahmen der Modellbildung.

Eine Einschränkung für die initiale Parametrierung eines Szenarios könnte keine kollidierenden Objekte erlauben. Als zusätzliche Randbedingung ließe sich einführen, dass Fahrzeuge stets auf einem Untergrund platziert werden müssen. Verallgemeinert lässt sich festhalten, dass für eine praxistaugliche Umsetzung der probabilistischen Testfallgenerierung für Assistenzsysteme ein *Limitierungsmodell* notwendig ist. Somit findet die zufällige Auswahl der Parameter nicht direkt auf dem Schema der Szenariobeschreibung statt, sondern auf dem *Limitierungsmodell*. Eine Anwendung sowie die Bewertung eines probabilistischen Ansatzes sind in Kapitel 5.1 beschrieben.

Zusammenfassung

Die Umsetzung abgesicherter Assistenzfunktionen und autonomer Fahrzeuge stellt wegen der softwarebasierten Perzeption und Kognition der Verkehrssituation eine große Herausforderung dar. Aufgrund des Umstands der funktionalen Unzulänglichkeit ist eine Vorab-Definition aller Eventualitäten nicht möglich. Um dennoch die Entwicklung der Assistenzfunktionen möglichst sicher und transparent zu realisieren, wurde in diesem Kapitel eine Absicherungsstrategie entworfen. Der iterative Ansatz berücksichtigt hierbei nachhaltig die Tatsache der unvollständigen Spezifikation und liefert ein ganzheitliches Bild, um sowohl reale als auch virtuelle Testfahrten einzubeziehen. Für das zentrale Element des Szenarienkatalogs wurde zudem eine Ontologie zur Beschreibung von Verkehrsszenarien präsentiert. Die abschließenden Ansätze zur Testfallgenerierung bieten eine Möglichkeit zur Erweiterung des Szenarienkatalogs.

Virtuelle Testfahrten sind bereits in der Absicherungsstrategie ein wichtiges Element. Grund hierfür ist unter anderem der Wunsch, eine möglichst hohe Testabdeckung zu realisieren. Im Rahmen von virtuellen Testfahrten lässt sich die Variation von Parametern leicht umsetzen und die Parallelisierung einzelner Simulationen ist möglich. Deshalb stellt die Simulation von Testfahrten eine Option zur Effizienzsteigerung dar. Folgendes Kapitel adressiert detailliert Aspekte, die zur Umsetzung virtueller Fahrversuche nötig sind.

4. Simulation von Fahrerassistenzsystemen

Simulationen werden in unterschiedlichen Stufen der Automobilentwicklung eingesetzt. Beispiele hierfür sind die dreidimensionale Strömungssimulation (engl. Computational Fluid Dynamics, CFD), die Simulation von Bauteileigenschaften und virtuelle Schwingungsanalysen der Fahrzeugkarosserie. Durch Anwendung und Kombination dieser Ansätze lässt sich die Fahrzeugentwicklung teilweise virtualisieren. So können teure Prototypen eingespart, Iterationszyklen während der Entwicklung verkürzt oder physikalische Vorgänge visualisiert werden. Auch bei der Entwicklung und Absicherung von Assistenzsystemen ist der Einsatz von Simulation sinnvoll. Können Teile des Gesamtsystems in Simulationsumgebungen anstatt in realen Probefahrten getestet werden, so lassen sich die aufwändigen Testfahrten reduzieren und die Iterationszyklen der Entwicklung autonomer Fahrzeuge verkürzen.

In diesem Kapitel werden unterschiedliche Perspektiven erörtert, um diejenigen Elemente zu identifizieren, die eine Simulation von Fahrerassistenzsystemen ermöglichen. Hierfür wird zu Beginn in Kapitel 4.1 eine Softwarearchitektur erarbeitet, die eine Kopplung von Modellen unterschiedlicher Domänen erlaubt und sich zur Simulation von Assistenzsystemen mit geschlossenen Regelkreisen eignet. Dabei wird auf die Kompatibilität zu Model-in-the-Loop, Software-in-the-Loop und Hardware-in-the-Loop Testverfahren geachtet, die in der Automobilindustrie entlang des V-Entwicklungszyklus Anwendung finden. Diese werden auch zukünftig zur Beschleunigung der Entwicklungen mittels Frontloading¹⁹ eine wichtige Rolle spielen.

Da Fahrerassistenzsysteme unterschiedliche Sensoren verwenden, ist eine konsistente Simulation der gesamten Umfeldsensorik notwendig. Hierfür werden in Kapitel 4.3 und Kapitel 4.4 zwei Modellierungsansätze für Sensormodelle vorgestellt. Die Ergebnisse der Modellvalidierung werden in Kapitel 5 präsentiert.

Auszüge dieses Kapitels wurden in [Feilhauer 2016a], [Feilhauer 2016b] sowie in [Feilhauer 2017a] publiziert.

¹⁹Frontloading beschreibt das Verlagern von Entwicklungsaufgaben in frühe Phasen des Entwicklungsprozesses, um eine frühzeitige Validierung getroffener Annahmen zu ermöglichen [Paulweber 2014, S.4 f.]

4.1. Simulationsarchitektur

Um das Gesamtfahrzeug inklusive Assistenzsystem sowie dessen Fahrzeugumgebung zu simulieren, sind verschiedene Modelle notwendig. Zur Strukturierung dieser heterogenen Modelle werden im Folgenden drei unterschiedliche Sichten präsentiert: Eine abstrahierte Sicht auf Verkehrsszenarien (Anwendersicht), eine daraus abgeleitete Sicht der involvierten Modelle (Kosimulationsstruktur) sowie eine detaillierte Präsentation der Struktur des Umgebungsmodells (Softwarestruktur des Umgebungsmodells).

a. Die Anwendersicht

Um einen prinzipiellen Überblick über mögliche Einflussfaktoren zu erhalten, ist in Abbildung 4.1 schematisch eine Verkehrsszene aus der Vogelperspektive visualisiert. Im linken Teil der Grafik ist ein Fahrzeug zu sehen, das durch

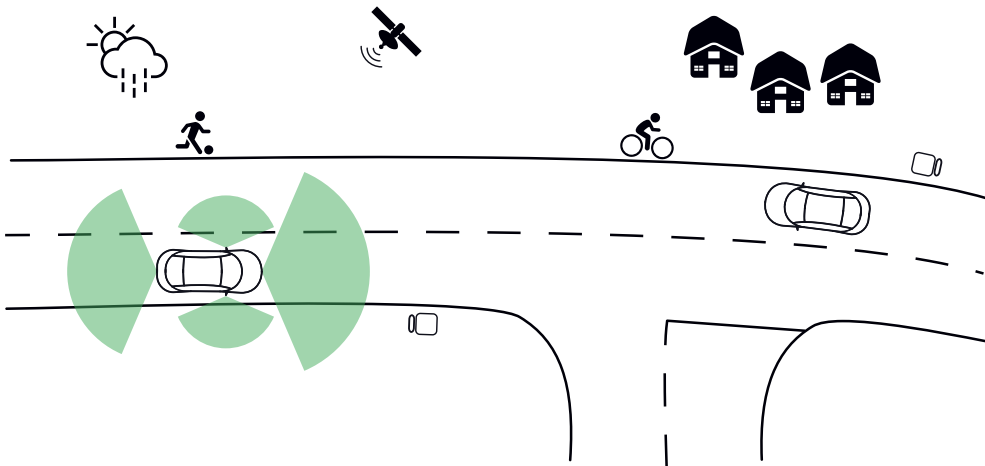


Abb. 4.1: Schematische Visualisierung einer Verkehrsszene

unterschiedliche Sensoren seine Umgebung erfasst. Für Assistenzsysteme werden zur Umfelderkennung vorrangig Videokameras, Ultraschall-, Radar- und Lidar-Sensoren verwendet. Neben den intrinsischen Sensoren des Fahrzeugs ist eventuell zusätzlich eine Kommunikation zu einem Positionierungssystem, anderen Verkehrsteilnehmern oder der Infrastruktur (engl. Car-to-X Communication, Car2X) vorhanden. Die Umgebung setzt sich aus der Straße, anderen Verkehrsteilnehmern und Fußgängern, Verkehrsschildern und Signalanlagen sowie Gebäuden und Randbebauung zusammen. Angedeutet ist außerdem der Einfluss des Wetters, was unterschiedliche Sichtbedingungen und verschiedene Straßeneigenschaften zur Folge hat.

Betrachtet man die Funktionsweise von Assistenzsystemen, so lässt sich folgende Wirkkette erkennen: Die Fahrzeugumgebung wird durch die Sinne des Fahrers und die parallel hierzu arbeitende Sensorik des Fahrzeugs erfasst (Perzeption). Anschließend schätzen der Fahrer und der Algorithmus des Assistenzsystems die Verkehrssituation ein (Kognition), um darauf basierend über Aktuatoren des Fahrzeugs dessen Trajektorie zu beeinflussen (Aktion) [Vgl. Donges 1982]. Zusätzlich zur direkten Beeinflussung der Fahrzeugbewegung geben Assistenzsysteme über die Mensch-Maschine-Schnittstelle des Fahrzeugs (engl. Human-Machine Interface, HMI) eine Rückmeldung an den Fahrer. Diese Wirkkette ist in Abbildung 4.2 visualisiert. Für den Transfer der Elemente dieser

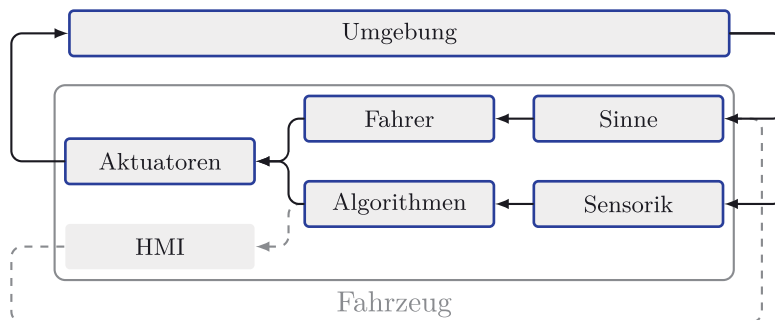


Abb. 4.2: Wirkkette von Fahrerassistenzsystemen

Wirkkette in eine Simulationsumgebung sind Modelle notwendig. Die Diversität der involvierten Komponenten ist groß und reicht von dreidimensionalen Modellen der Objektgeometrien, über Verhaltensmodelle der Verkehrsteilnehmer hin zu physikalischen Fahrdynamikmodellen. Zur Modellierung der unterschiedlichen Komponenten sind spezialisierte Werkzeuge vorhanden. Für die Simulation der dargestellten Wirkkette müssen diese Komponenten aus unterschiedlichen Domänen zu einer Gesamtsimulation integriert werden. Dies erfordert die im folgenden Abschnitt vorgestellte Kosimulationsstruktur.

b. Die Kosimulationsstruktur

Eine Simulationsumgebung für Assistenzsysteme muss die Einbindung und Interaktion unterschiedlicher Simulationsmodelle ermöglichen. Die Modelle unterscheiden sich teilweise sehr stark voneinander, sodass eine gemeinsame Simulation auch als Kopplung unterschiedlicher *Simulations-Domänen* bezeichnet wird. Die technische Heterogenität der Modelle resultiert aus der Tatsache, dass es für unterschiedliche Bereiche etablierte Anwendungen zur Modellierung gibt. So sind [3DS MAX] und [Blender] spezialisierte Anwendungen, die sich zur Mo-

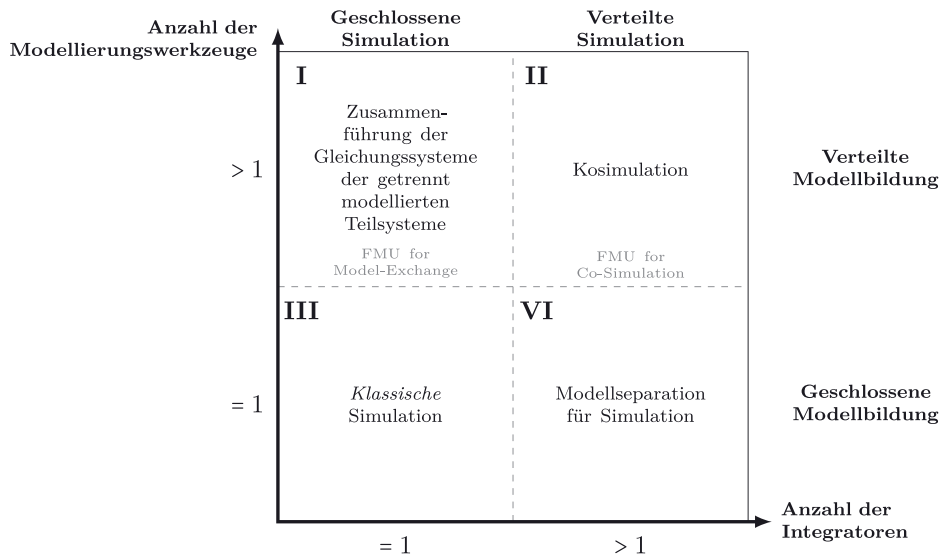


Abb. 4.3: Modellbildungsvarianten nach [Geimer 2006]

dellierung dreidimensionaler Objekte eignen. [Simulink] sowie [Modelica] sind hingegen Anwendungen, die eine grafische Modellierung physikalischer Systeme ermöglichen. Für die Simulation von Assistenzsystemen werden Resultate dieser Programme benötigt. Die Verwendung spezialisierter Anwendungen erlaubt die effiziente Umsetzung einzelner Modellteile. Deshalb haben die Programme dieser unterschiedlichen Domänen auch weiterhin ihre Berechtigung. Dies führt dazu, dass die Simulation von Fahrerassistenzsystemen eine Kosimulationsaufgabe darstellt: Unterschiedliche Modelle, die eventuell eigene Gleichungslöser (engl. Solver) verwenden, müssen zu einer Gesamtsimulation integriert werden.

In [Geimer 2006] wird die in Abbildung 4.3 dargestellte Begriffsvereinheitlichung für die unterschiedlichen Kombinationen von Modellbildungsansätzen vorgeschlagen. Diese Nomenklatur wird auch im Rahmen dieser Arbeit verwendet. Für eine Simulation von Assistenzsystemen sind Modelle aus unterschiedlichen Modellierungswerkzeugen notwendig, die teilweise eigene Integratoren verwenden. Deshalb ordnet sich der Aufbau einer Simulation für Assistenzsysteme nach [Geimer 2006] in die Kategorie *Kosimulation* (Quadrant **II** in Abbildung 4.3) ein.

Die folgenden Modelle einer exemplarischen Gesamtsimulation verdeutlichen diese Einordnung: Für den Antriebsstrang eines Fahrzeugs würden aus früheren Entwicklungen Motor- und Getriebemodelle existieren. Diese seien durch

einen datenbasierten Ansatz²⁰ erstellt. Aus der Entwicklung des Elektronischen Stabilitätsprogramms (ESP) stünde ein physikalisch-modelliertes Fahrdynamikmodell zur Verfügung. Für die detaillierte Untersuchung des Bremswegs sei ein FEM²¹-basiertes Reifenmodell vorhanden. Zur Repräsentation der Umgebung stünde eine Programmbibliothek der Spieleindustrie zur Verfügung, die sich zur effizienten Darstellung dreidimensionaler Objekte eignet. Straßeninformationen, wie die Anzahl der Spuren, seien durch ein separates Straßenmodell abrufbar. Auf dieses Modell der Straße wiederum greife ein Fahrermodell zu. Ein Algorithmus zur Objekterkennung sei in einer hierfür entwickelten Umgebung (beispielsweise in dem Tool Automotive Data and Time-Triggered Framework, ADTF) realisiert. Tabelle 4.1 listet diese exemplarischen Komponenten auf und ordnet sie unterschiedlichen Simulationsdomänen zu.

Simulations- domäne	Modell			
	Gesamtfahrzeug	Fußgänger	Ampel	Straße
Physikalisch	Fahrdynamik			Pfad
Datenbasiert	Antriebsstrang			
FEM	Reifen			
3D Daten	Sensor	Geometrie	Geometrie	Geometrie
	Geometrie			
Filter	Objekterkennung			
Quellcode	Fahrer	Verhalten	Schaltzeiten	Pfad

Tabelle 4.1: Modelle einer exemplarischen ADAS Simulation sowie dazugehörige Simulationsdomänen

Solche Modelle werden von unterschiedlichen Parteien der Automobilindustrie erstellt. Dies erfordert unternehmensintern einen abteilungsübergreifenden Austausch, teilweise auch einen Austausch über Unternehmensgrenzen hinweg. Hierbei muss einerseits der Know-how-Schutz gewährleistet werden, andererseits die technischen Schnittstellen für die Integration der unterschiedlichen Modelle detailliert genug offengelegt werden. Eine Realisierung des Modellaustauschs

²⁰Hierbei dienen Messdaten als Basis für eine Funktionsannäherung. Solche Modelle bieten durch deren analytische Beschreibung eine hohe Performance.

²¹Finite-Element-Lösers

kann beispielsweise mit Hilfe des Standards Functional Mock-up Interface (FMI) realisiert werden [Vgl. Modelisar 2014].

FMI ist ein standardisiertes Interface zur Lösung zeitlich-gekoppelter Systeme, die sich aus zeit-kontinuierlichen und zeit-diskreten Subsystemen zusammensetzen [Arnold 2013]. Dies ermöglicht die Realisierung einer Integrationsplattform, um mehrere Functional Mock-up Units (FMUs) zu einer Gesamtsimulation zu verbinden. Die einzelnen FMUs können dabei entweder eigene Solver implementieren (FMI for Co-Simulation), die durch die Integrationsplattform angesteuert werden, oder sie machen die Gleichungen der Modelle zugänglich, um durch einen externen Gleichungslöser der Integrationsplattform gelöst zu werden (FMI for Model Exchange). Im folgenden Abschnitt wird der Einfluss unterschiedlicher Gleichungslöser bei der Kopplung von Simulationsmodellen betrachtet.

Numerische Aspekte bei der Kopplung von Simulationsmodulen

Auch in aktuellen Computerspielen werden komplexe virtuelle Welten visualisiert, die teilweise auch die oben angesprochenen Modelle beinhalten. Hierbei wird jedoch primär ein Einschnitt-Euler-Verfahren zur Berechnung der Modelle verwendet. Auf Grund kurzer Schrittweiten ist dieses Verfahren ausreichend, um ein flüssiges und realistisch wirkendes Spielverhalten zu realisieren. Folgendes hybrides²² Beispiel der Simulation eines Notbremsassistenten vergleicht die Verwendung von Gleichungslösern mit fester oder variabler Schrittweite.

Für einen Testaufbau zur funktionalen Absicherung eines Radar-basierten Notbremsassistenten sind folgende Modelle durch eine Integrationsplattform zu einer Gesamtsimulation verbunden:

- Zwei Fahrdynamikmodelle zur Berechnung der Longitudinal- und Lateralbewegung der Fahrzeuge
- Ein Modell des Radarsensors, das basierend auf dreidimensionalen Repräsentationen der Fahrzeuge und der Umgebung das Sensorsignal nachbildet
- Ein Objekterkennungsalgorithmus, der basierend auf dem Sensorsignal eine Objektliste generiert, um die Objekte des Umfelds semantisch zu beschreiben
- Der zu testende Algorithmus des Notbremsassistenten. Basierend auf der Objektliste schätzt dieser die aktuelle Verkehrssituation ein, um eventuell eine Bremsanforderung an das Fahrzeug zu senden. Die Funktionsspezifikation besagt, dass bei einer voraussichtlichen Zeitdauer bis zur Kollision (engl.

²²Hybrid wird hier im Sinne der Kombination zeitkontinuierlicher sowie zeitdiskreter Systeme verstanden.

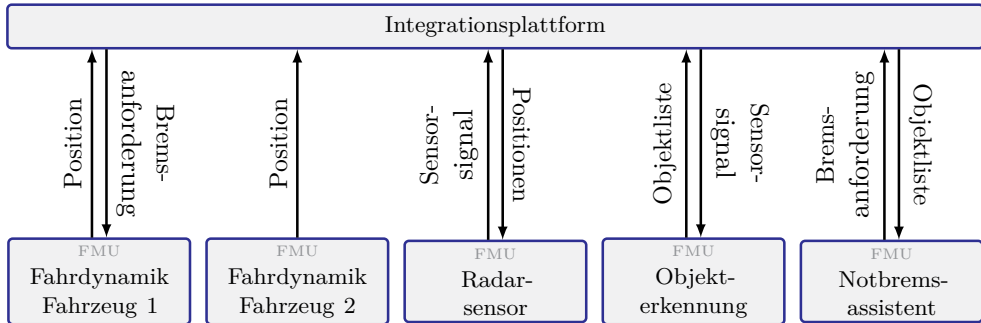


Abb. 4.4: Exemplarische Kosimulationsstruktur eines Notbremsassistenten

Time To Collision) $TTC \leq 1$ Sekunde die Bremsanforderung gesendet werden muss.

All diese Modelle stehen als FMU zur Verfügung und können sowohl durch Modell-interne Gleichungslöser (Kosimulation) als auch durch Modell-externe Gleichungslöser (Model-Exchange) betrieben werden. Abbildung 4.4 visualisiert die Systemsicht dieses Testaufbaus sowie vereinfacht die dazugehörigen Ein- und Ausgangssignale. Eine solche Gesamtsimulation wurde im Rahmen dieser Arbeit realisiert und in zwei unterschiedlichen Einstellungen ausgeführt: Zum einen mittels fester Berechnungsschrittweite der Gesamtsimulation von $\Delta t = 0,1$ Sekunden und zum anderen mittels variabler Berechnungsschrittweite. Abbildung 4.5 zeigt den Vergleich beider Simulationen. Im ersten Diagramm ist die berechnete voraussichtliche Kollisionszeit (TTC) über der Simulationszeit für die variable und feste Berechnungsschrittweiten aufgetragen. Während bei der Simulation mit variabler Schrittweite die Bremsanforderung bei $TTC = 1$ Sekunde erfolgt, erfolgt diese Berechnung bei fester Schrittweite erst bei einer Simulationszeit von $t = 3,8$ Sekunden, was hier einem TTC von etwa 0,92 Sekunden entspricht. Wie auch im zweiten Diagramm zu erkennen ist, erfolgt die Bremsanforderung im Fall fester Berechnungsschrittweiten später. Der Nutzen der höheren Genauigkeit im Fall der variablen Schrittweite erfolgt jedoch auf Kosten erhöhter Berechnungsdauer, was in den Diagrammen 3 und 4 deutlich wird. Der variable Schrittweiten-Löser nähert im Simulationszeitraum $t \in [3,65 \dots 3,85]$ s das Event der Anforderungsauslösung iterativ an, was mit einem erhöhten Berechnungsaufwand einhergeht. Dahingegen bleibt der Berechnungsaufwand bei fester Schrittweite konstant.

Dieses Beispiel verdeutlicht die Unterschiede der beiden Berechnungsansätze, wobei beide Methoden ihren Anwendungsbereich haben: Da bei der Auslegung von Systemeigenschaften die Anforderung an die Genauigkeit über der Berechnungsperformance steht, erscheint der Einsatz von Gleichungslösern

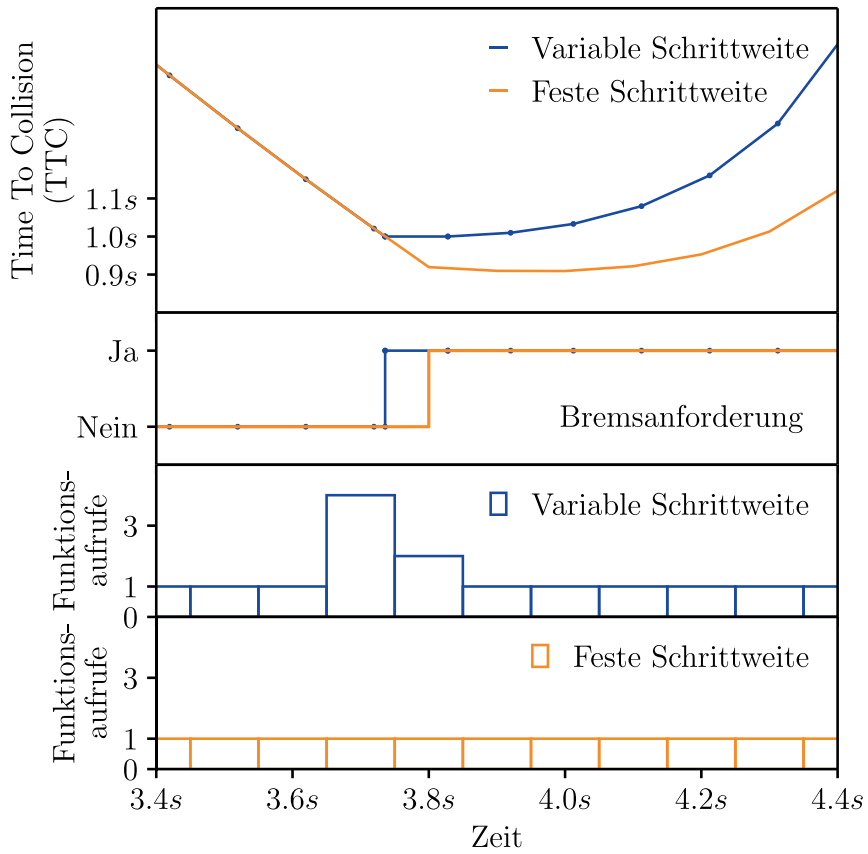


Abb. 4.5: Vergleich variabler und fester Berechnungsschrittweiten am Beispiel eines Notbremsassistenten

mit variabler Berechnungsschrittweite etwa in Model-in-the-Loop (MiL) oder Software-in-the-Loop (SiL) Test- und Entwicklungsumgebungen sinnvoll. Werden reale Prototypen in späteren Stufen des Entwicklungsprozesses getestet, ist ein gleichbleibender Berechnungsaufwand in Hardware-in-the-Loop (HiL) Umgebungen notwendig, um die harten Echtzeitanforderungen der realen Systeme zu erfüllen.

Nach diesem Exkurs über numerische Aspekte bei der Kopplung von Simulationsmodellen wird nachfolgend ein Entwurf für die Softwarestruktur des Umgebungsmodells erarbeitet. Das Umgebungsmodell nimmt eine Sonderstellung hinsichtlich der benötigten Simulationsmodelle ein: In der Automobilindustrie lag der Fokus bisher hauptsächlich auf Komponenten innerhalb des Fahrzeugs.

c. Die Softwarestruktur des Umgebungsmodells

Ein zentrales Element von Fahrerassistenzsystemen ist die Umfeldsensorik. Um die Sensorik auch simulativ abbilden zu können, ist ein Modell der Fahrzeugumgebung notwendig. Da sich dieses Modell von bisherigen, meist physikalisch-motivierten Modellen des Antriebsstrangs oder der Fahrdynamik unterscheidet, wird an dieser Stelle näher auf dessen Eigenschaften eingegangen. Zusätzlich wird eine mögliche Struktur des Modelles, motiviert aus Ansätzen von Spiele-Engines²³, vorgestellt.

Eine physikalisch motivierte Modellierung unterschiedlicher Sensoreffekte, wie beispielsweise Reflexion oder Absorption, erfordert eine dreidimensionale Repräsentation der umgebenden Objekte. Um die Komplexität des Umgebungsmodells nicht unnötig zu erhöhen, ist es sinnvoll interne, zeitabhängige Zustände zu vermeiden: Alle Zustände der Umgebungsobjekte sind durch externe Simulationsmodule zu steuern. Dies erlaubt es, numerisch aufwändige Berechnungen in externen Simulationsmodulen zu kalkulieren und den Fokus des Umgebungsmodells auf die Platzierung und konsistente Darstellung der Geometrien zu beschränken. Diese Trennung der Verantwortlichkeiten ermöglicht es, dass eine auf dem Umgebungsmodell arbeitende Sensorsimulation zu jedem beliebigen Simulationszeitpunkt berechnet werden kann. Somit ist das Umgebungsmodell grundsätzlich kompatibel zu Model-Exchange und Kosimulations Ausführungen.

Die Vielzahl möglicher Umgebungsobjekte und deren unterschiedliche Eigenschaften lassen eine allgemeine Struktur aufwändig erscheinen. In Spiele-Engines

²³Als Spiele-Engine (häufig auch Game-Engine) werden vorgefertigte Programmbibliotheken verstanden, die einen zentralen Teil eines Computerspiels ausmachen. Diese werden zur Aufwandsreduktion nicht für jedes Spiel separat implementiert, sondern als Bibliotheken wiederverwendet. Hierzu gehören beispielsweise Plattform-unabhängige Komponenten des Renderings, der Netzwerkkommunikation oder der Audioausgabe [Vgl. Gregory 2014, S.11 ff.].

ist ein Ansatz zur Strukturierung oft durch die Verwendung sogenannter *Game-Objekt* Softwareklassen realisiert. Hiermit lassen sich komplexe virtuelle Welten erschaffen, in denen unterschiedlichste Objekte miteinander agieren: Real oder von künstlicher Intelligenz (engl. Artificial Intelligence, AI) gesteuerte Spieler treten in aufwändig gestalteter Umgebung gegeneinander an. Umgebungsobjekte können geworfen werden, kollidieren oder stellen Hindernisse dar. In Anlehnung an diese Struktur wird hier der Begriff ***SimulationObject*** eingeführt: Aus dieser abstrakten Klasse lassen sich mögliche Umgebungsobjekte ableiten, wobei die Entitäten aus Variablen zur Beschreibung der Geometrie, des Materials sowie eventueller Klänge²⁴ komponiert werden. Um die Datenabfragen bei Veränderungen der Umgebung zu beschleunigen, wird die Unterscheidung in statische sowie dynamische Simulationsobjekte vorgenommen. Statische Simulationsobjekte ändern ihre Eigenschaften während der Simulation nicht. Das dreidimensionale Modell eines Verkehrsschildes stellt beispielsweise ein statisches Simulationsobjekt dar. Dynamische Simulationsobjekte können ihre Eigenschaften während der Simulationsdurchführung ändern und müssen deshalb bei einem Aktualisierungsvorgang berücksichtigt werden. Ein sich bewegendes Fahrzeug stellt ein dynamisches Simulationsobjekt dar. Zur Änderung der Modelleigenschaften, beispielsweise dessen Position, ist eine Schnittstelle zu einem externen Simulationsmodul notwendig, das diese Eigenschaft berechnet. Wie oben begründet soll dies im Sinne der Trennung von Verantwortlichkeiten nicht im Umgebungsmodell geschehen. Das Umgebungsmodell dient als Grundlage für die Simulation der dreidimensionalen Umgebungsobjekte sowie der Umfeldsensorik. Das Element Sensor stellt deshalb ebenfalls ein Kompositum der Klasse dynamisches Simulationsobjekt dar. Als Sensoren sind jedoch nicht ausschließlich Modelle der Umfeldsensorik zu betrachten, wie virtuelle Videokameras oder Radarsensoren, sondern auch abstrakte Sensoren, die allgemeine Informationen des Umgebungsumfelds bereitstellen. Das Umfeldmodell besitzt Informationen bezüglich der Geometrie und der Position aller Objekte. Ein Sensor-Objekt kann deshalb beispielsweise auch eine Abstandsinformation zu bestimmten Hindernissen oder Verkehrsteilnehmern liefern. Um die Abbildung komplexerer Objekte zu ermöglichen, kann ein dynamisches ***SimulationObject*** selbst wieder mehrere ***SimulationObjects*** beinhalten. Abbildung 4.6 visualisiert den Aufbau der Klasse.

Diese Struktur ermöglicht beispielsweise die Abbildung eines virtuellen Fahrzeugs im Umgebungsmodell. Die Instanz Chassis vom Typ `DynamicSimulationObject` kann eine Geometrie des Fahrgestells beinhalten sowie einen relativ dazu positionierten `VideoSensor` vom Typ `Sensor`

²⁴Für die Anwendungsfall eines Fahrsimulators sind auch Klänge der Objekte, wie Motorgeräusche, notwendig.

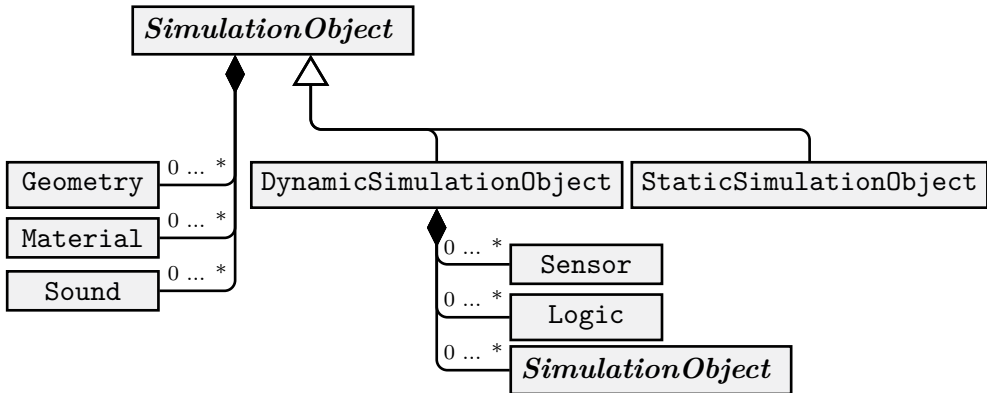


Abb. 4.6: Aufbau der abstrakten Klasse *SimulationObject*

und ein *Logic* Interface zur Positionierung. Zur passenden Positionierung der Reifen können diese wiederum als einzelne Objekte vom Typ *DynamicSimulationObject* mit entsprechenden Geometrien hinzugefügt werden und deren Raddrehung über ein *Logic* Interface übergeben werden. Abbildung 4.7 zeigt die Darstellung einer virtuellen Welt. Das blau hervorgehobene Fahrzeug kann mittels der beschriebenen Instanz des Typs *SimulationObject* in der virtuellen Welt abgebildet werden.

Durch die Verwendung der Klasse *SimulationObject* lassen sich komplexe virtuelle Welten erstellen, die aus transformierbaren, dreidimensionalen Objekten und Sensoren bestehen. Die vorgestellte Klassenstruktur erlaubt außerdem eine einfache Prozedur, um die Berechnung des Umgebungsmodells und dessen Sensorik auszuführen. Algorithmus 1 beschreibt die Ausführung eines Simulationsschrittes des Umgebungsmodells. Grundsätzlich besteht dieser aus zwei Teilen: Zuerst werden die Eigenschaften aller Objekte, basierend auf den extern berechneten Informationen, angepasst. Dies erfolgt für jedes Objekt des Typs *DynamicSimulationObject* durch den rekursiven Funktionsaufruf *UPDATE*. Anschließend werden alle virtuellen Sensoren mittels der *SIMULATE* Prozedur berechnet. Als Resultat stehen anschließend alle Sensorinformationen zur Verfügung, die von anderen Simulationsmodulen genutzt werden können. Eine detaillierte Beschreibung eines Ansatzes zur Berechnung der Sensormodelle für die Umfeldsensorik wird in Kapitel 4.4 vorgestellt.

Ein Anwendungsgebiet des hier vorgestellten Simulationsansatzes sind Hardware-in-the-Loop (HiL) Prüfstände. Hierbei werden reale Komponenten mit den synthetischen Sensordaten stimuliert. Im HiL Umfeld ist die Rechendauer der Simulationsmodelle ein kritischer Faktor. Bei der Verwendung mehrerer virtueller Sensoren kann eine sequentielle Berechnung zu langsam sein.

Algorithmus 1 Simulation des Umgebungsmodells

Require: All inputs i are calculated externally

```
1: for all  $O_{\text{dyn}} \in \text{Environment}$  do
2:   UPDATE( $O_{\text{dyn}}, i$ )
3: end for
4: for all  $O_{\text{dyn}} \in \text{Environment}$  do
5:   SIMULATE( $O_{\text{dyn}}$ )
6: end for

7: procedure UPDATE( $O_{\text{dyn}}, i$ )
8:   Adjust internal states of  $O_{\text{dyn}}$  depending on  $i$ 
9:   for all  $O_{\text{dyn}}$  members as  $o$  do
10:    UPDATE( $o$ )
11:   end for
12: end procedure

13: procedure SIMULATE( $O_{\text{dyn}}$ )
14:   Simulate Sensors
15:   for all  $O_{\text{dyn}}$  members as  $o$  do
16:    SIMULATE( $o$ )
17:   end for
18: end procedure
```



Abb. 4.7: Anwendung der *SimulationObject* Klasse in einer virtuellen Umgebung

Besitzen die unterschiedlichen Sensoren keine gegenseitigen Wechselwirkungen, so lässt sich deren Berechnung parallelisieren.

Die Eingänge des Umgebungsmodells bieten sich als Schnittstelle zur Parallelisierung an. So bleibt die einfache Struktur des Umgebungsmodells erhalten. Zudem handelt es sich bei den Eingangsgrößen in der Regel um geringe Datenvolumen. Beinhaltet jede Recheninstanz das gesamte Umgebungsmodell, so können Informationen zur Szenen-Änderung als Broadcast an alle Instanzen verteilt werden. Auf jedem Knoten wird die `update` Prozedur durchgeführt, wodurch das Umgebungsmodell konsistent über die Instanzen hinweg aktualisiert wird. Daraufhin wird der jeweilige Sensor simuliert und dessen Ergebnisse anschließend zur Verfügung gestellt. Aus Sicht anderer Simulationsmodelle der Gesamtsimulation ändern sich die Schnittstellen in das Umgebungsmodell nicht. Als Eingangsinformationen werden auch im Falle einer Parallelisierung der Sensormodelle die gleichen Eingangsinformationen an das Umgebungsmodell übermittelt. Als Resultat werden die Sensor-Ergebnisse vom Umgebungsmodell den Simulationsmodellen zur Verfügung gestellt. Abbildung 4.8 visualisiert das Ablaufdiagramm der vorgestellten Parallelisierung.

Der vorgestellte Simulationsaufbau stellt ein Konzept dar, welches unabhängig von einer konkreten Implementierung die Simulation von Assistenzsystemen inklusive eines Modells des Eigenfahrzeugs und der Fahrzeugumgebung ermöglicht.

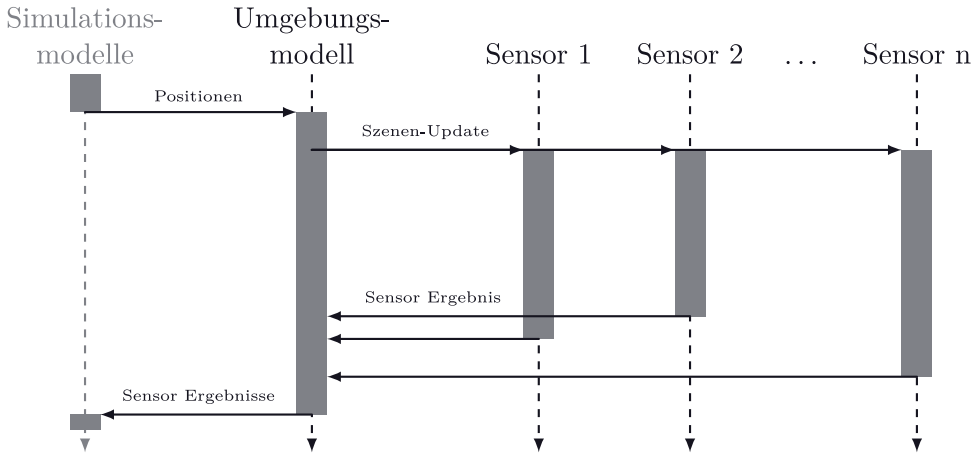


Abb. 4.8: Ablaufdiagramm für verteilte Sensorsimulationen

Vorhandene Simulationsmodelle können aufgrund der Kosimulationsstruktur wiederverwendet werden. Der objektorientierte Ansatz des Umgebungsmodells ermöglicht eine Strukturierung komplexer, dreidimensionaler Elemente in einer virtuellen Welt. Diese virtuelle Welt stellt die Grundlage zur Modellierung von Sensoren dar.

Der folgende Abschnitt abstrahiert die Funktionsweise von Assistenzsystemen, um diese unabhängig eines konkreten Sensortyps beschreiben zu können. Dies dient schließlich als Grundlage für ein allgemeines Sensormodell.

4.2. ADAS Pipeline

Funktionale Sichten der Wirkkette von Sensoren in Fahrerassistenzsystemen werden in [Dietmayer 2015, S.454 f.] und [Bernsteiner 2015, S.74 f.] dargestellt. Allgemein ist allen Sensoren, dass sie basierend auf deren Wirkprinzip über einen Sensorkanal mit dem Umfeld interagieren. Für eine Videokamera wird der Kanal über die Ausbreitung optischen Lichts beschrieben. Lidar-Sensoren basieren auf elektromagnetischen Abstands- und Frequenzmessungen. Für Radarsensoren stellt die Ausbreitung elektromagnetischer Wellen im Gigahertz-Frequenzbereich und deren Interaktion mit realen Objekten den Sensorkanal dar. Ultraschallsensoren arbeiten basierend auf dem Echo-Effekt mittels Luftdruckwellen im Kiloherz-Frequenzbereich oberhalb der Hörschwelle (Vgl. Kapitel 2.2). Eine ideale Kanalantwort stellt das Ergebnis der Interaktion eines Impulses mit der Umgebung dar.

Die Charakteristik eines Sensors modifiziert diese idealisierte Kanalantwort.

Bei einem Ultraschallsensor wird dies hauptsächlich durch die Sende- und Empfangscharakteristik der Piezo-Antenne verursacht, die beispielsweise eine Trägheit aufweist und somit den stimulierten Sendeimpuls verzerrt. Bei einer Videokamera stellen beispielsweise Linsenverzerrungen eine solche Sensorcharakteristik dar. Der Einfluss der Sensorcharakteristik liefert als Ergebnis das im Folgenden als Sensorrohdaten bezeichnete Signal.

Bei Assistenzsystemen werden diese Sensorrohdaten einer Signalverarbeitung (engl. Digital Signal Processing, DSP) unterzogen. Dabei werden nötige Merkmale für die anschließende Klassifizierung extrahiert, beziehungsweise die Daten für eine Objekterkennung optimiert. Bei Radarsensoren besteht dieser Schritt aus der Generierung von Reflexpunkten, die Informationen zum Abstand und der Relativgeschwindigkeit zu stark reflektierenden Objekten enthalten.

In der nachgeschalteten Objekterkennung werden aus den verarbeiteten Sensordaten Objekte extrahiert. Hier wird versucht Objekte wie Fahrzeuge, Radfahrer oder Fußgänger zu erkennen sowie bewegte von stehenden Objekten zu unterscheiden. Im Fall einer Videokamera wird nach Objekten innerhalb eines Bildes gesucht, teilweise basierend auf charakteristischen Merkmalen oder mit Hilfe trainierter neuronaler Netze. Bei Radarsensoren werden Reflexpunkte gruppiert, die sich örtlich an ähnlicher Stelle befinden und sich mit ähnlicher Geschwindigkeit bewegen. Je nach Reflexionsintensität und Anzahl der Reflexpunkte wird auf den Objekttyp geschlossen.

Die darauf aufbauende Situationsanalyse schätzt anhand des zeitlichen Verlaufs der Objektlisten den aktuellen Verkehrszustand ein. Diese Informationen der aktuellen Situation bilden schließlich die Grundlage für Anwendungen wie einen Notbremsassistenten.

Sind im Fahrzeug mehrere Sensoren verbaut, so können die Informationen der verschiedenen Sensoren innerhalb der beschriebenen Wirkkette zusammengeführt werden. Dies kann an unterschiedlichen Punkten realisiert werden. Grundsätzlich gilt, je früher die Fusion der Sensordaten stattfindet, desto mehr Informationen sind vorhanden, die im Rahmen der Zusammenführung verwendet werden können, da bisher noch keine vermeintlich *unnötigen* Informationen verworfen wurden. Gleichzeitig erfordert eine Fusion in frühen Schritten der Wirkkette auch eine hohe Kommunikationsbandbreite, da es sich hierbei um große Datenmengen handelt. Die Fusion der Rohdaten von acht Videostreams²⁵ führt unkomprimiert bei einer Auflösung von 1920×1080 Pixeln, 50 Bildern je Sekunde und 24 Bit/Pixel zu einem Datenaufkommen von rund 2,5 GByte/s.

²⁵Acht Videokameras werden beispielsweise in Fahrzeugen der Firma Tesla verwendet [Vgl. The Tesla Team 2016]

Eine zum jetzigen Zeitpunkt häufig anzutreffende Stelle der Datenfusion basiert auf der Zusammenführung von Objektlisten²⁶. Diese sind das Resultat der Objekterkennung und bestehen aus einer Auflistung der erfassten Objekte sowie dazugehöriger Merkmale. So können beispielsweise videobasierte und radarbasierte Objektlisten miteinander verglichen werden. Die geringen Datenraten erleichtern zudem den Datentransfer auf gängigen Netzwerkverbindungen im Automobil, wie dem CAN-BUS oder Flexray. Die Fusion von acht Objektlisten, die je 10 Objekte mit einem Datenvolumen von 32 Bit/Objekt beinhalten, verursacht bei 50 Objektlisten je Sekunde und Sender ein Datenaufkommen von lediglich 16 kByte/s. Abbildung 4.9 zeigt die beschriebene abstrahierte funktionale Wirkkette von Fahrerassistenzsystemen sowie mögliche Fusionspunkte.

Die möglichen technischen Layouts zur Umsetzung dieser Wirkkette sind vielfältig: Sogenannte Sensor Control Units verbinden in einem Gehäuse den Sensor, die Datenverarbeitung, die Objekterkennung sowie teilweise auch die Anwendung selbst. So können Assistenzsysteme, wie ein Radar-basierter Notbremsassistent, in einer kompakten und kostengünstigen Bauweise realisiert werden. Dem gegenüber stehen Layouts mit einer Vielzahl verteilter Sensoren, deren Informationen in einem zentralen Steuergerät fusioniert werden. Auf dem zentralen Steuergerät können die Klassifizierung der Objekte, die Situationsanalyse sowie die Anwendung implementiert sein. Tabelle 4.2 listet typische Beispiele für *Sensorrohdaten* und *Verarbeitete Daten* der vier im Automobilumfeld gängigen Sensoren auf.

Anhand der dargestellten Wirkkette lassen sich mögliche Schnittstellen für die Modellbildung erkennen. Das im vorherigen Unterkapitel beschriebene Umgebungsmodell bildet die Grundlage zur Beschreibung virtueller, dreidimensionaler Objekte. Für die Modellierung eines Sensors, der auf dem Szenegraphen arbeitet, gibt es unterschiedliche Ansätze. Der nachfolgende Vergleich dieser Ansätze basiert auf der Kategorisierung von [Schubert 2015].

Ideale Sensormodelle generieren Objektlisten, die auf fehlerfreien Informationen der Objekte und Verkehrsteilnehmer im Sichtfeld des virtuellen Sensors basieren. Darin enthaltene Positions-, Typen- und Geschwindigkeitsinformationen werden direkt aus dem Umfeldmodell verwendet. Der Rechenaufwand ist gering, sodass diese Modelle sich gut für den Einsatz in Echtzeit-Simulationen eignen [Vgl. VIRE Simulationstechnologie GmbH 2016]. Mit diesen Modellen können Ground-Truth-Daten generiert werden, also die unverfälschte Information der virtuellen Umgebung, die als Referenz für Objekterkennungsalgorithmen genutzt werden können. Auch für funktionale Tests, bei denen von idealem Sensorverhalten ausgegangen wird, können diese Modelle verwendet werden.

²⁶In Anhang A ist der Aufbau von Objektlisten detaillierter beschrieben.

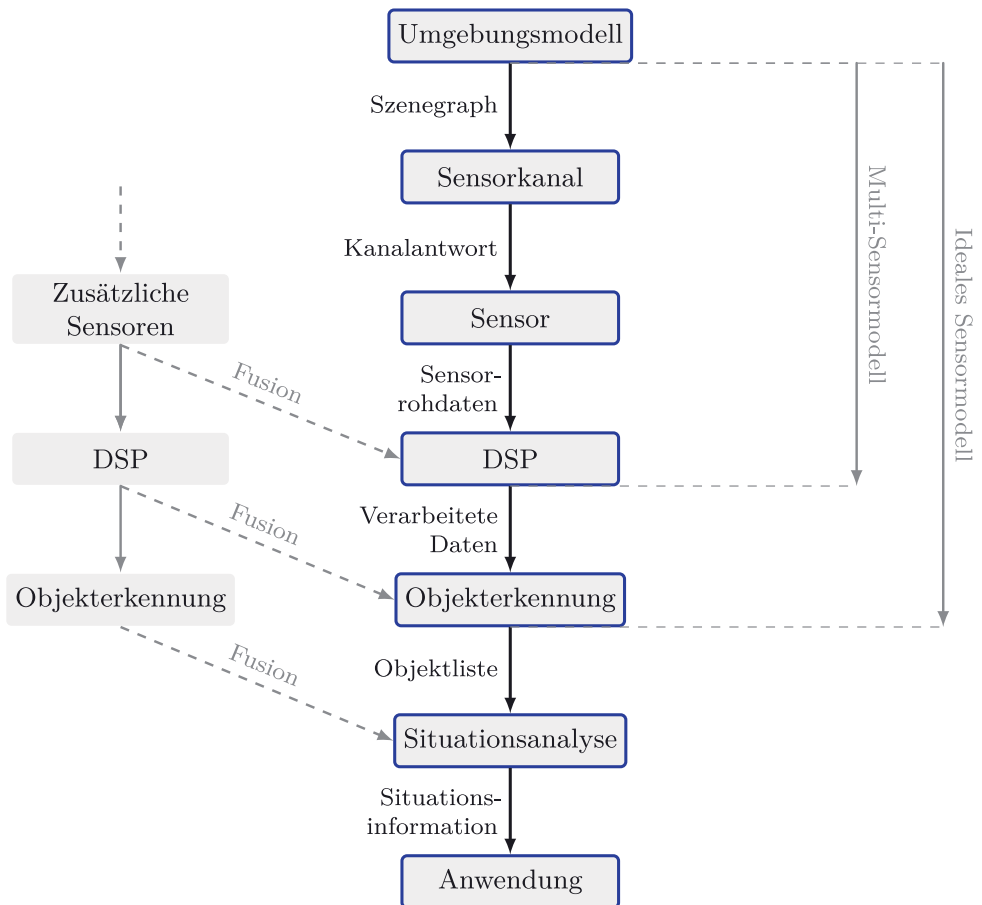


Abb. 4.9: Abstrahierte, funktionale Wirkkette von Fahrerassistenzsystemen

Probabilistische Sensormodelle bauen auf einem idealen Sensormodell auf und fügen statistisch motiviertes Fehlverhalten hinzu. So werden Einträge der Objektliste variiert, indem beispielsweise Distanzen zu Verkehrsteilnehmern mit Rauschen beaufschlagt werden. Hiermit lässt sich die Robustheit einer *Situationsanalyse* oder *Anwendung* analysieren. Zur Realisierung dieser Sensormodelle sind Statistiken des Sensorfehlverhaltens notwendig. Der probabilistische Ansatz kann auch zur Modellierung von Sensorfehlern verwendet werden, deren Abbildung in einem physikalischen Modell zu komplex ist.

Physikalische Sensormodelle bilden das Wirkprinzip des Sensors und damit dessen Interaktion mit der Umgebung detailliert nach. Bei der Umsetzung vorhandener physikalischer Sensormodelle liegt der Anwendungsfokus meist auf der Systemauslegung. Beispielsweise wird im Rahmen der Dissertation [Mayer 2008] das Modell eines frequenzmodulierten Dauerstrichradars (engl. Frequency Modulated Continuous Wave, FMCW) entwickelt, das sich zur Systemauslegung eines realen Radarsensors eignet. Hierbei wird das Wirkprinzip des Sensors auf physikalischer Ebene sehr detailliert modelliert. Damit gehen hohe Rechenzeiten einher, was den Einsatz solcher Modelle in Echtzeitsimulationen meist ausschließt. Die physikalischen Modelle existieren zudem je für einen bestimmten Sensortyp. Ein Transfer auf andere Sensortypen ist mit hohem Aufwand verbunden. Tabelle 4.3 stellt die drei Modellierungsansätze gegenüber.

In den nachfolgenden Unterkapiteln wird die Modellierung zweier Ansätze detailliert ausgeführt: Einerseits wird ein einfaches Modell entworfen, welches sich zum Testen der *Situationsanalyse* und der *Anwendung* realer Assistenzsysteme eignet. Dieses **ideale Sensormodell** abstrahiert die Module *Sensorkanal*, *Sensor*, *DSP* sowie *Objekterkennung*. Es arbeitet somit auf dem Umgebungsmodell und stellt eine Objektliste als Ergebnis zur Verfügung.

Andererseits wird ein detaillierteres Sensormodell entworfen, das sich zusätzlich zur Stimulation der *Objekterkennung* mittels virtueller Daten eignet. Das hierfür vorgestellte Modell wird im Folgenden als **Multi-Sensormodell** bezeichnet. Es arbeitet ebenfalls auf dem Umgebungsmodell und stellt *Verarbeitete Daten* als Ergebnis zur Verfügung. Die Modellbildung basiert auf einem physikalisch motivierten Ansatz.

Die Herausforderungen, dass physikalische Modelle sehr rechenintensiv und wenig anpassbar sind, werden folgendermaßen angegangen: Zur Reduktion der Komplexität wird das Sensormodell auf die im Testfokus stehenden Größen und Effekte begrenzt. Zusätzlich wird die Berechnung soweit wie möglich auf Hardware-Beschleuniger verlagert. Dies ermöglicht eine Reduktion der Berechnungsschrittweiten. Grundsätzlich wird im Folgenden ein Strahlen-basierter Ansatz zur Darstellung der räumlichen Signalausbreitung verwendet (Raytra-

cing). Dieser lässt sich sehr effizient parallelisieren und kann mit Hilfe von Grafikkarten beschleunigt werden. Die Fähigkeit zur Adaption unterschiedlicher Sensortypen wird bei der Modellbildung von Beginn an berücksichtigt. Dadurch ist es nicht nötig ein fertiges Modell zu transferieren, sondern das **Multi-Sensormodell** dient als gemeinsames Framework, das die Virtualisierung von Ultraschall-, Radar-, Lidar- und Videosensoren erlaubt.

4.3. Ideales Sensormodell

Je nach Testfokus ist es häufig empfehlenswert, im Sinne Albert Einsteins, die „Dinge so einfach wie möglich, aber nicht einfacher“ zu modellieren. Kann im Rahmen von Tests der Funktionsmodule *Situationsanalyse* oder *Anwendung* von idealisiertem Sensorverhalten ausgegangen werden, so können gröbere Vereinfachungen bei der Sensormodellierung angenommen werden. Sensormodelle, die auf einem Umgebungsmodell arbeiten und als Ergebnisse eine Objektliste liefern, sind in Abbildung 4.9 als **ideale Sensormodelle** klassifiziert. Sie beinhalten die Module *Sensorkanal*, *Sensor*, *DSP* und *Objekterkennung* der Wirkkette von Fahrerassistenzsystemen.

Eine grundlegende Eigenschaft der in Assistenzsystemen eingesetzten Sensoren ist der Sichtbereich. Dieser beschreibt denjenigen Raum um einen Sensor, innerhalb dessen eine Stimulation wahrgenommen werden kann. In erster Annäherung wird dieser Bereich hart abgegrenzt. Somit lässt sich die Funktionsweise des idealen Sensormodells folgendermaßen beschreiben: Wenn sich ein virtuelles Objekt im Sichtbereich des Sensors befindet, so wird es in die Objektliste aufgenommen (Algorithmus 2).

Bereits mit diesem einfachen Ansatz können unterschiedliche Testszenerien realisiert werden. Die zentrale Funktionalität, die durch dieses Modell bereitgestellt wird, ist die geometrische Transformation von Objekten in das lokale Koordinatensystem des virtuellen Sensors. Als Eingang in das Modell dienen die globalen Objektinformationen und als Ergebnis wird eine Liste mit Objektinformationen relativ zum Sensor bereitgestellt.

Um diese geometrischen Transformationen nicht von neuem zu implementieren, ist die Verwendung von Programmbibliotheken sinnvoll. Hierfür gibt es im Umfeld der Computergrafik optimierte Bibliotheken, die sich für eine Realisierung des Sensormodells eignen. Neben einer effizienten Umsetzung geometrischer Operationen muss eine geeignete Bibliothek zusätzlich die Arbeit mit erweiterbaren, dreidimensionalen Gitterstrukturen erlauben. Im Hinblick auf die iterative Absicherungsstrategie ist es notwendig, dass neue Effekte in die Simulation mitaufgenommen werden können. Das Sensormodell ist hierbei ein möglicher Einstiegspunkt, weshalb eine Bibliothek zur Repräsentation

Algorithmus 2 Ideales Sensormodell

```

1: objectlist = create empty objectlist
2: objects = All objects in virtual world
3: for all objects do
4:   if RECOGNIZED(object) then
5:     Add relative object information to objectlist
6:   end if
7: end for

8: procedure RECOGNIZED(object)
9:   if object is in sensor's Field-of-View then
10:    return True
11:  else
12:    return False
13:  end if
14: end procedure

```

des Umfelds einfach erweiterbar sein sollte. Optische Effekte werden durch Bibliotheken für Computergrafiken ausgiebig unterstützt. Zusätzlich ist die Komposition von Kenngrößen anderer Sensorsysteme zur Realisierung der vorgestellten Klasse *SimulationObject* notwendig. Im Rahmen dieser Arbeit wird für die dreidimensionale Repräsentation der Umgebung die quelloffene C++-Bibliothek [OpenSceneGraph] verwendet. Diese Library stellt effiziente Implementierungen von Matrix-Multiplikationen zur Darstellung geometrischer Transformationen zur Verfügung. Gleichzeitig wird die OpenGL Pipeline abstrahiert, was die Darstellung eines Szenegraphen auf Grafikkarten erlaubt. Neben dem idealen Sensormodell steht somit zusätzlich stets eine optische Darstellung der virtuellen Welt zur Verfügung. Diese Visualisierung kann gleichzeitig als synthetisches Videosensormodell dienen. In der Modellbildung wird hier nicht näher darauf eingegangen. Im Rahmen der Validierung wird dieses Video-Sensormodell ebenfalls betrachtet.

Lässt man die Visualisierung des idealen Sensormodells außer Betracht, so hat das bisherige Modell keinerlei Sensorsystem-spezifischen Eigenschaften. Bisher basieren die Einträge der Objektliste ausschließlich auf der Bedingung, dass sich ein Objekt im Sensorsichtfeld befindet. Die Performance realer Klassifikatoren wird maßgeblich von den Materialeigenschaften und Objektgeometrien beeinflusst. Werden virtuelle Objekte mit Material- und Geometrie-Informationen versehen, so können diese im Sensormodell verwendet werden. Für ein Modell eines Radarsensors kann beispielsweise jedem virtuellen Objekt ein Radar-

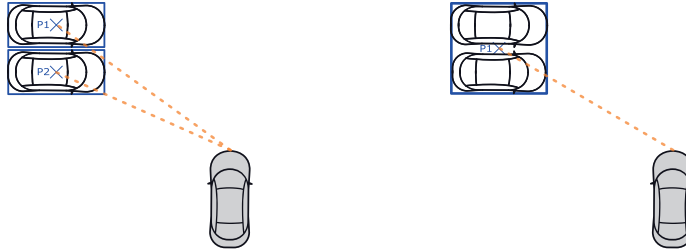


Abb. 4.10: Modellierung einer Fehlererkennung

querschnitt (engl. Radar Cross Section, RCS-Wert) zugeordnet werden. Diese Eigenschaft definiert, wie stark Radarsignale vom Objekt reflektiert werden und hat somit großen Einfluss auf die Qualität der Objekterkennung. Objekte mit einem kleinen RCS-Wert werden von einem Radarsensor eventuell nicht erkannt. Als Erweiterung des idealen Sensormodells wird ein RCS-Schwellwert definiert werden, der notwendig ist, damit ein Objekt erkannt und in die Objektliste mitaufgenommen wird. Analog wird für die akustische beziehungsweise optische Reflexivität von Objekten bei Ultraschall- und Lidar-Sensoren vorgegangen. Für Video-basierte Systeme wäre der Kontrast eines Objekts zu seiner Umgebung interessant. Diese Größe ist jedoch nicht von einem Objekt allein abhängig und lässt sich im Rahmen des idealen Modells nicht direkt ableiten.

Der Vorteil des idealen Sensormodells liegt in dessen Einfachheit. Deshalb ist die Performance des Modells, im Vergleich zu komplexeren, physikalisch motivierten Ansätzen, sehr hoch. Wenn der Detailgrad des Modells ausreicht, so sollte stets ein ideales Sensormodell bevorzugt werden. Zur Umsetzung der in Kapitel 3 vorgestellten Absicherungsstrategie ist es eventuell nötig, neue Effekte, die im Rahmen realer Testfahrten erkannt wurden, in die Simulation zu überführen. Das folgende Beispiel veranschaulicht, wie dieses Vorgehen konkret aussehen kann.

Während Fahrversuchen würde sich herausstellen, dass die Objekterkennung eines Radarsensors zwei Fahrzeuge unter bestimmten Bedingungen „verschmilzt“. Es wird somit anstatt zwei separater Objekte nur ein einzelnes Objekt erkannt. Abbildung 4.10 veranschaulicht dieses Fehlverhalten. Nach einer Analyse würde sich herausstellen, dass dieser Effekt dann auftritt, wenn zwei Objekte einen geringen Abstand zueinander sowie einen ähnlichen RCS-Wert haben und weiter als 10 Meter vom Sensor entfernt sind. Lässt sich dieses Fehlverhalten am System nicht beheben, so ist es sinnvoll diesen Effekt in die Simulation zu transferieren. Der Algorithmus des idealen Sensormodells kann daraufhin um die Beschreibung des beobachteten Fehlverhaltens erweitert werden. Algorithmus 3 listet den Pseudo-Code des idealen Sensormodells inklusive

des beschriebenen Fehlverhaltens auf. Das Ergebnis dieses erweiterten Modells

Algorithmus 3 Ideales Sensormodell mit Fehlverhalten

```

1: objectlist = create empty objectlist
2: objects = All objects in virtual world
3: for all objects do
4:   if RECOGNIZED(object) then
5:     Add relative object information to objectlist
6:   end if
7: end for
8: ADDERRORS(objectlist)

9: procedure ADDERRORS(objectlist)
10:  for all objects do
11:    if Other object fullfills error criteria then
12:      Modify object information
13:      Remove other object from objectlist
14:    end if
15:  end for
16: end procedure

```

ist eine Objektliste, die einen aus realen Testfahrten erkannten Effekt abbildet. Dieses Vorgehen zeigt, wie die Simulation beziehungsweise das Sensormodell im Rahmen des iterativen Absicherungsansatzes erweitert werden können.

Soll in Tests zusätzlich die Objekterkennung miteinbezogen werden, so ist dafür das ideale Sensormodell nicht ausreichend. Im nächsten Kapitel wird hierfür das Multi-Sensormodell entworfen.

4.4. Multi-Sensormodell

Für Closed-Loop Tests von Assistenzsystemen, bei denen auch die Objekterkennung miteinbezogen werden soll, ist die synthetische Generierung von *Sensorrohdaten* beziehungsweise *Verarbeiteten Daten* nötig (Vgl. Kapitel 4.2). Hierfür wird im Folgenden ein Modell entworfen, das die physikalischen Wirkprinzipien der vier Sensortypen vereinfacht in einer virtuellen Umgebung abbildet.

Sensor	Sensorrohdaten	Verarbeitete Daten
Ultraschall	Zeitsignal	Primär- & Sekundär-Reflexe
Radar	Zeitsignal	Distanz & Relativ-Geschwindigkeit zu Reflexzentren
Lidar	Impulslaufzeiten	Punktewolke mit Distanzen & Intensitäten
Video	Lichtintensitäten	Prozessiertes Bild

Tabelle 4.2: Sensorrohdaten und verarbeitete Daten unterschiedlicher Sensoren

Kriterium	Ideale Modelle	Probabilistische Modelle	Physikalische Modelle
Sensorfehler	Nicht modelliert	Realistische Statistik	Realistische Einzelmessung
Rechenkomplexität	Gering	Gering	Sehr hoch
Anpassbarkeit an unterschiedliche Sensoren	Nicht anwendbar	Sehr hoch	Gering

Tabelle 4.3: Vergleich dreier Modellierungsansätze nach [Schubert 2015, S. 70]

Anforderungen an das Multi-Sensormodell

Die Anforderungen an das Sensortypen-übergreifende Modell werden in die Bereiche *Systemanforderungen* und *Physikalische Effekte* eingeordnet. Im Rahmen der Abschlussarbeit [Olscher 2016] wurde eine solche Kategorisierung für ein Radarsensormodell erarbeitet. Hierauf aufbauend listet Tabelle 4.4 die Anforderungen unabhängig von einem konkreten Sensorsystem auf.

Eigenschaft	Beschreibung
Systemanforderungen	
Datenhaltung	Konsistente Umfeldinformation über alle Sensortypen hinweg Single-Source-Prinzip
Integration	Verwendbarkeit in MiL, SiL und HiL Umgebungen
Erweiterbarkeit	Möglichkeit der Erweiterung hinsichtlich physikalischer Effekte und Materialien Parametrierung von Modellgrößen
Rechenzeit	Größenordnung $t \lesssim 10^{-2}$ s/Sensor-Rohsignal
Physikalische Effekte	
Umfeldmodell	Konsistente Materialeigenschaften
Signal-eigenschaften	Verdeckung Reflexion Transmission Absorption Atmosphärische Dämpfung
Antennen-eigenschaften	Räumlich aufgelöste Intensitätsverteilung Abbildung der zeitlichen Antennencharakteristik Variable Auflösung (Genauigkeitsraster)

Tabelle 4.4: Kategorisierung der Modellanforderungen

Die konsistente Datenhaltung wird durch die in Kapitel 4.1 vorgestellte Architektur sichergestellt. Sensoren können Objekten des Typs `DynamicSimulationObject` zugefügt werden. Um Objektgeometrien und Materialeigenschaften für unterschiedliche Sensorsysteme innerhalb eines Objekts zu beschreiben, werden die Membervariablen `Geometrie` und `Material` folgendermaßen realisiert: Die Klassen werden so implementiert, dass unterschiedliche

Geometrie	Material
mesh : default	texture : default
mesh : uss	texture : uss
mesh : radar	texture : radar
mesh : lidar	texture : lidar
mesh : video	texture : video
get()	get()
get(sensor_type)	get(sensor_type)

Abb. 4.11: Klassenstruktur der Typen **Geometrie** und **Material**

Geometrien und Materialien für die verschiedenen Sensortypen angegeben und abgefragt werden können. Der Aufbau dieser Klassen ist in Abbildung 4.11 dargestellt. Diese Struktur ermöglicht eine einheitliche Transformation je Objekt und die gleichzeitige Darstellung unterschiedlicher Eigenschaften je Sensor. Dadurch können separate dreidimensionale Geometrien und Texturen zur Beschreibung der Signalinteraktion auf dem Gitter angegeben werden. So lässt sich das Modell eines Fahrzeugs für den virtuellen Videosensor mittels optischer Texturen abbilden. Gleichzeitig kann ein vereinfachtes Fahrzeugmodell für den virtuellen Ultraschallsensor mittels *akustischer Texturen* beschrieben, oder ein Sub-RCS Modellierungsansatz für den virtuellen Radarsensor angewandt werden. Tabelle 4.5 listet exemplarische Sensorsystem-spezifische Modelle für ein Fahrzeug auf. Für die Sensoren Lidar, Radar und Ultraschall werden die Reflexintensitäten des Modells per Grauwert visualisiert. Der in Kapitel 3.2 vorgestellte Objektkatalog ist für die Verwendung dieses konsistenten Umgebungsmodells entsprechend zu erweitern, damit dieser Modelle für unterschiedliche Sensortypen beinhaltet.

Um eine einheitliche Beschreibung der Wirkprinzipien unterschiedlicher




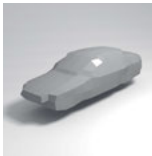
Video	Lidar	Radar	Ultraschall
			

Tabelle 4.5: Exemplarisches Fahrzeugmodell für unterschiedliche Sensortypen

Sensoren zu erhalten, wird darauf zurückgegriffen, dass Ultraschall-, Radar- sowie Lidar-Sensoren grundsätzlich nach dem Echolot-Prinzip arbeiten. Ein Sendeimpuls wird ausgestrahlt und interagiert mit der Umgebung. Ein Teil des gesendeten Impulses erreicht schließlich den Empfänger. Demgegenüber gibt es bei Videokameras keinen aktiven Sendeimpuls. Hierbei wird in der Umgebung vorhandenes Licht vom Sensor registriert. Betrachtet man jedoch die unterschiedlichen Lichtquellen, die in der Regel vom Sensorsystem nicht aktiv beeinflussbar sind, so stellen diese im Sinne des obengenannten Wirkprinzips den Sender dar. Durch diese Verallgemeinerung können alle vier Sensortypen basierend auf einem Sende-Empfangsmechanismus modelliert werden.

Deutlich unterscheiden sich die Sensoren hinsichtlich der verwendeten Wellenlänge und Wellengeschwindigkeit, der Sendecharakteristik sowie der damit verbundenen Winkeltrennfähigkeit. Die radial abstrahlende Charakteristik der Schallwellenfront von Ultraschallsensoren hat eine Wellenlänge im Millimeter-Bereich. An Oberflächen wird die Welle vorrangig diffus reflektiert. Ähnlich verhält sich die Reflexion von Radar-Keulen, die gebündelt mit einer Wellenlänge im Millimeter-Bereich mit Lichtgeschwindigkeit transportiert werden. Lidar stellt stark gebündelte elektromagnetische Strahlen mit Wellenlängen von mehreren hundert Nanometern, teilweise auch im sichtbaren Lichtspektrum, dar. Videosensoren arbeiten im sichtbaren Bereich des Lichts (Vgl. Kapitel 2.2). Diese Eigenschaften²⁷ der vier Sensoren sind zusammenfassend in Tabelle 4.6 aufgelistet. Zudem ist vereinfachend eine Größenordnung für typische Reichweiten der Sensoren sowie dazugehörige Impulslaufzeiten angegeben.

Im Vergleich zu den restlichen Sensoren ist die Impulslaufzeit des Ultraschallsensors deutlich länger. Bewegt sich das Fahrzeug während der Sensormessungen, so hat dies am meisten Einfluss auf das Signal des Ultraschallsensors. Aufgrund von Störgeräuschen bei hohen Geschwindigkeiten werden Ultraschallsensoren vorrangig im niederen Geschwindigkeitsbereich eingesetzt. Bei einer Fahrzeuggeschwindigkeit von $v = 10 \frac{\text{km}}{\text{h}}$ ergibt sich eine Änderung der Fahrzeugposition während einer maximalen Impulslaufzeit $t = 2 \cdot 1,7 \cdot 10^{-2} \text{s}$ von $d = v/t \approx 9,4 \text{ cm}$. Im Rahmen der Modellbildung des Multi-Sensormodells wird die Sensoreigenbewegung nicht explizit berücksichtigt. Soll dieser Einfluss dennoch in Betrachtungen miteinbezogen werden, so kann die Eigenbewegung durch eine Diskretisierung und anschließende Mittelwertbildung angenähert werden. Dieses Vorgehen wird im Umfeld der Computergrafik für unterschiedliche Anwendungen verwendet: Bewegungsunschärfe-Effekte synthetischer Bilder werden über zeitlich hoch-aufgelöste Farbinformationen und anschließender

²⁷Basierend auf folgenden Daten: Ultraschall Frequenz 45 kHz [Noll 2015, S. 247], Radar Frequenz 76,5 GHz [Winner 2015b, S. 260], Lidar Wellenlänge 905 nm [Gotzig 2015, S. 331], $c_{Luft} \approx 3 \cdot 10^2 \frac{\text{m}}{\text{s}}$ und $c_{Licht} \approx 3 \cdot 10^8 \frac{\text{m}}{\text{s}}$





Sensor	Sende- charakteristik	Wellenlänge	Reichweite	Impuls- laufzeit
Ultraschall		$\approx 8 \cdot 10^{-3} \text{ m}$	$\approx 5 \text{ m}$	$< 1,7 \cdot 10^{-2} \text{ s}$
Radar		$\approx 4 \cdot 10^{-3} \text{ m}$	$\approx 200 \text{ m}$	$< 1,3 \cdot 10^{-6} \text{ s}$
Lidar		$\approx 9 \cdot 10^{-7} \text{ m}$	$\approx 200 \text{ m}$	$< 1,3 \cdot 10^{-6} \text{ s}$
Video		$\approx 5 \cdot 10^{-7} \text{ m}$	$\approx 80 \text{ m}$	$< 5,3 \cdot 10^{-7} \text{ s}$

Tabelle 4.6: Merkmale der unterschiedlichen Sensortypen

Mittelwertbildung gewonnen (engl. Motion-Blur). Eine Kantenglättung wird über die Mittelwertbildung örtlich hoch-aufgelöster Farbinformationen erzielt (engl. Anti-Aliasing) [Vgl. Treuille 2009]. Für den nachfolgend verwendeten Ansatz des Raytracings können diese Prinzipien des örtlichen und zeitlichen Supersamplings ebenfalls angewandt werden.

Zur Simulation des Sende-Empfangsmechanismus wird für das Multi-Sensormodell das Prinzip des Raytracings genutzt: Dabei wird das Signal eines Senders entlang seiner Ausbreitungspfade verfolgt, interagiert mit einer virtuellen Umgebung und Teile des Impulses erreichen einen Empfänger. Zur Darstellung fotorealistischer, dreidimensionaler Szenen in der Computergraphik ist dieser Ansatz weit verbreitet.

Raytracing

Zur allgemeinen mathematischen Beschreibung aller Renderalgorithmen, die auf unterschiedliche Weise versuchen die Ausbreitung von Licht zu simulieren, hat James Kajiya in [Kajiya 1986] die fundamentale Rendergleichung aufgestellt. Basierend auf dem Prinzip der geometrischen Optik beschreibt diese, wie sich der Energiefluss von Licht über Flächen hinweg ausbreitet:

$$I(x, x') = g(x, x') \cdot \left[\epsilon(x, x') + \int_S p(x, x', x'') \cdot I(x', x'') dx'' \right] \quad (4.1)$$

Hierbei stellt $I(x, x')$ die Strahldichte [$\text{W}/(\text{m}^2 \cdot \text{sr})$] des Lichts dar, die vom Punkt x' zu dem Punkt x übertragen wird. Der „geometrische Term“ $g(x, x')$

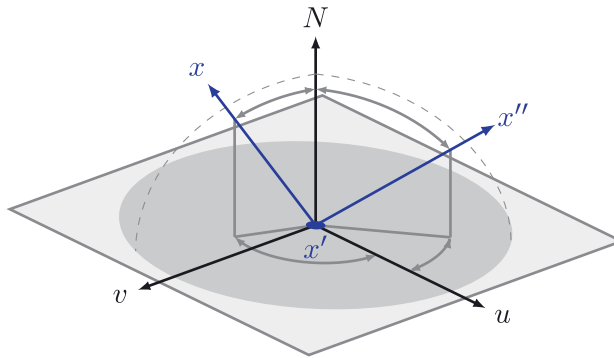


Abb. 4.12: Schematische Darstellung der bidirektionalen Reflektanzverteilungsfunktion nach [Gebhardt 2003]

beschreibt die örtliche Relation der zwei Punkte x und x' . $\epsilon(x, x')$ spiegelt den Anteil des vom Punkt x' zu Punkt x emittierten Lichts wider. Dieser Anteil existiert somit nur für selbstleuchtende Objekte, wie beispielsweise Lichtquellen. Der Term $p(x, x', x'')$ stellt den Anteil des Lichts dar, der von x'' über x' zu x geleitet wird. Dieser Term wird als bidirektionale Reflektanzverteilungsfunktion (engl. Bidirectional Reflectance Distribution Function, BRDF) bezeichnet und ist schematisch in Abbildung 4.12 visualisiert. Durch den BRDF lassen sich somit beliebige optische Materialeigenschaften beschreiben. Um die Eigenschaften realer Objekte in der Computergrafik nachzubilden, ist es notwendig die jeweilige bidirektionale Reflektanzverteilungsfunktion anzunähern. Dies kann einerseits durch Messungen an realen Objekten erfolgen oder andererseits durch die Formulierung als analytische Funktion. Reale Messungen sind sehr zeitintensiv und erfordern einen hohen Speicherbedarf, weshalb in der Computergrafik meist analytische Funktionen verwendet werden. Diese werden als *lokale Beleuchtungsmodelle* bezeichnet. Bekannte Vertreter dieser Modelle sind *Lambert*²⁸ oder *Phong*²⁹. Eine ausführliche Beschreibung dieser Modelle liefert [Gebhardt 2003].

Eine direkte Umsetzung des Prinzips der geometrischen Optik unter Berücksichtigung der Eigenschaften der Rendergleichung stellt Raytracing dar. Um ausbreitende Lichtwellen durch Strahlen anzunähern müssen nach [Haferkorn 2003, S. 37] die folgenden Axiome gelten:

²⁸Lambert-Modelle beschreiben diffus reflektierende Oberflächen.

²⁹Das Phong-Modell ist ein nicht physikalisch basierter Ansatz, der aus einem glänzenden und diffusen Anteil besteht.

1. Licht breitet sich in homogenem Material entlang von Geraden aus.
2. An der Grenze zwischen zwei homogenen isotropen Materialien wird Licht nach dem Reflexionsgesetz reflektiert und nach dem Brechungsgesetz gebrochen.
3. Der Strahlengang ist umkehrbar.
4. Lichtstrahlen durchkreuzen einander, ohne sich gegenseitig zu beeinflussen.

Die Annäherung der Ausbreitung von Wellen durch Strahlen konvergiert nur für sehr hohe Frequenzen. Dies bedeutet, dass die Wellenlänge deutlich kleiner als die Abmessung der Umgebungsobjekte sein muss. Aufgrund dessen wird Raytracing auch als asymptotisches Hochfrequenz-Lösungsverfahren bezeichnet [Vgl. Bachhuber 2011, S.95]. Die Annäherung eignet sich deshalb besonders für Video- und Lidar-Sensoren. Die Wellenlängen von Radar- und Ultraschall-Sensoren bewegen sich im Millimeterbereich, weshalb für diese Sensoren mit einem größeren Fehler aufgrund des Strahlenansatzes zu rechnen ist (Vgl. Tabelle 4.6).

Der algorithmische Ansatz des Raytracings bildet einen Rahmen, um die Phänomene der geometrischen Optik auf einem Computer abzubilden. Grundsätzlich werden hierbei virtuelle Strahlen von einer Quelle emittiert. Jeder Strahl wird anschließend an den Objekten einer Szene reflektiert und gebrochen. Die wiederum hieraus entstehenden Strahlen werden weiterverfolgt, um erneut an Objekten reflektiert und gebrochen zu werden. Nach einer beliebigen Anzahl an Reflexions- und Brechungsvorgängen wird die Strahlenverfolgung beendet.

Beim beschriebenen Raytracingalgorithmus werden viele unnötige Strahlen generiert, da in der Regel nur ein Bruchteil der erzeugten Strahlen den Betrachter erreicht. Hierfür gibt es mehrere Optimierungsansätze, um die Performance des Algorithmus zu verbessern. Eine grundlegende Änderung basiert auf der Umkehrbarkeit des Strahlengangs (Axiom 3): Die Strahlen werden nicht vom Sender, sondern vom Empfänger ausgesandt (Backward Raytracing). Dadurch werden nur Strahlen erzeugt, die Einfluss auf den Betrachter haben. Basierend auf den vom Sender ausgesandten Strahlen (Primärstrahlen) wird außerdem nur eine begrenzte Anzahl neuer Strahlen je Schnittpunkt erzeugt (Sekundärstrahlen). Die Anzahl der generierten Strahlen hängt von den Materialeigenschaften ab. Bei einem opaken Material müssen beispielsweise keine Strahlen zur Berechnung der Brechung generiert werden. Die Anzahl der Reflexionsstrahlen wird je Schnittpunkt zudem häufig auf einen einzelnen Strahl begrenzt, dessen Ausbreitung dem Reflexionsgesetz entspricht. Zur effizienten Berechnung des lokalen Beleuchtungsmodells wird je Schnittpunkt ein Strahl

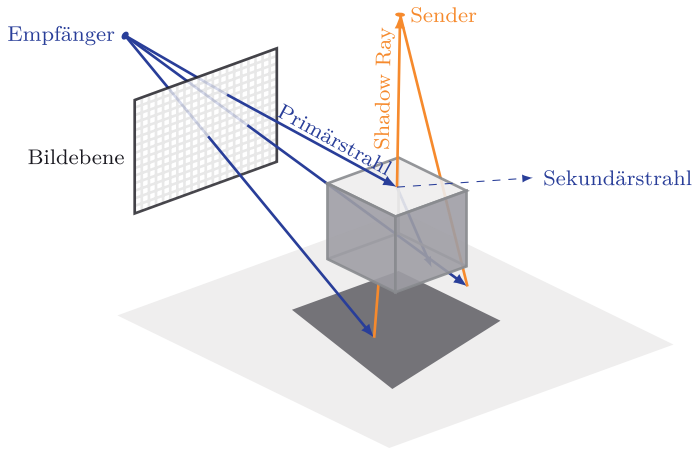


Abb. 4.13: Prinzipskizze des Backward-Raytracings

zu allen Emittlern generiert (Shadow Ray), um die Hauptlichtquellen bei der Auswertung des BRDF zu berücksichtigen. Abbildung 4.13 visualisiert den für das Multi-Sensormodell genutzten Raytracing-Ansatz.

Algorithmus 4 beschreibt den Pseudo-Code einer rekursiven Realisierung des Backward-Raytracings. Hierbei werden zuerst die Primärstrahlen erstellt (Zeile 1). Anschließend wird die Ausbreitung jedes Strahls in der virtuellen Szene verfolgt (engl. TRACE). Zentrales Element dieser Prozedur ist die Verdeckungsberechnung. Hierbei wird ermittelt, welche virtuelle Fläche einen Schnittpunkt mit der betrachteten Halbgeraden aufweist. Bei großen virtuellen Szenen ist dieser Test sehr aufwändig, da die Schnittpunktberechnung im einfachsten Fall für jedes Flächenelement durchgeführt werden muss. Dieser Fall hat für n Flächenelemente lineare Komplexität $\mathcal{O}(n)$. Um die Schnittpunktabfrage zu beschleunigen kann die virtuelle Szene in hierarchische Volumenelemente gegliedert werden (engl. Bounding Volume Hierarchy). Die Verdeckungsberechnung beginnt somit nur auf den Eltern-Volumenelementen und propagiert bei positivem Schnitttest zu den Kinder-Volumenelementen. Durch diese Optimierung kann die Schnittpunktberechnung auf logarithmische Komplexität $\mathcal{O}(\log(n))$ reduziert werden. Da die Strahlen je Simulationsschritt unabhängig voneinander berechnet werden können, lässt sich das Raytracing-Problem parallelisieren. Hierfür gibt es unterschiedliche Software-Frameworks mit vorgefertigten Parallelisierungs- und Optimierungs-Routinen. Im Rahmen dieser Arbeit wird hierfür [OptiX] verwendet, da es sich parallel auf der Grafikkarte ausführen lässt und unterschiedliche Optimierungsalgorithmen beinhaltet.

Ist der Schnittpunkt mit einem virtuellen Objekt gefunden, so ist dessen Ein-

Algorithmus 4 Verwendeter Raytracing-Ansatz

```
1: rays = Generate Primary Rays
2: for all rays do
3:   intensity = TRACE(ray, 0)
4: end for

5: procedure TRACE(ray, depth)
6:   if object got hit then
7:     return SHADE(object, ray, depth)
8:   else
9:     return backgroundIntensity
10:  end if
11: end procedure

12: procedure SHADE(object, ray, depth)
13:   Initialise intensity
14:   if depth  $\geq$  maxDepth then
15:     return intensity
16:   end if
17:   for all emitters do
18:     Generate shadowRay to emitter
19:     Gather emitterInfluence
20:     intensity = intensity + emitterInfluence
21:   end for
22:   Generate secondaryRays based on BRDF of object
23:   for all secondaryRays do
24:     intensity = intensity + TRACE(secondaryRay, depth + 1)
25:   end for
26:   return intensity
27: end procedure
```

fluss auf die am Empfänger auftreffende Intensität zu ermitteln (SHADE). Neben Abbruchbedingungen wird der Einfluss von Emittlern, wie Lichtquellen, betrachtet. Zudem werden abhängig von den Materialeigenschaften am Auftreffpunkt Sekundärstrahlen generiert, die wiederum ebenfalls Einfluss auf die Gesamtintensität haben (Zeile 22). Hierfür erfolgt der rekursive Methodenaufruf zur Strahlverfolgung. Der Berechnungsaufwand wird vorrangig durch die Anzahl generierter Sekundärstrahlen bestimmt, die abhängig von den Eigenschaften des jeweiligen Materials erstellt werden.

Wendet man diesen Raytracing-Ansatz zur Abbildung der Strahlenausbreitung optischen Lichts, so dient dies als Modell eines **Videosensors**. Die Auflösung des Bildrasters bestimmt dabei die Anzahl an Primärstrahlen. Die Kameraoptik beeinflusst die Richtung der Primärstrahlen, indem diese abhängig von der Kamera-Brennweite initialisiert werden. Somit erlaubt der vorgestellte Ansatz die Erzeugung synthetischer Bilddaten zur Repräsentation einer Videokamera.

Zur Generierung einer Punktwolke aus Reflexpunkten, wie sie ein **Lidar-Sensor** liefert, kann der Raytracing-Ansatz leicht vereinfacht angewandt werden. Lidar basiert, wie in Abschnitt 2.2-c beschrieben, auf der Laufzeitmessung gebündelter Lichtimpulse. Um diese Laufzeitmessung in einer virtuellen Umgebung mittels Raytracing darzustellen, kann Algorithmus 4 um die Längenmessung der Primärstrahlen erweitert werden. Da sich bei Lidar der Sender und Empfänger örtlich an derselben Stelle befinden, sind zudem keine Sekundärstrahlen und Shadow Rays notwendig. Die Signallaufzeit kann aus der doppelten Länge des Primärstrahls durch $t = 2 \cdot d_{\text{primär}} / c_{\text{Licht}}$ angenähert werden.

Im Hinblick auf die ADAS-Pipeline bilden sogenannte Radar-Locations die Eingangsdaten der Objekterkennung für **Radar-Sensoren**. Diese Schnittstelle besteht aus mehreren Reflex-Zentren, die im Rahmen einer Signalverarbeitung aus dem Radar-Rohsignal extrahiert wurden. Diese Reflex-Zentren stellen die zum Sensor relativen Positions- und Geschwindigkeitsinformationen dar. Wird Algorithmus 4 dahingehend erweitert, dass Geschwindigkeits-Informationen an den Schnittpunkten abgefragt werden, so ist die synthetische Generierung von Radar-Locations möglich. Hierfür werden die Informationen der Primärstrahlen an örtlich nahe gelegenen Positionen akkumuliert und zu einer Radar-Location zusammengefasst.

Zur Modellierung des Rohsignals eines **Ultraschall-Sensors** ist die Generierung einer zeitlich aufgelösten Signalantwort notwendig. Die emittierte Schallwelle des Senders breitet sich kugelförmig in der Umgebung aus. Im Hinblick auf den verwendeten strahlenbasierten Ansatz kann das empfangene Schallsignal durch die Summe der Intensitäten aller Pfade zwischen dem Sender und dem Empfänger angenähert werden. Anders als beim Raytracing zur

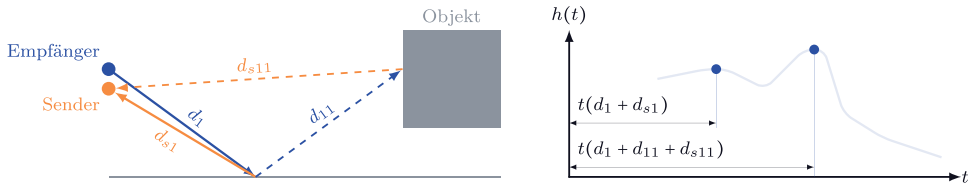


Abb. 4.14: Vereinfachte Visualisierung der Ausbreitung eines Primärstrahls sowie daraus resultierende Anteile der Impulsantwort

Bildgenerierung müssen nicht nur die Intensitäten der sich aufbauenden Pfadhierarchien summiert werden, sondern es ist die gesamte Pfadlänge an jedem Reflexpunkt sowie dessen Intensität notwendig. Abbildung 4.14 visualisiert diesen Ansatz, wobei exemplarisch ein Primärstrahl vom Empfänger ausgesandt wird. Dieser Strahl wird am Boden reflektiert und trifft anschließend auf ein Objekt. Beide Reflexpunkte können zum Sender durch Geraden verbunden werden. Jeder dieser geschlossenen Pfade i zwischen Empfänger und Sender muss durch dessen Laufzeit $t_i = d_i/c_{\text{schall}}$ sowie dessen resultierende Intensität I_i zur Annäherung der Impulsantwort berücksichtigt werden.

In [Siltanen 2010] wird ein Raytracing-basierter Modellierungsansatz zur Berechnung der Raumakustik präsentiert. In Anlehnung an die Rendering-Gleichung (Gleichung 4.1) wird dabei folgende *Akustik-Rendergleichung* hergeleitet:

$$L(y, \Omega_e) = L_0(y, \Omega_e) + \int_G R(x, y, \Omega_e) \cdot L(x, -\Omega_i) dx \quad (4.2)$$

Hierbei stellt $L(y, \Omega_e)$ die akustische Radianz an Punkt y in Richtung Ω_e dar. L_0 beschreibt die akustische selbst-emittierende Radianz an Punkt y . Ansonsten ist die an y auftretende Radianz das Integral über die Fläche G , bestehend aus der an Punkt x eintreffenden Radianz $L(x, -\Omega_i)$ und einem Reflexions-Kernel. Dieser Kernel beinhaltet die akustische Reflektanzverteilungsfunktion. Die Analogie der in [Siltanen 2010] präsentierten Modellierung der akustischen Signalausbreitung mittels Raytracing unterstreicht den Modellierungsansatz des Multi-Sensormodells für unterschiedliche Sensorsysteme.

Unter Berücksichtigung aller Ausbreitungspfade lässt sich somit die Kanalantwort des Umgebungsmodells auf einen Dirac-Impuls annähern. Zusätzlich kann die zeitliche Charakteristik des Senders $s(t)$ und Empfängers $e(t)$ durch eine Faltung der Signale unabhängig von der Berechnung der Impulsantwort hinzugefügt werden:

$$h'_{\text{USS}}(t) = s(t) * h(t) * e(t)$$

Dies ergibt die zeitlich aufgelöste Information der *Sensorrohdaten* für einen Ultraschallsensor. Eine nachgelagerte Signalverarbeitung ermittelt im Fall eines

Ultraschallsensors markante Reflexintensitäten. Konkret bedeutet dies beispielsweise, dass, abzüglich des üblichen Rauschsignals aufgrund von Reflexen an der Straßenoberfläche, das erste Maximum der Impulsantwort gesucht wird. Aus diesem Maximum wird anschließend die Distanzinformation zu einem Objekt ermittelt. Das resultierende Signal der *Verarbeiteten Daten* des Ultraschallsensors besteht somit aus einer Distanzinformation zum nächsten auffällig reflektierenden Objekt. Damit wird versucht, im Rahmen der nachgelagerten Objekterkennung, unter Verwendung weiterer Ultraschall-Sensorsignale auf ein Objekt im Umfeld des Fahrzeugs zu schließen.

Mit Hilfe des Raytracing-basierten Multi-Sensormodells kann die Generierung der *Verarbeiteten Daten* im Falle eines Ultraschallsensors alternativ auch vereinfacht direkt über die Abstandsinformation zu einem auffällig reflektierenden Objekt auf Basis der Primärstrahlen ermittelt werden. Damit ist der Anteil der Signalverarbeitung direkt in den Modellierungsansatz integriert. Gleichzeitig wird dabei jedoch eine eventuell auftretende Fehlinterpretation der Signalverarbeitung aufgrund von Mehrfachreflexionen des Ultraschallimpulses vernachlässigt.

Der Modellierungsansatz des Ultraschallmodells würde somit sowohl eine synthetische Erzeugung der *Sensorrohdaten* sowie der *Verarbeiteten Daten* erlauben. Der Strahlen-basierte Ansatz zur Distanzermittlung ist konsistent mit der Signalmodellierung der restlichen Sensortypen, da hierbei direkt Teile der Signalverarbeitung im Modellierungsansatz abstrahiert werden.

Zusammenfassung

In diesem Kapitel wurde eine Simulationsarchitektur anhand dreier unterschiedlicher Abstraktionsniveaus erarbeitet, um die Kopplung eines Umfeldmodells zur Virtualisierung der Fahrzeugumgebung mit vorhandenen Simulationskomponenten zu ermöglichen. Basierend auf Prinzipien der Spieleindustrie ist dazu die Klassenstruktur des *SimulationObject* zur Vereinheitlichung diverser Entitäten erarbeitet worden. Diese Struktur erlaubt es komplexe Objekte in einer virtuellen Umgebung zu platzieren und eröffnet damit die Möglichkeit zur Integration von Sensormodellen. Die anschließende Modellbildung basiert auf einer abstrahierten Beschreibung der Funktionsweise von Assistenzsystemen, der ADAS Pipeline. Das darauf aufbauende ideale Sensormodell arbeitet auf der virtuellen Umgebung und liefert Objektlisten als Resultat. Dabei wurde auch eine Erweiterung des Modells als exemplarische Anwendung der iterativen Absicherungsstrategie präsentiert. Zusätzlich wurde das physikalisch motivierte Multi-Sensormodell entworfen, das die Wirkprinzipien der vier gängigen Sensorsysteme auf eine Impulsantwort zurückführt und so eine konsistente Modellierung ermöglicht.

Das nachfolgende Kapitel behandelt Anwendungen und Validierungen der in Kapitel 3 erarbeiteten Absicherungsstrategie und der in diesem Kapitel vorgestellten Simulationsarchitektur sowie der Sensormodelle.

5. Anwendung und Validierung

In diesem Kapitel werden die vorgestellten Methoden und Modelle im Rahmen unterschiedlicher Anwendungen betrachtet und validiert. Einerseits werden Elemente der Absicherungsstrategie angewandt, wobei die Beschreibung der Testfälle sowie deren Aussagefähigkeit im Fokus der Betrachtung stehen. Andererseits werden die unterschiedlichen Sensormodelle hinsichtlich ihrer Performance sowie Genauigkeit untersucht und darauf basierend der Mehrwert bei deren Einsatz erläutert.

5.1. Anwendung und Nutzen der Absicherungsstrategie

Um eine Bewertung der präsentierten Absicherungsstrategie vorzunehmen, werden die folgenden Aspekte untersucht: Die eingeführte Ontologie zur Szenariobeschreibung wird für den Transfer realer in virtuelle Fahrscenarien genutzt. Anschließend werden Szenarien automatisiert mit Hilfe eines modellbasierten und eines probabilistischen Ansatzes generiert. Hieran schließt sich die Anwendung der Beschreibung als Parametrierung eines HiL Testsystems an, bevor die Simulationsarchitektur und Sensormodelle detaillierter betrachtet werden.

Zentrales Element stellt der iterativ erweiterbare Katalog von Fahrscenarien dar. Unterschiedliche Pfade der Absicherungsstrategie haben eine Erweiterung des Katalogs zur Folge. Nachfolgend wird die Anwendbarkeit der ontologischen Beschreibung hinsichtlich des Transfers realer Fahrscenarien untersucht.

Transfer realer Fahrscenarien

Die in Kapitel 3.2 vorgestellte Ontologie ermöglicht die Beschreibung von Verkehrsszenarien. Basierend auf frei zugänglichen Messdaten des [KITTI 2018]-Datensatzes lässt sich die Ontologie zur Beschreibung der in Abbildung 5.1 visualisierten Verkehrssituationen verwenden. Die linke Spalte zeigt hierbei das Verkehrsgeschehen aus Sicht einer realen Videokamera. Demgegenüber listet die rechte Spalte die dazugehörigen virtuellen Pendanten derselben Szenen auf. Die der Visualisierung zugrundeliegende Simulation wurde unter Verwendung der Szenario-Ontologie parametrierung. Die jeweiligen XML-Dokumente sind in Anhang E aufgeführt.

Zum Transfer der realen Verkehrssituationen werden unterschiedliche Elemente der Ontologie genutzt. So lässt sich beispielsweise die reale Straße beschreiben, was sich am Verlauf der Mittellinie, der Anzahl der Spuren und



Abb. 5.1: Transfer realer Verkehrssituationen in die Simulation, basierend auf [KITTI 2018]

der Fahrbahnmarkierungen erkennen lässt. Diese Attribute entsprechen in der virtuellen Welt dem realen Gegenstück.

In Relation zu dem Straßenkoordinatensystem sind mehrere statische Objekte definiert. So sind Bäume, Sträucher, Verkehrsschilder und Leitpfosten zu sehen. Deren Position und Ausrichtung ist im XML-Dokument ebenfalls in Anlehnung an die reale Situation beschrieben. Zusätzlich zu den statischen Objekten wird die Beschreibungsmöglichkeit globaler Eigenschaften innerhalb des Elements `Environment` genutzt. Hierzu gehört die Positionierung der Sonnenlichtquelle, was sich auf die Beleuchtung und den Schattenwurf der Szene auswirkt. Unterschiedliche Hintergrundgrafiken werden eingesetzt, um stark vereinfacht weit entfernte Objekte und Umgebungsinformationen in die Simulation zu transferieren.

Zusätzlich beinhalten die Beschreibungen mehrere Fahrzeuge, die als dynamische Akteure definiert sind. Deren Platzierung erfolgt relativ zum Straßenverlauf und sie referenzieren die entsprechenden dreidimensionalen Modelle.

Nachfolgend wird ein Einparkszenario zur Demonstration der Anwendung des dynamischen Teils der Szenariobeschreibung präsentiert. Hierfür werden unterschiedliche Eckpunkte des Manövers definiert, die von einem (virtuellen) Fahrer auszuführen sind. Die Beschreibung des Einparkvorgangs ist in Quellcode 5.1 aufgeführt. Der Einparkvorgang besteht aus 5 Schritten (`Step`) und ist Teil der XML-Beschreibung. Die Schritte werden sequentiell ausgeführt, wobei die je Schritt definierten Manöver nebenläufig sind.

Der erste Schritt des Szenarios besteht aus zwei Manövern (`AccelerateToTargetSpeed` und `DriveToSPosition`). Hierbei wird auf eine Zielgeschwindigkeit beschleunigt und eine s-Position im Straßenkoordinatensystem angefahren. Sind beide Manöver beendet, so ist der erste Schritt

Quellcode 5.1: Beschreibung eines Einparkvorgangs

```
<Step>
  <AccelerateToTargetSpeed value="3" />
  <DriveToSPosition value="10" />
</Step>
<Step>
  <DriveToTPosition value="-12.625" >
    <CancelConditions>
      <SPositionReached value="19" />
    </CancelConditions>
  </DriveToTPosition>
</Step>
<Step>
  <AccelerateToTargetSpeed value="-3" />
  <DriveToTPosition value="-7.625" />
</Step>
<Step>
  <AccelerateToTargetSpeed value="3.1" />
  <DriveToSPosition value="19.5" />
</Step>
<Step>
  <Brake />
</Step>
```

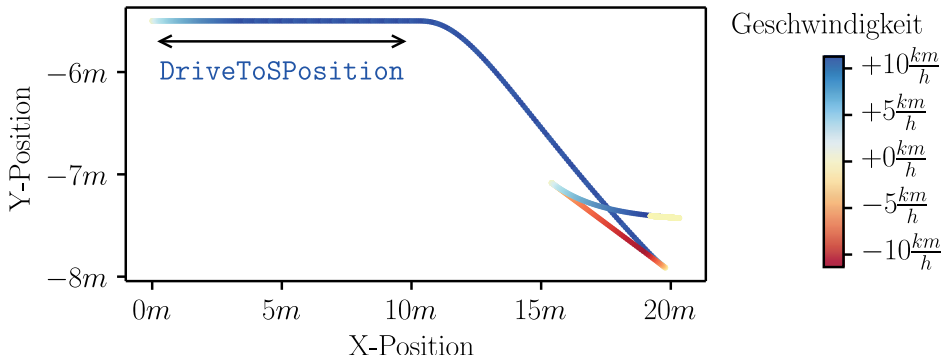


Abb. 5.2: Visualisierung des dynamischen Verlaufs eines Einparkenszenarios

beendet.

Im Rahmen des nächsten Schrittes wird eine t-Position im Straßenkoordinatensystem angefahren, wobei das Manöver `DriveToTPosition` abgebrochen wird, falls vor dem Erreichen der Zielposition, eine definierte S-Position überfahren wurde. Die nachfolgenden Schritte beschreiben einen Rangiervorgang in die Parklücke. Abbildung 5.2 visualisiert den Verlauf der Fahrzeugposition und -geschwindigkeit bei einer Ausführung des Einparkenszenarios. Hierbei ist das Manöver `DriveToSPosition` des ersten Schrittes gut zu erkennen und in der Abbildung hervorgehoben. Anschließend steuert der Fahrer das Fahrzeug auf eine Parklücke zu, reduziert die Geschwindigkeit und das Fahrzeug wird nach einem Rangiervorgang in der Parklücke abgestellt.

Die vorgeschlagene Ontologie eignet sich zur Beschreibung von Verkehrsszenarien und erlaubt somit die Generierung eines Szenarienkatalogs. Sowohl transferierte als auch fiktiv generierte Fahrscenarien können Teil des Katalogs werden. Zudem ist es möglich komplexe dynamische Anteile der Szenarien, wie einen Einparkvorgang, mit Hilfe weniger Elemente Manöver-basiert zu spezifizieren. In Kapitel 4.3 wurde am Beispiel eines Sensorfehlers bereits dargestellt, wie eine iterative Modellerweiterung erfolgen kann.

Das maschinenlesbare XML-Format zur Beschreibung der Verkehrsszenarien ermöglicht zudem die Parametrierung einer Simulationsumgebung. Dabei erschwert die maschinenlesbare Form eine manuelle Erstellung der XML-Dokumente ohne zusätzlichen Editor. Für die praktische Umsetzung des Transfers realer Verkehrsszenarien in Szenariobeschreibungen gibt es unterschiedliche Optionen.

Einerseits können die zu beschreibenden Entitäten händisch dem Dokument zugefügt werden. Diese Art der Interaktion ist für den Entwickler jedoch nicht intuitiv, da aufgrund fehlender Visualisierungsmöglichkeit keine direkte

Bewertung erfolgen kann. Solange die manuelle Nachbearbeitung beim Transfer von Verkehrsszenarien benötigt wird, ist deshalb ein Editor notwendig. Primär muss ein solcher Editor schnelle Iterationen ermöglichen, indem Änderungen der Szenariobeschreibung visualisiert werden.

Alternativ kann die Generierung von Szenarien automatisiert erfolgen. Dabei kann der Fokus auf einem automatisierten Transfer realer Fahrscenarien liegen. Ziel hierbei ist es, möglichst ohne menschliches Zutun eine Beschreibung des Verkehrsgeschehens aus Sensormessdaten zu erhalten. Dazu ist eine Sensorik nötig, die eine vollständige Rekonstruktion der Fahrsituation erlaubt. Diese Anforderung an die Messtechnik entspricht nicht direkt den Anforderungen an die Fahrzeugsensorik eines autonomen Fahrzeugs. Deshalb kann der automatisierte Transfer in der Regel nur mit speziellen Messfahrzeugen realisiert werden. Wie durch den iterativen Absicherungsansatz zudem deutlich wird, sind zum Zeitpunkt der Messfahrten nicht immer alle Einflussparameter bekannt, sondern werden teilweise erst nachfolgend entdeckt. Diese Tatsache erschwert eine *vollständige* Rekonstruktion des Verkehrsgeschehens, wenn beispielsweise ein neu entdeckter Einflussparameter bisher nicht von der Referenzsensorik erfasst wurde.

Neben dem automatisierten Transfer realer Fahrscenarien kann ein alternativer Ansatz darin bestehen, Szenariobeschreibungen automatisiert zu generieren. Die Ergebnisse haben somit nicht zwangsläufig ein reales Pendant, wären aber aufgrund der zugrundeliegenden Ontologie in Realfahrten übertragbar. Für die Generierung von Verkehrsszenarien sind nachfolgend eine modellbasierte und eine probabilistische Anwendung dargestellt, deren Herleitung in Kapitel 3.3 präsentiert wurde.

Modellbasierte Testfallgenerierung

Die Anwendung des modellbasierten Vorgehens wird exemplarisch an einem Notbremsassistenten (engl. Emergency Brake Assist, EBA) durchgeführt: Dieser kann während eines Testdurchlaufs entweder *inaktiv* bleiben, eine *Warnung* oder eine *Bremsanforderung* emittieren. Außerdem ist er in eine Umgebung eingebettet, die aus dem EGO-Fahrzeug, einem Verkehrsfahrzeug und einer geraden Straße besteht. So spiegelt dieser Aufbau vereinfacht die Struktur des realen Einsatzortes der Unit-Under-Test wider, erlaubt die Einbeziehungen von Umgebungseinflüssen und ermöglicht den Einsatz der Szenariobeschreibung als Parametrierung des Tests. Abbildung 5.3 visualisiert den Aufbau des Systemmodells, wobei zusätzlich die Unit-Under-Test sowie die Testparameter dargestellt sind. Die Anwendung des in Kapitel 3.3 vorgestellten Vorgehens auf diesen Testaufbau ergibt folgenden Ablauf:

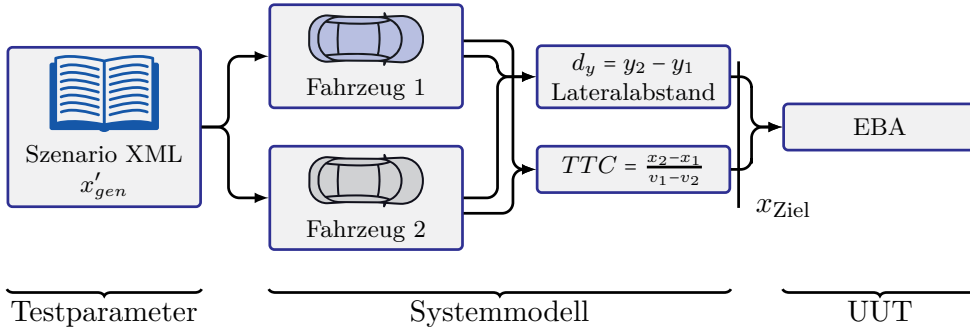


Abb. 5.3: Systemaufbau für das Anwendungsbeispiel der Testfallgenerierung

1. Die Eingangsdimensionen in den Notbremsassistenten sind der Lateralabstand d_y sowie die voraussichtliche Zeitdauer bis zu einer möglichen Kollision TTC .
2. Basierend auf den fiktiven Anforderungen des Notbremsassistenten muss dieser eine Notbremsung auslösen, wenn es in voraussichtlich weniger als 1 Sekunde zur Kollision mit einem Hindernis kommt. Stattdessen ist eine Warnung zu emittieren, falls eine Kollision in den nächsten 3 Sekunden zu erwarten ist oder der Lateralabstand d_y zu einem Hindernis in einem engen, aber nicht kollidierenden Bereich von $d_y \in (2 \dots 2,2]$ Metern liegt. Ansonsten darf keine Aktion vom Notbremsassistenten ausgehen. Basierend auf den Anforderungen lassen sich drei Äquivalenzklassen c_i^d je Eingangsdimension d herleiten:

$$c_0^{d_y} := \{d_y \in \mathbb{R} \mid |d_y| \leq 2\}$$

$$c_1^{d_y} := \{d_y \in \mathbb{R} \mid 2 < |d_y| \leq 2,2\}$$

$$c_2^{d_y} := \{d_y \in \mathbb{R} \mid |d_y| > 2,2\}$$

$$c_0^{TTC} := \{TTC \in \mathbb{R}^+ \mid 0 \leq TTC \leq 1\}$$

$$c_1^{TTC} := \{TTC \in \mathbb{R}^+ \mid 1 < TTC < 3\}$$

$$c_2^{TTC} := \{TTC \in \mathbb{R}^+ \mid TTC < 3\}$$

3. Für die Auswahl der Repräsentanten wird in diesem Beispiel je ein Wert jeder Äquivalenzklasse verwendet. Dadurch ergeben sich die $3 \cdot 3 = 9$

Zielzustände x_{Ziel}

$$\left\{ \begin{bmatrix} 1,9 \\ 0,9 \end{bmatrix}, \begin{bmatrix} 1,9 \\ 2,0 \end{bmatrix}, \begin{bmatrix} 1,9 \\ 3,5 \end{bmatrix}, \begin{bmatrix} 2,1 \\ 0,9 \end{bmatrix}, \begin{bmatrix} 2,1 \\ 2,0 \end{bmatrix}, \begin{bmatrix} 2,1 \\ 3,5 \end{bmatrix}, \begin{bmatrix} 2,3 \\ 0,9 \end{bmatrix}, \begin{bmatrix} 2,3 \\ 2,0 \end{bmatrix}, \begin{bmatrix} 2,3 \\ 3,5 \end{bmatrix} \right\}$$

Diese Zustände sind während der Closed-Loop Simulation zu erreichen, um die unterschiedlichen Reaktionen des Notbremsassistenten zu stimulieren. Im Rahmen einer Testaussage muss die erwartete Systemreaktion y_{erw} mit der Reaktion der Unit-Under-Test y unter Einhaltung der Zielzustände x_{Ziel} verglichen werden.

4. Zur Nutzung eines realen Closed-Loop-Systems unter Verwendung der Szenariobeschreibung als Testparametrisierung ist die Überführung der Zielzustände in die Parameter des Systemmodells notwendig. Für den Beispielfall kann diese Überführung mittels einer Invertierung des Systemmodells realisiert werden. Die Fahrzeuge des Systemmodells sind durch Punktmassenmodelle konstanter Beschleunigung und konstanter Querposition angenähert. Durch die Wahl der Randbedingungen $d_x(t=TTC) = 5 \text{ m}$, $v_{\text{Fahrzeug2}} = 3 \frac{\text{m}}{\text{s}}$, $x_{\text{Fahrzeug2}} = 100 \text{ m}$, $y_{\text{Fahrzeug2}} = 0 \text{ m}$ lässt sich die Invertierung analytisch lösen. Verwendet man als Szenarioparameter $x'_{\text{gen}} = [x_1, y_1, v_{1,\text{ziel}}, x_2, y_1, t(v_{2,\text{start}}), v_{2,\text{ziel}}]$, so ergeben sich aus der Invertierung des Systemmodells folgende 9 Testfälle

$$\left\{ \begin{bmatrix} 51,2 \\ -1,9 \\ 14,4 \\ 100 \\ 0 \\ 3,5 \\ 10 \end{bmatrix}, \begin{bmatrix} 76,7 \\ -1,9 \\ 12,0 \\ 100 \\ 0 \\ 1,6 \\ 10 \end{bmatrix}, \begin{bmatrix} 84,5 \\ -1,9 \\ 11,1 \\ 100 \\ 0 \\ 0,9 \\ 10 \end{bmatrix}, \begin{bmatrix} 51,2 \\ -2,1 \\ 14,4 \\ 100 \\ 0 \\ 3,5 \\ 10 \end{bmatrix}, \begin{bmatrix} 76,7 \\ -2,1 \\ 12,0 \\ 100 \\ 0 \\ 1,6 \\ 10 \end{bmatrix}, \begin{bmatrix} 84,5 \\ -2,1 \\ 11,1 \\ 100 \\ 0 \\ 0,9 \\ 10 \end{bmatrix}, \begin{bmatrix} 51,2 \\ -2,3 \\ 14,4 \\ 100 \\ 0 \\ 3,5 \\ 10 \end{bmatrix}, \begin{bmatrix} 76,7 \\ -2,3 \\ 12,0 \\ 100 \\ 0 \\ 1,6 \\ 10 \end{bmatrix}, \begin{bmatrix} 84,5 \\ -2,3 \\ 11,1 \\ 100 \\ 0 \\ 0,9 \\ 10 \end{bmatrix} \right\}$$

Die XML-basierten ontologischen Szenariobeschreibungen dieser Testvektoren sind in Anhang D zu finden.

Das beschriebene Vorgehen zeigt, wie man unter Verwendung und Invertierung eines Systemmodells Szenariobeschreibungen erhält, deren Ausführung zu den gewünschten Zielzuständen x_{ziel} führen sollte. Das verwendete Systemmodell ist eine Vereinfachung der Realität. Deshalb muss während der Ausführung der Szenariobeschreibungen die Erreichung der Zielzustände beobachtet und in die Testauswertung miteinbezogen werden. Abbildung 5.4 visualisiert die

sich aus der Simulation aller neun Szenarien ergebenden Fahrzeugtrajektorien. Zusätzlich ist der Zeitpunkt beim Eintritt des erwarteten Zielzustandes grau markiert. Es ist zu erkennen, dass beide Fahrzeuge sich entsprechend der Szenariobeschreibungen annähern. Fahrzeug 1 beginnt direkt zu beschleunigen, während Fahrzeug 2 an dessen Startposition bis zum Zeitpunkt $t(v_{2,start})$ wartet, um auf die Zielgeschwindigkeit $v_{2,ziel}$ zu beschleunigen.

Das zur Invertierung verwendete Systemmodell vereinfacht durch das verwendete Punktmassenmodell konstanter Beschleunigung die Fahrdynamik. Deshalb unterscheiden sich auch die vorhergesagten von den real eintretenden Trajektorien und Zielzuständen. Abbildung 5.5 stellt die vom invertierten Systemmodell vorhergesagten den in der Simulation eingetretenen Zuständen *TTC* gegenüber. Zusätzlich sind die Bereiche der Äquivalenzklassen eingetragen. Solange sich die ergebenden Trajektorien innerhalb der gewünschten Äquivalenzklasse befinden, ist das Systemmodell zur Generierung der Szenarien ausreichend. Dies ist in diesem Beispiel für alle Trajektorien der Fall. Bei komplexeren Trajektorien oder sehr schmalen Äquivalenzklassen sind die Anforderungen an das Systemmodell höher beziehungsweise die akzeptablen Abweichungen zwischen Vorhersage und Ausführung geringer. Gleichzeitig sind bereits für das simple Punktmassenmodell mehrere Randbedingungen bei der Invertierung nötig. Ist die Verwendung eines komplexeren Systemmodells notwendig, so ist dessen analytische Invertierung im Allgemeinen nicht möglich. Werden Opti-

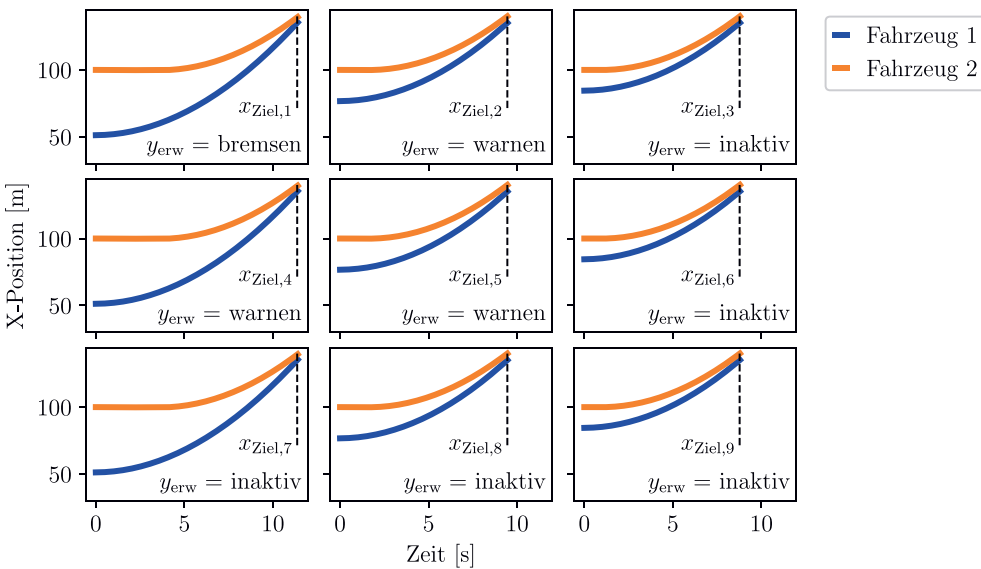


Abb. 5.4: Trajektorien zweier Fahrzeugmodelle für neun generierte Szenarien

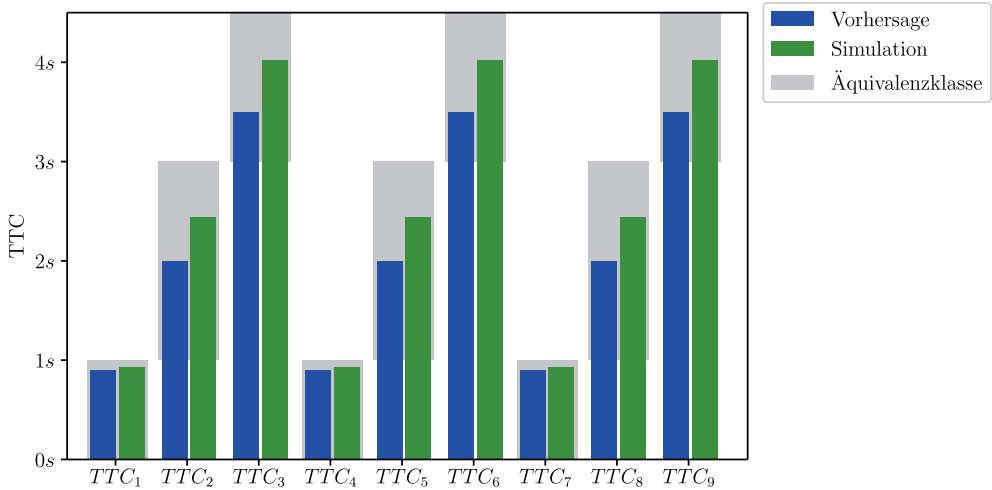


Abb. 5.5: Vergleich zwischen Vorhersage und Simulation des Zielzustandes TTC

mierungsverfahren genutzt, die keine Invertierung voraussetzen, so können auch komplexe Systemmodelle verwendet werden. Während des Optimierungsvorgangs wird das Modell mehrfach evaluiert. Durch die kurze Berechnungsdauer eines Systemmodelles entsteht im Vergleich zum real zu testenden System ein zeitlicher Vorteil bei der Findung passender Szenarioparameter.

Probabilistische Testfallgenerierung

Neben der systematischen Generierung von Testfällen kann auch ein probabilistischer Ansatz gewählt werden. Wie in Kapitel 3.3 beschrieben ist die direkte probabilistische Variation von Parametern auf der Szenariobeschreibung wenig sinnvoll, da ein Großteil der so generierten Szenarien unplausibel wäre. Deshalb wird ein *Limitierungsmodell* verwendet, in dessen Rahmen sich die Parametervariation bewegt. Das folgende Anwendungsbeispiel zeigt die Verwendung limitierender Annahmen zur probabilistischen Variation von Parametern.

Um die Variabilität zu begrenzen werden n_{Road} vordefinierte Straßenlayouts verwendet. Diese stammen aus bisherigen Testläufen und wurden aus realen Straßenverläufen abgeleitet. Zu Beginn des Vorgehens wird je Szenario zufällig einer dieser Straßenverläufe gewählt. Zusätzlich wird eine Liste vorhandener Umgebungsobjekte, Hintergrundgrafiken, Fahrzeuggeometrien und Straßenschilder verwendet. Mehrere Instanzen dieser Elemente werden zufällig relativ zum Straßenverlauf platziert. Um die Plausibilität der Szenarien zu wahren werden

alle Objekte kollisionsfrei zueinander platziert. Als dynamisches Element erhält jeder Akteur eine Zielgeschwindigkeit. Algorithmus 5 beschreibt das angewandte Vorgehen. Dieser Algorithmus wurde unter Verwendung von $n_{\text{Road}} = 14$

Algorithmus 5 Vorgehen zur probabilistischen Szenariengenerierung

- Require:** List L_{Road} of n_{Road} different road layouts
Require: List $L_{\text{Background}}$ of $n_{\text{Background}}$ different background maps
Require: List L_{Objects} of n_{Objects} different object geometries
Require: List L_{Geo} of n_{Geo} different actor geometries
Require: List L_{Signs} of n_{Signs} different road signs
- 1: Create an empty Scenario
 - 2: Add 1 random background map from $L_{\text{Background}}$
 - 3: Add 1 random road from L_{Road}
 - 4: Add terrain to the left and right of the road
 - 5: Add n_1 objects from L_{Objects} non-colliding randomly to the terrain
 - 6: Add 1 EGO-Vehicle
 - 7: Add n_2 actors from L_{Geo} non-colliding on the road
 - 8: **for all** actors **do**
 - 9: Add a target velocity $v_{\text{target}} \in [0; 180] \frac{\text{km}}{\text{h}}$
 - 10: **end for**
-

unterschiedlichen Straßenlayouts ausgeführt. Neben unterschiedlichen Hintergrundgrafiken und Umgebungsobjekten wurde zusätzlich zum EGO-Fahrzeug Verkehr mit einer Dichte im Bereich von $[0,8 \dots 32]$ *Fahrzeuge je Kilometer und Spur* hinzugefügt. Abbildung 5.6 zeigt exemplarisch 12 Visualisierungen der 1000 probabilistisch generierten Szenarien aus Sicht einer Kamera des EGO-Fahrzeugs. Zum Vergleich mit dem vorher ausgeführten modellbasierten Ansatz visualisiert Abbildung 5.7 den minimal am EGO-Fahrzeug auftretenden *TTC* Wert, unter Berücksichtigung eines Lateralabstandes $|d_y| \leq 2,2\text{m}$. Simuliert wurden jeweils 10 Sekunden der Fahrsituation. Der Anteil der Szena-



Abb. 5.6: Visualisierungen probabilistisch generierter Verkehrsszenarien

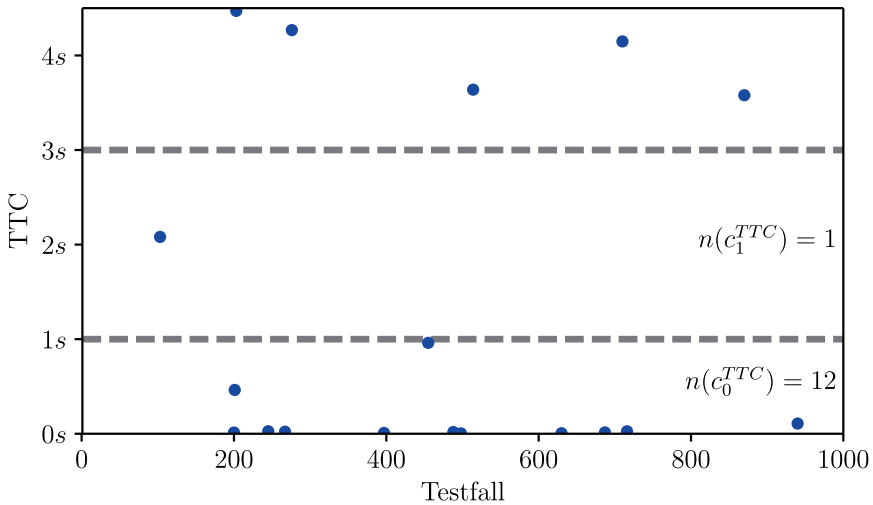


Abb. 5.7: Minimale TTC -Werte aus probabilistisch erstellten Szenarien

riobeschreibungen, bei dem ein Notbremsassistent reagieren müsste, liegt im Beispiel bei 1,3%. In 987 Szenarien ist keine Reaktion des Notbremsassistenten zu erwarten.

Dieses Beispiel verdeutlicht die unterschiedlichen Anwendungsszenarien der vorgestellten Ansätze zur Testfallgenerierung: Mithilfe des Systemmodells lassen sich im modellbasierten Ansatz alle Szenarien generieren, die für einen minimalen und vollständigen Test der Zielzustände notwendig sind. Diese systematische Konstruktion der Szenarien erfordert aufgrund der Modellierung und Invertierung detailliertes Wissen über die zu testende Komponente und deren Umfeld. Zudem ist die Invertierung nur für einfache Systemmodelle analytisch zu lösen. Im probabilistischen Ansatz werden die Zielzustände erst durch eine höhere Anzahl an Testfällen erreicht. Jedoch ist die Generierung der Szenarien einfacher und auch für komplexe Umgebungen möglich. Wie im Beispiel deutlich wird, ergibt sich bei der probabilistischen Generierung eine große Anzahl unterschiedlicher Szenarien. Hierdurch stellt sich im Vergleich zum modellbasierten Vorgehen eine breitere Abdeckung von Zuständen der Äquivalenzklassen ein. Beim probabilistischen Ansatz besteht zusätzlich die Herausforderung, dass die zu erwartende Reaktion y_{erw} im Falle komplexer Assistenzsysteme so allgemein formuliert werden muss, dass sie für alle automatisch generierten Szenarien definiert ist. Im gezeigten Beispiel wurde die zu erwartende Reaktion anhand der Äquivalenzklassenzuordnung der Zielzustände realisiert.

5.2. Simulationsarchitektur und Sensormodelle

In diesem Abschnitt wird der Nutzen der präsentierten Simulationsarchitektur im Hinblick auf deren Anwendbarkeit in XiL-Testumgebungen diskutiert. Anschließend werden Simulationsergebnisse der vorgestellten Sensormodelle präsentiert und analysiert.

HiL Testsystem eines Notbremsassistenten

Die vorgestellte Beschreibung von Verkehrsszenarien sowie die Simulationsarchitektur kann dazu verwendet werden, ein reales Steuergerät im Rahmen eines Hardware-in-the-Loop Aufbaus zu testen. Ein solcher Anwendungsfall deckt eine Vielzahl unterschiedlicher Anforderungen an das Gesamtsystem ab.

Einerseits müssen alle involvierten Simulationsmodelle die Echtzeitanforderungen des Systems erfüllen. Andererseits sollten bereits vorhandene Modelle wiederverwendet werden können. Physikalisch-modellierte Fahrzeugmodelle sind notwendig, um die Steuergeräte-internen Plausibilisierungen zu befriedigen. Hierzu gehört beispielsweise eine Fahrzeugbewegung, die keine Sprünge im Geschwindigkeitsverlauf aufweist. Außerdem sind interne Entscheidungen des Notbremsassistenten von weiteren Zustandsinformationen des Fahrzeugs abhängig. So wird etwa der aktuelle Lenkradwinkel zur Schätzung der Fahrzeugtrajektorie verwendet, um in Kurvenfahrten keine False-Positive Bremsauslösungen zu verursachen. Solche Informationen werden in HiL Anwendungen üblicherweise über eine sogenannte Restbus-Simulation der UUT zur Verfügung gestellt. Die synthetischen Daten für die Restbus-Simulation können größtenteils aus vorhandenen Fahrzeugsimulationsmodellen genutzt werden, wodurch mit deren Wiederverwendung Zeitersparnis beim Aufbau eines Testsystems einhergeht.

Die Szenariobeschreibung liefert Parameter für alle Modelle. So beinhaltet sie den Straßenverlauf, Startpositionen aller Objekte, Informationen zum Sensoreinbauort sowie die Beschreibung des dynamischen Ablaufs. Im Rahmen dieser Arbeit wurde ein HiL Testsystem für einen Radar-basierten Notbremsassistenten unter Verwendung der vorgestellten Simulationsarchitektur umgesetzt. Die dafür verwendeten Module und deren Interaktionen sind in Abbildung 5.8 abgebildet. Im linken Teil sind die Modelle des Ego-Fahrers, der Fahrdynamik sowie das reale Steuergerät aufgelistet. Abgesehen vom Sensor stellen diese Komponenten das Gesamtmodell des Eigenfahrzeugs dar. Weitere Verkehrsfahrzeuge sind einfacher modelliert und Teil der Verkehrskomponente. Alle Fahrzeuge übermitteln deren Position an das Umgebungsmodell. Das Storyboard steuert die Ausführung der dynamischen Abläufe, indem es basierend auf Informationen des Umgebungsmodells die Fahrermodelle mittels entsprechender Fahrhinweisen instruiert. Neben der Restbussimulation, die auf dem

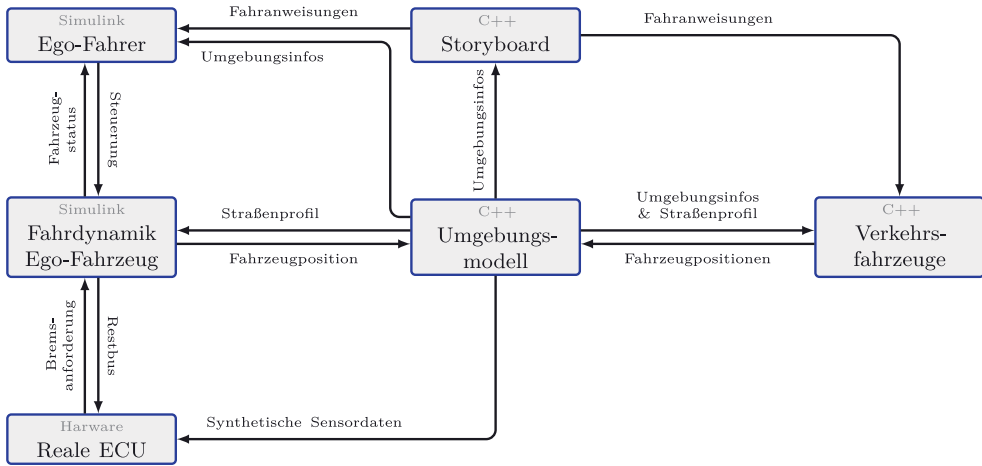


Abb. 5.8: Systematischer Aufbau des HiL Prüfstandes

Fahrdynamikmodell des Ego-Fahrzeugs basiert, erhält das reale Steuergerät synthetische Sensordaten vom Umgebungsmodell.

Das im HiL Aufbau verwendete Steuergerät bietet eine Schnittstelle auf Objektlistenebene. Mittels Nachrichten über den CAN-Bus werden die synthetisch generierten Objektlisten in das Steuergerät eingespeist, um die anschließende *Situationsanalyse* und *Anwendung* (in diesem Fall eine Notbremsfunktion) zu testen. Aufgrund der zur Verfügung stehenden Schnittstelle wird für den HiL Aufbau das ideale Sensormodell verwendet.

Abbildung 5.9 visualisiert zwei Fahrzeugtrajektorien, die sich bei der Ausführung eines HiL Tests ergeben. Das Szenario besteht aus einem EGO-Fahrzeug (Fahrzeug 1) mit idealem Objektlistensensor sowie einem Verkehrsfahrzeug (Fahrzeug 2). Das Verkehrsfahrzeug startet 30 Meter vor dem EGO-Fahrzeug und beschleunigt aus dem Stand auf rund $40 \frac{\text{km}}{\text{h}}$. Das EGO-Fahrzeug beginnt nach 3 Sekunden auf dessen Zielgeschwindigkeit von rund $70 \frac{\text{km}}{\text{h}}$ zu beschleunigen. Die Notwendigkeit, die Fahrt des Eigenfahrzeugs aus dem Stillstand zu beginnen, resultiert aus internen Plausibilisierungsroutinen des Steuergerätes. Ist das Eigenfahrzeug bei Beginn eines Szenarios und nach dem Einschalten des Steuergerätes nicht im Stillstand, geht das Steuergerät direkt in den Fehlermodus und ein Test der *Situationsanalyse* und *Anwendung* ist nicht möglich³⁰.

Im Rahmen des durchgeführten Tests emittiert das Steuergerät bei einer

³⁰Für zukünftige Entwicklungen von ADAS-Steuergeräten sollte eine Option zur vollständigen Initialisierung aller internen Parameter geschaffen werden. Dadurch würde die Notwendigkeit entfallen alle Zielzustände anzusteuern. Dies hätte einen deutlichen Effizienzgewinn beim Testen realer Steuergeräte zur Folge.

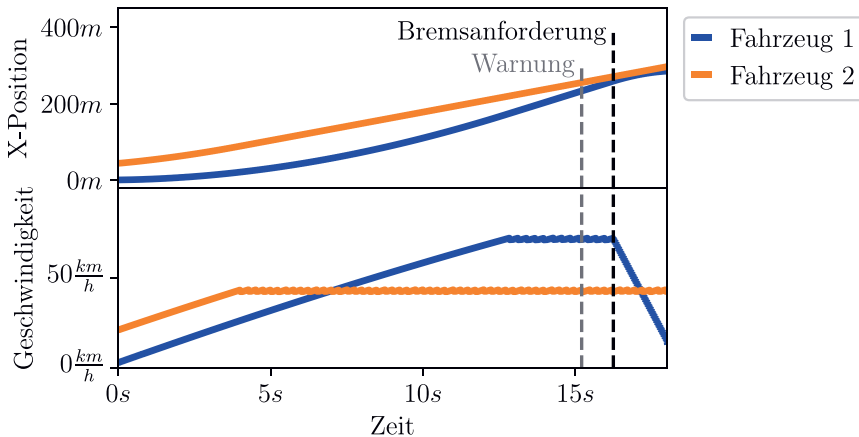


Abb. 5.9: Trajektorien zweier Fahrzeuge eines HiL Testlaufs

nahenden Kollision bei $t \approx 15$ s eine Warnung. Da sich das Fahrzeug anschließend ohne Reaktion des Fahrermodells weiterbewegt, wird kurz darauf automatisiert eine Notbremsung eingeleitet.

Die Ausführung dieses HiL Tests zeigt die Möglichkeit, wie mittels der vorgestellten Simulationsarchitektur, unter Wiederverwendung bestehender Modelle und einfacher Parametrierung durch die Szenariobeschreibung, der Test eines realen Steuergeräts realisiert werden kann.

Das im HiL Aufbau genutzte Umgebungsmodell fügt sich aufgrund der verwendeten Softwarearchitektur problemlos in die Struktur bisheriger Testsysteme ein. Vorhandene Simulationsmodelle und damit auch die Restbussimulation können wiederverwendet werden. Die Verantwortlichkeit des Umgebungsmodells beschränkt sich auf die synthetischen Sensormodelle, wobei zusätzliche Sensoren als Informationslieferanten für Fahrer- und Fahrdynamikmodelle realisiert worden sind. Der vorgestellte Gesamtaufbau erfüllt außerdem die Echtzeitanforderungen der realen Hardware. Die gemessenen Simulationszeiten der Modellteile *Storyboard*, *Fahrzeuge*, *Umgebung* sowie die zusätzlichen Laufzeiten der sequentiellen Gesamtausführung sind in Abbildung 5.10 aufgeführt. Der Modellteil *Fahrzeuge* beinhaltet zwei Fahrzeug- und Fahrer-Modelle. Dabei sind in blau jeweils die minimalen und in orange die maximalen Laufzeiten visualisiert, die während der Gesamtsimulation von $t = 10$ Sekunden, bei einer Schrittweite von $\Delta t = 1$ ms, auftraten. Im schlechtesten Fall würde die sequentielle Ausführung aller Modelle rund $70 \mu\text{s}$ betragen. Diese Konstellation ist während der Simulation nicht eingetreten. Hier lag die Ausführungsdauer zwischen $9 \mu\text{s}$ und $33 \mu\text{s}$, bei einem Mittelwert von $10 \mu\text{s}$. Das reale Steuergerät erwartet alle 40 ms eine aktualisierte Objektliste. Die Berechnungsdauer der

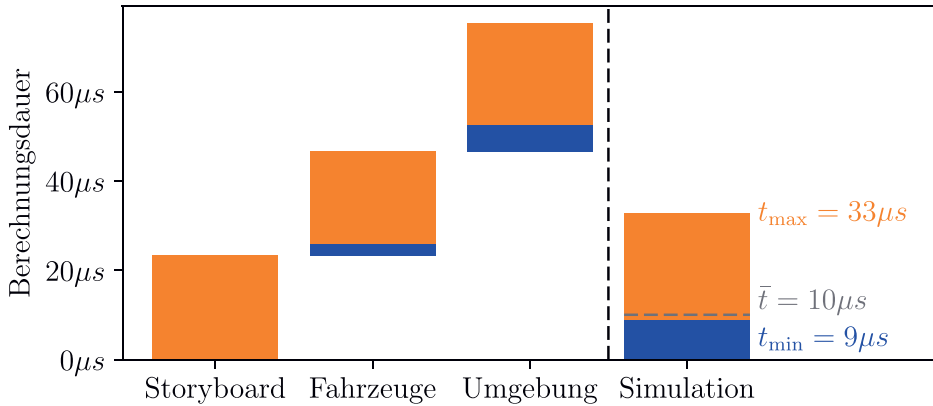


Abb. 5.10: Simulationszeiten involvierter Modelle

Gesamtsimulation liegt somit deutlich unterhalb des zur Verfügung stehenden Zeitfensters, das durch den realen Notbremsassistenten vorgegeben wird, und eignet sich dadurch für den HiL Einsatz.

Eine Anmerkung zur Schwankung der Modelllaufzeiten hinsichtlich der Echtzeitfähigkeit: Im HiL Kontext gelten Echtzeitanforderungen. Dies bedeutet, dass die Berechnungsergebnisse innerhalb eines vorgegebenen Zeitfensters zur Verfügung stehen müssen. Ein echtzeitfähiges Betriebssystem garantiert, dass es keine zusätzlichen Prozesse gibt, die nicht-deterministisch die Performance des Gesamtsystems beeinflussen. Ein Beispiel eines solchen Prozesses wäre ein zufällig eintretender Update-Vorgang. Diese Garantie führt dazu, dass ein Programm bei den selben Eingabewerten stets dieselbe Laufzeit hat. Die in Abbildung 5.10 sichtbaren Unterschiede zwischen minimalen und maximalen Laufzeiten der Modelle stehen dazu nicht im Widerspruch, wie das Beispiel in Quellcode 5.2 verdeutlicht.

Quellcode 5.2: Funktion mit wechselnder Komplexität

```

1  int getValue(double simulationTime) {
2      int result = 0;

4      if (simulationTime > 10.0) {
5          for (int i=0; i<simulationTime; i++) {
6              result = result + i;
7          }
8      }

10     return result;
    }

```

Für die Eingabewerte `simulationTime` ≤ 10.0 wird der Wert 0 zurückgegeben und die Komplexität der Funktion `getValue` ist konstant ($\mathcal{O}(1)$). Andernfalls ergibt die Bedingung in Zeile 4 *wahr* und die Komplexität der Funktion ist $\mathcal{O}(n)$. Bei der Visualisierung von 3D Objekten kommen eine Vielzahl von Performance-Optimierungen zum Einsatz. So werden beispielsweise im Rahmen eines *Culling* Schritts diejenigen Objekte aus nachfolgenden Betrachtungen entfernt, die nicht im Sichtfeld der Kamera zu finden sind. Deshalb ist für das Umgebungsmodell mit einer Schwankung der Laufzeit für einzelne Berechnungsschritte zu rechnen. Diese ist abhängig vom aktuellen Szenario und den Objekten, die sich im Sichtfeld der virtuellen Sensoren befinden, und somit nicht allgemein vorherzusagen. Gleichzeitig garantiert ein Echtzeitsystem jedoch eine stets gleiche Ausführungsdauer mehrerer Simulationsläufe bei gleichen Eingabeparametern.

Anwendung der Simulationsarchitektur zur verteilten Sensorsimulation

Die in Kapitel 4.1 entwickelte Architektur zur Simulation von Assistenzsystemen entkoppelt die Verantwortlichkeiten zwischen dem Umgebungsmodell zur Sensorsimulation und anderen Simulationsmodulen. Die Klassenstruktur des *SimulationObject* erlaubt die simultane Ausführung unterschiedlicher Sensorinstanzen auf einem konsistenten Szenegraphen. Als Schnittstelle zur Parallelisierung bieten sich die unterschiedlichen Sensorinstanzen an. Jede Recheneinheit wird über die Änderungen im Szenegraphen benachrichtigt, passt alle entsprechenden *SimulationObject*-Instanzen an und simuliert den jeweils zugeordneten Sensor. So ist es möglich Instanzen des idealen sowie des Raytracing-basierten Multi-Sensormodells zu parallelisieren. Abbildung 5.11

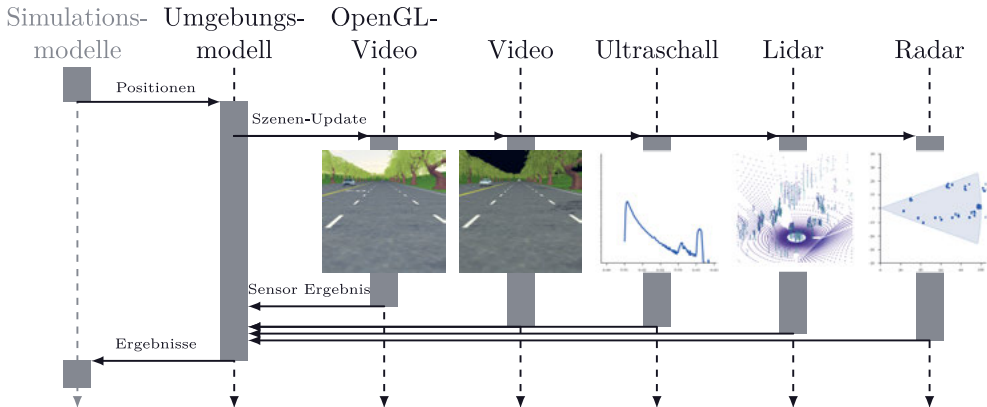


Abb. 5.11: Ablaufdiagramm für verteilte Sensorsimulationen

visualisiert das Ablaufdiagramm einer konsistenten Umgebungssimulation für fünf unterschiedliche Sensorinstanzen. Basierend auf den von externen Simulationsmodellen berechneten Positionen der dynamischen Akteure aktualisiert das Umgebungsmodell die dreidimensionale Repräsentation der Szene. Dieses Szenen-Update wird an die fünf Sensorinstanzen kommuniziert, worauf diese dieselben Änderungen an den Instanzen der *SimulationObjects* vornehmen. Dadurch stehen konsistente Daten des aktuellen Simulationszeitpunktes allen Sensorinstanzen zur Verfügung. Daraufhin erfolgt jeweils die Sensorsimulation, wobei Abbildung 5.11 exemplarisch die Daten eines Simulationszeitpunktes visualisiert. Zu sehen ist das OpenGL-basierte sowie das Raytracing-basierte Videobild. Dabei haben beide Sensormodelle die gleichen Kameraparameter, wie Position und Brennweite. Zudem werden in beiden Fällen eine mehrspurige Straße, Bäume sowie dreidimensionale Fahrzeugmodelle verwendet.

Während für den OpenGL-basierten Videosensor eine Hintergrundgrafik dargestellt wird, fehlt diese beim Raytracing-basierten Videosensor. Dieser hingegen stellt die Beleuchtungssituation inklusive Schatten dar, die im OpenGL-basierten Videosensor nicht vorhanden sind. Diese Unterschiede in der Darstellung eines Videosensors unterstreichen die Notwendigkeit, dass alle benötigten Features in die separaten Implementierungen eines synthetischen Sensors transferiert werden müssen.

Zusätzlich ist die Impulsantwort des Ultraschallsensormodells für diesen Simulationszeitpunkt dargestellt. Außerdem ist eine Lidar-Punktwolke visualisiert. Hierbei stellt jeder Datenpunkt die Reflexion des Laserscanners mit der virtuellen Umgebung dar, wobei jeder Punkt aus dem relativen Abstand in den drei Raumkoordinaten und der Reflexionsintensität besteht. Die fünfte dargestellte Visualisierung zeigt das Signal des synthetischen Radarsensors.

Hierbei sind die Reflexzentren an Objekten der virtuellen Umgebung zu sehen.

Das Beispiel veranschaulicht die Möglichkeit mit Hilfe der vorgestellten Simulationsarchitektur mehrere Sensoren zu instanzieren, die auf einer konsistenten Darstellung der virtuellen Umgebung arbeiten und eine parallele Ausführung erlauben.

Laufzeit Analyse

Sowohl zur Abschätzung der Simulationsdauer als auch für den HiL Anwendungsfall ist die Ausführungsdauer der unterschiedlichen Sensormodelle interessant. Abbildung 5.12 liefert hierzu einen Überblick. Die Messungen wurden auf dem in Anhang F beschriebenen System ausgeführt. Alle verwendeten Szenarien bestehen aus einer Vielzahl an Objekten, wobei Optimierungen hinsichtlich der Objektinstanziierung aus Gründen der Vergleichbarkeit explizit deaktiviert wurden. Zu sehen ist im linken Diagramm die logarithmisch aufgetragene Ausführungsdauer unterschiedlicher Sensormodelle in Abhängigkeit zur Polygonanzahl innerhalb der virtuellen Umgebung. Aufgrund von Laufzeiten des idealen Sensormodells im Mikrosekundenbereich liegt dessen Kurve unterhalb des dargestellten Wertebereichs. Die untere blaue Linie zeigt die Simulationsdauer des OpenGL basierten Videosensormodells. Im Vergleich zum Raytracing-basierten Videomodell ist dessen Laufzeit bei geringer Polygonanzahl um Faktor drei bis vier schneller. Wie auch im rechten Diagramm zu sehen ist, skaliert das Videosensormodell im Vergleich zu allen anderen Ansätzen schlechter in Bezug auf die Polygonanzahl. Erklären lässt sich die bessere Skalierbarkeit der Raytracing-basierten Modelle aufgrund der angewandten Bounding Volume Hierarchy Optimierung der Schnittpunktberechnung. Hierdurch wird der Aufwand der Schnittpunktanfrage für n Polygone auf $\mathcal{O}(\log(n))$ reduziert. Die Lidar-, Radar- und Ultraschall-Modelle liegen

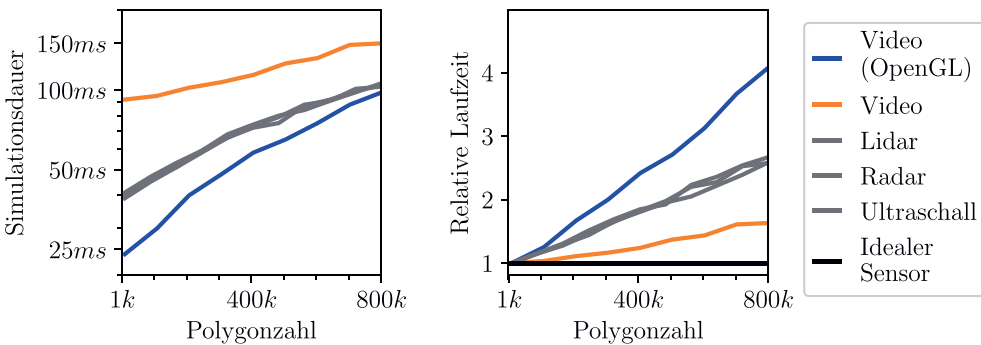


Abb. 5.12: Simulationsdauer in Abhängigkeit der Polygonanzahl

zwischen den beiden Videosensormodellen, was aus der reduzierten Anzahl an generierten Primärstrahlen im Vergleich zum Raytracing-basierten Videomodell resultiert. Das ideale Sensormodell ist nicht abhängig von der Polygonanzahl der Objekte innerhalb des Szenarios, da dieses auf abstrahierten Elementen arbeitet. Dies zeigt sich an dessen relativer Laufzeit im zweiten Diagramm von Abbildung 5.12.

Simulation eines Ultraschallsensors mit Hilfe des Multi-Sensormodells

Zur Validierung des Multi-Sensormodells werden reale Labormessungen eines Ultraschallsensors verwendet. Der Messaufbau sieht dabei folgendermaßen aus: Im Messraum steht ein Testobjekt definierter Größe an einem definierten Ort. Hinter dem Messobjekt sind Schall-absorbierende Wände angebracht. Von diesen Wänden ist keine Reflexion zu erwarten. Der Ultraschallsensor wird entlang einer Geraden am Testobjekt vorbei bewegt. Dabei werden Zeit, Position des Sensors sowie die ausgegebenen Primär- und Sekundärdistanzen aufgezeichnet. Die Primärdistanz ist derjenige Abstand, der einem ersten lokalen Maximum der Impulsantwort zugeordnet werden kann. Die Sekundärdistanz ist derjenige Abstand, der einem zweiten lokalen Maximum der Impulsantwort zugeordnet werden kann. Phänomenologisch erklären sich diese Abstände einerseits aus der direkten Reflexion des Impulses bei Sichtkontakt zum Objekt, andererseits aus der indirekten Reflexion des Impulses über den Boden. Aufgrund des Einbauorts der Ultraschallsensoren im Fahrzeug sowie der Annahme, dass sich in der Regel eine Schall-reflektierende Oberfläche unterhalb des Fahrzeugs befindet, wird stets versucht ein zweites Maximum der Impulsantwort zu detektieren. Abbildung 5.13 visualisiert zwei der beschriebenen Messsituationen, wobei ein Zylinder sowie ein Quader als Testobjekte dienen. Zusätzlich sind der Bewegungspfad sowie der Sichtbereich des Ultraschallsensors angedeutet.

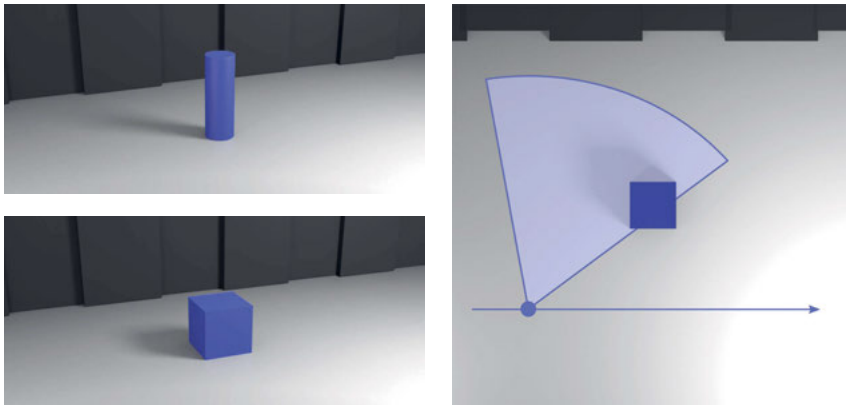


Abb. 5.13: Visualisierungen der Messsituation

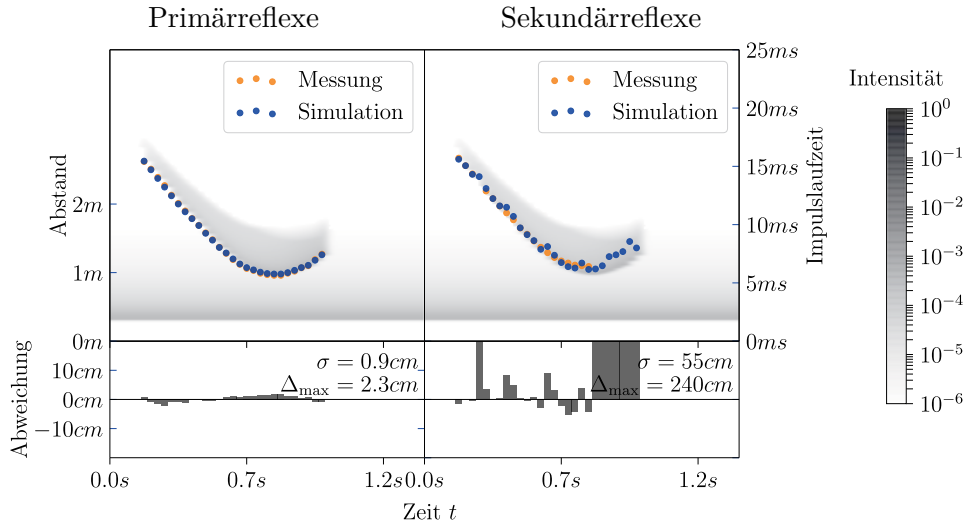


Abb. 5.14: Validierung des Ultraschall-Sensormodells (Zylinder)

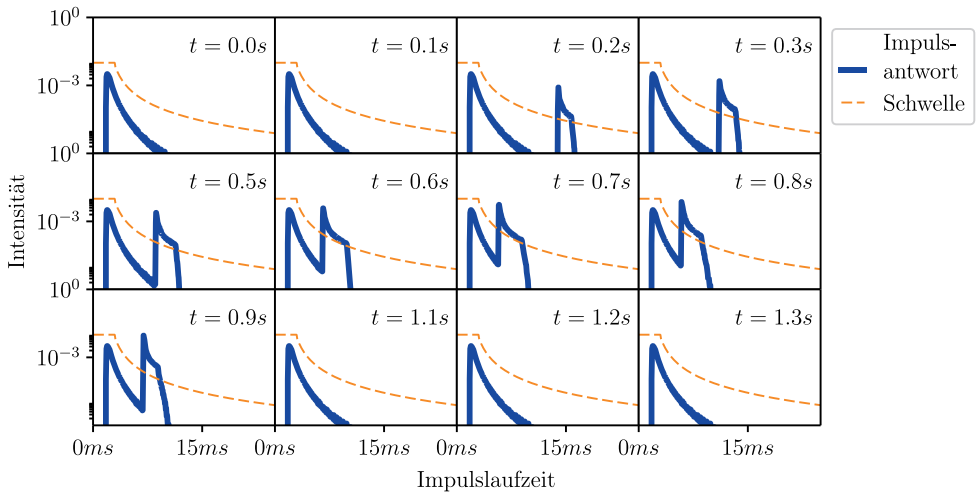


Abb. 5.15: Impulsantworten des Ultraschall-Sensormodells (Zylinder)

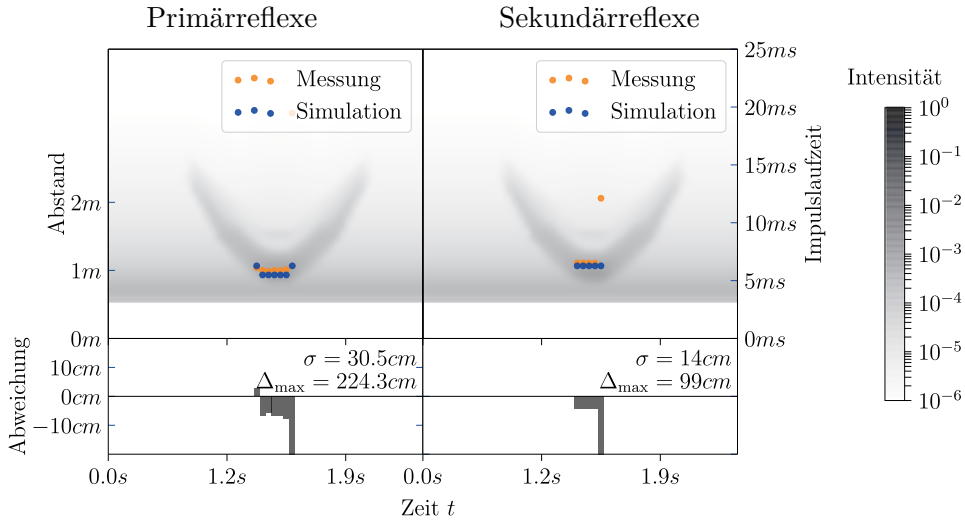


Abb. 5.16: Validierung des Ultraschall-Sensormodells (Quader)

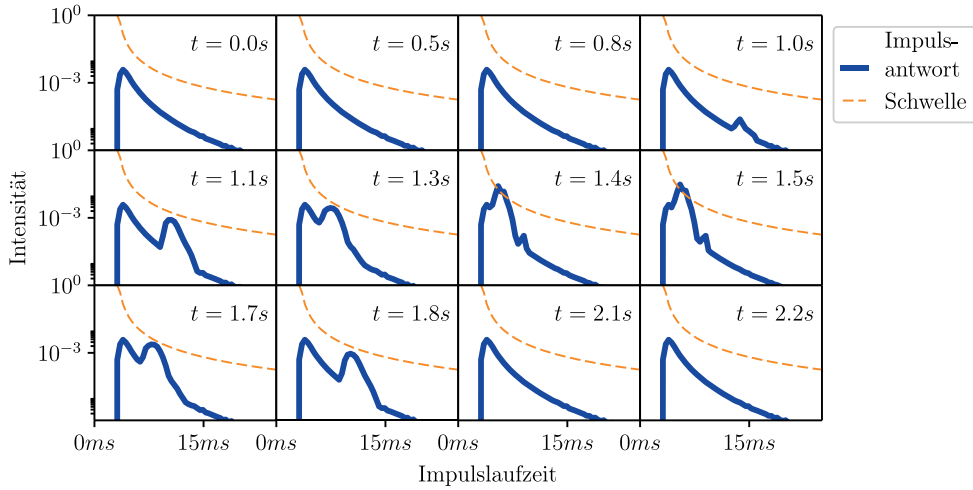


Abb. 5.17: Impulsantworten des Ultraschall-Sensormodells (Quader)

Die Abbildungen 5.14 bis 5.17 visualisieren die Mess- und Simulationsergebnisse beider Testaufbauten. Hierbei sind jeweils die Primär- und Sekundärreflexe der Messung und Simulation dargestellt sowie die Abweichung zwischen realen und synthetischen Daten aufgetragen. Zusätzlich sind die synthetischen Impulsantworten des Ultraschall-Sensormodells für je zwölf Simulationszeitpunkte dargestellt. Als Signalverarbeitungs-Modul wurde beim zylindrischen Testobjekt die Schwellwertkurve

$$s(t_{\text{Impuls}}) = \begin{cases} 10^{-2} & \text{wenn } t_{\text{Impuls}} < 3 \text{ ms} \\ \frac{10^{-2}}{1,3 \cdot (t_{\text{Impuls}} - 2)^{2,2}} & \text{sonst} \end{cases}$$

abgezogen und anschließend die ersten zwei Maxima gesucht. Hierdurch ergeben sich die Primär- und Sekundärdistanzen, die sich für einen Vergleich mit den Messdaten eignen. Die Messsituationen beinhalten den Zylinder beziehungsweise den Quader, der zusammen mit dem Boden und der Schall-absorbierenden Wand in eine virtuelle Umgebung überführt wurde.

Für die Simulation mit zylindrischem Messobjekt werden zu Beginn weder synthetische noch reale Primär- oder Sekundärreflexe detektiert. Anschließend nähert sich der Sensor weiter dem Testobjekt an, so dass dieses bei einem Abstand von etwa 2,5 m in den Sichtbereich des Sensors eintritt und Reflexe detektiert werden. Bei etwa 0,8 s nähert sich der Sensor dem Testobjekt bis zu einem Meter an, anschließend entfernt er sich wieder. Die asymmetrische Punkteverteilung lässt sich aus dem Azimutwinkel erklären, mit dem sich der Sensor relativ zum Testobjekt entlang einer geraden Trajektorie bewegt. Für die Primärdistanzen gibt es zu jedem Messpunkt einen nahegelegenen Simulationspunkt. Die Standardabweichung hinsichtlich der Primärdistanzen bewegt sich mit $\sigma = 0,9 \text{ cm}$ in einem sehr guten Bereich, der sich im Rahmen der Messgenauigkeit aktueller Ultraschallsensoren [Robert Bosch GmbH 2018] befindet.

Beim zylindrischen Objekt erfasst das reale System die Sekundärreflexe in einem kleineren Bereich als das Simulationsmodell. So gibt es einzelne Messpunkte, bei denen im Versuch keine Sekundärdistanz ermittelt werden konnte. In der Simulation wird für jeden Messpunkt, sobald das Objekt sich im Sichtbereich befindet, auch ein Sekundärreflex ausgegeben. Dieser ergibt sich aus dem kürzesten Reflexionspfad, der sich vom Sender, über den Boden, das Testobjekt und den Empfänger erstreckt, unter Einbeziehung einer minimal notwendigen Signalintensität, die den Empfänger erreicht.

Auch für die zweite Messsituation mit Quader liegt die Standardabweichung noch in einem guten Bereich. Einzelne grobe Messabweichungen werden vom synthetischen Simulationsmodell nicht wiedergegeben. An diesen Punkten liegt deshalb auch die größte Abweichung zwischen Mess- und Simulationswert.

Das Beispiel zeigt die Anwendungsmöglichkeiten des Multi-Sensormodells. So lassen sich die Sensorsignale für einzelne Szenarien sehr genau in der Simulation reproduzieren. Diese Szenarien können anschließend in der Simulation variiert werden, um beispielsweise erste qualitative Untersuchungen hinsichtlich der Auswirkungen unterschiedlicher Einbauwinkel oder Sichtbereiche von Sensoren zu untersuchen. Anschließend können einzelne der virtuell generierten Szenarien zur Validierung mit einem realen Pendant herangezogen werden. Dieses Vorgehen zur Validierung der Sensormodelle passt zur vorgestellten Absicherungsstrategie. Auch hier dienen reale Testfahrten als Grundlage zur Validierung der virtuellen Testfahrten, wobei die Sensormodelle eine von mehreren Simulationskomponenten darstellen, deren Validität es nachzuweisen gilt.

Synthetische Videobilder zur Absicherung von Assistenzsystemen

Die vorgestellte Simulationsarchitektur unter Verwendung der Sensormodelle ermöglicht auch die Generierung synthetischer Sensorsignale für Videokameras. So kann die funktionale Wirkkette von Assistenzsystemen ab der *Objekterkennung* in Tests einbezogen werden. Dadurch lassen sich synthetisch erzeugte Videobilder zur Stimulation der *Objekterkennung* verwenden.

Für Videobilder werden als Objekterkennungsalgorithmen aktuell vorrangig Ansätze unter Verwendung neuronaler Netze fokussiert. Deshalb basiert die folgende Untersuchung ebenfalls auf der Verwendung eines neuronalen Netzes als Objekterkennungsalgorithmus. Hierfür wurde das Framework [TensorFlow] sowie der Objekterkennungsalgorithmus Faster R-CNN verwendet [Vgl. Google LLC 2018c]. Dieser wurde mittels realer Videobilder des [KITTI 2018]-Datensatzes trainiert.

Die Stimulation des Objekterkennungsalgorithmus erfolgt schließlich mit realen und synthetischen Daten, um die Qualität der synthetischen Videobilder zu bewerten. Abbildung 5.18 zeigt exemplarisch je sechs reale und synthetische Bilder sowie die überlagerten Objekterkennungsbereiche. Hierbei wurden alle realen Szenarien in die virtuelle Welt überführt und auf beide Bildersets der gleiche Objekterkennungsalgorithmus angewandt. Grundsätzlich lässt sich erkennen, dass die synthetischen Bilder eine Stimulation des auf Realdaten trainierten Objekterkennungsalgorithmus erlauben. In beiden Datensätzen werden Fahrzeuge erkannt, wobei der Algorithmus häufig die gleichen Objekte im synthetischen sowie im realen Bild identifiziert. Die in der untersten Zeile aufgeführten Beispiele zeigen auf synthetischen Daten eine False-Negative und False-Positive Klassifizierung. In Bild 3-C wird ein Fahrzeug nicht erkannt, welches auf dem realen Pendant (3-A) als Fahrzeug klassifiziert wird. In Bild

3-D wird der blaue Van als Fahrzeug klassifiziert, der in Bild 3-B nicht als Fahrzeug klassifiziert wird.

Der Algorithmus ist auf mehrere Objekttypen trainiert. Hierzu gehören Fahrzeuge, Fußgänger und Busse, wobei der in Bild 3-D abgebildete Van ein Grenzfall zwischen Fahrzeug und Bus darstellt.

Ziel der synthetischen Daten ist es nicht, eine möglichst gute Klassifizierung von Objekten auf synthetischen Daten zu ermöglichen, sondern Bilder zu erzeugen, die für den Klassifikator den realen Bildern ähneln. Die in Abbildung 5.18 dargestellten Beispiele vermitteln einen ersten Eindruck, in welchen Grenzen sich die Stimulation von Objekterkennungsalgorithmen mittels synthetischer Daten bewegt. Zwar kann grundsätzlich eine Stimulation stattfinden, jedoch kann weder die Aussage getroffen werden, dass ein Objekterkennungsalgorithmus stets besser oder schlechter auf synthetischen als auf realen Daten arbeitet. Da es sowohl False-Positive als auch False-Negative Erkennungen gibt, darf eine quantitative Bewertung des Objekterkennungsalgorithmus nicht ausschließlich auf synthetischen Daten erfolgen. Diese können lediglich einen groben Richtwert liefern.

Neben dem direkten Transfer einzelner Szenarien lässt sich ein Vergleich der Erkennungseigenschaften auch auf größeren, nicht-korrelierten Datensätzen ziehen. Hierfür werden neben 400 Bildern des [KITTI 2018]-Datensatzes zusätzlich 400 synthetisch erzeugte Bilder zufällig generierter Szenarien verwendet. Für beide Datensätze stehen die Ground Truth Daten erkannter Fahrzeuge zur Verfügung. Durch die Variation des Schwellwerts, ab welcher Erkennungsgüte ein Objekt als erkannt gewertet wird, lässt sich das in Abbildung 5.19 dargestellte Diagramm ableiten. Diese als Receiver Operating Characteristics (ROC) bezeichnete Kurve stellt den Anteil an True-Positive Erkennungen dem Anteil an False-Positive Erkennungen gegenüber. Hiermit lässt sich die Qualität von Klassifikatoren vergleichen. Als gut wird ein Klassifikator dann bewertet,

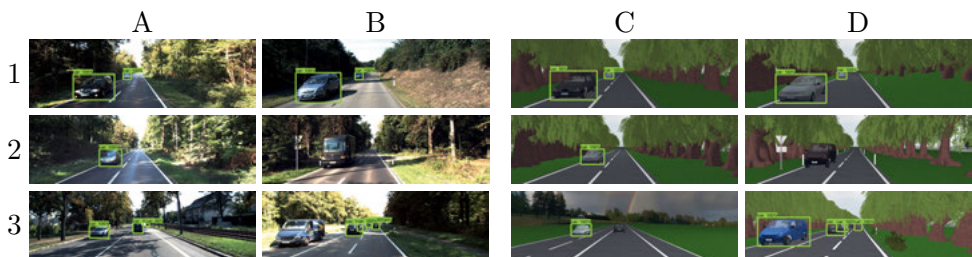


Abb. 5.18: Exemplarische Objekterkennung auf transferierten Szenarien, basierend auf [KITTI 2018]

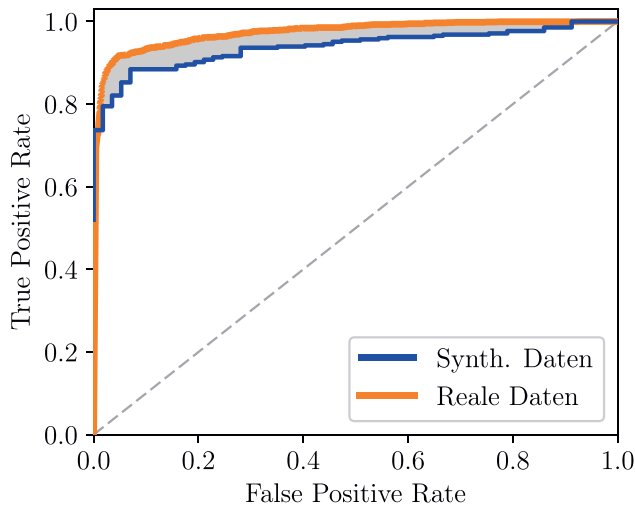


Abb. 5.19: Receiver Operating Characteristic eines Klassifikators für synthetische und reale Daten

wenn ein möglichst hoher Anteil an True-Positive Klassifikationen bei wenigen False-Positive Klassifikationen vorliegt. Ein idealer Punkt befände sich links oben im Diagramms.

Zur Bewertung der Qualität synthetischer Daten kann dieses Diagramm nun dahin interpretiert werden, wie stark sich der Kurvenverlauf eines Algorithmus für synthetische und reale Daten ähnelt. Ziel möglichst guter synthetischer Daten ist die Minimierung der Fläche zwischen den Kurven. Wie in Abbildung 5.19 zu erkennen ist, zeigen beide ROC-Kurven einen ähnlichen Verlauf. Der grau visualisierte Flächeninhalt zwischen den Kurven beträgt $A = 0,038$.

Dies erlaubt die Stimulation eines Objekterkennungsalgorithmus mittels synthetischer Daten im Rahmen von qualitativen Regressionstests. Würde sich der Kurvenverlauf auf synthetischen Daten nach Anpassungen des Algorithmus deutlich ändern, so wäre dies ein Indiz dafür, dass eine ähnliche Änderung auch auf Realdaten zu erwarten ist.

Zusammenfassung

In diesem Kapitel wurden Elemente der Absicherungsstrategie angewandt und bewertet, die vorgestellte Simulationsarchitektur in ein HiL Testsystem integriert und analysiert sowie Validierungen der Sensormodelle vorgenommen. Die ontologische Beschreibung ermöglicht es reale Fahrscenarien in virtuelle Umgebungen zu überführen. Dies konnte durch exemplarische Beschreibungen und Transfers mehrerer realer Messfahrten dargestellt werden. Zudem lässt sich das Storyboard zur Definition komplexer Manöver verwenden, wie am dynamischen Einparkmanöver präsentiert. Unter Verwendung der Szenariobeschreibung wurden zwei Ansätze zur Generierung von Testfällen angewandt. Beim modellbasierten Vorgehen konnte eine vollständige Testabdeckung der aufgestellten Äquivalenzklassen bei minimaler Anzahl notwendiger Testfälle generiert werden. Das probabilistische Vorgehen ist demgegenüber deutlich einfacher anzuwenden, da kein invertierbares Systemmodell notwendig ist. Mit der hohen Anzahl an Testfällen kann dabei eine breitere Testabdeckung erzielt werden. Um Repräsentanten aller Äquivalenzklassen des Beispielfalls zu erhalten, waren im probabilistischen Ansatz jedoch sehr viel mehr Testfälle nötig.

Ein reales HiL Testsystem wurde realisiert, um die Anwendbarkeit der vorgestellten Simulationsarchitektur zu demonstrieren. Hierbei konnten bereits existierende Modellteile integriert und mit dem Umgebungsmodell verknüpft werden, um die Stimulation eines realen Notbremsassistenten zu ermöglichen. Die Ausführungsdauer der involvierten Modelle blieb deutlich unterhalb des erlaubten Maximum von 40 Millisekunden. Für das Multi-Sensormodell wurden reale Ultraschall-Messsituationen virtualisiert und Messdaten den Simulationsergebnissen gegenübergestellt. Hierdurch konnte für ein vereinfachtes Szenario der Validierungsaspekt der Absicherungsstrategie für einzelne virtuelle Testfahrten dargestellt werden. Abschließend wurde die Stimulation eines Objekterkennungsalgorithmus durch synthetisch generierte Videobilder untersucht. Dabei gelang die Stimulation des auf Realdaten trainierten neuronalen Netzes mittels synthetischer Videobilder. Jedoch traten hierbei sowohl False-Positive als auch False-Negative Erkennungen im Vergleich zu den realen Pendanten auf. Dies schließt eine detaillierte, quantitative Bewertung des Algorithmus auf synthetischen Daten aus. Zur qualitativen Bewertung wurde der Vergleich der Receiver Operating Characteristic eines Algorithmus zwischen realen und synthetischen Daten eingeführt. Durch die Interpretation des Flächeninhalts zwischen beiden Kurven als Performance-Index lassen sich größere Abweichungen im Rahmen von Regressionstests bereits auf synthetischen Daten identifizieren.

6. Zusammenfassung und Ausblick

6.1. Zusammenfassung

Die statistisch motivierte Betrachtung der Absicherung von Assistenzsystemen erlaubt den klaren Schluss, dass dieser Ansatz aus zeitlichen und wirtschaftlichen Aspekten nicht praxistauglich ist. Gleichzeitig versagt der Versuch, Spezifikationen autonomer Fahrfunktionen zu Beginn einer Entwicklung vollständig zu erfassen. Hierfür sind die relevanten Einflussfaktoren zu umfassend. Zudem ist nicht jede kritische Kombination von Verkehrssituationen bekannt (Funktionale Unzulänglichkeit). Auf diesen Randbedingungen basierend stellt diese Arbeit eine Absicherungsstrategie vor, deren zentrales Element ein iterativ erweiterbarer Katalog an Testszenarien ist. Analog zum Verständnis von *Robustheit* in Testverfahren der klassischen Softwareentwicklung dokumentiert die Strategie den aktuellen Wissenstand hinsichtlich relevanter Einflussfaktoren und eignet sich für kontinuierliche Tests des Assistenzsystems innerhalb eines breiten Anwendungsspektrums. So stellt die ontologische Beschreibung von Verkehrsszenarien eine transparente und nachvollziehbare Dokumentationsgrundlage dar, die zudem als Parametrierung von Tests verwendet werden kann. Der Unvollständigkeit einer aktuellen Systemspezifikation wird dadurch begegnet, neue Erkenntnisse sowie Testfälle konsequent durch die Erweiterung des Szenarienkatalogs in die Absicherung zu integrieren. Basierend auf der vorgestellten ontologischen Beschreibung für Verkehrsszenarien wurde deren Eignung zum Transfer von realen zu virtuellen Fahrscenarien sowie für die modellbasierte und probabilistische Testfallgenerierung dargestellt.

Aufgrund der hohen Anzahl möglicher Fahrscenarien und der damit verbundenen Aufwände realer Testfahrten besteht der Wunsch zur Virtualisierung. Hierfür wurde eine Softwarearchitektur entworfen, die eine Beschreibung komplexer dreidimensionaler Umgebungen, bestehend aus diversen Entitäten, erlaubt. Der Fokus der Umgebungsrepräsentation auf die Umsetzung von Sensormodellen ermöglicht eine klare Trennung der Verantwortlichkeiten und öffnet die Möglichkeit zur Wiederverwendung vorhandener Simulationsmodelle. Die Anwendbarkeit der Architektur sowie der entworfenen Klassenstruktur wurde an einem echtzeitfähigen Hardware-in-the-Loop Prüfstand zum Test eines realen Notbremsassistenten diskutiert.

Die präsentierte abstrahierte funktionale Wirkkette von Assistenzsystemen stellt die Grundlage der Modellbildung zweier Sensormodelle dar: Zum einen wird ein ideales Sensormodell entworfen, welches sich zur performanten Gene-

rierung von Ground-Truth-Daten eignet. Außerdem wird das Strahlen-basierte Multi-Sensormodell entwickelt, das die synthetische Generierung von Sensordaten für Ultraschall-, Radar-, Lidar- und Videosensoren ermöglicht. Die Kombination der vorgestellten Architektur mit den Sensormodellen erlaubt die Parallelisierung der Sensoren bei Verwendung eines konsistenten Umgebungsmodells. Die Validierungen der Modelle zeigen, dass ein Transfer von Messsituationen in die virtuelle Umgebung grundsätzlich möglich ist und sich für einzelne Messungen auch Abweichungen im Bereich der Ultraschall-Sensormessgenauigkeit erzielen lassen. Die Performance-Untersuchungen eines auf neuronalen Netzen basierenden Objekterkennungsalgorithmus zeigen, dass sich die synthetischen Sensordaten grundsätzlich zur Stimulation der Objekterkennung eignen. Da sich jedoch sowohl False-Positiv als auch False-Negativ Klassifizierungen ergaben, sollte eine quantitative Bewertung von Objekterkennungsalgorithmen nicht auf synthetischen Daten erfolgen. Mit Hilfe des vorgestellten Vergleichs des Verlaufs mehrerer Receiver Operating Characteristic-Kurven lässt sich die Güte der synthetischen Daten für einzelne Objekterkennungsalgorithmen bewerten. Gleichzeitig liefert diese Auswertung auch auf rein synthetischen Daten einen qualitativen Anhaltspunkt und lässt sich dadurch im Rahmen von Regressionstests verwenden.

6.2. Ausblick

Die praktische Umsetzung der Absicherung autonomer Fahrzeuge und die Anwendung der vorgestellten Strategie birgt noch zukünftige Herausforderungen.

Der präsentierte Transfer realer in synthetische Fahrscenarien basierte auf Informationen der verwendeten Referenzsensorik, bedurfte jedoch zusätzliche Nacharbeit der Szenariobeschreibungen. Die Schaffung effizienter Prozesse zur automatisierten Umsetzung dieses Transfers ist ein zentraler Punkt für die Realisierung eines belastbaren Szenarienkatalogs und für die zuverlässige Auslegung autonomer Fahrfunktionen. Diese Herausforderung stellt eine fachbereichsübergreifende Aufgabe dar. Aufgrund der großen Menge anfallender Daten der Referenzsensorik realer Testfahrten ist ein effizientes und zentrales Datenhandling notwendig. Die semantische Suche ausgewählter Parameterkonstellationen aus historischen Messdaten erfordert einen performanten und erweiterbaren Labeling-Ansatz.

Stellt sich beispielsweise heraus, dass Verkehrssituationen, in denen ein weißer Lastwagen die Fahrbahn kreuzt, zu Fehlverhalten der Unit-Under-Test führen, so ist die Analyse der Messdaten vergleichbarer Situationen interessant. Um Messdaten automatisiert zu filtern, ist die nachträgliche Anwendung eines entsprechenden Labeling-Algorithmus auf dem gesamten verfügbaren Mess-

datensatz nötig. Dieses Beispiel verdeutlicht die Notwendigkeit effizienten Datenhandlings bei der Absicherung von Assistenzsystemen.

Der Transfer von umfangreichen Testfahrten in semantisch beschriebene Szenario-Dateien ist manuell nicht sinnvoll möglich. Zur Automatisierung muss einerseits eine Referenzsensorik innerhalb der Testfahrten verwendet werden, die mindestens alle als bisher relevant betrachteten Parameter zuverlässig aufzeichnet. Um diese Sensordaten in die Ontologie der Szenariobeschreibung zu überführen, sind andererseits unterschiedliche Labeling-Algorithmen notwendig. Gelingt diese Automatisierung, so können kritische Szenarien realer Testfahrten sehr schnell als Startpunkt virtueller Testkampagnen verwendet und der Szenarienkatalog entsprechend einfach erweitert werden.

Zusätzlich ist es nötig, die gesamte Entwicklung der autonomen Systeme dahingehend zu optimieren, kontinuierlich die hinzugewonnenen Testfälle in allen Entwicklungsschritten zu verwenden. Dadurch können Fehler an der frühestmöglichen Stelle erkannt werden. Zudem arbeiten so alle Entwicklungsphasen auf einem Testdatensatz, der den aktuellen Wissensstand hinsichtlich relevanter Einflussfaktoren und kritischer Verkehrssituationen repräsentiert. Wird zusätzlich bei der Umsetzung von Steuergeräten auf eine Initialisierungsmöglichkeit aller internen Parameter für Testzwecke geachtet, so kann die Effizienz der HiL Tests deutlich erhöht werden. Hierdurch würde für einen Großteil der Testfälle die Notwendigkeit entfallen, die Zielzustände aufwändig anfahren zu müssen. Stattdessen könnten die Zielzustände eingestellt und die Systemreaktion auch im HiL Kontext direkt betrachtet werden.

Aufgrund der vielen unterschiedlichen Entitäten, die im Rahmen der Simulation von Verkehrsszenarien beachtet werden müssen, stellt deren Validierung eine der großen Herausforderungen für einen aussagefähigen Transfer realer in virtuelle Umgebungen dar. Neben den vorgestellten Validierungen der Sensormodelle sind zusätzliche Validierungen nötig. Nebst weiterer Untersuchungen der Sensorik, beispielsweise detaillierten Betrachtungen von Wettereinflüssen, sind Validierungen von Fahrermodellen, Verkehrsmodellen und anderer Verkehrsteilnehmer erforderlich. Zusätzlich zum beträchtlichen Aufwand der Datenakquise ist die Identifikation passender Validierungsparameter für diese Modelle notwendig.

Wie die Betrachtungen der Objekterkennung auf synthetischen Sensordaten gezeigt haben, eignen sich diese nur bedingt zur Bewertung des Klassifikators. Synthetische Bilddaten können auch durch die Verwendung von 3D-Modellierwerkzeugen generiert werden, wie beispielsweise mit Hilfe der Software [Blender]. Durch diese Ankopplung wird die Echtzeitfähigkeit und damit der HiL Anwendungsfall aufgegeben, da sich hier schnell Renderzeiten von mehreren Stunden je Bild einstellen, gleichzeitig erlauben diese Werkzeuge



Abb. 6.1: Mit [Blender] synthetisch erzeugte Bilder
[Van Rossom 2018; Zapata 2017]

jedoch die sehr fotorealistische Erstellung synthetischer Bilder. Abbildung 6.1 vermittelt mit zwei Beispielen einen Eindruck der damit einhergehenden Möglichkeiten. Während die Werkzeuge zur synthetischen Bildgenerierung weit entwickelt sind, existieren bisher keine direkt äquivalenten Pendanten zu anderen Sensortypen, die im Bereich der Assistenzsysteme eingesetzt werden.

Auch wenn die speziellen Rendering-Tools den Weg zur rein synthetischen Bewertung Video-basierter Objekterkennungsalgorithmen ebnen, so ist diese Umsetzung dennoch mit hohem Aufwand verbunden. Deshalb ist momentan die Durchführung von Tests der Objekterkennungsebene für alle Sensortypen Open-Loop mit aufgezeichneten Sensordaten effizienter. Die sich daran anschließenden Module der Wirkkette von Assistenzsystemen können mittels idealer Sensormodelle getestet werden. Diese ermöglichen sehr kurze Simulationszeiten und erlauben so die Simulation von deutlich mehr Verkehrsszenarien. Dennoch ist die Reaktion der *Situationsanalyse* und *Anwendung* auch für fehlerbehaftete Objektlisten interessant. Deshalb sollte sich die zukünftige Weiterentwicklung auf ideale Sensormodelle fokussieren, um diese durch eine effiziente Abbildung von Sensorfehlern zu erweitern.

Ein möglicher Ansatz zur Modellierung von Sensorfehlern auf idealisierten Sensorsignalen stellt die Anwendung von Stil-Transfers unter Verwendung neuronaler Netze dar. So lassen sich etwa zwei nicht korrelierte Datensätze dazu verwenden, den Stil des einen Datensatzes auf einen anderen anzuwenden. Dieser Ansatz wird häufig für den Transfer einer Fotografie in einen Mal-Stil verwendet. Analog hierzu lassen sich auch nicht direkt korrelierte Datensätze idealisierter und realer Sensorsignale nutzen, um dem idealisierten Signal Sensorcharakteristiken und -fehler hinzuzufügen. Hierdurch lässt sich die Notwendigkeit aufwändiger Statistiken bezüglich des Sensorfehlverhaltens reduzieren. Unter Verwendung des in [Junginger 2018] präsentierten Ansatzes wurde



Abb. 6.2: Anhand eines Stiltransfers augmentierte synthetische Daten³¹

dieser Stiltransfer auf synthetische Videobilder angewandt. Ein Ergebnis solch augmentierter Bilder ist in Abbildung 6.2 dargestellt. Die Anwendung dieses Ansatzes auf unterschiedliche Sensortypen und der Vergleich zu Realdaten mittels der vorgestellten Receiver Operating Characteristics (ROC)-Kurven-Analyse stellt einen interessanten Aspekt zukünftiger Forschung dar.

³¹Augmentierte Daten von A. Junginger, persönliche Kommunikation 2018

Anhang

A. Exemplarische Objektliste

Um eine Vorstellung einer Objektliste zu vermitteln, zeigt Tabelle A.1 eine solche exemplarische Liste. Die Objektliste basiert zusammenfassend auf Informationen aus [Neumann-Cosel 2014, S. 89] und [Reif 2010, S. 204]. Bewusst wurde auch der leere vierte Objekteintrag aufgelistet. Da in Steuergeräte meist keine dynamische Speicherallokation erlaubt ist, wird im Rahmen der Interfaces auf konstante Datengrößen geachtet. Gleichzeitig kann dadurch auch die Auslastung des Kommunikationsbusses vorhergesagt werden.

Um das Datenvolumen beim Transport zu reduzieren, werden meist spezielle Speicherlayouts zur Repräsentation der Objektliste verwendet. Gleichzeitig werden die Wertebereiche der Einträge entsprechend ihrer zu erwartenden Inhalte skaliert. So wird der longitudinale Abstand nicht als gewöhnliche 32 Bit Gleitkommazahl³² dargestellt, sondern möglicherweise mit nur 10 Bit und einem Wertebereich von $[0 \dots 80]$.

Parameter	Inhalt
Objekt-ID	1
Objekt erkannt	ja
Typ	PKW
Abstand longitudinal	9,3 m
Abstand lateral	2,7 m
Rel. Geschwindigkeit longitudinal	$-9,7 \frac{\text{m}}{\text{s}}$
Rel. Geschwindigkeit lateral	$-0,2 \frac{\text{m}}{\text{s}}$

³²Nach dem IEEE-754 Standard mit Werten zwischen $\pm 3,403 \cdot 10^{38}$

Parameter	Inhalt
Objekt-ID	2
Objekt erkannt	ja
Typ	PKW
Abstand longitudinal	12,2 m
Abstand lateral	-4,2 m
Rel. Geschwindigkeit longitudinal	2,1 $\frac{\text{m}}{\text{s}}$
Rel. Geschwindigkeit lateral	0,7 $\frac{\text{m}}{\text{s}}$
Objekt-ID	3
Objekt erkannt	ja
Typ	LKW
Abstand longitudinal	19,4 m
Abstand lateral	0,2 m
Rel. Geschwindigkeit longitudinal	3,6 $\frac{\text{m}}{\text{s}}$
Rel. Geschwindigkeit lateral	0,2 $\frac{\text{m}}{\text{s}}$
Objekt-ID	4
Objekt erkannt	nein
Typ	-
Abstand longitudinal	-
Abstand lateral	-
Rel. Geschwindigkeit longitudinal	-
Rel. Geschwindigkeit lateral	-

Tabelle A.1: Exemplarische Objektliste mit vier Einträgen

Abbildung A.1 visualisiert schematisch den Inhalt dieser Objektliste. Dies veranschaulicht die Informationen anhand derer ein Assistenzsystem die Verkehrssituation einzuschätzen hat, wenn die Objektliste als Grundlage verwendet wird.

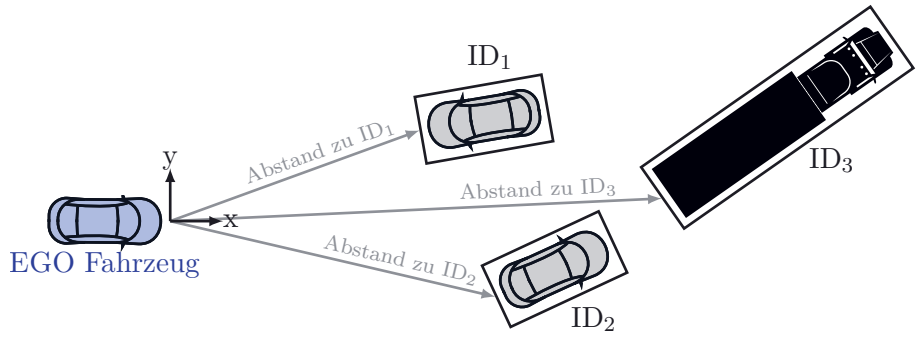


Abb. A.1: Visualisierung der exemplarischen Objektliste

B. XML-Schema der Szenariobeschreibung

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Scenario" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="maneuverType">
    <xs:attribute name="ref" type="xs:string"/>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="sensorType">
    <xs:attribute name="id" type="xs:string" use="required"/>
    <xs:attribute name="posX" type="xs:string"/>
    <xs:attribute name="posY" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="followerType">
    <xs:complexContent>
      <xs:extension base="sensorContent">
        <xs:attribute name="visibility" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="hudType">
    <xs:complexContent>
      <xs:extension base="sensorType">
        <xs:attribute name="height" type="xs:string"/>
        <xs:attribute name="width" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="topViewType">
    <xs:complexContent>
      <xs:extension base="sensorType">
        <xs:attribute name="height" type="xs:string"/>
        <xs:attribute name="width" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="visitorType">
    <xs:complexContent>
      <xs:extension base="sensorType">
        <xs:attribute name="height" type="xs:string"/>
        <xs:attribute name="width" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="distanceSensorType">
    <xs:complexContent>
      <xs:extension base="sensorType">
        <xs:attribute name="height" type="xs:string"/>
        <xs:attribute name="width" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="videoType">
    <xs:complexContent>
      <xs:extension base="sensorContent">
        <xs:attribute name="fovy" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="boundingBoxType">
    <xs:complexContent>
      <xs:extension base="sensorContent">
        <xs:attribute name="fovy" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

```

<xs:complexType name="dofVideoType">
  <xs:complexContent>
    <xs:extension base="sensorContentType">
      <xs:attribute name="fovy" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="physicalVideoType">
  <xs:complexContent>
    <xs:extension base="sensorContentType">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="sensorContentType">
  <xs:complexContent>
    <xs:extension base="sensorType">
      <xs:choice maxOccurs="unbounded">
        <xs:element maxOccurs="unbounded" minOccurs="0" name="eye">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="vectorType">
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="center">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="vectorType">
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="up">
          <xs:complexType>
            <xs:complexContent>
              <xs:extension base="vectorType">
              </xs:extension>
            </xs:complexContent>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs:attribute name="height" type="xs:string"/>
      <xs:attribute name="width" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="depthSensorType">
  <xs:complexContent>
    <xs:extension base="sensorContentType">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="physicalLidarType">
  <xs:complexContent>
    <xs:extension base="sensorContentType">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="physicalRadarType">
  <xs:complexContent>
    <xs:extension base="sensorContentType">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="physicalUltrasonicType">
  <xs:complexContent>
    <xs:extension base="sensorContentType">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

</xs:complexType>

<xs:complexType name="vectorType">
  <xs:attribute name="x" type="xs:string"/>
  <xs:attribute name="y" type="xs:string"/>
  <xs:attribute name="z" type="xs:string"/>
</xs:complexType>

<xs:complexType name="rbgType">
  <xs:attribute name="r" type="xs:string"/>
  <xs:attribute name="b" type="xs:string"/>
  <xs:attribute name="g" type="xs:string"/>
</xs:complexType>

<xs:complexType name="posType">
  <xs:attribute name="posX" type="xs:string"/>
  <xs:attribute name="posY" type="xs:string"/>
  <xs:attribute name="posZ" type="xs:string"/>
</xs:complexType>

<xs:complexType name="arrowType">
  <xs:complexContent>
    <xs:extension base="posType">
      <xs:attribute name="direction" type="xs:string"/>
      <xs:attribute name="scale" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="sphereType">
  <xs:complexContent>
    <xs:extension base="posType">
      <xs:attribute name="radius" type="xs:string"/>
      <xs:attribute name="scale" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="inputType">
  <xs:choice maxOccurs="unbounded">
    <xs:element maxOccurs="unbounded" minOccurs="0" name="arrow"
      type="arrowType"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="sphere"
      type="sphereType"/>
  </xs:choice>
  <xs:attribute name="id" type="xs:string"/>
</xs:complexType>

<xs:complexType name="actorType">
  <xs:complexContent>
    <xs:extension base="vectorType">
      <xs:choice>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="input"
          type="inputType"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="sensors"
          type="sensorsListType"/>
      </xs:choice>
      <xs:attribute name="geometryReference" type="xs:string"/>
      <xs:attribute name="geometry" type="xs:string"/>
      <xs:attribute name="id" type="xs:string"/>
      <xs:attribute name="hdg" type="xs:string"/>
      <xs:attribute name="s" type="xs:string"/>
      <xs:attribute name="t" type="xs:string"/>
      <xs:attribute name="geometryPhysicalVideo" type="xs:string"/>
      <xs:attribute name="geometryPhysicalLidar" type="xs:string"/>
      <xs:attribute name="geometryPhysicalRadar" type="xs:string"/>
      <xs:attribute name="geometryPhysicalUltrasonic" type="xs:string"/>
      <xs:attribute name="roadId" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="sensorsListType">
  <xs:choice maxOccurs="unbounded" minOccurs="0">
    <xs:element maxOccurs="unbounded" minOccurs="0" name="hud" type="hudType"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="follower"
      type="followerType"/>
  </xs:choice>
</xs:complexType>

```

```

<xs:element maxOccurs="unbounded" minOccurs="0" name="depthSensor"
  < type="depthSensorType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="video"
  < type="videoType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="distanceSensor"
  < type="distanceSensorType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="physicalVideo"
  < type="physicalVideoType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="physicalUltrasonic"
  < type="physicalUltrasonicType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="physicalLidar"
  < type="physicalLidarType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="physicalRadar"
  < type="physicalRadarType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="visitor"
  < type="visitorType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="topView"
  < type="topViewType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="dofVideo"
  < type="dofVideoType" />
<xs:element maxOccurs="unbounded" minOccurs="0" name="boundingBoxSensor"
  < type="boundingBoxType" />

<xs:element maxOccurs="unbounded" minOccurs="0" name="objectListSensor">
  <xs:complexType>
    <xs:attribute name="id" type="xs:string" />
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>

<xs:complexType name="overlayType">
  <xs:attribute name="roadId" type="xs:string" />
  <xs:attribute name="s" type="xs:string" />
  <xs:attribute name="t" type="xs:string" />
  <xs:attribute name="texture" type="xs:string" />
  <xs:attribute name="height" type="xs:string" />
  <xs:attribute name="width" type="xs:string" />
  <xs:attribute name="hdg" type="xs:string" />
</xs:complexType>

<xs:complexType name="objectType">
  <xs:choice>
    <xs:element name="object" type="objectType" maxOccurs="unbounded"
      < minOccurs="0" />
  </xs:choice>
  <xs:attribute name="geometry" type="xs:string" />
  <xs:attribute name="geometryPhysicalVideo" type="xs:string" />
  <xs:attribute name="geometryPhysicalLidar" type="xs:string" />
  <xs:attribute name="geometryPhysicalRadar" type="xs:string" />
  <xs:attribute name="geometryPhysicalUltrasonic" type="xs:string" />
  <xs:attribute name="s" type="xs:string" />
  <xs:attribute name="t" type="xs:string" />
  <xs:attribute name="hdg" type="xs:string" />
  <xs:attribute name="x" type="xs:string" />
  <xs:attribute name="y" type="xs:string" />
  <xs:attribute name="z" type="xs:string" />
  <xs:attribute name="roadId" type="xs:string" />
</xs:complexType>

<xs:complexType name="ObjectsType">
  <xs:choice maxOccurs="unbounded">
    <xs:element maxOccurs="unbounded" minOccurs="0" name="object"
      < type="objectType" />
    <xs:element maxOccurs="unbounded" minOccurs="0" name="building">
      <xs:complexType>
        <xs:attribute name="id" type="xs:string" />
        <xs:attribute name="x" type="xs:string" />
        <xs:attribute name="y" type="xs:string" />
        <xs:attribute name="type" type="xs:string" />
        <xs:attribute name="name" type="xs:string" />
      </xs:complexType>
    </xs:element>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="tree">
      <xs:complexType>
        <xs:attribute name="id" type="xs:string" />
        <xs:attribute name="x" type="xs:string" />

```

```

        <xs:attribute name="y" type="xs:string"/>
        <xs:attribute name="type" type="xs:string"/>
        <xs:attribute name="name" type="xs:string"/>
    </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>

<xs:element name="Scenario">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded" minOccurs="0">
      <xs:element name="actors">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0" name="actor">
              <type="actorType"/>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="vehicle">
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="vectorType">
                    <xs:choice maxOccurs="unbounded">
                      <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="sensors" type="sensorsListType"/>
                      <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="actor" type="actorType"/>
                      <xs:element name="input" type="inputType"/>
                      <xs:element maxOccurs="unbounded" minOccurs="0"
                        name="geometry">
                        <xs:complexType>
                          <xs:attribute name="path" type="xs:string"/>
                        </xs:complexType>
                      </xs:element>
                    </xs:choice>
                    <xs:attribute name="geometry" type="xs:string"/>
                    <xs:attribute name="id" type="xs:string"/>
                    <xs:attribute name="hdg" type="xs:string"/>
                    <xs:attribute name="s" type="xs:string"/>
                    <xs:attribute name="t" type="xs:string"/>
                    <xs:attribute name="roadId" type="xs:string"/>
                    <xs:attribute name="type" type="xs:string"/>
                    <xs:attribute name="speed" type="xs:string"/>
                    <xs:attribute name="length" type="xs:string"/>
                    <xs:attribute name="width" type="xs:string"/>
                    <xs:attribute name="height" type="xs:string"/>
                    <xs:attribute name="ussGeometry" type="xs:string"/>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
  <xs:element name="objects" type="ObjectsType"/>
  <xs:element name="input" type="inputType"/>
  <xs:element name="environment">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="objects" type="ObjectsType"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="light">
          <xs:complexType>
            <xs:choice maxOccurs="unbounded">
              <xs:element maxOccurs="1" minOccurs="0" name="color">
                <xs:complexType>
                  <xs:complexContent>
                    <xs:extension base="rbgType">
                      <xs:choice maxOccurs="unbounded">
                        <xs:element maxOccurs="1" type="xs:string"
                          minOccurs="0" name="r"/>
                        <xs:element maxOccurs="1" type="xs:string"
                          minOccurs="0" name="g"/>
                        <xs:element maxOccurs="1" type="xs:string"
                          minOccurs="0" name="b"/>
                      </xs:choice>
                    </xs:extension>
                  </xs:complexContent>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>

```

```

<xs:element maxOccurs="1" minOccurs="0" name="ambient">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="r" type="xs:string"/>
        <xs:attribute name="g" type="xs:string"/>
        <xs:attribute name="b" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="1" minOccurs="0" name="diffuse">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="r" type="xs:string"/>
        <xs:attribute name="g" type="xs:string"/>
        <xs:attribute name="b" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="1" minOccurs="0" name="position">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="vectorType">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="1" minOccurs="0" name="specular">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="r" type="xs:string"/>
        <xs:attribute name="g" type="xs:string"/>
        <xs:attribute name="b" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0" name="terrain">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element maxOccurs="1" minOccurs="0" name="left">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0"
              name="laneSection">
              <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                  <xs:element maxOccurs="unbounded" minOccurs="0"
                    name="polynom">
                    <xs:complexType>
                      <xs:attribute name="tOffset" type="xs:string"/>
                      <xs:attribute name="a" type="xs:string"/>
                      <xs:attribute name="b" type="xs:string"/>
                      <xs:attribute name="c" type="xs:string"/>
                      <xs:attribute name="d" type="xs:string"/>
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
              </xs:complexType>
            </xs:element>
            <xs:attribute name="s" type="xs:string"/>
            <xs:attribute name="t" type="xs:string"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
      <xs:element maxOccurs="1" minOccurs="0" name="right">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0"
              name="laneSection">

```



```

        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0"
              ↵ name="polynom">
              <xs:complexType>
                <xs:attribute name="tOffset" type="xs:string"/>
                <xs:attribute name="a" type="xs:string"/>
                <xs:attribute name="b" type="xs:string"/>
                <xs:attribute name="c" type="xs:string"/>
                <xs:attribute name="d" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:choice>
          <xs:attribute name="s" type="xs:string"/>
          <xs:attribute name="t" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:choice>
<xs:attribute name="maxSize" type="xs:string"/>
<xs:attribute name="slope" type="xs:string"/>
</xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0" name="weather">
  <xs:complexType>
    <xs:attribute name="streetcond" type="xs:string"/>
    <xs:attribute name="temperature" type="xs:string"/>
    <xs:attribute name="visibility" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0" name="skybox">
  <xs:complexType>
    <xs:attribute name="type" type="xs:string"/>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="roadNetwork">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element maxOccurs="unbounded" minOccurs="0" name="overlay"
        ↵ type="overlayType"/>
    </xs:choice>
    <xs:attribute name="id" type="xs:string"/>
    <xs:attribute name="OpenDRIVEfile" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element name="junctionBehavior">
</xs:element>
<xs:element name="storyboard">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element maxOccurs="unbounded" minOccurs="0" name="step">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0"
              ↵ name="accelerateToRelativeSpeed">
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="maneuverType">
                </xs:extension>
              </xs:complexContent>
            </xs:element>
            <xs:element maxOccurs="unbounded" minOccurs="0"
              ↵ name="AccelerateToTargetSpeed">
              <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                  <xs:element maxOccurs="unbounded" minOccurs="0" name
                    ↵="CancelConditions">
                    <xs:complexType>
                      <xs:choice maxOccurs="unbounded">

```

```

        <xs:element maxOccurs="unbounded" minOccurs="0"
          ↵ name="SPositionReached">
          <xs:complexType>
            <xs:attribute name="value" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  </xs:choice>
  <xs:attribute name="value" type="xs:string"/>
</xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↵ name="DriveToTPosition">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element maxOccurs="unbounded" minOccurs="0" name
        ↵="CancelConditions">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0"
              ↵ name="SPositionReached">
              <xs:complexType>
                <xs:attribute name="value" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:choice>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↵ name="DriveToSPosition">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element maxOccurs="unbounded" minOccurs="0" name
        ↵="CancelConditions">
        <xs:complexType>
          <xs:choice maxOccurs="unbounded">
            <xs:element maxOccurs="unbounded" minOccurs="0"
              ↵ name="SPositionReached">
              <xs:complexType>
                <xs:attribute name="value" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:choice>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↵ name="DriveToJunction">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="maneuverType">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0" name="Brake">
  <xs:complexType>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↵ name="DriveToRelativePositionS">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="maneuverType">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

```

```

</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↳ name="DriveToRoadObject">
  <xs:complexType>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↳ name="WaitForStepToFinish">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="maneuverType">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↳ name="IncreaseSpeed">
  <xs:complexType>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" minOccurs="0"
  ↳ name="WaitSeconds">
  <xs:complexType>
    <xs:attribute name="value" type="xs:string"/>
  </xs:complexType>
</xs:element>
</xs:choice>
<xs:attribute name="actorId" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:choice>
<xs:attribute name="version" type="xs:string"/>
<xs:attribute name="enableShadows" type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:schema>

```

C. Exemplarische Szenariobeschreibung eines HiL Setups

```
<?xml version='1.0' standalone='yes'?>
<Scenario>
  <roadNetwork OpenDRIVEfile='Testroad_0001.xodr' >
    <overlay roadId='1' s='10' t='-1.6'
      ↪ texture='overlay_60_kmh' />
    <overlay roadId='1' s='30' t='-8'
      ↪ texture='overlay_60_kmh' />
  </roadNetwork>
  <objects>
    <object roadId='1' s='30' t='-11'
      ↪ geometry='streetLamp' />
    <object roadId='1' s='60' t='-11'
      ↪ geometry='streetLamp' />
    <object roadId='1' s='5' t='-12' hdg='5'
      ↪ geometry='Tree1' />
    <object roadId='1' s='18' t='-13' hdg='18'
      ↪ geometry='Tree2' />
    <object roadId='1' s='31' t='-11' hdg='31'
      ↪ geometry='Tree3' />
    <object roadId='1' s='44' t='-15' hdg='44'
      ↪ geometry='Tree4' />
    <object roadId='1' s='6' t='12' hdg='5'
      ↪ geometry='Tree1' />
    <object roadId='1' s='15' t='13' hdg='18'
      ↪ geometry='Tree2' />
    <object roadId='1' s='30' t='11' hdg='31'
      ↪ geometry='Tree3' />
    <object roadId='1' s='44' t='15' hdg='44'
      ↪ geometry='Tree4' />
  </objects>
  <environment>
    <terrain maxSize='40' slope='0.01' />
    <light>
      <position x='0' y='0' z='50' />
      <ambient>0.85</ambient>
      <diffuse>0.85</diffuse>
    </light>
  </environment>
</Scenario>
```

```
        <specular>0.07</specular>
    </light>
    <background map='BrightSunshine' />
</environment>
<actors>
    <vehicle id='1' roadId='1' s='0' t='-1.6'
        ↳ geometry='Coupe_Blue' >
        <sensors>
            <objectListSensor id='101' />
            <video id='102' width='512' height='512'
                ↳ >
                <eye x='1.3' z='1.5' />
            </video>
        </sensors>
    </vehicle>
    <vehicle id='2' roadId='1' s='20' t='-1.6'
        ↳ geometry='Coupe_Blue' />
</actors>
<storyboard>
    <step actorId='1'>
        <AccelerateToTargetSpeed value='5' />
    </step>
</storyboard>
</Scenario>
```

D. Szenariobeschreibungen der modellbasierten Testfallgenerierung

Quellcode A.1: Testfallgenerierung 1

```
<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="20" t="-1" />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↵
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />

  <storyboard>
    <step actorId="1">
      <accelerateToTargetSpeed value="13" />
    </step>
    <step actorId="2">
      <waitSeconds value="5" />
    </step>
    <step actorId="2">
      <accelerateToTargetSpeed value="3" />
    </step>
  </storyboard>
</scenario>
```

Quellcode A.2: Testfallgenerierung 2

```
<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="89.375" t="-1" />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↵
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />
```

```

<storyboard>
  <step actorId="1">
    <accelerateToTargetSpeed value="5.5" />
  </step>
  <step actorId="2">
    <waitSeconds value="1.25" />
  </step>
  <step actorId="2">
    <accelerateToTargetSpeed value="3" />
  </step>
</storyboard>
</scenario>

```

Quellcode A.3: Testfallgenerierung 3

```

<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="95.4688" t="-1"
      ↪ />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↪
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />

  <storyboard>
    <step actorId="1">
      <accelerateToTargetSpeed value="4.25" />
    </step>
    <step actorId="2">
      <waitSeconds value="0.625" />
    </step>
    <step actorId="2">
      <accelerateToTargetSpeed value="3" />
    </step>
  </storyboard>
</scenario>

```

Quellcode A.4: Testfallgenerierung 4

```
<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="20" t="-2.1" />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↵
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />

  <storyboard>
    <step actorId="1">
      <accelerateToTargetSpeed value="13" />
    </step>
    <step actorId="2">
      <waitSeconds value="5" />
    </step>
    <step actorId="2">
      <accelerateToTargetSpeed value="3" />
    </step>
  </storyboard>
</scenario>
```

Quellcode A.5: Testfallgenerierung 5

```
<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="89.375" t="-2.1"
    ↵ />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↵
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />

  <storyboard>
    <step actorId="1">
```

```

        <accelerateToTargetSpeed value="5.5" />
    </step>
    <step actorId="2">
        <waitSeconds value="1.25" />
    </step>
    <step actorId="2">
        <accelerateToTargetSpeed value="3" />
    </step>
</storyboard>
</scenario>

```

Quellcode A.6: Testfallgenerierung 6

```

<?xml version="1.0" standalone="yes"?>
<scenario>
    <actors>
        <vehicle id="1" roadId="1" s="95.4688"
            ↪ t="-2.1" />
        <vehicle id="2" roadId="1" s="100" t="0" />
            ↪
    </actors>

    <roadNetwork id="3" file="StraightLine.xodr" />

    <storyboard>
        <step actorId="1">
            <accelerateToTargetSpeed value="4.25" />
        </step>
        <step actorId="2">
            <waitSeconds value="0.625" />
        </step>
        <step actorId="2">
            <accelerateToTargetSpeed value="3" />
        </step>
    </storyboard>
</scenario>

```

Quellcode A.7: Testfallgenerierung 7

```
<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="20" t="-2.5" />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↵
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />

  <storyboard>
    <step actorId="1">
      <accelerateToTargetSpeed value="13" />
    </step>
    <step actorId="2">
      <waitSeconds value="5" />
    </step>
    <step actorId="2">
      <accelerateToTargetSpeed value="3" />
    </step>
  </storyboard>
</scenario>
```

Quellcode A.8: Testfallgenerierung 8

```
<?xml version="1.0" standalone="yes"?>
<scenario>
  <actors>
    <vehicle id="1" roadId="1" s="89.375" t="-2.5"
    ↵ />
    <vehicle id="2" roadId="1" s="100" t="0" />
    ↵
  </actors>

  <roadNetwork id="3" file="StraightLine.xodr" />

  <storyboard>
    <step actorId="1">
```

```

        <accelerateToTargetSpeed value="5.5" />
    </step>
    <step actorId="2">
        <waitSeconds value="1.25" />
    </step>
    <step actorId="2">
        <accelerateToTargetSpeed value="3" />
    </step>
</storyboard>
</scenario>

```

Quellcode A.9: Testfallgenerierung 9

```

<?xml version="1.0" standalone="yes"?>
<scenario>
    <actors>
        <vehicle id="1" roadId="1" s="95.4688"
            ↪ t="-2.5" />
        <vehicle id="2" roadId="1" s="100" t="0" />
        ↪
    </actors>

    <roadNetwork id="3" file="StraightLine.xodr" />

    <storyboard>
        <step actorId="1">
            <accelerateToTargetSpeed value="4.25" />
        </step>
        <step actorId="2">
            <waitSeconds value="0.625" />
        </step>
        <step actorId="2">
            <accelerateToTargetSpeed value="3" />
        </step>
    </storyboard>
</scenario>

```

E. Szenariobeschreibungen der Transfersituationen

```
<?xml version="1.0" encoding="UTF-8"?>
<Scenario>
  <actors>
    <vehicle id="0" x="20" y="-1.5" z="1.65"
      ↪ hdg="0">
      <sensors>
        <video id="101" width="1242"
          ↪ height="375" fovy="28">
          <eye y="0"/>
          <center x="1" y="0" z="-0.02"/>
        </video>
        <boundingBoxSensor id="102" width="1242"
          ↪ height="375" fovy="28" >
          <eye y="0"/>
          <center x="1" y="0" z="-0.02"/>
        </boundingBoxSensor>
      </sensors>
    </vehicle>
    <vehicle id="1" x="32.64" y="1.68" z="-0.03"
      ↪ hdg="3.13" geometry="Van1_Black"/>
  </actors>
  <roadNetwork
    ↪ OpenDRIVEfile="2011_09_26_drive_0027.xodr"/>
  <environment>
    <terrain maxSize="40"/>
    <light>
      <position x="0" y="10" z="10"/>
      <diffuse r="0.8" g="0.8" b="0.8"/>
      <ambient r="0.3" g="0.3" b="0.3"/>
      <specular r="0.9" g="0.9" b="0.9"/>
    </light>
    <skybox type="Yokohama2"/>
  </environment>
  <objects>
    <object geometry="Tree1" roadId="1" s="0.41"
      ↪ t="7.67" hdg="0.53"/>
  </objects>
</Scenario>
```

```
<object geometry="Tree1" roadId="1" s="65"
  ↳ t="12.69" hdg="0.23"/>
<object geometry="Tree1" roadId="1" s="114.78"
  ↳ t="12.58" hdg="5.85"/>
<object geometry="Tree1" roadId="1" s="32.8"
  ↳ t="7.27" hdg="5.40"/>
<object geometry="Tree1" roadId="1" s="4.9"
  ↳ t="12.95" hdg="0.09"/>
<object geometry="Tree1" roadId="1" s="48.27"
  ↳ t="13.36" hdg="3.62"/>
<object geometry="Leitpfosten" x="33" y="-3.2"
  ↳ hdg="-1.57"/>
<object geometry="Leitpfosten" x="32.5"
  ↳ y="+3.4" hdg="+1.57"/>
<object geometry="Leitpfosten" x="53" y="-3.2"
  ↳ hdg="-1.57"/>
<object geometry="Leitpfosten" x="53" y="+3.2"
  ↳ hdg="+1.57"/>
<object geometry="Leitpfosten" x="73" y="-3.2"
  ↳ hdg="-1.57"/>
<object geometry="Leitpfosten" x="73" y="+3.2"
  ↳ hdg="+1.57"/>
<object geometry="Leitpfosten" x="93" y="-3.2"
  ↳ hdg="-1.57"/>
<object geometry="Leitpfosten" x="93" y="+3.2"
  ↳ hdg="+1.57"/>
<object geometry="Schild1" x="30" y="+4"
  ↳ hdg="3.14"/>
</objects>
</Scenario>
```

```

<?xml version="1.0" encoding="UTF-8"?>
<Scenario>
  <actors>
    <vehicle id="0" x="20" y="-4.4" z="1.65"
      ↪ hdg="0">
      <sensors>
        <video id="101" width="1242"
          ↪ height="375" fovy="28" >
          <center x="1" z="-0.02" />
        </video>
        <boundingBoxSensor id="102" width="1242"
          ↪ height="375" fovy="28" >
          <center x="1" z="-0.02" />
        </boundingBoxSensor>
      </sensors>
    </vehicle>
    <vehicle id="1" x="31.6" y="1.5848621"
      ↪ z="-0.03" hdg="3.13"
      ↪ geometry="Van1_Blue"/>
    <vehicle id="2" x="39.3" y="-1.5" z="-0.03"
      ↪ hdg="0" geometry="Hatchback2_Black"/>
    <vehicle id="3" x="48.3" y="-1.5" z="-0.03"
      ↪ hdg="0" geometry="Hatchback1"/>
    <vehicle id="4" x="51.2" y="-4.7" z="-0.03"
      ↪ hdg="0" geometry="BMW_Black"/>
    <vehicle id="5" x="62.8" y="-2.4" z="-0.03"
      ↪ hdg="0.02" geometry="Hatchback2_Black"/>
    <vehicle id="6" x="62.2" y="-4.75" z="-0.03"
      ↪ hdg="0.0" geometry="BMW_Red"/>
    <vehicle id="7" x="78.8" y="-2.45" z="-0.03"
      ↪ hdg="0.05" geometry="Sedan"/>
  </actors>
  <roadNetwork
    ↪ OpenDRIVEfile="2011_09_26_drive_0029.xodr"/>
  <environment>
    <terrain maxSize="40"/>
    <light>
      <position x="0" y="20" z="20"/>
      <diffuse r="0.8" g="0.8" b="0.8"/>
    </light>
  </environment>
</Scenario>

```

```
    <ambient r="0.6" g="0.6" b="0.6"/>
    <specular r="0.8" g="0.8" b="0.8"/>
  </light>
  <skybox type="Yokohama2"/>
</environment>
<objects>
  <object geometry="Tree1" roadId="1" s="0.41"
    ↳ t="7.6" hdg="0.53"/>
  <object geometry="Tree1" roadId="1" s="65"
    ↳ t="12.61" hdg="0.23"/>
  <object geometry="Tree1" roadId="1" s="11"
    ↳ t="12.58" hdg="5.85"/>
  <object geometry="Tree1" roadId="1" s="32"
    ↳ t="7.27" hdg="5.40"/>
  <object geometry="Tree1" roadId="1" s="4"
    ↳ t="12.95" hdg="0.099"/>
  <object geometry="Tree1" roadId="1" s="48"
    ↳ t="13.36" hdg="3.62"/>
  <object geometry="Tree1" roadId="1" s="146"
    ↳ t="14.02" hdg="1.53"/>
  <object geometry="Leitpfosten" x="40" y="-7.2"
    ↳ hdg="-1.57"/>
</objects>
</Scenario>
```

F. Systeminformationen

Die Simulationszeiten der Sensormodelle wurde auf folgendem System ermittelt:

Name	HP ZBook 17 G3 Mobile Workstation
Prozessor	Intel [®] Xeon [®] E3-1535M v5
Arbeitsspeicher	64 GB DDR4-2133 ECC SDRAM
Festplatte	1 TB HP Z Turbo Drive G2 (NVMe PCIe SSD)
Grafikkarte	NVIDIA [®] Quadro [®] M2000M (4 GB GDDR5 dediziert)

Tabelle A.2: Systeminformationen

Literaturverzeichnis

Monographien

- [Christoph 2002] Gerd Christoph und Horst Hackel (2002): *Starthilfe Stochastik*. Wiesbaden: Vieweg+Teubner Verlag.
- [Gassmann 2017] Oliver Gassmann, Karolin Frankenberger und Michaela Csik (2017): *Geschäftsmodelle entwickeln. 55 innovative Konzepte mit dem St. Galler Business Model Navigator*. 2. Aufl. München: Hanser.
- [Gregory 2014] Jason Gregory und Richard Lemarchand (2014): *Game engine architecture*. 2. Aufl. Boca Raton (USA): CRC Press, Taylor & Francis Group.
- [Haferkorn 2003] Heinz Haferkorn (2003): *Optik. Physikalisch-technische Grundlagen und Anwendungen*. 4. Aufl. Weinheim: Wiley-VCH.
- [Liggesmeyer 2002] Peter Liggesmeyer (2002): *Software-Qualität. Testen, Analysieren und Verifizieren von Software*. 2. Aufl. Heidelberg: Spektrum, Akademischer Verlag.
- [Paulweber 2014] Michael Paulweber und Klaus Lebert (2014): *Mess- und Prüfstandstechnik. Antriebsstrangentwicklung · Hybridisierung · Elektrifizierung*. Der Fahrzeugantrieb. Wiesbaden: Springer Vieweg.
- [Reif 2010] Konrad Reif (2010): *Fahrstabilisierungssysteme und Fahrerassistenzsysteme*. Bosch Fachinformation Automobil. Wiesbaden: Springer Fachmedien.

Beiträge in Sammelwerken

- [Dietmayer 2015] Klaus Dietmayer, Dominik Nuß und Stephan Reuter (2015): “Repräsentation fusionierter Umfelddaten”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Bd. 3. Wiesbaden: Springer Vieweg, S. 453–480.
- [Gotzig 2015] Heinrich Gotzig und Georg Geduld (2015): “LIDAR-Sensorik”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan

Hakuli, Felix Lotz und Christina Singer. Bd. 3. Wiesbaden: Springer Vieweg, S. 317–334.

- [Hakuli 2015] Stephan Hakuli und Markus Krug (2015): “Virtuelle Integration. Durchgängiges Testen und Bewerten im virtuellen Fahrversuch”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Wiesbaden: Springer Vieweg, S. 127–138.
- [Niehsen 2005] Wolfgang Niehsen, Rainer Garnitz, Michael Weilkes und Martin Stämpfle (2005): “Informationsfusion für Fahrerassistenzsysteme”. In: *Fahrerassistenzsysteme mit maschineller Wahrnehmung*. Hrsg. von Markus Maurer. Berlin: Springer-Verlag, S. 43–58.
- [Noll 2015] Martin Noll und Peter Rapps (2015): “Ultraschallsensorik”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Bd. 3. Wiesbaden: Springer Vieweg, S. 243–258.
- [Punke 2015] Martin Punke, Stefan Menzel, Boris Werthessen, Nicolaj Stache und Maximilian Höpfl (2015): “Kamera-Hardware”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Wiesbaden: Springer Vieweg, S. 347–368.
- [Schiele 2015] Bernt Schiele und Christian Wojek (2015): “Kamerabasierte Fußgängerdetektion”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Wiesbaden: Springer Vieweg, S. 421–436.
- [Schubert 2015] Robin Schubert, Norman Mattern und Roy Bours (2015): “Simulation von Sensorfehlern zur Evaluierung von Fahrerassistenzsystemen”. In: *Fahrerassistenzsysteme und Effiziente Antriebe*. Hrsg. von Wolfgang Siebenpfeiffer. Wiesbaden: Springer Vieweg, S. 69–73.
- [Seiniger 2015] Patrick Seiniger und Dirk Alexander Weitzel (2015): “Testverfahren für Verbraucherschutz und Gesetzgebung”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Wiesbaden: Springer Vieweg, S. 167–182.

- [Stiller 2015] Christoph Stiller, Alexander Bachmann und Andreas Geiger (2015): “Maschinelles Sehen”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Wiesbaden: Springer Vieweg, S. 369–394.
- [Wachenfeld 2015] Walther Wachenfeld und Hermann Winner (2015): “Die Freigabe des autonomen Fahrens”. In: *Autonomes Fahren. Technische, rechtliche und gesellschaftliche Aspekte*. Hrsg. von Markus Maurer, J. Christian Gerdes, Barbara Lenz und Hermann Winner. Berlin: Springer-Verlag, S. 439–464.
- [Wilhelm 2015] Ulf Wilhelm, Susanne Ebel und Dirk Alexander Weitzel (2015): “Funktionale Sicherheit und ISO 26262. Validierung von Systemen mit funktionaler Unzulänglichkeit”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Bd. 3. Wiesbaden: Springer Vieweg, S. 85–103.
- [Winner 2015a] Hermann Winner (2015a): “Quo vadis, FAS?” In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Bd. 3. Wiesbaden: Springer Vieweg, S. 1167–1186.
- [Winner 2015b] Hermann Winner (2015b): “Radarsensorik”. In: *Handbuch Fahrerassistenzsysteme. Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Hrsg. von Hermann Winner, Stephan Hakuli, Felix Lotz und Christina Singer. Bd. 3. Wiesbaden: Springer Vieweg, S. 259–316.

Artikel in Zeitschriften

- [Arnold 2013] Martin Arnold, Christoph Clauss und Tom Schierz (2013): “Error Analysis and Error Estimates for Co-Simulation in FMI for Model Exchange and Co-Simulation V2.0”. In: *Archive of Mechanical Engineering* 60 (1), S. 75–94.
- [Bernsteiner 2015] Stefan Bernsteiner, Zoltan Magosi, Daniel Lindvai-Soos und Arno Eichberger (2015): “Radarsensormodell für den virtuellen Entwicklungsprozess”. In: *ATZelektronik* 10 (2), S. 72–79.
- [Birnbacher 2016] Dieter Birnbacher und Wolfgang Birnbacher (2016): “Automatisiertes Fahren. Ethische Fragen an der Schnittstelle von Technik und Gesellschaft”. In: *Information Philosophie* 2016 (4), S. 8–15.

- [Donges 1982] Edmund Donges (1982): “Aspekte der Aktiven Sicherheit bei der Führung von Personenkraftwagen”. In: *Automobil-Industrie* 27 (2), S. 183–190.
- [Feilhauer 2017b] Marius Feilhauer und Jürgen Häring (2017b): “Anwendungsorientierte Absicherungsstrategie für Fahrerassistenzsysteme”. In: *ATZextra* 22 (3), S. 32–35.
- [Feilhauer 2016b] Marius Feilhauer, Jürgen Häring und Sean Wyatt (2016b): “Current approaches in HiL-based ADAS testing”. In: *SAE International Journal of Commercial Vehicles* 9 (2), S. 63–69.
- [Geimer 2006] Marcus Geimer, Thomas Krüger und Peter Linsel (2006): “Co-Simulation, gekoppelte Simulation oder Simulatorkopplung. Ein Versuch der Begriffsvereinheitlichung”. In: *Zeitschrift für Fluidtechnik* 50 (11-12), S. 572–576.
- [Gerstenberg 2015] Jan Gerstenberg, Helmut Hartlief und Stephan Tafel (2015): “RDE-Entwicklungsumgebung am hochdynamischen Motorprüfstand”. In: *ATZextra* 20 (8), S. 36–41.
- [Junginger 2018] Andrej Junginger, Markus Hanselmann, Thilo Strauss, Sebastian Boblest, Jens Buchner und Holger Ulmer (2018): “Unpaired High-Resolution and Scalable Style Transfer Using Generative Adversarial Networks”. In: *CoRR* abs/1810.05724. arXiv: 1810.05724.
- [Kajiya 1986] James T. Kajiya (1986): “The rendering equation”. In: *ACM Siggraph Computer Graphics* 20 (4), S. 143–150.
- [Sattler 2015] Kathrin Sattler, Christian Diedrich und Thomas Brandmeier (2015): “Manöverbasiertes Testen in Kombination mit evolutionären Algorithmen”. In: *at - Automatisierungstechnik* 63 (6), S. 450–464.
- [Schubert 2014] Robin Schubert, Norman Mattern und Roy Bours (2014): “Simulation of sensor models. For the evaluation of Advanced Driver Assistance Systems”. In: *ATZelektronik* 9 (3), S. 26–29.
- [Segner 2015] Christian Segner, Frank Försterling, Stefan Lüke, Guido Meier-Arendt und Georg Ortner (2015): “Die größten Baustellen. Auf dem Weg zum automatisierten Fahren”. In: *Elektronik automotive* 2015 (8/9), S. 24–29.

Beiträge in Tagungsbänden

- [Ebel 2009] Susanne Ebel, Ulf Wilhelm, A. Grimm und U. Sailer (2009): “Wie sicher ist sicher genug? Anforderungen an die funktionale Unzulänglichkeit von Fahrerassistenzsystemen in Anlehnung an das gesellschaftlich akzeptierte Risiko”. In: *6. Workshop Fahrerassistenzsysteme. Bewertung der Wahrnehmung von Fahrerassistenzsystemen*. Karlsruhe: fmrt, S. 48–57.
- [Feilhauer 2017a] Marius Feilhauer und Jürgen Häring (2017a): “A real-time capable multi-sensor model to validate ADAS in a virtual environment”. In: *Fahrerassistenzsysteme. 3. Internationale ATZ-Fachtagung Automatisiertes Fahren*. Hrsg. von Rolf Isermann. Wiesbaden: Springer Vieweg, S. 227–256.
- [Feilhauer 2016a] Marius Feilhauer, Jürgen Häring, Deepa Vijayaraghavan und Johannes Wagner (2016a): “Efficient ADAS Test and Validation along the stages of the Development Cycle”. In: *6. AutoTest Fachkonferenz. Test von Hardware und Software in der Automobilentwicklung*. FKFS. Stuttgart.
- [Feilhauer 2016c] Marius Feilhauer, Benjamin Schnarr und Jürgen Häring (2016c): “Systematische Parametervariation zur Erstellung von Testszenarien zur Absicherung von Fahrerassistenzsystemen”. In: *8. Wissenschaftsforum Mobilität*. Universität Duisburg-Essen.
- [Habiger 2018] Marc Habiger, Marius Feilhauer und Jürgen Häring (2018): “Systematically generated and complete tests for complex driving scenarios”. In: *4. Internationale ATZ-Fachtagung Automatisiertes Fahren*.
- [Hanke 2015] T. Hanke, N. Hirsenkorn, B. Dehlink, A. Rauch, R. Rasshofer und Erwin Biebl (2015): “Generic architecture for simulation of ADAS sensors”. In: *16th International Radar Symposium (IRS)*. IEEE, S. 125–130.
- [Kurakin 2017] Alexey Kurakin, Ian Goodfellow und Samy Bengio (2017): “Adversarial examples in the physical world”. In: *International Conference on Learning Representations*.
- [Mitsch 2013] Stefan Mitsch, Khalil Ghorbal und André Platzer (2013): “On provably safe obstacle avoidance for autonomous robotic ground vehicles”. In: *Proceedings of Robotics: Science and Systems*.
- [Szegedy 2014] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan (2014): “Intriguing properties of neural networks”. In: *International Conference on Learning Representations*.

- [Ulbrich 2015] S. Ulbrich, T. Menzel, A. Reschka, Fabian Schuldt und Markus Maurer (2015): “Definition der Begriffe Szene, Situation und Szenario für das automatisierte Fahren”. In: *10. Workshop Fahrerassistenzsysteme FAS*.
- [Yang 2006] B. Yang, M. Schoor, R. Weigel, P. Wenig, T. Schöberl (2006): “Radarsystemtechnik und Radarsignalverarbeitung für Automobilanwendungen”. In: *VDE-Kongress 2006*. VDE Verlag GmbH, S. 303–308.

Hochschulschriften

- [Bachhuber 2011] Martin Bachhuber (2011): “Analyse und Modellierung der Funkausbreitung in Passagierkabinen von Großraumflugzeugen”. Dissertation. Erlangen: Universität Erlangen-Nürnberg.
- [Mayer 2008] Winfried Mayer (2008): “Abbildender Radarsensor mit sendeseitig geschalteter Gruppenantenne”. Fakultät für Ingenieurwissenschaften und Informatik. Dissertation. Ulm: Universität Ulm.
- [Neumann-Cosel 2014] Kilian von Neumann-Cosel (2014): “Virtual Test Drive. Simulation umfeldbasierter Fahrzeugfunktionen”. Lehrstuhl für Echtzeitsysteme und Robotik. Dissertation. München: Technische Universität München.
- [Olscher 2016] Michael Olscher (2016): “Prototyp zur Modellierung eines Radarsensors für den Einsatz in HiL-Test”. Institut für Kraftfahrzeuge. Masterarbeit. Aachen: RWTH.
- [Schnarr 2016] Benjamin Schnarr (2016): “Spezifikation und systematische Variation von Testszenerarien für Fahrerassistenzsysteme”. Institut für Luftfahrtsysteme. Masterarbeit. Stuttgart: Universität Stuttgart.
- [Siltanen 2010] Samuel Siltanen (2010): “Efficient physics-based room-acoustics modeling and auralization”. Dissertation. Espoo, Finnland: Aalto University School of Science and Technology.
- [Wang 2017] Siyu Wang (2017): “Integration manöverbasierter Testfallbeschreibungen in einem Hardware-in-the-Loop Aufbau für radarbasierte Notbremsassistenten”. Lehrstuhl für Fahrzeugtechnik. Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie.

Gesetze, Normen und Berichte

- [StVO 2013] Bundesministerium der Justiz und für Verbraucherschutz (2013): *Straßenverkehrs-Ordnung (StVO) — Anlage 2 (zu § 41 Absatz 1) — Vorschriftzeichen*. URL: https://www.gesetze-im-internet.de/stvo_2013/anlage_2.html.
- [DIN EN 61709] DIN EN, Hrsg. (1. Jan. 2012): *Elektrische Bauelemente – Zuverlässigkeit – Referenzbedingungen für Ausfallraten und Beanspruchungsmodelle zur Umrechnung*. Berlin: Beuth Verlag GmbH.
- [Modelisar 2014] *Functional Mock-up Interface for Model Exchange and Co-Simulation* (2014). MODELISAR consortium, Modelica Association Project “FMI”.
- [BAST 2012] Tom M. Gasser, Clemens Arzt, Mihiar Ayoubi, Arne Bartels, Lutz Bürkle (2012): *Rechtsfolgen zunehmender Fahrzeugautomatisierung*. Bergisch Gladbach: Bundesanstalt für Straßenwesen.
- [ISO 17361] ISO, Hrsg. (1. Feb. 2007): *Intelligent transport systems — Lane departure warning systems — Performance requirements and test procedures*. Version 1. Genf.
- [ISO 17387] ISO, Hrsg. (1. Mai 2008): *Intelligent transport systems — Lane change decision aid systems (LCDAS) — Performance requirements and test procedures*. Version 1. Genf.
- [ISO 15622] ISO, Hrsg. (15. Apr. 2010): *Intelligent transport systems - Adaptive Cruise Control systems - Performance requirements and test procedures*. Version 2. Genf.
- [ISO 3888] ISO, Hrsg. (15. März 2011a): *Passenger cars — Test track for a severe lane-change manoeuvre*. Version 2. Genf.
- [ISO 26262] ISO, Hrsg. (15. Nov. 2011b): *Road vehicles — Functional safety*. Version 1. Genf.
- [ISO 22839] ISO, Hrsg. (1. Juni 2013a): *Intelligent transport systems — Forward vehicle collision mitigation systems — Operation, performance, and verification requirements*. Version 1. Genf.
- [ISO 15623] ISO, Hrsg. (15. Juli 2013b): *Intelligent transport systems — Forward vehicle collision warning systems — Performance requirements and test procedures*. Version 2. Genf.

- [SAE 2014] *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems* (2014). SAE On-Road Automated Vehicle Standards Committee.
- [United Nations 2014] United Nations (2014): *Report of the Working Party on the Strategic Framework and the Programme Budget on its sixty-eighth session*. Genf: Economic Commission for Europe, Inland Transport Committee, Working Party on Road Traffic Safety.
- [Weitzel 2014] Alexander Weitzel, Hermann Winner, Cao Peng, Sebastian Geyer, Felix Lotz und Mohsen Sefati (2014): *Absicherungsstrategien für Fahrerassistenzsysteme mit Umfeldwahrnehmung*. Bergisch Gladbach: Bundesanstalt für Straßenwesen.
- [Destatis 2017] *Zeitreihen - Verkehrsunfälle* (2017). Statistisches Bundesamt.

Internetdokumente

- [Assendorpf 2016] Dirk Assendorpf und Erik Coelingh (2016): *Es kommt der Tag, an dem wir sagen: Jetzt musst du nicht mehr auf den Verkehr achten*. ZEIT ONLINE GmbH. URL: <https://www.zeit.de/2016/29/selbstfahrende-autos-volvo-erik-coelingh> (besucht am 10.05.2018).
- [AUDI AG 2017] AUDI AG (2017): *Audi pre sense. Audi Technology Portal*. URL: <http://www.audi-technology-portal.de/de/elektrik-elektronik/sicherheitssysteme/audi-pre-sense> (besucht am 12.01.2017).
- [3DS MAX] Autodesk (2018): *3DS MAX. Software für Modellierung, Animation und Visualisierung in 3D*. URL: www.autodesk.de/products/3ds-max (besucht am 07.05.2018).
- [Bhuiyan 2016] Johana Bhuiyan (2016): *The complete timeline to self-driving cars. Self-driving cars are coming. The question is when and how*. URL: <http://www.recode.net/2016/5/16/11635628/self-driving-autonomous-cars-timeline> (besucht am 06.06.2016).
- [BMW 2013] BMW AG (2013): *Aktive Geschwindigkeitsregelung mit Stop & Go-Funktion. BMW Techniklexikon*. URL: https://legacy.bmw.com/com/de/insights/technology/technology_guide/articles/active_cruise_control_stop_go.html?source=index&article=active_cruise_control_stop_go (besucht am 30.04.2018).
- [Breitinger 2016] Matthias Breitinger (2016): *Kabinett erlaubt teilautomatisiertes Fahren. Autonomes Fahren*. ZEIT ONLINE GmbH. URL:

- <http://www.zeit.de/mobilitaet/2016-04/autonomes-fahren-gesetzentwurf-verkehrsrecht-alexander-dobrindt> (besucht am 21. 12. 2016).
- [Cameron 2017] Oliver Cameron (2017): *An Introduction to LIDAR: The Key Self-Driving Car Sensor*. Voyage. URL: <https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff> (besucht am 25. 04. 2018).
- [Dabrock 2017] Peter Dabrock (2017): *Ethische Dilemmata unterlaufen oft Akzeptanz von autonomen Fahrzeugen*. Tagesspiegel. URL: <https://causa.tagesspiegel.de/gesellschaft/autonomes-fahren-sind-wir-bereit-fuer-selbstfahrende-autos/ethische-dilemmata-unterlaufen-oft-akzeptanz-von-autonomen-fahrzeugen.html> (besucht am 28. 03. 2017).
- [Daimler-AG 2018] Daimler-AG (2018): *Driving assistance systems: Connected with all senses*. URL: <http://media.daimler.com/marsMediaSite/en/instance/ko/Driving-assistance-systems-Connected-with-all-senses.xhtml?oid=9904890> (besucht am 10. 10. 2017).
- [DPMA 2016] Deutsches Patent- und Markenamt (2016): *Anzahl der Patentanmeldungen in Deutschland nach Unternehmen*. Hrsg. von Statista. URL: <https://de.statista.com/statistik/daten/studie/258128/umfrage/anzahl-der-patentanmeldungen-in-deutschland-nach-unternehmen/> (besucht am 09. 10. 2017).
- [INCA-FLOW] ETAS GmbH (2018): *INCA-FLOW. Guided Calibration and Automation*. URL: https://www.etas.com/en/products/inca_flow.php (besucht am 10. 05. 2018).
- [Euro NCAP 2017] Euro NCAP (2017): *Safety Assist*. URL: <https://www.euroncap.com/en/for-engineers/protocols/safety-assist/> (besucht am 05. 10. 2017).
- [Fasse 2018] Markus Fasse und Britta Weddeling (2018): *Autonomes Fahren: Tödlicher Unfall bei Uber schockt die Autobranche*. Hrsg. von Handelsblatt. URL: <http://www.handelsblatt.com/my/unternehmen/industrie/autonomes-fahren-toedlicher-unfall-bei-uber-schockt-die-autobranche/21090846.html> (besucht am 08. 05. 2018).
- [Gebhardt 2003] Nikolaus Gebhardt (2003): *Einige BRDF modelle*. URL: <http://www.irrlicht3d.org/papers/BrdfModelle.pdf> (besucht am 15. 05. 2016).
- [KITTI Setup] Andreas Geiger, Philip Lenz, Christoph Stiller und Raquel Urtasun (2018a): *The KITTI Vision Benchmark Suite. Sensor Setup*. Karlsruher Institut für Technologie. URL: <http://www.cvlibs.net/datasets/kitti/setup.php> (besucht am 31. 01. 2018).

- [KITTI 2018] Andreas Geiger, Philip Lenz, Christoph Stiller und Raquel Urtasun (2018b): *The KITTI Vision Benchmark Suite*. Karlsruher Institut für Technologie. URL: <http://www.cvlibs.net/datasets/kitti/> (besucht am 31. 01. 2018).
- [Google LLC 2018a] Google LLC (2018a): *Protocol Buffers*. URL: <https://developers.google.com/protocol-buffers> (besucht am 07. 05. 2018).
- [TensorFlow] Google LLC (2018b): *TensorFlow. An open-source machine learning framework for everyone*. URL: www.tensorflow.org (besucht am 08. 05. 2018).
- [Google LLC 2018c] Google LLC (2018c): *Tensorflow Object Detection API*. URL: https://github.com/tensorflow/models/tree/master/research/object_detection (besucht am 10. 05. 2018).
- [Guhlich 2017] Anne Guhlich (2017): *Interview mit Bosch-Chef Denner. „Das ist extrem belastend“*. Hrsg. von Stuttgarter Zeitung. URL: <http://www.stuttgarter-zeitung.de/inhalt.interview-mit-bosch-chef-denner-das-ist-extrem-belastend.3e58b4cf-9eea-4554-b06c-d3f976961912.html> (besucht am 09. 10. 2017).
- [Hanke 2017] T. Hanke, N. Hirsenkorn, C. vn-Driesten, P. Garcia-Ramos, M. Schiemenz und S. Schneider (2017): *Open Simulation Interface. A generic interface for the environment perception of automated driving functions in virtual scenarios*. URL: <https://www.hot.ei.tum.de/forschung/automotive-veroeffentlichungen/> (besucht am 10. 10. 2017).
- [CarMaker] IPG Automotive GmbH (2017): *CarMaker*. URL: <https://ipg-automotive.com/products-services/simulation-software/carmaker/> (besucht am 07. 05. 2018).
- [Knezevic 2018] Nikola Knezevic (2018): *Foto: Tree on a mountain plain*. URL: <https://unsplash.com/photos/RfSo95x1tj8> (besucht am 01. 07. 2018).
- [Köllner 2017] Christiane Köllner (2017): *Autohersteller setzen auf Lidar-Sensoren*. Hrsg. von Springer Professional. URL: <https://www.springerprofessional.de/sensorik/automatisiertes-fahren/autohersteller-setzen-auf-lidar-sensoren-/12051902> (besucht am 24. 04. 2018).
- [Lagandré 2017] Arnaud Lagandré und Thomas Laux (2017): *Lidar ganz ohne bewegliche Teile: 3D-Flash-Lidar*. Hrsg. von all-electronics.de. URL: <http://www.all-electronics.de/lidar-ganz-ohne-mechanik-3d-flash-lidar> (besucht am 24. 04. 2018).

- [Simulink] MathWorks (2018): *Simulink. Simulation und Model-Based Design*. URL: www.mathworks.com/products/simulink.html (besucht am 12.02.2018).
- [Mazzega 2017] Jens Mazzega (2017): *Über PEGASUS*. Hrsg. von PEGASUS-Projektbüro. URL: <http://www.pegasus-projekt.info/de/about-PEGASUS> (besucht am 04.10.2017).
- [Michelin 2017] Michelin (2017): *ViaMichelin. Stuttgart-München*. URL: <https://www.viamichelin.com/web/Routes> (besucht am 10.10.2017).
- [PKW 2018] MikesPhotos (2016): *Foto: BMW E65 Arlon red aluminium*. URL: <https://pixabay.com/de/bmw-auto-fahrzeug-transport-1274297> (besucht am 10.05.2018).
- [Modelica] Modelica Association (2018): *Modelica*. URL: www.modelica.org (besucht am 12.02.2018).
- [OptiX] NVIDIA Corporation (2018): *NVIDIA® OptiX™ Ray Tracing Engine. A software development kit for achieving high performance ray tracing on the GPU*. URL: <https://developer.nvidia.com/optix> (besucht am 26.02.2018).
- [Ohnsman 2017] Alan Ohnsman (2017): *Waymo Taps Phoenix For A Big Public Test Of Google Self-Driving Car Tech*. Forbes. URL: <https://www.forbes.com/sites/alanoohnsman/2017/04/25/waymo-taps-phoenix-for-a-big-public-test-of-google-self-driving-car-tech/#4663be086141> (besucht am 06.10.2017).
- [Porada 2019] Will Porada (2019): *Foto: Imperfect Stop Sign*. Unsplash. URL: <https://unsplash.com/photos/ZaGcU6BxJEC/info> (besucht am 04.01.2019).
- [Reidl 2018] Andrea Reidl (2018): *"Rettungsdienste werden viel weniger Unfallopfer zu versorgen haben". Selbstfahrende Autos*. Hrsg. von ZEIT ONLINE GmbH. URL: <https://www.zeit.de/mobilitaet/2018-04/selbstfahrende-autos-rettungsdienste-gesellschaft-veraenderung-andreas-herrmann/komplettansicht> (besucht am 02.05.2018).
- [Robert Bosch GmbH 2018] Robert Bosch GmbH (2018): *Ultraschallsensor. Umfeldsensor zur Ermittlung von Abständen zu Hindernissen und zur Überwachung des Raums beim Parken und Rangieren*. URL: <https://www.bosch-mobility-solutions.de/de/produkte-und-services/pkw-und-leichte-nutzfahrzeuge/fahrerassistenzsysteme/baustellenassistent/ultraschallsensor/> (besucht am 07.05.2018).

- [Simanowski 2017] Roberto Simanowski (2017): *Künstliche Intelligenz: Der Todesalgorithmus*. ZEIT ONLINE GmbH. URL: <http://www.zeit.de/kultur/2017-09/kuenstliche-intelligenz-algorithmus-spam-autonomes-fahren/komplettansicht> (besucht am 10.10.2017).
- [Blender] Stichting Blender Foundation (2018): *Blender. Open Source 3D creation*. URL: www.blender.org (besucht am 07.05.2018).
- [PreScan] Tass International (2017): *PreScan. Simulation of ADAS and active safety*. URL: <https://www.tassinternational.com/prescan> (besucht am 10.10.2017).
- [OpenSceneGraph] *The OpenSceneGraph Project Website* (2015). URL: <http://www.openscenegraph.org/> (besucht am 06.07.2015).
- [The Tesla Team 2016] The Tesla Team (2016): *All Tesla Cars Being Produced Now Have Full Self-Driving Hardware*. Hrsg. von Tesla Motors. URL: https://www.tesla.com/de_AT/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware (besucht am 05.12.2016).
- [Treuille 2009] Adrien Treuille (2009): *Advanced Ray Tracing*. Carnegie Mellon University. URL: <https://www.cs.cmu.edu/afs/cs/academic/class/15462-s09/www/lec/01/lec01.pdf> (besucht am 10.05.2018).
- [Straße 2015] User Free-Photos (2015): *Foto: Straße-nacht-langzeitbelichtung*. pixabay. URL: <https://pixabay.com/de/strasse-nacht-langzeitbelichtung-2737288> (besucht am 10.05.2018).
- [Van Rossom 2018] Van Rossom und Quentin (2018): *Mitsubishi EVO X Rally*. URL: <https://www.artstation.com/artwork/BRPZ4> (besucht am 23.04.2018).
- [Velodyne Lidar] Velodyne LiDAR (2018): *Our Products*. URL: <http://www.velodynelidar.com/products.html> (besucht am 25.04.2018).
- [VDA 2017] Verband der Automobilindustrie, Hrsg. (2017): *Innovation und Technik*. URL: <https://www.vda.de/de/themen/innovation-und-technik.html> (besucht am 02.10.2017).
- [OpenDRIVE] VIRES Simulationstechnologie GmbH (2015): *OpenDRIVE*. URL: <http://www.opendrive.org/> (besucht am 09.09.2015).
- [VIRES Simulationstechnologie GmbH 2016] VIRES Simulationstechnologie GmbH (2016): *VTD Perfect Sensor*. URL: <https://www.vires.com/products.html> (besucht am 05.09.2016).

- [Volkswagen 2016] Volkswagen (2016): *Parklenkassistent „Park Assist“*. VW *Technik-Lexikon*. URL: http://www.volkswagen.de/de/technologie/techniklexikon/parklenkassistent_park_assist.html (besucht am 12. 11. 2016).
- [Volvo Cars 2017] Volvo Cars (2017): *Drive Me. the self-driving car in action*. URL: <https://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving/drive-me#> (besucht am 06. 10. 2017).
- [Walker 2017] Jon Walker (2017): *The Self-Driving Car Timeline. Predictions from the Top 11 Global Automakers*. Hrsg. von Tech Emergence. URL: <https://www.techemergence.com/self-driving-car-timeline-themselves-top-11-automakers/> (besucht am 02. 05. 2018).
- [Weihard 2010] Sebastian Weihard (2010): *Radarsensorik in der Fahrzeugtechnik*. Fernuniversität Hagen. URL: <http://www.fernuni-hagen.de/imperia/md/content/fakultaetfuermathematikundinformatik/agjob/vortrag-weihard.pdf> (besucht am 26. 04. 2018).
- [Xan Griffin 2018] Xan Xan Griffin (2018): *Foto: Long Drive Chilling*. Unsplash. URL: <https://unsplash.com/photos/Fe6iP0UX810> (besucht am 08. 05. 2018).
- [Zapata 2017] Sebastian Zapata (2017): *Through the Woods*. URL: <https://blenderartists.org/forum/showthread.php?424459-Through-The-Woods> (besucht am 23. 04. 2018).
- [Zeit Online 2016] ZEIT ONLINE GmbH (2016): *Tesla-Autopilot hielt Lkw für Verkehrsschild*. URL: <http://www.zeit.de/mobilitaet/2016-07/autonomes-fahren-tesla-unfall-model-s-autopilot-software> (besucht am 14. 10. 2016).

