Virtual Sensorics: Simulated Environmental Perception for Automated Driving Systems

Timo Hanke

München 2020

TECHNISCHE UNIVERSITÄT MÜNCHEN
Professur für Höchstfrequenztechnik

Virtual Sensorics: Simulated Environmental Perception for Automated Driving Systems

Timo Hanke

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Ulrich Wagner
Prüfer der Dissertation:
1. Prof. Dr.-Ing. Erwin Biebl
2. apl. Prof. Dr.-Ing. Walter Stechele

Die Dissertation wurde am 19.11.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 21.04.2020 angenommen.

Abstract

The level of sophistication and the scope of machine control in current and upcoming automated driving systems are steadily on the rise. On the road towards highly and fully automated driving, this new complexity poses significant challenges for development and validation procedures of these systems. A virtual world provides an environment that is safe and completely controlled with perfectly reproducible conditions. These are the perfect qualifications to facilitate and accelerate the development, testing, and validation of automated driving systems. One of the core components of an automated driving system is the perception of the vehicle's environment. This environmental perception is performed by sensors mounted on the vehicle. They generate the necessary input data that ultimately serve as input for the decision making units of the automated driving system. In a virtual environment, sensor models serve the function of providing data that match as close as possible the data from the actual vehicle sensors in a corresponding real world scenario. Ideally, the subsequent components in the processing chain are not able to distinguish whether they are operating in a virtual environment or the real world. As perception sensors like radar, lidar, or camera sensors generally provide only a lossy description of the environment, the purpose of the sensor model is to capture and reproduce these characteristics for the perception of the virtual environment.

Investigating virtual environmental perception is the focus of this work. For this purpose, different types of sensor models are defined according to their level of abstraction of the perception process: Statistical models on the one hand comprise the whole perception process in a unified model and generally operate on higher abstraction levels like that of object lists. Physical models on the other hand deal with the measurement process exclusively and generate sensor-specific low level data that can be used as input for a perception function to obtain higher level output. In this work, a special emphasis is placed on the architecture of the virtual environmental perception processing chain and its interfaces. For the construction of statistical sensor error models, a modular architecture is defined. A classification scheme for the various types of sensor errors that can be identified in real sensor characteristics is proposed and matched with the modular architecture. The meaning of statistical equivalence of sensor output between real and virtual sensors is closely investigated for several exemplary quantities measured by vehicle sensors. Additionally, a physical sensor model for an automotive lidar sensor is introduced. The model is based on a ray tracing implementation for the generation of a simulated point cloud as low level sensor output on the basis of the three-dimensional geometry of the virtual world. The simulated point cloud is then coupled with subsequent processing functions that explicitly build on the low level sensor interfaces for the construction of an internal environmental representation of the vehicle.

Danksagung

An dieser Stelle möchte ich allen an der Entstehung dieser Arbeit beteiligten Leuten von Herzen danken. Dies sind insbesondere meine beiden thematischen Doktorandenkollegen Nils und Alex, die mich auf dieser Reise im Zuge ihrer eigenen Arbeiten zur Simulation von Sensoren begleitet haben. Ebenfalls gilt mein Dank all meinen Kollegen bei BMW, welche ich im Laufe dieser Arbeit kennen lernen durfte. Vorrangig dabei sind Bernhard, Andreas, und Carlo, welche immer für fachliche Diskussionen offen waren, sowie das Team der Fahrsimulation für ihre großartige Hilfestellung bei der initialen Inbetriebnahme der Simulationen zu einem Zeitpunkt als ich noch ganz neu zu diesem Thema gekommen war. Weiterhin gilt mein Dank allen Leuten am Fachgebiet Höchstfrequenztechnik der TUM, Herrn Prof. Biebl sowie der herzlich-freundlichen Gruppe an Doktoranden, in deren Mitte ich mich immer gut aufgehoben fühlen und so manches Neue dazulernen konnte. Danken möchte ich auch der großartigen Gruppe des BMW Doktorandenkreises, welche eine Bereicherung durch die gesamte Zeit und der Anlass für eine Vielzahl von schönen Momenten waren, die mir lange in Erinnerung bleiben werden.

Schließlich und allen voran danke ich meiner Familie, meinen Freunden, und ganz besonders meiner Partnerin Sina, welche mich alle zusammen tatkräftig auf diesem, durchaus langen Weg bis zum finalen Abschluss dieser Arbeit unterstützt haben.

Contents

1 Introduction

The advent of advanced computational capabilities in embedded and mobile devices brought with it a newfound wealth of potential applications in vehicle technology. Among these is the potential of an automated system taking over an increasing share of driving responsibilities, going hand in hand with increased safety and comfort for the driver. In the long term, these developments are expected to have fundamental implications for the automotive industry at large up to a general revolution of personal mobility [1].

In order to fully realize the technology's potential for automated driving, a plethora of unresolved challenges have to be tackled and solved. Among the core challenges is the system's ability to perceive, understand, and react to its dynamic environment, an environment that typically involves a multitude of interacting road users of all kinds. The foundation for this ability is formed by sufficiently advanced environmental perception that is performed by gathering raw detections through the vehicle's sensors and processing and interpreting these sensory input data. The development and validation of these sensors and systems for environmental perception are key to moving forward on the road to higher degrees of automation in on-road driving.

1.1 Driving Automation Systems

The development of all driving automation systems is motivated by two central goals: the safety and the comfort of road users. In this context, the term road users spans the full spectrum from the vehicle's driver and the other passengers to any other traffic participant, regardless whether they are motorized or not. Originally starting from active safety systems, increasing levels of automation of the dynamic driving task take load off the driver or provide the driver with additional information. A detailed discussion of the history of driving automation systems can be found in [1, 2].

In order to establish a common vocabulary on the road to automated driving systems, the Society of Automotive Engineers (SAE International) has defined a taxonomy consisting of six levels of varying degree of driving automation [3]. The definition of these automation levels centers on the respective roles of (human) driver and driving automation system in the driving task. They therefore describe quite well a specific system or feature from the human as well as the technical perspective by defining guidelines of the tasks expected to be performed by the human and the driving automation system.

Shown as an overview in Fig. 1.1, the meaning of the individual levels is as follows:

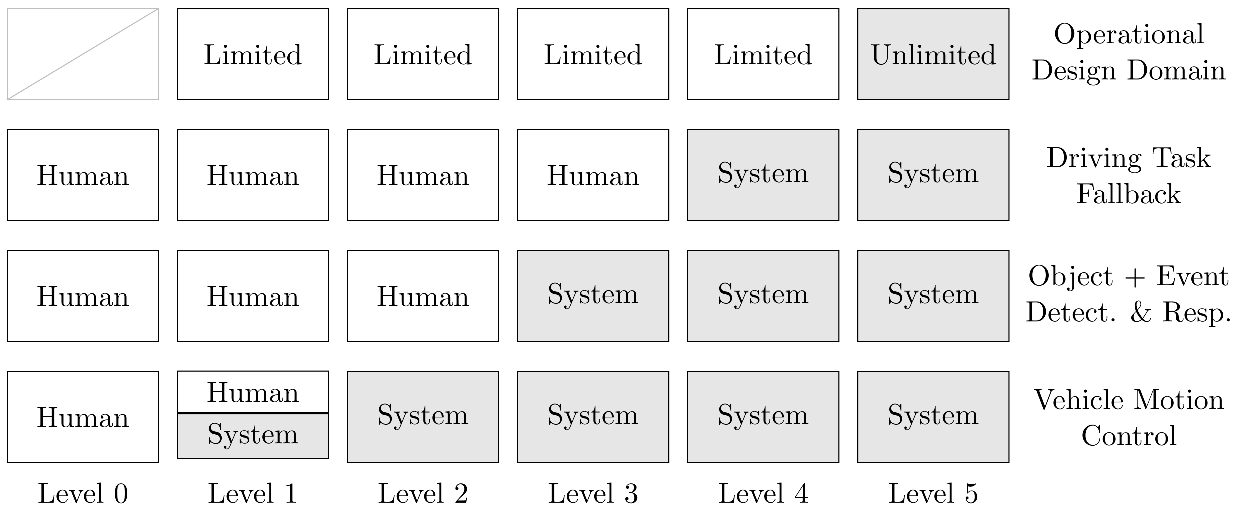


Figure 1.1: Levels of driving automation as specified by SAE J3016$^{\mathrm{TM}}$ [3].

- **Level 0 – No Driving Automation**: The driver performs the entire dynamic driving task. The driver may be aided by active safety systems that have a short-term influence on vehicle control, e.g. electronic stability control or automated emergency braking. Further, the driver may be given additional information or warnings, e.g. from a navigation system or lane departure warnings. Systems like cruise control may relieve the driver from having to continuously perform a specific action, e.g. keeping up pressure on the throttle pedal. However, this level is defined by the absence of any system that takes over any part of the dynamic driving task on a sustained basis.

- **Level 1 – Driver Assistance**: A driving automation system performs either the longitudinal or lateral vehicle motion control subtask in a sustained manner while the driver performs the rest of the dynamic driving task. This includes adaptive cruise control (longitudinal) or lane keeping (lateral) assistance systems.

- **Level 2 – Partial Driving Automation**: A driving automation system performs both longitudinal and lateral vehicle motion control while at all times supervised by the driver. The driver is also responsible for object and event detection and response, especially when the system's behavior may lead to a dangerous situation.

- **Level 3 – Conditional Driving Automation**: An automated driving system performs the entire dynamic driving task while engaged. This includes full vehicle motion control as well as object and event detection and response. The driver's role is that of a fallback-ready user who must be receptive at all times to timely requests for takeover by the system.

- **Level 4 – High Driving Automation**: In its specific operational design domain, the automated driving system performs the entire dynamic driving task including fallback. In case of failures or exit of its operational design domain, the

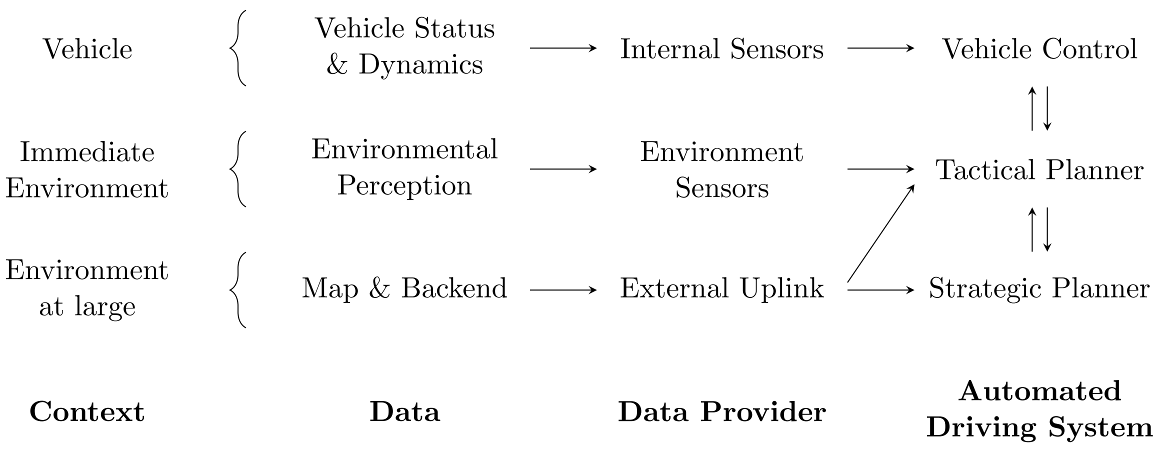


Figure 1.2: Schematic overview of data streams for automated driving systems.

system's fallback capabilities need to be able to achieve a minimal risk condition (a "safe state") in any situation. At this level, the driver becomes a passenger in the vehicle while the system is engaged and is not expected to be available as a fallback.

- **Level 5 – Full Driving Automation**: The automated driving system performs the entire dynamic driving task and fallback without restriction to a specific operational design domain. The system is expected to be able to operate the vehicle under all on-road conditions manageable by a human driver.

Within this six-level taxonomy all systems that automate at least some part of the dynamic driving task for a sustained period of time, i.e. level one or higher, are termed *driving automation systems*. In contrast, the expression *automated driving systems* applies only to those systems that may perform the dynamic driving task entirely for a sustained period of time, i.e. level three or higher. Throughout this work the expressions *automated driving system* or *automated driving feature* are used exclusively to highlight the focus on this type of system. In many cases, the discussion may also be applicable more generally to driving automation systems without explicit mention. In cases where a logical separation from the environmental perception is required, the expression *automated driving feature* is used to specifically refer to the part of the system that is responsible for determining the automated driving response, e.g. trajectory planning, based on the input provided by the environmental perception.

The input data streams used by automated driving systems can be divided into three general contexts depending on the source and scope of data: The vehicle specific context encompasses internal state data and in particular includes measurements of the vehicle dynamics. The context of environmental perception is comprised of data of the immediate environment of the vehicle as it is perceived by that vehicle. Finally, the context of the environment at large includes data provided by maps or from an uplink to an external source. An overview of these three input stream contexts and their relation to parts of the automated driving system is shown in Fig. 1.2.

Data in the vehicle specific context come from in-vehicle sensors that measure the dynamic state of the vehicle, e.g. wheel speed sensors and inertial measurement units for odometry measurements. This type of data is especially important on the operational level, i.e. vehicle motion control, for automated driving systems. It also enables active safety systems including anti-lock brake system (ABS) and electronic stability control, as well as comfort features such as cruise control. Note that all of these systems by themselves are categorized as level 0 (no driving automation) under the SAE J3016$^{\text{TM}}$ taxonomy (see Fig. 1.1).

Data provided by maps and external sources, including location information on the basis of global navigation satellite systems (GNSS), are used in order to provide information to the driver. For automated driving systems, these data are important on the tactical level, e.g. to know (in advance) the number and semantic role of lanes at a specific location or the location of accidents, as well as on the strategic level, e.g. for navigation and route planning.

Data provided by environmental perception are used to respond to the immediate environment of the vehicle. This includes reaction to the layout of the vehicle's surroundings and fixed obstacles (static environment) as well as the behavior of other road users (dynamic environment). While input data from all data stream contexts in Fig. 1.2 are essential, all automated driving systems rely fundamentally on the data provided by environmental perception. Without this input their operation is straight out impossible as reactions to the dynamic surroundings cannot be performed. Because of this central position, the environmental perception in vehicle systems in particular has seen a wealth of research activity over recent years. This has enabled great strides in driving automation, allowing several seconds of hands-off time with longitudinal and lateral control of the vehicle managed by a driving automation system in series production vehicles, and enabling research vehicles to navigate specific scenarios completely on their own [4, 5, 6].

1.2 Environmental Perception

A vehicle navigating the complexity of on-road traffic in a typical road environment has to react to and interact with a variety of different road users as well as static infrastructure. In order to handle this complexity, an automated driving system first needs to have a clear, comprehensive, and up-to-date internal representation of its environment. This is the goal of environmental perception. The system's perception has to achieve a high level of performance under a multitude of different environmental conditions. This then forms the foundation for higher levels of automation in automated driving systems that aim to increase safety and comfort for all road users. The increasing abilities of environmental perception within vehicle systems give rise to a number of increasingly sophisticated driving automation systems, summarized well in reference [7]. In terms of road user safety, the ultimate goal of these systems is formulated by the Vision Zero: "Vision Zero means that eventually no one will be killed or seriously injured within the road transport system" [8].

Environmental perception is performed by sensors mounted on the vehicle. The sensors consist of measurement units that interact with the physical environment of the vehicle as well as processing units that interpret the measurement data to derive a semantic representation of the environment. Using the fused data from multiple individual vehicle sensors, a central environment model can be constructed that underlies the automated driving feature for higher levels of automation. Typical automotive sensors for the environmental perception in automated driving systems are radar, lidar, camera, and ultrasound sensors [2, 4, 9, 5, 6]. Fig. 1.3 shows a schematic view of multiple sensors mounted at different points on a vehicle and their approximate fields of view covering the area around the vehicle.

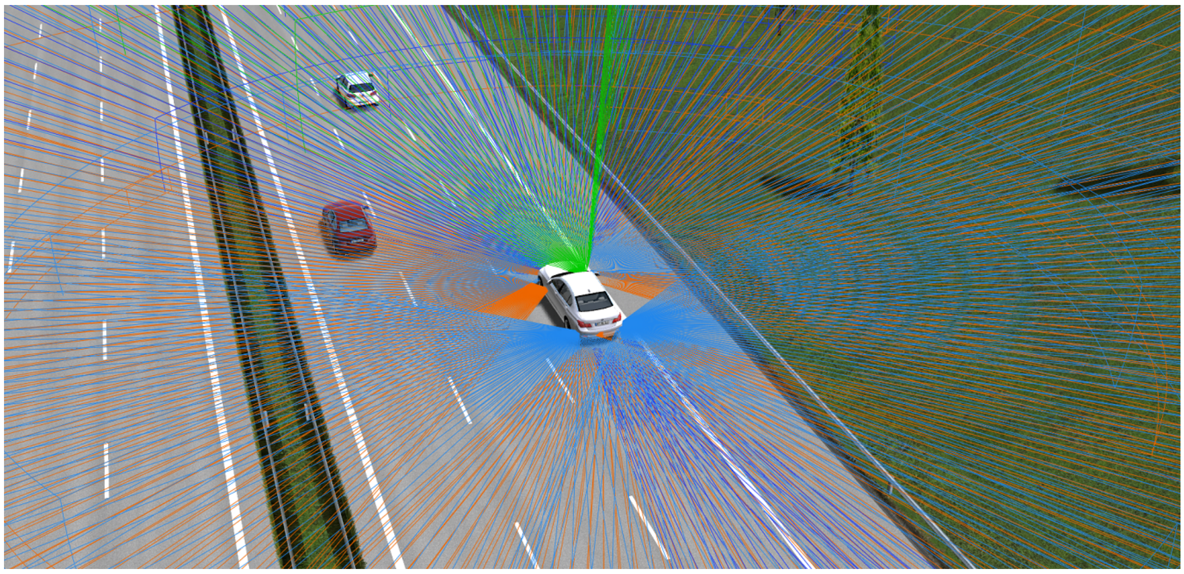


Figure 1.3: Schematic, exemplary view of the various sensors around a vehicle with a driving automation system. The picture is taken from a highway scene within a virtual environment. The colored sensor cones show each sensors approximate field of view. The different colors indicate different types of sensors (radar, lidar, and camera sensors).

Sensors based on the radio detection and ranging (*radar*) principle have been used in the automotive industry since the introduction of adaptive cruise control systems in 1999 [1, 10, 11]. Automotive *radar sensors* are active sensors, i.e. they transmit a signal and detect the echo response. Typical operating frequencies are 24 GHz and 76 GHz, with the former being phased out from active use. Radar sensors are especially useful to detect objects, especially metallic targets like vehicles, and in addition to object positions can measure object speed directly using the Doppler effect. Furthermore, radar sensors can achieve high detection ranges, exceeding 200 m, and are largely unaffected by adverse environmental conditions like rain, snow, or fog that trouble human drivers and optical sensors. However, they suffer from low angular resolution due to the usually very restricted package size for automotive use. Further, object classification based on the radar cross section is generally not highly reliable.

Operating in a different part of the electro-magnetic spectrum, light detection and ranging (*lidar*) *sensors*, sometimes also referred to as laser detection and ranging (ladar) sensors, typically use wavelengths in the near infrared (around 905 nm). These wavelength provide increased eye-safety and better attenuation properties than light in the visible spectrum. Lidar sensors, usually in the form of laserscanner technology with mostly mechanical scanning, enjoy widespread use in research vehicles [12, 4, 9, 5], yet at the current time are virtually non-existent in series mass production vehicles as even early forms of automated driving systems have not yet passed homologation. Related techniques to laserscanner technology like structured light scanning have also seen advances [13], but are currently still unsuited to the automotive use case. Although their use is still in the early stages, lidar sensors are prime candidates to enable automated driving systems due to their high angular and distance resolution that allows detection of features and therefore classification of objects, obstacles, and drivable terrain with high fidelity. For this reason, there is increased attention on developments to make lidar technology fit for automotive series production, by aiming to lower both size and form factor [14, 15, 16, 17, 18, 19]. As lidar sensors are optical sensors, a major downside compared to millimeter wave radar sensors is their susceptibility to adverse weather influences that can seriously degrade their performance in rainy, foggy, or snowy conditions [20].

Camera sensors use a different operating principle. These are passive sensors, where the detector converts incoming light from external illumination sources, e.g. the sun, street lamps, or the car's headlights, into brightness values in a two-dimensional pixel array. With the exception of the specialty night-vision systems, cameras in the automotive context operate in the spectrum of visible light. The recorded image frames require sophisticated image processing algorithms to extract usable information. This requires the availability of large computational resources, yet offers a very broad spectrum of information that is hardly or not at all available from other sensors. Besides object detection, this includes the detection and classification of traffic signs, traffic lights, and road markings. Further, object classification is usually superior due to the larger pool of detected features. The angular resolution is among the highest of any available sensor. On the downside, there is no direct measurement of distance, although stereo- or multi-camera systems can achieve remarkable depth accuracy, at least for shorter distances up to 20 or 30 m. Additionally, operating in the optical spectrum brings with it a high susceptibility to negative influences from bad weather conditions like rain, fog, or snow. Due to the dependency on external sources of illumination, there is also a major difference in sensor performance between day-time and night-time.

Ultrasound sensors are commonly used for short range obstacle detection. These acoustic sensors detect obstacles using the time-of-flight principle for distance measurements and a setup consisting of multiple transducers for angular resolution via triangulation. This type of sensor is commonly used for parking assistance systems that first entered the market in the mid 1990s [21]. As ultrasound sensors are very small and cost effective, multiple transducers are used to cover the short range area around the vehicle. The shortcomings of the acoustic principle are the low range of just a couple of meters, the high susceptibility to external influences, e.g. acoustic damping of snow or dirt, and the inability to differentiate between obstacles that may be harmful or harmless to the vehicle, which may overly limit detected available driving space. Regardless of these shortcomings, their small cost and the widespread use of parking assistance systems make ultrasound sensors the most common environment sensor found in series production vehicles around the world today.

Further possible sources of information on the direct environment of the vehicle include sensors detecting the environmental state, e.g. temperature, rain, or time-of-day, a data connection to backend servers or direct vehicle-to-vehicle communication that transmit information about dangerous road conditions or other events. Additionally, information about the driver or other vehicle occupants may be gathered by internal monitoring sensors. This type of information is especially important if the system has to monitor the driver's ability to perform the driving task fallback or take over the driving task upon exit of the operational design domain in level 3 or 4 automated driving systems (see Fig. 1.1 for the overview of levels of automation).

Until recently, it has been common for a specific driving automation feature to be directly coupled to the input data of a specific sensor [2]. This coupling allows a separation of concerns during development, reducing complexity and allowing individual vendors to easily offer ready-made solutions for integration into a vehicle. However, with increasing functional complexity the requirements on the environmental perception have also rapidly increased. This requires a unified internal representation of the environment that combines data from all vehicle sensors in order to provide the functional logic algorithms with the required information for their decision making. The type of description typically in use today are lists of objects in the environment. These objects represent other road users, e.g. vehicles, bikes, pedestrians, and static obstacles in the environment. Additionally, a representation of drivable space is required for navigation by an automated driving system. For this a type of occupancy grid mapping is typically used that gives a probabilistic view on the vehicle's surroundings [22, 23, 24].

Due to the crucial role of environmental perception for automated driving systems, having a controlled environment for development and validation of the system is a large benefit. This role can be filled by the virtual environment of driving simulators. Environmental perception close to reality in the virtual environment is the task of sensor models. The type of sensor model required depends on the specific application and the goals of the simulation. The focus of this thesis is the discussion of sensor models and their role in the development and validation of automated driving systems.

1.3 Thesis Scope and Goals

In the quest to enable automated driving systems to take over the dynamic driving task for extended periods of time, a major focus of research is the navigation of research vehicles through real world environments. This has been publicly demonstrated on numerous occasions [4, 5, 6]. Several challenges remain that have to be solved before automated driving systems leave the stage of research vehicles with trained safety drivers and become available to the general public. One of the central challenges is ensuring the safe conduct of the automated driving system in the highly diverse driving environment of roads around the world [1]. It is estimated that several billion kilometers of real world test drives are required to allow a statistically significant conclusion to be drawn regarding the safety of an automated driving system [25].

Using virtual environments with sensor models for environmental perception facilitates the advancement of this new technology by providing quick feedback on the viability and robustness of automated driving systems and features. The advantages of virtualization are having a controlled environment with reproducible conditions, the early availability as there is no need for hardware prototypes, and the ability to reconfigure or alter the system including sensor setup and sensor properties at the click of a button. In order to achieve the goal of increased system quality, it is important that within the virtual environment the input to the automated driving feature, i.e. the functional logic components of the automated driving system (see Fig. 1.2), matches the real world on-road scenario as faithfully as possible. To achieve this matching, there has been research dedicated to the construction of sensor models emulating the sensor behavior in the virtual environment. Some sensor models are based on a probabilistic approach [26, 27, 28, 29, 30, 31], and others are derived from a physical description of the measurement process [32, 33]. Similar sensor models have been in use outside of the automotive context for a longer period of time [34, 35]. Common to these models is an integrated approach that uses data from the virtual environment and directly transforms them to sensor output data by a variety of ways that each highlight the particular focus of each research project.

This thesis aims to provide a systematic approach to sensor models. This approach covers categorizing sensor models according their particular design domain depending on the application, for which they are used. Further, this approach deconstructs the building of sensor models into a series of steps with defined interfaces and individual, separate logical components. This lays guidelines for the creation of sensor models purpose-built for various applications as well as iterative extensions of models and the ability to integrate content from multiple sources or vendors into a common framework. The creation of abstraction layers that allow multi-vendor integration is particular focus of the research work. Building on this groundwork, within this thesis sensor model processing chains are constructed that perform real time generation of virtual sensor data and that are used to build an exemplary setup of an end-to-end tool chain of a driving simulation framework with a sensor model as an intermediate creating the input for an automated driving feature in the research stage. This work aims to provide the foundation for future research and development efforts in the creation of virtual world development and testing frameworks.

This thesis is structured as follows: A general framework for sensor models based on a modular approach is introduced and guidelines for the gathering of reference data and model parametrization are outlined in chapter 2. In chapter 3, the probabilistic approach of sensor error models built out of modular components is discussed that aims to reproduce the statistical properties of the perception process in the vehicle. Chapter 4 deals with sensor measurement models for a physical simulation of the measurement process and introduces a first example of a full processing chain using a bottom-up approach to sensor simulation to generate low level lidar point cloud data that are fed as input to the initial high level environmental perception function of an automated driving feature in a real time setup. Finally, the thesis closes with a summary in chapter 5 that completes the circle by highlighting the implications of this work and provides an outlook on potential future research opportunities building on this work.

2 Virtual Environmental Perception

In an automated driving system, the perception of the vehicle's environment provides one of the key input data streams for the automated driving feature, the decision making algorithms within an automated driving systems. In a virtual world setup, therefore, a proper model of the environmental perception is one of the most central aspects for the success of virtual development and testing. This virtual environmental perception is achieved by means of sensor models that generate synthetic sensor data. This chapter discusses sensor models in the context of virtual development and testing and provides an overview on the current state of the art in sensor models. To complete the chapter, this is closely followed by introducing the work of this thesis in defining different sensor model types with their respective applications and architecture as well as introducing a systematic, modular approach to the construction of sensor models with separated interfaces and logical components. As a new, open standard for the interface description, the open simulation interface (OSI), is introduced. These components serve as the groundwork on which the work of the subsequent chapters is built.

2.1 Virtual Development and Testing

Due to advancements in sensor technology and data processing algorithms in recent years, research is making great strides to enable automated driving systems that have the purpose of increased safety and comfort for the vehicle driver and occupants as well as other road users. Yet, due to their complexity, one of the main challenges remains ensuring and validating the safe conduct of automated driving systems in order to make them fit for public use [1, 25]. Extensive road tests, as carried out for driver assistance systems publicly available today, are certainly part of the solution. Yet, as the system has to autonomously handle an increasing variety and increasing complexity of situations, there is a tremendous rise in testing requirements [1]. To some degree this can be handled by making use of virtual world testing that, for this reason, is seeing an increasing amount of attention [36, 37, 38] This section discusses how real world and virtual world testing can complement each other to cope with the rising testing requirements for automated driving systems, and shows where virtual environmental perception comes into play in a development approach relying on virtual world testing.

2.1.1 Real World and Virtual World Testing

To see the contribution of simulations and virtual world testing in this context, the general outline of the development process needs to be considered. The development of

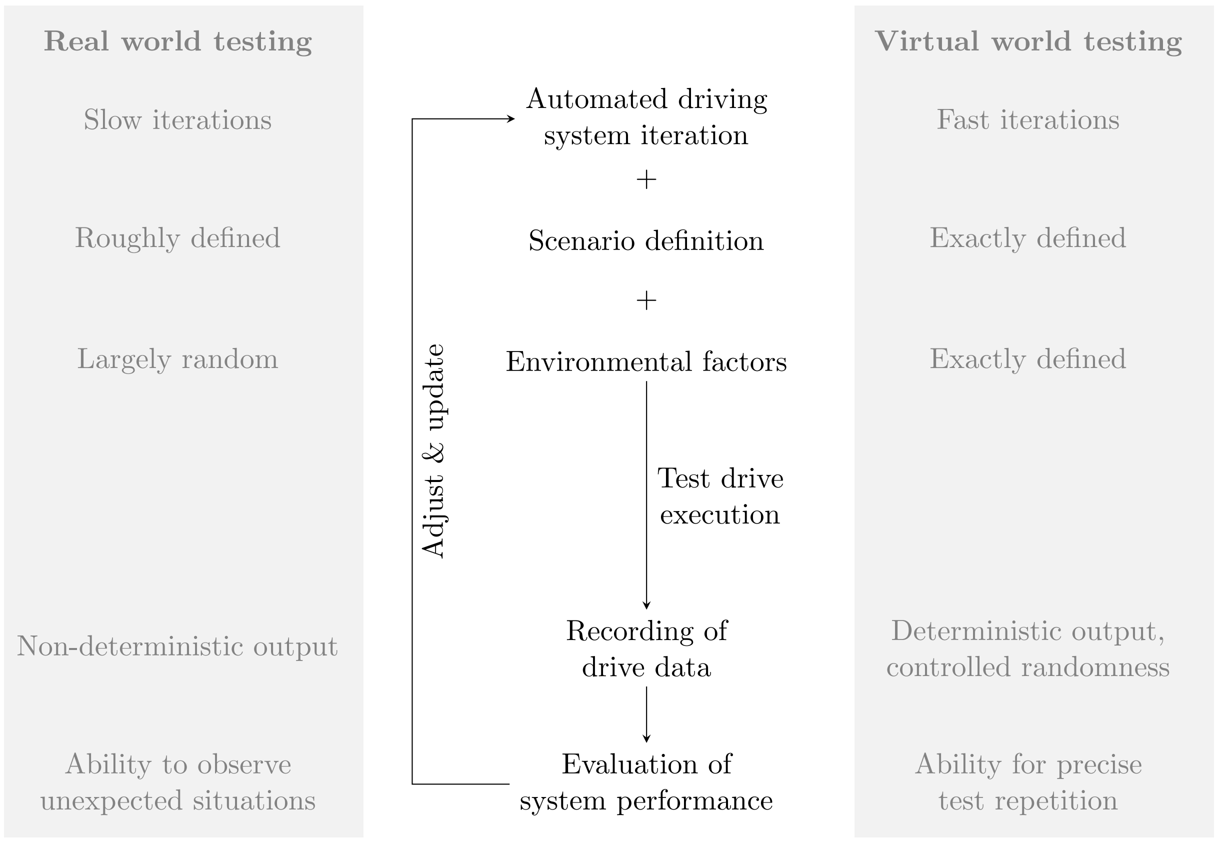


Figure 2.1: The process chart in the center illustrates the test and development process loop for iterative updates of an automated driving system based on data gathered during real world or virtual world test drives. Each test drive is the result of the current iteration of the automated driving system, the definition of the driving scenario, and the environmental factors at play during the test drive. The result of each test drive is a set of recorded data that is then used for evaluation of the system's performance. The properties of real world testing (left gray box) and virtual world testing (right gray box) are given for each part of input or output for comparison in order to highlight their difference. As a result both methods complement each other quite well in achieving the overall goal of overall system improvement.

automated driving systems, as with most other complex systems, follows an iterative, empirical process, where development increments are tested, the performance of the system evaluated, and adjustments made for the next iterations. A process chart of this test and update iteration cycle is shown in the center of Fig. 2.1. For clarity, the chart only shows a single process loop, whereas in a large scale development project typically a multitude of these loops are running in parallel, testing different parts of the system under development. For each test run, on an abstract level, the input can be summarized as consisting of three components: The first is the current iteration of the automated driving system, which stands for the combination of hardware versions and setup as well as software versions that defines the system at the current point in development. The second is the definition of the driving scenario, which outlines the general parameters of the test environment, e.g. the start and end points of the test run, road types, time of day, general weather conditions, and other parameters. Finally, the third is the sum of environmental factors, which summarize all outside influences that cannot be defined as part of the scenario. These include traffic conditions and pathways of other road users relevant to the automated driving system, detailed environmental influences like the current point of the sun, the placement of slippery road segments, and so on. With these three general components as input, the test drive is executed and drive data recorded. Based on these data, an evaluation of the system's performance under the circumstances it encountered during the test drive is performed and adjustments to the system are made. As shown in Fig. 2.1, this concludes one cycle of the iterative development loop .

Based on this general description of the underlying iterative development process, a comparison of real world and virtual world testing can be made. This is shown in Fig. 2.1 in the gray boxes to the left and right side of the process chart. Considering the three generalized input components to a test run, the defining differences between real world and virtual world testing become easily apparent: On the one hand, providing the test setup itself, i.e. the automated driving system, for a test run in the real world requires preparing hardware and software, safety measures, a dedicated test track or a license for public road tests, and more. On the other hand, providing the test setup for a virtual world test run can be fully automated to happen at the click of button, with the main effort required for the initial setup of the virtual environment. This leads to comparably much slower iterations in real world testing, while virtual world testing allows fast iterations in the system's development. For the scenario definition, in a real world test the scenario run can only be roughly defined as many parameters (e.g. surrounding traffic or weather conditions) cannot be controlled in detail and are instead part of the largely random environmental factors. This is in contrast to a simulation environment, which provides the ability to exactly define the scenario and all environmental factors. If required to sample a certain configuration space with a sequence of test runs, a well-defined stochastic variation can be applied.

On the output side, the limited control of inputs in real world test drives results in generally non-deterministic output. To get the full picture and allow analysis, the input parameters encountered by the system have to be recorded in addition to the system's behavior as a reference. However, this also has an important upside: due to the lack of full control of test parameters, at any time unexpected situations can be observed in real world testing. These can range from the microscopic, e.g. a temporary lack of internal data transmission from a sensor, to the macroscopic, e.g. encountering an unexpected obstacle or traffic situation. Virtual world testing, in contrast, shows fully deterministic output with controlled randomness, as defined for the input. This enables precise repetition of tests and allows precise variation of individual variables, either within the automated driving system or within the scenario or environmental input, and observing and evaluating the results. Given these different and complementing properties of real world and virtual world testing, neither can and should be the single way to go. Instead, splitting the testing effort between the two worlds can boost development speed as well as enhance system quality and accomplish a further step towards the goal of automated driving systems in large scale use.

2.1.2 Perception Models within the Development Process

From the line of thought in the previous discussion of real and virtual world testing, the benefits of virtual world testing complementing real world road tests become clear. Yet, as cars are ultimately meant to drive in the real world, two key components are required for meaningful virtual world testing: On the one hand, the driving scenario and road conditions in the virtual world have to closely follow conditions encountered in the real world, which is the task of proper generation of virtual testing scenarios as described in [39, 40, 41, 42]. On the other hand, the input data for the automated driving system generated by the simulation environment have to closely match the data it receives when installed in a real car under comparable real world conditions. The main input data are provided by the sensory perception of sensors such as radar, lidar, and camera systems mounted on the car. To generate realistic sensor data from the information in the virtual environment, a model of the generally lossy sensory perception process is required. There is an ongoing effort to build a standardized, cross-vendor tool chain for the test and validation of automated driving systems including a simulation tool chain with models of the sensory perception as part of the European cooperation project *Pegasus*. Some of the work of this thesis, especially regarding the proposal of standardized interfaces for such a tool chain presented in the last section of this chapter, has in part been triggered by the *Pegasus* project and the results have found their way into the project work-stream for a common, cross-vendor simulation tool chain.

Models of the sensory perception can be built in a variety of ways for different applications. These applications and the type of models required for them can most easily be understood when considering the framework of the well known V-model for system development that can also be applied to the development of automated driving systems. The initial left side of the *V* describes the specification on the system, sub-system, and component levels in this order of increasing detail. Following the implementation, the right side of the *V* describes the testing and validation efforts on component, sub-system, and system level that are usually performed in this order of increasing complexity.

For the description of the processing chain for the simulation of automated driving systems, the same three levels of abstraction can be used: system, sub-system, and component level. Shown as an overview in Fig. 2.2, the input for an automated driving system is always given by a representation of the virtual world and the output consists of the system's and therefore the car's behavior. Naturally, on a more detailed level a more detailed output can be given and a more detailed world representation on the input side is required. While for an abstract simulation on the system level, an abstract representation of the static environment and dynamic actors in the virtual world is sufficient, a detailed component level simulation requires a detailed representation of the three-dimensional geometry of all elements in the environment. The relation between these three abstraction levels and the scale of the simulation of the automated driving system's processing chain is introduced in the following.

On the *system level*, the scale of the simulation is the scale of the whole car. In

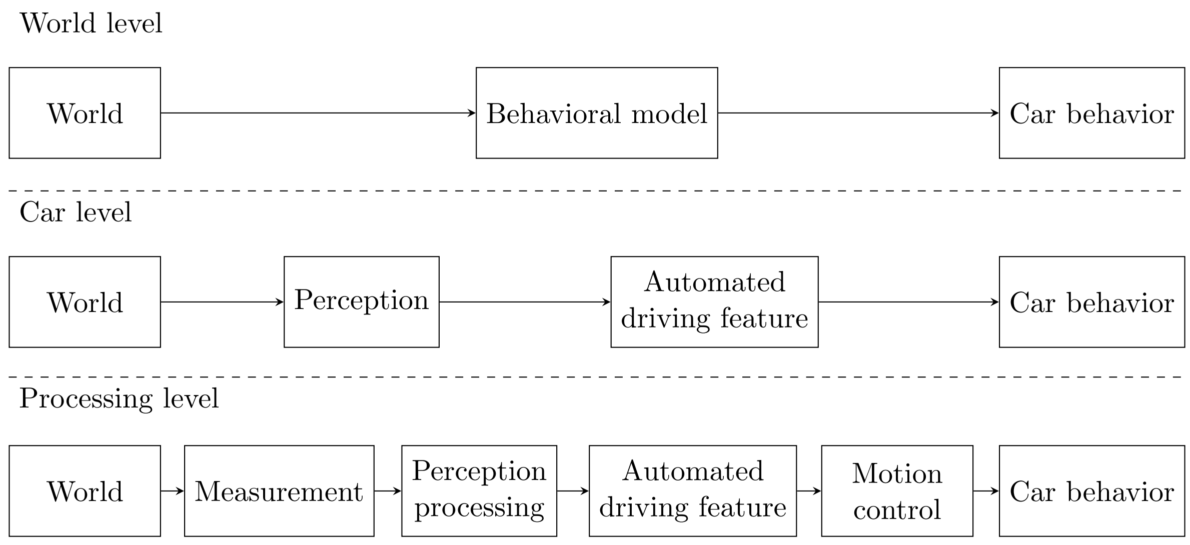


Figure 2.2: Overview of the different levels of abstraction when considering the processing chain in automated driving system. With regards to a simulation environment, it has to be determined whether the simulation is to be performed on the world, car, or processing level, which correspond to the system, sub-system, and component levels of the V model in system development. The type of sensor model handling the perception process is different depending on the level of abstraction chosen for the simulation.

this context, the focus is on the general behavior of the car and its interaction with its surroundings. As shown in the top row of Fig. 2.2, a system level simulation requires no more than an abstract model of the behavior of the car, where the car may be steered by means of an automated driving system. Models for the perception process on this level will be concerned with basic perception properties, e.g. the field of view and occlusion, if they are used at all.
On the *sub-system level*, the goal is to investigate the behavior of the individual subsystems within in the car as well as their interaction. In the context of automated driving systems, sensor models play a role in this type of simulation when investigating the decision making of the automated driving function in its interaction with the environment. Perception is important for modeling the object and event detection and response part of an automated driving system (see Fig. 1.1 in the introduction).
On the most detailed level, the *component level*, the simulation deals separately with each component in the processing chains within an automated driving system. The goal usually is either to closely investigate the interaction of different components or zoom into the details of one single component and study its behavior, especially when changing some of its parameters. At this level of simulation, it is important that a sensor model manages to very closely reproduce real world sensor output so that a subsequent step in the processing chain may be stimulated with adequate input values. Depending on the context or when investigating the sensing process itself, it may be required to build a model that includes the physical details of the measurement process.

Depending on whether a test is supposed to run on sub-system or component level, different types of sensor models are required, tailored to the specific requirements.

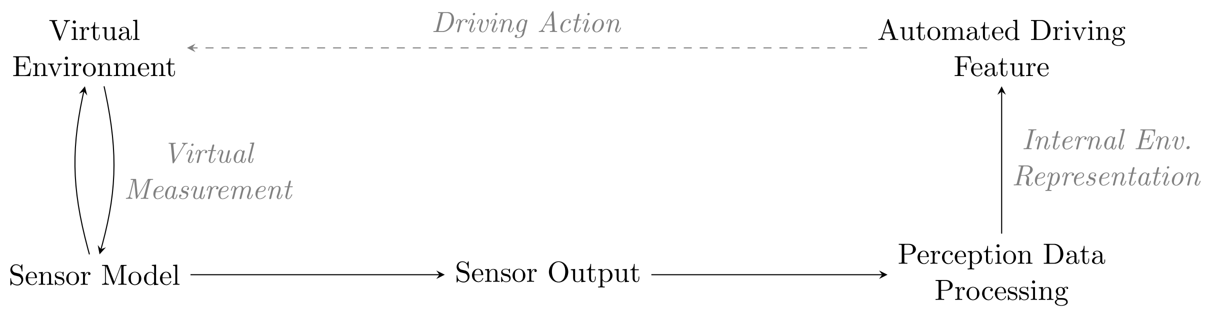

Virtual Environment
Driving Action
Automated Driving Feature
Virtual Measurement
Internal Env. Representation
Sensor Model
Sensor Output
Perception Data Processing

Figure 2.3: Data flow in the processing chain for the simulation of automated driving systems. Starting from the virtual environment, a sensor model performs virtual measurements, resulting in sensor output data. For the purpose of re-usability of sensor models for sensors of different type, vendor, or generation, a generic interface is used for the model's output. In the next step, this is transformed to the specific interface defined by the sensor under investigation and passed on to the perception data processing of the automated driving system in the same format as the sensor data in a real world setup. The resulting internal environment representation is used by the algorithms of the automated driving system to determine the driving actions undertaken by the vehicle. Forwarding these to driving simulator for the update of the virtual environment results in a closed loop simulation.

Even though their focus is mostly testing, sensor models may be used on either side of the V model, providing valuable insight for the specification early on as no hardware samples for sensors are required. Sensor models may in general be divided in two categories: statistical and physical models. The level of detail that a physical model provides is especially useful on the component level and allows detailed investigations, e.g. about the specific placement and field of view of a sensor, that are not possible with a statistical model. On the other hand, the superior performance in execution speed of statistical models makes them a perfect candidate for the use on the system and sub-system levels, where exploring a large number of scenarios to determine the system's behavior resulting from a wide variation of input signals is more important than simulating the exact behavior of a component under very specific conditions down to the last detail.

The defining task of a sensor model within the processing chain for virtual development and testing is the generation of sensor output data that captures the real world process of environmental perception within the virtual world. As these perception data are key for many of the computations of the automated driving feature, the fidelity by which the sensor model is able to reproduce the real world sensor behavior is a deciding factor for the viability and validity of virtual development and testing. Therefore, the main goal of sensor modeling is providing synthetic sensor data that is indistinguishable from real world sensor data. This task has been a long running quest [43]. Furthermore, depending on which part of the processing chain is under investigation and on the goals that are to be achieved with the virtual world testing, the requirements on the desired virtual sensor output data vary by a large margin. For each case, a specific sensor model has to be designed and implemented to fit these requirements.

Deriving from the view of the processing chains for different levels of abstraction shown in Fig. 2.2, the general processing chain for a simulation making use of sensor models involves the steps shown in Fig. 2.3. With the aim of employing the virtual environment and sensor model in the development of an automated driving system, the goal is the setup of a closed loop simulation. Closed loop refers to environment data from the virtual environment being passed as input to the automated driving system and the results of the system's processing, i.e. the actions taken by the car at the current simulation update step, being fed back to the driving simulation framework for calculating the next update step of the virtual environment. This loop involves the following steps: Starting from the virtual environment, a sensor model performs a virtual measurement resulting in synthetically generated sensor output data. At the next step, this information is transmitted to the software framework handling processing for the automated driving system in a data representation form that is structurally identical to the data format used by the sensor under investigation. At this point, the sensor model's output data becomes, at least in the way it is stored and used, indistinguishable from a real world setup. Using the input data in the way as it would use output data from a real sensor, the perception data processing of the automated driving system, e.g. sensor fusion and data interpretation algorithms, construct an internal environment representation that is used by the subsequent processing steps of the automated driving feature, e.g. a trajectory planner. Finally, closing the simulation loop shown in Fig. 2.3, this environment information is used by the automated driving feature to calculate the driving action of the vehicle that is reported back to the driving simulation framework for the calculation of the next simulation update step.

One aspect of the work presented in this thesis is an effort to allow multiple different implementations of the general processing chain for simulation of automated driving systems shown in Fig. 2.3 to reuse components, specifically sensor model implementations. For the virtual environment, a variety of driving simulation frameworks from different vendors exist that each have different strong points in line with their main application focus. Similarly, for the algorithms of the perception data processing and the automated driving feature, multiple frameworks exist and are in active use. For the goal of enabling these components to readily interact with each other, standardized interfaces on the input and output of the sensor model can be utilized. Using standardized interfaces for sensor models further allows to readily use different types of sensor models, e.g. vendor specific models for different types of sensors, within a compact setup of the closed loop simulation processing chain. In this work, a standardized interface for this purpose is introduced. This is the Open Simulation Interface (OSI), of which a detailed discussion is the topic of section 2.5. Further, with the help of a standardized interface, the internal layout of sensor models can be separated into a sequence of modular components that helps to define a procedure for constructing sensor models. This is discussed in section 2.4, which introduced the systematic approach to sensor models proposed in this thesis.

In the context of the different views on the processing chain for automated driving systems introduced in Fig. 2.2, it has become clear that different types of sensor models are needed to properly suit different types of application. These model types range from general statistical models as an abstract description of the perception process on the abstract level to detailed physical models of the measurement process when an in-depth modeling of the steps in the perception chain is aimed for. In the next section, the current, typically integrated, approach to building sensor models is shown.

2.2 Sensor Simulation – General Concepts

In order to discuss the variants of sensor models, one has to start by looking at the series of steps performed during the sensing and perception process. Depending on how this series of steps is implemented in a simulation model, different types of models for different applications emerge. In a general view of the perception process, a sensor maps sensor targets in the environment (e.g. road users or static obstacles) onto a digital representation that can then be used for further processing ranging from simple visualization to the sophisticated processing of an automated driving function. The digital representation is usually a list of objects, although alternative representations like occupancy grids exist [44, 23]. The sensing process consists of two subsequent steps: measurement and processing as shown in Fig. 2.4. For a real world sensor, the measurement is the detection of sensor targets in the environment by means of an electro-magnetic (radar, lidar, camera) or acoustic (ultrasound) interaction. The measurement is performed by either detecting an incoming signal for passive sensors, e.g. dectecting light in a camera, or transmitting a probing signal and detecting the responses from the environment for active sensors, e.g. a lidar sending out a laser pulse and detecting the reflected light. The detection is performed by the analog front-end, converted to a digital signal by an analog-digital (A/D) converter, then subjected to low-level processing algorithms that clean the received signal and turn it into a usable low level representation. This low level representation is sensor raw data that may take many forms that are largely dependent on the type of sensor as well as the vendor. For a lidar sensor this is typically a lidar point cloud of reflection points, for a radar sensor this representation can take the form of a list of radar detections obtained by analyzing the Fourier transform of the reflected wave. From this raw data representation finally a signal processing unit extracts objects and object properties, often by also making use of tracking algorithms that take into account multiple subsequent measurements. This processing chain is shown in Fig. 2.4 (a).

In a virtual environment the input consists of virtual objects provided and updated by a driving simulation framework. In the mapping between these virtual objects and the output list of detected objects, three general classes of model applications can be distinguished. The differentiating factor between these is the treatment of their respective treatment of the measurement and processing steps of the sensing process. The most straightforward type is the direct mapping from the virtual objects of the simulation environment to the output object list. This direct mapping encompasses both the measurement and processing steps as shown in Fig. 2.4 (b) and is achieved by means of an object-based sensor model. This type of model is typically a stochastic model as used for example in references [26, 45, 27, 28, 30, 31]. It can also incorporate some input from the physical quantities involved as discussed e.g. in [46] or the model can be based on some type of machine learning approach, either based on data from previously recorded data sets[47, 48] or on measurements from a previous point in time as used in [49] for a robotic model. Treating the two steps, measurement and processing, separately within a simulation model comes at the cost of increased complexity and requires a raw data interface within the model as shown in Fig. 2.4 (c). This type of model actually consists of two separate parts: a measurement model for the measurement step that is generating sensor raw data and a signal processing model for the processing step that turns the sensor raw data into the final output object list. With the help of a measurement model generating sensor raw data that uses the identical format as a corresponding real world sensor, it is also possible to directly feed the generated raw data into the signal processing algorithms within the electronic control unit (ECU) of the sensor. This is shown in Fig. 2.4 (d) and allows to build a hardware-in-the-loop setup from sensor data generated in a virtual environment. A practical limitation of this approach, however, is the requirement for an input interface on the sensor ECU that allows feeding sensor raw data from an external source, which is not necessarily provided on typical automotive sensor hardware.

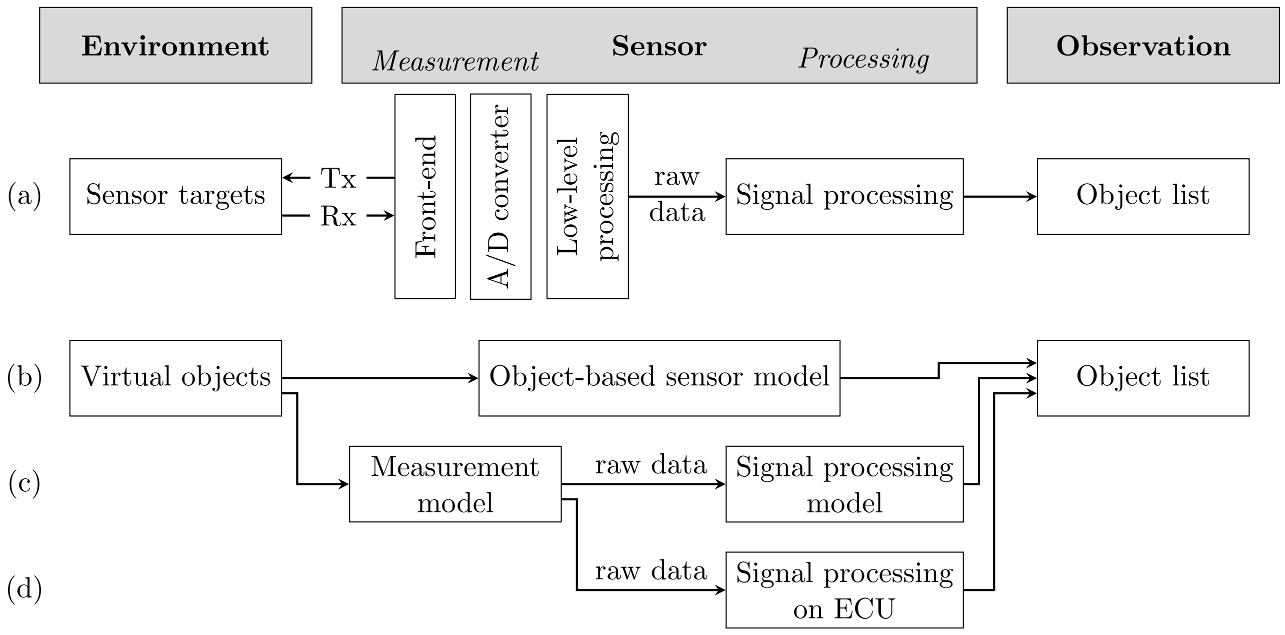


Figure 2.4: Overview of the sensing process: (a) Detection and processing chain of a real sensor. (b) Direct mapping of virtual objects in the simulated environment onto an output object list. (c) Separate models for the measurement and processing steps connected by a raw data interface. (d) Sensor raw data generated by a measurement model used as stimulation for the signal processing on the sensor ECU in a hardware-in-the-loop setup.

With the goal of achieving results as close to reality as possible, the signal processing based on sensor raw data generated from a measurement model should typically make use the actual algorithms also used in the corresponding real world sensor. The task of the sensor model in this case is then the generation of the low level sensor data. With this focus on the measurement model, with regards to the sensor model implementation the processing chains for cases Fig. 2.4 (c) and (d) with a signal processing model in software or on the target hardware can be viewed as having identical requirements. On the large, therefore, two types of sensor models can be differentiated that differ significantly in their approach to modeling sensor behavior: On the one hand, there are object-based sensor models that aim to generate equivalent sensor output to a real sensor based on higher level interfaces after the measurement and processing steps. These types model typically try to emulate the properties of the sensing process on a statistical level. Examples can be found in [28, 29, 30, 31]. On the other hand, measurement models aims to generate low level sensor output by modeling the measurement step in the sensing process. Typically, these types of models employ a physical description of the measurement process in order to achieve realistic output of the resulting low level data that can then be input back into the sensors processing chain. Physical measurement models are commonly used for a variety of sensors in robotics research [34, 35, 50], where they are also used inversely to learn the layout of the three-dimensional static environment from noisy sensor measurement data [51, 52]. In the automotive context, first measurement models for automotive radar sensors are only just emerging [32, 33]. As a special case, measurement models of camera sensors are in their most fundamental form identical to the image rendering performed by any simulation framework. For this reason, there are efforts to utilize synthetic images for the training of image recognition algorithms [53, 54].

Given this disparity in model types, a clear definition is important when using the term *sensor model*, as it may apply to a model of the measurement process or may include the perception function interpreting the measurement data. A dedicated measurement model has its specific advantages inherent to a physical or otherwise low level approach regarding realism and free configuration of the sensor position. However, capturing the complexity of the measurement process to a sufficient degree with low enough computational requirements to allow for real-time processing as well as the availability of the perception function software to interpret the generated raw data are significant challenges. An integrated object-based, typically stochastic model of the sensory perception encompasses the whole process of mapping environmental data to the output interface specific to the sensor of interest. For this type of model, computational complexity is usually much lower. Furthermore, the challenge of realism is partially shifted to the acquisition of appropriate reference data used as input of the model. Inherent to the approach, equivalence to the real sensor behavior is established on a statistical level.

Following this discussion, it is clear that one should abandon the notion of a single equivalent sensor model of a virtual sensor in the virtual world representing the real sensor mounted on a real car with regard to every imaginable application. Instead, the specific application establishes the design goals and determines the specifics of the simulation model most appropriate for the task at hand. The design application for a specific model is mainly determined by the unit under test in the target setup and therefore closely linked to the different levels in the system description introduced in Fig. 2.2 in the discussion of perception models within the development process. Generally, on higher levels of abstraction in the system description, e.g. when the unit under test is the system as a whole, more generic and fast object-based sensor models are used. As the level of detail increases, down to the component level, also the level of detail in the sensor simulation needs to increase. Detailed measurement models generating low level sensor data can be construction to fit this type of application. It is also possible to use very detailed physical sensor models for the sensor development, e.g. in figuring out the positioning of sensors within the vehicle or the required sensor parameters like range or field or view.

This work aims to provide a structured approach to the construction of sensor models for all the different types of applications. For this reason, starting from these general concepts for the sensing process and sensor simulation, a formal description of the sensing process and therefore sensor models is needed. This is the focus of the next section.

2.3 Formal Description of Sensor Models

The process of measurement and perception of the environment by means of sensors can be broken down into several sequential steps. Fig. 2.5 shows the flow of data: Objects present in the environment (a) are represented after the measurement process (b) by measurement raw data (c). This sensory input is interpreted by a perception function (d) handling object detection and tracking. The results are object data that in turn are encoded and packaged in a specific sensor output interface format (e). In this form, the information is transmitted on vehicle bus level by the sensor for further processing. It may then be used directly by the functional component of a driver assistance system. Alternatively, collective data from multiple sensors may be used to construct an internal, comprehensive representation of the environment required for sophisticated automated driving systems.

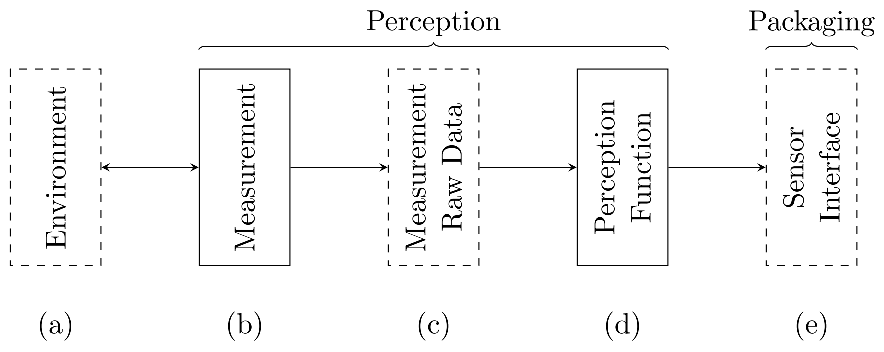


Figure 2.5: Processing chain, starting with the environment (a) that is mapped in a measurement step (b) onto measurement raw data (c). The raw data are processed by a perception function (d) and the resulting data encoded in the sensor output interface (e) during the packaging stage.

In the context of the perception chain shown in Fig. 2.5, the different types of sensor models operate on different parts of the chain. Object based sensor models cover the whole perception chain including the measurement process and perception function. In contrast, measurement models as indicated by their name are specific to the measurement step of the perception chain.

For the simulation of sensors, it is required that the simulation environment handling the virtual world provides all the necessary input data to the sensor model. The details of the information required by a specific sensor model depend on the complexity of the model and may include besides information about all the static and dynamic objects in the virtual world also information about weather, the day-night cycle, road conditions, and small objects that could cause stray reflections but are not explicitly model within the simulation. These environment conditions constitute a further input for the sensor model and need to be provided by the simulation framework. In addition to all this information about the virtual environment, the internal parameters of the sensor model have to determined and defined for a simulation run.

In the following the sensing process is put into mathematical terms. The output as the result of the perception process including all the objects with their properties as detected by the specific sensor define the perception data, denoted by the vector $\widetilde{\Phi}$. These data are the result of sensory perception of the physical environment, denoted by the state vector Φ for a real world environment. In short, sensory perception can therefore be seen as the mapping

$$S : \Phi \to \widetilde{\Phi}.$$

This relation is illustrated in Fig. 2.6 (a). In a simulation environment, the state vector of the virtual world is denoted as Ψ. As illustrated in Fig. 2.6 (b), the action of a sensor model is then a mapping of the state vector Ψ to the simulated perception data $\widetilde{\Psi}$, i.e. the model is defined by

$$M : \Psi \to \widetilde{\Psi}.$$

While the exact state of the simulation environment is known at any time and available as ground truth data providing a virtual reference $\overline{\Psi}$, the state vector Φ of the real world environment is never known exactly and always has to be approximated by measurement. For this reason, a reference measurement $\overline{\Phi}$ is introduced that also constitutes a sensor measurement typically using specialized very high fidelity sensors in addition to analysis in post-processing to generate the best known approximation of the world state Φ. This reference measurement is then defined as

$$S_{\text{ref}} : \Phi \to \overline{\Phi},$$

also illustrated in Fig. 2.6 (a). This raises an important issue: As a glance at the flow of information in Fig. 2.6 reveals, the sensor models operate in the same way as the sensor in the real world by mapping the current world state to perception data. The quality of a sensor model is then defined by its ability to match the behavior of its real world counterpart as faithfully as possible. However, as the true state Φ in the real world is never known, gauging the quality of a sensor model and performing a model validation is not possible in a straight forward manner. Instead, it always depends on additional

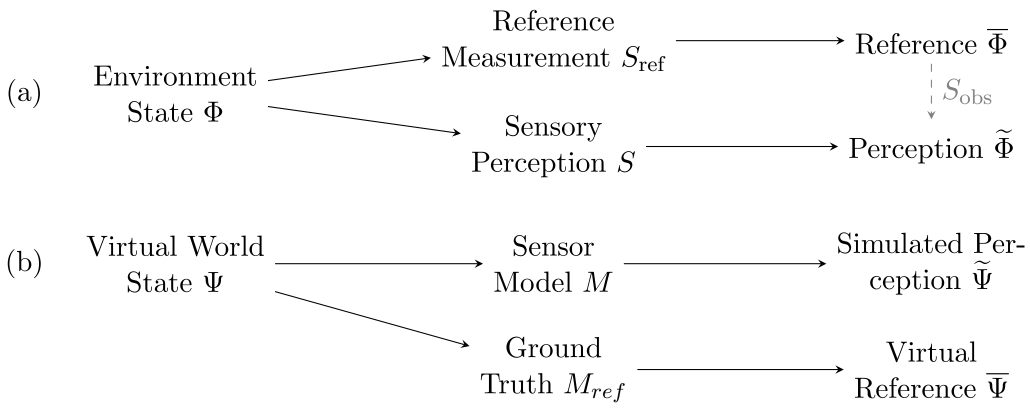

(a)
Environment State Φ
Reference Measurement S_{ref}
Sensory Perception S
Reference $\overline{\Phi}$
S_{obs}
Perception $\widetilde{\Phi}$
(b)
Virtual World State Ψ
Sensor Model M
Ground Truth M_{ref}
Simulated Perception $\widetilde{\Psi}$
Virtual Reference $\overline{\Psi}$

Figure 2.6: Overview of relations between data: (a) In real world scenarios, the sensory perception process maps the environmental state Φ of the physical world on perception data $\widetilde{\Phi}$. Similarly, reference data $\overline{\Phi}$ are obtained in a reference measurement. (b) In a sensor simulation, the current state Ψ of the virtual world is mapped onto simulated perception data $\widetilde{\Psi}$ by means of a sensor model. Virtual world reference data $\overline{\Psi}$ are easily obtained from the simulation ground truth. The aim of any sensor model is achieving as close to equivalence as possible of simulated perception data $\widetilde{\Psi}$ to the real sensors perception $\widetilde{\Phi}$ given identical or largely similar world states Φ and Ψ. As the exact true state Φ of the real world is never known, it has to be approximated by a reference measurement $\overline{\Phi}$.

reference measurements $\overline{\Phi}$ to provide a close enough approximation of the state of the environment. Moreover, the underlying task of model construction as finding a virtual correspondence to the sensor mapping $S : \Phi \to \widetilde{\Phi}$ is ill-formed as no information about this mapping is available. Instead, model construction and parametrization will always be based on the observable sensor mapping of reference to perception data of the sensor under investigation, i.e. $S_{\text{obs}} : \overline{\Phi} \to \widetilde{\Phi}$, which is illustrated by the dashed gray arrow in Fig. 2.6 (a).

For the subset of object data, the model mapping $M : \Psi \to \widetilde{\Psi}$ can be written more explicit. We define the set $X \in \Psi$ of ground truth objects, where each element $\mathbf{x_i} \in X$ is a vector containing all the object properties. In same way, we define the set $Z \in \widetilde{\Psi}$ as the set of output objects, where each element $\mathbf{z_i} \in Z$ is also a vector containing the object properties of a detected object. Note that the properties for the elements in the sensor output Z differ from the properties of elements in the ground truth X. The sensing process for objects is then defined as the mapping M that maps the set of sensor targets $X = \{\mathbf{x}_i\}_{i\in[1,N_X]}$ onto the set of observed targets $Z = \{\mathbf{z}_j\}_{j\in[1,N_Z]}$, depending on sensor parameters $\mathbf{p}$ and environmental conditions $\mathbf{c}$:

$$M(\mathbf{p}, \mathbf{c}) : \{\mathbf{x}_i\}_{i\in[1,N_X]} \to \{\mathbf{z}_j\}_{j\in[1,N_Z]} \,. \tag{2.1}$$

The indices i and j enumerate the objects in the respective set. Note that the number of elements $N_X = |X|$ and $N_Z = |Z|$ in the sets X and Z need not match and mapping need not be on a per-element basis, therefore in general $\mathbf{x}_i \not\to \mathbf{z}_i$ and $|X| \neq |Z|$. This reflects the fact that not all targets in the environment are detected at all times, i.e. false negatives are present in addition to a limited range and field of view of the sensor, and that also additional ghost targets, i.e. false positives, may be present in the set of observed targets.

The general processing chain for sensor simulation, initially shown in Fig. 2.3, starts from a virtual environment and then goes through the virtual measurement step by means of a sensor model generating output data, which is transformed to the sensor specific interface format inherited from the corresponding real sensor and given as input to the perception data processing of the automated driving system. Based on this definition, it becomes apparent that the task of creating virtual sensors mainly focuses on the setup of the specific processing chain for the design application, the construction of a sensor model of the appropriate type, and the definition of generic input and output interfaces used by the sensor model. A systematic approach towards this task of sensor model construction will be introduced in the following.

2.4 Virtual Sensorics – A Systematic Approach to Sensor Models

Generally, two types of sensor models can be distinguished: *Sensor error models* aim to reproduce the statistical characteristics of errors, i.e. deviations between the perceived and true values, of the measurement and perception performed by vehicle sensors [28, 29, 30, 31]. *Sensor measurement models* are based on a physical description of the measurement process and generate low level measurement data based on the virtual scene. Models of this type are commonly used for a variety of sensors in robotics research [34, 35], while first measurement models for automotive radar sensors are only just emerging [32, 33, 48].

As illustrated in Fig. 2.7, operating on high level object data is the domain of sensor error models. For these statistical models, the design application is centered around the generation of input data for an automated driving system to evaluate its behavior in a large number of driving scenarios. For this purpose, the entire simulation tool chain operates with data on the level of objects. In this case, objects denote entities in the environment that are of significance to the functional behavior and detected by the car's sensors. These include dynamic objects such as other road users, static obstacles that limit the available driving space, traffic signs and lights, and road markings. This type of sensor model aims to simulate the object output of a sensor, i.e. the result of the measurement process already interpreted by sensor internal perception functions. Owing to its design use case, a sensor error model's main objective is computational speed to allow efficient running of a large number of scenarios.

In the case of sensor measurement models, which are physically motivated models of the measurement process, the design application is the generation of low level sensor data that can either be used to understand the sensors behavior in the driving context or used to generate input data for low level perception processing functionality. This is illustrated in the lower part of Fig. 2.7. The output of this low level perception processing can be again used as input for higher level perception processing in the same manner as the object level output of a sensor error model. In contrast, the tool

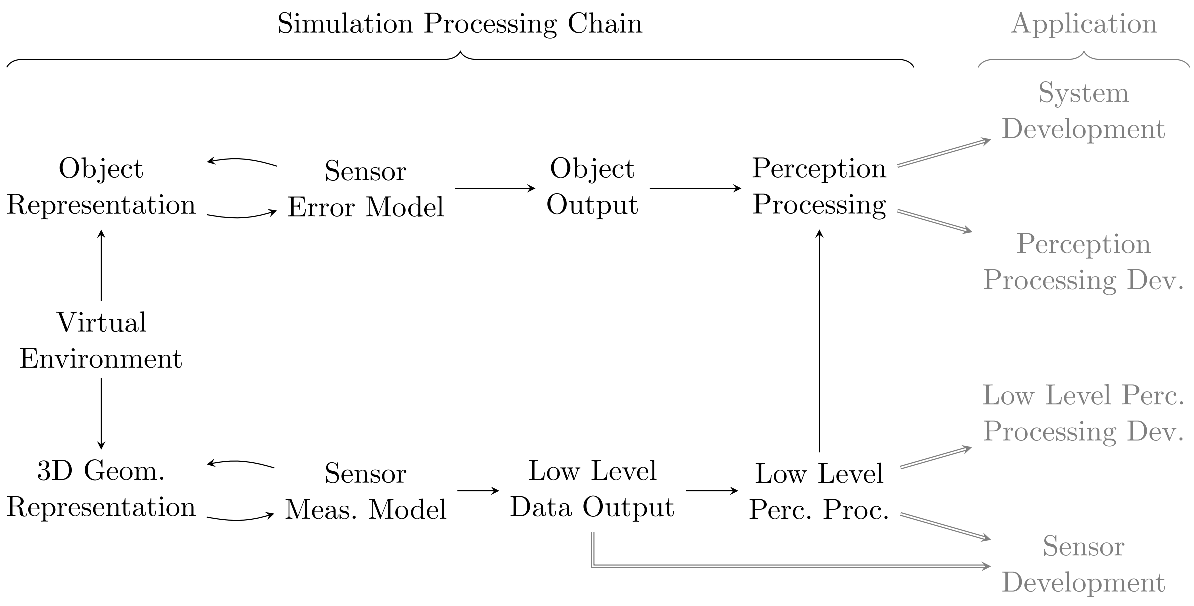


Figure 2.7: Overview of different types of models and simulation tool chains, allowing different types of applications. For the development of higher level functionality within an automated driving system, where the unit under test is either the system as a whole in a closed loop environment or the high level perception processing in an open loop environment, a sensor error model can give the required input and provide a high level of computational performance. For development of lower level functionality like the low level perception processing or investigations into the sensor setup, a sensor measurement model is needed that is able to generate the required low level output data. The more complex tool chain including the low level processing can be linked to the higher level processing, which allows applications on every abstractions level, although the computational requirements prohibit many large scale applications.

chain including a sensor measurement model is purpose-built to achieve a very high degree of realism for a specific scenario. Its high computational requirements usually make it prohibitive to be used on large scale simulation testing in the same way as a sensor error model. The ability of sensor measurement models to generate very realistic low level sensor output data opens up the ability to use it directly in the development of a sensor or a setup of sensors mounted on a vehicle. As this type of model is based around a physically motivated approach, no statistical input data obtained from a specific sensor setup is required to define the model's parameters. This allows among others investigations around the placement of a sensor on the vehicle as well as tuning of the sensor's specification parameter, e.g. its field of view, and estimating the impact of adding sensors to or removing them from a vehicle's sensor setup without the need for time-consuming and costly real world vehicle preparations. This opens up a significant tool box for sensor development in the context of automated driving systems.

Aside from the design application of a sensor model, there is an additional practical limitation in the construction of a sensor model describing the behavior of a specific sensor. As the simulation tool chain including the sensor model is always based on the real world tool chain, the output format of the original sensor has to be exactly reproduced at some point in the simulation tool chain. This point in the tool chain is the first point at which a comparison of the model's output data with real world reference data can be performed. Therefore, if a specific sensor only has an output interface on a higher abstraction layer, e.g. object descriptions, it is impossible to validate the output data of intermediate model steps if a more complex model type has been chosen. In many research applications, e.g. robotics simulations [34, 35], sensor models are typically used to simulate the sensor measurement process as the sensors used in this context usually offer the required low level data interfaces. However, in an automotive context sensors are often a package of the measurement hardware and sophisticated perception functions that provide functionality such as object detection and tracking. The sensor model is therefore limited to the use of the corresponding high level interfaces.

In many cases, even if the sensor additionally offers a low level data interface, several component of the automated driving system make use of the high level output of the sensors internal perception processing. As the specifics of the sensor internal perception processing is usually propriety intellectual property of the sensor manufacturer, they are typically not available for the development of a simulation-based processing chain. In addition to their computational complexity, this limits the scope of applications for sensor measurements models. Sensor error models, which are statistical models of the perception process, avoid this problem by their nature they include the complete perception step including measurement and the processing by sensor internal perception functions. Examples of statistical models can be found in references [26, 27, 28, 55, 31, 56, 30, 29]. Statistical models however turn the main challenge in model construction towards obtaining suitable reference data that is used to determine the model parameters and behavior. This holds regardless of whether the model is using a parametric [26, 27, 28, 55, 31] or nonparametric statistical approach [56, 30].

2.5 Open Simulation Interface

One of the most critical aspects in the development of sensor models is the usage of interfaces between components in the virtual world setup. As individual components should be as independent of each other as possible to allow reuse in a variety of different setups, e.g. using driving simulation frameworks that each have different strong points and are therefore used for different purposes, it is important to decouple the sensor model from the specifics of any one simulation framework. For this reason, a proposal for a standardized interface, the Open Simulation Interface (OSI), is introduced that has been extensively developed within the scope of this work.

2.5.1 Design Objectives of the Open Simulation Interface

The open simulation interface (OSI) is designed an abstraction layer for the purpose of developing sensor models independently of the specific implementation of the simulation

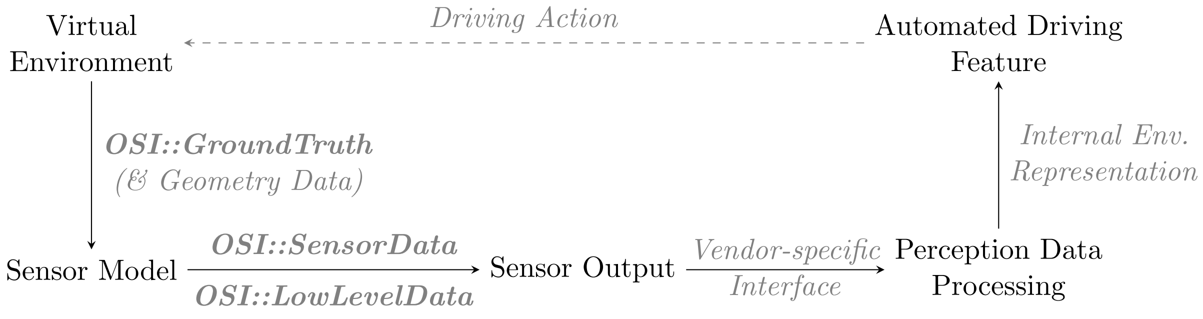


Figure 2.8: Data flow in the processing chain for the simulation of automated driving systems using the interfaces definitions of the Open Simulation Interface (OSI), highlighted in bold. The figure is based on the general data flow shown in Fig. 2.3. The *OSI::GroundTruth* interface is used as output of the virtual environment and standardized input for the sensor model. In case of physical sensor models, it may be complemented by 3D geometry data. The output of the sensor model is either in the form of the *OSI::SensorData* or *OSI:LowLevelData* interfaces, depending on the type of model. If correspondence with a specific hardware setup is required, these data need to be transformed into a vendor-specific sensor interface to serve as structurally identical input for the perception data processing of the automated driving system.

environment and the automated driving system's processing chain. OSI encapsulated the link between the ground truth given by the virtual environment on the input side of virtual perception and the resulting sensor data in the form used for further processing on the output side. Given the necessary adapters, a sensor model developed with OSI can therefore be deployed in a multitude of environments.

Within the processing chain for the simulation of automated driving systems, initially introduced in Fig. 2.3, sensor models constitute the link between the input and output sides of the virtual perception step. The goal of OSI is to provide data structures for the input and output of sensor models as well as their inner workings. The updated processing chain using the OSI interfaces is shown in Fig. 2.8. Therefore OSI defines a generic object-level interface between the virtual environment and the sensor model and sensor data output interface on the object-level as well as the level of low level sensor data. Respectively, these correspond to the *object representation*, *object output*, and low level data output steps shown in the illustration of the simulation tool chains in Fig. 2.7. A three-dimensional geometric representation of the virtual environment as required by a sensor measurement model is not part of OSI, as such a representation has very high requirements to computational efficiency and does not lend itself to the generic approach behind a standardized exchange interface like OSI. The first publication of the open simulation interface (OSI) by the author and collaborators can be found in reference [57]. In the time since this first publication, OSI has been publicized as an active open source project maintained by a consortium consisting of members from several automotive companies and research institutions. The most recent version of OSI can be found in the open source project's repository in reference [58].

2.5.2 Semantic Structures of the Open Simulation Interface

Within the Open Simulation Interface, there are two different types of data structures defined: osi::GroundTruth and osi::SensorData define high level object-type data and osi::LowLevelData defines low level sensor data. For the purpose of OSI, objects are defined as any non-transient entity that has a collection of physical and non-physical attributes and can be easily referenced by a static identifier. This includes vehicles, pedestrians, static road infrastructure, lane segments and so on. The sum of all objects at a specific point in time makes up a static snapshot of the environment at that time. Low level data on the other hand is a description of the usually transient physical values that represent detailed sensor measurements. For the example of a laserscanner, these include the detected reflections making up the three dimensional point cloud with their measurement parameters like signal strength or derived physical quantities such as the reflection coefficient of the reflecting surface.

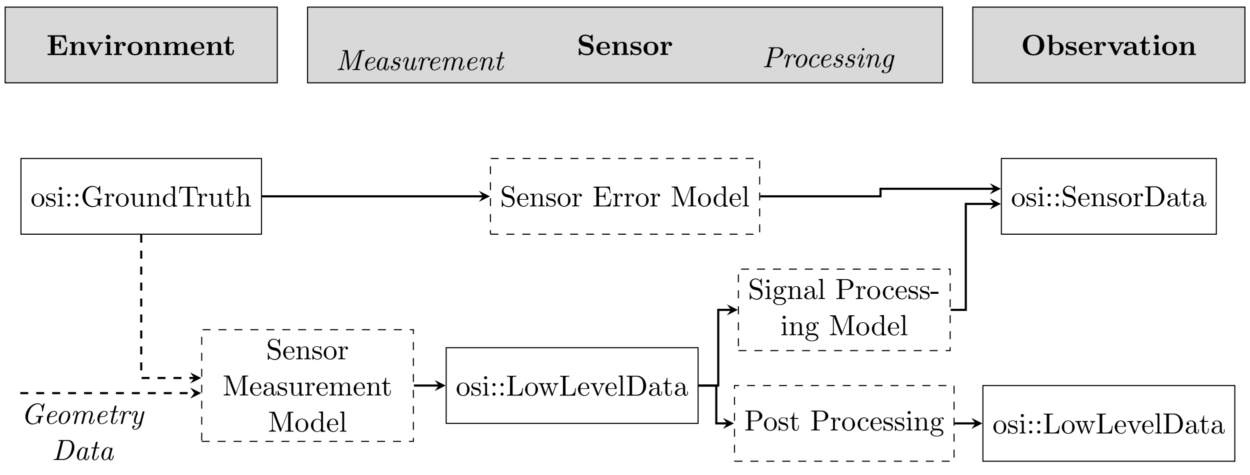


Figure 2.9: Processing chains for the Open Simulation Interface (OSI) illustrating the relation between the different message types of the interface. Transitions between the different interfaces are performed by different types of sensor models.

The interplay between the different OSI interfaces and their location within the sensor model tool chain is illustrated in Fig. 2.9. For an object based sensor model like a sensor error model osi::GroundTruth defines the input from the virtual environment and osi::SensorData defines the output of the objects detected by the virtual sensor. For a physically motivated sensor measurement model the osi::LowLevelData interface defines the format of the output sensor data. On the input side, it is possible to use the object-level description in osi::GroundTruth, however usually a representation of the three-dimensional environment that is specific to the simulation framework will be used for performance reasons. As physical calculations for the generation of low level sensor output usually require highly optimized graphics processing with their own optimized environment representation, these kind of optimized interfaces are considered out of scope for OSI, which instead focuses standardizing the output for all types of sensor models.

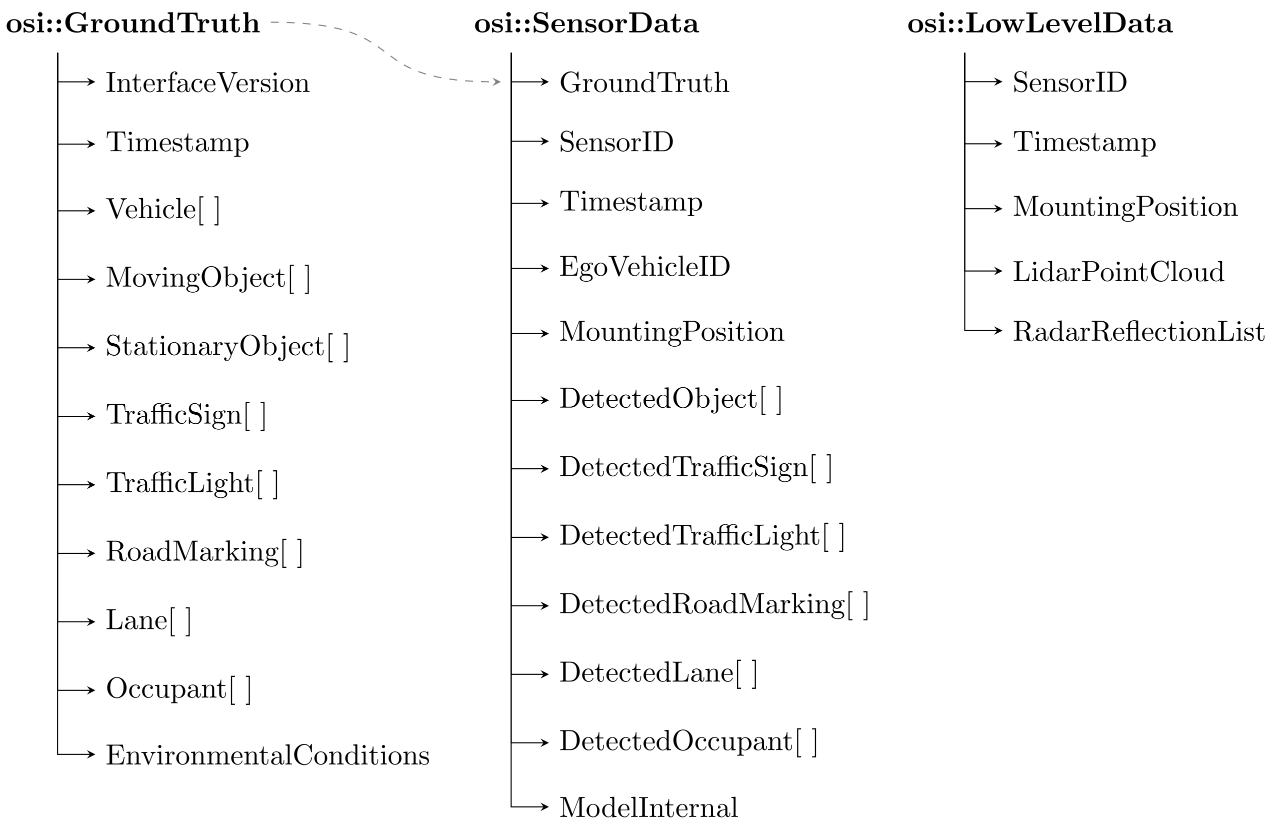


Figure 2.10: Overview of the information contained in the main interface of the Open Simulation Interface (OSI). The interface osi::GroundTruth is designed as object level output data of a simulation environment that describes the current scene and is used by a sensor model to derive sensor output data. The interface osi::SensorData is designed to be used internally by a sensor model and to provide a generic output interface for synthetic sensor data generated by a sensor model. It also contains a copy of the original ground truth data used for internal processing by a sensor model as well as to allow a direct comparison of the model's output to the original data. The interface osi::LowLevelData is designed for low level sensor output data as the result of a sensor measurement model.

The top level elements for each of the three OSI interfaces are shown in Fig. 2.10. In the object-type interfaces osi::GroundTruth and osi::SensorData, the object data are stored in arrays for the different semantic types of objects. These arrays are marked by brackets [ ] in Fig. 2.10. Each element of an array corresponds to one object. For each object, a set of object properties is stored that differs for every semantic type resulting in the different arrays in the interface. Among these properties are for example a unique identifier, the position, bounding box, and the orientation for each object. The osi::GroundTruth interface stores all object properties exactly as they appear at one specified update step in the virtual environment. In order to decrease the interface size for very large virtual environments, only a certain distance around each ego vehicle has to be included in the ground truth interface. An ego vehicle is defines as any vehicle that may hold a sensor that is represented by a sensor model that uses the ground truth data for simulation of the sensor output. The specific distance is defined by the implementation of the respective simulation framework. In addition to the arrays containing the objects, the interface includes its own version as meta data, and the current simulation timestamp as well as a set of environmental conditions that may be relevant for perception by sensors, e.g. the ambient lighting conditions, which may be relevant for a sensor model representing a camera sensor dependent on ambient light.

The osi::SensorData interface as shown in the central column of Fig. 2.10 contains a arrays for the storing of objects similar to the ground truth interface. As the osi::SensorData interface represents the output data of a sensor model, it contains a semantic separation of objects are well as object properties for each of the objects that matches the data that can be perceived by a sensor. As an example, while the ground truth interface differentiates between vehicles, (other) moving objects, and stationary objects as each has a different set of properties with only a shared subset, in the SensorData output interface these are merged to one single type: DetectedObject. The reason for this is that an object detection algorithm can only estimate a probability for a detected object to be of a certain semantic type, which has to be reflected in the interface. Additionally, some properties like an existence probability or a sensor internal tracking identifier are added to the data types in the osi::SensorData interface that do not have a direct logical correspondence in the ground truth description. In addition to the arrays containing the objects, the osi::SensorData interface includes information about the sensor that is not present in the ground truth interface but instead is part of the sensor's and therefore also sensor model's parameter. These are the identifier of the sensor within the vehicle, the identifier of the ego vehicle itself, and the mounting position of the sensor on the vehicle relative to the vehicle's own coordinate system. Furthermore, the interface contains the time stamp of the sensor data, which can differ from the ground truth time stamp if the sensor model takes the sensor's measurement time into account. A copy of the original ground truth data allows for an easy matching of sensor model output and original data. This is mainly used for validation purposes and for internal processing by the sensor model. For internal processing by the sensor model, there is an additional ModelInternal object that is used by sensor models using the osi::SensorData interface as their internal data structure to store temporary values. This objects is removed from any final output data of the sensor model.

Finally, the third interface osi::LowLevelData does not store object data, but holds low level sensor data that is very specific to each type of sensor. A general definition for low level data is data that relates directly to the sensor's measurement process and might be processed by specific algorithms for cleaning or de-noising, yet has not been processed by higher level perception functions. Aside from the general items of the sensor's identifier, the sensor time stamp and the sensor's mounting position, a lidar point cloud and a radar reflection list are defined in the interface. At the current point in time, sensor measurement models for lidar and radar sensors have been investigated and as such have a representation in the interface defintion. Within the scope of this work, the lidar sensor measurement model is discussed in chapter 4. The lidar point cloud represents the low level output data for an automotive lidar sensor and consists of a list of reflection points in three dimensional space. Each reflection point is additionally enriched with data of additional measured quantities depending on the specifics of the lidar sensor. Examples are the classification of reflecting target or the intensity of reflected light.

A closing remark concerning the open simulation interface is owed to coordinate systems. The ground truth interface osi::GroundTruth is defined independent of any specific object or sensor and therefore uses a global coordinate system that can be defined by the simulation framework following some standardized rules, e.g. using a right-hand coordinate system. As the osi::SensorData and osi::LowLevelData interfaces each hold data that represents the output of a specific sensor, the environment description within these interfaces is given with respect to the sensor's own coordinate system. Using the information about the position and orientation of the sensor with respect to the standardized vehicle coordinate system that is encoded in the MountingPosition item within each interface description, the sensor data can be transformed from the sensor coordinate system to the vehicle coordinate system if this is required.

2.6 Summary

In this chapter general concepts regarding virtual environmental perception and sensor simulation are introduced and discussed. This starts with an introduction of the framework and processing chains for virtual development and testing. It is discussed that depending on the design application for a sensor model, which is closely linked to the level of abstraction that is considered, a different type of sensor model is required. For the purpose of virtual testing, the most relevant abstraction levels are the object level and the level of low level sensor data processing. Within the object level the focus lies on the dynamics of objects making it most suited for testing the behavior of the full system in a virtual environment. The lower processing level has its focus on the details of the perception process and is therefore more suited to understand and test the perception part of the system. The definition of the sensor error model as a statistical description of the sensing process and the sensor measurement model as a physically motivated approach to a low level description of the sensor measurement process are introduced. As a main pain point in the development of sensor models are varying input and output formats, the Open Simulation Interface (OSI) is introduced as a proposed standard for use by sensor models. The layout of OSI is directly based on the target architecture for the development of sensor models for virtual perception. This provides the foundation for the discussion of the sensor error and sensor measurement models in the next chapters.

3 Sensor Error Model

As one of the general two types of sensor models from the introductory overview in section 2.4, sensor error models aim to reproduce the sensor output after processing by the perception function on a statistical level. In the simulation of environmental perception using this type of model, the sensory perception process is handled by a software component that maps the ground truth to the sensor output, which is usually a high-level description of objects. The software component containing the sensor error model modifies the properties of the ground truth objects in the virtual environment in such a way that the resulting output exhibits the same statistical properties as the output of the original real world sensors.

This chapter discusses first the scope of sensor error models and their typical application, followed by the introduction of a new modular approach as a general recipe for the construction of sensor error models. Then, the real world setup for gathering reference data used in model construction and the virtual world setup for using the model with the driving simulation environment are introduced as used in the scope of this thesis. Finally, for a number of modules the derivation from reference data for each module's particular scope is discussed and a representation in the simulation model defined based on the reference data.

3.1 Scope and Applications of Sensor Error Models

Sensor error models have the purpose to form a single model for the virtual representation of an automotive sensor that provides high level object data as its output. This type of sensor model has two main design applications: They can reach a high computational performance due to their low complexity. This allows their use for a large scale simulations that include a wide variety of different scenarios as they are used for the functional validation of an automated driving system Further, as this type of model operates as a black box model on the object level interfaces of a sensor, constructing and using a sensor error model does not require intricate knowledge of the inner workings of a sensor. This knowledge is in many cases is not readily available, thereby forcing the use of a sensor error model. For these reasons, sensor error models are mainly used to either test the high level perception processing, e.g. sensor fusion, on the object level within an automated driving system or the automated driving system as a whole.

The general context for the usage of sensor error models in the larger context of virtual development and testing is shown in Fig. 3.1. The fast computation of sensor

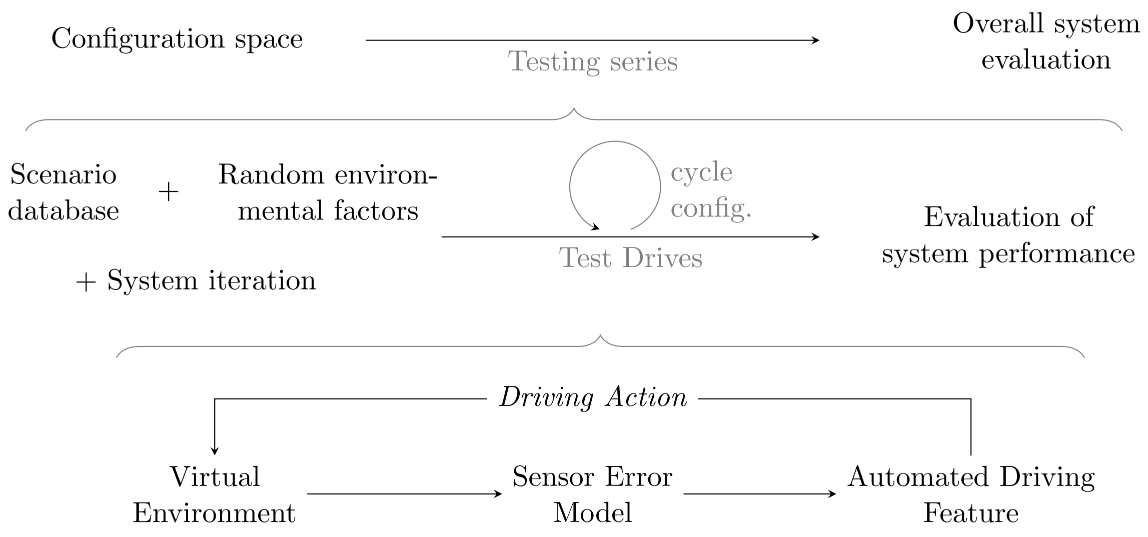


Figure 3.1: Main application of sensor error models in the larger context of virtual development and testing. The fast computation of sensor error models enables running closed loop simulations with an automated driving feature under test at faster than real time speed (lowest part of the figure). This in turn enables a large number of test drives being run in short order, cycling through different configuration for either the scenario or environmental factors (mid part of the figure). Finally, this allows an overall evaluation of the system based on a large configuration space sampled by the testing series of repeated virtual world test drives (upper part of the figure).

error models enables running closed loop simulations with an automated driving feature under test at faster than real time speed. This enables enables a large number of test drives being run in short order, cycling through different configuration for either the scenario or environmental factors. This is performed by either running a long-term continuous simulation of a large scale scenario with a large number of actors or by repeated tests of individual scenarios with a variation of parameters, e.g. parameters of the dynamic environment like traffic density or parameters of the static environment like road layout, number of lanes, highway exits, lane mergers, or similar. As a result, simulations using sensor error models allow a speedy overall evaluation of the system based on a sampling a large configuration space by means of virtual world test drives.

The sensor error model is a statistical model that aims to reproduce the statistical properties of the sensor output data. For this reason, the parametrization of a sensor error model requires statistics gathered from observed sensor data in a variety of test drive scenarios. In order to arrive at valid statistics, the sensor setup, i.e. type and location of sensors on the vehicle, is fixed for these test drives. Owing to this, a sensor error model with a specific parametrization always reproduces data from this exact setup. It is not possible to investigate the influence of changes in the sensor setup with this type of model. This is the domain of physically motivated sensor measurement models discussed in chapter 4. However, for some parts of the model, e.g. a simple field of view model, it is possible to use parameters directly from the sensor specification without the need to gather sophisticated statistics data from test drives.

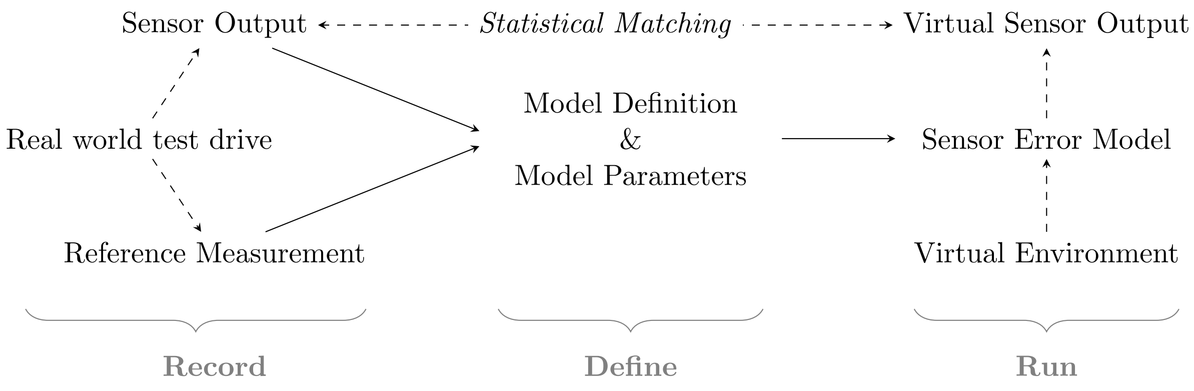


Figure 3.2: Sequence of steps in the construction of a sensor error model, from left to right: Record, Define, Run. From a statistical analysis of recorded sensor output and reference measurements from real world test drives, the model and its parameters can be defined. The sensor error model constructed in this way can then be used to produce virtual sensor output data from a virtual environment that matches the real world sensor output on a statistical level.

We consider the sensor model statistically matching the sensory perception data of a real sensor if the generated sensor output data stimulate the automated driving system in the same way over a large number of driving scenarios. An exact matching of the data generated by a sensor in a specific situation is not required (or even possible) as long as the model reproduces the relevant statistical properties, e.g. variances, and correctly handles trigger conditions for the occurrence of specific effects. Note that a direct inclusion of causal relations of trigger conditions to an error in the sensor output data is not required. Instead, a correct reproduction of the relevant correlations between effects that impact the behavior of the automated driving system is sufficient.

Reference data come in two flavors, dynamic or static, depending on the object property in question. This has a significant impact on the gathering of reference data and is also reflected in the category of affected model units according to Fig. 3.6. Dynamic properties like position and velocity of a dynamic object in the environment constantly change in value and the true value corresponding to a measurement only exists for the instant that the measurement was taken. Therefore, reference data have to be acquired at the same time as the sensor measurement data and with generally higher precision. In contrast, static properties have a fixed true value that does not change with time, e.g. the classification and size of an object have fixed values that the sensor is attempting to determine correctly during the perception process. This simplifies the acquisition of reference data immensely as any one point in time may be used.

The series of steps required for construction a sensor error model are shown in Fig. 3.2. This series of steps follows the sequence: Record, Define, Run. A sensor error model is always built based on real world test drive data, where during a data analysis step the recorded sensor output is matched against a reference measurement. This is the Record step, and followed by the Define step that deduces the mathematical definition of the model as well as the model parameters from the analysis of the test drive data.

Finally, in the Run step, the sensor model needs to be implemented and embedded in a virtual world perception processing chain using ground truth data from the virtual environment of a driving simulator to generator virtual sensor output data. If all steps during model constructing have been performed correctly, this virtual sensor output data should match the real world sensor output on a statistical level. Validating this based on reference measurements and the simulation ground truth is then the task of sensor model validation.

3.2 A Modular Approach to Sensor Model Construction

There are a variety of ways to construct a sensor model that maps ground truth data from the simulation framework to virtual sensor output. With an iterative development methodology, simplifying future extensions as well as allowing the reuse of the sensor model in a variety of contexts in mind, a modular approach to sensor model construction naturally arises to fit these requirements. This approach lies at the heart of the discussions of sensor error models in the scope of this thesis and the idea behind the approach is introduced in this section.

3.2.1 From integrated to modular sensor models

In section 2.3 on the formal description of sensor models, equation 2.1 was introduced, which renders the perception process as a mapping M, depending on sensor parameters $\mathbf{p}$ and environmental conditions $\mathbf{c}$ and mapping the set of sensor targets $X = \{\mathbf{x}_i\}_{i\in[1,N_X]}$ onto the set of observed targets $Z = \{\mathbf{z}_j\}_{j\in[1,N_Z]}$:

$$M(\mathbf{p},\mathbf{c}) : \{\mathbf{x}_i\}_{i\in[1,N_X]} \to \{\mathbf{z}_j\}_{j\in[1,N_Z]}\,. \tag{3.1}$$

Based on this equation, it is possible to introduce the idea of splitting the mapping representing the sensor model into a chain of subsequent partial mappings, i.e.

$$M = M^{(n)} \circ M^{(n-1)} \circ \cdots \circ M^{(2)} \circ M^{(1)},$$

where the output of each individual mapping becomes the input for the next mapping. It is a good idea to define the first mapping $M^{(1)}$ as a mapping between the ground truth data format and a standardized intermediate and output format, which then becomes the identical format for input and output of all subsequent mapping, i.e.

$$\begin{aligned} M^{(1)}(\mathbf{p},\mathbf{c}) &: \{\mathbf{x}_i\}_{i\in[1,N_X]} \to \{\mathbf{z}_j\}^{(1)}_{j\in[1,N_Z^{(1)}]}\,,\\ M^{(k)}(\mathbf{p},\mathbf{c}) &: \{\mathbf{z}_i\}^{(k-1)}_{i\in[1,N_Z^{(k-1)}]} \to \{\mathbf{z}_j\}^{(k)}_{j\in[1,N_Z^{(k)}]}\,, k\in[2,n]. \end{aligned} \tag{3.2}$$

In this formulation, the sensor model is broken down into a series of modules that are applied to the current iteration of the sensor data in sequence. Due to the identical format of the input and output data, it is possible to execute the modules in any order

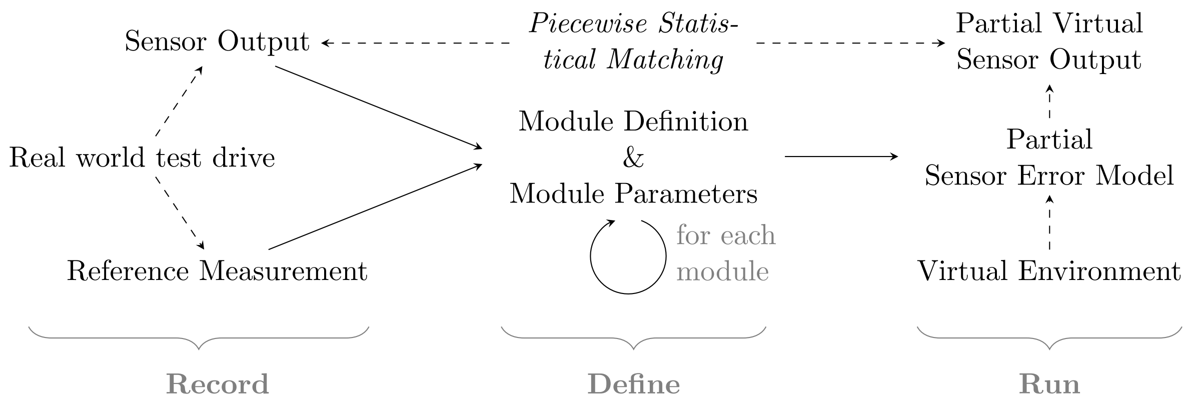

Sensor Output
Piecewise Statistical Matching
Partial Virtual Sensor Output
Real world test drive
Module Definition & Module Parameters
Partial Sensor Error Model
for each module
Reference Measurement
Virtual Environment
Record
Define
Run

Figure 3.3: The Record, Define, Run sequence of steps in the construction of a sensor error model modified for a model using modules. The sequence of deriving model and parameters from statistical analysis of recorded data remains the same and is executed individually for each module, iteratively expanding the model as a whole.

as long as any modules do not operate on the same properties within the data. This allows to build individual modules for the modeling of a specific effect or aspect of the detection process, which enables a continuous, iterative improvement of the model as well as a separation of effects. When using a standardized format for the intermediate and final output data like the osi::SensorData format introduced in section 2.5, it also becomes possible to easily combine modules from different developers into one single sensor model.

The characteristics of the sensor mapping M in (2.1) are split into n modules $M^{(k)}$ with $k \in [1, n]$. Operating in sequence, each module takes a set of objects $Z^{(k-1)} = \{\mathbf{z}_i\}_i^{(k-1)}$ as input and returns a modified set $Z^{(k)}$. The composition of all individual modules then defines M. The specific order of modules is an assumption of the model and the composition is in general not commutative. Modules are interchangeable if their effect on the final output set $Z^{(n)}$ is restricted either to separate subsets of elements in $Z^{(n)}$, e.g. targets in front of the vehicle and those behind it, or separate subsets of target properties, e.g. target position and size of bounding box. The set of configuration parameters for module $M^{(k)}$ is denoted by γ_k and comprises the relevant subset of the sensor properties $\mathbf{p}$ and environmental conditions $\mathbf{c}$ or quantities calculated from these depending on the specific requirements of the module. The sensor model is then defined by

$$M(\mathbf{p}, \mathbf{c}) = M^{(n)}(\gamma_n) \circ \ldots \circ M^{(2)}(\gamma_2) \circ M^{(1)}(\gamma_1), \tag{3.3a}$$

$$M^{(k)}(\gamma_k) : \{\mathbf{z}_i\}_{i\in[1,N_{k-1}]}^{(k-1)} \to \{\mathbf{z}_j\}_{j\in[1,N_k]}^{(k)}, \tag{3.3b}$$

where $\{\mathbf{z}_i\}_i^{(0)} \equiv \{\mathbf{x}_i\}_i$ corresponds to ground truth data and $\{\mathbf{z}_j\}_j^{(n)} \equiv \{\mathbf{z}_j\}_j$ to the final output data of the sensor model. Correlations between modules or correlations in time can be taken into account by considering the output of previous modules in the sequence as well as the sensor output Z_{t-1} from the last update for the configuration γ_k

Applying the general methodology for construction of a sensor error model shown in Fig. 3.2 to the modular approach, results in the adapted process shown in Fig. 3.3. The sequence Record, Define, Run in the sensor model construction remains unchanged. The difference is that now step with definition of the model an its parameters now becomes a series of steps for the definition of individual modules and their module parameters. Each of the modules constitutes a partial sensor error model that is responsible for a part of of the simulated sensor output and therefore for each model a piecewise statistical matching between real world sensor output and virtual world sensor output can be performed. With each added module the sensor error model should come closer to the final goal of fully matching all effects observed in the real world sensor data.

For the construction of the statistical sensor model, we define atomic units of the model, each capturing one specific error resulting from the lossy perception chain, i.e. $M^{(k)} : \Psi \to \widetilde{\Psi}^{(k)}$, where $\widetilde{\Psi}^{(k)}$ is a subset of the simulated perception data. The direct sum of the output of all n model units then defines the complete output interface, i.e. $\widetilde{\Psi} = \bigoplus_{k=1}^{n} \widetilde{\Psi}^{(k)}$, where the combination of the n units defines the model for the task at hand. For modeling the perception process, each unit is tasked with the mapping of environmental data to one specific object property in the output interface in a way that replicates errors and uncertainties of the perception process for this particular quantity on a statistical level. Packaging errors concern the sensor interface at large. Each unit therefore deals with specific limitations of the interface and data transmission errors that may occur until the sensor data arrive at the intended recipient.

In this framework, each of the modules constituting the sensor model is responsible for modeling a specific effect or aspect of the detection process. This structure allows a continuous, iterative improvement of the model as well as a separation of effects. A straight-forward example for a module is the modeling of the sensor field-of-view via the removal of all elements in the set $Z^{(k-1)}$ with positions outside of the defined area. Other possibilities include adding or removing elements to account for false-positives or false-negatives, and the stochastic variation of target properties like object position. An implementation of the latter is shown in the following section.

For the general concept of a modular approach to sensor model construction introduced in this section, it is easy to build a very generic software design that can be enhanced step by step in continuous iterations. This makes the modular approach a great asset in the development of sensor models operating on the object level. In the following section, the software design for the realization of the modular architecture is introduced in detail.

3.2.2 Modular Architecture

The model architecture discussed in this work enables a stochastic model corresponding to a mapping as illustrated in Fig. 2.6 (b) of the previous chapter. The design aim is to provide a flexible framework facilitating an iterative development. This is achieved by

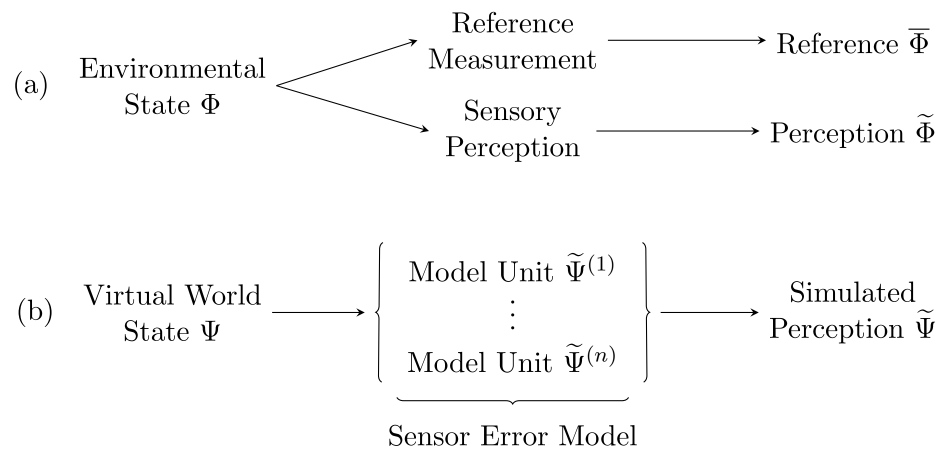

(a)
Environmental State Φ
Reference Measurement
Reference $\overline{\Phi}$
Sensory Perception
Perception $\widetilde{\Phi}$
(b)
Virtual World State Ψ
Model Unit $\widetilde{\Psi}^{(1)}$
Model Unit $\widetilde{\Psi}^{(n)}$
Simulated Perception $\widetilde{\Psi}$
Sensor Error Model

Figure 3.4: Overview of relations between data: (a) The sensory perception process maps the environmental state Φ of the physical world on perception data $\widetilde{\Phi}$. Similarly, reference data $\overline{\Phi}$ are obtained in a reference measurement. (b) In the simulation, the current state Ψ of the virtual world is mapped onto simulated perception data $\widetilde{\Psi}$ by means of a sensor error model. This model is comprised of several (atomic) units $\widetilde{\Psi}^{(k)}$ with index $k \in [1, n]$. The aim of the statistical sensor model is an equivalence of sensory perception data $\widetilde{\Phi}$ and simulated perception data $\widetilde{\Psi}$ on a statistical level given comparable input states Φ and Ψ.

decoupling the sensor model logic from the specifics of the simulation framework using well-defined interfaces as described in the section 2.5, introducing the open simulation interface, as well as specifying a modular structure for the model components.

The proposed architecture is shown in Fig. 3.5. [see Fig. 3.5].

This type of flexible architecture allows to build a sensor model by combination of individual modules, where each module corresponds to an atomic unit of the model. The specific combination is determined by the output interface (or subset thereof) required for the use case of the desired application. The sequence order of model units belonging to the perception categories follows the natural ordering of the perception sequence as shown in Fig. 3.6. Packaging model units related to the interface contents are placed at the end of the sequence as they modify the interface in its entirety. An exception are effects that result in no data being transmitted at all, e.g. the sensor startup state. In this case the corresponding module should be first in sequence to avoid unnecessary computations. Packaging model units on the interface level are not part of the sequence that processes and modifies interface contents. Instead, these require implementation on framework-level, especially in regard to timing specifics.

3.2.3 Sensor Error Classification

On the object level, all data output by the sensor are the result of sophisticated object detection and tracking algorithms. However, the processing required to arrive at the sensor output data from measurement raw data varies significantly for the various object properties. Given a distance measuring sensor like a radar system, calculation

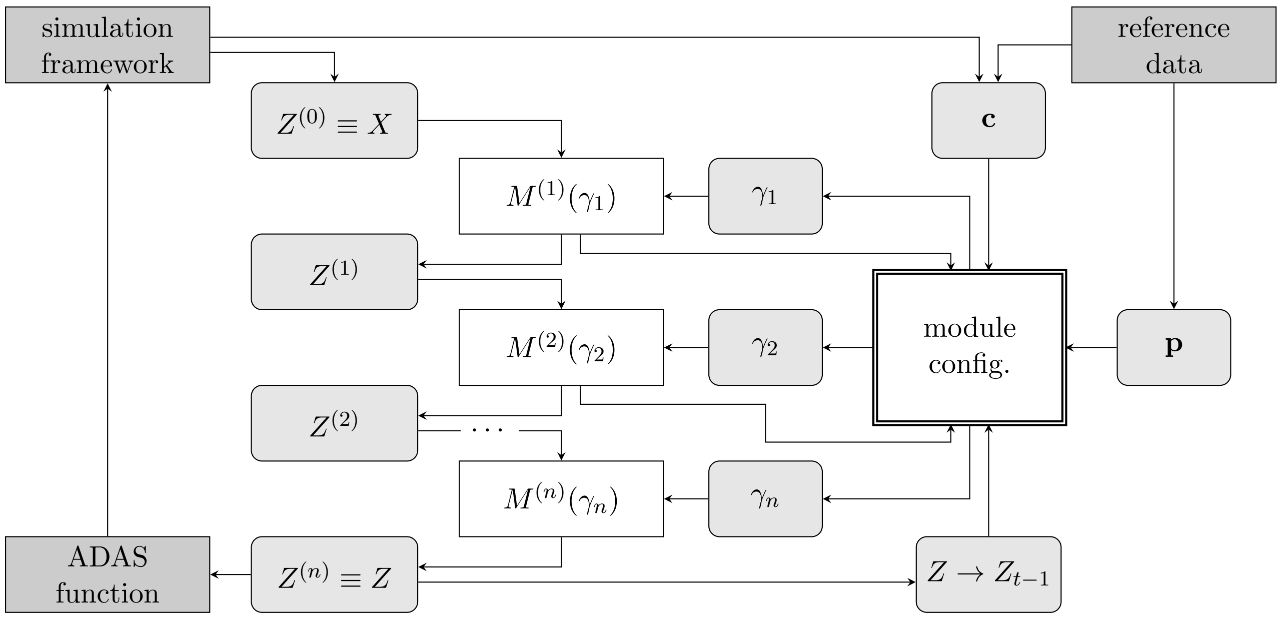


Figure 3.5: Modular architecture of the sensor model: The driving simulation framework provides the ground truth data $Z^{(0)} \equiv X$ that is then operated on by the n sensor model modules $M^{(k)}(\gamma_k)$ in sequence, each producing a modified set of objects $Z^{(k)} = \{\mathbf{z}_i\}_i^{(k)}$. The final result is the list of detected objects $Z^{(n)} \equiv Z$ that can be used as input for the ADAS function algorithms to establish a closed control loop. The individual modules represent the sensor characteristics. Their configuration is given by the set of module parameters γ_k that is determined in a previous step. It depends on sensor properties $\mathbf{p}$ and environmental conditions $\mathbf{c}$ as well as the output of previous modules $M^{(1..k-1)}$ and the sensor output Z_{t-1} from the last update to account for correlations in time and between modules. Values for parametrization are determined using reference data, for example statistics for relative position readings from a sensor in comparison to higher precision measurements made by a reference system.

of the target object position from the measured data is rather straightforward. In contrast, differentiating between different types of objects on the basis of the abstract radar cross section will require significantly more processing. The complexity of the respective processing will in turn significantly influence the behavior of the quantity when considering a sensor update in the simulation, i.e. the generation of the current simulated sensor output data from the state of the virtual environment. For a systematic approach to sensor model construction, it is useful to sort sensor errors and their corresponding model units into different classification categories according to their position in the processing chain.

The choice of categories is motivated by two factors: the type of stochastic description constituting the specific model unit and the type of data required for reference and validation purposes. On this basis, we propose a classification scheme defined by the classification hierarchy shown in Fig. 3.6. An overview of where the sensory perception of several typical object properties fits into the classification hierarchy is given in Table 3.1.

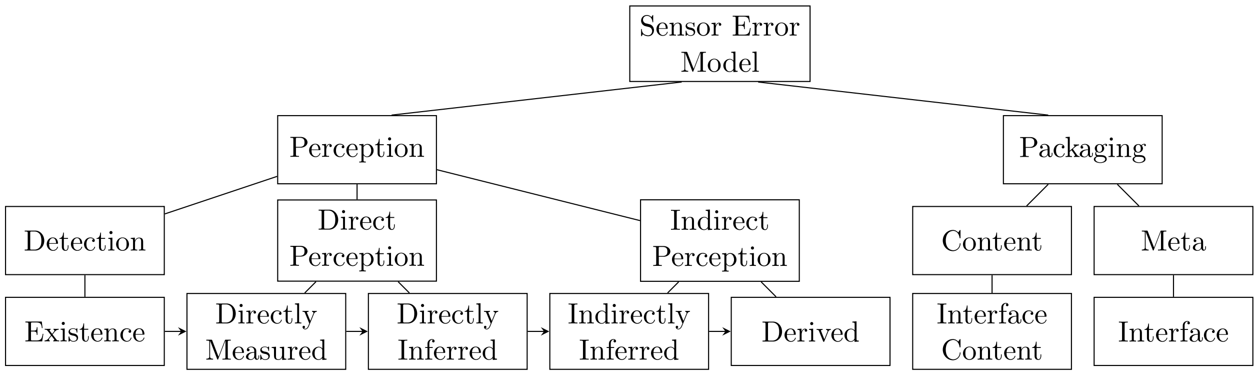


Figure 3.6: Classification hierarchy for model units constituting the sensor error model. Perception errors are divided into three classes: detection, direct or indirect perception. Direct perception encompasses those object properties that have a direct relation to measurement raw data through straightforward calculation whereas indirect perception signifies probabilistic inference. The arrows indicate the perception sequence.

On the most basic level, errors are divided into perception errors pertaining to the measurement and perception process, and packaging errors concerning the encoding of perception data into the sensor output interface. For perception, we further distinguish between detection, direct and indirect perception.

Direct perception usually deals with dynamic data. In this case, the statistical properties of the sensor measurement are deduced from a comparison of sensor readings and simultaneous reference measurements. It is important that variance of reference measurements is small compared to that of the sensor and that correlations between sensor and reference measurements are insignificant. The former increases, and the latter decreases the variance extracted for the sensor model, both of which would require a heuristic correction factor.

Indirect perception usually deals with quantities that are either static in nature, e.g. object classification, or where the true value is up to manual interpretation for each specific scenario, e.g. deciding whether another vehicle should be classified as cutting into the ego lane at the current point in time. For the former, it is important that the sensor model reproduces the transitions from the initial value at the time of first detection of the object to updated values of the probabilistic estimation of the sensor's perception function. For the latter, the model only has to adequately reproduce correlations between trigger conditions and state transitions over a large set of driving scenarios.

For models in the indirect perception category, this allows the definition of two parameters for model optimization and proof of validity by applying the model logic to recorded traces of real world reference data. If reference data show changes in the quantity handled by a specific model unit that are not reproduced by the corresponding trigger conditions, the model is not *comprehensive* and requires extension. In contrast, if trigger conditions predict a change that is not present in the data, the model is not *specific* and requires additional constraints.

Table 3.1: Error classification for various measured quantities depending on sensor type

Meas. Quantity	**Radar**	**Lidar**	**Camera**
Detection	existence	existence	existence
Position	directly meas.	directly meas.	directly meas.
Velocity	directly meas.	directly inf.	directly inf.
Acceleration	directly inf.	directly inf.	directly inf.
Bounding box	indirectly inf.	directly meas.	directly meas.
Orientation	directly inf.	directly meas.	directly inf.
Rotation rate	directly inf.	directly inf.	directly inf.
Type/Class.[1]	indirectly inf.	indirectly inf.	indirectly inf.
Associated lane	derived	derived	derived
Cutting in[2]	derived	derived	derived
Light state	—	—	directly meas.
Lane markings[3]	—	directly inf.	directly inf.

[1] *Type* defines the general category of an object, e.g. vehicle, static obstacle, or traffic sign. *Classification* defines a fine-grained subtype, e.g. car, truck, or bike for vehicles.
[2] *Cutting in* is exemplary of similar derived qualifiers depending on multi-object data (in this case: ego vehicle, other vehicle, and ego lane data).
[3] *Lane markings* is used as shorthand for all lane and lane marking related information, e.g. type, color, width, curvature.

There is a natural ordering to the perception sequence indicated by the arrows in Fig. 3.6. If there are correlations between two effects, errors occurring earlier in the perception sequence have an impact on those in subsequent steps but not vice versa. For example, an incorrect position measurement, positioning a vehicle in an adjacent lane closer to the ego lane than it actually is, will impact the lane mapping or the detection whether the vehicle is cutting into the ego lane. However, an erroneous association of the vehicle with the ego lane will not impact the position measurement.

3.3 Virtual and Real World Driving

As introduced in the discussions so far, the basic premise of a sensor error model is the use of real world test drive and reference data to define the model and model parameters. These reference measurements are used to deduce parameters for the sensor model so that it matches the specific real world sensor setup that has been used during recording

of the data. The sensor error model constructed in this way can be considered to statistically match the sensory perception data of a real sensor if the generated sensor output data stimulate the automated driving system in the same way over a large number of driving scenarios. An exact matching of the data generated by a sensor in a specific situation is not required (or even possible) as long as the model reproduces the relevant statistical properties, e.g. variances, and correctly handles trigger conditions for the occurrence of specific effects.
In the following the real world setup is introduced that is used for performing the reference measurements to determine model parameters, followed by the introduction of the virtual world setup for sensor simulation.

3.3.1 Real World Setup for Gathering Reference Data

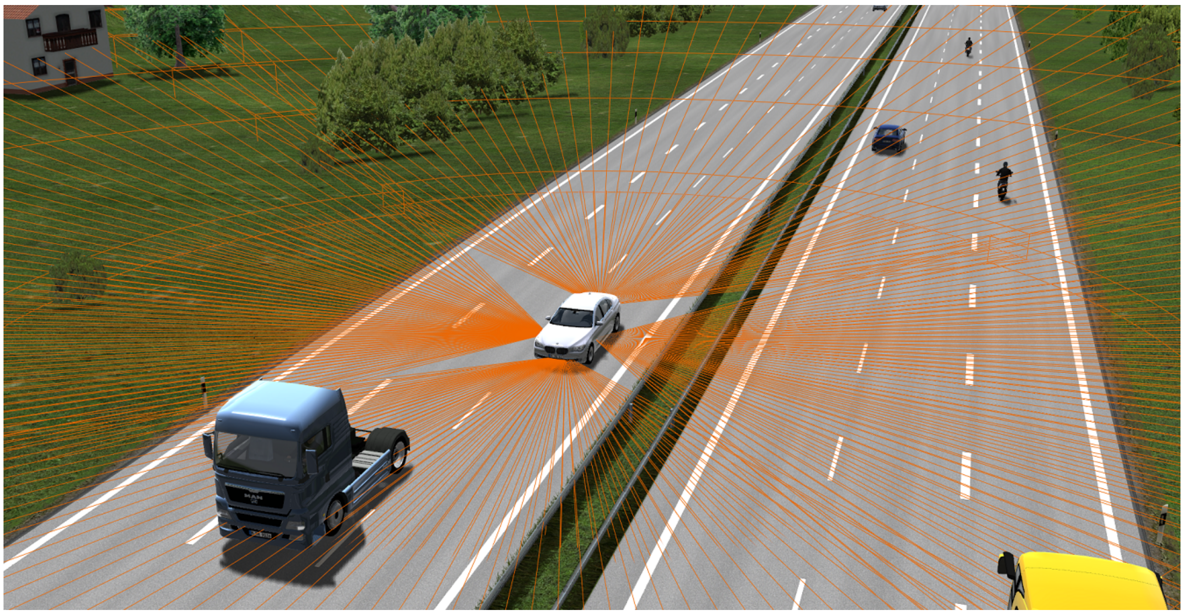
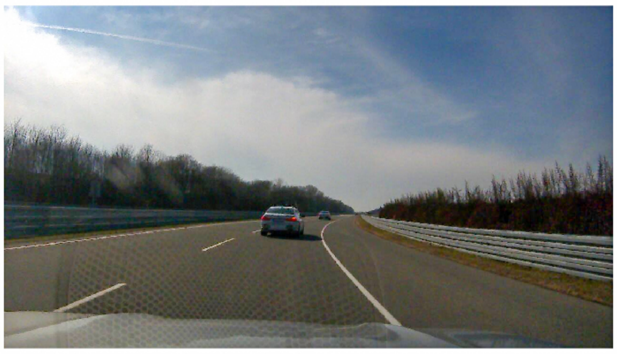
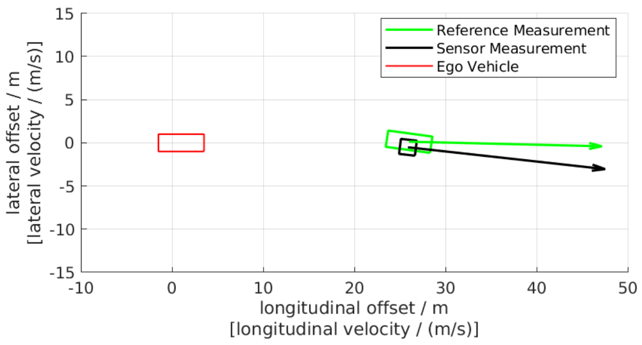


Figure 3.7: Rendering of the position and approximate field of view of four IBEO Lux laserscanner units placed on each side of the vehicle. The research vehicle used in real world test drives employs this configuration, which is used gathering of sensor and reference data, with the reference provided by a differential GPS system. The same sensor configuration is simulated in the virtual world.

In order to gather reference data that can be used in the definition of a sensor model and its parameters, two things are required: the sensor under investigation to record its output and another reference sensor that provides measurements of sufficiently higher quality to be matched as ground truth against the sensor's output. For the purposes of the analysis shown in this work, a setup consisting of two research vehicles is used. On of the vehicle, the so called *ego vehicle*, hosts the sensor under investigation: an IBEO Lux laserscanner setup. This sensor setup consists of four IBEO Lux laserscanner units mounted directly within the chassis at the front, rear and to each side of the vehicle, and a separate electronic control unit for data processing. The mounting positions and the respective approximate field of view of each sensor are shown in a virtual rendering in Fig. 3.7. The IBEO Lux sensors used in this setup use a 905 nm laser and the time of flight principle to detect the distance to a reflecting surface for each scanning point. For these test drives, a 4-layer sensor has been used that has a horizontal field of view of 110° and a vertical field of view of 3.2°. In the setup consisting of four sensors located on each side of the vehicle, the horizontal field's of view overlap, creating (after some initial distance) a full 360° coverage of the surrounding. For this reason, the resulting fused object list contains objects that may be in any direction from the car as long as they are within the sensor's range. The electronic control unit handling the data processing of the laserscanner setup receives the raw data streams directly from each of the four sensor units. It runs the proprietary sensor fusion, object detection and tracking algorithms of the sensors' vendor and offers output interfaces for the resulting fused object list as well as the point cloud raw data for each of the sensor units at an update rate of 25 Hz. As the input streams for the four sensors are merged before object detection, only the fused object list can be obtained as the individual lidar sensors in this setup do not output an object list based only on their own measurements.



(a)



(b)

Figure 3.8: Real world setup for gathering reference data. On the left side, (a) shows a still frame recorded by a camera mounted on the ego vehicle's dashboard that shows the target vehicle to the front of the ego vehicle. On the right side, (b) shows data gathered at this point in time. The black outline shows the target vehicle's relative position, size as well as orientation and the black arrows shows its velocity vector as reported by the lidar sensor setup mounted on the ego vehicle. The green outline and arrow show the ground truth data as recorded by the reference measurement system. The ego vehicle is shown in red, with the center point of it's rear axis defining the origin of the relative coordinate system.

In addition to the laserscanner setup, a camera is mounted on the ego vehicle's dashboard in order to provide a visual reference. A still frame snapshot of a test drive during a highway-like scene on a test track is shown in Fig. 3.8 (a). As can be seen on this camera frame, a second vehicle, usually referred to as *target vehicle*, is part of the test setup and driving within the sensor's field of view. Both, ego and target vehicles, are equipped with differential GPS measurement units and linked by means of an OxTS RT-Range setup to provide high accuracy ground truth measurement for relative position, velocity and orientation that can be compared to the lidar sensor's measurements. The output from this setup is shown in Fig. 3.8 (b). The frame of reference is defined by the standard coordinate system of the ego vehicle where the center of the ego vehicle's rear axis defines the origin. The outline of the ego vehicle is shown in red in the figure, the outline and position of the target vehicle according to the reference measurement system is shown in green, and the data recorded from the lidar sensor setup is shown in black. The green and black arrows indicate the absolute velocity vectors of the target vehicle as reported by reference and lidar sensors, respectively.

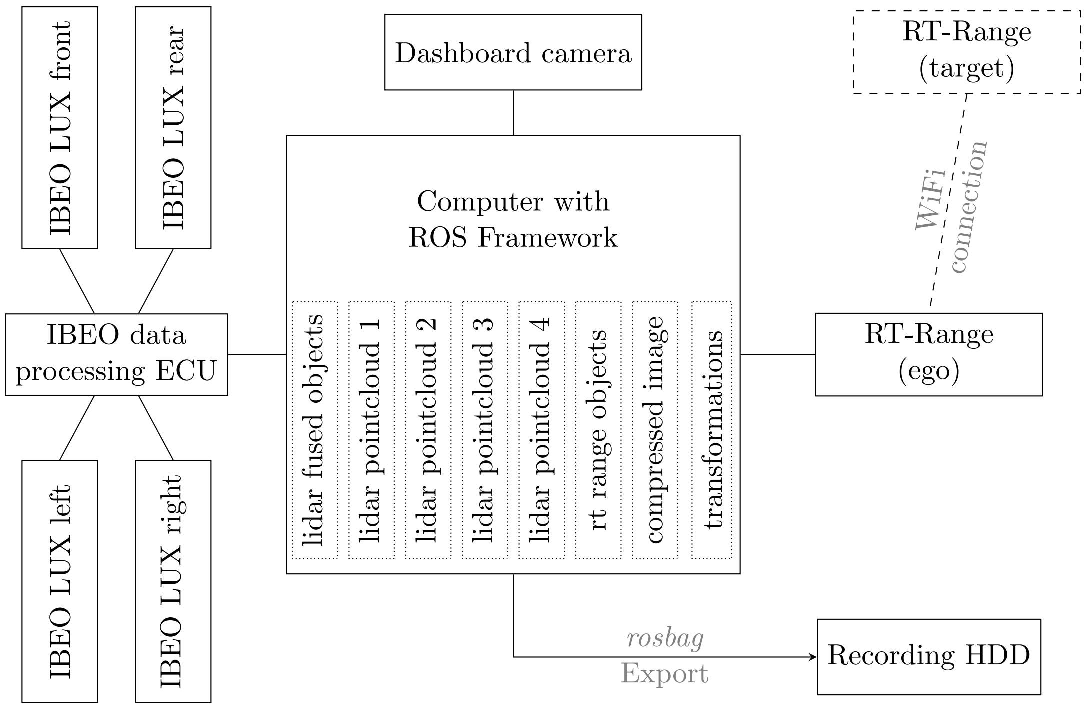


Figure 3.9: Data acquisition setup during real world test drives for gathering sensor and reference data. All the input data are handled by a computer running several sensor processing nodes within the Robot Operating System (ROS). Inputs to this computer are provided by an IBEO data processing ECU, a dashboard camera and an RT-Range system. The IBEO ECU provides lidar point cloud data from four IBEO Lux sensors mounted on the ego vehicle as well as the fused object list. The dashboard camera provides a compressed image stream for visual reference data. The RT-Range system is coupled by a wireless link to another RT-Range system on the target vehicle and provides the reference measurements. All data are handled as ROS topics (marked by dotted outlines) and recorded using the standard rosbag mechanism to a hard disk for offline processing and analysis.

To combine all of these individual input, the ego vehicle is outfitted with a computer running the Robot Operating System (ROS). The full set for acquisition of sensor and reference data in the research vehicle is shown in Fig. 3.9. For each of the data input streams, lidar sensor data from the IBEO ECU, a camera image stream from the dashboard camera, and the reference measurements from the RT-Range system, a

ROS node is installed that decodes the respective data streams from their specific wire format and publishes the data within ROS in the from of ROS topics. For the setup considered for the scope of this work, the topics carrying relevant data are the fused object list from the lidar sensor setup and the lidar point cloud data from each of the sensors, the image stream from the dashboard camera, the reference object data from the RT-Range setup, and the transformations topic carrying information about the ego vehicle's motion as the frame of reference. These data streams are saved to a hard disk for offline analysis outside of the vehicle using the standard rosbag mechanism provided by the framework. Timestamping of data is performed by the ROS Framework upon reception of data, assuming that no significant lag in data forwarding occurs in any of the different streams in relation to the others. This is a necessary trade-off due to the limitation of a lacking synchronized time source that can push a time stamp to each component.

After gathering data during a set of test drives with different driving scenarios involving the ego and target vehicle, the recorded data needs to be cleaned and prepared for analysis. These steps in particular include the matching of the observed object list from the lidar sensor against the reference measurements, associating objects, and synchronizing data. As the lidar sensor and the reference sensor have different frame rates and do not perform synchronized measurements, the measurement time points have to synchronized in order to allow a proper comparison of sensor measurement against reference ground truth data. For this purpose, the data from the reference sensor is linearly interpolated to exactly fit with the sensor data's timestamps. Further processing and preparation needs to be applied before the data are ready for analysis for the purpose of obtaining sensor model parameters. These steps are explained in more detail in reference [59], which is the work of a colleague of the author, who is largely responsible for the implementation of the data preparation framework that allows turning the recorded rosbag files from test drives into usable Matlab data for analysis.

Armed with the data from these real world test drives, a sensor model can be constructed that is employed in a virtual environment. The setup used for this virtual environment is introduced in the following.

3.3.2 Virtual World Setup for Sensor Error Model

The virtual world setup for running a sensor error model consists of three components, each of which is a framework running one part of the sensor simulation processing chain: the driving simulation framework, the sensor simulation framework, and the framework running the automated driving feature or other subsequent components using the virtual sensor data. During the research for this work, different frameworks have been employed at times for each of these three components. However, the following setup has emerged as the standard setup for most of the scope of this work due its accessibility and usability: As driving simulation framework the software Vires Virtual Test Drive (VTD), described in references [37, 60], is used. As an alternative driving simulation framework, especially for testing the compatibility of sensor simulation with different frameworks, the SPIDER framework [61] has been occasionally used. The sensor simulation framework is a stand-alone application that has been custom-built for the scope of this work. As the framework handling sensor data output, the Robot Operating System (ROS) is used as in the real world vehicle setup including the same ROS nodes that make use of the sensor data topics.

Using the same output handling framework as in the real world setup allows a direct reuse between the real and virtual world setups for the implementation of the automated driving function making use of the sensor data. The difference in the ROS configuration between the real and virtual environments lies solely in the use and configuration of ROS nodes that read sensor data. While the real world setup uses nodes for each sensor that decode the specific sensor data stream to publish them as a ROS sensor data topic, in the virtual world setup a custom-built *virtual sensorics* node is implemented that receives data from the sensor simulation framework using the OSI::SensorData (or OSI::LowLevelData) interfaces. The virtual sensorics node then converts these input data streams based on its configuration to the exact data structures of the ROS topics that are published for real world sensor data of the respective sensor. The virtual sensorics node is therefore the link between the simulation environment and the components making use of sensor data by providing an abstraction that prevents any of these components from realizing whether they are running in a real or virtual world environment.

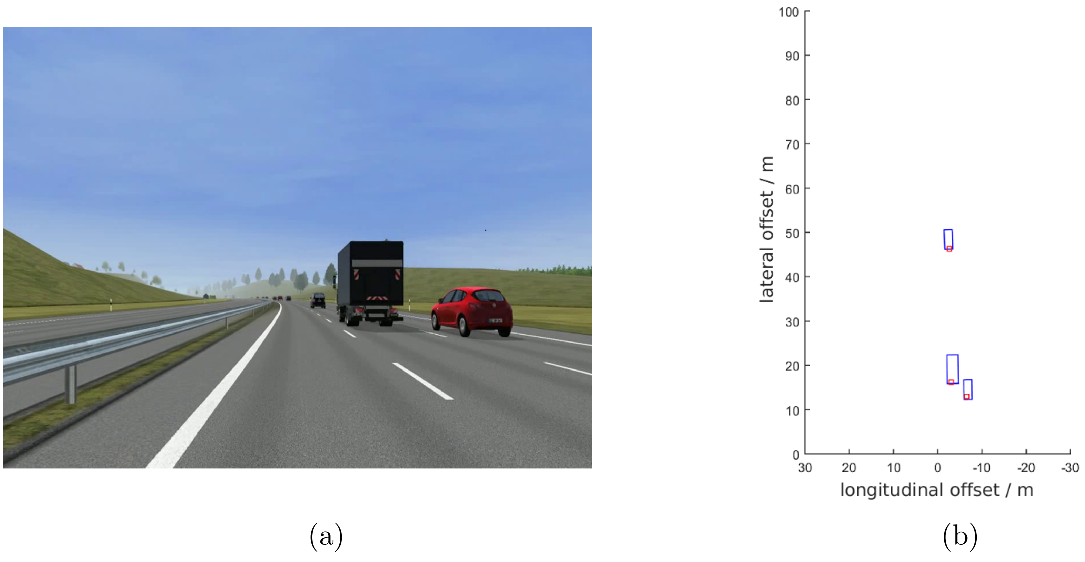


Figure 3.10: Output from the virtual environment showing the workings of the sensor model processing chain. On the left side, (a) shows a rendering of a highway scene from the ego vehicle's point of view as a visual reference. On the right side, (b) shows the outline of objects in the ground truth (blue) as well as the position measurement of a reference point on the back of the target vehicles as output by a sensor error model that only handles position measurement simulation (red).

Data handling before the virtual sensorics ROS node is heavily reliant on the interfaces of the open simulation interface (OSI) that have been introduced in section 2.5. Within the driving simulation framework a sensor plugin is implemented that sends the simulation's ground truth data at each update step of the simulation using the OSI::GroundTruth interface over a TCP/IP connection to the sensor simulation framework. In addition graphics data may be grabbed directly from the driving simulation framework's graphics renderer to provide a visualization similar to the dashboard camera of the real world setup. Such a graphical rendering of a highway scene in a virtual environment is shown in Fig. 3.10 (a). The OSI::GroundTruth data are received by the sensor simulation framework, which generates multiple instances of the data, one for each configured sensor, and transforms it into the relative sensor coordinate frame using the OSI::SensorData interface to store the data. Each data set is then processed by a series of modules forming the sensor error model for the respective sensor with continuous use of the OSI::SensorData format. The output of each sensor model is provided in the update rate of the target sensor, e.g. 25 Hz for the IBEO Lux lidar sensor model, and transmitted by a TCP/IP connection to the virtual sensorics node in the ROS framework.

Due to the use of TCP/IP connections between the driving simulation framework, sensor simulation framework, and ROS framework, each component can easily be partitioned on the same or different physical machines. For the scope of this work, mostly an integrated setup with all components on a single machine is used. This workstation runs an Ubuntu Linux system on an Intel Xeon CPU with 8 cores at 3.5 GHz each, 64 GB of working memory, and an Nvidia Quadro M4000 graphics unit for graphical rendering. The driving simulation framework Vires VTD in version 2.1 and the sensor simulation framework are run directly on the Linux operating system, while the ROS framework is running in a virtual machine based on the same image that is also used for in vehicle testing.

Equipped with the real world setup for the recording of reference data and running data analysis and the virtual world setup for running the simulation and resulting sensor error model described in this section, a series of modules can be defined and parametrized that define the sensor error model. This is discussed in the next section.

3.4 Modules for Sensor Error Models

A sensor error model consists of a sequence of modules that each simulate a specific characteristic of the sensor behavior. For each of the different types of sensor errors introduced in section 3.2.3, a specific modeling approach is best suited to replicate sensor behavior in the simulation environment. In the following, several exemplary modules are introduced based on the reference data gathered from the setup introduced in section 3.3.1.

3.4.1 Detection

Detection is typically the first step when dealing with sensor data on the object level. This part of the model is responsible for the existence or non-existence of objects in the output interface, i.e. whether the simulated sensor reports seeing the object. In a statistical sensor error model, the mathematical description of the detection module takes the form of the following mapping:

$$M_{detection}(\gamma_d) : \{\mathbf{z}_i\}_{i\in[1,N]} \to \left\{\mathbf{z}'_j\right\}_{j\in[1,N']}, \tag{3.4}$$

where $\mathbf{z}_i$ are a set of N objects in the list of objects before applying the module, and $\mathbf{z}'_j$ are N' objects after applying the module. For a field of view model, the modules parameter set γ_d consists of the horizontal and vertical field of view angles as well as the range of the sensor, assuming all objects are already given in coordinates relative to the sensor's position and orientation. Due to the nature of the sensor setup under investigation, with a combination of four lidar sensors with overlapping field of views as shown in Fig. 3.7 and the sensor output only providing a fused object list, the field of view of this sensor setup does not show any borders in the horizontal field of view. Unfortunately, from the set of measurements obtained from the real world test drives, a detailed analysis of the range of the sensor setup could not be determined. However, as a simple substitute, the range given in the sensor's specification sheet can be used for the implementation of a simple detection module. With possible further additions, requiring a larger database of real world sensor data, the detection module may be extended to additionally add false positives, i.e. non-existent ghost objects, to the data. Furthermore, requiring two target vehicles equipped with reference sensors, it would be possible to determine the behavior of shadowing, when one of the vehicles it blocking the sensor's view of the other. Due to the tracking algorithms used, this may not immediately mean the loss of the shadowed object in the sensor data.

3.4.2 Direct Perception

Direct perception encompasses all those object properties where straightforward calculations from measurement raw data may yield current values. Note that on the object level, filter algorithms during object detection and tracking have been applied to the measured data to arrive at the output values instead of straightforward calculations. In the framework of the statistical sensor error model, we assume a direct relation with a probabilistic mapping between the current environmental state and the sensor output. However, one has to be clear that this is not equal to a probabilistic model of the measurement process as it implicitly includes effects of the perception function.

A direct perception model can be constructed in two ways: Either as a probabilistic mapping $M^{(k)}(\gamma_k) : \Psi_t \to \widetilde{\Psi}_t^{(k)}$ that the state of the virtual environment Ψ_t at time t on the subset of the perceived state of the environment $\widetilde{\Psi}_t^{(k)}$, i.e. the sensor output. The upper index (k) defines the number of the module in the sequence in the notation of Eq. 3.3. Or, alternatively, including possible auto-correlations in the measurements, a direct perception model unit takes as input the current environmental state as well as the state during last update and the resulting model output, approximating the process as a Markov chain. This is an approximation of the categorization and more complex time dependencies require the model unit to shift to the indirect perception category. In this case, the probabilistic mapping takes the form $M^{(k)}_{direct}(\Psi_{t-1}, \widetilde{\Psi}^{(k)}_{t-1}) : \Psi_t \to \widetilde{\Psi}^{(k)}_t$, where Ψ_{t-1} denotes the state of the virtual environment at the previous time increment $t-1$, and $\widetilde{\Psi}^{(k)}_t$ denotes sensor output at $t-1$.

We further split direct perception into those quantities accessible through direct measurement or direct inference. The current values of directly measured quantities are directly accessible from a sensor measurement, e.g. object position for a lidar sensor. Note that the representation and coordinate system are insignificant for the classification as long as there exists a static transformation, e.g. object position may be given in polar or Cartesian coordinates and relative to the sensor or the vehicle reference frame. In contrast, the current value of directly inferred quantities is calculated from a sequence of sensor measurements. For a lidar sensor, an example for a directly inferred quantity is the object velocity as the current value of the object velocity is calculated from a series of position measurements that in turn are directly measured quantities. The the following exemplary modules implementing position and velocity measurements are discussed.

3.4.2.1 Position Measurement

The position measurement of a target object is performed by the lidar sensor setup based on object detection from the fused point cloud data of the four individual lidar sensors. In Fig. 3.11, a set of traces for the position measurement relative to the ego vehicle from test drive data are shown. The setup consists of the ego vehicle hosting the sensor and the a target vehicle as described in section 3.3.1. All measurements are transformed to the relative coordinate system of the ego vehicle, which is shown as a red outline in the graph. The forward, longitudinal direction of the ego vehicle is the rightwards direction in the graph. The relative positions reported by the reference measurement system based on a differential GPS system are shown in green in Fig. 3.11. Due to the nature of the test track, the target vehicle can have a large distance from the ego vehicle along the longitudinal direction, but only a comparably small lateral offset owing to the boundaries of the road. The measurements of the lidar sensor system with four IBEO Lux laser scanner sensors covering each side of the vehicle, is shown as the black line. As a comparison of the traces from the reference and sensor measurements clearly show, there typically is a difference in the measured position, constituting a sensor error to be captured by the simulation model.

Using a set of 21 such measurement series of sensor output matched against reference data as input, including a total of 32550 measurement points, a analysis of the data can be performed to determine model parameters. Using the relative coordinate system of the ego vehicle and denoting the longitudinal offset of the target vehicle relative to the ego vehicle as x and the lateral offset as y, the distribution of sensor measurement error

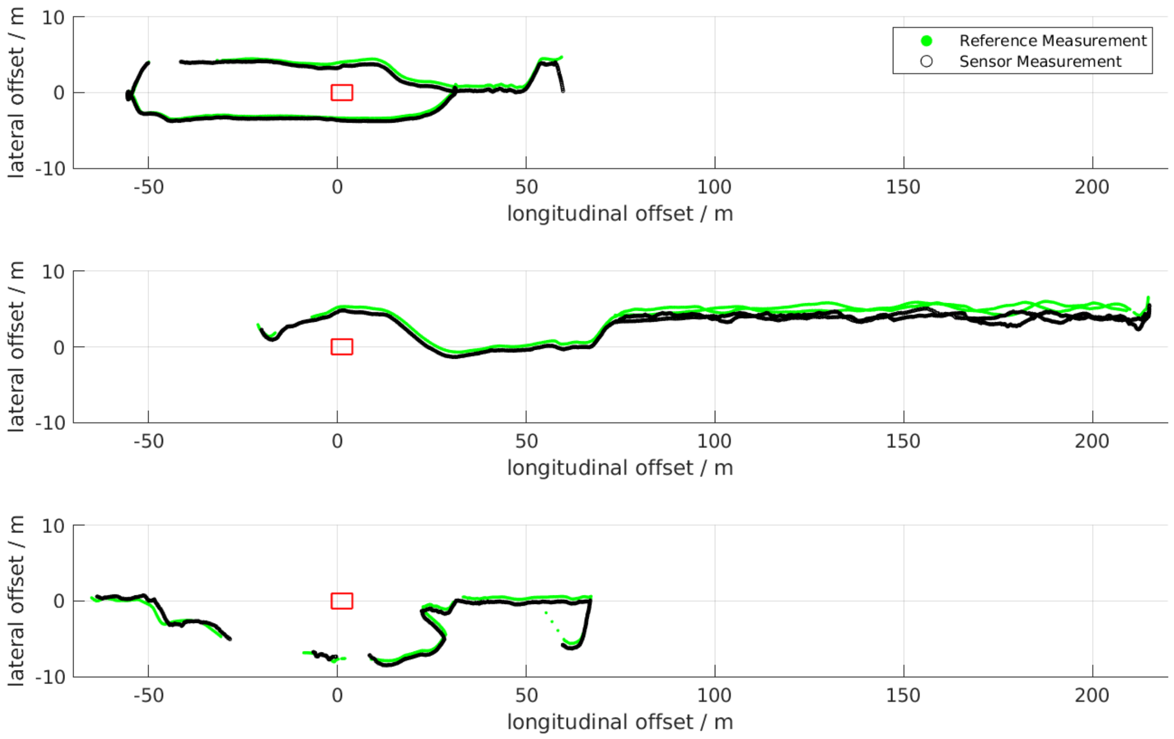


Figure 3.11: A set of traces of test drive data showing the relative position measurements of a target vehicle from sensor and reference measurements. The setup consists of two vehicles: The ego vehicle hosting the sensor and the target vehicle. The ego vehicle is shown as a red box with the center of its rear axle as the origin of the coordinate system. The forward, longitudinal direction of the ego vehicle is the rightwards direction in the graphic. The relative positions reported by the reference measurement system are shown in green and the sensor position measurements are shown in black. Note that the lateral offset is scaled differently to the longitudinal offset by a factor of 2 for better clarity.

$p(\hat{x})$ in the longitudinal direction and $p(\hat{y})$ in the lateral direction can be computed. The sensor error in longitudinal direction $\hat{x} = \widetilde{x} - \overline{x}$ is given as the difference of the sensor measurement $\widetilde{x}$ and the reference measurement $\overline{x}$. Same applies for the lateral measurement: $\hat{y} = \widetilde{y} - \overline{y}$. The resulting normalized histograms approximating the probability density functions are shown in Fig. 3.12. No further processing is applied at this stage.

The distribution of the lateral position offset, shown on the right side of Fig. 3.12, shows a very symmetric distribution with an offset of $\mu = -0.511\,\text{m}$ in the negative y-direction, where μ is the mean of the sample. The standard deviation of the sample is $\sigma = 0.442\,\text{m}$. A normal distribution defined by these parameters is shown as reference as the red line in the figure. A notable difference is that the measured distribution appears more sharply peaked than the fitted normal distribution. A similar behavior can be observed in the distribution of the longitudinal position offset, shown on the

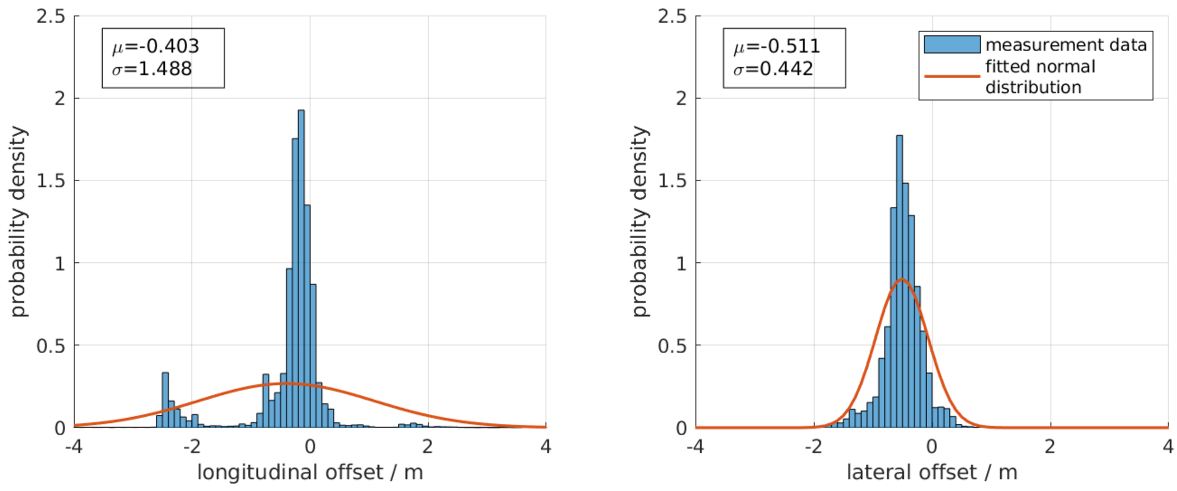


Figure 3.12: Probability density function for the offset between the longitudinal (left) and lateral (right) position measurement of the sensor setup and the position reported by the reference sensor for the same point in time. To each measured distribution, a normal distribution is fitted (red line). The resulting fit parameters for the mean μ and standard deviation σ computed from the measurement data are given in the top left corner of each graph.

left side of Fig. 3.12. For this distribution, the mean is $\mu = -0.403\,\text{m}$ and the standard deviation is $\sigma = 1.488\,\text{m}$. Comparing the measured distribution to the fitted normal distribution, reveals two characteristics: One, the computed standard deviation seems very large in comparison to the sharpness of the central peak, and two, there appears to be a smaller peak at around $-2\,\text{m}$.

The first observation is explained by long tails with low probability (therefore not visible in the figure) of the distribution due to some very large deviations between the position measurements of the sensor and reference. This can be seen when looking at the minimum and maximum values found in the data: for the longitudinal offset between sensor and reference $\min(\hat{x}) = -44.6\,\text{m}$, $\max(\hat{x}) = 28.4\,\text{m}$, and for the lateral offset $\min(\hat{x}) = -8.5\,\text{m}$, $\max(\hat{x}) = 11.1\,\text{m}$. These long tails have a significant impact on the standard deviation, but may simply be a result of low accuracy for high ranges that have occurred during some of the test drives. Whether these long tails still have a significant statistical weight within a massively increased data set, remains to be seen. The second observation of small secondary peaks can be explained by the observation of a great variation of the mean of the measured data for each individual test drive. The distribution of these means of the measurements of each individual continuous object tracking series is shown in Fig. 3.13.

From the histograms shown in Fig. 3.13 for the mean longitudinal and lateral offsets for each individual tracking series during the test drives, the large variation of the series' means in the longitudinal direction becomes apparent. These deviations between the means in the measurement and reference data very likely are a combination of three effects: First, the calibration of the reference system and the sensor system within the

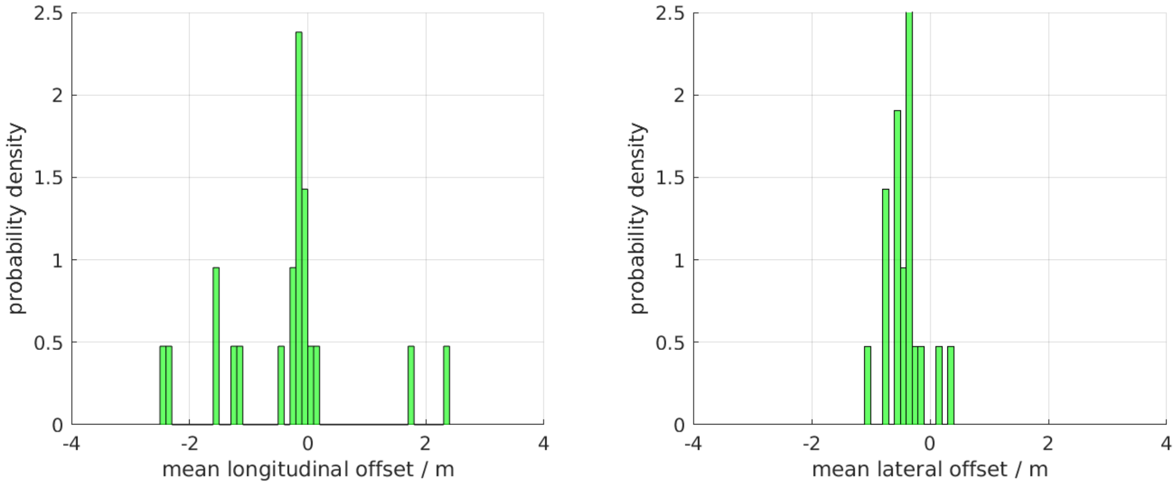


Figure 3.13: Histogram normalized as a probability density function of the mean offset between the sensor position measurement and reference position, with the mean calculated over each individual, continuous measurement series of an object. The mean offset in the longitudinal direction on the ego vehicle's relative coordinate system is shown on the left, and the mean offset in the lateral direction is shown on the right.

ego and target vehicles is likely not perfect as minimizing this systematic error requires immense effort in the hardware setup of the vehicle. Second, the reference point used in the reporting of the target vehicle may likely have been falsely estimated by the sensor's tracking algorithms in some cases. As typically one of the corners of the target is used as a reference point, with the most likely being the closest corner to the lidar sensor setup, a false estimation of the reference corner leads to a continuous offset of the reported trace on the order of the distance between corners, i.e. the vehicle's dimensions. Such an effect is possibly responsible for the larger variations seen in the mean longitudinal offsets on the left side of Fig. 3.13. Finally, third, the logic of the object tracking algorithms can lead to a sustained offset during a continuous tracking series of an object if such an offset was reported in the beginning as the history of previous measurements typically figures heavily into the object tracking algorithm.

These effects that systematically cause a shift in the mean offset during a tracking series, defined as the continuous trace of an object from initial detection to object loss for the tracking algorithm, may be handled in a separate model from the error of the position measurement during individual sensor update cycles. In order to differentiate these effects, the position measurements of the sensor and reference during a continuous trace of an object are corrected for the mean of that trace, i.e. $\hat{x}_{zm} = (\widetilde{x} - |\widetilde{x}|_{TrS}) - (\overline{x} - |\overline{x}|_{TrS})$, where the mean is taken over each continuous tracking series. The resulting distribution is shown in Fig. 3.14. By definition, the distribution in this case has a zero mean. Both, the distribution of the offset in the longitudinal direction and the lateral direction, are more sharply peaked around zero and exhibit a smaller standard deviation of $\sigma = 1.333\,\text{m}$ in the longitudinal and $\sigma = 0.390\,\text{m}$ in the lateral direction, when computed in this way compared to the distributions without the mean correction shown in Fig. 3.13. However, the effect of the long tails still persists,

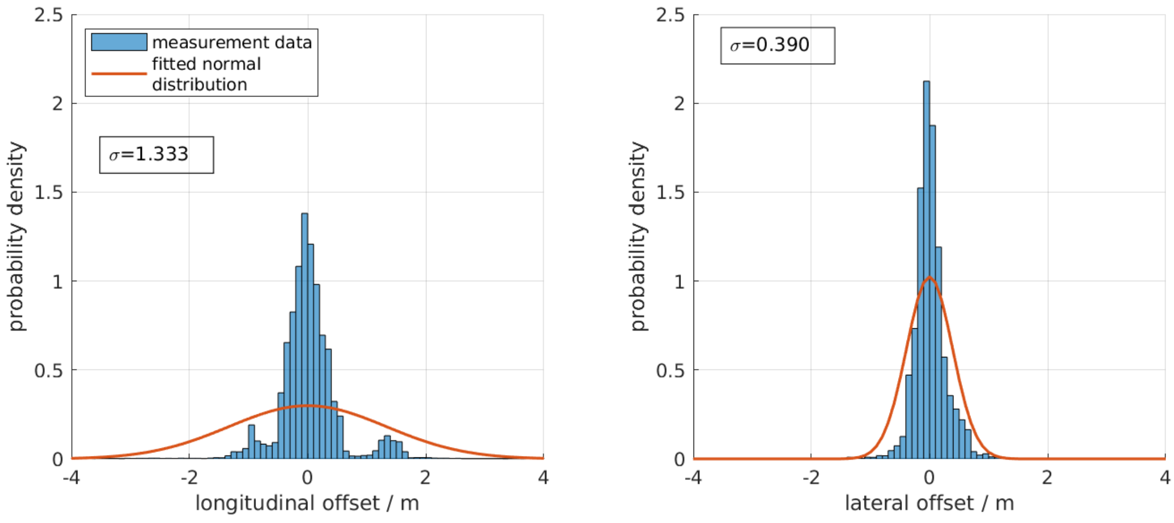


Figure 3.14: Probability density function for the offset between the longitudinal (left) and lateral (right) position measurement of the sensor setup and the position reported by the reference sensor for the same point in time. For each tracking series the mean of the sensor data and the reference data has been calculated and subtracted from the data in order to arrive at a zero mean distribution solely showing the random variation with separate handling of the mean values. To each measured distribution, a normal distribution is fitted (red line), whose standard deviation σ is given in the top left corner of each graph.

raising the question, whether these are a feature of these distributions and needs to be accounted for in a simulation model by using a non-gaussian distribution. As more extensive data are required for a conclusive investigation, this topic is not followed up within the scope of this work and left for future research.

Another interesting effect can be observed in the data, when considering the time evolution of the error distribution over the course of a tracking series. An a priori assumption would be that after the initial detection of an object with the accumulation of additional measurements over the course of a tracking series, the filtering applied by the tracking algorithms would lead to a decrease in the sensor error. Based on the available measurement data and the methods of analysis introduced so far, this hypothesis can be easily investigated. This is done by splitting the tracking series of objects at a certain point into a first an second segment and computing the error distributions separately for each of the segments. The results for such a split with the first segment consisting of the first 20% of the tracking series and the second segment consisting of the remaining 80% is shown in Fig. 3.15. Each measurement series is again corrected for the mean to focus on the error in the measurement at each sensor update. The split at a fixed percentage of the length of the tracking series has been chosen to allow for easier processing of drives with varying length, however splitting the drive after a fixed amount of measurements to separate the initial and latter part of a tracking series reveals largely the same results. As can be seen from the difference in the distribution between the first segment (top row in Fig. 3.15) and the second segment (bottom row), there is a dynamic behavior in the sensor error distributions.

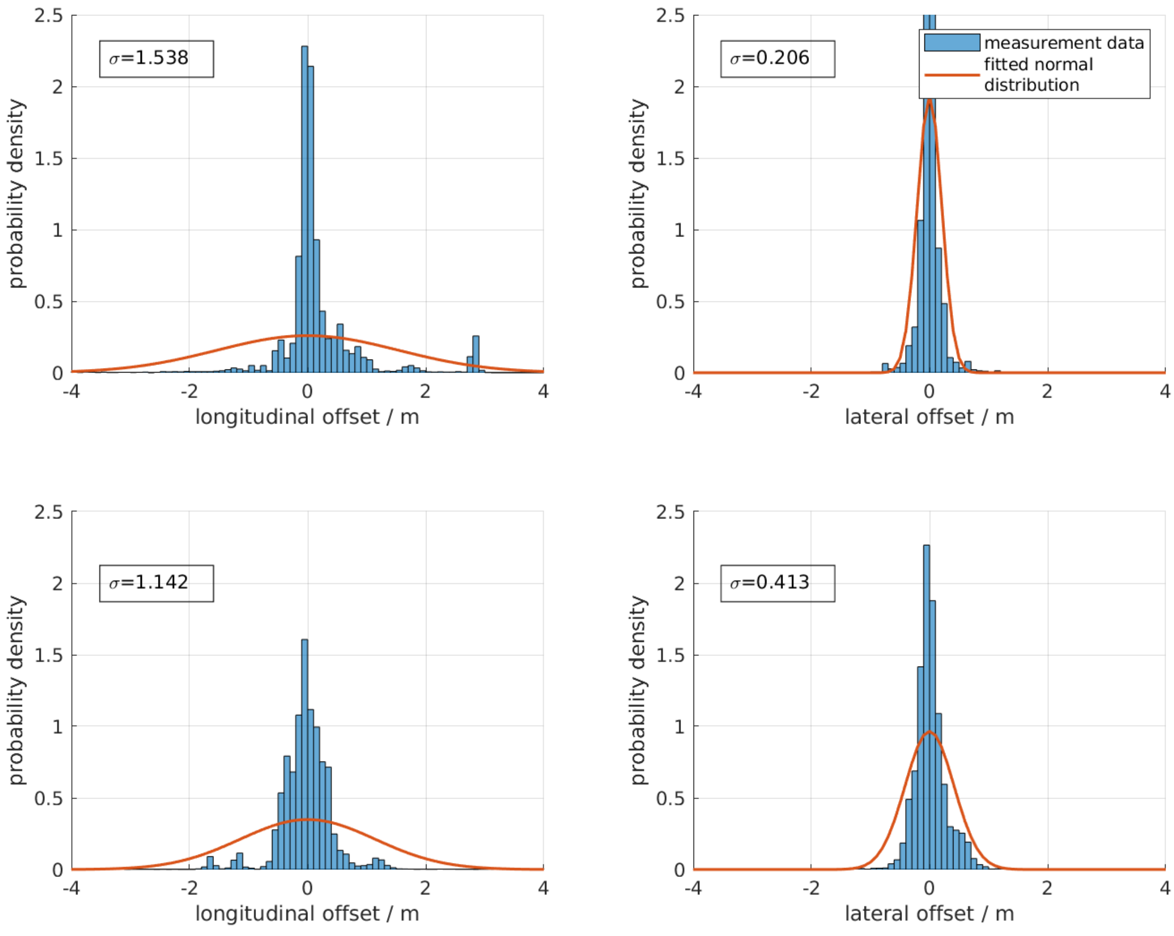


Figure 3.15: Probability density function for the offset between the longitudinal (left) and lateral (right) position measurement of the sensor setup and the position reported by the reference sensor for the same point in time. The tracking series for each object has been split into a segment containing the first 20% of the drive (top row) and another segment containing the remainder (bottom row). For each tracking series the mean of the sensor data and the reference data has been calculated and subtracted from the data in order to arrive at a zero mean distribution solely showing the random variation with separate handling of the mean values. The standard deviation σ to each distribution is given in the top left corner of each graph with the corresponding normal distribution shown in red.

For the longitudinal distribution, the width of the central peak around 0 increases while the standard deviation σ decreases, which is attributed to a decrease in the likelihood of large offset represented by the distributions long tails. From the data, these long tails are more pronounced during the first segment, matching the expectation that accuracy is lowest during initial object detection and increases due to the effects of the tracking algorithms. However, for the longitudinal offset, where the long tails are not as pronounced, the widening of the distribution causes an increase in the standard deviation.

From the point of view of modular sensor error model construction, each of these effects can be captured within its own module. A model can use a sequence of modules separately treating the distribution in the mean offset during a tracking series by randomly selecting an offset once for each tracked object, then the random error for individual measurements can be applied by using the a distribution with parameters obtained from the mean corrected measurements, and finally even the dynamic quality can be accounted for by introducing e.g. a module that applies a random distribution with a parameter set depending on how long an object has already been tracked.

3.4.2.2 Velocity Measurement

A similar analysis to the position measurements can be performed for the measurement of the object's velocity. In contrast to the position measurements, for the velocity, it is better to use the absolute values instead of the relative velocity to the ego vehicle. The direction of the relative velocity vector may easily experience large changes simply due to changes in the speed of the ego vehicle and the computation of a movement model for object tracking is more readily performed using absolute velocities. For these reasons, the absolute velocity of the target objects is better suited to data analysis and model building.

A set of sample traces for the absolute velocity is shown in Fig. 3.16. In the figure, the current velocity of the target object is represented by a vector arrow with its base at the reported position of the target. The coordinate system for both position and velocity vectors is defined by the position and orientation of the ego vehicle, with the ego vehicle's forward driving direction defining the longitudinal direction (towards the right in the figure). Both vehicles move in the same direction, indicated by the general orientation of the target vehicle's velocity vectors in the longitudinal direction of the ego vehicle. Both, reference (green) and sensor (black) measurements, are shown in the figure. From the figure, it can be discovered that both the orientation and the magnitude of the velocity vectors can differ between the reference and sensor measurements. For the same set of test drives used during the analysis of the position measurements, a similar analysis is performed for the velocity vectors. The probability density function for the error in the measurements of the components of the absolute velocity vector is defined as $p(\hat{v}_x)$ and $p(\hat{v}_y)$, with $\hat{v}_x = \widetilde{v}_x - \overline{v}_x$ being the longitudinal component of the difference between the velocity detected by the sensor $\widetilde{\mathbf{v}}$ and the velocity reported by the reference system $\overline{\mathbf{v}}$, and $\hat{v}_y = \widetilde{v}_y - \overline{v}_y$ being the lateral component. A plot of this function as computed from the measurement data is shown in Fig. 3.17.

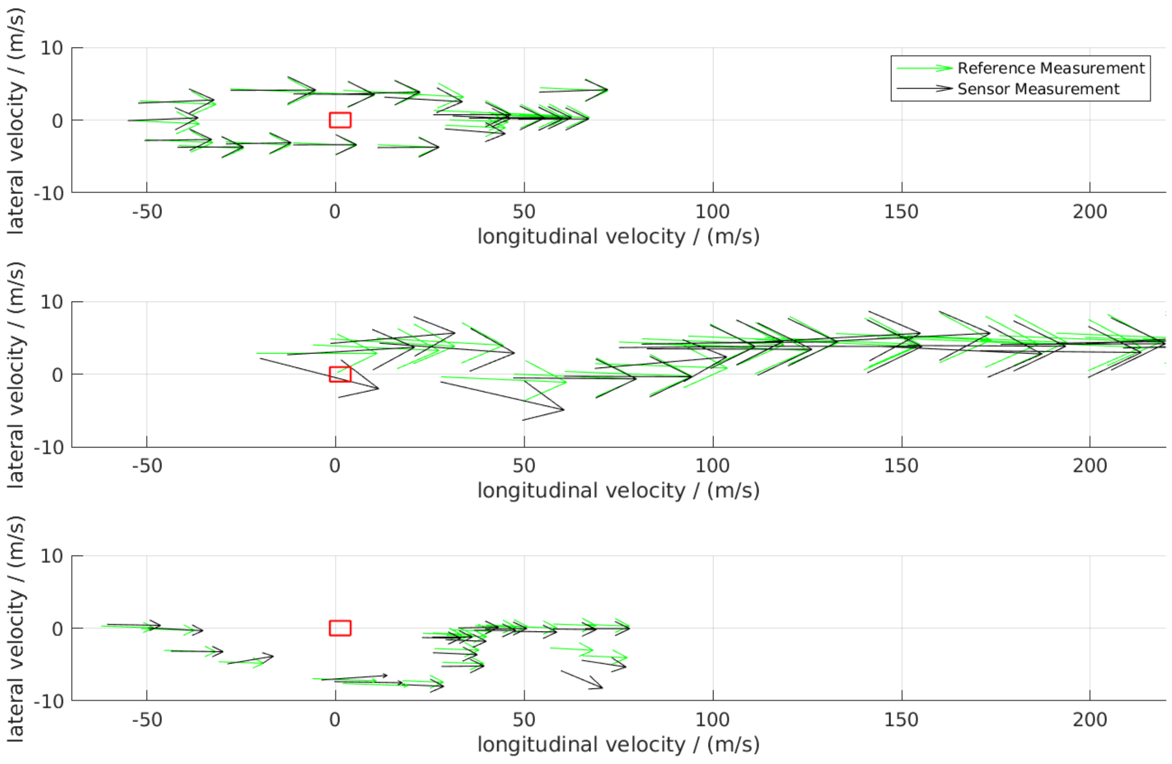


Figure 3.16: Set of traces of test drive data showing the absolute velocity of a target vehicle from sensor and reference measurements. Reference measurements for the velocity are shown as green vector arrows, sensor measurements shown as black vector arrows. The base of each vector arrow gives the position of the target vehicle at the moment of measurement as reported by either the reference system or the sensor. The position of the ego vehicle in the coordinate origin is shown by a red outline. Only a small subset of measurements,equally spaced in time, for each trace is shown to improve readability.

As can be seen in Fig. 3.17, the components of the velocity vector show the reverse characteristic compared to the position measurements. For the velocity, the lateral component (right in the figure) shows a very wide distribution with a standard deviation of $\sigma = 2.415\,\mathrm{m\,s^{-1}}$ indicating a low accuracy, while the longitudinal component(left in the figure) shows a much narrower peaked distribution with a standard deviation of $\sigma = 0.486\,\mathrm{m\,s^{-1}}$. Comparing the shapes of the distributions, the longitudinal velocity error shows an asymmetric shape with the peak towards positive error values, i.e. overestimating the speed of the target, while the lateral velocity error shows a largely symmetric shape. It is striking that the lateral component does not show a distribution centered around zero, with a mean of $\mu = 0.718\,\mathrm{m\,s^{-1}}$. This can likely be attributed to the target vehicle not driving equally on both sides of the ego vehicle and in this way creating a bias in the distribution. However, this discovery reveals an important characteristics that can be captured as part of a sensor model: the lateral velocity error seems to depend on the relative position of the target vehicle.

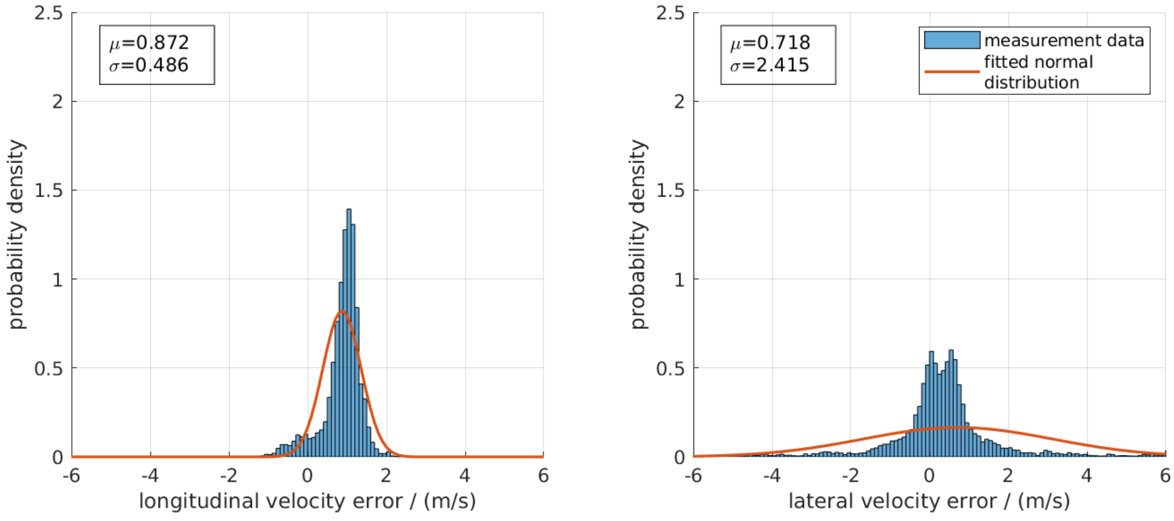


Figure 3.17: Probability density function for the error in the longitudinal (left) and lateral (right) velocity measurement of the lidar sensor compared to the velocity reported by the reference sensor for the same point in time. For each tracking series the mean of the sensor data and the reference data has been calculated and subtracted from the data in order to arrive at a zero mean distribution solely showing the random variation with separate handling of the mean values. To each measured distribution, a normal distribution is fitted (red line), whose standard deviation σ is given in the top left corner of each graph.

Focusing on the asymmetric shape of the longitudinal velocity error in Fig. 3.17 and the clear bias towards positive error values with the sample mean of $\mu = 0.872\,\mathrm{m\,s^{-1}}$, the question arises if this behavior is consisting over a number of object tracking series. This question can most easily be answered by computing the distribution of mean errors, where in analogy to the corresponding analysis in the position measurements the mean is computed over each continuous tracking series. This distribution of mean errors is

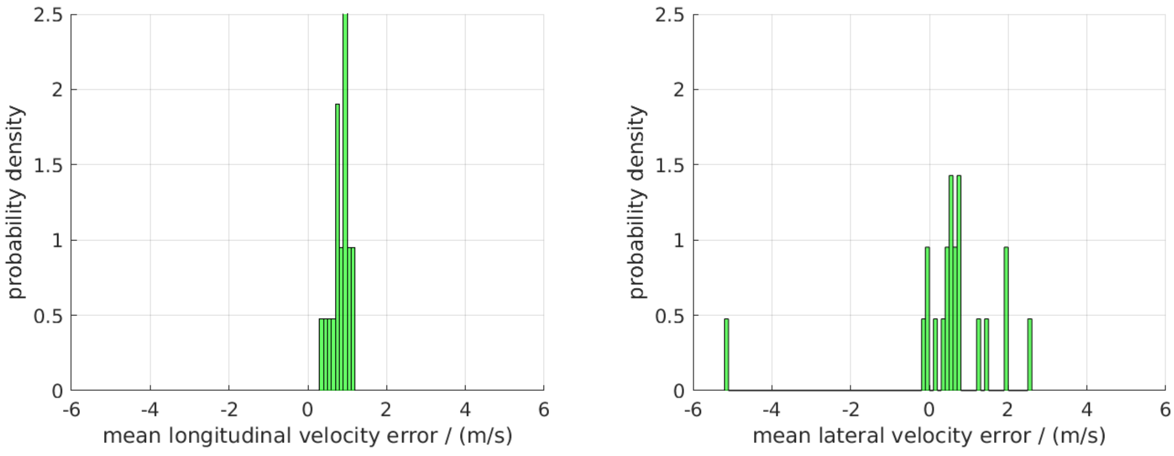


Figure 3.18: Distribution of the mean error in the measurement of the longitudinal (left) and lateral (right) velocity compared to the reference.

shown in Fig. 3.18. It can be clearly seen from the narrow distribution of the mean longitudinal velocity errors (left in the figure) that there is a consistent bias for each individual tracking series towards a positive error, overestimating the target's speed in the direction of the ego vehicle. This is a property that can easily incorporated into a sensor error model. The distribution of mean lateral velocity errors however shows no clear trend with a wide distribution of values.

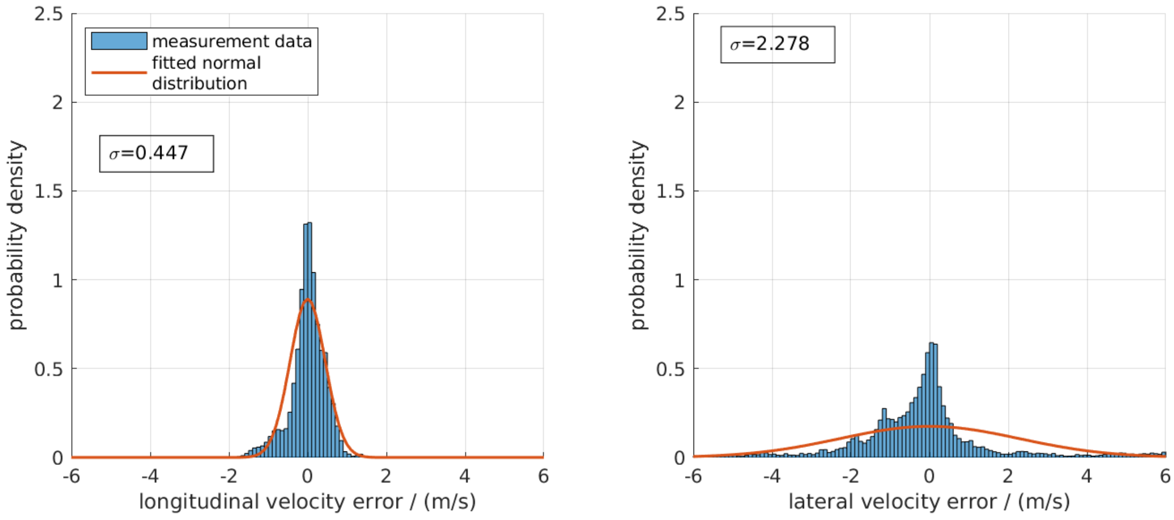


Figure 3.19: Probability density function for the error in the longitudinal (left) and lateral (right) velocity measurement of the lidar sensor compared to the velocity reported by the reference sensor for the same point in time. To each measured distribution, a normal distribution is fitted (red line). The resulting fit parameters for the mean μ and standard deviation σ computed from the measurement data are given in the top left corner of each graph.

As a next step, the distribution of longitudinal and lateral velocity errors is corrected by the mean for each continuous tracking series, in order to differentiate between systematic effects affecting the mean and the random variations for individual measurements. The result of computing the probability density functions corrected in this way to zero mean distributions is shown in Fig. 3.19. As expected by the narrow distribution of mean values for the longitudinal velocity component (Fig. 3.18), this operation for the most part only shifts the distribution to a zero mean, yet has hardly any influence on the standard deviation, only slightly changing it from $\sigma = 0.486\,\mathrm{m\,s^{-1}}$ to $\sigma = 0.447\,\mathrm{m\,s^{-1}}$, or the shape of the distribution. The distribution keeps its slightly asymmetric shape with the peak shifted towards larger values, indicating that this behavior is a general feature of the longitudinal velocity measurement for this type of sensor. In contrast, the distribution for the lateral velocity error has changed its shaped with this operation from the largely symmetric shape in Fig. 3.17 to an asymmetric one. Given that the mean values for the lateral velocity component for most tracking series are offset to the same side, this is likely an indication that the lateral velocity measurement exhibits a similar behavior as the longitudinal component, with an asymmetric error distribution that favors a direction that is likely dependent on the relative position of the target vehicle. The last statement is an assumption that unfortunately cannot be proven from these data, yet warrants further analysis with larger data set.

Equipped with the knowledge gathered from the analysis of the data for the position and velocity measurements, it is now possible to construct an exemplary module for a sensor error model that can be parametrized with the parameters obtained during the analysis of this section.

3.4.2.3 Model Implementation

We construct a stochastic model for errors in the position measurement as an exemplary implementation of the architecture shown in Fig. 3.5. This model consists of one module that varies the two-dimensional positions, given by coordinates x_j and y_j for the object with ID j. The variation is introduced by a random Gaussian white noise offset Δx_j, Δy_j that has zero mean and is uncorrelated in time and space. The constitutive equations (3.3) of the model then become:

$$M(\mathbf{p}, \mathbf{c}) = M^{(g)}(\gamma_g), \tag{3.5a}$$

$$M^{(g)}(\gamma_g) : (x_j^{(0)}, y_j^{(0)}) \to (x_j^{(0)} + \Delta x_j, y_j^{(0)} + \Delta y_j) \ \forall j, \tag{3.5b}$$

where $(x_j^{(0)}, y_j^{(0)})$ denotes the ground truth position. The module configuration is given by the standard deviations of the noise distributions: $\gamma_g = \{\sigma_x, \sigma_y\}$, assuming that σ_x and σ_y are not functions of the relative position. Figure 3.20 shows a sample trajectory obtained using the model described in (3.5). In general, values for the parameters σ_x, σ_y can be determined using test drive data, where high-precision reference sensors in the ego and target vehicles provide the ground truth data in addition to the relative positions reported by the sensor under consideration.

3.4.3 Indirect Perception

Indirect perception comprises those quantities that have no straightforward relation to directly measured sensor data. The calculation of these quantities makes use of complex tracking techniques involving probabilistic hypotheses created, validated or invalidated taking into account a larger number of previous measurements. Examples are object classification or the mapping of a road user to a specific driving lane. Typically, the output of indirect perception is one value out of a discrete set of available states.

In the framework of the statistical sensor model, we treat indirect perception as a probabilistic state transition process based on predefined trigger conditions and variable transition rates. This is expressed in the probabilistic mapping $M^{(k)}_{indirect}(\Psi^{*(k)}_{t-1}) : \Psi_t \to (\widetilde{\Psi}^{(k)}_t, \widetilde{\Psi}^{*(k)}_t)$. The output consists of the subset of the sensor output interface $\widetilde{\Psi}^{(k)}_t$ for which the model unit is responsible as well as additional internal state data denoted as $\widetilde{\Psi}^{*(k)}_t$ that monitors the status of the trigger conditions.

An example of a trigger condition is the relation between the estimated bounding box and the object classification. Assuming for example a lidar sensor initially can

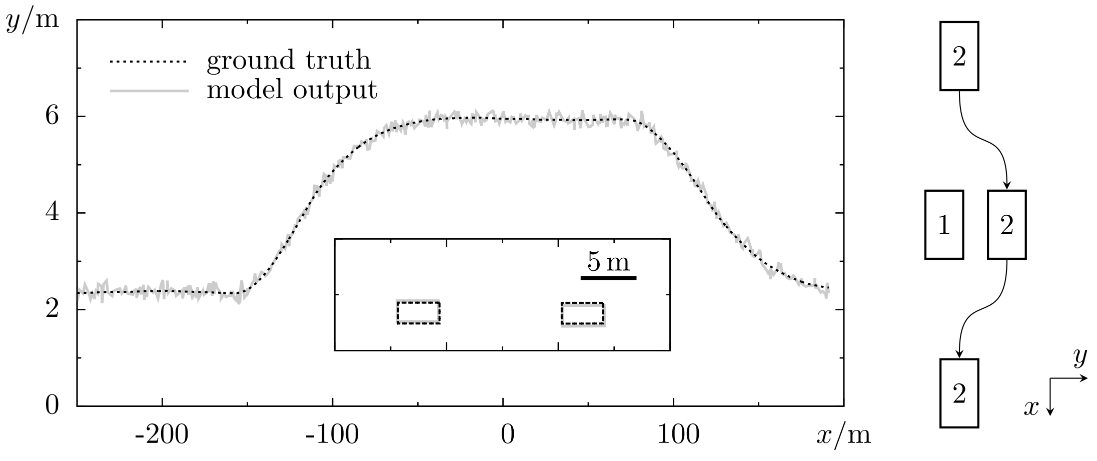


Figure 3.20: Comparison of simulated ground truth trajectory (dashed black line) and corresponding model output (solid gray line) with a Gaussian white noise applied to the vehicle's x and y coordinates. The vehicle is moving with speed $v \approx 70\,\mathrm{km/h}$. The update rate is 20 Hz and $\sigma_x = \sigma_y = 0.1\,\mathrm{m}$. A sketch of the scene (not to scale) is shown on the right: vehicle 2 approaches from behind and passes by the ego vehicle (labeled 1) statically positioned at the origin. To avoid the ego vehicle that is partly blocking its path, it initiates a lane change at $x \approx -150\,\mathrm{m}$. The inset shows the bounding box of vehicle 2 at ground truth position and after variation at subsequent points in time spaced one second apart.

only properly estimate the width but not length of an object due to the geometric configuration of the scene and further is not able to properly classify the object; if a change in the object classification output is usually accompanied by a change in the length estimation, this correlation can be exploited in the model. The change in the bounding box estimation constitutes the trigger condition for the change in the classification. Note that causation between trigger condition and effect is neither implied nor required, only correlation.

Indirect perception is further subdivided depending on whether the inference of a specific object property is restricted to that specific object or an advanced understanding of the scene is required. The former are termed indirectly inferred quantities. These are characteristics of the specific object inferred based on probabilistic estimation. Examples are the classification of a dynamic object or estimation of the motion state, i.e. moving, parking, or static. The latter require a more comprehensive understanding of the scene beyond the specific object and are termed derived qualifiers. An examples is the mapping of a (dynamic) object to a driving lane requiring information about the object position as well as information on position and course of driving lanes.

As an example, we consider the classification of dynamic objects, specifically the transition of an unclassified object to being classified as a car. Fig. 3.21 shows two exemplary traces where the length or width estimation of a lidar sensor correlate with the object classification as determined by the sensor's perception function. The five points in time shown in Fig. 3.21 (a) to (e) have an offset of 5 frames each, corresponding to approximately 0.2 s at the sensor update frequency of 25 Hz. In both cases, classification

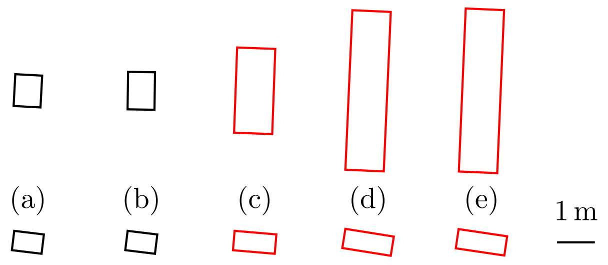


Figure 3.21: Time traces showing a correlation between width and length estimation and the object classification of a lidar sensor. The upper part shows the length estimation for a vehicle passing the ego vehicle on the left and the lower part shows the width estimation for a vehicle about 90 m in front of the ego vehicle as a larger part of the target object becomes visible to the sensor. Size and orientation data are taken directly from sensor output (scale bar for 1 m in the lower right corner). Positions have been set solely for illustration purposes and are not to scale.

changes at (c) from unclassified (black) to car (red). In these traces, this corresponds to the point in time where for the first time estimated length surpasses 2.0 m (upper trace) or estimated width surpasses 1.0 m (lower trace). These two conditions can be defined as trigger conditions, such that when either length or width estimation surpasses the threshold value for the first time, a change in classification is predicted by the model to occur immediately or after a small time lag. A first evaluation of a data set containing about 1400 of these classification transitions is shown in Fig. 3.22, where the number of observed transitions after a specific number of sensor updates after the trigger condition has matched is plotted. The graph shows that based on these very simple trigger conditions a large number of the unclassified to car transitions can be correctly predicted. However, when evaluating the incorrectly predicted transitions, it turns out that these occur on a rate of approximately one time for every three correct predictions. This very simple model therefore does not score particularly high on comprehensiveness or specificity, yet provides a sound starting point for refinement of the trigger conditions.

3.4.4 Packaging

During packaging, i.e. encoding and transmitting of the sensor output interface, we differentiate between two types of error-like effects. Those concerning the content of the interface, but not specific to any object, and those that impact the interface and data transmission on a general level. Examples of the former are incorrect or varying timestamps of the data, limitations of the interface specifications, e.g. a maximum number of objects reported by the sensor due to restrictions on message size, or the absence of any data due to the sensor being temporarily in a blocked, startup, or calibration state. The latter deal mostly with transmission timing. Varying processing times of the perception functions usually result in irregularly timed update intervals. As timing of input data is relevant for the subsequent functional processing chain of the automated driving system, the sensor model has to account for these effects for a

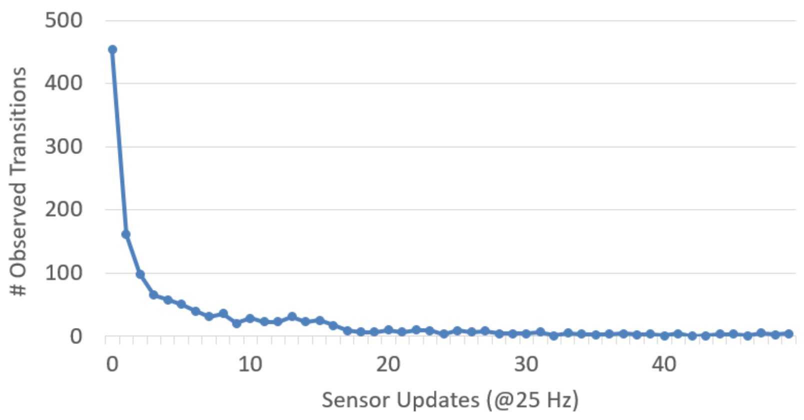


Figure 3.22: Chart of the number of transitions from unclassified object to object being classified as a car as a function of the number of sensor updates after a trigger conditions using the sensor output for the object's width and length has matched.

correct emulation of sensor behavior. Correct handling of timing is largely a task for the underlying simulation framework that coordinates the transmission of output data of the sensor model.

3.5 Summary

In this chapter, sensor error models are introduced that aim to reproduce the characteristics of sensors in a virtual environment on a statistical level. A sensor error is defined as any difference between the sensor output and the original ground truth data arising during the perception process, giving the sensor error model its name. Due to the nature of a statistical model, it requires statistics obtained from observation data in order to derive the appropriate parameters required for the simulation of sensor behavior with regard to any quantity observed by the sensor. Obtaining these data requires reference measurements for each quantity that is being investigated. A generic modular approach for the design and construction of sensor error models is introduced. The modular approach enables the building of individual modules that are executed in sequence, allowing each module to handle a different aspect of the sensing process. Moreover, this design enables an iterative development process for a sensor model, where each iteration can add an additional feature in the form of an extra module handling a specific aspect that has been analyzed and replicated in the framework of the model.

For the construction of a sensor error model using this modular approach, reference data are required. The real world setup for gathering reference data is introduced as well as the virtual world setup for implementing a sensor error model. Based on the data gathered from the real world test drives, an in-depth analysis is performed for a number of object properties detected by the sensor setup under investigation, a setup consisting of four IBEO Lux lidar sensors. Properties for the model are extracted from the data analysis that can be used for the parametrization of a sensor error model.

As an outlook, further additions to the sophistication of the sensor error model may be achieved by using machine learning techniques like a generative model using neural networks to replicate characteristic output based on learned behavior from test drive data. As the main design feature of the modular approach to model building introduced in this work is its flexibility, it is a straightforward extension to include a model unit that makes use of e.g. a neural network. An interesting and very recent approach to generate sensor data from mobile devices using adversary neural networks can be found in [62]. Adopting this approach for automotive sensors may prove worthwhile for future investigations.

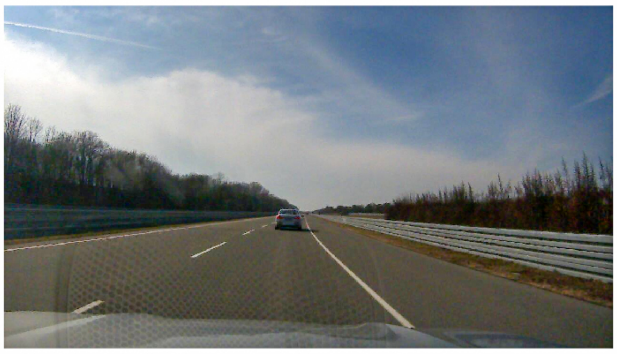
4 Sensor Measurement Model

As the second of the general two types of sensor models from the introductory overview given in section 2.4, sensor measurement models aim for a physically motivated bottom-up approach for the simulation of the sensor measurement process. These types of models allow a high level of flexibility as they are not intrinsically reliant on reference measurements from an existing sensor and can even be used to evaluate the effect of changing some of the sensor properties to create a sensor not (yet) existing in a real world setup. However, due to their by far increased computational complexity, they require special optimized interfaces from the driving simulation framework, limiting their scope of applications.

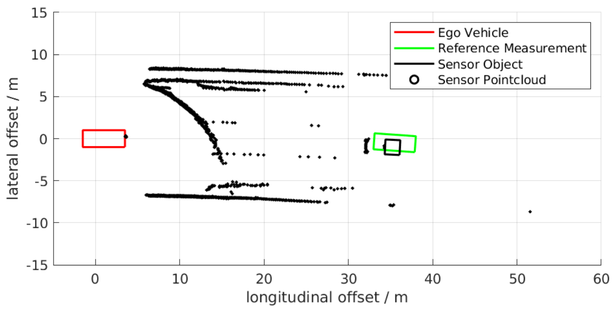
This chapter discusses first the scope of sensor measurement models and their typical applications, followed by the discussion of the physical simulation of the sensor measurement process for a lidar sensor. A ray tracing based sensor measurement model for the simulation of lidar point cloud data is introduced, based on real world observations. Finally, a direct application for the low level data output of the sensor measurement model is discussed, where simulated low level data are used to stimulate in real time the input for a higher level perception function based on an occupancy grid implementation. Using the occupancy grid representation, a side-by-side comparison between the real and virtual world sensor data generation and usage is performed.

4.1 Scope and Applications of Sensor Measurement Models

A sensor measurement model is a physically motivated model of the sensor measurement process that aims for a high degree of realism in the generation of low level sensor output data. The purpose of this type of sensor model is to serve as the first step in the perception chain, linking the geometry and properties of the three-dimensional virtual world with the sensor data processing of the perception system under test. Low level sensor data are largely unprocessed data directly derived from the sensor measurements. In the case of automotive lidar sensors, which are the focus of this work, these data typically have the form of a lidar point cloud, i.e. a set of points in three-dimensional space that outline the reflecting obstacles encountered by the sensor's measuring laser beam. An example of such data is shown in Fig. 4.1. On the left side, Fig. 4.1 (a) provides a visual reference for the scene, which consists of two vehicles on a closed test track. On the right side, Fig. 4.1 (b) shows the point cloud output from the front lidar sensor of the ego vehicle as well as the output of object tracking and reference measurements.



(a)



(b)

Figure 4.1: Real world example for low level sensor data from a lidar sensor that a sensor measurement model aims to simulate. On the left side, (a) shows a still frame recorded by a camera mounted on the ego vehicle's dashboard that shows a target vehicle to the front of the ego vehicle. On the right side, (b) shows data gathered at this point in time: The black dots show a top view of the point cloud data output from the front lidar sensor of the ego vehicle, the black outline shows the output of the object detection. The green outline shows the reference measurement for the target vehicle's position. The ego vehicle is shown in red, with the center point of it's rear axis defining the origin of the relative coordinate system.

Due to their ability to generate synthetic low level data, sensor measurement models are used for applications that require a high degree of realism and flexibility. This comes at a significant computational cost, but enables two areas of applications that are largely inaccessible to other types of sensor models: One, the simulation-based study of sensor types and configurations even if the sensor types themselves or vehicles with the specific configuration are not (yet) available. Two, the individual components in the perception chain, i.e. measurement and perception processing, can be individually studied using the low level data interface of the sensor measurement model as the connecting element. To this end, it is possible to use software or even hardware components in the perception chain that make direct use of the low level sensor data. Using a sensor measurement model, it is therefore possible to build a simulation setup where the only difference to real world testing setups is the substitution of the measurement apparatus of the sensor by a simulation model of the physical processes within the sensor.

Comparing sensor measurement models to the sensor error models discussed in chapter 3, the main difference between the two types of sensor models is that applications using sensor measurement models aim for a detailed and exact replication of the individual steps of the perception chain whereas applications using sensor error models aim for a unified single model of the complete perception chain that replicates the final output on a statistical level. By limiting the scope of the sensor model to the physical measurement process, the model can be used in a variety of use cases where a detailed view of the perception process is aimed for. Among these use cases is the study of the effect of the vehicle's sensor setup, i.e. number and location of sensors within the vehicle, as the position of the sensors can be easily modified when a physical measurement model is used. Further, sensor measurement models enable any application where raw data or low level sensor output is required, e.g. when investigating the behavior of a specific sensor fusion or object detection algorithm in the perception stage. One such sensor fusion algorithm, the occupancy grid, is briefly discussed later in section 4.4 as an example for a consumer of the low level sensor data generated by the sensor measurement model discussed in this work.

The capabilities of sensor measurement models to enable additional applications due to their flexibility, however, come with the cost of requiring more detailed input data when building and validating the model. As the model's output includes measurement data from the sensor, the same type of data from real world sensors has to be used in order to define and test the model and the model's parameters. This type of low level sensor data in the required quality is in many cases not easily available, especially as appropriate ground truth data are required to make use of the real world measurements of a sensor under test. However, ground truth data in the quality required for e.g. lidar measurements are very hard to come by. To some degree this limits the ability to validate these types of models. Nevertheless, when deriving a sensor measurement model from basic physical principles as well as available measurements, a model can be constructed that is useful for a variety of applications. In contrast, for the sensor error models of chapter 3, it is easier to come by valid measurements as the object level interfaces that are employed by these models are much more commonly used in current systems and therefore easy to record. A further difference between the two types of models is their computational performance: The statistical high level approach of the sensor error model can achieve fast computational performance and therefore lends itself to the simulation of a large number of scenarios. This is in contrast to the detailed approach of the sensor measurement model that has high computational requirements, e.g. for ray tracing computations, and is more readily suited to fit small scale simulations with a limited scope, e.g. for research into optimal sensor placements or sophisticated low level based sensor fusion algorithms.

4.2 Physical Simulation of the Sensor Measurement Process

This section describes the integration of the sensor measurement model in the virtual perception chain. The model represents the first step of environmental perception building directly on the three-dimensional virtual world without an abstract functional world representation, e.g. an object level representation, as an intermediary. The interaction between the sensor and the virtual world is captured within the model using a ray tracing approach.

4.2.1 Processing Chain for the Low Level Data Model

The perception process in the simulation of an automated driving system can be split into individual, sequential components. The resulting perception chain was initially introduced in chapter 2. The perception chain starts with the virtual environment on the input side and culminates on the output side in the internal environmental description that is used by the decision making algorithms of the automated driving system, i.e. the automated driving feature. Within the context of the perception chain, a sensor measurement model constitutes the initial link between the environment of the virtual world and the first low level representation of the virtual environment This low level representation is directly derived from the sensor measurement process and used for further processing in the perception chain of the automated driving system.

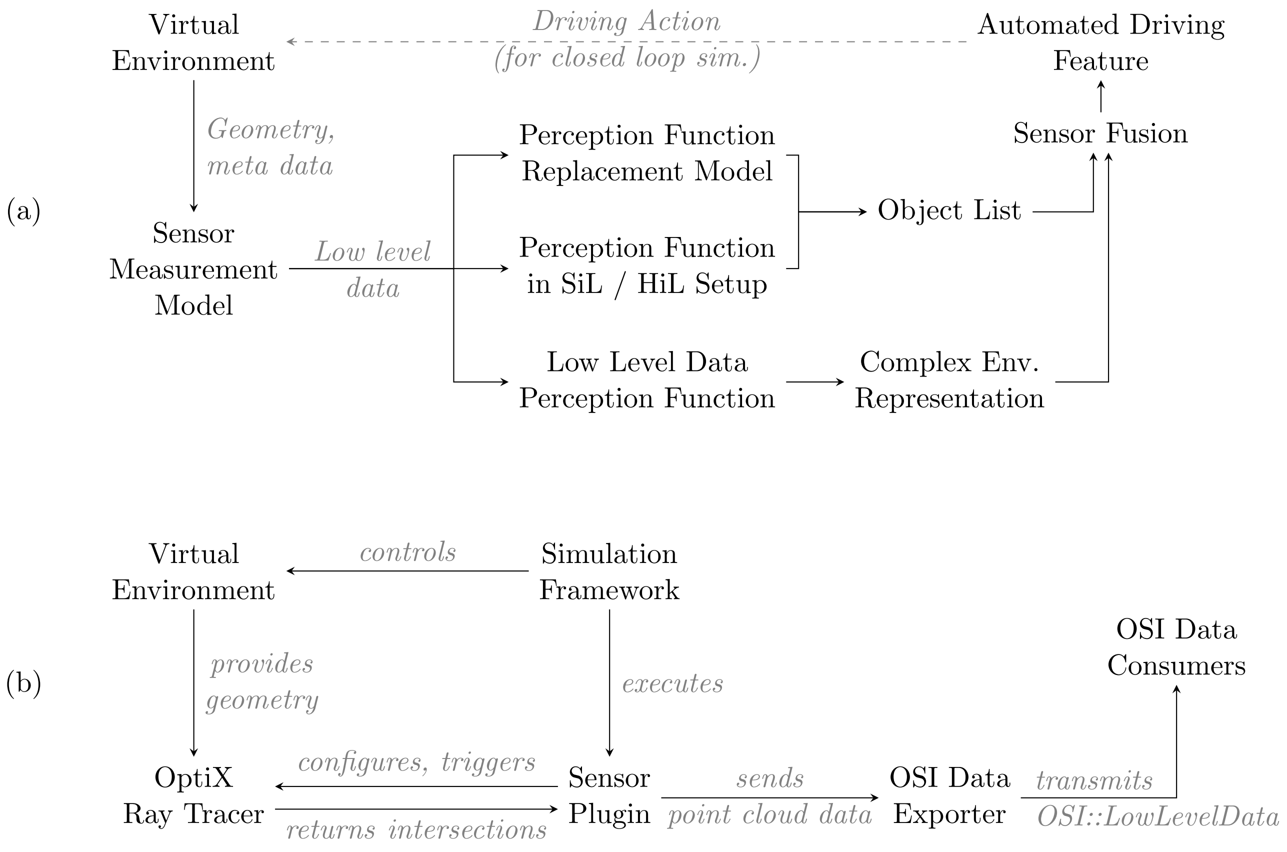


Figure 4.2: Sensor measurement model in the context of the simulation setup. In the top figure (a), the variations of the processing chain for perception using a sensor measurement model is shown: The low level data output from the sensor measurement model enables the use of a real world sensor's perception function for object detection within a SiL or HiL Setup, or the use of a non-object based perception function operating directly on the low level sensor data. The bottom figure (b) shows the technical components and interfaces of the sensor measurement model: The model is implemented as part of a sensor plugin executed by the simulation framework. The plugin code makes use of the OptiX ray tracing engine to determine intersection points within the geometry provided by the virtual environment, and processes these data into a point cloud output. An OSI data exporter converts the data into the OSI::LowLevelData format and offers an interface to data consumers.

Based on this understanding of the purpose of a sensor measurement model, the placement of the model is shown in Fig. 4.2 (a) in the context of the simulation setup as a whole. The sensor measurement model receives geometry and meta data defining the virtual environment as input and returns sensor specific low level data as output. The next component in the perception chain can take one of three forms, depending on the goals of the simulation setup. If the goal of the simulation setup is served by re-creating the output of an existing real world sensor on the level of an object list, then either a replacement model of the perception function, i.e. object detection, of this sensor is required or the identical code of the sensor's perception function can be used. The latter case is used in either a software-in-the-loop setup (SiL) or hardware-in-the-loop setup (HiL), which also allows the testing of the perception function behavior itself and constitutes a valid use case for simulations setups making use of a sensor measurement model. If the goal of the simulation setup is different, however, and does not make use of an existing real world sensor or the object list representation, the low level data output of the sensor measurement model can be used as input for a custom low level data perception function that constructs a complex environment representation for the use e.g. in sophisticated raw data based sensor fusion algorithms. In either case, if it is desired to build a closed loop simulation, the output of sensor measurement models and perception functions for multiple sensors is fused and can in this way serve as input for the automated driving feature. The automated driving feature in turn can report the driving actions taken by the ego vehicle back to the simulation framework for the next update step in the closed loop simulation.

For automotive lidar sensors as the focus of this work, the low level representation of sensor data usually has the form of a point cloud that consists of individual reflection points in three-dimensional space. For each of the points, a number of associated properties specific to the measurement process of the sensor is stored in addition to the position measurement of the reflecting object.

Within the scope of this work, a ray tracing approach is used for the purpose of constructing a sensor measurement model for an automotive lidar sensor. Ray tracing is a flexible simulation approach that has originally been introduced in the context of computer graphics in the 1980s with the goal of generating high quality computer rendered images [63, 64, 65]. Ray tracing is well suited for the purpose of simulating the propagation and scattering of electro-magnetic waves and can easily be employed for simulation on various levels of complexity, examples can be found in references [66, 67, 68, 69, 70]. The components used in the implementation of the model are shown in Fig. 4.2 (b): The simulation framework controls the virtual environment and executes the sensor plugin, which contains all the custom code for the implementation of the sensor measurement model. The sensor plugin configures the ray tracing engine, i.e. defines ray parameters like starting points, directions, and hit functions, and triggers the sending of rays on each sensor update. The ray tracing engine uses the geometry data provided by the virtual environment to calculate intersections between rays and surfaces in the three-dimensional virtual world, which are returned to the sensor plugin for further processing. The sensor plugin uses these data to derive low level sensor data in the form of a point cloud for the simulation of a lidar sensor. These data are sent to the OSI data exporter, which transforms them into the format of the OSI::LowLevelData interface, introduced in section 2.5 on the open simulation interface (OSI). The OSI data are then transmitted to the consumers further processing, which is usually performed by a perception function that uses low level sensor data as input. For the definition of the model within the sensor plugin, using the basic assumption that each ray in the model adheres to the same physical behavior, it is the task of the model to properly capture the interaction of one ray with its environment. With this assumption the number and direction of rays can be set and enhanced arbitrarily, which allows for a very high flexibility in the application of the model. In particular, this allows to approximate the behavior of many different real world sensors with basic information from their specification sheet alone as well as allowing a largely free placement of the sensor within the vehicle.

For the construction of the sensor measurement model, the scanning type of lidar sensor is considered that is typically used for automotive applications. This type of sensor determines distance by measuring the time of flight of a laser pulse that is sent by the sensor, reflected by a target surface, and then again detected by the sensor. Its angular resolution is achieved by means of scanning, i.e. by moving the transmitted laser beam as well as the selective field of view of the optical detector array successively over the sensor's complete field of view. Most commercially available systems at this time employ a mechanically rotating mirror for the scanning task. The operating principle of this type of sensor lends itself to a modeling approach using ray tracing techniques. The specific implementation chosen for the sensor measurement model discussed within this work is introduced in the following section.

4.2.2 Ray Tracing Simulation Framework

The virtual environment for the proposed sensor model is provided by the Vires VTD driving simulation software, which offers a ray tracing framework based on the Nvidia OptiX ray tracing engine as an optional extension. A detailed description of the ray tracing engine can be found in [71]. The use of this framework for the simulation of automotive radar sensors has recently been reported in [32, 33].

The purpose of the sensor measurement model is the mapping of the three-dimensional virtual world as input to sensor output data that should be largely indistinguishable from the data provided by a real sensor mounted on a vehicle. A detailed graph of the control and data flow in the model and virtual environment is shown in Fig. 4.3. As introduced in the previous section and the high level view of the model shown in Fig. 4.2, the model is implemented as a sensor plugin in the driving simulation framework. In a more detailed view, the sensor plugin contains several components as described in the following, whose relations are illustrated in Fig. 4.3. Generation of rays is handled by the virtual sensor's camera program. This encompasses the calculation of the ray starting points, their launch directions at the current simulation update step, and the appropriate configuration of the ray tracing engine. The camera program is invoked by the sensor plugin on each update frame of the sensor. After configuration is finished, the ray tracing engine is invoked by launching the rays. The ray tracing engine then performs a traversal of the node graph representing the virtual environment for each ray in order to determine the intersection points with the world's three-dimensional geometry. The specialized OptiX node graph used for traversal of rays is updated on each simulation update step by the virtual environment of the driving simulation framework. For each intersection found during node graph traversal, a material lookup is performed by the ray tracing engine. The material lookup is performed in a custom, purpose-built material database that defines the appropriate hit program and the relevant material properties. The relevant hit program is then invoked and given the position of the intersection point, the ray's properties, and the material properties as context parameters. The hit programs and material database both constitute parts of the sensor measurement model and are central to its implementation. The reflection model is defined in the hit program, which includes the calculation of the contribution of light sources, the definition and launching of any secondary rays if desired, and even determining whether an intersection needs to be included for further calculations at all or disregarded, e.g. in case of a transparent material. Finally, the hit programs modify the payload of the primary rays by saving the calculated reflection points and required context data and returning these to the camera program for post-processing. All of these steps, starting from ray generation in the camera program, are computed in parallel on the GPU for each ray and therefore for each point in the point cloud in order to achieve real-time performance. During post processing in the camera program the point cloud and associated quantities are calculated for export of data. After post processing, the camera program forwards the data to an OSI data exporter component for further handling.

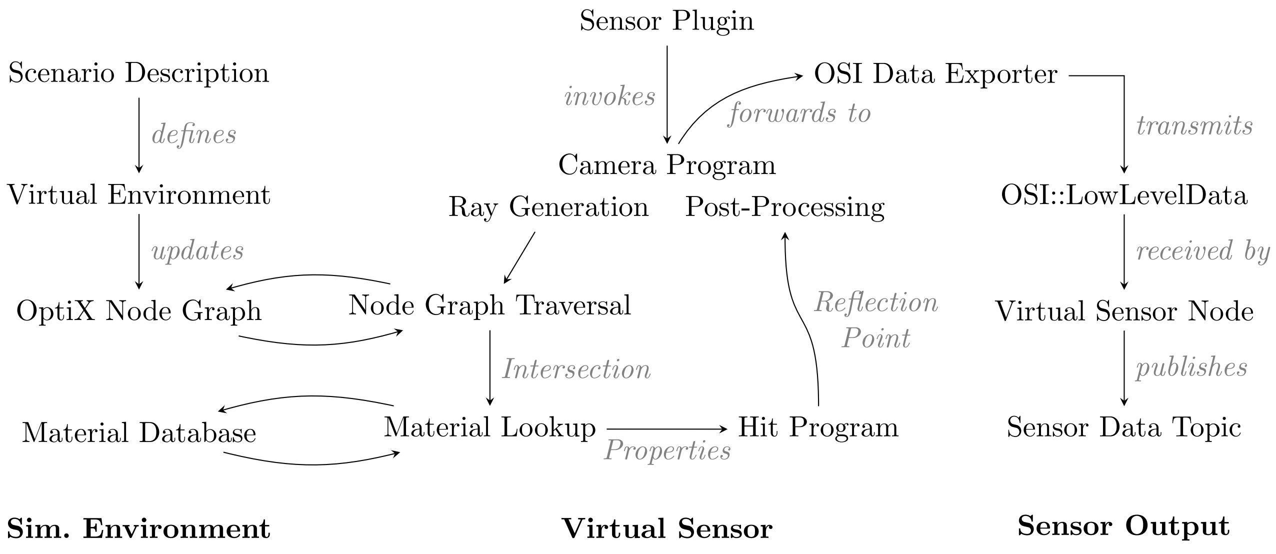


Figure 4.3: Data and control flow in the functional processing chain of the sensor model, split into the domains of the simulation environment, virtual sensor, and sensor output. The sensor model is defined by the components of the sensor plugin, specifically the camera and hit programs, as well as the purpose-built material database.

The OSI data exporter establishes a TCP connection to a pre-defined target address and transmits the data to the registered consumers listening on the target port. Before transmission, the OSI data exporter converts the data to an interface format specified by the OSI standard. For the serialization of the generated data and their transmission to the Robot Operating System (ROS) used as development framework on the output side, the point cloud representation in the OSI::LowLevelData message is used. The open simulation interface and its messages are described in [57] and have been introduced in section 2.5. The resulting data stream is received by a *virtual sensorics* ROS node and translated to ROS internal sensor data topics that publish the exact same message format as a corresponding sensor node for the lidar sensor in a real vehicle setup. As the sensor measurement model described in this chapter is capable of real time output of generated low level sensor data, it is at this point that ROS nodes that consume the sensor data output of the virtual sensorics node do not experience a difference in the data format arriving from a real or virtual world setup. This serves the original design goal of the sensor measurement model of producing synthetic sensor data largely indistinguishable from real world sensor data.

4.2.3 Virtual World Setup for Sensor Measurement Model

The virtual world setup for running the sensor measurement model is similar to the setup described in section 3.3.2 on the virtual world setup used for the sensor error model. However, for the purpose of running the sensor measurement model additional components are required. In general, the virtual world setup consists of three components: the driving simulation framework responsible for updating and controlling the virtual world, the framework running the sensor model, and the framework hosting the components that are using the generated sensor data, i.e. perception function and automated driving feature. For the purpose of this work, the following setup has been used for the most part for developing and running the sensor measurement model: As hardware a workstation with an Intel Xeon CPU as main processing unit is used, which has 8 logical cores running at 3.5 GHz. The machine further has 64 GB of main working memory and uses for graphics processing a single graphics card with an Nvidia Quadro M4000 GPU with 8 GB of graphics memory. As operating system an 64-bit Ubuntu Linux in version 16.04 LTS is installed. For the driving simulation framework, the software Vires Virtual Test Drive (VTD) in Version 2.1 is used.

Vires VTD offers with an additional extension the integration of a ray tracing framework into its graphics processing pipeline, which can be extended with custom code for the purpose of e.g. sensor simulation. As ray tracing engine Vires VTD makes use of the Nvidia OptiX ray tracing engine [71], which is used in version 4.0.2. The implementation of the sensor measurement model described in this work makes use of the interfaces offered by the Nvidia OptiX integration within Vires VTD, which sacrifices portability of the model but is necessary to achieve the required computational performance to make a real time ray tracing approach feasible. The OptiX integration within the graphics processing pipeline of the driving simulation framework takes care of the syncing of the simulation framework's internal scene graph as the representation of the three-dimensional geometry of the current scene and the OptiX node graph used by the ray tracer. The custom code for the implementation of the sensor measurement model can then be used to directly program the ray tracing engine that interacts with the node graph provided by the framework. This allows to lay the focus of development work on the architecture of the sensor measurement model and its inner workings while the framework handles the actual ray tracing and interaction with the scene. The sensor plugin developed within the scope of this work hosting the custom code for the sensor measurement model exports all data in the format of the Open Simulation Interface (OSI) described in section 2.5, and uses zeroMQ [72] as messaging library for communication with consumers. The version 2.0.0 of OSI and version 4.1 of zeroMQ is used. The consumer for the low level sensor data is a *virtual sensorics* node within the Robot Operating System (ROS), receiving messages in the OSI::LowLevelData format from the sensor measurement model and other required data from the simulation framework. The virtual sensorics node then transforms these input data into messages of equal format to an existing laser scanner system, which makes further handling of data indistinguishable between the simulation and a comparable real world setup. Further processing and usage of the data generated by the sensor measurement model is similar to the usage described in section 3.3.2 for the sensor error model. The main difference is the type of data received, low level sensor data compared to object level data, and therefore the consumers handling further processing of the data within the ROS framework are different. However, the processing of generated sensor data is the focus of this work and therefore the further usage of data within the ROS framework is not discussed in detail. An exemplary use case will be briefly introduced in section 4.4, describing an occupancy grid implementation making use of the synthetic low level sensor data generated by the sensor measurement model.

To measure the performance of the sensor plugin implementing the sensor measurement model, the following setup is investigated: The sensor model is configured to generate point cloud output data matching the specification of the IBEO Lux laser-scanner sensor, with the simulated sensor located on the front of the ego vehicle. This sensor covers a horizontal field of view of 110° and a vertical field of view of 3.2°. In the horizontal direction, the resolution, i.e. azimuthal distance between scan points, varies in different segments of the field of view between 0.25° in the central segment and 0.5° on the sides, resulting in a total of 390 scan points per layer. In the vertical direction, the sensor has four layers that are spaced 0.8° apart in the elevation angle. The scan grid for the instantiation of measuring rays in the ray tracing framework therefore is a matrix containing 1560 entries of tuples of azimuth and elevation angles. For each scan point, one central ray and two additional rays above and below the central ray in the elevation direction are launched. The additional rays account for dispersion of the ray and can result in multiple echos for a single scan tuple, which will be discussed in detail during the model introduction. This results in a total of 4680 primary rays generated and traced during each update cycle of the sensor measurement model. As an exemplary scenario a standard setup from Vires VTD of a highway scenario with the ego vehicle and one single target vehicle in front of the ego vehicle is used. The distribution of relative frequencies of the sensor plugin's cycle time during a run of the simulation is shown in Fig. 4.4.

Each cycle of the sensor plugin includes the steps from calculating ray starting points and directions at the current simulation update step for configuration of the ray tracing

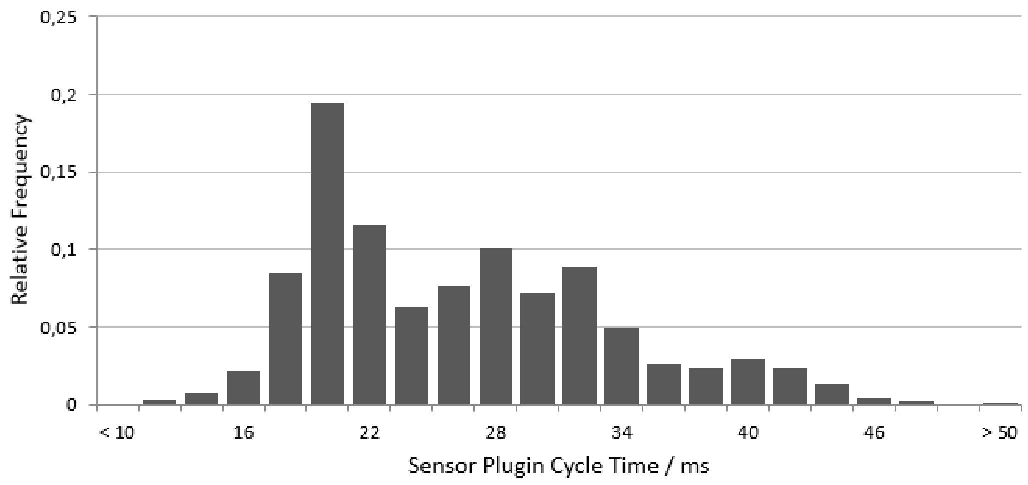


Figure 4.4: Relative frequencies in the distribution of cycle times of the sensor plugin implementing the sensor measurement model. Each bar represents an interval of 2 ms. One cycle of the sensor plugin includes ray generation and ray tracing, post processing of results, and exporting the resulting point cloud data.

engine, the ray tracing including evaluation of material properties and hit programs for each intersection, the post processing of ray tracing data, and the export of data after conversion into the OSI::LowLevelData format. The average cycle time of the sensor plugin is 25.4 ms, with individual values in the range from 11 ms to 71 ms. As can be seen in Fig. 4.4, the vast majority of values is below 40 ms, which marks the required boundary to enable a sensor update rate of 25 Hz to match the real world sensor. For this reason it is safe to say that the sensor measurement model discussed in this work enables the real time generation of low level sensor data. However, not a large margin of computational reserves remain, which further highlights the need to start with a simple model during the development of the sensor measurement model and increase complexity, e.g. by including higher order reflections through secondary ray generation, only when there is a dire need. Note that although a fixed number of 4680 primary rays are generated during each update cycle of the model, the number of points in the resulting point cloud data is not a fixed number as any rays without intersection within the defined maximum ray tracing distance, e.g. any ray pointing skywards, does not result in an entry in the point cloud. The same applies for the additional rays that can detect potential multiple echos, which only in some situations occur. For the test scenario described here, the point cloud has an average number of 575 points, with individual values ranging from 217 to 1201 points. While the number of points in the point cloud has an effect on the processing steps, especially post processing and data export, there is no direct relation apparent between the number of points reported during an update cycle and the cycle time of the sensor plugin.

In addition to the single machine hardware setup described in this section, another setup has been built as a proof of concept for multi sensor integration. This larger setup uses four separate workstations, each running the Vires VTD simulation framework with the OptiX extension and the sensor plugin implementing the sensor measurement model. Each model is configured to match one of four lidar sensors placed on the front, back, and to the sides of a vehicle, replicating part of the sensor setup of a real world research vehicle. One of the simulation instances controls the setup and updates the simulation and replicates the current state of the simulation in each update step to the other instances. All four instances send the generated point cloud data in the format of the OSI::LowLevelData interface using ZeroMQ connections to a fifth workstation running the ROS framework and the data consumers. The real time input of simulated data from all four sensors is used as input to a sensor fusion algorithm developed by another team. As this is not the focus of this work, this multi machine setup is not discussed in further detail. However, mentioning it here serves the purpose of highlighting that the approach to the construction of a sensor measurement model described in this work is scalable and can be used in a variety of applications, where controlled generation of virtual low level sensor data in real time is required.

4.3 Simulation of Lidar Point Cloud Data

The sensor measurement model described in this section is built to simulate the point cloud data output of an automotive lidar sensor. For a lidar sensor this constitutes a low level data format that contains the directional information of each light pulse sent and received as well as the distance to the reflecting target surface. This information can be encoded as points in a three-dimensional space, which form the point cloud output of the lidar sensor. In a subsequent processing step, these data can be used by a higher level perception function, e.g. an object detection or a grid mapping algorithm, which constitutes a first step in the interpretation of the measurement data.

4.3.1 Ray casting Model

For the purpose of this work, a specific type of ray tracing model, also called *ray casting*, is used. In this section, the reasoning for the choice of the model in this work is discussed, followed by the description of the ray casting model. This description of the basic ray casting model forms the foundation for the step-wise introduction of extensions to increase the sophistication of the model, which will be discussed in the next section.

4.3.1.1 Selection of the Model

There are several approaches for handling the simulation of ray propagation. The basic principle is always the same: A ray tracing model aims to trace the path of light between a light source and a light detector. When using ray tracing for image rendering purposes, the typical approach is tracing rays *backwards*, i.e. starting from the detector and not the light sources. This approach is therefore also typically referred to as backward ray tracing. For the purpose of determining the intensity of light that hits the detector in a defined angular segment, this segment is sampled by a certain number of rays, which are called primary rays. The number and distribution of rays within the angular segment are some of the model's parameters. Usually a higher number of sampling rays will result in increased fidelity at the cost of additional computational expenses.

For each of the primary rays, the first intersection point with the geometry of the environment is calculated by the ray tracing engine. For each of the intersection points, the direct contribution of all relevant light sources is calculated. This is done by determining how much light from each light source is scattered from the area around the intersection point in the direction of the detector, i.e. along the direction of the ray that was traced starting from the detector to determine the intersection point. The sum of all of these contributions yields the incoming light from first-order reflections, when light coming directly from the light sources is scattered by the surface that this angular segment of the detector is aiming at. If higher-order contributions should be accounted for, a recursive mechanism starting from each reflection point needs to be employed in order to determine the light falling on the surface segment that is not coming directly from one of the light sources. This is performed by using so called secondary rays starting from the initial point of intersection, and recursively applying the same logic as for the primary rays' intersection points at each further intersection point. With the increase in the order of reflections that is accounted for by a ray tracing model, the number of rays and therefore the required computational power increases exponentially.

In terms of computational requirements, the most efficient choice is choosing the lowest order that still yields the desired model fidelity. A natural starting point for a sensor measurement model for an automotive lidar sensor is therefore a model employing only primary rays, i.e. a *ray casting model*. In this type of model, the intersection points of the primary rays form the basis for directly deriving the point cloud output of the sensor. Given the high computational performance of a ray casting model compared to more complex ray tracing models, this type of model lends itself to the application in the real time environment of an automotive driving simulator.

Besides the advantage regarding the computational requirements, the following reasoning supports the implementation of a ray casting model as a starting point for a lidar sensor measurement model: Both the light source and the detector in an automotive lidar sensor are usually part of the same package and their relative distance is in the order of a few centimeters. In comparison, the measured distances range in the span of one to over a hundred meters. For this reason, the light source and detector may be considered as being located in the same position. In a scanning type of lidar sensor, the illuminated area of the sensor's light source as well as the detector's field of view are both moving in a synchronized scanning motion and largely overlap. By design, the detector is only susceptible to a narrow bandwidth around the wavelength of the sensor's light source. This is achieved by using a filter that is designed to minimize outside influence, especially ambient solar lighting. For this reason, the sensor's own light source is by a large margin the dominant source of light within the momentary field of view of the detector. Under the assumption that the sensor's own light source is the largely dominant light source, the influence of higher order reflections also becomes largely negligible when considering that the light source is only illuminating an area that for the most part corresponds with the detector's current field of view. Therefore, light from the sensor's own light source will typically only be scattered back to the detector directly or is not scattered back at all.

As the area illuminated by the sensor's light source as well as the potential surface area covered by the detector's field of view increases quadratically with the distance from the sensor and the desired sensor range in an automotive application is rather high, this area can grow beyond the typical size of a target depending on the specific build of each sensor. For the purpose of a sensor measurement model, it is important to incorporate this effect, which can be done by employing multiple primary rays and their intersection points in the calculation of the final point cloud as sensor output. This shows, that starting from a simple ray casting model, bit by bit additions can be made that incorporate additional effects observed in real world sensor data. Therefore, the ray casting model forms a natural starting point for an iterative model development cycle that is closely matched to comparison with real world data. In this context, an extension to a full ray tracing model including secondary, tertiary, and higher order rays for higher order reflections is just one possibility that can be followed if the need for such an extension to the simulation model arises.

4.3.1.2 Description of the Model

As described in the flow diagram for data and control flow in the sensor measurement model introduced earlier in Fig. 4.3, the generation of primary rays is part of the camera program of the sensor measurement model. The specification parameters of the lidar sensor that is to be simulated by the sensor measurement model are reflected in the parameters used by the camera program in the generation of the primary rays. These parameters include the sensor's field of view, i.e. the range in azimuth and elevation angle that is covered by the sensor, the number of scanning points in each direction, and the maximum range of the sensor. The field of view and the number of scanning points is transformed to a set of azimuth and elevation tuples that characterize the sensor. The camera program proceeds to generate a primary ray for each tuple in this set, where the sensor origin defines the eye point and the specific azimuth and elevation angles define the direction of the ray. This is shown schematically in Fig. 4.5, where Fig. 4.5 (a) shows the top view with rays generated in a defined pattern in the azimuth direction, and Fig. 4.5 (b) shows the side view with layers defined by discrete values of the elevation angle. Note that the azimuth and elevation angles are typically not measured quantities in a laser scanning sensor but rather statically defined by the hardware setup. The static definition in the model is therefore an exact match to the output data of a real sensor. The ray generated for each azimuth and elevation tuple results in a point in the output point cloud if a valid distance measurement is obtained. For the distance measurement, the sensor's maximum range is used as the maximum limiting boundary required by the ray tracing engine for performance reasons.

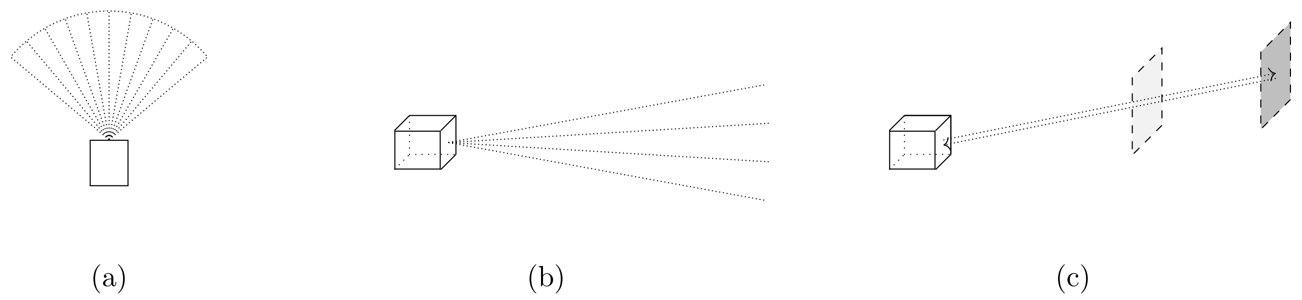


Figure 4.5: Schematic sketch of the basic principle of the ray casting model. (a) *Top view:* from the sensor's eye point primary rays are generated at defined angles spaced over the sensor's horizontal field of view. (b) *Side view:* Vertically, rays are grouped in layers with a common elevation angle. (c) Schematic path of light that is scattered back to the detector by an opaque surface element of a sensor target illuminated by the sensor's light source after passing through a transparent surface. The offset between rays is for illustration purposes only.

The ray tracing engine takes over the task of finding for each ray its intersections with the virtual world geometry, if any exist within the specified maximum range. When intersections are found, they are processed in the order of their distance from the ray's origin. The additional intersections past the first one for the same ray are important to take into account transparency of surfaces. For the scope of the basic ray casting model, the first intersection with a non-transparent material has to be found. This is sketched in Fig. 4.5 (c): A ray starts from the sensor and passes all transparent surfaces until it hits the first non-transparent surface in its path. From this point it is assumed that a fraction of the light is scattered back to the sensor and this point is the base for deriving the synthetic point cloud output. For this purpose, the light source and the detector within the sensor are assumed to be at the same position, which means that the ray traced to determine the intersection point can be considered as a backwards traced ray from the detector and at the same time as a forwards traced ray from the sensor's light source, eliminating the need for additional ray calculations.

At this point, the surface material properties at the intersection points come into play. Within the context of the ray tracing engine, there is a material model accompanying the node graph that defines all objects and surfaces in the virtual environment. The material model is defined by a material database that contains information about the properties for each type of surface. At each intersection point, the ray tracing engine performs a lookup of the material properties for the particular surface. The material model of the ray tracing engine allows the definition of any number of custom properties. These properties include any number of scalar parameters as well as the definition of a material specific *hit program* that is executed for each intersection point with the following context parameters: the ray's properties, the intersection point's properties, and the respective material properties from the material database. The hit programs and the custom-built material database form the foundation of the sensor model's ability to emulate several of the specific effects observed in the lidar point cloud data, as detailed in the next section on extensions of the basic model. Custom properties include the definition of physical properties, e.g. reflection coefficients, as well as semantic properties, e.g. type of object, for use by the sensor model.

One of the most important uses of the material model and therefore part of the basic functionality of the ray casting sensor model is the handling of transparency. The graphics renderer of the driving simulation framework uses the rendering technique of *software sprites* for many static objects with a complex outline, e.g. trees, to decrease computational load and make their real time rendering feasible. In this technique, a rough bounding box with a simple two-dimensional shape, e.g. a rectangle, is drawn at the location of the object that is always oriented perpendicular to the viewing direction of the camera eye point. On this surface, the outline is rendered using a texture with an alpha channel. The alpha channel determines the transparency value for each pixel. All the area outside of the complex outline is transparent and therefore only the object's actual outline is apparent to the viewer. As the same graphics objects are used by the ray tracing engine, intersections are found for rays crossing the bounding box of any such shape at any point. In order to avoid phantom detections from these shapes, the model has to perform a texture lookup for each intersection and decide based on the alpha channel value of the texture at the point of intersection whether the respective intersection point should be discarded from the list of candidates. This look-up and decision is part of the hit program for any surface type and therefore an integral part of the basic model.

Finally, when the first intersection point with a non-transparent surface has been found for each primary ray originating from the sensor's eye point, this point is used for further processing. The position of this point forms the base for post-processing in determining the properties the corresponding point that is part of the synthetic point cloud output. Within the context of the ray casting model, this point is the closest approximation for where the sensor's light beam hits a sensor target. Based on this definition of the model, it is possible to determine an output point for each tuple of azimuth and elevation angles that is scanned by the lidar sensor. This constitutes the basic definition of a sensor measurement model that can emulate the point cloud output of a lidar sensor within a virtual environment and provide real time sensor output.

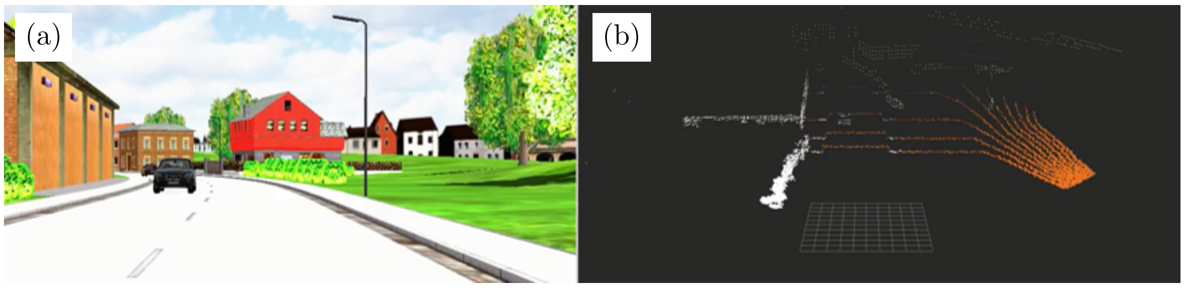


Figure 4.6: Snapshot of the output of point cloud data generated by the ray casting model in a dynamic virtual driving scene. On the left, (a) shows the output of the standard visual renderer with the eye point set to the sensor's position within the ego vehicle. The standard renderer of the Vires VTD driving simulation framework is used. On the right, (b) shows the point cloud of a simulated hypothetical 10-layer laser scanner sensor mounted on the ego vehicle. The color of the points denotes the object classification: orange points are ground points, white points are dynamic or static objects and obstacles in the environment.

The output of the ray casting model is shown in Fig. 4.6. The usual rendering of the scene with the camera located at the point of view of the sensor is shown as reference on the left side in Fig. 4.6 (a). The unnatural looking colors of the scene are the result of a currently unresolved issue in the graphics renderer that occurs when the ray tracing engine is used in parallel, however it does not affect the workings of the sensor measurement model in the slightest. On the right side, in Fig. 4.6 (b) the corresponding snapshot of the synthetic point cloud generated by a sensor measurement model as described here is shown rendered within a three-dimensional environment. The sensors parameters were chosen arbitrarily for illustration purposes with ten layers in the elevation direction with 2° distance between layers and a field of view of 140° in the azimuth direction. Note that the reference camera rendering has a much smaller field of view to avoid optical distortions.

The synthetic point cloud output generated by the model, shown in Fig. 4.6 (b), clearly contains all relevant features of the scene, including the static obstacles like buildings, trees, and shrubbery as well as the dynamic obstacles, i.e. cars, in the scene. In addition, the outline of the ground is clearly visible, flat on the street and with an upwards slope on the grassy surface to the right. For better illustration, ground points are shown in orange while objects are shown in white. This distinction between object and ground points within the model is one of the possible extensions of the basic ray casting model described in this section. In general, this implementation of a sensor measurement model lends itself to a variety of extensions that serve the purpose of increasing realism within the model as well as adding additional properties to the point cloud output. These additional properties are important in order to allow the substitution of a real sensor's data stream with a synthetic data stream from a sensor measurement model within the processing chain of an automated driving system that uses a simulated environment as its source of input. Building on the basic ray casting model, extensions of the model are the subject of the next section.

4.3.2 Model Extensions

Using the basic ray casting model introduced in the previous section as a starting point, a refinement of the sensor measurement model can be performed in a step-wise fashion. Based on observations in real sensor data, additional effects can be identified and a corresponding component added to the sensor measurement model. These effects include the classification of reflection points depending on the type of target that caused the reflection, multiple echos due to beam divergence, a detection threshold based on the light intensity received from a reflection, and the addition of noise effects. Typically, each addition requires modification in multiple components of the sensor measurement model, which consist of the camera program for ray generation and post processing of ray data, the material definitions within the material database, the hit programs performing calculations on intersections found by the ray tracing engine, and the data exporter that forwards data to the consumers for further processing. The overview of the sensor measurement model's components is given in Fig. 4.7 in a simplified view with a focus on the different components to provide a reference for the discussion of this section. The incorporation of these various effects into the sensor measurement model is the subject of this section, with the exception of noise. Noise effects are discussed separately in the next section.

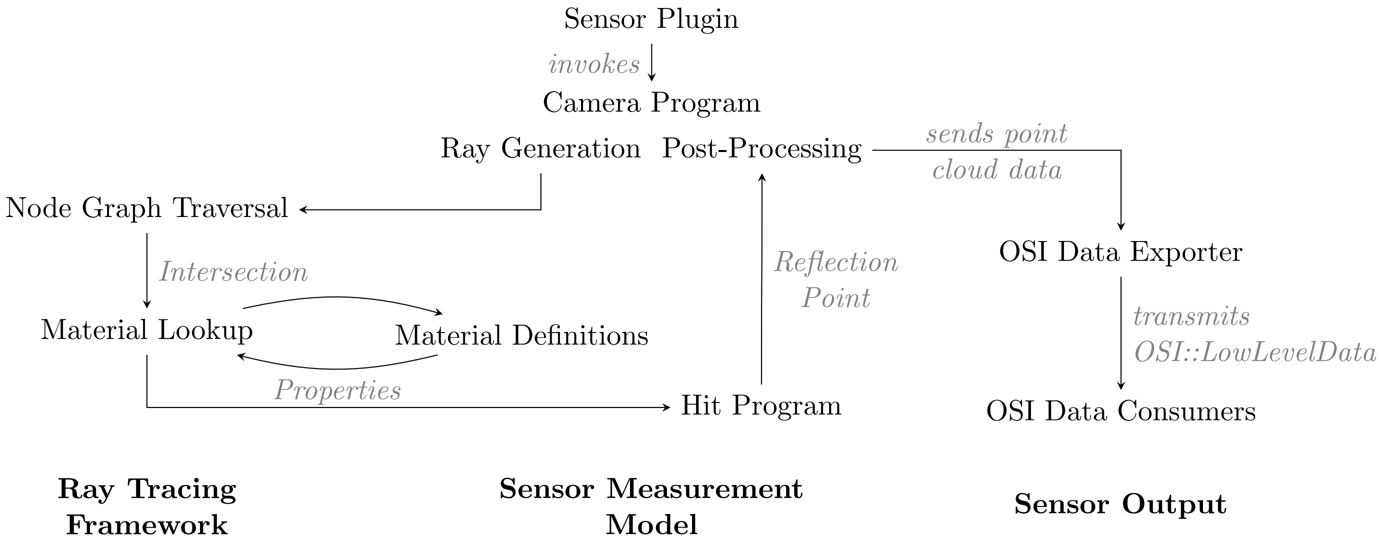


Figure 4.7: Components defining the sensor measurement model and their relation.

Classification

The point cloud output of an automotive lidar sensor typically contains not only the coordinates for each reflection point, but also additional information for each point that is the result of sensor internal low level processing on the results of the measurement process. One of these properties is the estimate for the classification of the sensor target that is responsible for the reflection. The classification algorithm used by the sensor is a proprietary implementation specific for each sensor and vendor. The classification is performed in an sensor internal processing step based on the measurement raw data from the physical measurement, as shown in Fig. 4.8 (a).

For the highest degree of realism, a sensor model has to generate measurement raw data from a model of the physical measurement within the virtual environment, which is done by the ray casting implementation described so far, and then pass these data to the identical classification algorithm that is used in the sensor internal low level processing step. This, however, this a major issue in the implementation of a sensor measurement model as typically there is no interface for this sensor internal intermediate raw data available and the detail of the proprietary classification algorithms are not accessible for the simulation model in the context of this work. This issue is the low level equivalent of the object detection algorithms described at the beginning of the chapter in Fig. 4.2 (a), where the original algorithms for object detection are usually not accessible and a replacement model has to be used. For this reason, within the scope of this work a replacement model for the point classification based on ground truth data is chosen. Instead of deriving the classification from the measured reflection

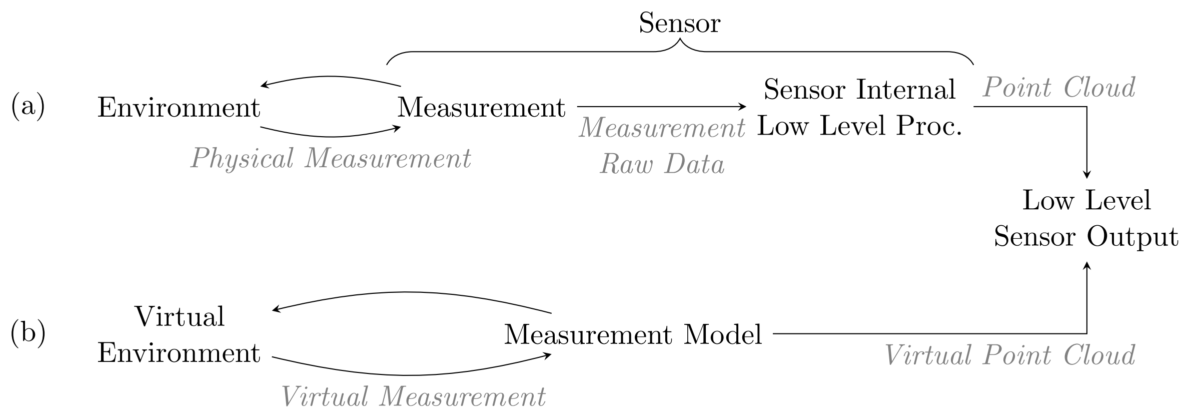


Figure 4.8: Sensor processing chain for low level sensor output data, i.e. point cloud data for a lidar sensor. The top row (a) shows the processing chain for a real sensor, the bottom row (b) shows the processing chain for a sensor measurement model. Whereas the processing chain for a real sensor can be separated into a physical measurement step and an internal processing step, without the corresponding internal interfaces for comparison of data the sensor measurement model is limited to incorporating both within one processing step.

points, the classification becomes a directly measured quantity in the simulation context as shown in as shown in Fig. 4.8 (b). As the information about the object type can be encoded in the virtual environment, the sensor measurement model reads the classification information directly during the virtual measurement step.

The classification categories are specific for each sensor, but usually follow a common pattern. For the purpose of the sensor measurement model, the following classification types are used: Ground reflections, i.e. surface area that is potentially accessible to the car, object reflections, i.e. obstacles to be avoided by the car, and other reflections marked as invalid, which may be caused by rain or spray if these effects are added to the model or generally any type of surface that is unable to be classified as either of the other categories. Object reflections are further subdivided into dynamic and static objects. Dynamic objects are all moving actors in the simulation, i.e. cars and pedestrians, whereas static objects are any fixed parts of the environment that do not classify as ground, e.g. trees, buildings, and guard railing. Ground reflections are further subdivided into reflections from road markings, which can be detected by some real world sensors due to their higher reflectivity characteristics, and generic ground, which applies to road surface and any other flat surfaces.

In the implementation of the sensor measurement model, the information about the object type is encoded as part of the material definitions. As shown initially in the overview of sensor measurement model components, Fig. 4.7, this allows the lookup of this material property when detecting an intersection. The semantic information of the object classification is then processed as a custom property in the material definition by the hit program. For this reason, the material database has to be extended with the type information for all surface materials, e.g. street surface materials are given the ground classification and car surface materials are given the dynamic object classification. This approach results in a ground truth approach to transmitting object classification, i.e.

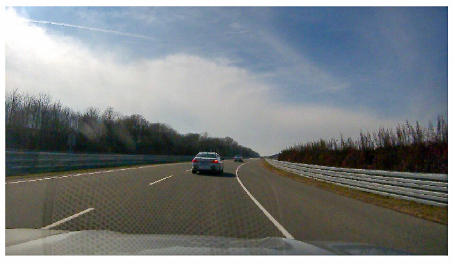
the value in the final output is always correct as taken directly from the description of the virtual world, and constitutes a basic replacement model for the low level processing within a sensor.

Beam Divergence and Multiple Echos

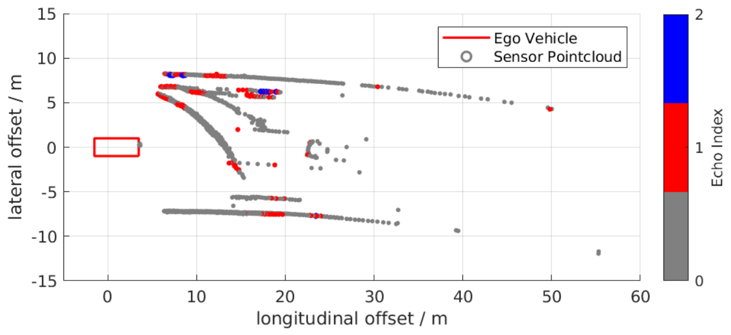
For each tuple of azimuth and elevation angles in the scanning grid, an automotive lidar sensor typically is able to return multiple echos resulting in multiple points in the output point cloud. These multiple echos are largely caused by partial illumination due to beam divergence. The area A_{beam} illuminated by the laser beam approximately scales with distance r from the sensor as $A_{beam} \propto r^2$. On the scale of distances that automotive lidar sensors operate, the beam divergence is usually noticable as the illuminated area quickly grows bigger than the size of targets. Partial illumination then occurs if some of the light is reflected by a target that is either smaller than the beam area $A_{beam}(r)$ at the given distance or if the beam hits the target along one of its edges. The remaining light can be reflected by another surface at a further distance, resulting in an additional echo reported by the sensor. Additionally, multiple echos can be caused by semi-transparent, partially reflecting surfaces. The occurrence of multiple echos in the point cloud data of an IBEO Lux sensor mounted on the front of a vehicle is shown in Fig. 4.9. As shown on the camera recording on the left side in Fig. 4.9 (a) for reference, the scenario is a highway like closed test track with a target vehicle in front of the ego vehicle and noticeable guard railings on each side of the road. The point cloud output of the lidar sensor with the echo index of each point defined by the color code is shown on the right side in Fig. 4.9 (b). Gray dots denote the closest echos (echo index 0), while red and blue dots denote second (index 1) and third (index 2) echos. In this scene, higher index echos occur mostly in the reflections from the guard railings, which is likely the effect of their surface structure leading to partial illumination and therefore multiple echos in many cases.

Further, Fig. 4.9 (b) shows that secondary reflections with echo index 1 occur only in some places and that the majority of reflections are primary reflections with echo index 0 as would be expected. Third echos with echo index 2 are even less likely to occur with only a small number present in the point cloud data. This visual impression can be quantified by calculating the relative frequencies of echo indices within the point cloud data averaged over the complete test drive instead of a single sensor frame. The resulting normalized distribution is shown in Fig. 4.10. The vast majority of points with 79% reports echo index 0, around 20% report echo index 1, and only about 1% of points have echo index 2. This confirms the assumption that higher index echos are increasingly less likely to occur. However, it also shows that there is a non-negligible percentage of higher index echos present in the data that should therefore be accounted for in the sensor measurement model.

As the cause of higher index echos is the divergence of the laser beam, the divergence needs to be incorporated into the model. The basic ray casting model introduced in the previous section 4.3.1 uses only a single primary ray to determine the reflection



(a)



(b)

Figure 4.9: Real world lidar point cloud data with echo index for each reflection point. On the left side, (a) shows a still frame recorded by a camera mounted on the ego vehicle's dashboard for a reference of the scene. On the right side, (b) shows the point cloud data from the ego vehicle's front lidar sensor with the echo index marked by each point's color. Gray dots show the first echo (index 0), red dots the second echo (index 1), and blue dots the third echo (index 2) as reported by the sensor. The ego vehicle is shown in red, with the center point of it's rear axis defining the origin of the relative coordinate system.

point for each tuple of azimuth and elevation angles in the scanning grid, which does not allow determining multiple reflection points. The divergence of the beam in the azimuth and elevation direction are fixed properties of the lidar sensor under investigation and usually known quantities as part of the sensor's specification sheet. For the purpose of extending the sensor measurement model, these therefore become additional model parameters that determine the illuminated area at a given distance. Within the ray tracing approach used by the sensor measurement model, the area illuminated by the

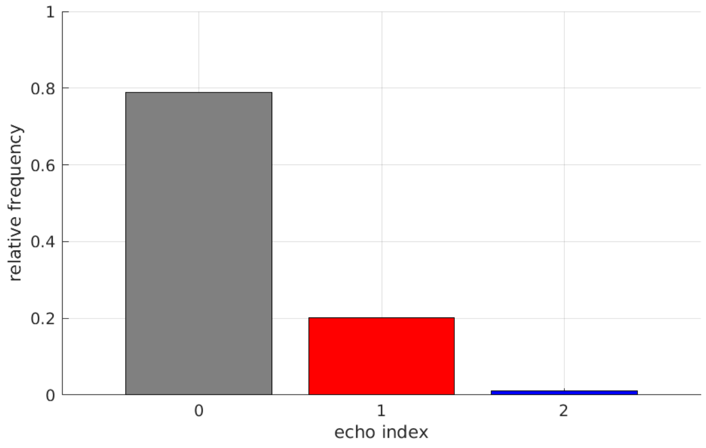


Figure 4.10: Relative frequencies of first, second and third echos (indices 0, 1, 2) within the point cloud data averaged over a recorded test drive.

beam needs to be sub-sampled by multiple primary rays for each azimuth and elevation tuple in the scanning grid in order to detect multiple intersecting surfaces that can cause the occurrence of multiple echos. This is illustrated in Fig. 4.11. For each azimuth and elevation tuple in the scanning grid, additional primary rays with an offset from the central direction are generated in addition to the central ray from the basic model introduced in the previous section 4.3.1. The number of rays used for sub-sampling is a trade-off between performance and the ability to detect additional reflecting surfaces. As a starting point, either two or four additional rays can be used. With two additional rays only the divergence in the elevation or azimuth direction is handled. With four additional rays both directions are handled. In this case two of the rays are offset by half the divergence angle in both directions of the elevation direction, and the other two rays in both directions of the azimuth direction. This is based on the assumption of an elliptical beam profile, where in addition to the center the end points of the major and minor axis are sampled. As very small objects are uncommon to have explicit representations in the virtual environment for performance reasons, this approximation captures the case of edge illumination nicely while requiring only a minimum of additional computational resources over the base model. A further performance improvement is the usage of only two rays, when assuming that divergence in either angle is the dominating factor, e.g. in the case of the IBEO Lux sensor, the illumination profile is larger in the elevation angle.

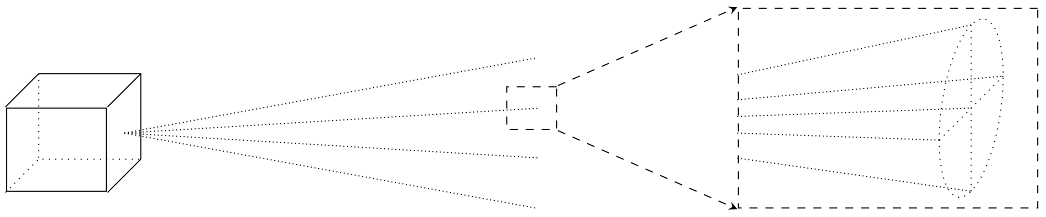


Figure 4.11: Schematic sketch of ray generation including the accounting for beam divergence within the scope of the lidar sensor measurement model. For each azimuth and elevation tuple characterizing a scan point, a number of sub-sampling rays around the central ray are generated in order to model the effect of beam divergence, as shown in the enlargement inset on the right.

From these three or five sub-sampling rays, a maximum of three or five valid first intersections can be obtained. However, most of the time all of the intersections belong to a surface on the same object. As the beam area is continuous in a real sensor, this case would not result in multiple, differentiated echos detected by the sensor. To counteract this artifact of the sensor measurement model, a small threshold distance is introduced. The distance measurements of all valid intersections are compared and if the distance obtained from any two intersections are smaller than the threshold distance these measurements are merged to one measurement result with a distance given by the average of both results. Only if the distance is larger than the threshold, two distinct echos are reported as independent points in the point cloud output. Note that from the sub-sampling rays only the distance measurements are used as all output points are mapped on the original azimuth and elevation tuples from the scanning grid. The scanning grid is fixed by the hardware layout of the scanner, so this behavior corresponds to the output of the real sensor. Choosing the threshold distance is done empirically with an upper bound obtained from investigating real world sensor data: The threshold must be chosen smaller than the lowest distance observed between any two points with identical azimuth and elevation angles in the point cloud output of the real sensor.

Intensity Detection Threshold

In the base model the only limit imposed on the maximum distance of reflection points in the output of the sensor measurement model is defined as a parameter of the model. This parameter is used as the range limit given to the ray tracing engine, which requires this limit for efficient node graph traversal. The parameter is taken from the maximum range given in the specification of the lidar sensor under investigation or estimated empirically by determining the maximum observed range in real world test data. With this definition, the output of the model constitutes a superset of reflections actually detectable for the lidar sensor. In order to increase realism in the sensor measurement model with respect to the detection range, an estimate for the reflected intensity of light is calculated. The reflected intensity is then compared to the light receiver's internal detection threshold, which constitutes an additional model parameter. Only if the intensity of a reflection is greater than the detection threshold, an output point is added to the point cloud.

For determining the reflected intensity, the ratio between received and transmitted power, P_{rx} and P_{tx}, is calculated as

$$\frac{P_{rx}}{P_{tx}} = L_i \, A_{rx}(r) \, R \, A_{tx}(r),$$

where A_{tx} is determined by the attenuation experienced by the beam while traveling a distance r to the reflecting surface, A_{rx} is determined by the attenuation experienced on the way back, R determines the fraction of incident power reflected by the target surface and includes effects of partial illumination, absorption, and relative geometric orientation as well as the material's reflection coefficient, and L_i is a factor accounting for internal losses of the receiver array. Assuming constant values for power P_n of receiver noise, transmitted power P_{tx}, and internal loss factor L_i as well as an exponential attenuation of the form $A_{rx/tx} \propto \exp^{-\mu r}$ with a fixed attenuation coefficient μ, the above equation results in a threshold distance for a fixed signal-to-noise ratio that is a function of the target's reflection properties:

$$r < d_{\text{threshold}} \left(R \right).$$

Neglecting all geometric influences on the reflected fraction of incident power R and focusing only on the target-specific properties, the threshold distance $d_{\text{threshold}}$ becomes a material property for the purpose of the sensor model. This property then is a parameter of the model and can be measured empirically from real world data for a specific set of environmental conditions and type of target. This allows the inclusion of various maximum detection distances for different types of targets in the context of the simulation environment.

4.3.3 Measurement Uncertainties

Up to this point, the construction of the sensor measurement model have all been based around finding a way to represent the physical properties of the measurement process in the virtual environment with the use of exact calculations. However, in order to finally arrive at the realistically seeming synthetic point cloud output shown earlier in the introduction of the sensor measurement model in Fig. 4.6 (b), a feature found in the observed real world sensor data is still missing: noise. There are a variety of reasons for the presence of noisy characteristics in the distance measurements of a lidar sensor, ranging from receiver internal noise to the target surface being not at all smooth on the scale of the beam area at the given distance as is e.g. the case for trees, grass and shrubbery.

The three-dimensional virtual environment of the driving simulation is largely an idealized representation as very high fidelity models for objects and surfaces in the environment could not be rendered in a real time environment. For this reason, the noisy behavior is added within the scope of the sensor measurement model in the final post-processing step (see Fig. 4.7 for the overview of the processing steps of the model). Before encoding the output data in point cloud form, a random variation in the position of the detected reflection point is introduced in the model in the form

$$(r, \epsilon, \alpha) \rightarrow (r + \eta_M, \epsilon, \alpha)\,,$$

where r is the radial distance coordinate of the measurement, ϵ the elevation, α the azimuth, and η_M is a random variable. Note that only the distance measurement is affected by the random noise as the elevation and azimuth angles are fixed on the scanning raster by the hardware layout of the sensor and are not measured quantities in both the real sensor and the sensor measurement model.

As a starting point for the model a Gaussian white noise distribution is used for the random variable η_M with a variance that is dependent on the material properties of the reflecting surface. This variance therefore becomes an additional model parameter as part of the material definitions that has to be determined empirically for different types of surfaces. This dependency on the type of reflecting surface is added due to the observed behavior of largely increased noise in the output point cloud of a lidar sensor in areas of non-flat surfaces, e.g. ground vegetation like grass, compared to flat surfaces, e.g. walls or the outline of cars. Flatness in this context is always relative to the illuminated beam area. In addition, other dependencies may be included in the distribution of the random variable. One example is the distance from the sensor, however, for lidar sensors the measurement distance plays an insignificant role in the noise characteristics of the output point cloud.

So far, several ingredients for the construction of a ray casting based sensor measurement model for an automotive lidar sensor have been discussed in this chapter. With all of these ingredients the model is able to produce a realistically seeming output point cloud as initially shown in Fig. 4.6 (b) that also includes all additional properties for each measurement point as present in a real sensor's output. Using the

OSI::LowLevelData data format (see section 2.5 for the description of the Open Simulation Interface OSI), this virtual point cloud data can be transmitted to a perception function of an automated driving system for the construction of the system's internal environment representation. This is the focus of the next chapter.

4.4 Creating Low Level Sensor Data in a Virtual World

With the tool of a sensor measurement model in hand that is able to generate a point cloud output including all properties also found in a real lidar sensor, we are equipped to turn to the task of using the sensor model to feed input to a perception function that generates an internal environment representation for an automated driving system. The low level sensor output data of a lidar sensor's point cloud can be especially used for modern environment representations that are more detailed than the object representation of static and dynamic objects in the environment. One such representation is the so called occupancy grid. The focus of this section is the coupling between the virtual environment of a driving simulation framework and the first step in the processing of an automated driving system, the construction of an internal environment representation.

4.4.1 Occupancy Grid and Occupancy Grid Map

The occupancy grid representation of the vehicle's environment segments the area surrounding the vehicle into a grid of uniform size and assigns a probability of occupancy to each grid cell. For each grid cell it is determined whether the respective cell is free of obstacles and drivable or part of an obstacle and has to be avoided in the trajectory calculation. As this decision is usually not a strictly binary decision based on the available evidence, a probability is calculated for each grid cell. The data source for this calculation is the sensor data from one or multiple of the vehicle's sensors. The occupancy grid method therefore also functions as a type of sensor fusion algorithm, merging the data obtained from several sensors. Details of the occupancy grid method can be found in references [44, 22, 23, 24]. The grid is usually a two-dimensional representation of the environment, although some three-dimensional extensions are being investigated.

An occupancy grid representation of the environment may be based on one static snapshot of sensor data or it can be updated as new sensor data are recorded while also taking in account the previous history. In the latter case, the motion of the vehicle itself with respect to its environment also has to be accounted for, which requires additional odometry sensor data. In order to clearly differentiate between the two cases, we refer to the case of a static snapshot as an *occupancy grid* and the case including the history of measurements as an *occupancy grid map*. The occupancy grid map can be seen as a fusion of all the current and previous sensor data available to the vehicle, resulting in the best representation of the environment within the limits of this type of perception description.

The occupancy grid map representation of the environment is especially suited to give a very good view on the static environment of the vehicle including the drivable and non-drivable space. This information can then be easily used by a trajectory planning algorithm or similar processing step within an automated driving system. The occupancy grid map is therefore a good candidate for an abstraction layer representing the output of a perception function that is based on low level sensor data. Serving as input for subsequent steps in the automated driving system, such an abstraction layer can be used to gauge the realism of the sensor measurement model. One goal of sensor models in the context of automated driving is the realistic stimulation of the algorithms of an automated driving feature, i.e. providing input data generated within a virtual environment that are indistinguishable for these algorithms from real world data. As it is very difficult to properly compare low level sensor output data like a lidar point cloud between a simulation environment and real world sensor data, a comparison on the abstracted level of the occupancy grid or occupancy grid map is favorable for this purpose.

In order to arrive at the occupancy grid map representation from a virtual environment several steps have to be performed. The output of each step is shown in Fig. 4.12. The driving simulation framework provides and updates the virtual world, which is shown in a typical camera rendering in Fig. 4.12 (a) for a standard highway scenario. Based on the virtual environment, a sensor measurement model as discussed in this chapter is implemented. Matching the setup of a real world test vehicle, the sensor model is parametrized for a four-layer lidar sensor with a horizontal field of view of 145° mounted at the front of the vehicle and oriented in the vehicle's forward direction. The point cloud output generated from the model is shown in Fig. 4.12 (b). Individual points are colored according to their classification, with ground points shown in orange, and reflections from objects shown in white. As objects, the outline of one vehicle in front of the ego vehicle as well as reflections from the central barrier of the road as well as some street signs are clearly visible.

The output of the sensor measurement model is then given as input to an occupancy grid generation algorithm in the same way that the algorithm would receive input data from a real sensor. The momentary snapshot of the occupancy grid is shown in Fig. 4.12 (c). The occupancy probability is encoded as a gray scale color for each grid cell: Grid cells with an obstacle, i.e. high occupancy probability are shown in black, grid cells with zero or low occupancy probability are shown in white or light grays. Using the physical properties of laser reflections, the occupancy grid generation algorithm can assign a zero or low occupancy probability to all grid cells between the sensor and the reflection point, while at the point of reflection a high occupancy probability can be assumed. Building on a time-series of occupancy grids, the occupancy grid map is generated as shown in Fig. 4.12 (d). As is easily recognized, the occupancy grid map representation contains information for areas to the side and behind the vehicle that are outside of the current field of view of the sensor. This information is the result of calculations using the history of previous sensor measurements and the

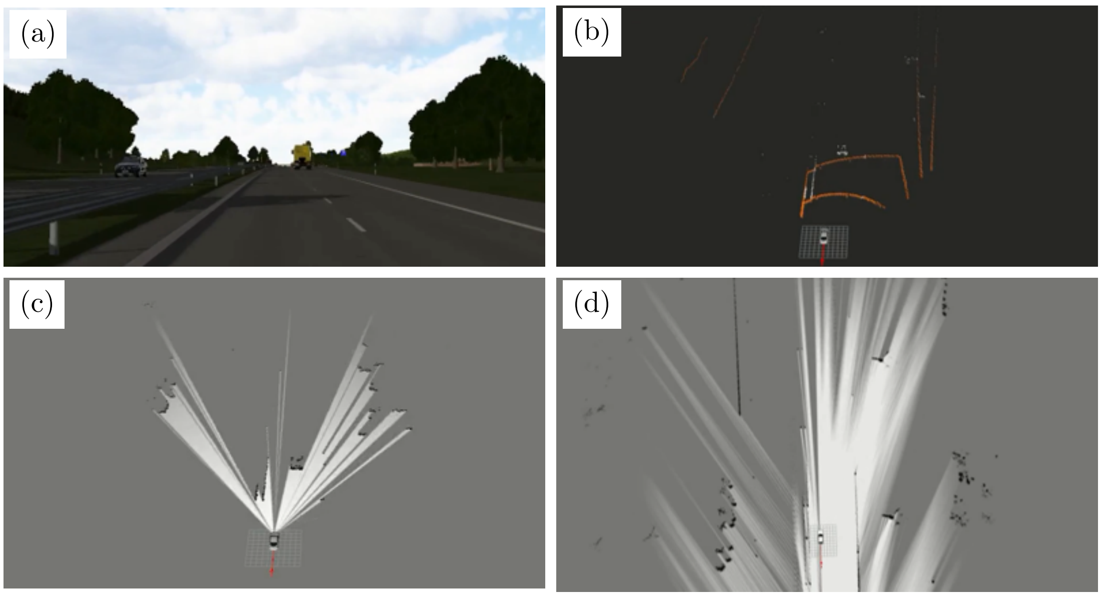


Figure 4.12: Different representations of the environment linked by the sensor measurement model. (a) shows a rendered snapshot of the virtual environment within a highway scenario, (b) shows the same scene in the form of the point cloud output from the sensor measurement model for a lidar sensor mounted on the ego vehicle (rendered for clarity in addition to the point cloud), (c) shows the occupancy grid representation of the environment as constructed from the sensor model's point cloud output, and (d) shows the occupancy grid map representation that also includes the history of measurements before the current snapshot in its construction.

vehicle's motion. The occupancy grid map representation in this form is visually very similar to the output seen during real world test drives and can be used for further processing by an automated driving system, realizing the original goal of stimulating an automated driving system's environmental perception with sensor data generated from virtual environment. The processing chain that realizes all the processing steps shown in Fig. 4.12 is the focus of the next section.

4.4.2 Occupancy Grid Generation in a Virtual Environment

Arriving at the occupancy grid map representation from the virtual environment is the result of a close interplay between components in the driving simulation framework and the framework handling the components of the automated driving feature. As driving simulation framework in this work, the software Vires VTD [60] is used. For the framework handling occupancy grid generation the Robot Operating System (ROS) [73] is used. For all interfaces between the driving simulation frameworks and the ROS framework, the interface definitions of the Open Simulation Interface (OSI) as introduced in section 2.5 are used. The goal in the setup of the processing chain is to maximize realism by utilizing components that are used in a real world test vehicle setup as much as possible. The use of a sensor measurement model generating low level sensor data adds to the realism as no abstractions, e.g. object hypothesis, are included in the sensor model.

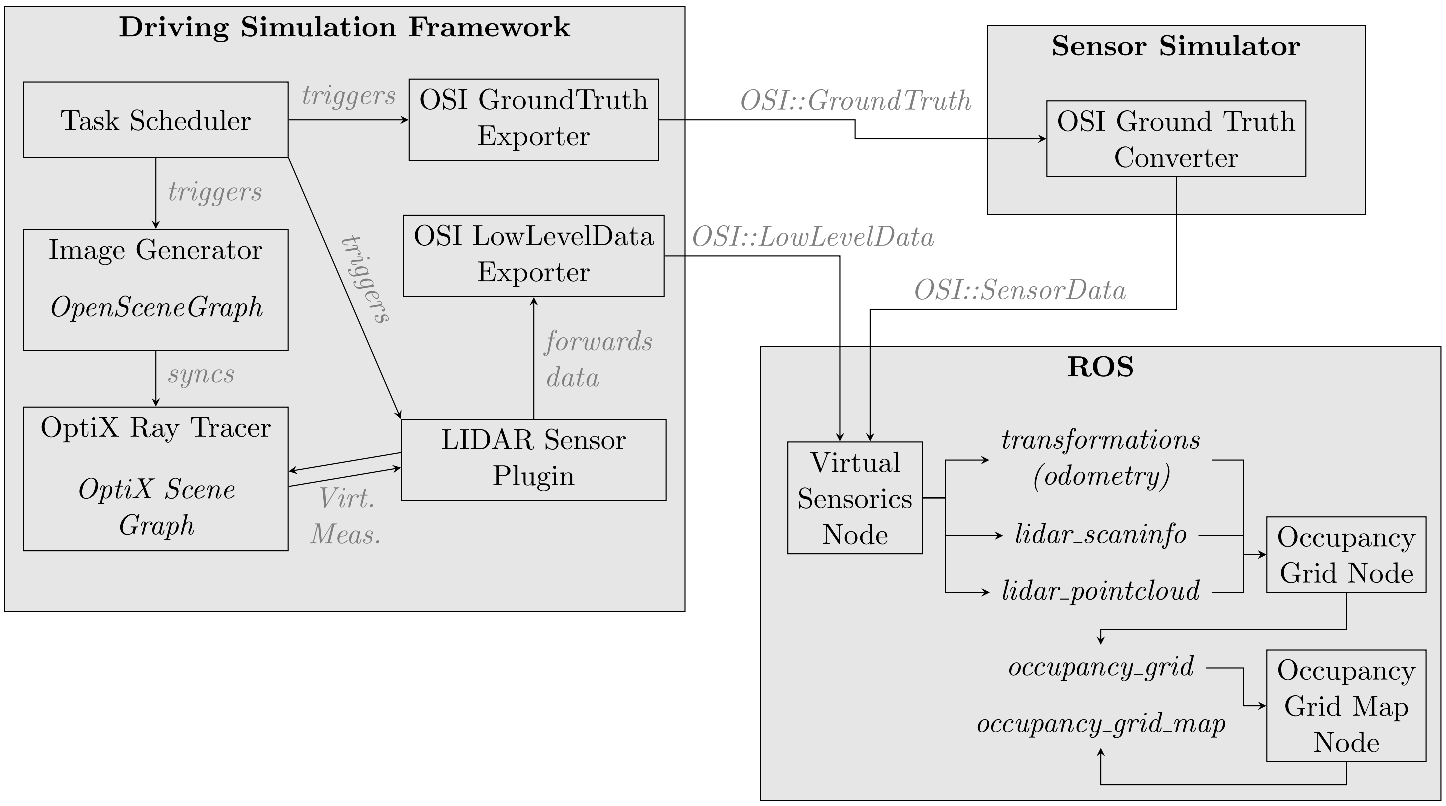


Figure 4.13: Processing chain including all components required for real time generation of the occupancy grid map environment representation from a virtual environment using virtual low level sensor data generated by a sensor measurement model The driving simulation framework is responsible for generating and updating the virtual environment and running the OptiX ray tracer used by the sensor measurement model. The sensor plugin generates a point cloud that is transferred to the ROS environment using the OSI::LowLevelData format. Additionally an OSI::GroundTruth export from the simulation framework is transmitted to the ROS framework containing the odometry information of the ego vehicle that is required for calculation of the occupancy grid map within the occupancy grid node in the ROS framework.

The components of the processing chain and their relations are illustrated in Fig. 4.13. This is a specific implementation including the definition of all the used interfaces derived from the general outline for a ray tracing processing chain introduced in the beginning of the chapter in section 4.2.1. There are three frameworks involved in the setup: The driving simulation framework is responsible for the virtual environment and running the sensor measurement model as a plugin. Its output is the low level sensor data from the sensor model as well as ground truth data. A separate sensor simulation framework is used to convert the osi::GroundTruth data to osi::SensorData. This sensor simulation framework is also used to implement sensor error models as discussed in chapter 3. In this context however, there is only a straight forward OSI converter present that does not modify the content of the ground truth data, but tranforms it to the osi::SensorData interface in order to satisfy the input requirements of the virtual sensorics node within the ROS framework. Finally, the ROS framework receives all these data streams in a similar way as it would receive input data from vehicle mounted sensors in a real world setup. These individual components are discussed in more detail in the following.

The driving simulation framework is at the heart of the virtual environment setup. Within the simulation framework the virtual environment is initially generated and continuously updated over the course of the simulation. As shown in the top left part of Fig. 4.13 a task scheduler works to trigger among others the two relevant components for the sensor processing chain at defined intervals: The OSI exporter operates on the object level and generates osi::GroundTruth data. The image generator updates its internal scene graph and uses it to render the current scene from a defined camera position to the screen, giving a visual representation of the environment as shown in Fig. 4.12 (a). In addition, the updated scene graph of the image generator is synchronized to the OptiX scene graph of the OptiX ray tracing engine. Once the OptiX scene graph has also been updated, the sensor plugin is invoked that contains the implementation of the sensor measurement model as described in this chapter. The output of the sensor measurement model is a point cloud in the osi::LowLevelData format. The point cloud data as well as the transformed object ground truth data for the odometry information is transmitted to the ROS framework via a network connection in a similar way as real sensor hardware would use its specific data format to send data to the computing unit running the processing of an automated driving feature.

In the ROS framework a specific virtual sensorics node is implemented in analogy to specific sensor nodes for each type of sensor sending data to the ROS framework in a real world setup. The virtual sensorics node has the task of handling the network connection, de-serializing of the input data and publishing it in the form of ROS topics for use by other components. For the purpose of a test setup for the generation of an occupancy grid map, three ROS topics have to be generated by the virtual sensorics node as shown in Fig. 4.13. These are the *transformation* topic that contains the odometry information from the ego vehicle, the *lidar_scaninfo* topic that contains information about the scanning sensor and meta information about the point cloud, and the *lidar_pointcloud* topic that contains the point cloud data. All three of these topics are not specific to the processing chain using a virtual environment input, but have been defined for the real world vehicle use case. The *lidar_scaninfo* and *lidar_pointcloud* topics are specific to the type of lidar sensor used in test vehicles, and therefore the guidelines for the task of model construction have been largely set in order to generate all the required data for these interfaces. It is at this point in the processing chain that from the data format alone, there is no difference to a real world processing chain. Therefore the combining link between the real world and the virtual environment is the virtual sensorics node in the ROS environment. The virtual sensorics node receives sensor model data from the standardized OSI interfaces and adopts these to the specific interface of the original sensor that the model aims to simulate.

The *transformation*, *lidar_scaninfo*, and *lidar_pointcloud* topics constitute the required input for the occupancy grid node, which is the next processing step for the sensor data within the ROS framework in our setup. This node is responsible for the generation of an occupancy grid from the current snapshot of sensor data, which it publishes in the form of another ROS topic. The visual representation of these data is shown in Fig. 4.12 (c). Using this most recent grid snapshot as input the occupancy grid map node then updates the occupancy grid map with the newly available information, which is again published in the form of a ROS topic. The visual representation of the occupancy grid map as the final step in the processing chain is shown in Fig. 4.12 (d). The occupancy grid map can then be used by any arbitrary other ROS node that implements the next step in the processing chain of an automated driving feature. As for the occupancy grid representation itself, this is the subject of ongoing research.

4.4.3 Using Low Level Sensor Data in the Real and Virtual World

Based on the processing chain discussed in the previous section for using a sensor measurement model to generate an abstracted representation of the environment in the form of an occupancy grid map on the basis of input data generated from a virtual environment, it is possible to perform a meaningful comparison of the model's output data with real world reference data. When performed in a systematic manner, this comparison of the data output can provide the foundation for the validation of the sensor measurement model. However, the validation of sensor models is not the focus of this work, but subject of an accompanying, ongoing investigation. First results have been published in collaboration with the author in references [74, 75]. In this section, only a short overview of the underlying idea of comparing data on an abstracted level and its application to the sensor model is given.

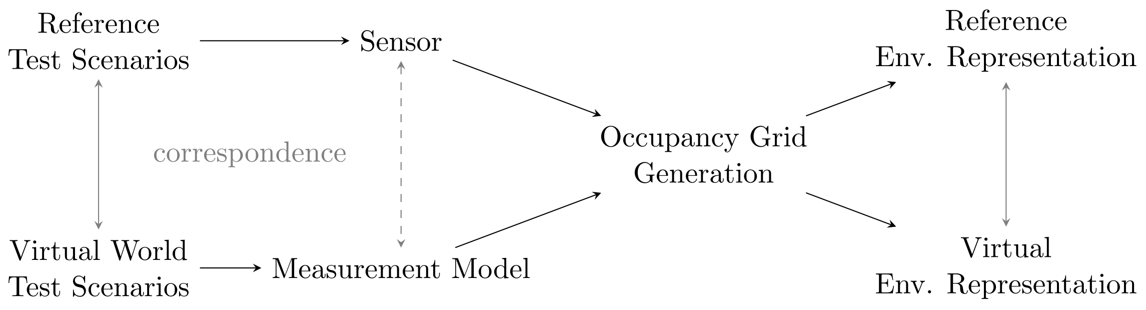

Reference Test Scenarios
Sensor
Reference Env. Representation
correspondence
Occupancy Grid Generation
Virtual World Test Scenarios
Measurement Model
Virtual Env. Representation

Figure 4.14: Outline of the principle behind the comparison of sensor model output data to real world reference data. Given a correspondence between reference test scenarios for real world test drives and virtual world test scenarios for virtual test drives, the low level output data in both cases can be given to the identical occupancy grid generation algorithm. The resulting environment representation in the reference case and the simulation should then show a close correspondence.

An outline of the flow of data for comparison of model and real world sensor output data is illustrated in Fig. 4.14. The starting point are reference test scenarios for real world test drives and corresponding virtual world test scenarios. The details of this correspondence depend on the manner of comparison used for the output data.

Generally two types of correspondences can be differentiated: statistical or exact. When the focus is on comparing statistical properties of the sensor output data, a statistical correspondence between the test scenarios is sufficient. The method chosen in e.g. reference [75] is the other one: With the addition of high-fidelity reference sensors, the environment including the time evolution of all object motion during a reference test drive is recorded as best as possible. From these data a virtual world scenario is generated that exactly matches the real world reference scenario within the limitations of the reference sensor's perception abilities.

With defined test scenarios, the low level output of the sensor in the real world testing as well as the corresponding output of the sensor model in the virtual environment testing are forwarded to the exactly same implementation of the occupancy grid generation algorithm. As has been discussed in the previous section, the component responsible for the occupancy grid generation receives the exact same input format in both cases. The resulting output of the occupancy grid map is then recorded as the reference environment representation in the real world test drive and the virtual environment representation obtained from the virtual environment. As highlighted by the double sided arrows in Fig. 4.14, there should be a close correspondence between the two recorded outputs matching the correspondence of the test scenarios in the beginning as long as the sensor measurement model accurately depicts the behavior of the sensor. To put it in different words: the fidelity of the sensor measurement model with regard to the relevant effects that influence the occupancy grid map generation can be tested qualitatively and potentially quantitatively by comparing the resulting occupancy grid map output data in a qualitative, respectively quantitative way.

At this point, it is important to note that the sensor measurement model has several parameters that can be adjusted to fit a specific type of sensor. Some of these parameters, e.g. the set of azimuth and elevation tuples in the scanning raster or the beam divergence, can be directly taken from the sensor's specification or independently measured in a lab environment. Other parameters like the variance of the distance noise added in post-processing are based on empirical observations of the sensor's behavior. It is important that the data set that is used to determine these parameters does not match the data set used for validation, and that at best another method is used to determine the model's parameters that does not involve the occupancy grid representation. This ensures that the sensor measurement model validation as described here remains valid.

A qualitative comparison between the perception processing chain in the real and virtual world is shown in Fig. 4.15. In both cases, similar but not identical scenarios are chosen: In the real world example a recorded test drive on a highway near Munich is used, while in the virtual world example a generic highway setup is used. The top row shows the scenario in a camera snapshot in Fig. 4.15 (a) for the real world and the rendering of the image generator in Fig. 4.15 (d) for the virtual world. For this point in time, the sensor output from the real lidar sensor located at the front of the ego vehicle and the sensor measurement model with matching parameters is shown in Fig. 4.15 (b) and (e) for the real and virtual world, respectively. In the real world example, reflections on drivable terrain have been filtered by the sensor while in the

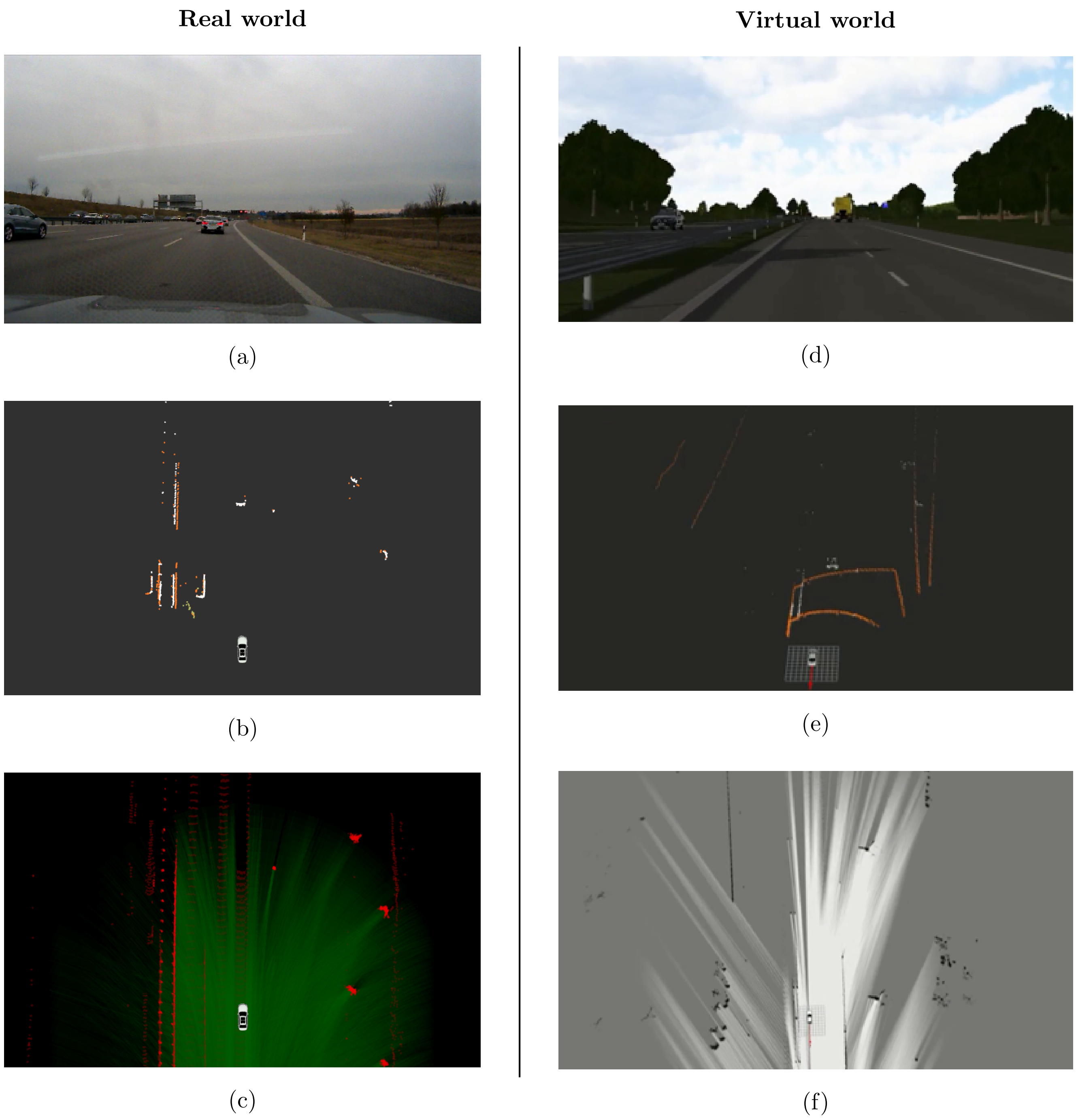


Figure 4.15: Side by side comparison between sensor data generation in the real world (left, a–c) and virtual world (right, d–f). On the real world side, (a) shows a camera snapshot of a highway scenario, (b) shows the point cloud data from a lidar sensor mounted on the front of the ego vehicle, and (c) shows the occupancy grid map generated from the point cloud data and previous measurements. On the virtual world side, a comparable but not identical scenario is shown: (d) shows the rendered snapshot of the virtual environment with similar highway scenario, (e) shows the point cloud as generated by the sensor measurement model, and (f) shows the occupancy grid map generated from the synthetic sensor data. Note that the color scheme for the occupancy grid map in the virtual world scenario is set differently and that reflections on drivable surfaces are shown in the sensor model's output, which have been filtered from the view in the real world sensor. A visual comparison between the real and virtual world data shows a clear correspondence, highlighting that the sensor measurement model has the ability to match the output of a real world sensor.

virtual world output all reflections points are shown. Aside from this filtering difference, the same features can be seen: the outline of other cars as well as the guard rails and the boundaries outside of the road. Finally, from these point cloud data in both cases an occupancy grid map is constructed, which is shown in Fig. 4.15 (c) and (f) for the real and virtual world, respectively. The drivable terrain, static obstacles outside of the road like trees and the guard rails are clearly visible in a similar manner in both cases. Note that moving objects are not present in this type of static occupancy grid map representation. The strong visual similarity between the occupancy grid map representations in the real and virtual world environments shows that, at least on a qualitative level, the original goal of providing low level sensor output data in real time from a virtual environment has been met by the sensor measurement model that has been the subject of discussion of this chapter.

4.5 Summary

In this chapter the construction of a sensor measurement model for an automotive lidar sensor based on a ray tracing approach is discussed. A sequence of actions for the sensor measurement model is given as well as a recipe for embedding the model in a driving simulation framework. The basic assumptions of the model are defined and several extensions added that are required to match all the information in the output format of low level point cloud data of a real lidar sensor. Using the sensor measurement model, a processing chain is defined that allows starting from the virtual environment of a driving simulator and passing through multiple processing steps in analogy to the sensing process to arrive at an occupancy grid map representation of the environment, which is an internal representation currently being investigates for internal use by the automated driving feature. In producing the low level sensor output data in the form of a lidar point cloud, the approach of the sensor measurement model allows to reuse the real world data processing chain of an automated driving system at a very early step. This is shown in an example with a side-by-side comparison of real world and virtual world perception resulting in an occupancy grid map generated from lidar point cloud data. This constitutes a major step towards increasing realism in sensor modeling as well as broadening the scope of applications for virtual environments in development and testing of automated driving systems.

5 Summary

The topic of this work is the simulated environment perception for automated driving systems. The underlying theme is the increasing rise of complexity in automated driving system, bringing new challenges for the development and testing of the systems. Within the scope of this work, the potential of the use of a virtual environment as a controlled environment for development and testing of automated driving systems is investigated with the focus on the system's environment perception and the sensors that perform this task. The goal is investigating chances and challenges in the simulation of automotive sensors and deriving guidelines that help in the implementation of such an simulation tool chain.

For this purpose, the different applications and the derived requirements for purpose-built sensor models are discussed. This results in a the identification and definition of different model types and the introduction of a generic simulation tool chain in chapter 2. This is followed by the discussion of statistical sensor error models in chapter 3, constituting a top-down development approach in the construction of a sensor model. In chapter 4, a bottom-up approach for the construction of a physically motivated sensor measurement model introduced and discussed in detail.

To start the discussion of virtual environment perception, the processing chain of an automated driving function is broken down with a focus on perception into different levels of abstraction. The levels of abstractions are identified from the well known V-model of system development: The system level corresponding to large scale simulation of an automated driving systems general behavior in a large number of driving scenarios. The sub-system level focusing on individual steps in the processing chain and separating the perception from the automated driving feature. Finally, the component level separating the perception process into the measurement step and subsequent perception processing to arrive at an internal environment representation for use by the automated driving system. For simulations that make use of sensors models, especially the sub-system and component levels are relevant. Based on this systematic approach, two types of sensor models are identified: the sensor error model for applications on the sub-system level and the sensor measurement model for applications on the component level.

Equipped with these definitions of the model types, a general tool chain for sensor simulation is introduced. In order to enable closed loop simulation that allows the real time testing of an automated driving system in a virtual environment, this tool chain features the following steps: Starting from the virtual environment of a driving simulator generating a standardized environment description, a sensor model performs virtual measurements resulting in synthetically generated sensor output data. In the next step, this output data in the standardized format of the sensor model is transformed into the specific format defined by the sensor that should be virtually replicated by the sensor model. From this point on, the tool chain is identical to a real world setup. Synthetic sensor data in the identical format as defined by the original sensor is processed by the automated driving system's perception processing and then used in the process of determining the vehicle's next actions.

A special emphasis is given to the standardized interfaces for input and output data of the sensor model. Typically, a variety of driving simulation frameworks as well as processing tool chains exist in the development of an automated driving system, resulting in severe difficulties for the development of multi-purpose sensor models. For this reason, the standardized open simulation interface (OSI) is introduced, which defines an interface for the description of object-level ground truth data as input for sensor error models as well as interfaces for object-level and low level sensor output data. Using the OSI interfaces in the construction of a sensor model gives the benefit of enabling easy re-usability of a sensor model implementation in a variety of different contexts. In addition, also using the OSI interface for the internal data structure of the sensor model, it is possible to easily source sensor model components from different developers and merge them to a single sensor model. This allows multi-source and parallel development on sensor models, which is a special feature of the design of the tool chain introduced in this work.

The sensor error models discussed in chapter 3 have the purpose to form a single model for the virtual representation of an automotive sensor that provides high level object data as its output. The sensor error model has two main design applications: Due to low complexity, it can reach a high computational performance that is required for a large scale simulations that include a wide variety of different scenarios. As it operates as a black box model on the object level interfaces of a sensor, constructing and using a sensor error model does not require intricate knowledge of the inner workings of a sensor, which in many cases is not readily available. Owing to the nature of its design applications, the sensor error model is a statistical model that aims to reproduce the statistical properties of the sensor output data.

Regarding sensor error models, a specific focus of this work is on the process of their construction and the model design. Due to the nature of a statistical model, it requires statistics obtained from observation data in order to derive the appropriate parameters required for the simulation of sensor behavior with regard to any quantity observed by the sensor. Obtaining these data requires a reference measurement for each quantity that is being investigated. For this reason, a modular approach in the design of sensor errors models is introduced in this work. The modular approach enables the building of individual modules that are executed in sequence, allowing each module to handle a different aspect of the sensing process. Moreover, this design enables an iterative development process for a sensor model, where each iteration can add an additional feature in the form of an extra module handling a specific aspect that has been analyzed and replicated in the framework of the model.

Based on the idea of the modular approach to sensor model construction, a classification hierarchy is introduced for different types of sensor errors. A sensor error is defined as any difference between the sensor output and the original ground truth data arising during the perception process, giving the sensor error model its name. The classification hierarchy sorts sensor errors in the categories of detection, direct perception, indirect perception, and packaging. The idea behind the classification is that some errors behave very similar while others have vastly different properties. As an example all sensor errors in the detection category concern the existence of objects in the sensor output, which includes effects such as limited range and field of view of the sensor as well as occlusion effects. The classification hierarchy also sorts the effects along the perception chain, giving a guideline with regard to the ordering of the sequence of modules that constitute the sensor model.

The sensor measurement model discussed in chapter 4 has the purpose to match the low level point cloud output generated by an automotive lidar sensor. Given the currently still largely experimental status of lidar sensors in the automotive context in combination with their high fidelity in the measurement of the three-dimensional structure of the vehicle's surrounding, the low level data for this sensor type is especially important for research purposes. Within the scope of this work, the goal in the construction of a sensor measurement model for a lidar sensor is the generation of a qualitatively realistic point cloud output that is generated in real time with a simulation framework. The synthetically generated point cloud data has to include all the relevant information that is also output by a real world lidar sensor in order to enable linking it to low level perception processing functions that are also used in real world test drives of research vehicles.

Within the scope of this work, the construction of the sensor measurement model for an automotive lidar sensor is based on a ray tracing based approach. The software tool chain for the implementation of a sensor measurement model is introduced using a driving simulation framework running the Nvidia OptiX ray tracing engine. The implementation of the model's components and their interplay with the ray tracing engine is described, as is the necessity of a custom-built database for material properties. Building the model in an iterative fashion, a basic ray casting, i.e. first-hit, approach is introduced and extended to include additional properties and effects observed in real world sensor data. With the inclusion of several model extensions, the full output data of a real world lidar sensor can be qualitatively reproduced, paving the way for the model's use in the study of low level perception processing functions. Utilizing the point cloud output data of the sensor measurement generated in real time, an environment representation in the form of an occupancy grid and occupancy grid map is generated. This constitutes the first reported use of synthetic low level lidar sensor data generated in real time by a driving simulation framework using a sensor measurement model for the construction of an environment representation used for internal processing by an automated driving system.

Besides the construction of a sensor measurement model generating low level sensor data, a major task is the testing and validation of the sensor model in comparison to the original sensor. Due to the nature of the low level point cloud data, a direct comparison is largely impossible on a meaningful quantitative level as even small inconsistencies between the three-dimensional geometry in the virtual environment and the original scene affect the point cloud in a way that is hard to compensate in a direct comparison metric. However, the occupancy grid environment representation can be used as an abstraction layer that provides a unified interface for the comparison of real world to virtual world scenarios. While first preliminary investigations into this topic have been performed and reported in reference [75], this largely represents a topic for future research.

As an outlook for other future research opportunities building on the results discussed in this work, many open points in regards to virtual environmental perception remain. The obvious points are the extension of the models described in this work to include additional effects, deriving parameter from a proper data set of observation data, extending them to other types of sensors. This holds true for both the sensor error models and the sensor measurement model introduced within the scope of this work, which only scratches the surface and aims to provide a solid starting point for future efforts in this area. Another major point is the design of testing procedures for automated driving systems that includes the abilities of simulation based testing. In this context falls the design of a scenario database with a standard set of test scenarios, both for automated driving systems and for the validation of sensor models. On the point of sensor model validation, research is ongoing on a methodology and tool chain to match sensor model output against reference data and obtain quantitative measures for the model's performance. On a different note, the ever growing availability of large scale computing power in addition to an increasing availability of observed sensor data as a foundation allows research into completely different types of sensor models than the ones described in this work, e.g. based on machine learning techniques and automatically learning and enhancing themselves as additional training data becomes available. Finally, establishing a standardized tool chain for use by many involved parties has the potential to benefit every user of a simulation framework running virtual environment perception components. The definition of the open simulation interface (OSI) and its publication as an open source software is a starting point for further development, which is currently ongoing.

Bibliography

[1] Klaus Bengler, Klaus Dietmayer, Berthold Farber, Markus Maurer, Christoph Stiller, and Hermann Winner. „Three Decades of Driver Assistance Systems: Review and Future Perspectives.“ In: *IEEE Intelligent Transportation Systems Magazine* 6.4 (2014), pp. 6–22. ISSN: 1939-1390. DOI: 10.1109/MITS.2014.2336271.

[2] Hermann Winner. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort.* 2., korr. Aufl. ATZ-MTZ-Fachbuch. Wiesbaden: Vieweg + Teubner, 2012. ISBN: 978-3-8348-1457-9.

[3] SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.* Sep. 2016. URL: http://standards.sae.org/j3016_201609/.

[4] Sebastian Thrun et al. „Stanley: The robot that won the DARPA Grand Challenge.“ In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692. ISSN: 15564959. DOI: 10.1002/rob.20147.

[5] Chris Urmson, Chris Baker, John Dolan, Paul Rybski, Bryan Salesky, William Whittaker, Dave Ferguson, and Michael Darms. „Autonomous Driving in Traffic: Boss and the Urban Challenge.“ In: *AI magazine* 30.2 (2009), pp. 17–28. ISSN: 0738-4602. DOI: 10.1609/aimag.v30i2.2238.

[6] Julius Ziegler et al. „Making Bertha Drive: An Autonomous Journey on a Historic Route.“ In: *IEEE Intelligent Transportation Systems Magazine* 6.2 (2014), pp. 8–20. ISSN: 1939-1390. DOI: 10.1109/MITS.2014.2306552.

[7] Steven H. Bayless, Adrian Guan, Patrick Son, Sean Murphy, and Anthony Shaw. *Connected vehicle insights : trends in roadway domain active sensing. Developments in radar, LIDAR and other sensing technologies and impact on vehicle crash avoidance/automation and active traffic management.* 2013. URL: https://rosap.ntl.bts.gov/view/dot/3397 (visited on 11/01/2019).

[8] Matts-Åke Belin, Roger Johansson, Johan Lindberg, and Claes Tingvall. „The Vision Zero and its consequences.“ In: *Proceedings of the 4th International Conference on Safety and the Environment in the 21st Century.* 1997, pp. 23–27.

[9] Chris Urmson et al. „Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge.“ In: (2007). DOI: 10.1184/R1/6561125.v1. URL: https://kilthub.cmu.edu/articles/Tartan_Racing_A_Multi-Modal_Approach_to_the_DARPA_Urban_Challenge/6561125 (visited on 11/01/2019).

[10] Hermann Winner and Michael Schopper. „Adaptive Cruise Control.“ In: *Handbook of Driver Assistance Systems.* Ed. by Hermann Winner, Stephan Hakuli, Felix Lotz, and Christina Singer. Cham: Springer International Publishing, 2014, pp. 1–44. ISBN: 978-3-319-09840-1. DOI: 10.1007/978-3-319-09840-1_46-1.

[11] W. D. Jones. „Keeping cars from crashing." In: *IEEE Spectrum* 38.9 (2001), pp. 40–45. ISSN: 00189235. DOI: 10.1109/6.946636.

[12] M. Spies and H. Spies. „Automobile Lidar Sensorik: Stand, Trends und zukünftige Herausforderungen." In: *Advances in Radio Science* 4 (2006), pp. 99–104. ISSN: 1684-9973. DOI: 10.5194/ars-4-99-2006. URL: https://www.adv-radio-sci.net/4/99/2006/.

[13] Nathan Matsuda, Oliver Cossairt, and Mohit Gupta. „MC3D: Motion Contrast 3D Scanning." In: *2015 IEEE International Conference on Computational Photography (ICCP 2015)*. IEEE, 2015, pp. 1–10. ISBN: 978-1-4799-8667-5. DOI: 10.1109/ICCPHOT.2015.7168370.

[14] Kay Fürstenberg and Florian Ahlers. „Development of a Low-Cost Automotive Laser Scanner – The EC Project MiniFaros." In: *Advanced Microsystems for Automotive Applications 2011*. Ed. by Gereon Meyer and Jürgen Valldorf. Berlin and Heidelberg: Springer, 2011, pp. 149–158. ISBN: 978-3-642-21381-6. DOI: 10.1007/978-3-642-21381-6_15.

[15] Ulrich Hofmann and Mika Aikio. „Biaxial Tripod MEMS Mirror and Omnidirectional Lens for a Low Cost Wide Angle Laser Range Sensor." In: *Advanced Microsystems for Automotive Applications 2012*. Ed. by Gereon Meyer. Berlin and Heidelberg: Springer, 2012, pp. 323–332. ISBN: 978-3-642-29673-4. DOI: 10.1007/978-3-642-29673-4_30.

[16] Ulrich Hofmann, Joachim Janes, and Hans-Joachim Quenzer. „High-Q MEMS Resonators for Laser Beam Scanning Displays." In: *Micromachines* 3.2 (2012), pp. 509–528. DOI: 10.3390/mi3020509.

[17] Ulrich Hofmann et al. „Resonant biaxial 7-mm MEMS mirror for omnidirectional scanning." In: *Journal of Micro/Nanolithography, MEMS, and MOEMS* 13.1 (2013), pp. 1–9. DOI: 10.1117/1.JMM.13.1.011103.

[18] Cristiano Niclass, Kota Ito, Mineki Soga, Hiroyuki Matsubara, Isao Aoyagi, Satoru Kato, and Manabu Kagami. „Design and characterization of a 256 x 64-pixel single-photon imager in CMOS for a MEMS-based laser scanning time-of-flight sensor." In: *Optics express* 20.11 (2012), pp. 11863–11881. DOI: 10.1364/OE.20.011863.

[19] Ulrich Hofmann, Frank Senger, Joachim Janes, Christian Mallas, Vanessa Stenchly, Thomas von Wantoch, Hans-Joachim Quenzer, and Manfred Weiss. „Wafer-level vacuum-packaged two-axis MEMS scanning mirror for pico-projector application." In: *MOEMS and Miniaturized Systems XIII*. Ed. by Wibool Piyawattanametha and Yong-Hwa Park. Vol. 8977. SPIE Proceedings. SPIE, 2014, pp. 47–60. DOI: 10.1117/12.2038249.

[20] R. H. Rasshofer, M. Spies, and H. Spies. „Influences of weather phenomena on automotive laser radar systems." In: *Advances in Radio Science* 9 (2011), pp. 49–60. ISSN: 1684-9973. DOI: 10.5194/ars-9-49-2011.

[21] Reiner Katzwinkel, Richard Auer, Stefan Brosig, Michael Rohlfs, Volkmar Schöning, Frank Schroven, Frank Schwitters, and Ulrich Wuttke. „Einparkassistenz." In: *Handbuch Fahrerassistenzsysteme*. Ed. by Hermann Winner, Stephan Hakuli, and Gabriele Wolf. Vol. 10. Wiesbaden: Vieweg+Teubner Verlag, 2012, pp. 471–477. ISBN: 978-3-8348-1457-9. DOI: 10.1007/978-3-8348-8619-4_32.

[22] Georg Tanzmeister, Martin Friedl, Dirk Wollherr, and Martin Buss. „Efficient Evaluation of Collisions and Costs on Grid Maps for Autonomous Vehicle Motion Planning.“ In: *IEEE Transactions on Intelligent Transportation Systems* 15.5 (2014), pp. 2249–2260. ISSN: 1524-9050. DOI: 10.1109/TITS.2014.2313562.

[23] Georg Tanzmeister and Dirk Wollherr. „Evidential Grid-Based Tracking and Mapping.“ In: *IEEE Transactions on Intelligent Transportation Systems* (2016), pp. 1–14. ISSN: 1524-9050. DOI: 10.1109/TITS.2016.2608919.

[24] Sascha Steyer, Georg Tanzmeister, Christian Lenk, Vinzenz Dallabetta, and Dirk Wollherr. „Data Association for Grid-Based Object Tracking Using Particle Labeling.“ In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 3036–3043. DOI: 10.1109/ITSC.2018.8569511.

[25] Walther Wachenfeld and Hermann Winner. „Die Freigabe des autonomen Fahrens.“ In: *Autonomes Fahren*. Ed. by Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner. Vol. 116. Berlin, Heidelberg: Springer Vieweg, 2015, pp. 439–464. ISBN: 978-3-662-45854-9. DOI: 10.1007/978-3-662-45854-9_21.

[26] Ralph Rasshofer, Johann Rank, and Guangyu Zhang. „Generalized Modeling of Radar Sensors for Next-Generation Virtual Driver Assistance Function Prototyping.“ In: *12th World Congress on Intelligent Transportation Systems*. 2005, pp. 2511–2522.

[27] Stefan Bernsteiner, Zoltan Magosi, Daniel Lindvai-Soos, and Arno Eichberger. „Phänomenologisches Radarsensormodell zur Simulation längsdynamisch regelnder Fahrerassistenzsysteme.“ In: *16. Internationaler Kongress Elektronik im Fahrzeug*. VDI-Berichte. Düsseldorf: VDI-Verlag, 2013, pp. 639–650. ISBN: 978-3-18-092188-4.

[28] Stefan Bernsteiner, Zoltan Magosi, Daniel Lindvai-Soos, and Arno Eichberger. „Radar Sensor Model for the Virtual Development Process.“ In: *ATZelektronik worldwide* 10.2 (2015), pp. 46–52. DOI: 10.1007/s38314-015-0521-1.

[29] Robin Schubert, Norman Mattern, and Roy Bours. „Simulation von Sensorfehlern zur Evaluierung von Fahrerassistenzsystemen.“ In: *Fahrerassistenzsysteme und Effiziente Antriebe*. Ed. by Wolfgang Siebenpfeiffer. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, pp. 69–73. ISBN: 978-3-658-08160-7. DOI: 10.1007/978-3-658-08161-4_10.

[30] Nils Hirsenkorn, Timo Hanke, Andreas Rauch, Bernhard Dehlink, Ralph Rasshofer, and Erwin Biebl. „Virtual sensor models for real-time applications.“ In: *Advances in Radio Science* 14 (2016), pp. 31–37. ISSN: 1684-9973. DOI: 10.5194/ars-14-31-2016.

[31] Timo Hanke, Nils Hirsenkorn, Bernhard Dehlink, Andreas Rauch, Ralph Rasshofer, and Erwin Biebl. „Classification of sensor errors for the statistical simulation of environmental perception in automated driving systems.“ In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016, pp. 643–648. DOI: 10.1109/ITSC.2016.7795621.

[32] Marco Weiskopf, Christoph Wohlfahrt, and Albrecht Schmidt. „Integrationslösung zur Absicherung eines realen Radarsensors im Systemverbund mit der Hardware-in-the-Loop Testtechnologie.“ In: *Automative - safety and security*. Ed. by Herbert Klenk, Hubert B. Keller, Erhard Plödereder, and Peter Dencker. Vol. 240. Lecture notes in informatics. Proceedings. Bonn: Gesellschaft für Informatik, 2015, pp. 29–40. ISBN: 978-3-88579-634-3.

[33] Nils Hirsenkorn, Paul Subkowski, Timo Hanke, Alexander Schaermann, Andreas Rauch, Ralph Rasshofer, and Erwin Biebl. „A ray launching approach for modeling an FMCW radar system." In: *2017 18th International Radar Symposium (IRS)*. 2017, pp. 1–10. DOI: `10.23919/IRS.2017.8008120`.

[34] Nick Jakobi, Phil Husbands, and Inman Harvey. „Noise and the reality gap: The use of simulation in evolutionary robotics." In: *Advances in artificial life*. Ed. by F. Morán. Vol. 929. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer Berlin Heidelberg, 1995, pp. 704–720. ISBN: 978-3-540-59496-3. DOI: `10.1007/3-540-59496-5_337`.

[35] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. „USARSim: a robot simulator for research and education." In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 1400–1405. DOI: `10.1109/ROBOT.2007.363180`.

[36] Maximilian Miegler and Mirko Nentwig. „Testing of Piloted Driving on Virtual Streets." In: *ATZ worldwide* 117.9 (2015), pp. 16–21. DOI: `10.1007/s38311-015-0044-7`.

[37] Kilian von Neumann-Cosel, Marius Dupuis, and Christian Weiss. „Virtual test drive - provision of a consistent tool-set for [D,H,S,V]-in-the-loop." In: *Proceedings of the Driving Simulation Conference Monaco*. 2009.

[38] Erwin Roth, Tobias Dirndorfer, Kilian von Neumann-Cosel, Alois Knoll, Thomas Ganslmeier, Andreas Kern, and Christian Weiss. „Analyse und Validierung vorausschauender Sensormodelle in einer integrierten Fahrzeug- und Umfeldsimulation." In: *VDI-Berichte* 2104 (2010).

[39] Ulrich Lages, Martin Spencer, and Roman Katz. „Automatic scenario generation based on laserscanner reference data and advanced offline processing." In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2013, pp. 153–155. ISBN: 978-1-4673-2755-8. DOI: `10.1109/IVS.2013.6629463`.

[40] Robin van der Made, Martijn Tideman, Ulrich Lages, Roman Katz, and Martin Spencer. „Automated generation of virtual driving scenarios from test drive data." In: *24th International Technical Conference on the Enhanced Safety of Vehicles (ESV) National Highway Traffic Safety Administration*. Vol. 15-0268. 2015.

[41] Stephanie Prialé Olivares, Nikolaus Rebernik, Arno Eichberger, and Ernst Stadlober. „Virtual Stochastic Testing of Advanced Driver Assistance Systems." In: *Advanced microsystems for automotive applications 2015*. Ed. by Tim Schulze, Beate Müller, and Gereon Meyer. Vol. 44. Lecture Notes in Mobility. Cham: Springer, 2015, pp. 25–35. ISBN: 978-3-319-20854-1. DOI: `10.1007/978-3-319-20855-8_3`.

[42] Andreas Wagener and Roman Katz. „Automated scenario generation for testing advanced driver assistance systems based on post-processed reference laser scanner data." In: *Fahrerassistenzsysteme 2016*. Ed. by Rolf Isermann. Wiesbaden: Springer Vieweg, 2018, pp. 175–190. ISBN: 978-3-658-21443-2. DOI: `10.1007/978-3-658-21444-9_12`.

[43] M. Siegel. „Sensor modeling and simulation: can it pass the Turing test?" In: *2001 IEEE International Workshop on Virtual and Intelligent Measurement Systems (VIMS)*. IEEE, 2001, pp. 92–96. ISBN: 0-7803-6568-2. DOI: `10.1109/VIMS.2001.924908`.

[44] Sebastian Thrun. „Learning Occupancy Grid Maps with Forward Sensor Models." In: *Autonomous Robots* 15.2 (2003), pp. 111–127. DOI: `10.1023/A:1025584807625`.

[45] Horst Kloeden, Daniel Schwarz, Erwin M. Biebl, and Ralph H. Rasshofer. „Effectiveness study of cooperative sensor systems for VRU-safety.“ In: *2012 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2012, pp. 879–884. ISBN: 978-1-4673-2118-1. DOI: 10.1109/IVS.2012.6232170.

[46] Peng Cao, Walther Wachenfeld, and Hermann Winner. „Perception sensor modeling for virtual validation of automated driving.“ In: *it - Information Technology* 57.4 (2015). ISSN: 1611-2776. DOI: 10.1515/itit-2015-0006.

[47] Nils Hirsenkorn, Hakim Kolsi, Moez Selmi, Alexander Schaermann, Timo Hanke, Andreas Rauch, Ralph Rasshofer, and Erwin Biebl. „Learning Sensor Models for Virtual Test and Development.“ In: *11. Workshop Fahrerassistenz und automatisiertes Fahren.* 2017, pp. 115–124. ISBN: 978-3-00-055656-2.

[48] Tim A. Wheeler, Martin Holder, Hermann Winner, and Mykel J. Kochenderfer. „Deep stochastic radar models.“ In: *2017 IEEE Intelligent Vehicles Symposium (IV 2017)*. IEEE, 2017, pp. 47–53. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995697.

[49] Tom Ziemke, Dan-Anders Jirenhed, and Germund Hesslow. „Internal simulation of perception: a minimal neuro-robotic model.“ In: *Neurocomputing* 68 (2005), pp. 85–104. ISSN: 09252312. DOI: 10.1016/j.neucom.2004.12.005.

[50] Maik Keller, Jens Orthmann, Andreas Kolb, and Valerij Peters. „A Simulation Framework for Time-Of-Flight Sensors.“ In: *2007 International Symposium on Signals, Circuits and Systems*. IEEE, 2007, pp. 1–4. ISBN: 1-4244-0968-3. DOI: 10.1109/ISSCS.2007.4292667.

[51] Yufeng Liu, Rosemary Emery, Deepayan Chakrabarti, Wolfram Burgard, and Sebastian Thrun. „Using EM to Learn 3D Models of Indoor Environments with Mobile Robots.“ In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2001, pp. 329–336. ISBN: 1-55860-778-1. URL: http://dl.acm.org/citation.cfm?id=645530.655822.

[52] Dirk Hähnel, Wolfram Burgard, and Sebastian Thrun. „Learning compact 3D models of indoor and outdoor environments with a mobile robot.“ In: *Robotics and Autonomous Systems* 44.1 (2003), pp. 15–27. ISSN: 09218890. DOI: 10.1016/S0921-8890(03)00007-1.

[53] Kilian von Neumann-Cosel, Erwin Roth, Daniel Lehmann, Johannes Speth, and Alois Knoll. „Testing of Image Processing Algorithms on Synthetic Data.“ In: *2009 Fourth International Conference on Software Engineering Advances*. IEEE, 2009, pp. 169–172. ISBN: 978-1-4244-4779-4. DOI: 10.1109/ICSEA.2009.34.

[54] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. „The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes.“ In: *29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*. IEEE, 2016, pp. 3234–3243. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.352.

[55] T. Hanke, N. Hirsenkorn, B. Dehlink, A. Rauch, R. Rasshofer, and E. Biebl. „Generic architecture for simulation of ADAS sensors.“ In: *16th International Radar Symposium IRS 2015*. Ed. by H. Rohling. Cuvillier Verlag, 2015, pp. 125–130. ISBN: 978-3-9540-4853-3. DOI: 10.1109/IRS.2015.7226306.

[56] N. Hirsenkorn, T. Hanke, A. Rauch, B. Dehlink, R. Rasshofer, and E. Biebl. „A non-parametric approach for modeling sensor behavior.“ In: *16th International Radar Symposium IRS 2015*. Ed. by H. Rohling. Cuvillier Verlag, 2015, pp. 131–136. ISBN: 978-3-9540-4853-3. DOI: `10.1109/IRS.2015.7226346`.

[57] Timo Hanke, Nils Hirsenkorn, Carlo van Driesten, Pilar Garcia Ramos, Mark Schiementz, and Sebastian Schneider. *Open Simulation Interface: A generic interface for the environment perception of automated driving functions in virtual scenarios.* 2017. URL: `http://www.hot.ei.tum.de/forschung/automotive-veroeffentlichungen/` (visited on 01/06/2019).

[58] *Open Simulation Interface (OSI) Repository.* URL: `https://github.com/OpenSimulationInterface/open-simulation-interface` (visited on 01/06/2019).

[59] Nils Hirsenkorn. *Modellbildung und Simulation der Fahrzeugumfeldsensorik.* 1. Auflage. Elektrotechnik. München: Dr. Hut, 2018. ISBN: 978-3-8439-3645-3. (Visited on 11/01/2019).

[60] Kilian von Neumann-Cosel. „Virtual Test Drive: Simulation umfeldbasierter Fahrzeugfunktionen.“ Dissertation. München: Technische Universität München, 2014. URL: `https://nbn-resolving.org/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20140206-1126934-0-2` (visited on 11/01/2019).

[61] Martin H. Strobl. „SPIDER - Das innovative Software-Framework der BMW Fahrsimulation.“ In: *VDI-Berichte* 1745 (2003).

[62] Moustafa Alzantot, Supriyo Chakraborty, and Mani Srivastava. „SenseGen: A deep learning architecture for synthetic sensor data generation.“ In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops).* IEEE, 2017, pp. 188–193. ISBN: 978-1-5090-4338-5. DOI: `10.1109/PERCOMW.2017.7917555`.

[63] Turner Whitted. „An improved illumination model for shaded display.“ In: *Communications of the ACM* 23.6 (1980), pp. 343–349. ISSN: 00010782. DOI: `10.1145/358876.358882`.

[64] Robert L. Cook, Thomas Porter, and Loren Carpenter. „Distributed ray tracing.“ In: *ACM SIGGRAPH Computer Graphics* 18.3 (1984), pp. 137–145. ISSN: 00978930. DOI: `10.1145/964965.808590`.

[65] James T. Kajiya. „The rendering equation.“ In: *ACM SIGGRAPH Computer Graphics* 20.4 (1986), pp. 143–150. ISSN: 00978930. DOI: `10.1145/15886.15902`.

[66] A. T. Ott, M. Shalaby, U. Siart, R. Brem, T. F. Eibert, J. Engelbrecht, and R. Collmann. „System simulation of a localization system based on power level detection with distributed antennas.“ In: *Proceedings of the 5th European Conference on Antennas and Propagation (EUCAP).* IEEE, 2011, pp. 19–23. ISBN: 978-1-4577-0250-1.

[67] Robert Brem and Thomas F. Eibert. „Transmitters with multi-radiation sources for electromagnetic ray tracing.“ In: *2012 6th European Conference on Antennas and Propagation.* IEEE, 2012, pp. 673–677. ISBN: 978-1-4577-0920-3. DOI: `10.1109/EuCAP.2012.6206290`.

[68] R. Brem and T. F. Eibert. „Scattering behavior comparison for field and current based high-frequency approximation methods." In: *Proceedings of the 2012 International Conference on Electromagnetics in Advanced Applications*. IEEE, 2012, pp. 144–147. ISBN: 978-1-4673-0335-4. DOI: 10.1109/ICEAA.2012.6328610.

[69] Robert Brem and Thomas F. Eibert. „Multi-Radiation Center Transmitter Models for Ray Tracing." In: *IEEE Transactions on Antennas and Propagation* 60.7 (2012), pp. 3382–3388. ISSN: 0018-926X. DOI: 10.1109/TAP.2012.2196956.

[70] Robert Brem and Thomas F. Eibert. „Accurate multi-antenna receive signal computations for ray tracing." In: *2013 IEEE Antennas and Propagation Society International Symposium*. IEEE, 2013, pp. 1070–1071. ISBN: 978-1-4673-5317-5. DOI: 10.1109/APS.2013.6711195.

[71] Steven G. Parker et al. „OptiX: A General Purpose Ray Tracing Engine." In: *ACM SIGGRAPH 2010 Papers*. Ed. by Tony DeRose and Hugues Hoppe. SIGGRAPH '10. ACM, 2010, 66:1–66:13. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778803.

[72] *ZeroMQ: An open-source universal messaging library*. URL: https://zeromq.org/ (visited on 08/25/2019).

[73] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. „ROS: an open-source Robot Operating System." In: *ICRA workshop on open source software*. Vol. 3. 2009, p. 5.

[74] Alexander Schaermann, Andreas Rauch, Nils Hirsenkorn, Timo Hanke, Ralph Rasshofer, and Erwin Biebl. „Validation of vehicle environment sensor models." In: *2017 IEEE Intelligent Vehicles Symposium (IV 2017)*. IEEE, 2017, pp. 405–411. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995752.

[75] Timo Hanke, Alexander Schaermann, Matthias Geiger, Konstantin Weiler, Nils Hirsenkorn, Andreas Rauch, Stefan-Alexander Schneider, and Erwin Biebl. „Generation and validation of virtual point cloud data for automated driving systems." In: *IEEE ITSC 2017*. Ed. by IEEE Intelligent Transportation Systems Conference. IEEE, 2017, pp. 1–6. ISBN: 978-1-5386-1526-3. DOI: 10.1109/ITSC.2017.8317864.

List of Publications

T. Hanke, N. Hirsenkorn, B. Dehlink, A. Rauch, R. Rasshofer, and E. Biebl. „Generic architecture for simulation of ADAS sensors.“ In: *16th International Radar Symposium IRS 2015*. Ed. by H. Rohling. Cuvillier Verlag, 2015, pp. 125–130. ISBN: 978-3-9540-4853-3. DOI: 10.1109/IRS.2015.7226306.

Timo Hanke, Nils Hirsenkorn, Bernhard Dehlink, Andreas Rauch, Ralph Rasshofer, and Erwin Biebl. „Classification of sensor errors for the statistical simulation of environmental perception in automated driving systems.“ In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016, pp. 643–648. DOI: 10.1109/ITSC.2016.7795621.

Timo Hanke, Alexander Schaermann, Matthias Geiger, Konstantin Weiler, Nils Hirsenkorn, Andreas Rauch, Stefan-Alexander Schneider, and Erwin Biebl. „Generation and validation of virtual point cloud data for automated driving systems.“ In: *IEEE ITSC 2017*. Ed. by IEEE Intelligent Transportation Systems Conference. IEEE, 2017, pp. 1–6. ISBN: 978-1-5386-1526-3. DOI: 10.1109/ITSC.2017.8317864.

N. Hirsenkorn, T. Hanke, A. Rauch, B. Dehlink, R. Rasshofer, and E. Biebl. „A non-parametric approach for modeling sensor behavior.“ In: *16th International Radar Symposium IRS 2015*. Ed. by H. Rohling. Cuvillier Verlag, 2015, pp. 131–136. ISBN: 978-3-9540-4853-3. DOI: 10.1109/IRS.2015.7226346.

Nils Hirsenkorn, Timo Hanke, Andreas Rauch, Bernhard Dehlink, Ralph Rasshofer, and Erwin Biebl. „Virtual sensor models for real-time applications.“ In: *Advances in Radio Science* 14 (2016), pp. 31–37. ISSN: 1684-9973. DOI: 10.5194/ars-14-31-2016.

Nils Hirsenkorn, Paul Subkowski, Timo Hanke, Alexander Schaermann, Andreas Rauch, Ralph Rasshofer, and Erwin Biebl. „A ray launching approach for modeling an FMCW radar system.“ In: *2017 18th International Radar Symposium (IRS)*. 2017, pp. 1–10. DOI: 10.23919/IRS.2017.8008120.

Nils Hirsenkorn, Hakim Kolsi, Moez Selmi, Alexander Schaermann, Timo Hanke, Andreas Rauch, Ralph Rasshofer, and Erwin Biebl. „Learning Sensor Models for Virtual Test and Development.“ In: *11. Workshop Fahrerassistenz und automatisiertes Fahren*. 2017, pp. 115–124. ISBN: 978-3-00-055656-2.

Timo Hanke, Nils Hirsenkorn, Carlo van Driesten, Pilar Garcia Ramos, Mark Schiementz, and Sebastian Schneider. *Open Simulation Interface: A generic interface for the environment perception of automated driving functions in virtual scenarios*. 2017. URL: http://www.hot.ei.tum.de/forschung/automotive-veroeffentlichungen/ (visited on 01/06/2019).

Alexander Schaermann, Andreas Rauch, Nils Hirsenkorn, Timo Hanke, Ralph Rasshofer, and Erwin Biebl. „Validation of vehicle environment sensor models.“ In: *2017 IEEE Intelligent Vehicles Symposium (IV 2017)*. IEEE, 2017, pp. 405–411. ISBN: 978-1-5090-4804-5. DOI: 10.1109/IVS.2017.7995752.