Kmeid Saad, M.Sc.Eng.

Automotive Camera Modeling and Integration With Standardized Interfaces

Dissertation

to achieve the university degree of:

Doctor of Technical Sciences

submitted at

Graz University of Technology

Advisors:

Prof. Martin Horn, Graz University of Technology, Austria
Prof. Stefan-Alexander Schneider, Kempten University of Applied Sciences, Germany

Examiners:

Prof. Martin Horn, Graz University of Technology, Austria
Prof. Kazuo Ohzeki, Shibaura Institute of Technology, Japan
Prof. Daniel Watzenig, Graz University of Technology, Austria

Faculty of Electrical and Information Engineering
Institute of Automation and Control

Graz, October 2019

Statutory Declaration

Abstract

Various technological challenges still prevail when it comes to the development and validation process of advanced driver-assistance systems and autonomous functions. This dissertation aims to introduce new approaches and methods that still adhere to the process flow of the V-cycle, and to address some of the existing development challenges. To satisfy these approaches, a physical sensor model for an advanced driver assistance camera has been created, and in the context of the proposed modeling approach, pertinent levels of abstraction and relevant optical and image sensor effects are identified. Additionally, applied concepts and employed methods for the camera model parametrization are presented as well.

Moreover, this dissertation illustrates the effectiveness of hybrid-development and test strategies. For the development part, the usage of a camera model in training image-based neural network algorithms is illustrated. Besides, concerning the test part, several classical computer vision algorithms and image-based neural network algorithms are tested and evaluated in a dedicated but generic/modular framework. Finally, co-simulation frameworks are presented for the integration and coupling process of various software components (sensor models, highly automated functions and simulation software) with standardized interfaces like the functional mockup interface and the open simulation interface.

Dedication

This thesis is dedicated to my parents: my mother Nada and my father Youssef, both of whom endlessly sacrificed for the opportunity of pursuing my goals. It is because of them, I have been able to appreciate the value of education.

اهدي هذا الإنجاز الى

امي الحبيبه **ندى نبيه سعد** و أبي الغالي **يوسف كميد سعد**

"Whether in a virtual or real world, it is only through distortion and aberration that we can see the divine truth."

— Kmeid Saad

Acknowledgments

My quest to start working on my dissertation started around four years ago, and it was made possible by Prof. Dr. Stefan-Alexander Schneider, for whom I am very grateful. Throughout the many conversations that we have had, you always managed to direct me flawlessly and to encourage me to pursue my own research ideas. I am genuinely fond of your supervision and sincerely thankful.

I am as well very thankful for Univ. -Prof. Dipl. -Ing. Dr. techn. Martin Horn for his continuous advice and supervision throughout my entire study.

I would also like to take this opportunity to thank, Prof. Kazuo Ohzeki, from Shibaura Institute of Technology, and Prof. Daniel Watzenig, from Graz University of Technology, acting as second and third examiners for their support and valuable observations and remarks.

I also want to thank Dr. Michael Paulweber, Director of Global ITS Research & Technology, for his technical and financial support and his willingness to help on many occasions.

I also want to thank all members of Kempten University of Applied Sciences, and especially Petra Lutz and Robert Stix for facilitating the work on my dissertation by always being there for me and granting me access to necessary lab equipment whenever required.

Finally, a huge thanks goes to my family and friends who were always by my side, providing me with moral support whenever needed!

Contents

List of Figures

List of Tables

Acronyms

ABS Anti-lock Brake System.
ACC Adaptive Cruise Control.
AD Automated Driving.
ADAS Advanced Driver Assistance Systems.
ADC Analog to Digital Converter.
ADC Analog-to-Digital Converter.
ANN Artificial Neural Network.
ASIL Automotive Safety Integrity Level.
ATC Automatic Transmission Control.

BOM Bill of Material.

CAN Controller Area Network.
CCD Charge-Coupled Device.
CFA Color Filter Array.
CIF Camera Interface.
CMOS Complementary metal-oxide-semiconductor.
CNN Convolutional Neural Network.
CUM Camera Under Modeling.
CV Computer Vision.

DN Digital Number.
DNN Deep Neural Network.
DOE Design of Experiment.
DoF Degree of Freedom.
DSNU Dark Signal Non-Uniformity.

ECU Electronic Control Unit.
EU European Union.

FMI Functional Mock-up Interface.
FMU Functional Mock-up Unit.
FMUs Functional Mock-up Units.
FOV Field of View.

FPN Fixed Pattern Noise.
fps frames per second.

HAF Highly Automated Driving Functions.
HFOV Horizontal Field of View.
HIL Hardware-in-the-Loop.
HOG Histogram of Oriented Gradients.
HSL Hue, Saturation, Lightness.
HW Hardware.

IRCF InfraRed-Cut-Off Filter.
ISP Image Signal Processor.
ITS Intelligent Transportation System.

KITTI Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago.

LD Lane Detection.
LED Light-Emitting Diode.
LEDs Light-Emitting Diodes.
lidar Light Detection and Ranging (LiDAR), referred to as *lidar* throughout this dissertation.
LKA Lane Keeping Assist.

mAP mean Average Precision.
MFC Multi-Functional Camera.
MFCs Multi-Functional Cameras.

NN Neural Network.
NR Noise Reduction.

OSI Open Simulation Interface.

PA Parking Assist.
P-ACC Predictive Adaptive Cruise Control.
PD Pedestrian Detection.
PDF Probability Density Function.
PRNU Photon Response Non-Uniformity.
PSF Point-Spread Function.
PVB Polyvinyl butyral.
PWM Pulse Width Modulation.
px pixel, referred to as *px* throughout this dissertation.

radar Radio Detection and Ranging (RaDAR), referred to as *radar* throughout this dissertation.
RGB Red, Green Blue.
RMS Root Mean Square.
ROI Region of Interest.
rpm rotation per minute.

SIL Software in the Loop.
SSD Sum Squared Difference.
SV Surround View.
SVC Support Vector Classifier.
SVM Support Vector Machine.
SW Software.

TCP Transmission Control Protocol.
TCS Traction Control System.
TLD Traffic Light Detection.
TSD Traffic Sign Detection.

UDP User Datagram Protocol.

V2X Vehicle to X.
VAP Virtual AUTOTILE Platform.
VD Vehicle Detection.
VDS Video Data Stream.
vECU virtual Electronic Control Unit.
VFOV Vertical Field of View.
vHIL virtual Hardware-in-the-Loop.
VIL Vehicle-the-Loop.
VKITTI Virtual KITTI.
VP Virtual Platform.

1 Introduction

Whether on a local national, European, or global scale, road transport plays a vital role in ensuring a continuous flow of goods and personnel. A subject of various technological innovations, road transport continuously undergoes new inventions and discoveries to ensure comfortable mobility. Numerous studies and a copious amount of research rely on active safety systems to achieve these goals.

Organizations in the European Union (EU) mobility sector have signed a road safety pledge aimed at achieving zero traffic fatalities by 2050, and, according to the European Commission, "Safe Vehicles" is one way to achieve this vision, proposing to make some essential safety features mandatory, which include: Intelligent Speed Assistance, Autonomous Emergency Braking, and improved direct vision for trucks [1].

To validate a specific Automated Driving (AD) function, during the testing process, it is necessary to cover all situations that the system may encounter, e.g., highly complex traffic situations and intersections, harsh weather conditions and other unusual situations like a zebra crossing the road. One approach in covering these situations is with simulation, for example, at Waymo[1] each day, as many as 25 thousand cars drive eight million miles in simulation testing out new skills and refining old ones [2].

The following subsections provide an introduction to Advanced Driver Assistance Systems (ADAS) and their role in AD; additionally, motivation and related work is illustrated, and research objectives are highlighted.

[1]Waymo is a self-driving technology development company. It is a subsidiary of Alphabet Inc. Waymo originated as a project of Google before it became a stand-alone subsidiary in December 2016. https://waymo.com/

1.1 ADAS for Automated Driving

Sensor technologies, like camera, Radio Detection and Ranging (RaDAR), referred to as *radar* throughout this dissertation (radar), Light Detection and Ranging (LiDAR), referred to as *lidar* throughout this dissertation (lidar), and infrared sensors perform environment perception in order to, among other things, automatically execute situation assessment and action implementation. Automated driving, a type of active safety system, implies the capacity of a car to drive partly or fully by itself, in which at least some aspects of a safety-critical function (e.g., steering, throttle, or braking) occur without direct driver input. Today, ADAS covers numerous tasks such as monitoring, warning, braking, and steering, and are continuously becoming enablers for AD.

1.1.1 Development and Testing Process

ADAS/AD functions are considered multi-domain systems, as they profoundly interact with almost every functional domain in the vehicle; moreover, since the development focus is increasingly geared towards autonomous driving, ADAS functions are becoming more safety-related, and with that, the system's complexity is rapidly increasing. Despite these challenges, ADAS global market is constantly increasing and is expected to reach $ 60.14 billion by 2020 [3].

To cope with the inevitable complexity of this system, the development of ADAS functions should be executed in a structured and reproducible manner. For the development of electric/electronic based systems, the so-called V-Model is widely adopted. Primarily developed for software based functions, the V-Model reproduces the technical aspects of the project cycle as a "Vee," starting with user needs on the upper left and ending with a user-validated system on the upper right [4]. The V-model is a phase-driven development process; the process is not plotted linearly against the time axis, but in the shape of the letter "V" [5].

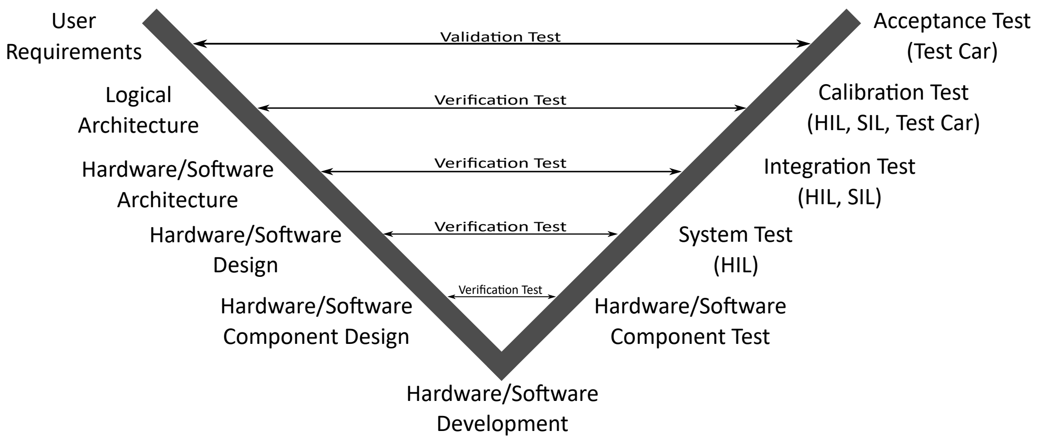

User Requirements
Validation Test
Acceptance Test (Test Car)
Logical Architecture
Verification Test
Calibration Test (HIL, SIL, Test Car)
Hardware/Software Architecture
Verification Test
Integration Test (HIL, SIL)
Hardware/Software Design
Verification Test
System Test (HIL)
Hardware/Software Component Design
Verification Test
Hardware/Software Component Test
Hardware/Software Development

Figure 1.1: V-Model for ADAS function development, adapted from [5].

Figure 1.1 represents an adapted V-model to cope with the technical domain of ADAS/AD functions; however, the general structure and concept of the V-model are preserved. The downward path of the V-Model contains the systematic analysis of user requirements and leads to the implementation of essential system components. The last step of the downward path simultaneously represents the first step in the upward path of the V-Model; during this step, the actual implementation development of all specified components is done. The upward path of the V-Model contains the systematic synthesis of the system and includes all testing and integration steps of single components and of the entire system.

Each part of the downward path has a corresponding part on the upward path. Test cases that tend to be formulated at each step of the downward path represent the links between the downward and upward paths.

Two methods are used to link each matching pair of the development process:

- **Validation**, *"Are we building the right system?"*
  - This step is used only for the link between user requirements and acceptance tests.
  - The method aims to ensure that all user requirements are fulfilled.

  - The validation test cases are the only ones that allow for the evaluation of the developed functions, and whether they fulfill all user requirements or not.
- **Verification**, *"Are we building the system correctly?"*
  - This step is used for any other pair of process steps in the V-Model.
  - The method aims to ensure that the specification of each process step in the downward path is fulfilled.
  - The purpose of verification is to ensure that the respective implementation adheres to specifications.

Due to the complexity of high-level ADAS functions, a toolchain is usually needed to ensure efficient and traceable development. Toolchains also enable automatic execution and evaluation of test cases. Additionally, in some situations where it is essential to keep the number of test cases low, engineers appeal to additional methods like Design of Experiment (DOE).

1.1.2 Modeling and Simulation

Modeling and simulation can be viewed as substitutes for physical experimentation. The computing power of modern computers allows users to calculate the results of a particular experiment based on a set of predefined modeling boundaries and abstraction levels. Models which contain all relevant parameters of desired systems are then integrated into a simulation environment where additional settings are set before starting the simulation, and eventually display the results.

In the context of ADAS, Fig. 1.2 represents a use case where test cases for ADAS/AD function development could benefit from the modeling and simulation process.

On the left of Fig. 1.2, it is shown how a test car, usually at the final step of the V-model, i.e., acceptance test, consumes a set of test cases and reports log the results. The feedback loop connecting the **Results** and the **Test Cases** blocks could, in some cases, lead to an endless repetition of the test cases, especially if certain combinations of the driver, vehicle, and environment parameters cannot be directly satisfied. For example, testing a Lane Keeping Assist (LKA) in foggy weather conditions, with deteriorated road marking on road segments, and at a distance that is always greater than 50 meters

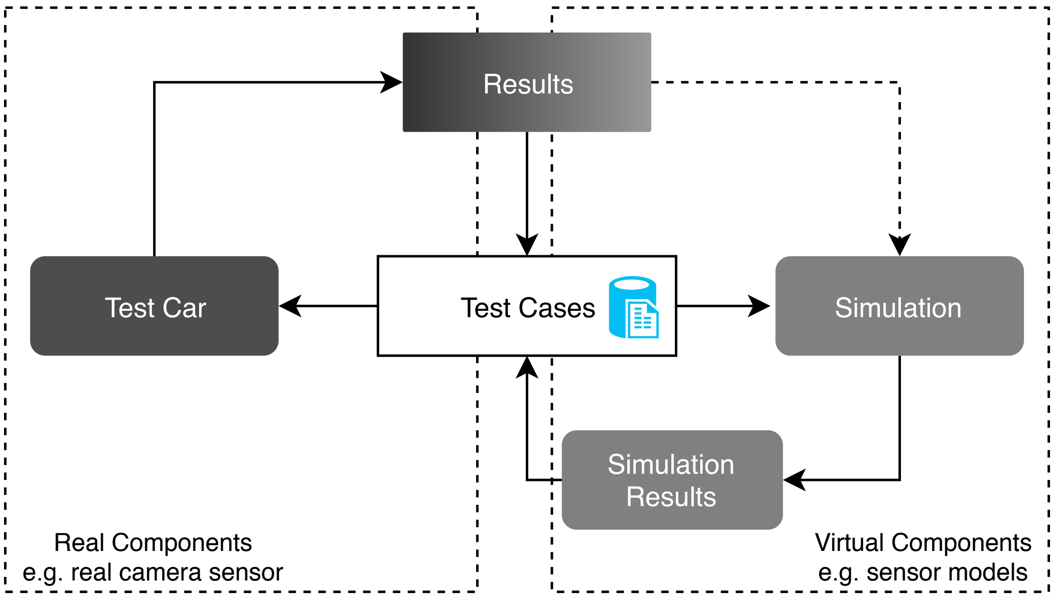


Figure 1.2: Modeling and simulation versus real tests.

between the ego vehicle and first-encountered traffic participant, could be very hard to satisfy throughout the entire test drive. In this context, only real components are used, i.e., real sensors, driver, vehicle, and environment.

On the right side of Fig. 1.2, it is shown how simulation could be used to execute several sets of test cases. In this setup, the feedback loop connecting the **Simulation Results** and the **Test Cases** blocks will not cause the system to loop endlessly or drastically increase the number of additional test cases; this is mainly due to the high flexibility of simulation environments. Theoretically, any combination of driver, vehicle, and environment parameters can be directly satisfied. In this context, virtual components are being employed, i.e., sensors and vehicle models, virtual drivers, and simulation environments.

As the advantages of modeling and simulation easily stand out in such a use case, the most critical prerequisite is the existence of reliable models that could be integrated in order to perform the underlying tasks.

"All models are wrong, but some are useful."

This quotation is from George Box, one of the greatest statistical minds of the 20th century [6]. When considering a specific model for ADAS development and AD, as George Box illustrated in his paper [7], the problem should be posed in the following form:

> For a model, there is no need to ask the question:
>
> ″Is the model true?″
>
> If ″truth″ is to be the ″whole truth″ the answer must be ″No.″
>
> The only question of interest is:
>
> ″Is the model illuminating and useful?″

By adopting this analogy, it should always be taken into consideration whether the integrated models reflect the necessary level of abstraction to successfully and reliably handle the issues at hand, i.e., is the generated camera model suitable for ADAS/AD applications?

Despite the possibility of creating highly accurate and detailed models, and even with regards to already existing models, developing/using them is subject to acceptable levels of complexity and to system runtime constraints. These constraints can also be seen during simulations, where a simulator re-enacts small subsets of its content for specific simulations as needed. The simulator's entire content is never activated all at once; only a small subset becomes active—that which is tailored to the constraints of the current situation [8].

1.1.3 Enablers for Hybrid Testing and Development

In Fig. 1.3, the inter-dependence between driver, vehicle, and environment are represented. Information exchange and interactions between these entities defines a vital role in the development of autonomous driving technology.

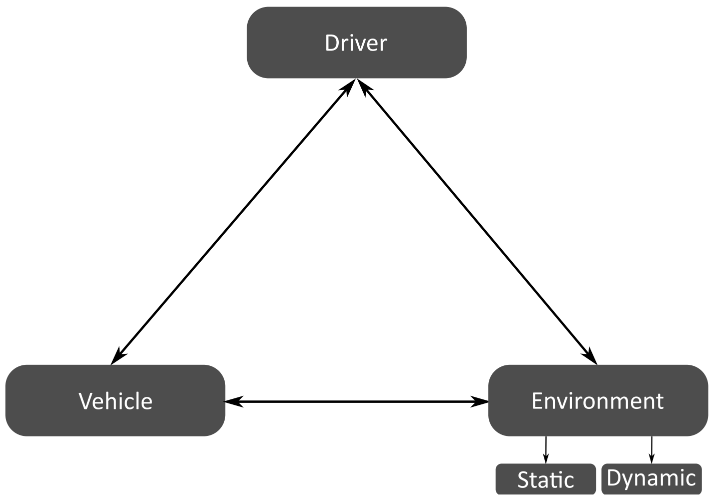


Figure 1.3: Driver-Vehicle-Environment system.

Currently, these relationships are still inseparable during the performance of a driving task. Vehicles equipped with ADAS rely on information (collected by various strategically mounted sensors) about the surrounding environment. Via human-machine interface, ADAS functions may utilize this information and command vehicle actuators. These relationships also illustrate the existing level of complexity in testing such a system, given that different uncontrollable parameters and factors are at stake.

Figure 1.4 represents a decomposition diagram showing some underlying components of the **Environment** block where ADAS/AD functions are meant to operate safely.

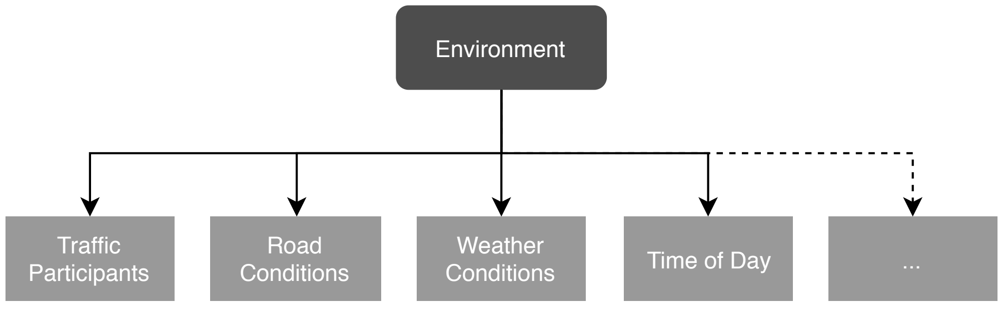


Figure 1.4: Environment decomposition.

In such an environment, one cannot guarantee similar behavior of traffic participants each time a test case is being executed. Moreover, complex interactions between boundary conditions such as different weather conditions make the repetition of particular test cases a challenging, if not impossible, task to achieve.

Perception, defined as the process of extracting scene representation from sensor inputs, plays a crucial role in obtaining an accurate representation of the surrounding environment, which is a crucial component for the correct development of ADAS/AD functions. Figure 1.5 shows how different factors may contribute independently and collectively to the complexity of scene variability.

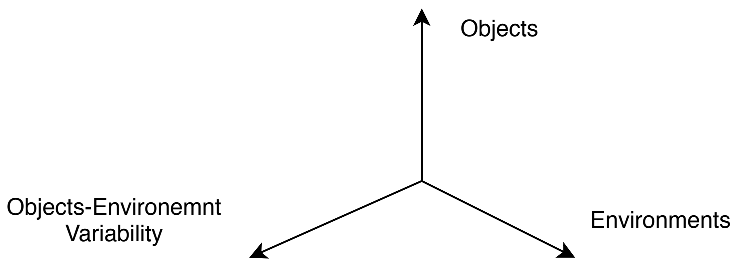


Figure 1.5: Dependency of perception variability.

The **Objects** axis represents the variability regarding the type of objects that could appear in a driving situation. This may include not only familiar objects, like other vehicles and pedestrians but also unexpected objects, like animals and disguised pedestrians, to be present in the driving environment. Figure 1.6 gives an idea of these occurrences.

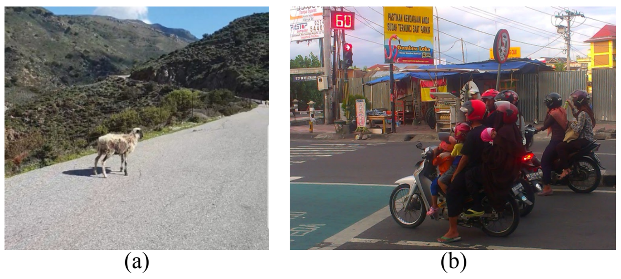


Figure 1.6: Unexpected objects on the road: (a) A sheep on the road, (b) Five people on a motorcycle.

As illustrated in Fig. 1.7, the **Environments** axis represents the variability in the surrounding environment, such as weather conditions and different times of the day.

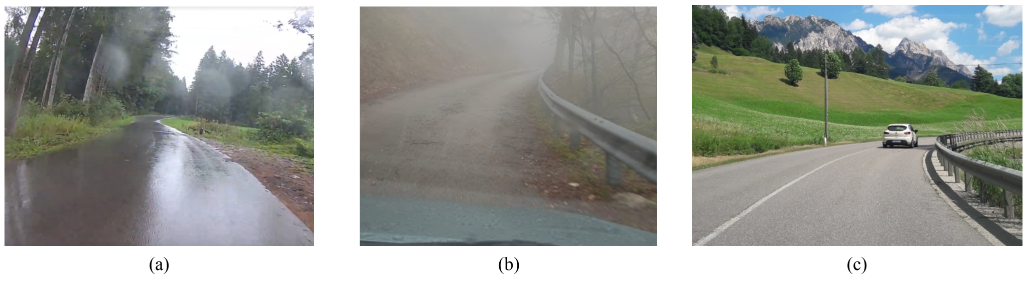


Figure 1.7: Variable Weather Conditions: (a) Rainy, (b) Foggy, (c) Sunny.

The last axis, **Objects-Environments Variability**, shows how different objects can coexist in different environments, creating a new dimension in which the complexity of perception increases. A person riding a horse is an excellent example of such a variability.

The driving activity cannot be restricted to a specific environment with a specific number and type of objects. Because vehicles are active on the road in different weather conditions, encountering objects and situations that cannot be controlled, ADAS and AD functions should develop reliable mapping functions from sensor data for a correct and pertinent representation.

Developing and testing such mapping functions is challenging on its own. Currently, the developing and testing steps still heavily rely on real sensor data, data that has been recorded on the streets during real test drives. The main drawback of this method is the lack of control over the driving environment. In other words, it is impossible to dictate weather conditions or the number and behavior of traffic participants. Additionally, studies have shown that even when dealing with a relatively controllable environment where the perception's variability is not so high (for instance driving on the motorway), ADAS functions still require a proof distance of 240 million km when considering the distance of 12 million km between two accidents involving personal injuries on German motorways and Poisson's statistical distribution [5].

Referring to the V-model illustrated in Fig. 1.1, another challenge appears in the fact that the validation process is not possible until the last step of the development cycle. For automated driving systems, the functionality is derived from the initial customer requirements, the very first step of the V-model. This means that any inaccurate or misinterpreted customer requirement will propagate undetected through all intermediate steps of the V- Model and all the way to the Implementation phase. Moreover, as pointed out in [13], another challenge is the absence of efficient feedback and information flow between different stages of the V-model.

All of the above-mentioned factors emphasize the need for hybrid development and testing strategies, where a combination of real and virtual setups provides a substitute for elaborate real tests in the development process. Hybrid development and test strategies would enable us to relax the primary constraints of the V-model by incorporating, stepwise, several internal loops along different levels of the V-model. Figure 1.8 illustrates this concept.

In addition, this creates the need for sensor models that are vital for virtual perception development and validation, and for integrating co-simulation tools to cover the overall implementation.

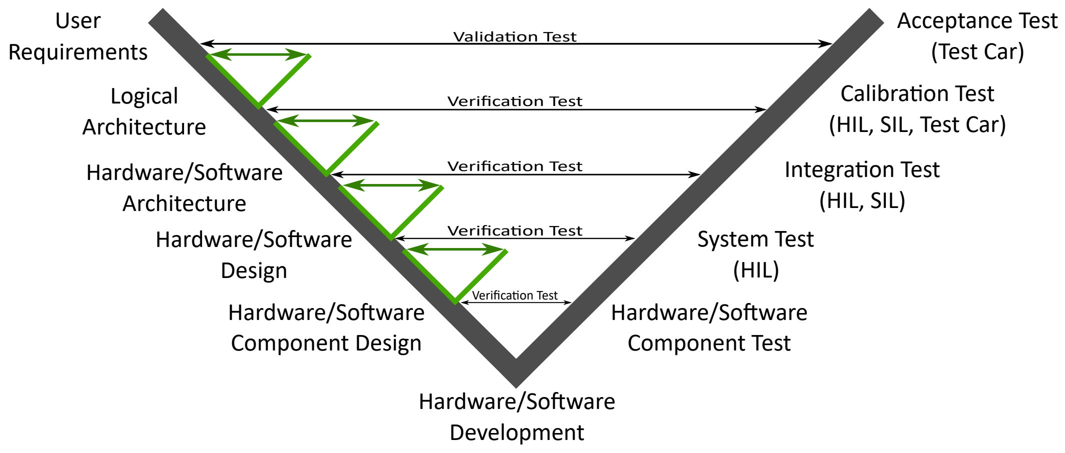

User Requirements
Validation Test
Acceptance Test (Test Car)
Logical Architecture
Verification Test
Calibration Test (HIL, SIL, Test Car)
Hardware/Software Architecture
Verification Test
Integration Test (HIL, SIL)
Hardware/Software Design
Verification Test
System Test (HIL)
Hardware/Software Component Design
Verification Test
Hardware/Software Component Test
Hardware/Software Development

Figure 1.8: Loops in a stepwise fashion in the V-Model.

1.1.4 Co-Simulation and Model Integration

Cyber-Physical Systems (CPS) are integrations of computation with physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [14]. From this definition, ADAS in their central core can also be considered CPS. Additionally, ADAS are multi-domain systems, meaning that different experts from varied disciplines need to work together simultaneously in order to reach the target goal. Though modeling and simulation (presented in section 1.1.2) are crucial for virtual development and testing, it is also essential for a multi-disciplinary solution to create a development process that could enable a heterogeneous model-based approach between different tools [15].

One possible solution is a co-simulation framework, where different teams can produce their conventional models and carry out their usual mono-disciplinary analysis. In addition, different models can be coupled together with the help of standardized interfaces, like the Functional Mock-up Interface (FMI)[2], which will also ease co-simulation between different tools [15], [16].

1.2 Motivation and Related Work

In section 1.1, it was discussed how ADAS/AD function development and testing procedures are executed using the conventional V-model, and how, with the help of modeling and simulations, numerous test cases could be efficiently driven and executed. Additionally, the need for hybrid test and development strategies, which could be embedded in a co-simulation framework with standardized interfaces, was emphasized.

Figure 1.9 represents a proposed co-simulation framework where simulation platforms like CarMaker[3], VIRES[4], and PreScan[5] are coupled with a virtual

[2]A tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code. https://fmi-standard.org/

[3] IPG Automotive develops software and hardware solutions for the application areas advanced driver assistance systems, automated and autonomous driving. https://ipg-automotive.com

[4] VIRES Simulationstechnologie GmbH provides simulation solutions for the automotive, railroad and aerospace industries https://vires.com

[5] PreScan is a physics-based simulation platform that is used in the automotive industry for development of Advanced Driver Assistance Systems (ADAS)

development and testing environment via standardized interfaces. In Fig. 1.9, data exchange between all block components is enabled via standardized interfaces, i.e., the FMI and the Open Simulation Interface (OSI)[6], that respectively ease co-simulation and facilitate information-transfer between automated driving functions and simulation platforms.

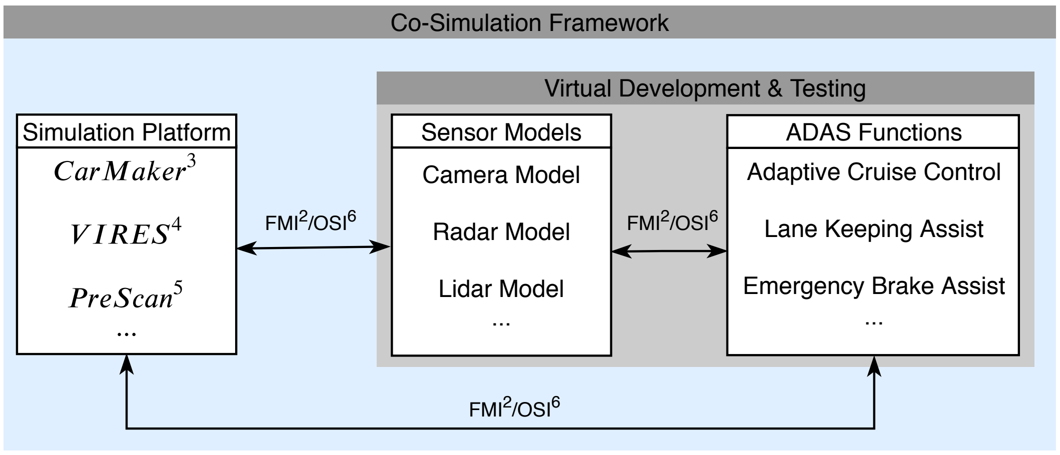


Figure 1.9: Proposed co-simulation framework.

Advanced driver assistance cameras are currently the primary computer vision solution for ADAS, and due to the complexity of driver assistance functions, it is necessary to evaluate their performance at an early stage of the development cycle. The availability of a comprehensive approach, one where a camera model is generated and implemented with standardized interfaces to test and analyze its effect on computer vision applications in virtual environments, is currently non-existent.

In the following section, work related to the usage of co-simulation in the automotive industry, ADAS/AD in particular, and to the development of advanced driver assistance camera models is presented.

that are based on sensor technologies such as radar, laser/lidar, camera and GPS. https://tass.plm.automation.siemens.com prescan

[6]Contains an object based environment description using the message format of the protocol buffers library developed and maintained by Google. https://www.hot.ei.tum.de/forschung/automotive-veroeffentlichungen/, https://github.com/OpenSimulationInterface/open-simulation-interface

1.2.1 Co-Simulation and Standardized Interfaces

Virtual design provides proof-of-concept and the possibility to test scenarios which are complicated and often impossible or too expensive to perform in real life, such as re-creating certain weather conditions and/or a simulated failure of sensor components (such as camera, lidar, radar). Co-simulation platforms enable virtual development and evaluation of ADAS in distinctive test cases and scenarios. By enabling various integrations of vehicle, environment, and sensor models, several automated driving functions could be tested and evaluated in reproducible simulations. In the following, related work for the usage of virtual platforms and co-simulation frameworks in automotive and ADAS/AD function development areas is presented.

In [17], Artuñedo *et al.* showed that in order to address the gap between traffic simulators and vehicle dynamics simulators, a co-simulation framework is necessary for the development of applications in the autonomous-driving field. Implemented in MATLAB Simulink, this approach allows Intelligent Transportation System (ITS), researchers, and developers to test onboard vehicle equipment such as sensors, actuators, or controllers, in addition to cooperative transport maneuvers in realistic urban scenarios [17].

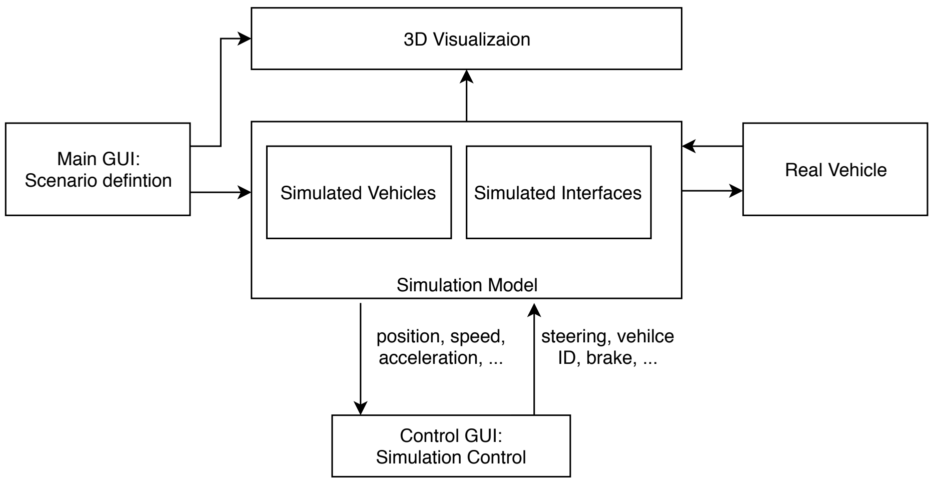


Figure 1.10: Framework diagram, adapted from [17].

As represented by the functional diagram in Fig. 1.10, the framework is structured in various yet modular components that increase its flexibility. In this approach, numerous models and components can be implemented at different levels in the simulation environment.

In another publication, [18] provides another co-simulation framework between three different tools. In order to evaluate the performance of automated driving functions under realistic traffic conditions, a co-simulation framework is used to handle the interaction between a microscopic traffic simulation and a highly detailed vehicle simulation.

Three tools integrated in a co-simulation framework were used: PTV VISSIM[7], IPG CarMaker as a vehicle simulation tool, and MATLAB Simulink[8] to demonstrate a Predictive Adaptive Cruise Control (P-ACC) case study. The third tool, VISSIM is a microscopic discrete traffic simulation system modeling motorway traffic as well as urban traffic operations [19]. To further highlight the advantages of Vehicle to X (V2X) based prediction model within the P-ACC, the approach is compared to a P-ACC with constant disturbance prediction. Thus, in this dissertation, ADAS and AD function evaluation was made possible via a co-simulation framework that combines the advantages of all utilized tools, and therefore improves development and the testing process.

In [20], Bücs *et al.* proposed a joint framework to facilitate ADAS prototyping via full virtualization and whole-system simulation. The proposed technique extends the boundaries of specialized simulation tools and models by providing means for cross-domain interconnection and joint control. By connecting the Hardware (HW)/Software (SW) simulation with the virtual environment of a driving simulator, the multi-domain approach can be used to fulfill the ISO 26262 requirement of full-system validation. At the heart of the system, a highspeed Virtual Platform (VP) was presented supporting distributed, multicore, virtual Electronic Control Unit (vECU) configurations and FMI coupling. Lastly, the capabilities of the joint tools were shown by prototyping an Automatic Transmission Control (ATC) and a LKA application in various system configurations, thus enabling function developers to perform tests in various virtual environments [20]. This work presented a multi-domain co-simulation framework system to establish an

[7]PTV Vissim is a flexible traffic simulation software for simulating complex vehicle interactions realistically on a microscopic level. https://www.ptvgroup.com/en/solutions/products/ptv-vissim

[8]MATLAB Simulink is a graphical programming environment for modeling, simulating and analyzing multidomain dynamical systems. https://www.mathworks.com/products/simulink.html

entire virtual ADAS rapid prototyping environment.

In the following subsection, the usage of co-simulation is further extended to include standardized interfaces like the FMI and the OSI.

In [21] , the authors demonstrated how multi-domain co-simulation via FMI interconnect all involved domains of a heterogeneous system emphasizing the fact that VPs should be transformed into Functional Mock-up Units (FMUs). SystemC[9] based VPs were in this regard integrated into heterogeneous multi-domain vehicular simulation systems via the FMI standard. The flexibility and interchangeability of a co-simulation setup were demonstrated by the evaluation of three different ADAS algorithms using the Virtual AUTOTILE Platform (VAP) in a virtual Hardware-in-the-Loop (vHIL):

- Anti-lock Brake System (ABS).
- Traction Control System (TCS).
- Adaptive Cruise Control (ACC).

Furthermore, the paper presented the possibility of fault injection in a vehicular multidomain simulation in order to increase system robustness and close the functional safety gap of virtual HW/SW systems with a simulation-based evaluation, testing, and verification approach.

In [23], Krammer *et al.* illustrated the integration of SystemC/SystemC-AMS[10] into the FMI standard without any changes to the standardized SystemC or SystemC-AMS libraries. The main advantages discerned were an ease of integration in a co-simulation framework and expansion into more complex simulation scenarios (a two-part battery system use case was presented and demonstrated a high level of transportability).

[9]SystemC is a C++ class library that provides a mechanism for modeling hardware and software together at multiple levels of abstraction. This standard defines SystemC®1 as an ANSI standard C++ class library for system and hardware design. The general purpose of SystemC is to provide a C++ based standard for designers and architects who need to address complex systems that are a hybrid between hardware and software [22]

[10]This standard defines the Analog/Mixed-Signal extensions for SystemC® 1, as an ANSI standard C++ class library based on SystemC for system and hardware design including analog/mixed-signal elements. The general purpose of the SystemC AMS extensions is to provide a C++ standard for designers and architects, who need to address complex heterogeneous systems that are a hybrid between hardware and software. [24]

In [25], it was argued that due to the increasing complexity of AD functions, predefined test catalogs are not able to cover the necessary test space of millions of kilometers to cover all functions' safety aspects. Therefore, different test instances like Vehicle-the-Loop (VIL), Hardware-in-the-Loop (HIL), and most importantly at early developmental stages, Software in the Loop (SIL), as represented in Fig. 1.1, are needed in combination with a scenario-based approach in order to expand and transfer the test space into the virtual world. The authors provided a first reference implementation of the upcoming standard, ISO 23150 for standardized hardware sensor interfaces, for the common development of sensor models. Therefore, OSI, endorsed by the Pegasus[11] Project [26], is introduced [25]. OSI, is a generic interface for information exchange between function development frameworks and simulation environments, and together with FMI can be integrated into a co-simulation framework that enlarges the scope towards other test instances using sensor models for development, verification, and validation.

1.2.2 Types of Sensor Models

As previously mentioned, in order to meet the challenges facing ADAS/AD function development and validation, physical test drives must be extended with virtual development and testing setups. For a successful integration of the ego vehicle in a simulation environment, vehicle dynamic models, in addition to ADAS/AD functions and sensor models, should be provided. Well-developed models for vehicle dynamics and ADAS/AD already exist [3] [4] [5]. However, there is still a gap in reliable and well-defined sensor models. The following section presents an overview of the already existing approaches for sensor modeling in general and camera models in particular. Furthermore, current classification methods for sensor models are also presented.

[11]PEGASUS delivers the standards for the automation of the future. With the PEGASUS joint project, promoted by the Federal Ministry for Economic Affairs and Energy (BMWi), key gaps in the field of testing of highly-automated driving functions will be concluded by the middle of 2019, https://www.pegasusprojekt.de/en/home.

In [25], sensor models are classified into three different categories:

1. **Ideal Sensor Models**:
   Described as an extended sensor data sheet, this model type provides object lists as an output in the sensor's reference coordinate system, i.e., the effects of the mounting position and other technical specifications like Field of View (FOV) and resolution range on the outputted object list. In such types of sensor models, weather conditions do not affect the object list.
2. **Phenomenological Sensor Models**:
   These represent an extension of ideal sensor models, where the FOV is not a binary decision that leaves the sensor blind at the boundary of the technical specification, but a stochastic transition that takes into consideration various aspects like detection certainty, false positives, false negatives, and weather conditions.
3. **Physical Sensor Models**:
   These are tasked with taking into account physical sensor properties and generating raw data, including images, point clouds, or reflection lists as outputs. For a camera use case, the simulation environment provides the ideal rendered image as an input, and the camera model implements various effects, such as lens distortion, to produce an augmented/more realistic image output to be consumed by Highly Automated Driving Functions (HAF). Traffic sign detection is one example of this.

The ability to model one sensor type or another does not only depend on the necessity of such models but also on the available resources, knowledge, and information regarding the sort of sensor to be modeled. Figure 1.11 provides an overview of the relationship between the depth of the modeling process and sensor model types. Note that even though the graph is only informative, i.e., based on an approximation assumption, the gap difference between the modeling depth of types of sensor models can still be relied upon.

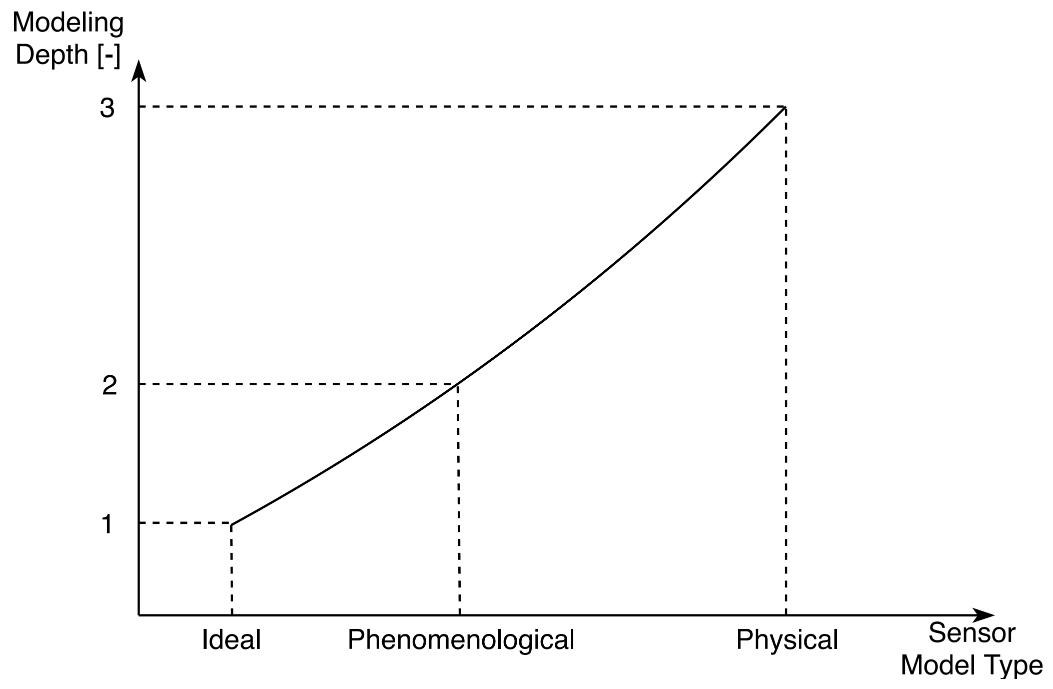


Figure 1.11: Modeling depth versus sensor model type.

The focus of this research work is on camera sensor models, so related work on available physical sensor models will be discussed in more depth. In the following, camera models like the one represented in [27] are not considered because although they reproduce a very accurate virtual image when compared to real camera output, they are subpar when it comes to operating in real time or even close to real time. In addition, research that only considers one specific camera effect that is modeled, like the modulation transfer function in [28] or blur model in [29], are also not included.

In [30], a numerical model for the Point-Spread Function (PSF) of an optical system that can efficiently model both experimental measurements and lens design simulations is presented. The mathematical basis for this model is a non-linear regression of the PSF with an Artificial Neural Network (ANN). Its main advantages lie in the portability and parametrization of this model, which yields to the possibility of applying this model to basically any optical simulation scenario. Figure 1.12 shows all parameters that were used as inputs to the ANN; defocus Δz, image height R and azimuth φ (used for training the ANN)).

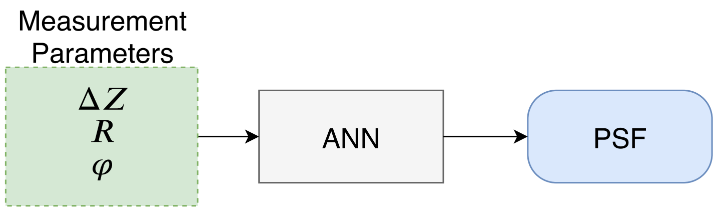


Figure 1.12: ANN for PSF model overview, adapted from [30].

A monochromatic Charge-Coupled Device (CCD) sensor, with a pixel size of 0.3070 μm in combination with 27 different lenses, was used to perform the required measurements. As the pixel size used for this experiment was too small (0.3070 μm—high resolution) a down-sampling procedure was applied to transform it into a resolution of 6 μm (lower resolution). Additionally, to obtain a better runtime, a bilinear interpolation technique was introduced to avoid generating PSFs for every pixel. The usage of this model lies in implementing the resulted PSF as a convolution kernel, thus applying it on ideal images to obtain a blurry, degraded image. Taking into consideration that a monochromatic sensor was used, neither the spectral resolution, the Color Filter Array (CFA) of the imager, nor the demosaicing process was handled. Additionally, all lens distortions are ignored.

In an attempt to reduce the gap between synthetic and real images, Carlson *et al.* introduced a physically based augmentation pipeline to vary sensor effects like chromatic aberration, blur, exposure, noise, and post-processing [31]. The modeling pipeline is illustrated in Fig. 1.13, where the second row of the image represents the augmentation pipeline that reflects the image formation process in a real camera (first row).

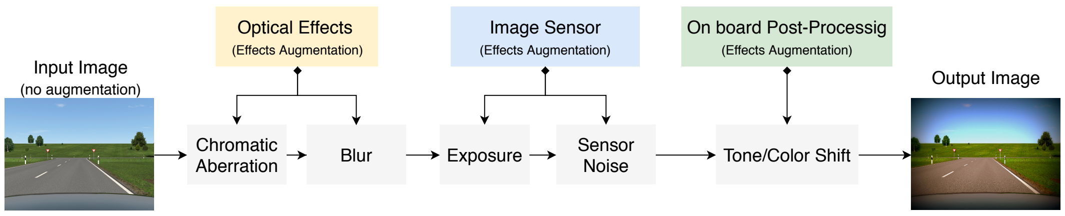


Figure 1.13: Schematic of image formation and processing pipeline, adapted from [31].

This paper also illustrated that synthetic image dataset augmentation with the proposed pipeline reduces the domain gap between synthetic and real domains for the task of object detection. In the proposed pipeline, all effects were modeled based on technical definitions from the literature, i.e., the proposed method lacks a sensor model parametrization procedure that can be applied to a specific advanced driver assistance camera.

Using Neural Network (NN), Carlson *et al.* extended their previous approach presented in [31] to propose a learned augmentation network composed of physically based augmentation functions. The proposed augmentation pipeline transfers specific effects of the sensor model—chromatic aberration, blur, exposure, noise, and color temperature—from a real dataset to a synthetic one [32]. The NN training pipeline is represented in Fig. 1.14, where the style loss trains the sensor effect parameter generators to effectively transfer the *sensor style* of the target dataset to the source dataset.

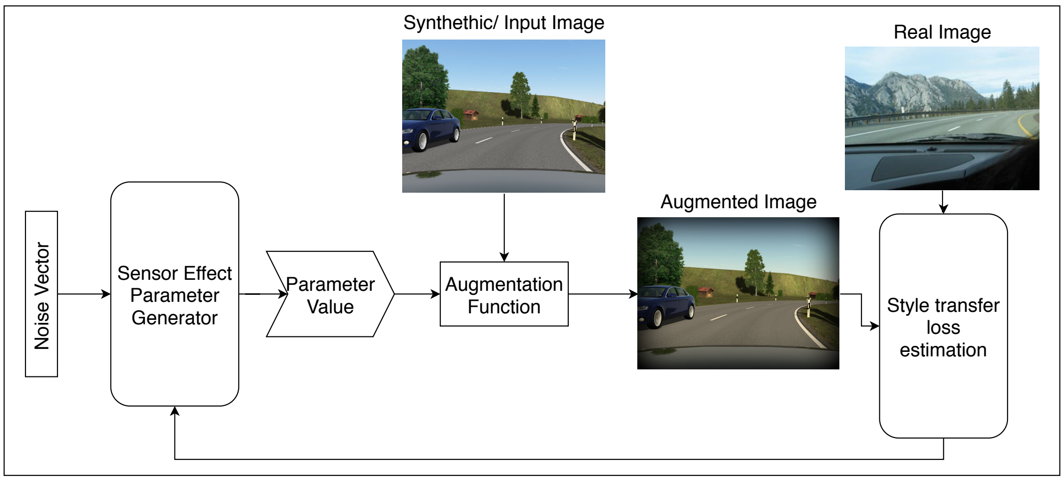


Figure 1.14: Schematic of the proposed sensor transfer network structure, adapted from [32].

Contrary to their previous method, here a sensor transfer network is developed, one that is capable of learning the optimal set of augmentations that would transfer sensor effects from a real dataset to a synthetic one. The training objective for each of these networks is to learn the distribution over its respective augmentation parameter(s) based on real data. Aside from the time needed for the sensor transfer network to learn the desired effects, this approach still heavily relies on real-world image data to successfully improve and implement different sensor effects on ideal synthetic images.

In [30]–[32], different approaches for physical sensor modeling were presented. Regardless of the number of modeled effects, none of the presented approaches included an implementation pipeline of its models.

1.3 Research Objectives and Outline

Due to the complexity and unpredictability of operating environments, many challenges still face the development and validation of ADAS/AD functions. Technological, safety, and economic aspects are in many ways constantly hindering the process flow presented in the V-Model. As other solutions may target these problems from different points of view, this thesis aims to address some of these challenges through virtualization and standardization. The following sections provide a more detailed explanation of the contributions made throughout this research and an outline for the rest of this thesis.

1.3.1 Thesis Contributions

The main contributions presented in this thesis can be summarized as follows:

- **Creating a physical sensor model for advanced driver assistance camera**. This contribution aims to extend the usage of sensor models, specifically a camera sensor model, to ADAS/AD function development phase. The proposed modeling approach identifies the necessary level of abstraction and relevant optical and image sensor effects, and implements them on a pre-rendered image provided by a simulation environment. Additionally, the proposed modeling approach presents the methods used and the applied concepts in camera model parametrization rather than just relying on technical definitions from the literature or on a massive number of real images for training NN as presented in 1.2.2.
- **Development of a hybrid test strategy for the evaluation of physical sensor models and ADAS/AD functions**. In order to evaluate the outputs of the camera model, a hybrid test strategy that represents a combination of real and virtual testing scenarios is developed. In this contribution, the camera model is coupled with classical Computer Vision (CV) and NN image-based AD functions. Among others, the

results show the effects of synthetic image augmentation using physics-based camera model on object detection algorithms.

- **Camera model integration in a co-simulation framework with standardized interfaces**. Creating a physical sensor model and a strategy to evaluate and test its effects on ADAS/AD functions development is not enough. In compliance with the motivation mentioned in section 1.2, the final contribution of this thesis is to implement the camera model in a co-simulation framework with standardized interfaces. In this regard, the Open Simulation Interface and the FMI were implemented in the developed camera model to show the advantages of virtual test drives by coupling the camera model with various HAF in a co-simulation framework.

1.3.2 Thesis Outline

This dissertation is divided into nine chapters. Chapter 1 provides an introduction to the main topics that are further analyzed and studied. Also, it explains the motivation behind this work and specifies all underlying contributions. Background information and definitions related to simulation software for virtual test driving, image based highly automatic functions, OSI, and FMUs are presented in Ch. 2. In Ch. 3, the main components and concepts behind a Multi-Functional Camera (MFC) and their role in ADAS/AD are presented, forming the necessary bedrock for the modeling methodology and modeling process applied in chapters four and five respectively. In Ch. 4, the modeling methodology is introduced with a focus on camera effects and their identification and classification, model parametrization, and correlation with target HAF. After MFC effects description and modeling in Ch. 5, Ch. 6 provides an overview of model implementation where a tool-chain is proposed for a co-simulation framework. In section. 6.4, the evaluation results of the modeling process are introduced. Chapter 7 demonstrates the application of an MFC model in various detection algorithms as well as in NN training, and finally Ch.8 concludes the dissertation and gives an overview that includes the usability of this approach in other use cases.

2 Background and Definitions

In this chapter, the aim is to provide background information about key elements, such as software tools and standardized interfaces, that are used throughout this dissertation. In addition, this chapter defines the context in which vision-based, highly automated functions are used, along with their relevance to camera sensor modeling and evaluation.

2.1 Simulation Software for Virtual Test Driving

As simulation starts to form a vital part of the regulatory framework affecting autonomous vehicles [33], virtual validation of ADAS and AD functions will not only depend on accurate and reliable sensor models, but also on the simulation software providing virtual representations of the driving environment.

Similar to real test drives, in order to perform a virtual test drive, the following main components are required:

1. A test vehicle that contains all parts of a real vehicle, e.g., powertrain, chassis, and sensors that are modeled via mathematical/physics-based approaches.
2. A test track that simulates a real course from real-life scenarios (digitized tracks) or even edge-case scenarios (computer modeled) purely for testing purposes.
3. A virtual test driver that simulates the actions of a real driver, including steering, braking, and shifting gears.

In this context, modeling and its implementation in SW is considered indispensable in incorporating all necessary elements for a virtual test drive. In a virtual test drive, computer-modeled representations of the required components seek to present themselves with a behavior that matches that of the real world.

Main benefits of ADAS/AD function development when using simulation software for virtual test driving lie in the ability to perform early stage testing. For example, in the case of the Electronic Control Unit (ECU), HIL testing can be performed by connecting real ECUs using I/O cards, Controller Area Network (CAN) bus, or other methods to a virtual vehicle instead of an actual one. Alternatively, SIL uses sensor models and software modeled controllers that can be integrated into the simulation software through a co-simulation environment.

Simulation software for virtual test driving like CarMaker[1], Vires-VTD[2], rFPro[3], and others are dedicated software to the testing and validation of automated driving. Aside from the technological and engineering advantages, simulation software has a direct impact on the financial aspect of the development phase, where early updates in design change before the production of mass produced vehicles or even prototypes are critical.

2.2 Image-Based Highly Automated Driving Functions

In Ch. 6.4 and 7, camera model evaluation and applications for different image-based functions are presented, and a separation between classical CV and NN-based functions is made; thus, this chapter introduces the general concept behind both approaches.

Vision-based ADAS/AD systems are rapidly growing research areas for autonomous driving; it is a cross/multi-disciplinary area encircling specific fields like computer vision, machine learning and NN. Most automotive manufacturers employ some form of a camera system in their models [34]. For example, Tesla's Autopilot had currently driven more than 1.6×10^9

[1]IPG CarMaker: `https://ipg-automotive.com/`
[2]VTD - Virtual Test Drive `https://vires.com/`
[3]rFPro: `http://www.rfpro.com/`

estimated km [35] on Hardware 1[4] and Hardware 2[5].

Figure 2.1 represents several vision-based applications, either intended for viewing applications like Surround View (SV), assistance functions like Parking Assist (PA), or more advanced automated functions like ACC.

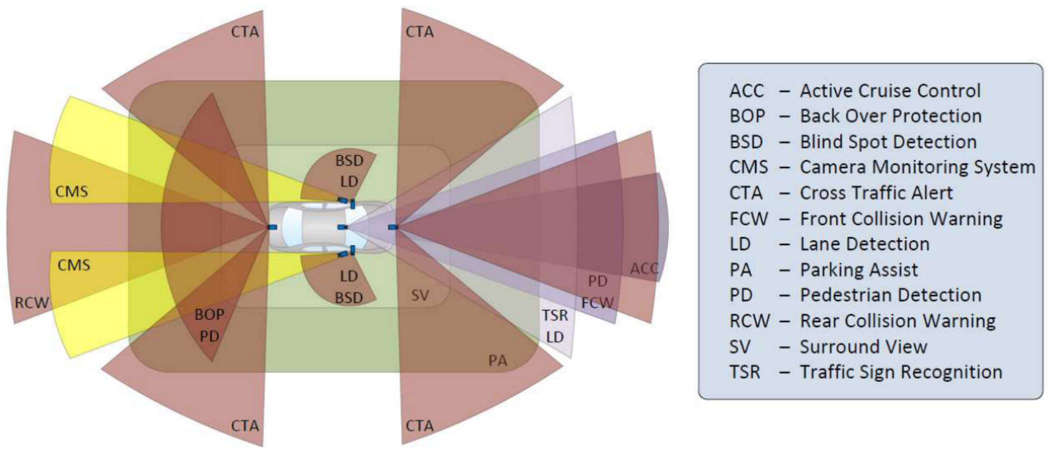

CTA
CTA
CMS
CMS
RCW
BSD
LD
LD
BSD
BOP
PD
SV
PA
TSR
LD
PD
FCW
ACC
CTA
CTA
ACC – Active Cruise Control
BOP – Back Over Protection
BSD – Blind Spot Detection
CMS – Camera Monitoring System
CTA – Cross Traffic Alert
FCW – Front Collision Warning
LD – Lane Detection
PA – Parking Assist
PD – Pedestrian Detection
RCW – Rear Collision Warning
SV – Surround View
TSR – Traffic Sign Recognition

Figure 2.1: Vision-based applications [34].

Widely considered the most versatile sensors and offering a broad range of applications, camera-systems are continuously stretching the limitations of CV, which differentiates between the following main tasks (adapted from [38]):

1. **Image Classification**:
   Defined as the task of assigning an input image with one label from a fixed set of categories, i.e., automatically assigning the "car" label for the image presented in Fig. 2.2 (a).
2. **Classification + Localization**:
   In addition to correctly classifying the input image, this task aims to locate the classified object inside the input image, i.e., to define its position inside the image with respect to a specific reference system.

[4]Hardware 1 refers to vehicles manufactured after late September 2014 that are equipped with a camera mounted at the top of the windshield, forward looking radar in the lower grille and ultrasonic acoustic location sensors in the front and rear bumpers that provide a 360-degree view around the car [36].

[5]Hardware 2 includes eight surround view cameras and 12 ultrasonic sensors, in addition to a forward-facing radar with enhanced processing capabilities[37]

Fig. 2.2 (b) shows how the ”car” is localized with the help of a yellow bounding box.

3. **Object Detection**:
   Refers to performing classification and localization on an input image that contains more than one object type. In Fig. 2.2 (c) object detection is performed on ”car,” ”Traffic Sign” and ”Pedestrian.”
4. **Instance Segmentation**:
   Resembles the object detection task; however, instead of localizing classified objects with bounding boxes, this task aims to label all the pixels that belong to each instance separately, see Fig. 2.2 (d).

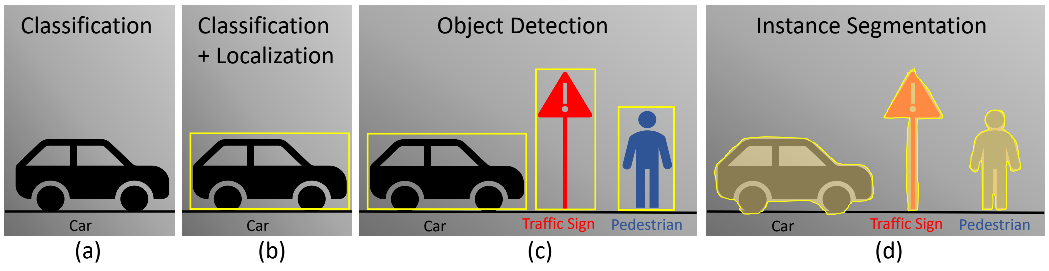


Figure 2.2: Computer vision tasks [38].

2.2.1 Based on Classical Computer Vision

For the purposes of this dissertation, classical CV algorithms refer to vision-based applications that primarily perform image manipulation using linear algebra and numerical techniques. Such techniques include, but are not limited to, matrix algebra and factorization methods, linear least squares, QR[6] decomposition for solving poorly conditioned least squares problems, iterative techniques like image derivatives and sliding-window approaches, and finally, Bayesian modeling and inference for solving problems that involve estimating some unknown model parameters from a given number of measurements.

For example, to estimate the motion between two or more consecutive frames (optical flow), an error metric should be chosen and implemented for frames comparison. Translational alignment is one method of achieving this by

[6]QR decomposition, also known as a QR factorization, is a decomposition of a matrix A into a product A = QR of an orthogonal matrix Q and an upper triangular matrix R.

merely shifting one frame relative to the other. Given a template image $I_0(x)$ sampled at discrete pixel locations $\{\boldsymbol{x_k} = (x_i, y_j)\}$, $i = 1, ..n$ $j = 1, ..m$, it's location in image $I_1(\boldsymbol{x})$ could be estimated by computing its displacement: $\mathbf{u}$. A least squares solution to this problem is to find the minimum of the $SumSquaredDifference(SSD)$ function [39]:

$$E_{SSD}(u) = \sum_{k=1}^{n\times m} \left[I_1\left(x_k + \mathbf{u}\right) - I_0\left(x_k\right)\right]^2 = \sum_{k=1}^{n\times m} e_k^2 \tag{2.1}$$

where:

$e_k = I_1(x_k + \mathbf{u}) - I_0(x_k)$ is called the *residual error* [39].

Additionally, various interpolation methods, bilinear or bicubic, and color correction techniques are applied to obtain better results.

For vision-based object detection tasks, a typical pipeline is to test for the presence of an object in a predefined bounding box at various positions and scales in the image. This technique, known as **sliding-window**, enables the bounding box of a certain size to slide on the image from top left to bottom right, and in most cases repeating with bounding boxes of different sizes; the size of targeted objects to be detected varies with respect to their distance from the camera. During the sliding window approach, and depending on the object of interest, several calculations are performed for each bounding box position. For example, in Fig. 2.3 histograms of two bounding boxes, red and orange, calculations are computed at all positions and are then compared with a reference histogram (in this case histogram of the bicycle) in order to detect the bicycle's position inside the image at different time stamps.

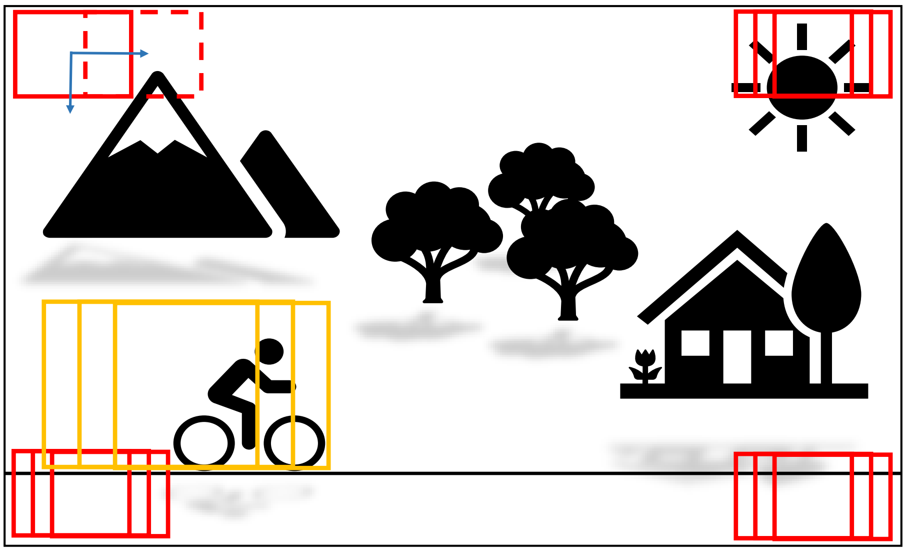


Figure 2.3: Sliding window approach.

This example does not entirely describe the detection pipeline, as several pre- and post-processing steps, such as noise compensation, dynamic references, image intensity equalization, and color-space conversion are not illustrated. The aim of this example is to show that with classical computer vision algorithms, especially those that rely on scanning the entire image or a large Region of Interest (ROI), vision-based applications usually rely on a set of predefined parameters, like histogram values, which are affected by the quality of the acquired image. Other detection algorithms like ”License Plate Detection” may have a much smaller ROI, restricted by the target vehicle and provided by other sensors like a radar sensor.

Prior knowledge on how these algorithms operate is a crucial point for understanding the effects of a camera model on vision-based applications, and it will be further illustrated in Ch. 3 and Ch. 4.

2.2.2 Based on Neural Networks

Due to driving environment variations, fully or partly autonomous vehicles necessitate the development of reliable algorithms that can generate environment models where vehicles can safely and autonomously navigate. In addition to classical computer vision algorithms (discussed in 2.2.1), engineers and researchers are more frequently focusing on NN based algorithms that possess learning capabilities for solving various computer vision tasks.

Driving environment exploration is typically fulfilled by data collection from various sensors like camera, lidar and radar. Via sensor fusion, the state of the ego-vehicle and its surrounding objects is not directly observed but instead deduced by several algorithms through a set of sensor readings; various autonomous driving algorithms would make use of these state vectors for path planning in such an environment [40]. With image-based NN algorithms, the state vectors are no longer a result of sensor fusion, but rather a sequence of snapshots acquired by the camera at different time stamps. This means that specific information like position, orientation, velocity and acceleration is no longer explicitly provided to the algorithm but rather deduced by a Deep Neural Network (DNN), more specifically a Convolutional Neural Network (CNN).

With deep learning, we do not program CNN to recognize these specific features. Rather, a CNN learns on its own how to recognize specific objects through forward- and back-propagation without being supplied with specific features to look for. In the case of vehicle detection, several hierarchy levels can be identified:

1. Simple shapes, like edges, curves and colored blobs/geometrical forms.
2. Complex objects, as a combination of simple shapes, like tail lamps, license plate, car doors and wheels.
3. The vehicle, as a combination of complex objects.

A CNN might have several layers, and each layer might capture a different level in the hierarchy of objects. The first layer is the lowest level in the hierarchy, where a CNN generally classifies small parts of the image into simple shapes like horizontal and vertical lines and simple blobs of color. In subsequent layers, higher hierarchical levels are considered, and generally more complex structures like shapes (combinations of geometrical shapes) are classified. Eventually, full objects (such as vehicles) are detected. To achieve this, several filters look at small pieces, or patches, of the image. Then a sliding-window (introduced in section 2.2.1) technique is applied to focus on different regions of the image.

In contrast to classical computer vision algorithms and non-convolutional NN, CNN groups adjacent pixels together and treats them as one entity. In doing so, the network takes advantage of the fact that pixels in an image are close together for a reason and have a specific meaning. This fact reflects directly on the effectiveness of using the output of physical camera models for advanced driver assistance cameras, and will be further elaborated in Ch. 7.

2.3 Functional Mock-up Interface

The Functional Mock-up interface (FMI[7]) is a tool independent standard for dynamic model exchange and more importantly for co-simulation. In order to improve the model-based design, FMI's primary goal is to support the exchange of simulation models between OEMs and suppliers.

The FMI standard consists of two main parts [15]:

1. **FMI for Model Exchange:** The intention is that a modeling environment can generate `C` code of a dynamic system model in the form of an input/output block that can be utilized by other modeling and simulation environments.
2. **FMI for Co-Simulation:** The intention is to couple two or more simulation tools in a co-simulation environment. The data exchange between subsystems is restricted to discrete communication points. During the time needed between two communication points, the subsystems are solved independently from each other by their individual solver.

For the scope of this dissertation, FMI for co-simulation (coupled with subsystem models that have been exported by their simulators together with its solvers as runnable code) has been used. Usually, FMI for co-simulation is contained in a Functional Mock-up Unit (FMU) which is a zip file with an extension `.fmu`.

[7]FMI development was initiated by Daimler AG with the goal to improve the exchange of simulation models between suppliers and OEMs[41].

The FMU contains the following main files [41]:

1. An `XML`-file, also referred to as a model description file, that contains the definition of all variables of the FMU in a standardized way.
2. `C` functions that represent the model equations. The `C` code for co-simulation is then distributed in binary form for one or several target machines, such as Windows dynamic link libraries `.dll` or Linux shared object libraries `.so`.

All information about a model and a co-simulation setup that is not needed during execution is stored in an `XML`-file called `modelDescription.xml`. The benefit is that every tool can use its favorite programming language to read this `XML`-file (e.g., `C`, `C++`, `C\#`, `C\#`, `Python`) and that the overhead, both in terms of memory and simulation efficiency, is reduced [15].

FMI for co-simulation is designed for both the coupling of simulation tools (simulator coupling, tool coupling) and coupling with subsystem models, which have been exported by their simulators together with their solvers as a runnable code.

FMI for co-simulation stand-alone will be used throughout the entire implementation. As represented in Fig. 2.4, in such use case the master and the slave run in the same process, where the master acts as an executable, and the slave contains the model as well as the solver. This enables the dynamic linking between FMU's libraries and simulation software.

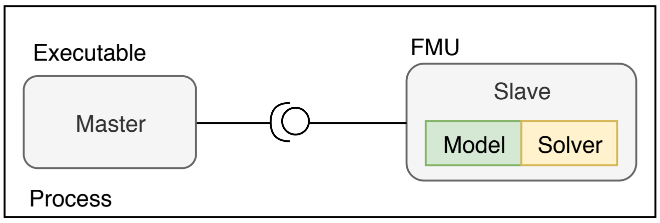


Figure 2.4: Single process stand-alone use case.

2.4 Open Simulation Interface

OSI, officially a part of the PEGASUS project, aims to develop a procedure for testing of automated driving functions in order to facilitate the rapid implementation of automated driving into practice. The scope directly aligns with the desire of the German automotive industry to establish a standardized interface in the field of testing of, and experimentation on, higher levels of automation [26].

OSI is a generic interface that enables the connection of automated driving functions with a variety of driving simulation frameworks in order to achieve easy and straightforward compatibility. The vision is to be able to connect any automated driving function to any driving simulator with ease while integrating a variety of sensor models, thus strengthening the usefulness of virtual testing. OSI contains an object-based environment description using the message format of the protocol buffers library developed and maintained by Google. Ensuring modularity, integrability, and interchangeability of individual components, the interface defines several top-level messages that are used to exchange data between separate models [42].

OSI defines top-level messages that are used to exchange data between separate models. Among others, main messages define:

1. **GroundTruth** interface which provides an exact view on the simulated objects in a global coordinate system, the world coordinate system. This message is populated using the data available internally and then published to external subscribers by a plugin within the driving simulation framework.
2. **SensorView** that is derived from GroundTruth and used as input to sensor models. The SensorView information is supposed to provide input to sensor models for simulation of actual real sensors; all information regarding the environment is given with respect to the virtual sensor coordinate system.
3. **SensorData** interface which describes the objects in the reference frame of a vehicle for environmental perception. It is generated from GroundTruth/SensorView messages and can be used to connect to an automated driving function using simulated data directly.

Figure 2.5 illustrates a possible use case where OSI messages can be generated and consumed by separate tools and models. The "Environment Simulation" block, usually a simulation software, provides the **GroundTruth** and **SensorView** messages as an output to be utilized by various sensor models like camera, lidar or radar. **SensorView** message may internally contain several packages of sensor view data corresponding to different virtual sensors employed in a test scenario. ”Sensor Models” on their turn process these messages and output one or several **SensorData** messages that can either be directly consumed by ADAS/AD functions or merged by a ”Fusion Model” and provided as a single **SensorData** message for the ”ADAS/AD Functions” block.

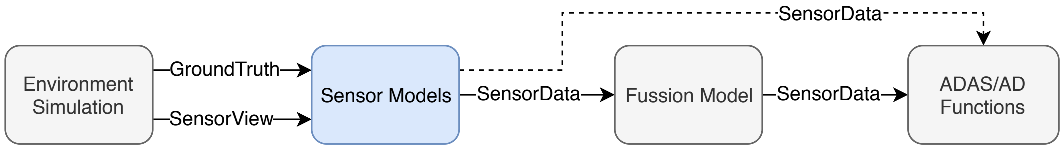


Figure 2.5: OSI top-level interfaces.

3 Multi-Functional Camera

This chapter provides a general introduction about the MFC and a more detailed description of the MFC's system architecture and optical path.

3.1 Introduction

Front view Multi-Functional Cameras (MFCs) are primarily designed for static and dynamic environment detection and recognition of all relevant road users, road markings, and road signs. Driving environments consist of various traffic participants, road marks, traffic signs, and others that are highly suited to human visual perception, which, from an automotive sensor's perspective, can directly be integrated into camera systems. MFCs' ability to enable versatile ADAS and AD function implementation makes them essential components for virtual test environments. In other words, camera models are imperative.

MFCs are usually mounted either behind the windshield of the vehicle, close to the rearview mirror, or integrated into the rearview mirror, and thus they are projecting a wide FOV and protected from external weather conditions. The FOV of an MFC is defined by the optical module and the image sensor. In Fig. 3.1, the FOV is divided into Horizontal Field of View (HFOV) and Vertical Field of View (VFOV). Based on the targeted ADAS functions (e.g., detection of large lane curvature), an HFOV angle larger than 40° is recommended [43].

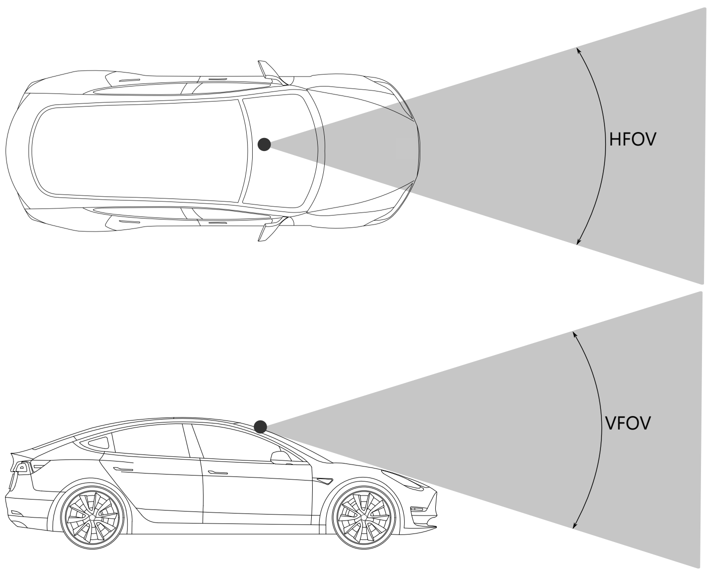


Figure 3.1: VFOV and HFOV of a MFC.

For the VFOV, a first estimate is performed based on the mounted height and the minimum required detection distance. The lower HFOV (Fig. 3.2) angle is calculated by applying Eq. (3.1) [43].

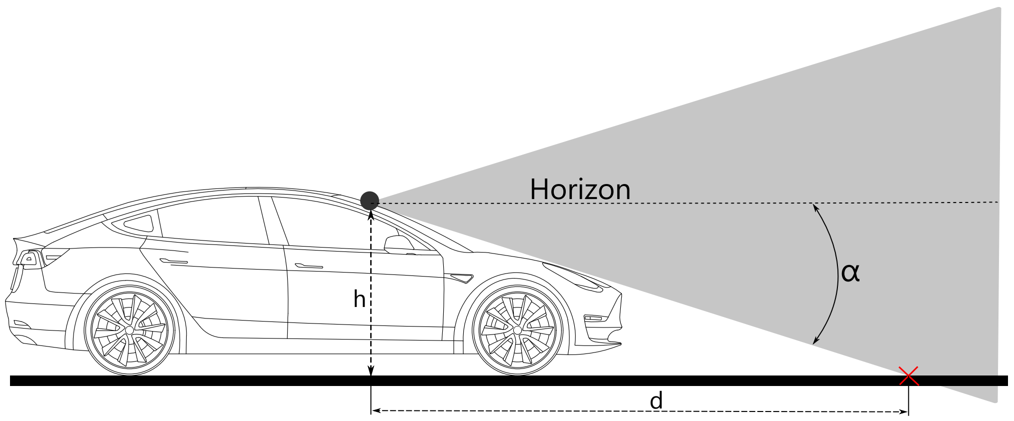


Figure 3.2: Lower VFOV of a MFC.

Lower field of view equation:

$$\alpha = tan^{-1}\left(\frac{h}{d}\right) \tag{3.1}$$

Where:

α is the lower VFOV angle.
h is the mounting height.
d is the minimum detection distance.

3.2 Multi-Functional Camera System Architecture

A basic camera section-view is shown in Fig. 3.3. The optical acquisition process starts by the projection of a bundle of light rays, representing an object or a scene, through the front lens onto the image sensor. Before reaching the image sensor, light rays are delimited via an aperture, deviated/focused by a lens train, and, more often than not, the infrared parts of the light spectrum are filtered out by the InfraRed-Cut-Off Filter (IRCF). Pixels of the image sensor convert the photons into an electronic output signal, representing the image acquisition process, which is later analyzed by a processing unit for further ADAS/AD function development.

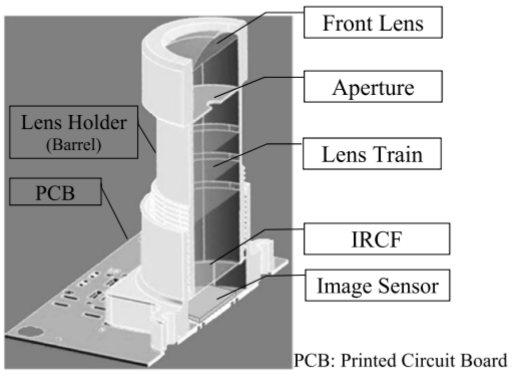


Figure 3.3: MFC 3D section view.

The basic system decomposition of the MFC module is illustrated in Fig. 3.4. The MFC module is divided into two main components: the Optical Module and the Imager Module. On a lower decomposition level, the system's subcomponents are also represented, thus identifying the first abstraction level of the modeling process to follow.

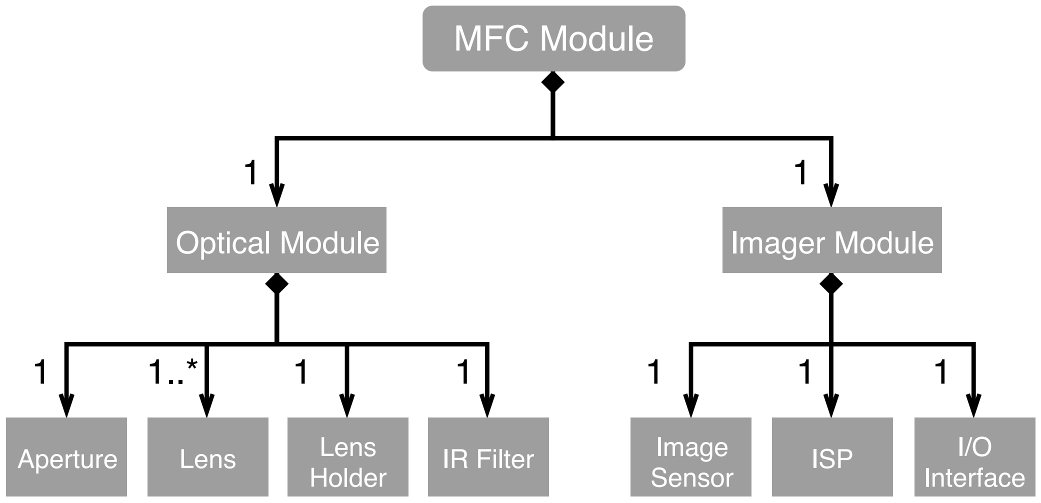


Figure 3.4: MFC system decomposition.

In Fig. 3.4, the "1" beside the arrows indicates the number of necessary components for the system (MFC in our case) to function properly. "1..*" indicates that one or many components are required, which varies depending on the number of necessary lenses inside the optical module.

The system's **Optical Module** consists of the following main components:

1. **Aperture**: Represents an opening through which light travels. More specifically, it limits the amount of light that can reach the image-sensor.
2. **Lens**: An optical lens, or a train of lenses, are usually made from material such as glass or plastic with the main purpose of focusing incident light to form an image. Depending on the fabrication process and tolerances, lenses introduce different degrees of distortions and aberrations that make the image an imperfect replica of the object.
3. **Lens Holder**: Usually a cylindrical design manufactured from plastic or a metal alloy. Its main purpose is to position and hold all optical module's components in their place. Additionally, the lens holder also connects the optical module to the imager module.
4. **IR Filter**: Infrared filters are designed to reflect or block close- and mid-range infrared wavelengths while passing visible light.

The system's secondary components are part of the **Imager Module** and consist of the following parts:

1. **Image Sensor**: A Complementary metal-oxide-semiconductor (CMOS) in this case; it converts received photons into electrons and collects the generated charges to be later converted into electric signals (voltage). The latter is then converted into a digital value with the help of an Analog to Digital Converter (ADC).
2. **ISP**: The Image Signal Processor (ISP) acts directly on the digital signal and includes several image-processing algorithms, such as Noise Reduction (NR), demosaicing, and image sharpening.
3. **I/O Interface**: The input/output interface enables image data transfer from the image sensor to other host processors. Data transfer is performed either by parallel or serial interface.

3.3 Optical Path and the Pixel Journey

The optical path represents the distance that light travels in a vacuum congruous with the time it takes to travel a distance d in a medium [44]. When a light ray travels through a series of optical media of thickness d, d', d'', ..., and refractive indices n, n', n'', ..., the total optical path (Δ) is merely the sum of the separate values:

$$\Delta = nd + n'd' + n''d'' + ... \tag{3.2}$$

The reason for identifying the optical path lies in the fact that photon light may slow down or speed up as it travels from one medium to another, may undergo a change of direction (refraction) at an interface between media, or could be absorbed (or reflected) when encountering certain substances or surfaces [45]. These stages and media that a source light passes through, especially at the "Lens Train" block, determine the errors related to lens imperfections, incorrect locations, or variations in lens geometry, which are collectively referred to as "distortion and aberration."

Taking into consideration the mounting position of the MFC inside the car as defined in section 3.1, light rays emerging from the environment (outside of the ego vehicle) should first pass through the windshield before reaching the MFC. Laminated glass manufactured by sandwiching a layer of Polyvinyl butyral (PVB) between two pieces of glass is used for the automotive windshield. Laminated glass is, later on, bent to the desired curvature, with a radius between 3300 and 4000 mm, Fig. 3.5 (a), possessing a specific optical power. Due to various factors involved in the manufacturing process, the windshield's optical power is not uniform throughout its surface area; this leads to non-uniform light distortion over the intersection between the camera's FOV and the windshield; such an intersection area is represented in Fig. 3.5 (b).

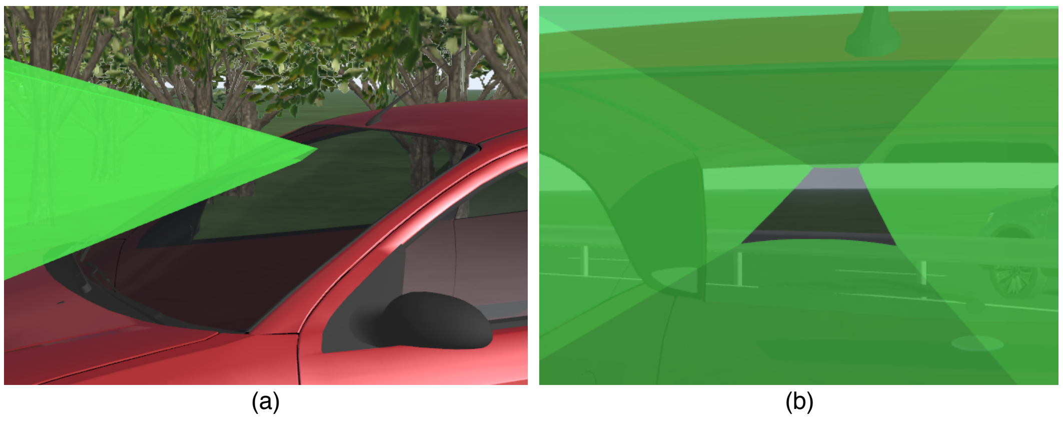


Figure 3.5: (a) Camera FOV representation behind the front windshield, (b) FOV intersection with the windshield.

Another important aspect is the refraction that occurs when light travels from one material into another material with a different refractive index. The direction of light changes because the speed of light changes. The speed of light slows down when entering an optically dense material, since different materials have different refractive indices [46]. Using Snell's law, the refractive index or angles for a certain situation can be calculated thus:

$$n_1 \times sin(\alpha) = n_2 \times sin(\alpha_2) \tag{3.3}$$

Where:

n_1 is the index of refraction for air.
α is the incident angle for.
α_2 is the angle of refraction.

For example, when light passes through a windscreen at an angle different from 90° (Fig. 3.6), the incoming and outgoing light has the same direction if the two surfaces of the windscreen are parallel, but a slight displacement of the beam will still exist [46].

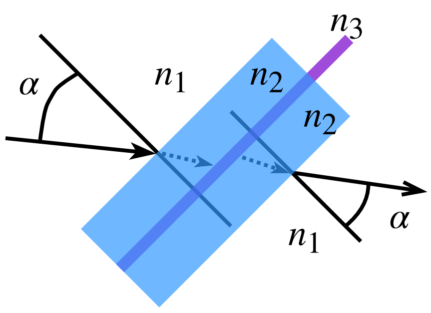


Figure 3.6: Windshield refraction.

Even though the windshield may have several effects on the final image acquired by an MFC, these effects are briefly mentioned in this section and are outside the scope of this dissertation.

Figure 3.7 represents the bases of the 2nd abstraction level to be considered in the modeling process. In Fig. 3.7, light, acting as a pencil of incident rays, travels from the environment to the camera sensor primarily through its front lens (not shown in the diagram) and is limited further on by the **Aperture**. It continues its path as an analog signal through the **Optical Train** and the infrared **Cut-Off Filter** (if available) until it reaches the **Image Sensor**.Eventually emerging as a digital signal to be consumed by various **Image-Based Functions**.

Depending on the quality assessment of the produced image (raw/Red, Green Blue (RGB)), a feedback signal could be sent to the **ISP** unit and/or to the image sensor unit to adjust their parameters, such as exposure time, white balance, etc. In some cases, image enhancements and histogram adjustments could directly be performed on a dedicated **Pre-Processing** unit (not shown in the diagram), which lies between the raw image and **Image-Based Functions**.

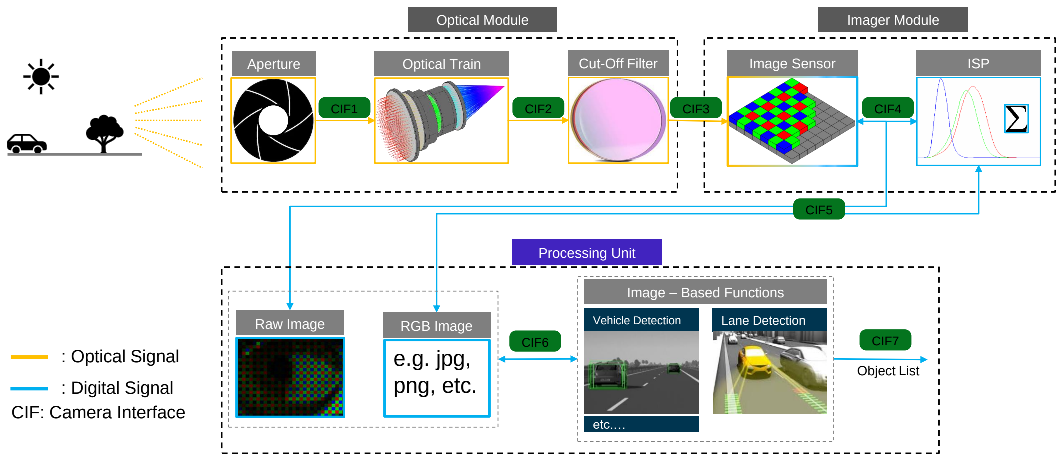


Figure 3.7: Camera interfaces.

Additionally, Fig. 3.7 depicts main Camera Interface (CIF) at different decomposition levels. This decomposition illustrates the type of information/signals being transmitted (analog, digital) and their usage:

- **CIF1, CIF2, CIF3**: Camera Optical Interface, with light being transmitted, absorbed, or reflected.
- **CIF4**: Internal data transfer via control registries. It is directly performed on the image sensor Chip itself (image without demosaicing, or raw image). Bidirectional arrows indicate that other units can send signals to the image sensor for better performance under different conditions (the same concept applies to CIF5 and CIF6).
- **CIF5**: Could be parallel or Series communication, e.g., HiSPi (image without demosaicing, or raw image). CIF5 usually connects different processing units to the camera.
- **CIF6**: Internal data transfer via control registries and memory access; directly performed on the processing unit.
- **CIF7**: Interface providing the object list to other control functions to be later passed on to the vehicle.

CIF represent a generic example of a basic interface architecture and the following points should be kept in mind:

1. The diameter of the aperture does not necessarily indicate the diameter of the effective aperture, i.e., there could be different optical stops inserted after the Optical Train..

2. The cut-off Filter's position could also vary or not even be present at all. Additional coating materials are also always used on lenses and on the lens holder.
3. The image sensor in Fig. 3.7 shows a Bayer filter implementation; however, it is important to point out that this may differ between varying image sensors.
4. The processing unit may or may not have an interface to acquire raw image and RGB image data; this depends on the unit itself and on the image sensor being used.

4 Modeling Methodology

4.1 Introduction

A physics-based model is a mathematical representation of an object or its behavior. Such representations usually incorporate physical characteristics, such as forces and energies, into the model, allowing a numerical simulation of its behavior, e.g. its deformation or acceleration. For example, in the case of a physics-based camera model, optical characteristics, such as the focal length, and image sensor characteristics, such as quantum efficiency and Analog-to-Digital Converter (ADC) resolution, could be mathematically represented to describe parts of the camera's behavior, which, in this case, is the generated image in various conditions.

Simulation software for virtual test driving is capable of directly generating rendered images based on data related to ego vehicle dynamics, traffic behavior, and driving environment. Such software, however, do not take into consideration all physical characteristics of an advanced driver assistance camera, and thus fail to incorporate several camera artifacts that may result from various optical and electrical component imperfections or design limitations.

Taking into consideration that the simulation environment already provides 3D shape representations of objects, their motions, and their interactions with the environment, the modeling methodology will rely solely on a pre-rendered image or a Video Data Stream (VDS) provided by the simulation environment. The model's output will depend on physics-based modeling of a set of relevant camera effects for ADAS/AD functions. These effects are identified and classified in sections 4.2 and 4.3 respectively.

4.2 Multi-Functional Camera System Analysis and Effects Identification

Due to the non-uniform distribution of camera effects over the entire image, it is crucial to analyze all relevant regions of interest for various CV algorithms.

Since the first demonstration of visual road vehicle guidance at speeds of up to $\approx$ 40km/h in 1986 [47], computer vision algorithms relied primarily on a specific ROI to achieve their designated tasks. Such ROIs were significant not only in reducing computing time, but also avoiding false positives and negatives due to various image artifacts that could appear if the entire image is set to be processed.

For a front facing MFC, different ROIs with different sizes and positions could be pre-defined for various CV algorithms. For example, in the case of Lane Detection (LD), represented in Fig. 4.1 (c), only the lower half of the image is important, whereas for Traffic Sign Detection (TSD) and Traffic Light Detection (TLD) in Fig. 4.1 (d and e; adapted from [48]), the image's margins and upper part are essential. In the case of dynamic object detection, as shown in Fig. 4.1 (a, b, and c) for Vehicle Detection (VD) or Pedestrian Detection (PD), a pre-defined ROI would prove difficult to create. In such cases, a vehicle or a pedestrian can occupy several positions in the image at various distances; thus, the size of the ROI changes as well.

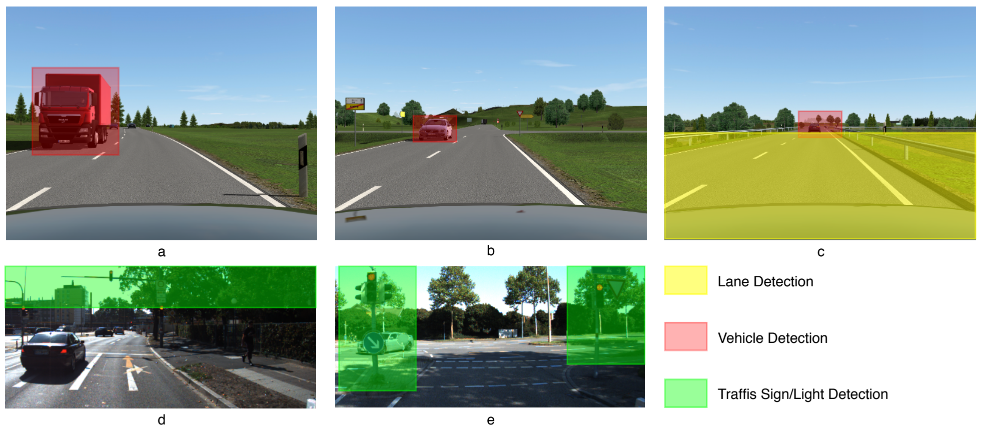


Figure 4.1: ROIs for ADAS/AD functions.

In Fig. 4.2, the overlapped ROIs from different ADAS/AD functions are represented and depicted in a 1240x980 px image. The gray color indicates the regions where no ROIs are extracted, which covers a very small area from the overall image. This justifies the need to apply the physics-based model on the entire image and take into consideration image distortions and aberrations that are most prominent on image margins.

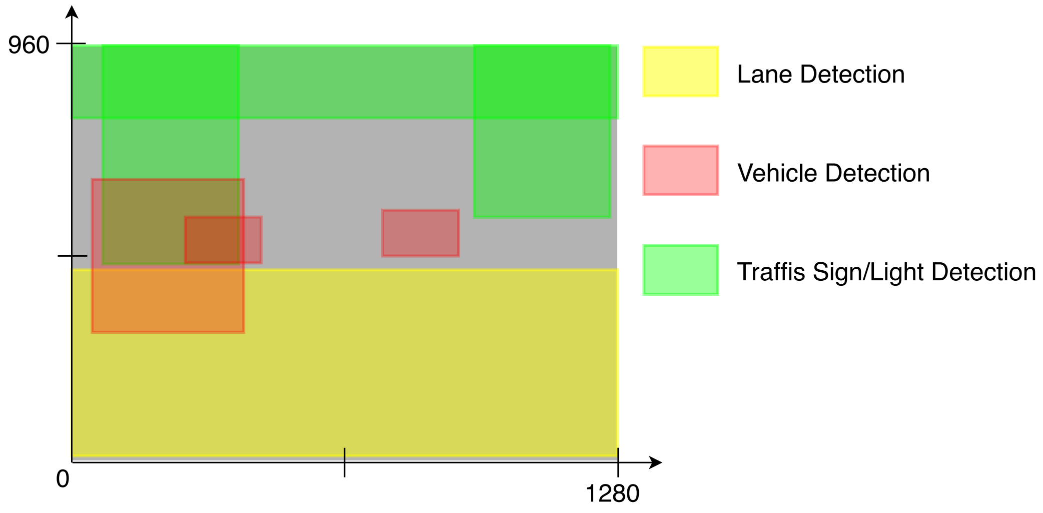


Figure 4.2: ROI overlap representation.

As represented in Fig.4.3, this also applies for ADAS and AD functions that rely on DNN, more specifically CNN where such networks don't rely on a specific ROI, but instead consider the whole image as an input.

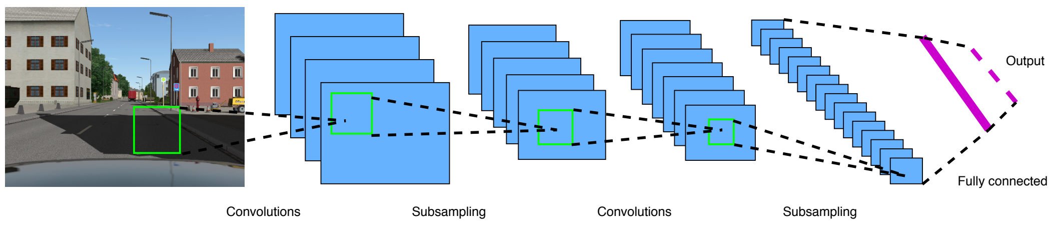


Figure 4.3: Example of a CNN architecture.

In Fig.4.3, the green bounding box will eventually slide over the whole image, subsampling each pixel and eventually generating a fully connected layer through various convolutions.

4.3 Classification of Imperfections

Based on the previously presented argumentation (section 4.2) and on MFC's system decomposition/optical path shown in Ch. 3; sources of various image artifacts can be traced back to the **Optical Module**; **Imager Module** or to **Post Processing**; the latter is included to provide an overall picture of the decomposition. However, it is out of the scope of this dissertation. In Fig. 4.4, a block definition diagram illustrating the main effect types and their decomposition is presented. The diagram is color coded. For example, the color transition from blue to violet of the **Intrinsic Blur** block signifies that it is caused by a combination of **Optical** and **Image Sensor effects**.

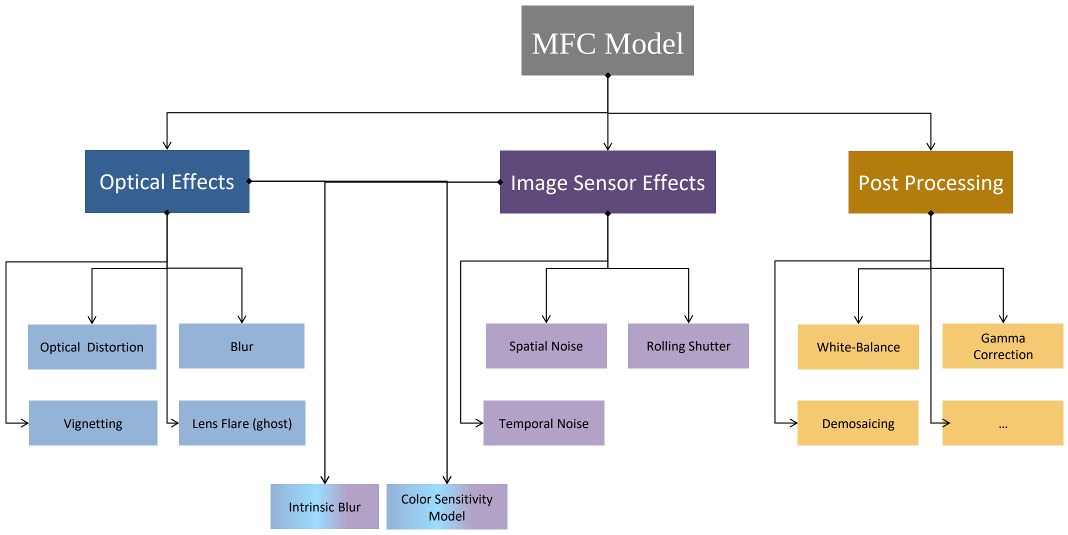


Figure 4.4: Block definition diagram for camera effects classification.

4.3.1 Optical Effects

In this section, effects that are generated by the optical module, their causes, and impact on ADAS/AD functions are illustrated. Optical aberrations are generally classified into two categories: chromatic (polychromatic, multiple wavelengths) and monochromatic (single wavelength). Chromatic aberrations occur because visible light consists of different wavelengths, each with a particular imaging property. From the lens train perspective, chromatic aberrations are caused by dispersion due to the variation of the lens's refractive index with the incident wavelength [49]. In contrast, monochromatic aberrations are caused by the geometry of the boundary surfaces (the incidence angle varies across a spherical lens surface) and occur when light rays are reflected or refracted [44], [50].

Despite the existence of well-defined technological procedures and solutions for chromatic and monochromatic aberrations, and to keep the total Bill of Material (BOM) low, optical modules suppliers do not entirely integrate them, and consequently aberrations and other optical effects continue to persist [43].

Distortion Effect

For an ideal optical system, the formed image should be geometrically similar to the object, i.e., image dimensions should be linearly related to those of the object. However, in practice, optical modules currently fail to maintain a linear relationship between the object and its image. This phenomenon, referred to as image distortion, occurs when the rays from an off-axis point do not converge perfectly at the image point; this is also known as radial distortion, as shown in Fig. 4.5.

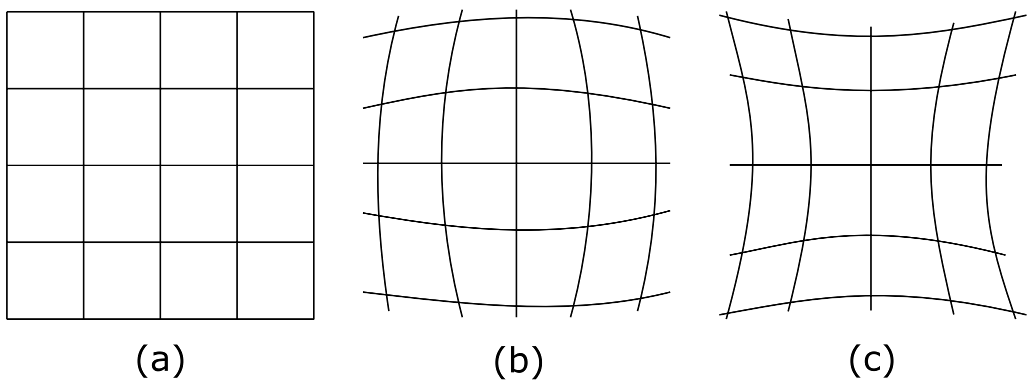


Figure 4.5: Radial Distortion: (a) No Distortion, (b) Barel Distortion, (c) Pincushion Distortion.

Radial distortion is considered a negative distortion when the actual image is closer to the optical axis, and a positive distortion when the image lies further from the axis than the ideal image. Fig. 4.5 (a) represents the undistorted image of an object consisting of a rectangular wire mesh; when suffering a negative distortion, the image takes on a barrel-like appearance in which the image magnification decreases with an increasing distance from the optical axis (Fig. 4.5 (b)). By contrast, when suffering a positive distortion, the image takes on a pincushion-like shape (pincushion distortion), where the image magnification increases with an increasing distance from the optical axis (Fig. 4.5 (c)).

Additionally, tangential distortion occurs when the lens and the image plane (image sensor) are not parallel or if the image is not formed on the focal plane as predicted by the paraxial equations as shown in Fig. 4.6.

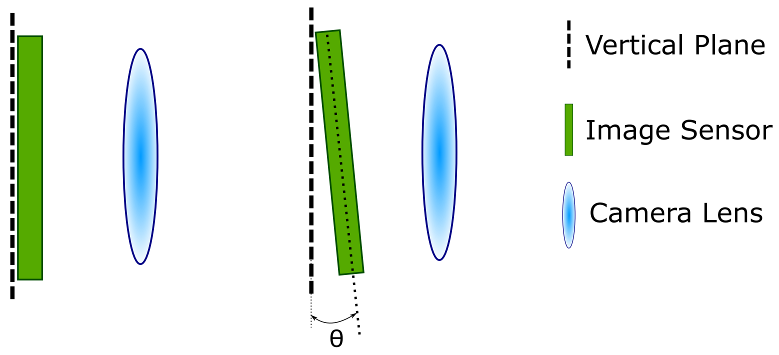


Figure 4.6: Tangential distortion.

Distortion due to false object representation may play an essential role in deceiving several ADAS/AD functions in image data extraction and interpretation. For example, to estimate the distance of the ego vehicle inside a driving lane, an LD function may rely on detected lane markings and on the camera's mounting position to evaluate its position. Fig. 4.7, illustrates the influence of distortion on estimating the vehicle's central offset between the solid yellow and the dashed white lane, where a difference of 0.05 m can be measured.

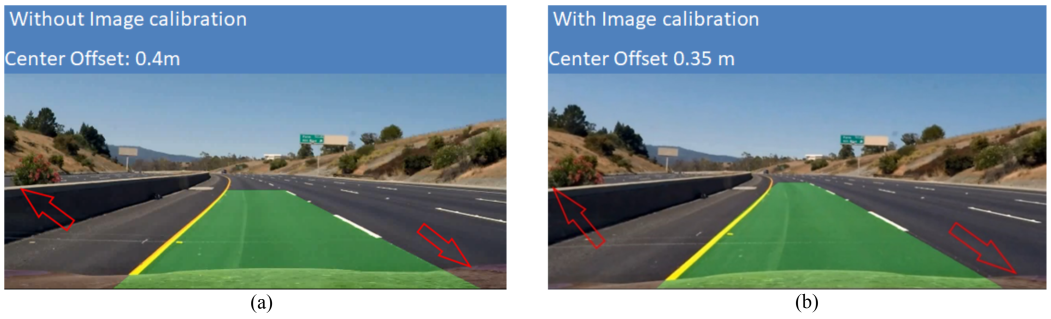


Figure 4.7: Distortion effects on image perception, calibration done via OpenCV API.

Vignetting Effect

Vignetting effect can be described as the reduction of the image's brightness or saturation at or around the periphery compared to the center of the image. In its essence, the optical module of an advanced driver assistance camera consists of a multi-lens system where front elements shade rear elements (closer to the image sensor). In some cases, this results in a reduced effective lens opening of the off-axis incident light, and, subsequently, a gradual decrease in light intensity toward the image periphery. Additionally, when extremes of object points move away from the optical axis, the effective aperture stop is reduced for the off-axis rays, thus further decreasing the brightness of the image at image points near the periphery [45].

In Fig. 4.8, the effect of the aperture's size on the vignetting effect is represented. In Fig. 4.8 (a), light rays traveling from image points AB go through the aperture and hit the lens optimally. By increasing the size of the object, the bundle of rays entering the lens is now reduced by the aperture, causing peripheral darkening (Fig. 4.8 (b)). In more extreme cases, if the object's size is made even more significant, and the camera is still at the same distance from the object, the field of view does not change, yet the bundle of rays that enters the focal plane is reduced even more.

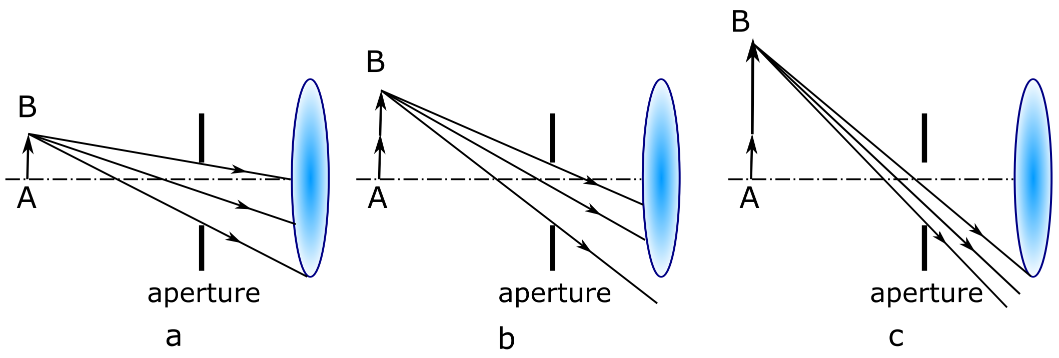


Figure 4.8: Effect of aperture size on vignetting.

Moreover, one other cause of vignetting is referred to as **Natural Vignetting** or **Natural Illumination Falloff**. Natural vignetting is associated with the $\cos^4$ law of illumination falloff [51]. The brightness of the image away from the optical axis falls at a rate proportional to the cosine to the fourth power of the angle that the light makes with the line perpendicular to the image sensor. It is essential to keep in mind that this principle only applies to a perfect system and that natural vignetting varies according to each lens design. A more detailed derivation of the $\cos^4$ law can be found under [51].

The vignetting effect can be detrimental to machine vision, as it impairs measuring results and detecting edges of specific objects.

Blur Effect

In an ideal situation, each small point within the object should be represented by a small, well-defined point within the image. In reality, the "image" of each object point is spread, or blurred, within the image. In every imaging process, blur places an absolute limit on the amount of detail (object smallness) that can be visualized. Blur most often occurs on out-of-focus objects due to camera motion. However, there is also an additional permanent lens blur caused by the optics of image formation, e.g., lens aberration and light diffraction [29].

Due to the wave nature of light and the finite aperture size of an optical system, the angular resolution can be estimated by the Rayleigh criterion. For the circular aperture (which is the case of most MFCs) θ, the angular resolution can be defined as follows:

$$\theta = 1.22 \times \frac{\lambda}{D} \tag{4.1}$$

Where:

θ is the angular resolution.
λ is the wavelength.
D is the diameter of the aperture.

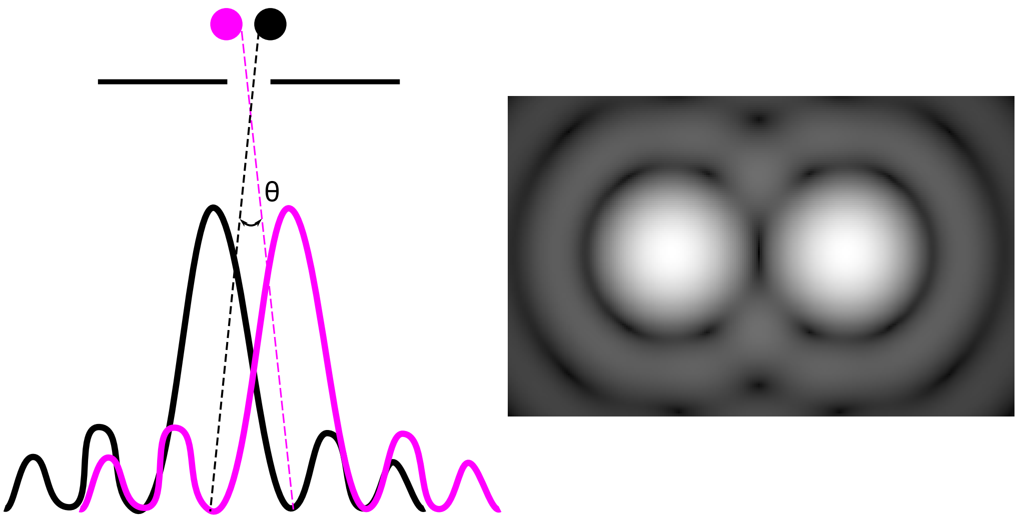


Figure 4.9: Angular resolution for two point objects.

The angle θ, (Fig. 4.9) in this case, represents the limit of how small an image can be distinguished and not rendered sharply (the latter is image sensor relevant). The airy disk of a circular slit consists of a widening and reshaping of the image of a point source, spreading the central point but also consisting of a series of secondary rings.

In a driving situation, this may play an important role when trying to distinguish between two cars at a certain lateral distance from one another, or even when trying to detect both headlights of the same car in night driving situations.

Flare and Ghost Effects

Lens flare, in its general form, occurs when light is scattered in a lens system due to a bright light source bouncing off of different optical module elements. Lens flare can be mainly divided into two main categories:

1. Veiling flare that manifests in a drastic reduction of image contrast by introducing haze in different colors.
2. Ghost flare, which may add different halos or other geometrical shapes into the image.

In the following section, the ghost effect that results from the lens's aperture shape is considered. As light travels through one medium to another, part of the incident light is reflected, part is transmitted, and the remainder is absorbed or scattered. Although for most uncoated optical glasses the reflected light only accounts for 4% of the total incident light (for air to glass), such losses may accumulate for a compound lens assembly [45]. For example, in a three-lens compound assembly, the final incident light that reaches the image sensor will be reduced by approximately 12%. The following equation represents the amount of reflected light at any given optical component:

$$r_n = \lambda * I_0 * (1 - \lambda)^n \tag{4.2}$$

Where:

r_n is the intensity of reflected light at component n.
λ is the reflection percentage with respect to the incident light.
I_0 is the intensity of the incident light that hits the first lens.

For example, in Fig. 4.10, if the incident light I_0 has an intensity value of 100, then according to Eq. (4.2) $I_3 = I_0 - (r_0 + r_1 + r_2) = 88.47$.

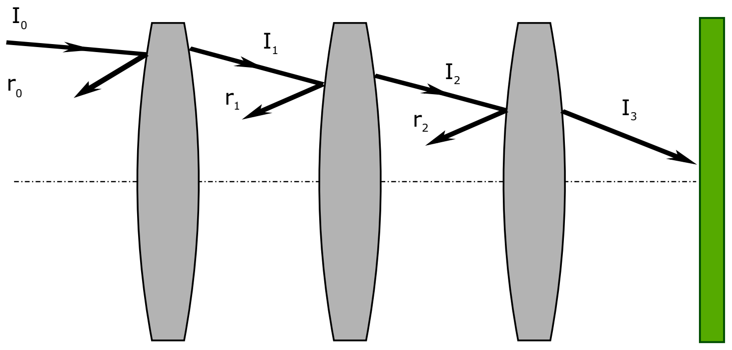


Figure 4.10: Reflection and refraction in a multi-lens optical system.

Furthermore, in an encapsulated optical system, the weak reflected rays represented in Fig. 4.11 might create ghost images superimposed on the image generated by the transmitted rays. For example, r_2 in Fig. 4.11 indicated in blue may undergo a total internal reflection and reach the image sensor, thus forming a ghost image.

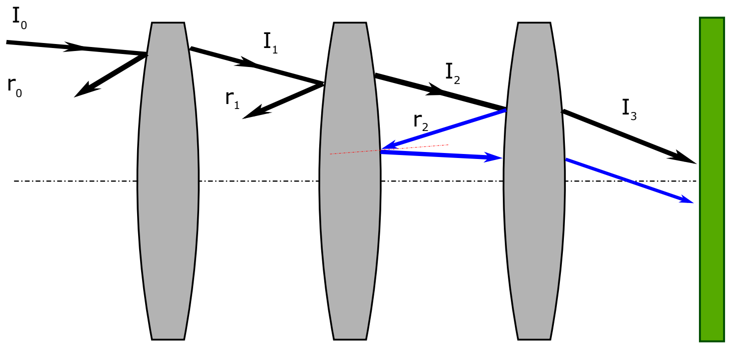


Figure 4.11: Ghost image formation.

4.3.2 Image Sensor Effects

As previously described in section 3.2, a digital image sensor converts photons hitting the active pixel area into a digital number, depending on the desired temporal resolution and other sensor properties. This process is governed by predefined exposure time. The temporal resolution, i.e., frames per second (fps) is often dictated by ADAS/AD functions.

The term ”noise” in image sensors may be defined as signal variations that lead to image deterioration. As a result, sensitivity and performance of an image sensor are determined by noise [52]. Image degradation caused by noise can be classified into the following two categories:

1. Spatial Noise.
2. Temporal Noise.

In addition to noise, in a CMOS image sensor, an electronic rolling shutter is usually used in order to read out the accumulated charge from the pixel array. The shifted readout exposure times distort the image of moving objects; this is also known as the rolling shutter effect.

For ADAS/AD functions, image noise may lead to a high number of false positive and negative detections. For example, in low light conditions (at night or in other low light situations like tunnels), cars cannot be detected by their visual features, and thus several detection algorithms rely on headlights/taillights [53]. This lack of visual features can affect detection and tracking processes, especially when coupled with high noise level due to bad illumination and other camera-specific limitations. In the next section, more details regarding image sensor effects are presented.

Spatial Noise

Signal variations in an image-sensor array that differ from one pixel to the other are referred to as spatial noise or Fixed Pattern Noise (FPN), and may occur when the camera is in either dark or illuminated environments.

As presented in table 4.1, FPN in dark conditions is caused by Dark Signal Non-Uniformity (DSNU) that may result from a thermal component, which depends on temperature and exposure time, and manifests itself directly as an offset that is always present on a pixel level. Another cause of DSNU that is independent from the exposure time is the manufacturing process of the image sensor, where gain variations and imperfections in the electronic circuits influence the spatial non-uniformities.

On the other hand, Photon Response Non-Uniformity (PRNU) occurs in illuminated scenarios. When uniform light falls on a camera sensor, each pixel should ideally output the same value [54]. Due to pixel geometry and variations in cell size and substrate material, pixels exposed to the same source of light output different values. PRNU is usually considered to be a common characteristic of the sensor since it is caused by the physical properties of the sensor and cannot be eliminated.

Table 4.1: Spatial Noise in Dark and Illuminated Environments

	Dark Conditions	Illuminated Conditions
Spatial Noise	DSNU	PRNU
	Row/Column-wise Fixed Pattern Noise	

Temporal Noise

Noise variation/fluctuation in the time domain is referred to as temporal noise and may present in dark and low light conditions. Table 4.2 represents the main components of temporal noise.

Table 4.2: Temporal Noise in Dark and Illuminated Environments

	Dark Conditions	Illuminated Conditions
Temporal Noise	Theram Noise (Dark Current)	Photon Shot Noise
	Read/Reset Noise	

When the sensor is placed in darkness, the sensor output should ideally be a black picture over its entire pixel array, and the output should be sustained throughout the entire functioning duration. However, in reality, this ideal is hardly attainable, since the image sensor or camera are often subjected to a so-called ”dark current noise” that is dependent on the exposure time and the imager’s operating temperature. Due to dark currents, the image quality may be drastically degraded when the camera is operating at low light conditions. In illuminated functioning conditions (low light levels), the photon shot noise contributes a large portion of the temporal noise. Since photons (quantum particles) possess a random arrival nature at the image sensor over a specific duration of time, even when the intensity of the light source is constant, the result is an inconsistent number of collected photons on the pixel level. As these events are independent of one another, photon shot noise obeys Poisson’s distribution [27].

Rolling Shutter and LED Flicker Effects

In a CMOS image sensor array with an electronic rolling shutter, a timing and control circuitry is usually required to sequence through the rows of the array, resetting and then reading each row in turn. In the time interval between resetting a row and reading that row, the pixels in the row integrate incident light. This means that the very last row of the image sensor is exposed late to light in comparison to the very top row of the sensor. This time shift from the top to the bottom of the image causes the rolling shutter effect, which can be seen as a bending or skewing of the image. This occurs when the object or the camera moves sideways as the image is taken. Fig. 4.12 (a) represents a photo of a stationary white disc mounted on a motor. In Fig. 4.12 (b), the rolling shutter effect can be observed when the black rectangle stretches over the entire disc when the motor starts spinning at a specific speed.

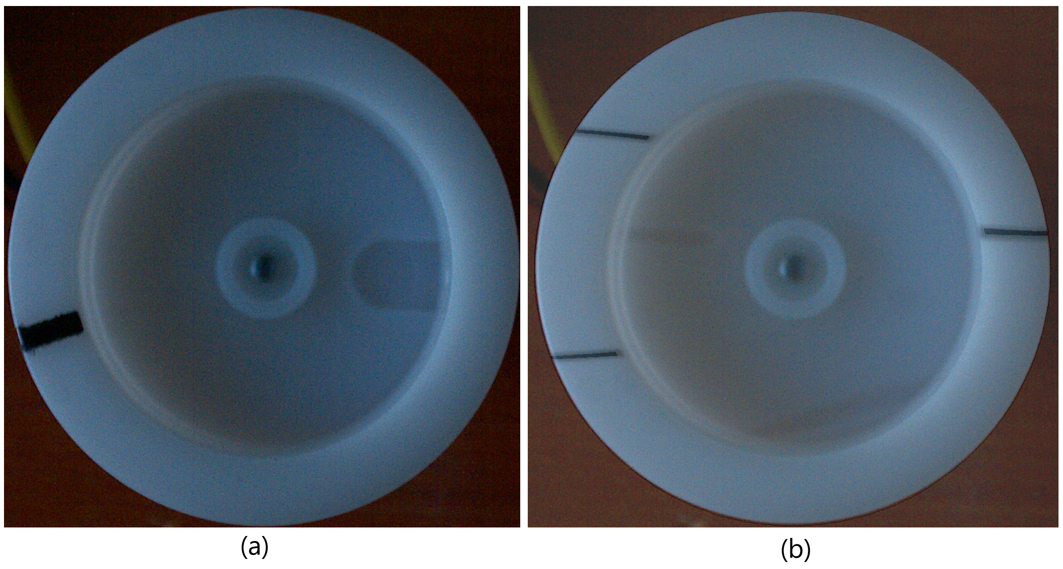


Figure 4.12: Rolling shutter effect, (a) Stationary white disc mounted on a motor, (b) Rolling shutter effect when the motor starts spinning.

Finally, when trying to capture an image of Light-Emitting Diode (LED) lighting (vehicle taillights, traffic signs or traffic lights), the so-called "LED Flicker" effect is observed, where an imaged light source appears to flicker, even though the light source appears constant to a human observer.

As illustrated in Fig. 4.13, a pulsed light source may appear to be off for the camera whenever the LED ON time and the camera's exposure time do not coincide.

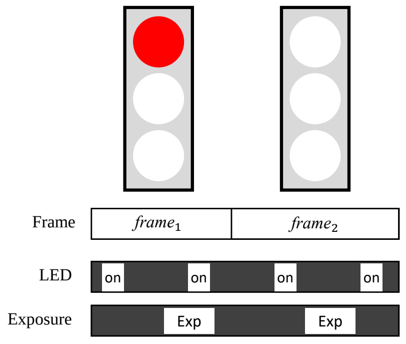


Figure 4.13: LED flicker effect, on/off.

This behavior could be observed whenever the exposure time of the camera is less than or equal to that of the light source [55]:

$$T_{exp} \leq \frac{1 - PWM_{dutycycle}}{PWM_{freq}} \tag{4.3}$$

Where:

PWM_{freq} is the frequency of the light source.
$PWM_{dutycycle}$ is the duty cycle of the light source.

Another effect of LED flicker occurs when different image frames are exposed to more than one pulse during the same exposure time. For example, in Fig. 4.13, frame one was exposed to two pulses of light during $frame_1$'s exposure time, whereas $frame_2$ only once. Consequently, $frame_2$ will have a lower brightness level.

This behavior could be observed whenever the exposure time of the camera is greater than or equal to that of the light source [55]:

$$T_{exp} \geq \frac{1 - PWM_{dutycycle}}{PWM_{freq}} \tag{4.4}$$

Where:

PWM_{freq} is the frequency of the light source.
$PWM_{dutycycle}$ is the duty cycle of the light source.

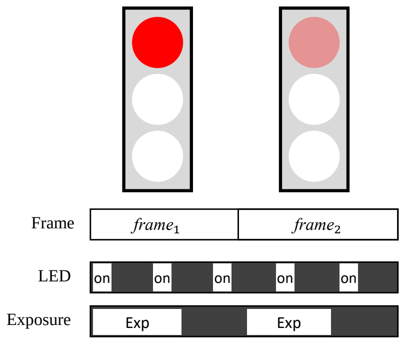


Figure 4.14: LED flicker effect, brightness variation.

Pulsed LED lights are increasingly used for traffic signals and other traffic signs, including variable speed signs and road work signs. LED flicker in such situations may cause miss or non-detection of traffic signs, leading to potentially hazardous implications.

4.4 Modeling Approach

An advanced driver assistance camera consists of many hardware components that interact intrinsically with each other and extrinsically with the surrounding environment. Additionally, there are often various software components governing these interactions and adapting them to various assistance systems. The main challenge in modeling a physics-based sensor lies in the difficulty of obtaining necessary specifications for various hardware and software components. In the following sections, challenges in model parametrization and compatibilities with simulation software are addressed.

4.4.1 Challenges in Model Parametrization

Most of the previously defined optical and image sensor effects are repeatedly studied and investigated, and on different occasions even modeled and simulated. Nevertheless, when it comes to advanced driver assistance cameras, obtaining the necessary parameters to derive and calibrate such models has proven to be an arduous, especially for parties who are not part of an optical or semiconductor manufacturer.

For example, if provided with the exact optical module composition, i.e., a number of lenses and their positions, lens diameter and thickness, coating materials, indices of refraction, etc., it is possible, via dedicated software and other techniques (like ray tracing model), to simulate the behavior of such a system with high precision. In reality, datasheets for optical modules and image sensors provide rudimentary information, and that is not enough to parametrize such physics-based models.

In order to overcome this challenge, the presented modeling approach relies on theoretical aspects, defined necessary parameters for a specific camera effect, and on experimental setups to calibrate and set these parameters. (Such parameters are derived empirically.) Additionally, theoretical aspects of specific effects are used to construct experimental setups and procedures for generating lookup tables that could, later on, be used for image augmentation and manipulation.

4.4.2 Compatibility with Integration Software

Several aspects regarding the camera model's input and output should be taken into consideration, especially when targeting various simulation software and integration platforms.

Aside from the model's parameters, the main input consists of pre-rendered images or VDS that can be provided either by a simulation software for virtual test drive or directly by any other synthetic image generator, like Unreal Engine 4[1] (Fig. 4.15). Thus, the final model should always contain a dedicated interface that is capable of adapting to the source of its input.

[1]Unreal Engine 4 contains a physics-based rendering engine with advanced dynamic shadow options, ray-tracing functionality, screen-space reflections and lighting channels https://www.unrealengine.com/en-US/features .

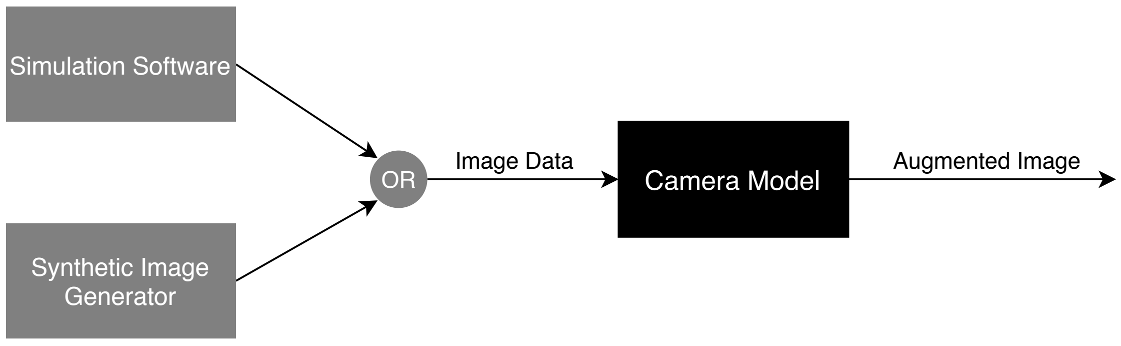


Figure 4.15: Camera model input.

On a second note, the model's augmentation pipeline should be flexible enough to bypass specific camera effects that may be provided by the image source, i.e., either a simulation software or a synthetic image generator. For example, in Fig. 4.16, if the input image (**Image Data***) is directly generated with optical distortion, the camera model in this case should not implement additional distortions on the received image. This is depicted by the red **Optical Distortion** box inside the camera model.

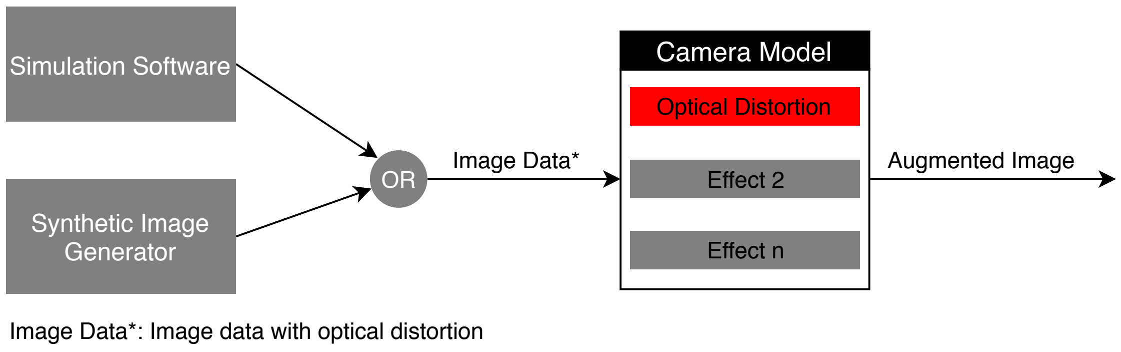


Figure 4.16: Selective effect implementation.

5 MFC Modeling and Parameters Calibration

5.1 Introduction

In this chapter, the modeling and parametrization process of various optical and image sensor effects is demonstrated. The obtained results highly depend on the abstraction levels, and the intended use. For this dissertation, one of the main focuses is to illustrate the need for physics-based (behavioral) sensor models during the development and validation of ADAS/AD functions. Therefore, starting by the modeling and parametrization process of optical effects, like distortions and vignetting, this chapter continues to describe a more abstracted model for image sensor effects, like temporal/spatial noise and rolling shutter.

5.2 Modeling Optical Effects

In this section, optical effects that are primarily generated by several imperfections in the optical module or light-related properties are modeled and parametrized.

5.2.1 Radial and Tangential Distortions

For the process of modeling radial and tangential distortions, the Brown-Conrady [56] [57] distortion model is used. For such a model, Eq. (5.1) and (5.2) describe the radial distortions, while Eq. (5.3) and (5.4) describe the tangential distortions:

$$x_u = x_d \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{5.1}$$

$$y_u = y_d \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{5.2}$$

$$x_u = x_d + \left[2 p_1 x_d y_d + p_2 \left(r^2 + 2 x_d^2\right)\right] \tag{5.3}$$

$$y_u = y_d + \left[2 p_2 x_d y_d + p_1 \left(r^2 + 2 y_d^2\right)\right] \tag{5.4}$$

Where:

(x_d, y_d) represent the pixel coordinates as projected on the image plane.
(x_u, y_u) represent the undistorted pixel coordinates.
k_n is the n^{th} radial distortion coefficient.
p_n is the n^{th} tangential distortion coefficient.

Additionally, the intrinsic camera parameters (represented as a 3x3 matrix) are calculated.

$$intrinsic\,matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where:

(c_x, c_y) represent the coordinates of the optical center.
(f_x, f_y) represent the camera focal lengths.

OpenCV[1] is used to calculate the distortion parameters and the camera's *intrinsic matrix*. The calibration process starts by taking several photos of a known calibration pattern and feeding them into a set of predefined calibration APIs provided by OpenCV.

Due to the accuracy of available corner detector algorithms, a chessboard calibration pattern of size 11x9 is used. Instead of printing out the calibration pattern and risking errors caused by printer issues or paper flatness during the image acquisition process, the chessboard pattern is directly displayed on a high-resolution screen and subsequently exposed to the camera. To

[1]OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. https://opencv.org/about/

form a well-posed system of equations, and to solve it, several chessboard pattern images are captured by the Camera Under Modeling (CUM). In order to account for noise and other image artifacts, it is recommended to use at least ten pictures from various distances and orientations with respect to the calibration pattern. During the calibration process, the focal length and distortion parameters are estimated using the least square method; so by moving the camera or the checkerboard, and acquiring a different number of images, we are actually also receiving more information that is used in obtaining more accurate results from the calibration process.

The process of image acquisition is automated with the help of a camera testbed prototype presented in Fig. 5.1. The kinematic diagram of the testbed represents one Degree of Freedom (DoF) for translation and three DoFs for rotation, i.e., for yaw, pitch and roll.

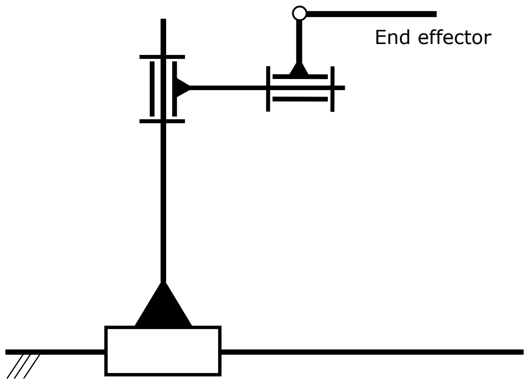


Figure 5.1: Camera testbed kinematic diagram.

During the image acquisition process, the camera is mounted at the end effector, and its orientation and position are manipulated by the testbed in a specific range of motion, assuring that the entire chessboard pattern, which is displayed on a high-resolution screen, is within the boundaries of the camera's FOV. The obtained results are then fed to a calibration algorithm (OpenCV), and the intrinsic calibration matrix is calculated. The following is an *ideal intrinsic calibration matrix* for a 1280x960 pixel, referred to as *px* throughout this dissertation (px) image format captured by the CUM: (Ideal values are obtained from the camera's data sheet.)

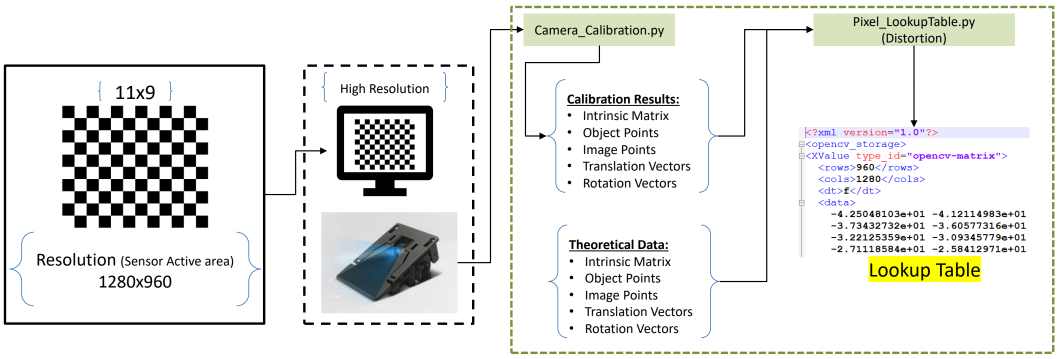
$$
Ideal\ intrinsic\ matrix = \begin{bmatrix} 1466 & 0 & 640 \\ 0 & 1466 & 480 \\ 0 & 0 & 1 \end{bmatrix}
$$

In addition to the *intrinsic calibration matrix*, a one-row matrix with five columns containing radial and tangential distortion parameters is also obtained:

$$
Distortion_{coef.} : \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix}
$$

$$
Ideal\ Distortion_{coef.} : \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}
$$

Figure 5.2 represents the overall process of generating a lookup table to distort ideal images.



```
<?xml version="1.0"?>
<opencv_storage>
<XValue type_id="opencv-matrix">
  <rows>960</rows>
  <cols>1280</cols>
  <dt>f</dt>
  <data>
    -4.25048103e+01 -4.12114983e+01
    -3.73432732e+01 -3.60577316e+01
    -3.22125359e+01 -3.09345779e+01
    -2.71118584e+01 -2.58412971e+01
```

Figure 5.2: Distortion model calibration.

The distortion-model calibration setup starts by projecting an 11x9 checkerboard on a high-resolution display. Using the testbed presented in Fig. 5.1, a number of pictures with the CUM from different distances and orientations is captured and fed to `Camera_Calibration.py`. After the calibration process has been completed, CUM's intrinsic matrix and distortion coefficients are obtained:

$$
intrinsic\ calibration\ matrix = \begin{bmatrix} 1484 & 0 & 655 \\ 0 & 1485 & 505 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$Distortion_{coef.} : \begin{bmatrix} 5.98 \times 10^{-2} & -0.56 & 1.02 \times 10^{-3} & -2.91 \times 10^{-4} & 0.96 \end{bmatrix}$$

The accuracy of the calibration process is then tested by computing the absolute L_2 Norm (Eq. (5.5)) between the position of reference checkerboard corners and projected corners using the obtained intrinsic and distortion matrices:

$$||ref - proj||_{L_2} = \sqrt{\sum_{i=1}^{n} ref(i) - proj(i))^2} \tag{5.5}$$

Where:

ref contains the position of reference corners in the image space.
$proj$ contains the position of projected corners in the image space.

Figure 5.3 represents the L2-Norm calculation for a total of 30 images. Based on the results obtained in Fig. 5.3, the calibration setup is considered to be accurate, where an total average error of 0.013 px is obtained. The reason for both outliers observed at images 7 and 30 is a failure in the corner detection algorithm to accurately detect the position of the checkerboard corners and not due to the calibration process.

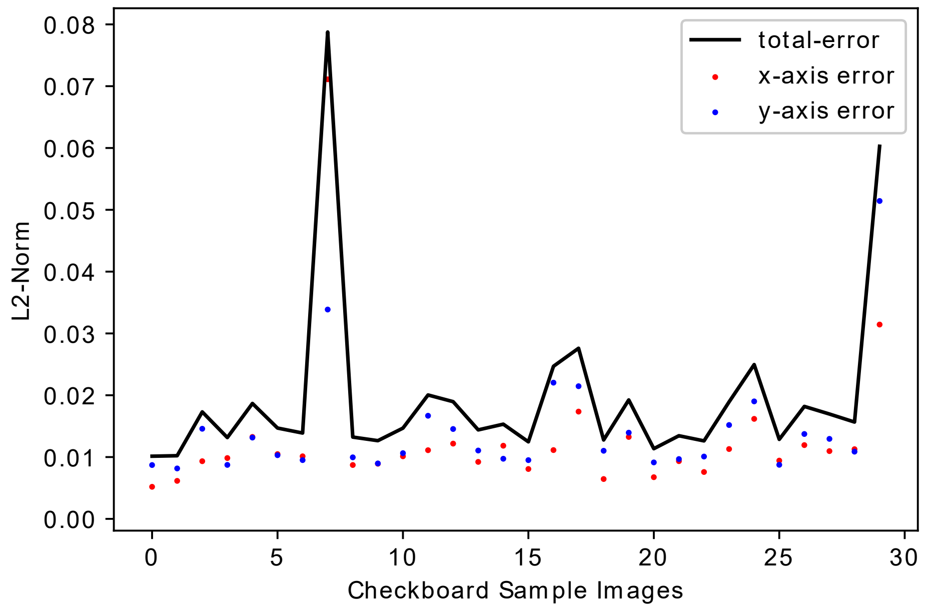


Figure 5.3: Calibration error.

It is essential to keep in mind that the aim of calculating the calibration and distortion matrices is not to calibrate the image, that is, they are not meant to remove distortions, but rather to introduce them on synthetic, undistorted images usually provided by any simulation software or rendering engine (Fig. 5.4).

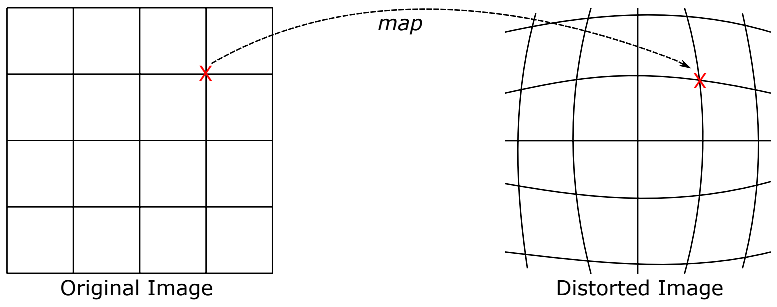


Figure 5.4: Pixel position mapping from original to distorted.

By using the camera calibration results and the theoretical data presented in Fig. 5.2, `Pixel_LookupTable.py` generates a lookup table that is used later on to map the pixels' position from original to distorted, i.e., as they would normally have been had they been captured by the CUM (using a lookup table implies that there is no need to calculate the pixels' position for each and every frame.)

In Fig. 5.5,the corners' position from a distorted image in red (image captured by the CUM) and that of the remapped corners in blue (using the lookup table) is presented. The overlay of red and blue circles indicates the accuracy of the model, yet in some places the circles do not overlay 100% (like in the magnified section), but the miss-projection error is always less than one pixel.

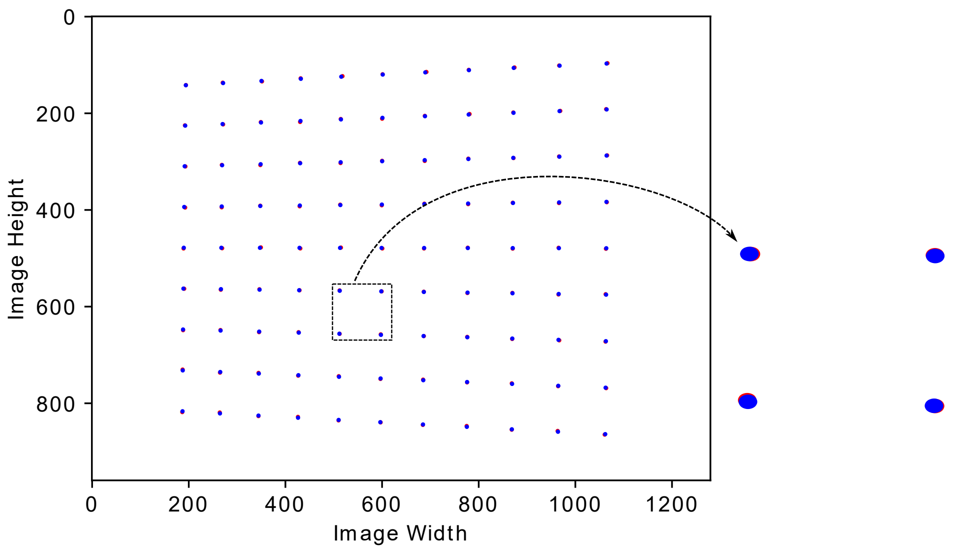


Figure 5.5: Mapping from ideal to distorted.

5.2.2 Vignetting Effect

Since modeling the light's intensity fall-off purely on physics/optical principles necessitates the knowledge of various parameters that cannot be easily obtained, the vignetting model and parametrization process is done experimentally with the help of the so-called integration sphere (Ulbricht-Kugel).

An integration sphere is an optical component, consisting of a hollow, spherical cavity with an interior covered with a diffuse, white, reflective coating, with small holes for entry and exit ports. Its most important property is the uniform scattering or diffusing effect [58]. The most significant characteristic of the integration sphere for the conception of the vignetting model is that light intensity at its opening surface is uniform. To further ensure this property, the opening of the integration sphere used for the experimental setup is fitted with a light diffuser that also ensures the homogeneity of the light's intensity at its surface. Theoretically, an image of the sphere's opening acquired by a camera should have uniform pixel intensity distribution throughout the entire image.

The experimental setup is presented in Fig. 5.6; a number of evenly distributed red, green, and blue LED arrays illuminate the integration sphere, and the camera under test is placed at a predefined distance from the light diffuser such that only the light diffuser is captured in its field of view.

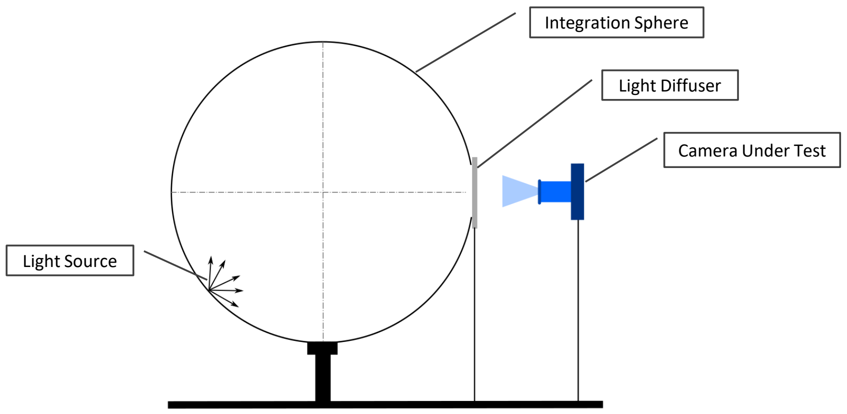


Figure 5.6: Experimental setup for the vignetting effect.

With the help of an LED color mixer, the color intensity of the LED arrays is altered to obtain a white color, and when using a frame grabber, a specific number of frames is captured — 100 in this case. The captured frames are averaged out in order to reduce the temporal noise. It is essential to point out that during this process the captured images should not be saturated at any pixel level, this can be ensured by constantly monitoring the pixels' intensity values.

In Fig. 5.7, the left picture represents the averaged image resulting from 100 frames, and on the right, the variation in average pixel intensity values per column is represented. When examining the graph presented in Fig. 5.7, it is observed that the maximum intensity is 0.683, whereas the minimum intensity is 0.570; this corresponds to approximately 16% drop in the average pixel intensity value. It is clear that this analysis may have been based on the radial position of pixels from the image center; however, for the sake of simplicity and because this does not have a direct effect on the derived model, presented variations are based on column averaging.

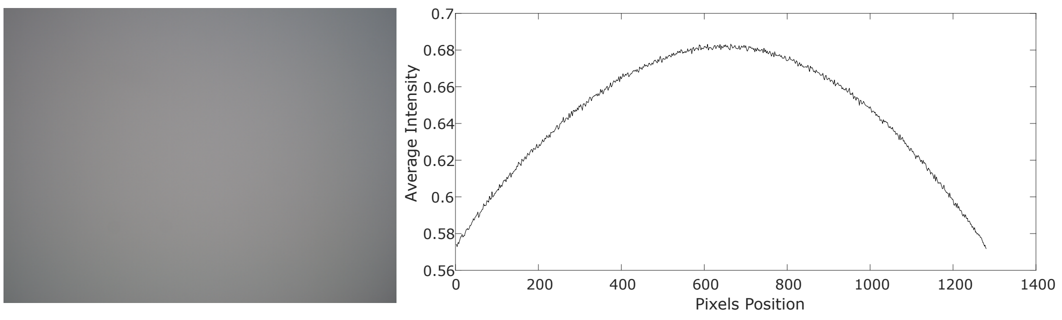


Figure 5.7: Averaged captured image and corresponding intensity graph.

With the maximum detected intensity value for each image channel, i.e., RGB, a uniform image is created and used as a reference to determine a pixel-wise lookup table that would indicate the intensity drop of each pixel for the vignetting model implementation. In Fig. 5.8, the reference image is represented, as well as its corresponding intensity graph, indicating a uniform illumination.

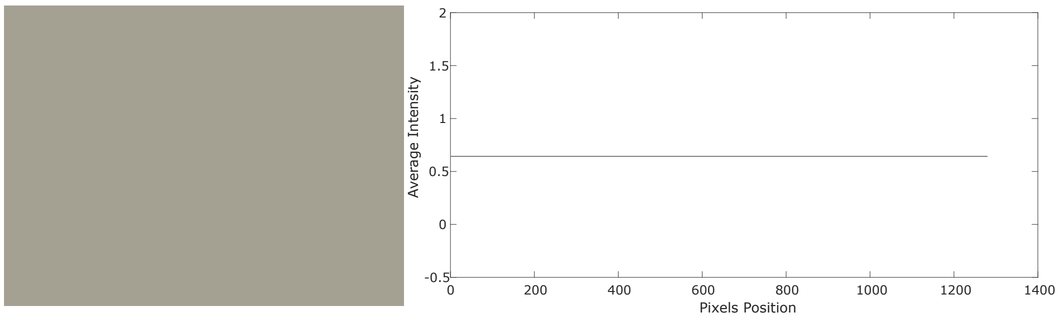
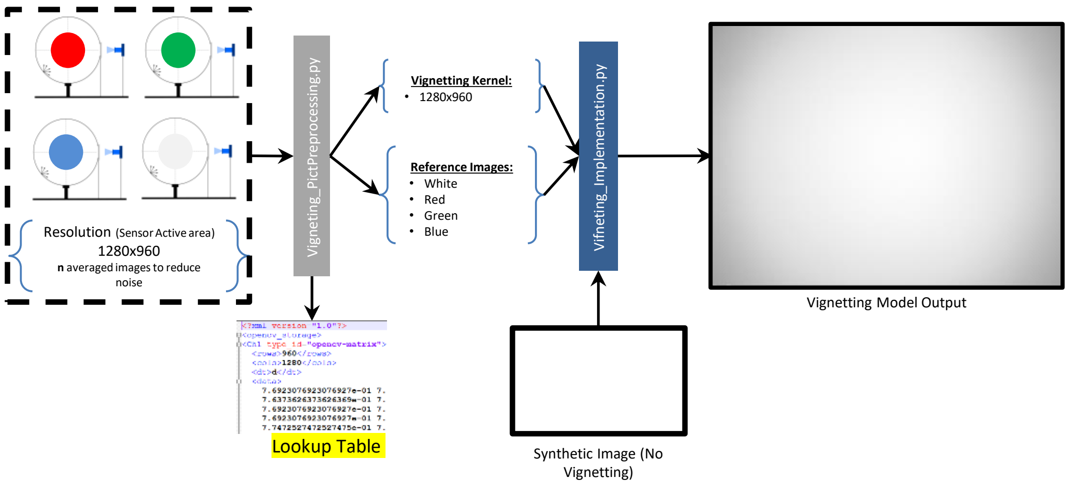


Figure 5.8: Uniform reference image and corresponding intensity graph.

The vignetting lookup table represents the pixel's relative intensity difference throughout the entire image. For example, if pixel $a(x, y)$ in the lookup table has a relative intensity value of 90%, then its value computed from a uniformly illuminated synthetic image of 0.7 intensity would be 0.63. Fig. 5.9 represents the vignetting effect's parameter calibration and test process.



```
<?xml version "1.0"?>
<opencv_storage>
<Ch1 type_id="opencv-matrix">
  <rows>960</rows>
  <cols>1280</cols>
  <dt>d</dt>
  <data>
    7.6923076923076927e-01 7.
    7.6373626373626369e-01 7.
    7.6923076923076927e-01 7.
    7.6923076923076927e-01 7.
    7.7472527472527475e-01 7.
```

Resolution (Sensor Active area)
1280x960
n averaged images to reduce noise
Vigneting_PictPreprocessing.py
Vignetting Kernel:
• 1280x960
Reference Images:
• White
• Red
• Green
• Blue
Vifneting_Implementation.py
Vignetting Model Output
Lookup Table
Synthetic Image (No Vignetting)

Figure 5.9: Vignetting model calibration.

The vignetting model is tested by comparing the average pixel intensity per column between a picture taken by the CUM and another picture created by the vignetting model. The target picture is a uniformly illuminated white screen, so a reference image for the model to augment is made available.

Figure 5.10 represents the average pixel intensity difference between the image generated by the vignetting model and that of the CUM. The total average error in pixel intensity difference is a negligible 0.0334 (0.49 % of the maximum intensity), and the average error per channel is represented below:

- Red Channel: 0.0272.
- Green Channel: 0.0326.
- Blue Channel: 0.0404.

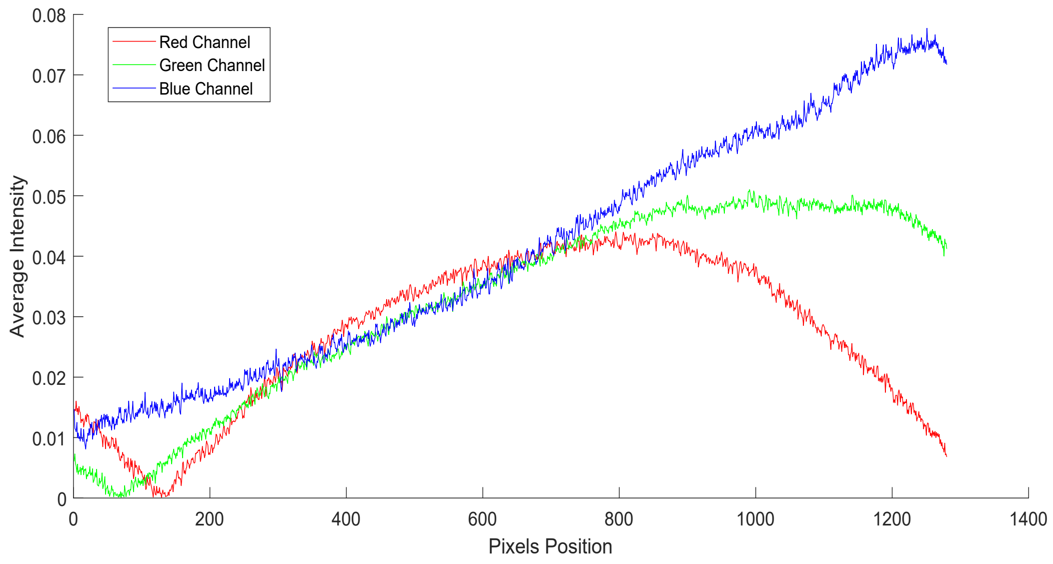


Figure 5.10: Average pixel intensity difference between vignetting model and CUM output.

5.2.3 Lens Flare and Ghost Effect

As previously mentioned in section 4.3.1, only the ghost effect resulting from the lens's aperture and internal reflections between optical elements is considered. The absence of a lens-system simulation prevents such a model from conveying a high level of realism to the rendering process, and so, only the ghost effect resulting from very bright lights, like sun or headlights, is considered. For example, in Fig. 5.11 a bright source of light (e.g. the sun) results in a circular ghost image that partially occludes the right taillight of the vehicle in Fig. 5.11 (a), and the entire vehicle in Fig. 5.11 (b).

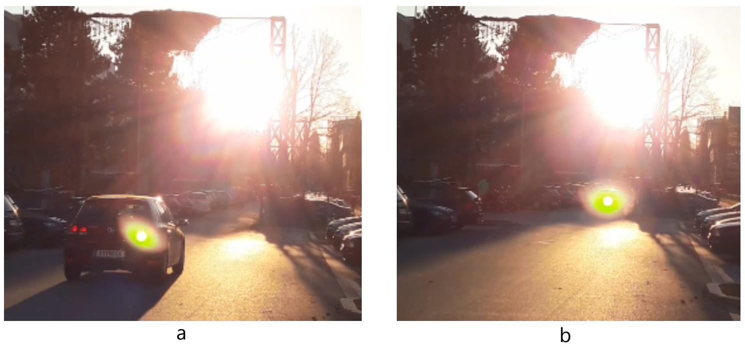


Figure 5.11: Ghost effect due to direct sun light.

Every time a ray of light hits an interface between two media, a part of it is reflected, while the rest is transmitted. It is the reflected part that gives rise to ghosting artifacts [59]. Reflected rays in conjunction with the aperture size and shape determine the ghost's shape and position in the image. For ADAS cameras, optical modules with circular aperture and fixed diameter are generally used.

For the modeling process, an experimental setup that consists of the CUM and bright light sources is constructed. With the help of the testbed, the CUM's position and orientation with respect to the plane that contains three bright light sources are controlled (Fig. 5.12). Ideally, only two light sources are needed; however, a third light source is used for control.

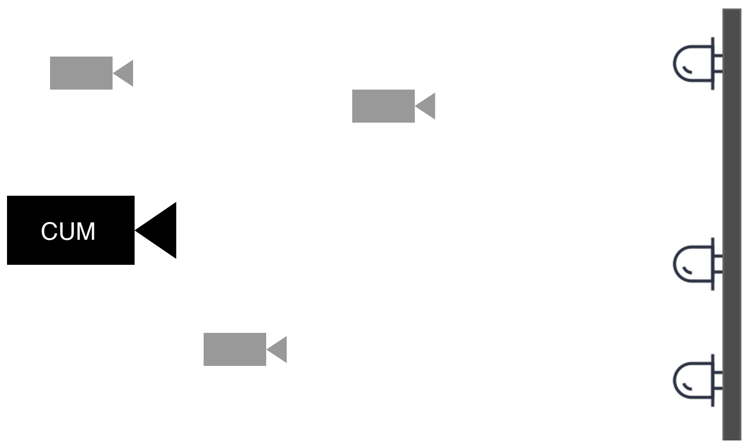


Figure 5.12: Experimental setup for ghost effect calibration.

Using a frame grabber, several pictures are captured, and the relative position of the Light-Emitting Diodes (LEDs) with respect to their ghost image is analyzed. In Fig. 5.13, the bright LEDs and their ghost image are represented, where each light source is connected to its ghost image by a yellow line.

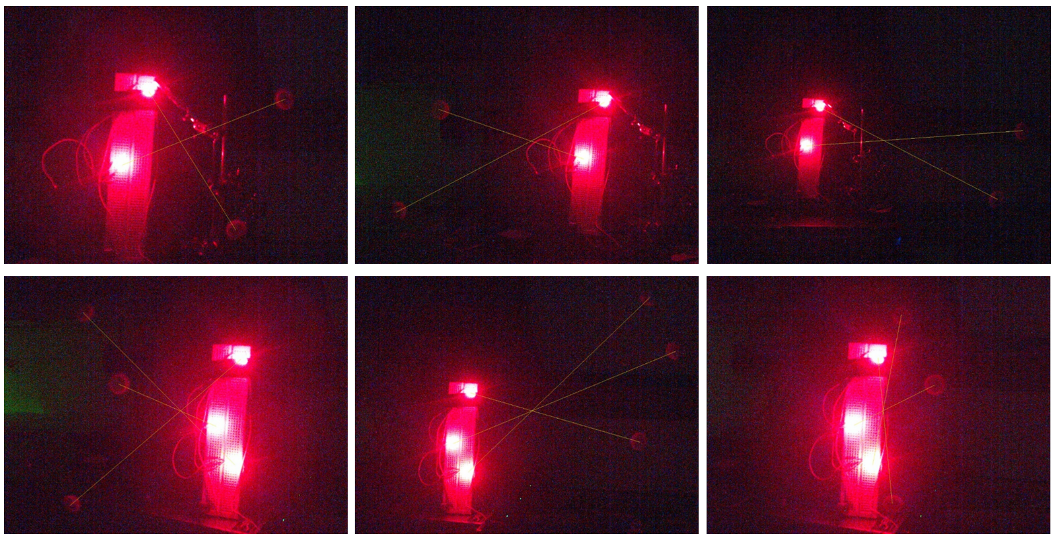


Figure 5.13: Ghost effect from the experimental setup.

After examining the pictures presented in Fig. 5.13, it becomes clear that the intersection point of the lines connecting two LEDs with their ghost image is independent of the camera's position and orientation, which is a clear indication that the ghost effect is due to internally reflected light in the optical module. The circular shape of the ghost image results from the camera having a circular aperture. By performing a number of measurements, the coordinates of the intersection point are defined, in addition to the ratio of the distance from the reflection point to the LED and its ghost image. In this setup, the third LED and its ghost image are used to verify that the line connecting them also passes through the same intersection point, thus not only validating its coordinates, but the aspect ratio as well.

For the current CUM, the calculated reflection point's coordinates are 658 and 502 in x and y respectively (origin: bottom left), and the aspect ratio is 0.557, which means that the distance from the reflection point to the ghost image is larger than that to the light source. Furthermore, the circle of influence where the ghost effect could occur is defined and represented in Fig. 5.14, where the red crosses represent the bright light source and the blue ones represent the ghost image. In Fig. 5.14, the circle of influence has its center at the reflection point; its radius (347 pixels) defines the maximum distance where the ghost of a bright light source will still be present. In the presented image, the calculated radius is considered with respect to the distance between the reflection point and the top right corner of the image, i.e., a light source may still be present inside the circle, but its ghost image is outside the image boundaries.

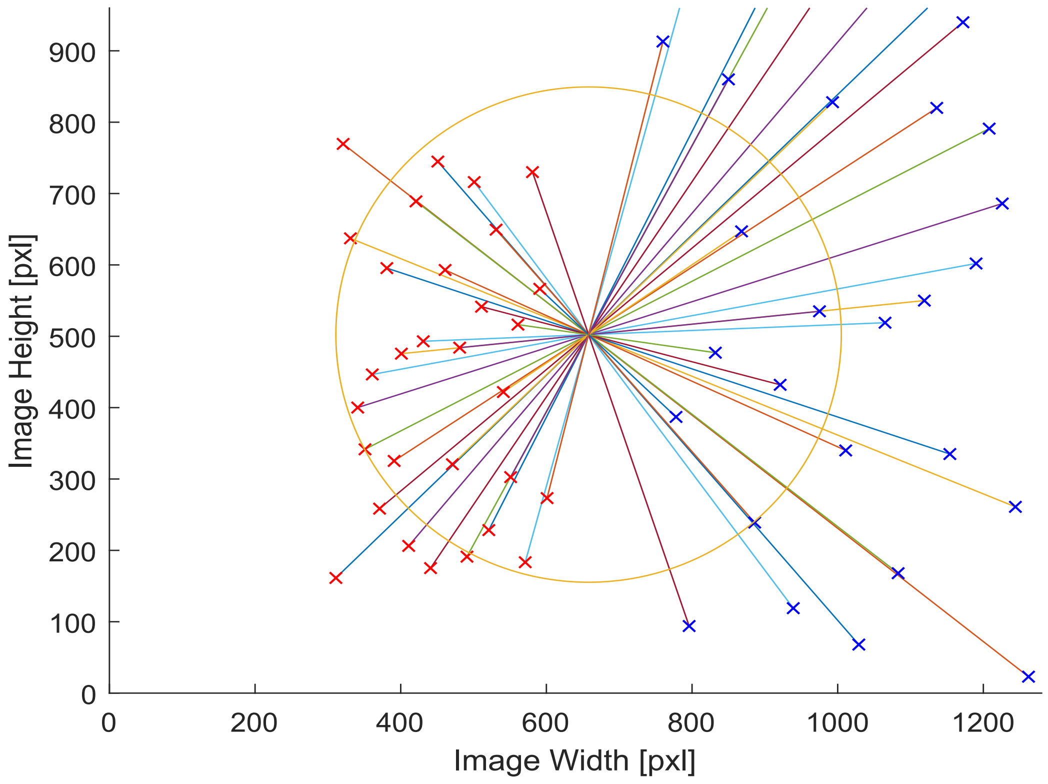


Figure 5.14: Ghost effect circle of influence.

5.3 Modeling Camera Blur

As previously described in Fig. 4.4, camera blur is due to several optical and image sensor effects, that is, the optical and imager modules both contribute to its existence in the final, acquired image. For this reason, this part is separated from sections 5.2 and 5.4. In the following, a rough estimate of the PSF is experimentally derived.

The experimental setup consists of a point light source, such as a laser pointer, a black background, and the CUM. With the help of a laser pointer, a dot of light is projected on a black background, and its diameter is measured. By adjusting the distance between the CUM and the black background plane, the projected size of the laser dot is controlled in such a way that it is smaller than the pixel pitch; this is necessary so that the PSF of the camera (and not that of the laser pointer) is measured.

In Fig. 5.15, the laser pointer's diameter, S_{object} is 2 mm and its spread diameter, S'_{object} is 6 mm. In Eq. (5.6), S'_{object} is considered in order to make sure that the total size of the point light source is smaller than the pixel's pitch size, i.e., less than 3.75 µm for the current CUM.

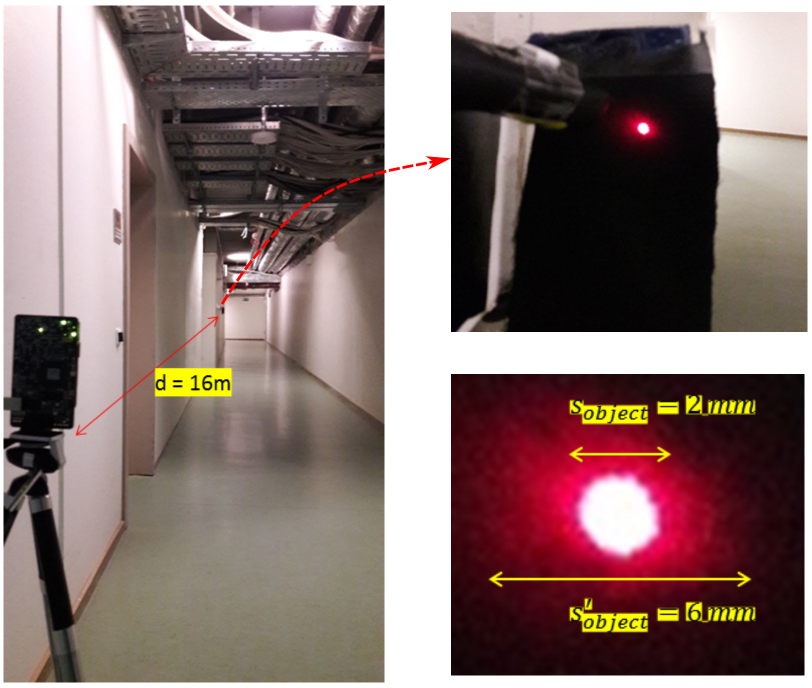


Figure 5.15: Camera blur experimental setup.

$$S_{image} = \frac{f * S'_{object}}{d} \tag{5.6}$$

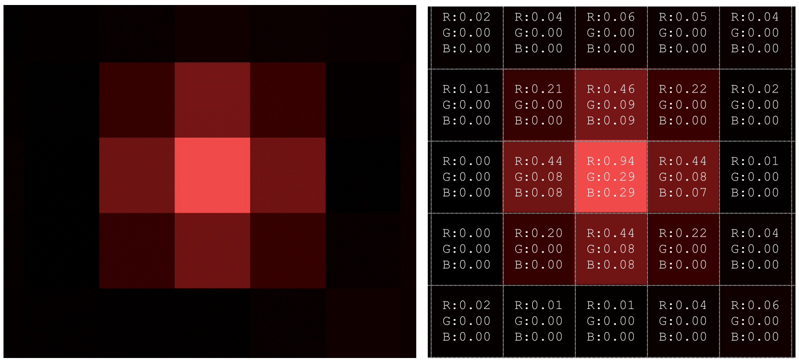
Where:

S_{image} is the laser dot size in the image space [mm].
$f = 1485 * 3.75\ \mu m \approx 5.56mm$ is the focal length [mm].
S'_{object} is the laser point spread diameter [mm].
d is the distance between the camera and the laser dot [mm].

Using Eq. (5.6) and placing the camera at a distance $d = 16 \times 10^3\ mm$ results in:

$$S_{image} \approx 2.1\,\mu\text{m} < \ 3.75\ \mu m$$

Fig. 5.16 represents the PSF of the camera when capturing a laser dot at a distance $d = 16 \times 10^3\ mm$. The average pixel intensity values are computed for a set of captured images covering various positions at the image corners, center, and other intermediate locations.



R:0.02 G:0.00 B:0.00	R:0.04 G:0.00 B:0.00	R:0.06 G:0.00 B:0.00	R:0.05 G:0.00 B:0.00	R:0.04 G:0.00 B:0.00
R:0.01 G:0.00 B:0.00	R:0.21 G:0.00 B:0.00	R:0.46 G:0.09 B:0.09	R:0.22 G:0.00 B:0.00	R:0.02 G:0.00 B:0.00
R:0.00 G:0.00 B:0.00	R:0.44 G:0.08 B:0.08	R:0.94 G:0.29 B:0.29	R:0.44 G:0.08 B:0.07	R:0.01 G:0.00 B:0.00
R:0.00 G:0.00 B:0.00	R:0.20 G:0.00 B:0.00	R:0.44 G:0.08 B:0.08	R:0.22 G:0.00 B:0.00	R:0.04 G:0.00 B:0.00
R:0.02 G:0.00 B:0.00	R:0.01 G:0.00 B:0.00	R:0.01 G:0.00 B:0.00	R:0.04 G:0.00 B:0.00	R:0.06 G:0.00 B:0.00

Figure 5.16: Laser point spread as seen by CUM.

From the computed intensity values of several images acquired by the setup described in 5.15, the PSF is estimated, by taking the average values over all images, with a 3x3 kernel represented below:

$$PSF\ kernel = \begin{bmatrix} 0.03734 & 0.12685 & 0.04564 \\ 0.12448 & 0.32128 & 0.14404 \\ 0.03852 & 0.12092 & 0.04090 \end{bmatrix}$$

Taking into consideration that the CUM is a fixed-focus camera, the camera is most probably focused at the **hyperfocal distance**, which means that the camera will hold a depth of field from $H/2$ to infinity (H being the **hyperfocal distance**). For this reason, the distance-dependent blur model is not considered and only the PSF kernel approximation is used for image blur; this means that objects at a closer distance than $H/2$ will not be correctly blurred. Fig. 5.17 represents the PSF kernel in a 3D view where the z-axis indicates the intensity value in x and y directions.

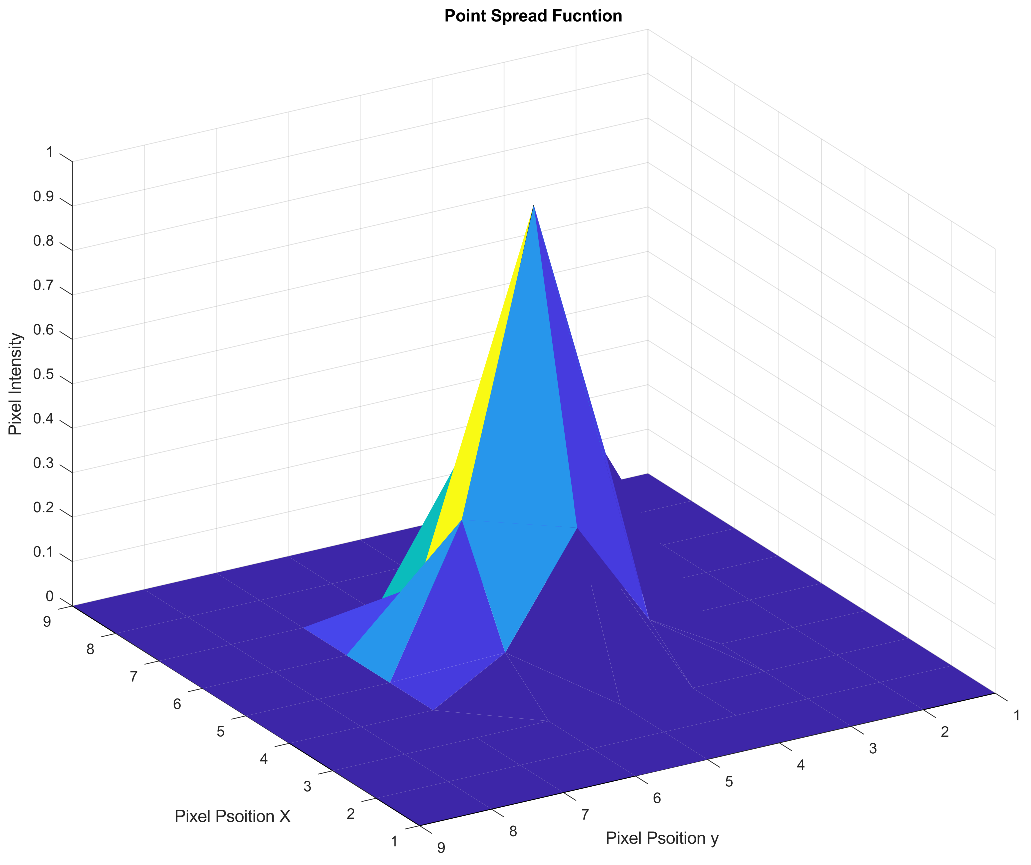
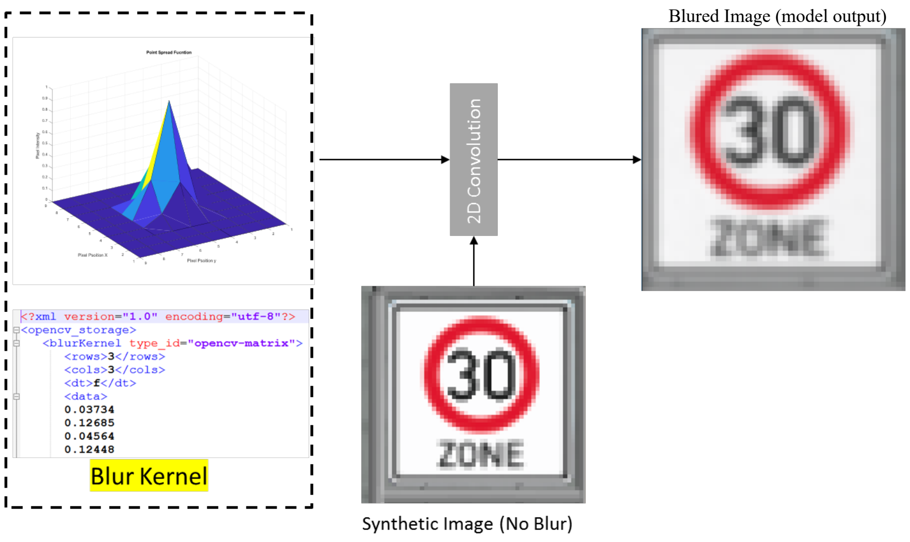


Figure 5.17: PSF kernel representation.

In Fig. 5.18, a 2D convolution implements the blur kernel on a synthetic image that was previously generated by a simulation software. The result is a blurred image that reflects the PSF of the CUM.



```
<?xml version="1.0" encoding="utf-8"?>
<opencv_storage>
   <blurKernel type_id="opencv-matrix">
      <rows>3</rows>
      <cols>3</cols>
      <dt>f</dt>
      <data>
      0.03734
      0.12685
      0.04564
      0.12448
```

Figure 5.18: Blur model generation.

5.4 Modeling Image Sensor Effects

Since the main goal is to show that physics-based/behavioral models are necessary for further ADAS/AD development and not to completely model all effects, only some image sensor effects such as noise, rolling shutter, and LED flicker effects are considered. Moreover, only temporal and spatial noises in the dark are modeled.

Before starting with temporal and spatial noise modeling, the sensor's output is studied beforehand; this is necessary in order to check for dead pixels or fixed defects in the sensor's image array. The measurement process starts by covering the camera's lens, and then placing it in a dark room, ensuring that no light can reach the image sensor. By taking a set of pictures at various exposure times (0-100 ms) and during different timestamps (over 10 sec), pixels values are monitored and tracked over time. Pixels with intensity values that are greater than zero and that are relatively constant over time are selected (a total of 53 px). The pixels' mean values and standard deviation are presented in Fig. 5.19, where it becomes apparent how the standard deviation of each pixel is miniscule with respect to the corresponding mean value. From this observation, the detected pixels are ignored in temporal and spatial noise modeling. In Fig. 5.20, the defected pixels are highlighted to the left (where the entire image is represented) and to the right, where zoomed-in sections are presented (the rest of the pixels are temporarily set to zero).

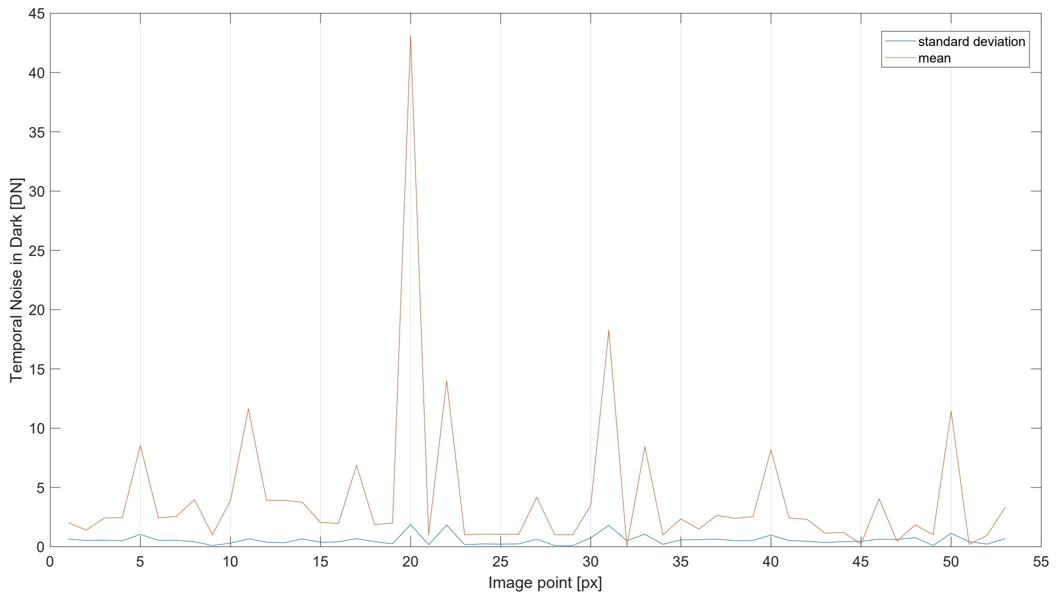


Figure 5.19: Standard deviation and mean values of defected pixels.

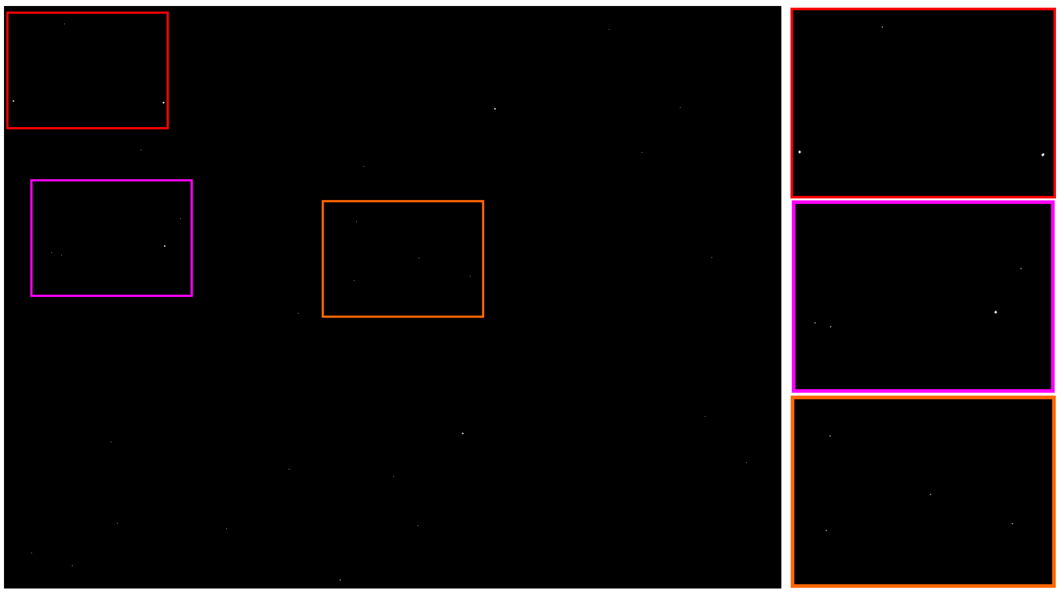


Figure 5.20: Visual representation of defected pixels.

5.4.1 Temporal Noise

In this section, temporal noise in the dark for the CUM is measured and modeled for implementation as a Probability Density Function (PDF). Only temporal model in the dark is considered, so the PDF will eventually represent the read and dark current/thermal noise.

For the measurement procedure, and in order to minimize the influence of DSNU, the integration time is kept as low as possible (in this case 11 µs). Moreover, and to reduce variations in the dark current of the pixel, the sensor's temperature (via an onboard sensor) is monitored and ensured that it is kept constant during the image acquisition process.

The process starts by capturing several images (30-50) to bring the camera into a stable state, and then by capturing a set of images (100 in this case) and storing them in raw format. The root mean squared error (Eq. (5.7)) for each pixel is then calculated, and the histogram representation of the calculated errors is generated.

$$Noise_{rms} = \sqrt{\frac{\sum_{n=1}^{N} \mid p_n \mid^2}{N}} \tag{5.7}$$

Where:

N is the total number of pixels.
p_n is the n^{th} pixel intensity values [DN].

In Fig. 5.21, the histogram representing the noise on pixel level is extracted. The presented distribution shows a mean value of $\lambda = 199.4$, which is also obtained when trying to fit the $Noise_{rms}$ values into a *Poisson* distribution. Note that in Fig. 5.21, since the y-axis has a scale factor of 10^5, the number of pixels with a temporal noise value less/grater than 200 Digital Number (DN) may appear to be zero in the printed out version of this dissertation. This is also true for other distributions presented in Fig. 5.22, Fig. 5.24, and Fig. 5.25.

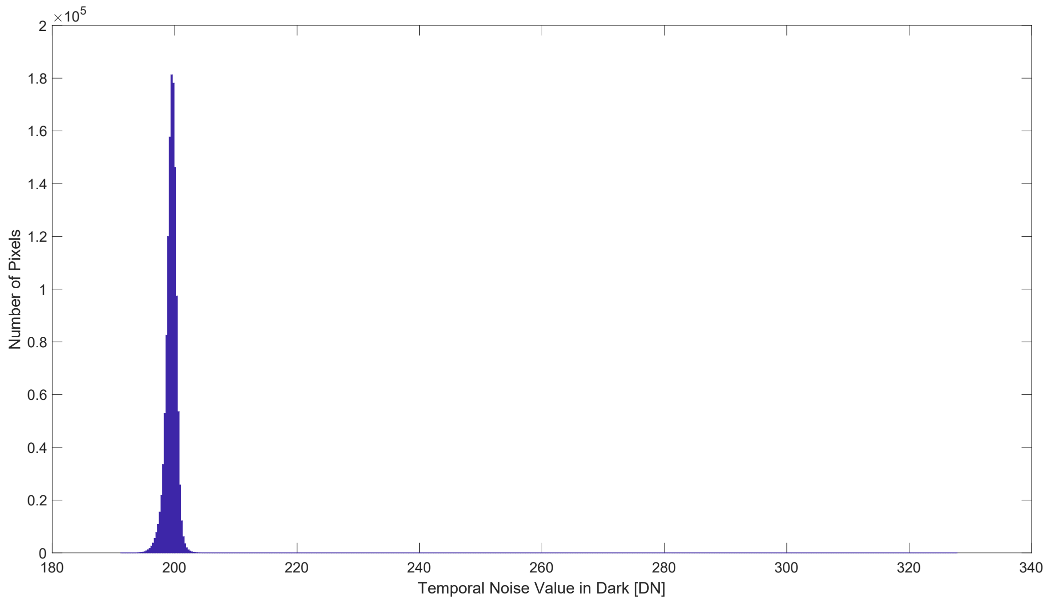


Figure 5.21: Histogram representation of temporal noise in the dark.

To model the calculated noise, random variables from the *Poisson* distribution with a mean value $\lambda = 199.4$ are generated. Considering the sensor's resolution, a total of 1280x960 pixel values are randomly generated. The distribution of the generated values is represented as a histogram diagram in Fig. 5.22.

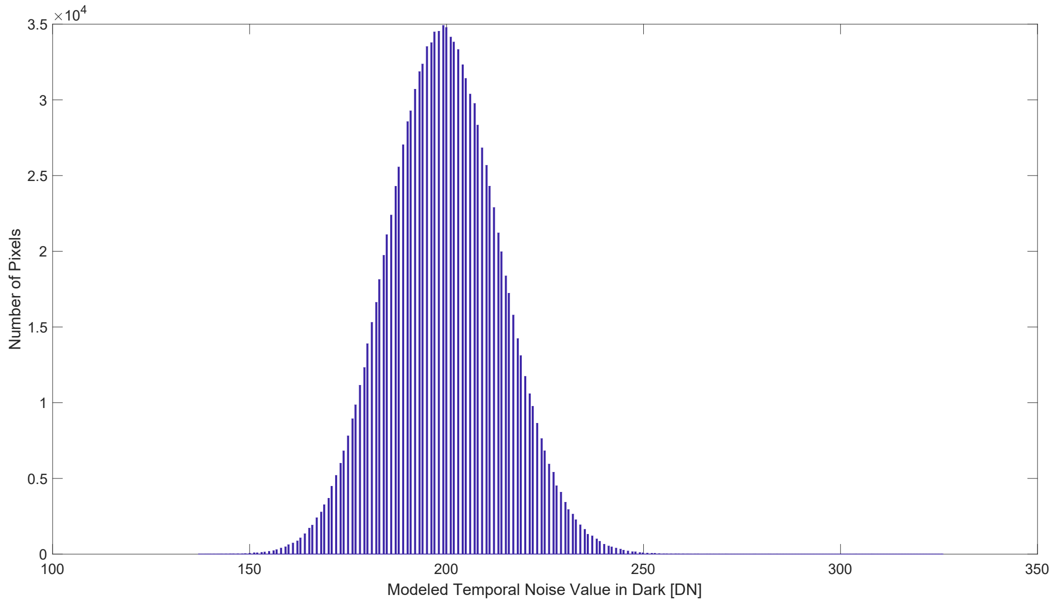


Figure 5.22: Histogram representation of modeled temporal noise in the dark.

To better enhance the model's approximation of temporal noise, the generated values are then passed through a low-pass filter, thus reducing the effect of randomly generated high frequencies. Fig. 5.23 (a) represents the ground truth noise generated by the sensor, Fig. 5.23 (b) displays the model's output without the low-pass filter, and finally, the filtered model's output is represented in Fig. 5.23 (c), which shows a better noise approximation.

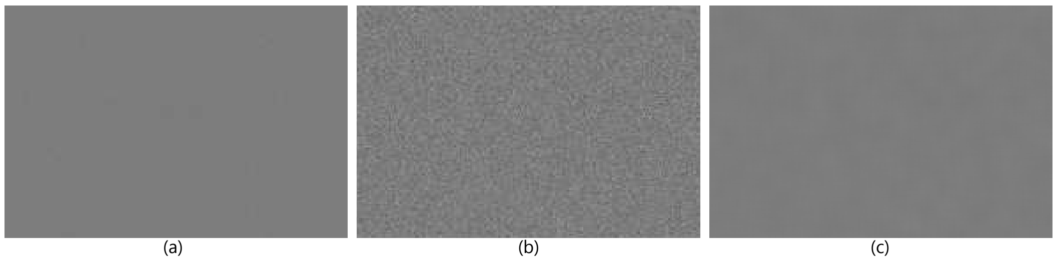


Figure 5.23: Visual representation of temporal noise, (a) ground truth noise generated by the sensor, (b) model's output without the low-pass filter, (c) model's output with a low pass filter.

5.4.2 Spatial Noise

Spatial noise represents the variations of the dark signal from one pixel to another in the same frame. Similar to temporal noise measurement, the temperature of the image sensor is carefully monitored and maintained constant throughout the image acquisition process.

The modeling process starts by capturing several images at different exposure times (0-100 ms); at each exposure time, several images are taken and averaged in order to decrease the influence of thermal noise. With the obtained images, the averaged spatial noise over the entire image is calculated for each exposure time. It is observed that for the current CUM, the spatial noise variation from one exposure time to the other is minimal; therefore, using Eq. (5.7) the Root Mean Square (RMS) error is calculated with respect to each exposure time, and a histogram representation is generated (Fig. 5.24).

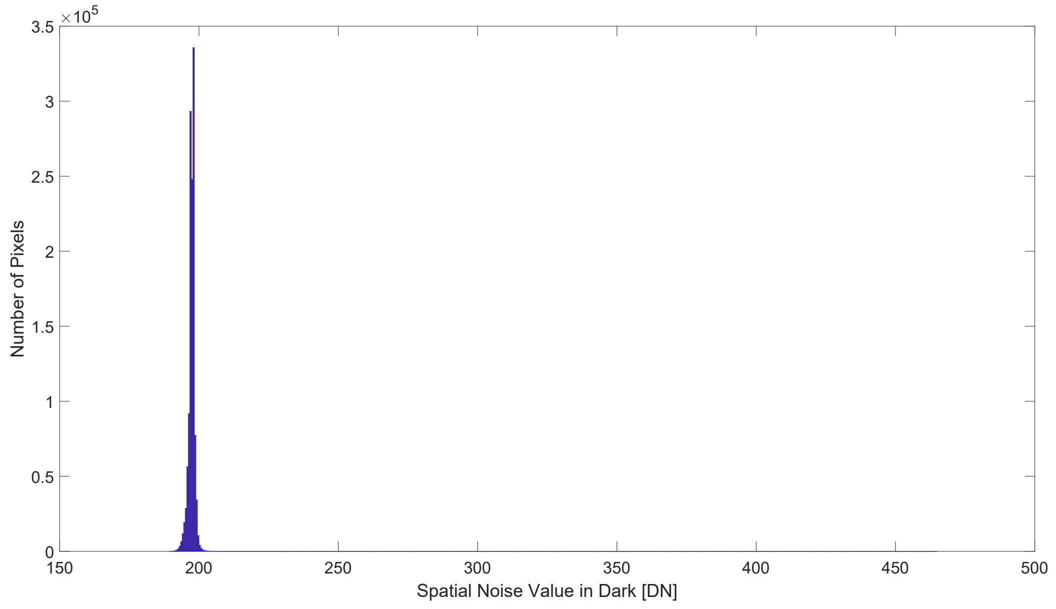


Figure 5.24: Histogram representation of spatial noise in the dark.

Following the same technique illustrated in section 5.4.1, with $\lambda = 197.3$, random spatial noise values that follow *Poisson's* distribution are generated and later passed through a low-pass filter for a better approximation of spatial noise.

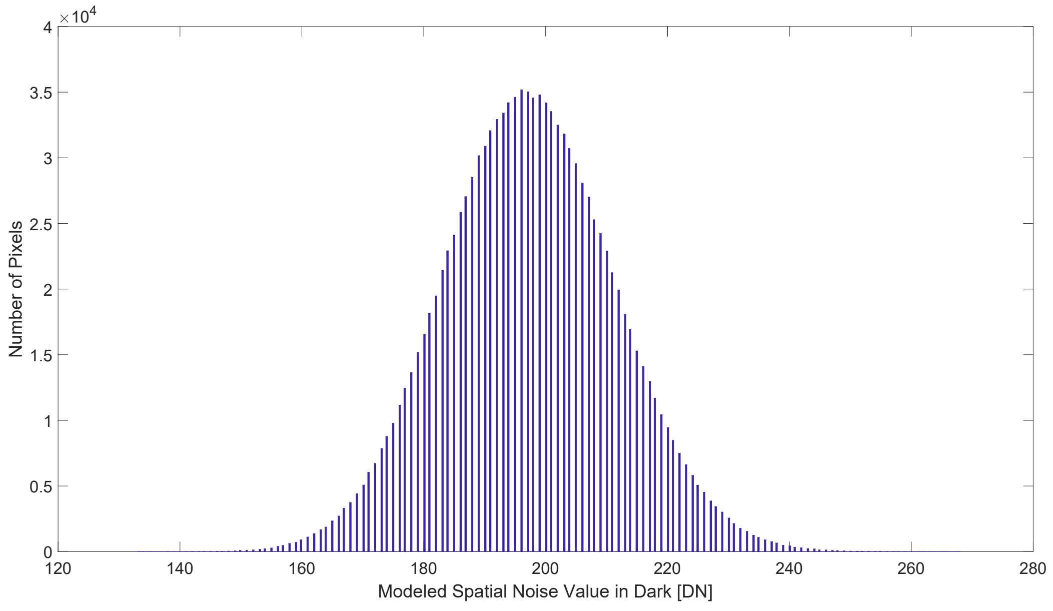


Figure 5.25: Histogram representation of modeled spatial noise in the dark.

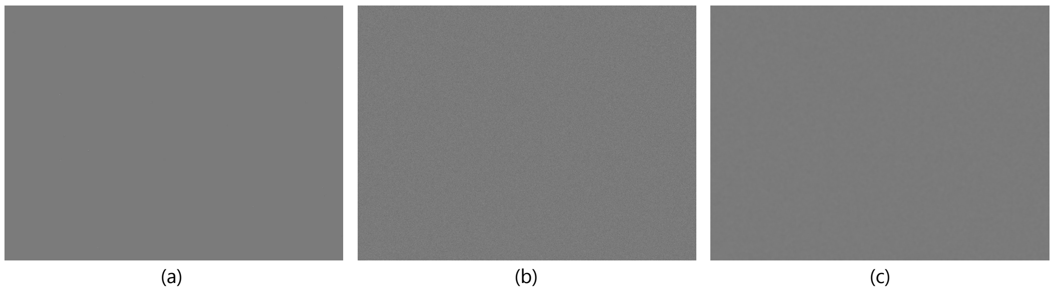


Figure 5.26: Visual representation of spatial noise, (a) ground truth noise generated by the sensor, (b) model's output without the low-pass filter, (c) model's output with a low pass filter.

5.4.3 Rolling Shutter and LED Flicker

A CMOS imager often adopts the electronic rolling shutter, where each row of a CMOS is exposed at different time intervals. Therefore, a fast relative motion between the camera and moving objects inside the FOV results in image distortions, which is referred to as the rolling shutter effect. In Fig. 5.27 a spatial illustration of image readout for the CUM is represented; horizontal and vertical blanking together represent the total blanking time between two consecutive frames. As illustrated in Fig. 5.28, the readout window is typically set to a region including only active pixels. Thus, horizontal and vertical blanking are excluded but still contribute to the total frame time, which can be calculated using Eq. (5.8).

$$F = V_{image} + H_b + V_b \tag{5.8}$$

Where:

F is the total frame time [ms].
V_{image} is the valid image readout and exposure time [ms].
H_b is the horizontal blanking time [ms].
V_b is the vertical blanking time [ms].

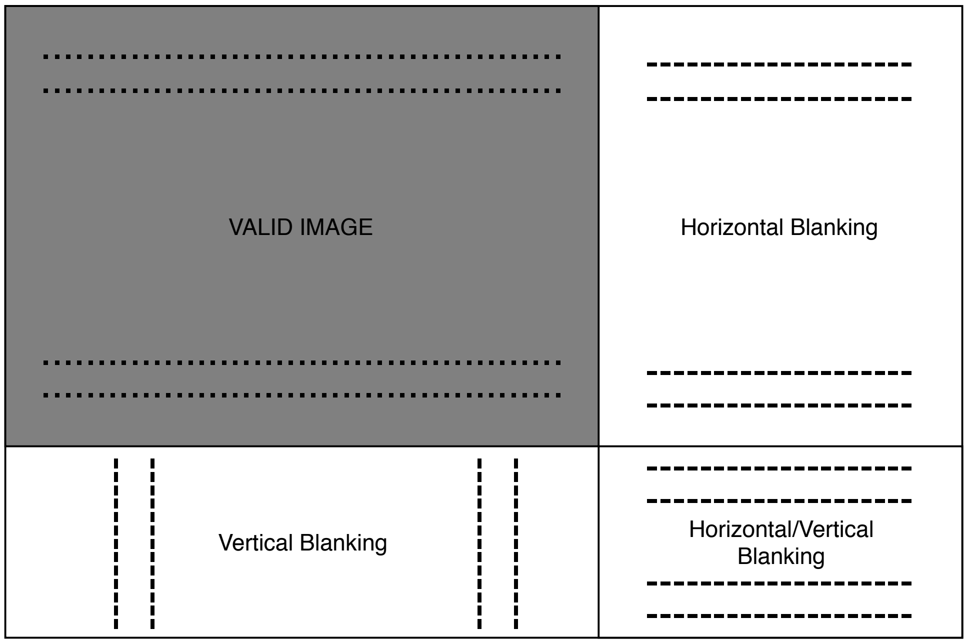


Figure 5.27: Illustration of image readout.

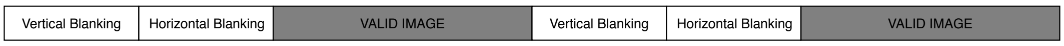


Figure 5.28: Pixel output timing.

When modeling the rolling shutter effect for a digital image with an active-pixel array of H x V px and a total frame time F, the spatial distribution of the image readout should, ideally, be taken into consideration (Fig. 5.27). Usually, the user does not have access to such information, and only primary data like frame rate and image resolution is provided. In the following, these factors are considered, and an abstract model of the rolling shutter effect is generated based only on the fps and image resolution. The model is tested and verified by an experimental setup.

For the experimental setup, a white disk with a black mark and a diameter of 48 mm is mounted on a brushless DC motor capable of 18,000 rotation per minute (rpm). The CUM is then mounted on a tripod and positioned above the spinning disk, as represented in Fig. 5.29. The motor's speed is controlled via a dedicated motor control board with a tolerance value of ± 5 rpm; more details on the theoretical concept can be found below [60].

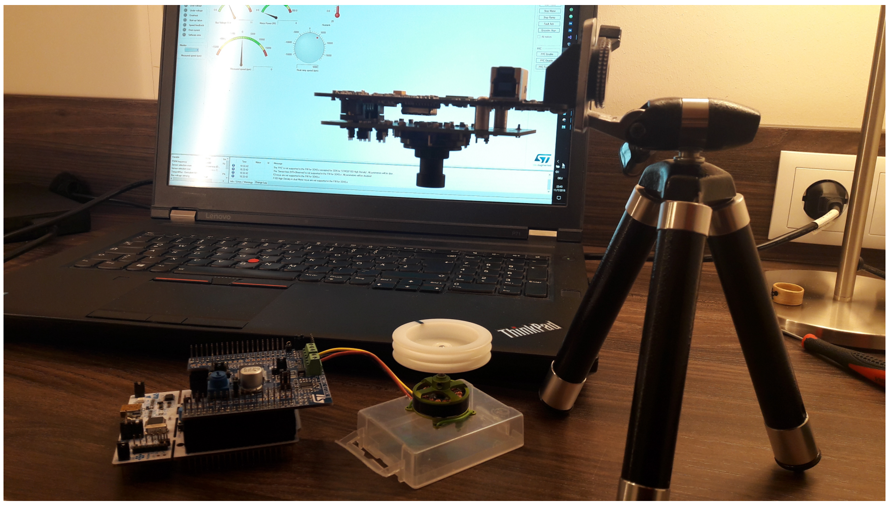


Figure 5.29: Rolling shutter experimental setup.

Fig. 4.12 represents the disk in a stationary situation and the rolling shutter effect when the disk is spinning at 4362 rpm and the camera is set to 30 fps. The active-pixel array of the image is set to 1280H x 960V. In Fig. 5.30, a virtual representation of the disk is generated with the exact black mark on it (the black circle is just for reference).

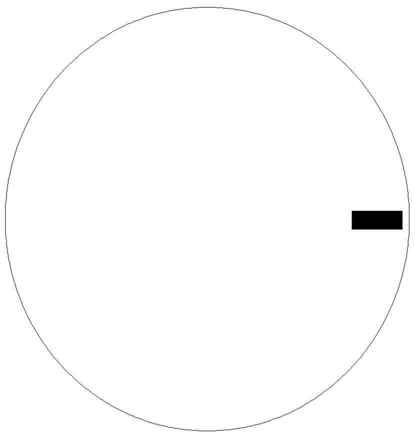


Figure 5.30: Virtual replica of the white disk.

By approximating the pixel line integration time $(\frac{1}{FPS \times V})$ and the speed of the motor, the radial position of the black mark can be calculated at all times, as well as its projection on the image sensor. Following the pixel readout timing scheme, a virtual representation of the black dot is represented in Fig. 5.31 with respect to the rolling shutter mechanism. To ensure that the model is providing the correct results, Fig. 5.32 represents the model, with the camera's output at the same frame rate but at a higher rpm (5220). The rolling shutter effect, visualized by the distortion patterns of the black marks, is identical.

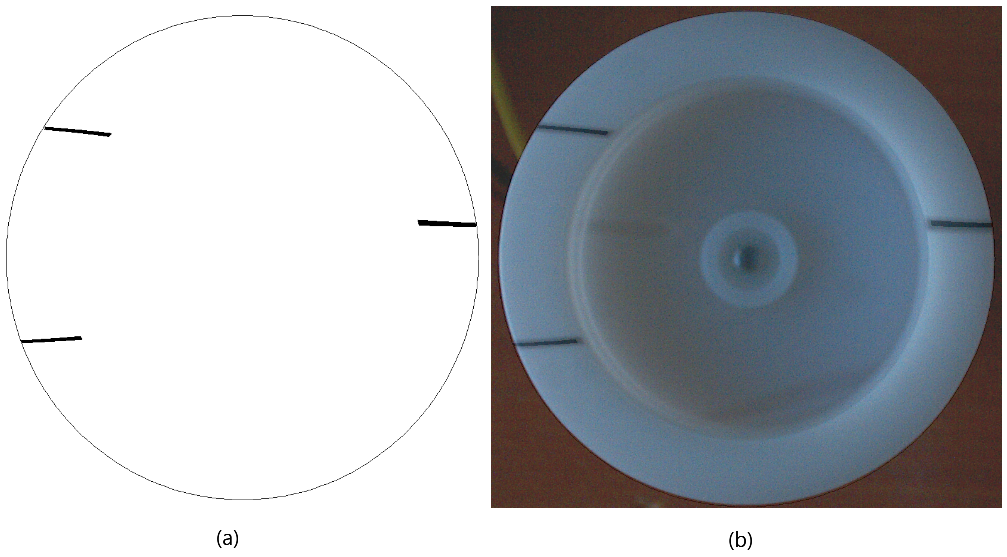


Figure 5.31: Rolling shutter pattern of the white disk 30 fps, 4362 rpm; (a) simulated, (b) as acquired by the camera.

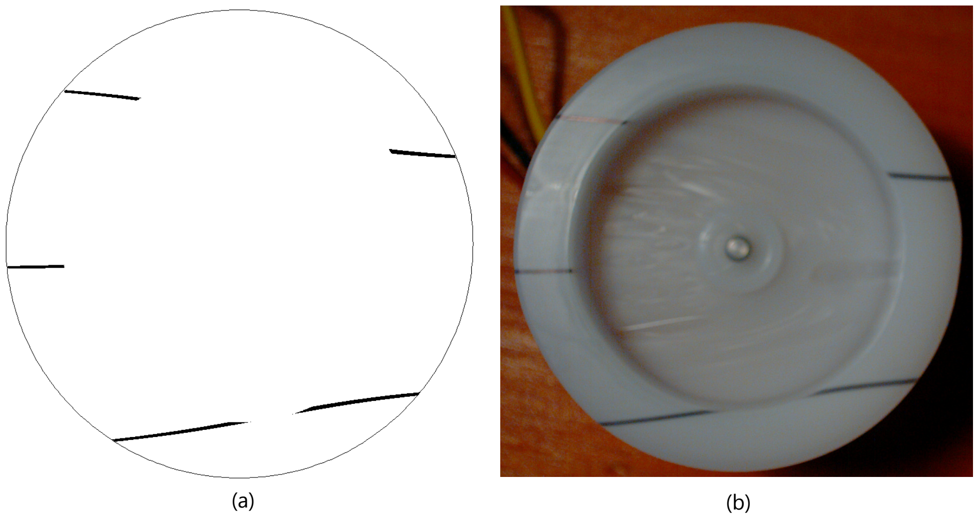


Figure 5.32: Rolling shutter pattern of the white disk 30 fps, 5220 rpm; (a) simulated, (b) as acquired by the camera.

In the context of this dissertation, the rolling shutter effect is not directly applied on the pixel level, but on the object list level. This is because the camera model's input is already a pre-rendered image (see 4.4.2). The model is adapted to accommodate lateral and transversal movements of target vehicles inside the camera's FOV. For example, in Fig. 5.33 (a), the detected vehicle — shown in Fig. 5.33 (b) — is represented. In an ideal simulation (no rolling shutter effect), the detected vehicle's bounding box will always be represented as in Fig. 5.33 (a); however, with the rolling shutter effect for a vehicle moving at 100 km/h towards the ego car, the bounding box will take the shape represented in Fig. 5.34, which will eventually have an altered center of mass.

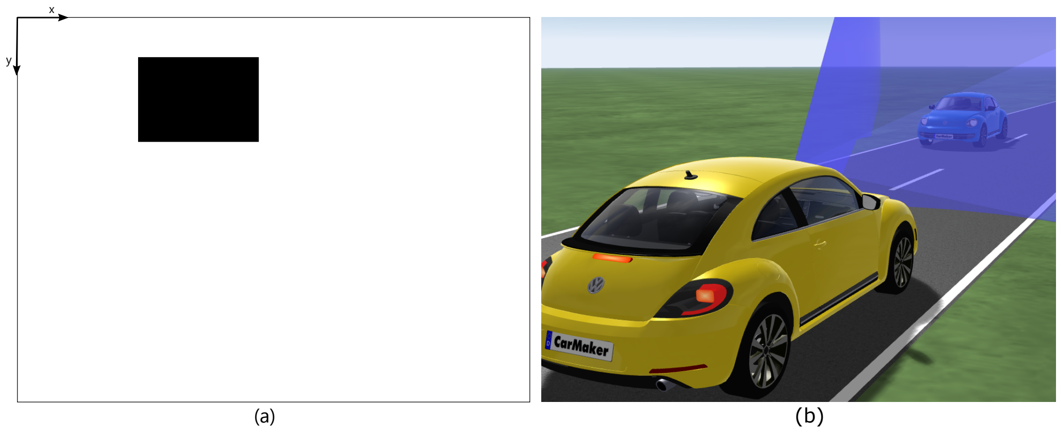


Figure 5.33: (a) bounding box representation of detected vehicle without the rolling shutter effect, (b) corresponding virtual scenario.

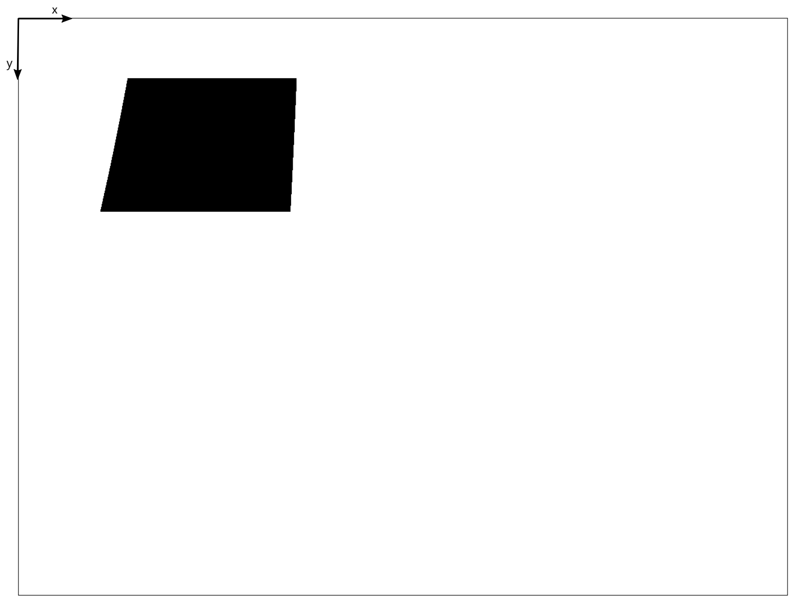


Figure 5.34: Bounding box representation of detected vehicle with the rolling shutter effect.

As for the LED flicker effect, the modeling approach relies on the sensor's exposure time as well as the Pulse Width Modulation (PWM) signal provided by a microcontroller development board, which drives the light source. The effect could lead to a false negative in a driving scenario (Fig. 5.35), where "ON" LEDs may be perceived as "OFF". It is important to point out that the probability of this behavior largely depends on how a bundle of LEDs (forming a traffic sign/light) is driven, and whether the condition described by Eq. (4.3) is satisfied or not. A more probable situation is the one represented in Fig. 5.36, where Eq. (4.4) is satisfied, and anti-flicker measures are taken either on the camera's side or that of the LEDs. Fig. 5.37 represents a rough output of a traffic sign captures at 30 fps; the traffic sign is driven by a PWM signal of 250 Hz and 20% duty cycle.

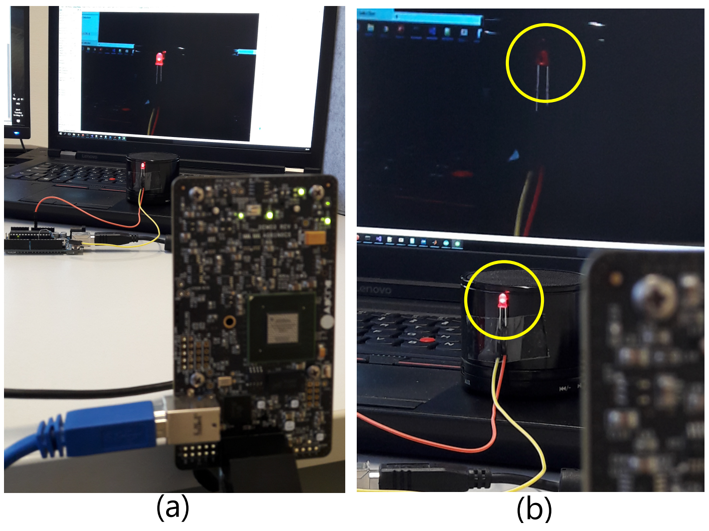


Figure 5.35: LED flicker effect on one LED.

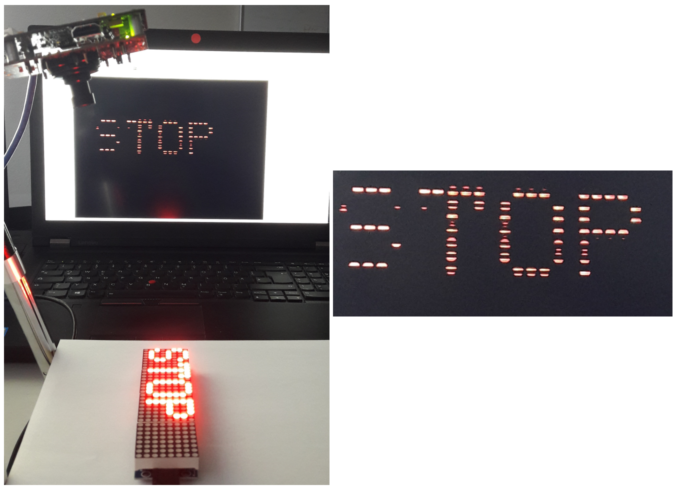


Figure 5.36: LED flicker effect on an LED matrix.

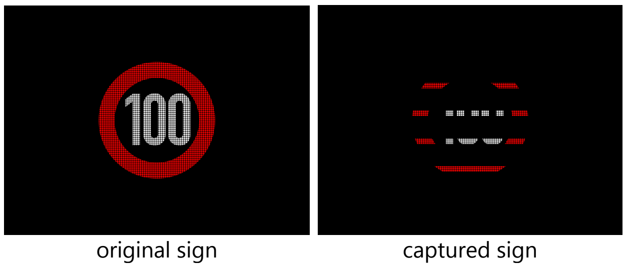


Figure 5.37: LED flicker effect, $frequency = 250\,Hz\ dutycycle = 20\,\%$.

6 Camera Model Implementation

This chapter presents how the modeling methodology presented in Ch. 4 is implemented. Chapter 5 presented several camera effects that depended on parameter files, VDS, and in some cases, ground truth data (like bounding boxes), which in most cases is directly provided by simulation software for a virtual test drive. In order to ensure a modular and flexible toolchain, in the following sections the usage of FMUs as enablers for co-simulation, and OSI as a standardized interface in simulation software, is illustrated.

6.1 Introduction

The main aim of camera model implementation is to establish a modular evaluation and development framework for ADAS/AD functions. Larger systems, which is the case of ADAS/AD function development and integration, usually rely on various models, simulation tools, and algorithms that make it challenging to study and investigate such heterogeneous systems [61]. Co-simulation, together with standardized interfaces, can be considered as a pragmatic approach in reducing the complexity of heterogeneous system simulation and integration, where it allows the exchange of various components on different integration test levels, e.g., on HIL or SIL level [18].

Through a modular co-simulation architecture, an iterative model development process may be achieved, thus further enabling the implementation of the updated V-Model presented in Fig. 1.8. Using a tool for model integration and co-simulation platform, virtual components and real components can be connected. Virtual components, i.e., the camera model and simulation software, can be provided as an executable library using FMI for co-simulation. Real components, as well, can be interfaced to the co-simulation platform, and the whole process may be executed via real-time operating systems whenever necessary.

Aside from integrating the camera model in an FMU and benefitting from various capabilities of the FMI standard, the camera model is also OSI enabled, i.e., wherever necessary the camera model may consume OSI messages like **GroundTruth** and **SensorView**, and provide **SensorData** to be used by convenient ADAS/AD functions (Fig. 2.5).

For a co-simulation to be successfully performed, the following steps are necessary:

1. **Authoring**:
   In the context of this dissertation, the model is programmed in `C/C++` and exported as an FMU using a dedicated framework provided by PMFS[1]. FMU-export is not limited to dedicated frameworks; many other tools like MATLAB/Simulink with AVL fmi.LAB or MapleSoft and others are also capable of performing FMU-exports.
2. **Model components' connection**:
   Basically refers to establishing the correct connections between various sub-models available in a co-simulation framework.
3. **Execution**:
   In the context of virtual simulation software, external model integration (like a camera model) may be directly integrated into the simulation software or incorporated via 3rd party software like Model.CONNECT[2]. Additionally, in the execution process, different configuration platforms and bitness (32bit/64bit) may be used, so it is vital to ensure that no compatibility issues arise due to the selected configuration.

In the following portion of the chapter, the camera model FMU is presented together with OSI integration; additionally, the model integration in simulation software featuring stand-alone co-simulation and a 3rd party tool is also illustrated.

[1](C) 2016 – 2017 PMSF IT Consulting Pierre R. Mai https://pmsf.eu. The Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0.

[2]Model.CONNECT™ is model integration and co-simulation platform, connecting virtual and real components, https://www.avl.com/de/-/model-connect.

6.2 Camera Model FMU

To ensure interchangeability between sensor models and other simulation software or ADAS/AD HAF, the developed camera model is integrated into a modular fashion based on standardized interfaces, like FMI, as well as based on specific interfaces, like OSI.

In Fig. 6.1, the integration process is illustrated, and it can be summarized in three main steps:

1. **Model Development**:
   The model development step lies entirely under the developer's responsibility, where the final model can be considered as a black box with dedicated access points/ports for model calibration, input, and output. In general, there should be no restrictions on the modeling/programming language or the model's abstraction levels.
2. **Standardized Interface**:
   Selecting a suitable interface is crucial for further integration and use of the developed model, so in general, a standardized interface is always preferable. Targeting a co-simulation framework, in this case the functional mock-up interface, is utilized. Additionally, integration feasibility between the selected interface and the generated model should always be assured. For example, if the selected interface does not support the programming language used for the model development, a suitable wrapper may always be used to ensure integration feasibility.
3. **Modular Integration**:
   The importance of this step lies in the fact that in large heterogeneous systems, it is rarely the case that one standardized interface can meet all simulation demands. A modular integration process enables the integration of standardized interfaces as well as other upcoming interfaces (like OSI and others) that should eventually lead to a harmonized co-simulation framework which satisfies all requirements. In this case, the following integration steps are considered:

   Integration Step 1:
   In this step, FMI is implemented by generating an FMU that mainly contains a description file and model executables. To enable the usage of other open source libraries like OpenCV, FMU's model equations

(defined as a set of C functions) are made accessible from a C++ based model.

Integration Step 2:
Depending on the number of additional interfaces needed for a simulation scenario, this step may be extended to include "n" other steps, where "n" is the number of necessary interfaces. For the current use case, only the OSI is implemented, and it can be seen in Fig.6.1 in the last section in orange.

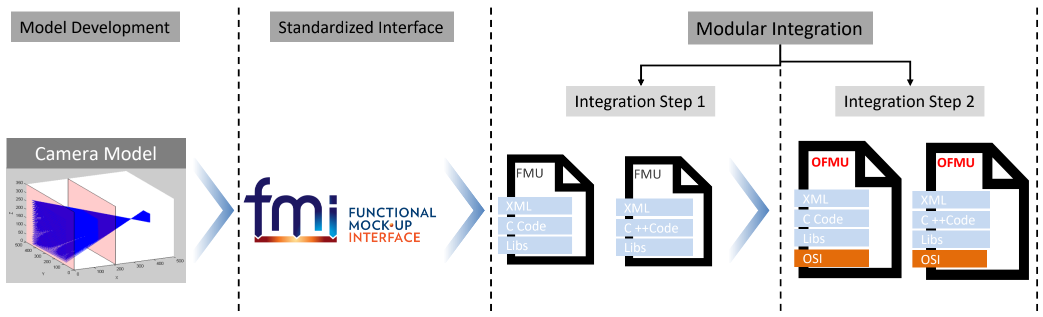


Figure 6.1: Camera model FMU integration.

To access the input/output data and status information during a co-simulation framework, the FMI standard defines a specific state machine for a functions-calling sequence. After a new FMU instance is successfully created (using `fmi2Instantiate()`), the following two main states follow:

1. **Initialization Mode**:
   In this state, the FMU is made ready for simulation by determining all outputs and other variables exposed by the exporting tool. In the `CameraModel.fmu` block diagram represented in Fig. 6.2, the **Model Parametrization** block receives its necessary inputs, and with that the function `CameraModel_doInit()` is called. The following code snippet illustrates how the vignetting model parameters are loaded and stored in OpenCV matrices.

```
/* GLOBAL VARIABLES */
vector<Mat> vigChannel; //stores all Vignetting parameters

/* FUNCTIONS */
int CameraModel_doInit(string filePath, string paramfileName){

//Vigneting parameterization

Mat placeHolder; //place holder matrix to perform necessary
    conversions
string sTemp;  //temporray file name for constructing the
    paramFile
string paramFile; //gets the parameter file name

   //three parameter files for Vignetting (for RGB)
   for (int i = 1; i < 4; i++) {
      paramFile = filePath + paramfileName;
      sTemp = to_string(i) + ".xml";
      paramFile += sTemp;

      //read the parameters and store them in matrix for each
          channel
      FileStorage fs(paramFile, FileStorage::READ);

      fs[paramfileName + to_string(i)] >> placeHolder;
      placeHolder.convertTo(placeHolder, CV_8U, 255);

      //populate the global variable vector with the
          corresponding vignetting parameters matrecies
      vigChannel.push_back(placeHolder);
      fs.release();
      }
   ....
   return 0;
}
```

2. **Slave Initialized**:
   In this state, the slave is initialized, and the co-simulation computation is performed. The most important function call in this state is the `fmi2DoStep`; each time this function is called, the `CameraModel_doCalc()` function automatically runs all necessary augmentations on image- and object-base level. For example, the following code snippet illustrates the vignetting effect implementation represented in Fig. 6.2 under **Optical Effects**.

```
int CameraModel_doCalc(Mat IdealImage){

   Mat imgIn; //input matrix
   Mat imgOut; //output matrix

   imgIn = IdealImage;

   //three matrecies for vignetting implementation, one for
       each channel
   Mat vignetingImage_1, vignetingImage_2, vignetingImage_3;
   vignetingImage_1 = Mat::zeros(imgIn.rows, imgIn.cols,
       vigChannel[0].type());
   vignetingImage_2 = Mat::zeros(imgIn.rows, imgIn.cols,
       vigChannel[1].type());
   vignetingImage_3 = Mat::zeros(imgIn.rows, imgIn.cols,
       vigChannel[2].type());

   Mat bgr[3];
   split(imgIn, bgr);

   // vignetting implementation
   for (int col = 0; col < imgIn.cols; col++) {
      for (int row = 0; row < imgIn.rows; row++) {
         vignetingImage_1.at<uchar>(row, col) =
             bgr[0].at<uchar>(row, col) *
             vigChannel[0].at<uchar>(row, col) / 255;
         vignetingImage_2.at<uchar>(row, col) =
             bgr[1].at<uchar>(row, col) *
             vigChannel[1].at<uchar>(row, col) / 255;
         vignetingImage_3.at<uchar>(row, col) =
             bgr[2].at<uchar>(row, col) *
             vigChannel[2].at<uchar>(row, col) / 255;
      }
   }
   vector<Mat> vigChannels{ vignetingImage_1,
       vignetingImage_2,vignetingImage_3 };

   Mat vignetingImage; //image with vignetting effect
   merge(vigChannels, vignetingImage);
.....
return 0;
}
```

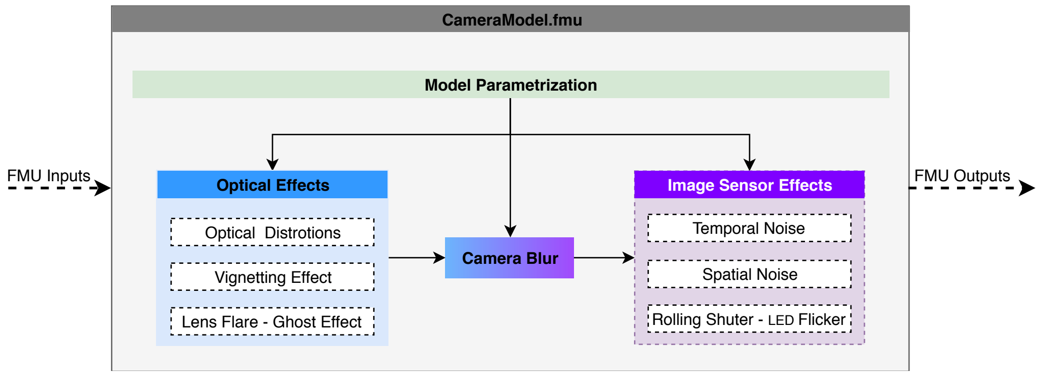


Figure 6.2: Camera model FMU.

In a similar fashion, **Image Sensor Effects** and the rest of the **Optical Effects** can be parameterized and implemented. The camera model's outputs, represented as `FMU Outputs`, rely entirely on the simulation framework; they may represent an augmented VDS (transmitted through a communication interface or simply by creating a local copy of the augmented VDS), OSI buffer, or even an object list in any other format.

6.3 Simulation Toolchains and Model Integration

Due to numerous possible applications of sensor models and different integration/simulation capabilities of simulation software that are currently on the market, this section provides a set of toolchains that covers the model integration approach for different use cases.

For the camera model FMU represented in section 6.2, the following use cases are considered:

1. **Co-simulation platform**:
   In Fig. 6.3, it is shown that by using a co-simulation platform like Model.Connect, the `CameraModel.fmu` can be integrated and coupled with various components. Starting from the very left-hand side of Fig. 6.3, the **Linux/Windows-Host** enables the integration of simulation software that is either running on a Linux or Windows operating system. For both the **Model Parameters** and **HAF**, the gradient color indicates that they may either exist on the same host machine, like the

Environment Simulation, or directly on the machine hosting the **Co-Simulation Platform**. Usually, **Model Parameters** are locally stored files that are in specific formats (`like XML, JSON`) and don't need to be executed but merely accessed, which is done by correctly configuring the `CameraModel.fmu`. In such a toolchain, the **HAF** may be integrated as a standalone executable or directly as an FMU (only in the co-simulation platform). In this toolchain, the **Co-Simulation Platform** is the one providing the master clock and managing the synchronization between all sub-components when executed in different threads.

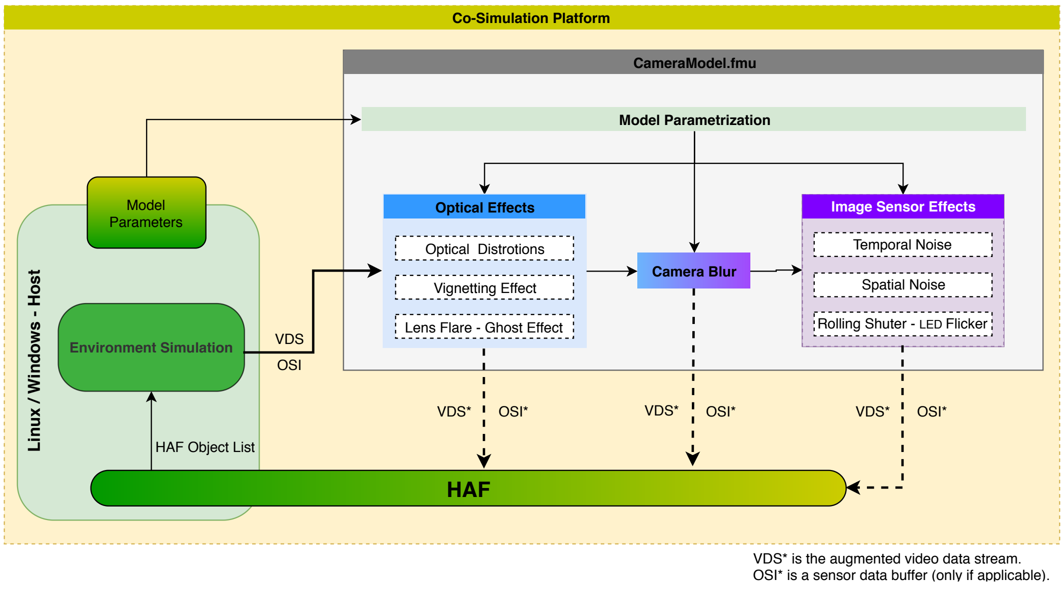


Figure 6.3: Toolchain for camera model integration in a co-simulation platform.

2. **Integration in simulation software**:
Several simulation software supports most functionalities of FMI interfaces and are capable of dynamically linking an FMU to its running application. In Fig. 6.4, integrating an FMU in such a toolchain is usually done over an FMU-plug-in dialog, which enables importing and configuring FMUs. For the `CameraModel.fmu`, the parametrization process is still done from the **Model Parameters** block; however, other inputs like VDS and OSI data is directly provided by the simulation software. Currently, neither FMI nor OSI supports a direct transfer of image data; consequently, the VDS can be transferred over any other communication protocol (like Transmission Control Protocol (TCP), User Datagram Protocol (UDP)) or by direct shared memory

access. Similar to the **Co-simulation platform** setup, **HAF** may be directly integrated into the **Simulation Software** or as an external standalone application. In such a toolchain, the **Simulation Software** is the master.

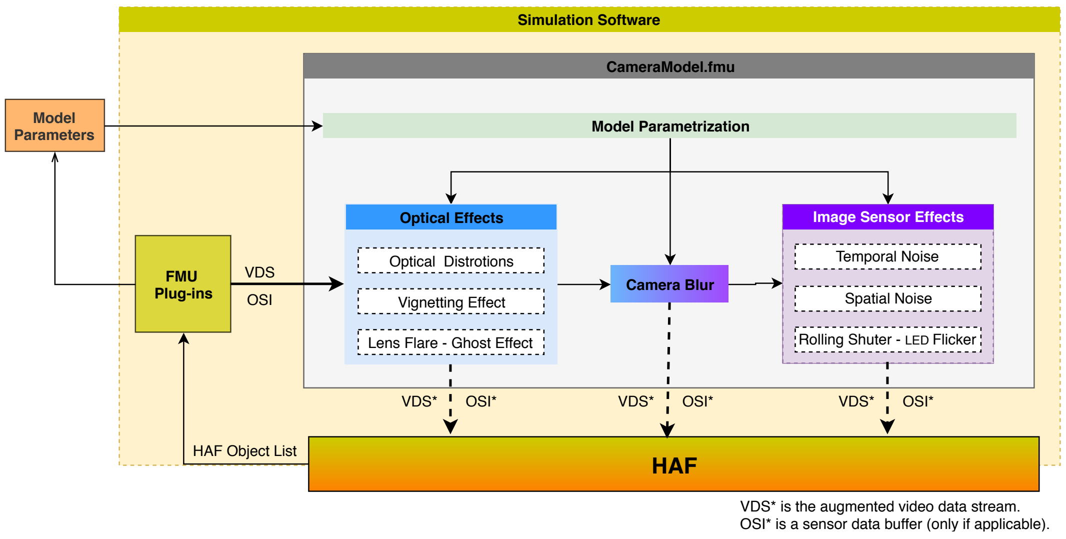


Figure 6.4: Toolchain for camera model integration with simulation software.

3. **Offline Integration**:
   Certain use cases necessitate offline usage of sensor models, more precisely of data generated by sensor models. For example, in the case of NN training and inferencing, the `CameraModel.fmu` outputs may be used. Moreover, offline integration may also be used for sensor model evaluation, where its output may be directly compared with that of real sensors in a post-processing step. In Fig. 6.5, the `CameraModel.fmu*` is exported as a standalone simulator by dedicated software like PMFS. The FMU still needs to be configured for parametrization and for accessing the VDS and OSI buffers (if applicable). In such a toolchain, the exported simulator is the master, and thus the simulation time, step size, and other parameters are directly controlled by it. Coupling the simulator with other **HAF** can be either attained with dedicated scripts (in this case the script may control the simulation time and the step size) or directly triggered by the `CameraModel.fmu*`.

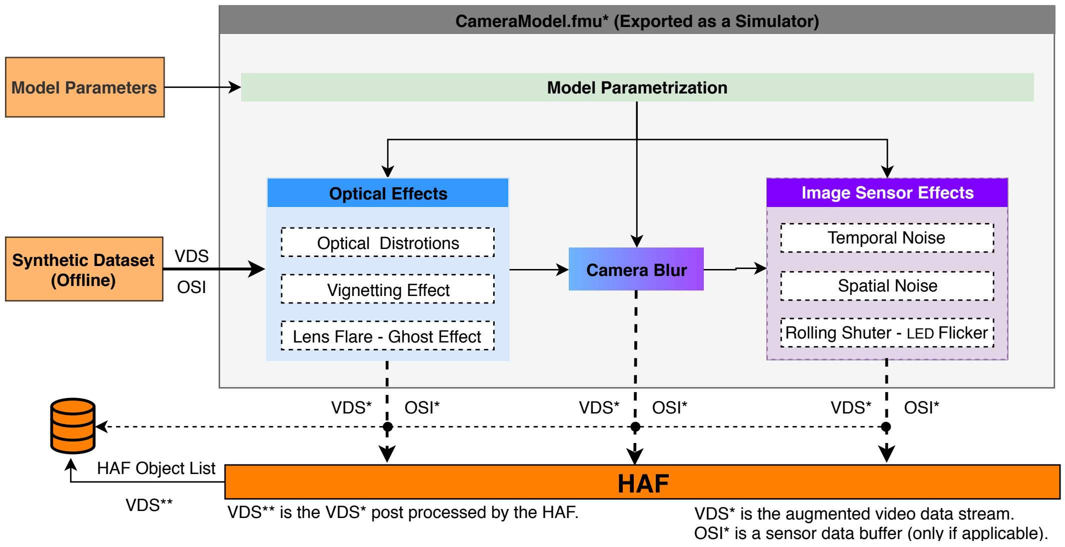

CameraModel.fmu* (Exported as a Simulator)
Model Parameters
Model Parametrization
Optical Effects
Optical Distrotions
Vignetting Effect
Lens Flare - Ghost Effect
Camera Blur
Image Sensor Effects
Temporal Noise
Spatial Noise
Rolling Shuter - LED Flicker
Synthetic Dataset (Offline)
VDS
OSI
VDS*
OSI*
HAF Object List
VDS**
HAF
VDS** is the VDS* post processed by the HAF.
VDS* is the augmented video data stream.
OSI* is a sensor data buffer (only if applicable).

Figure 6.5: Camera model integration as a simulator.

6.4 Evaluation of the Modeling Process

In this chapter, the camera model calibration and implementation processes represented in Ch. 5 and Ch. 6 are put to the test. The process is implemented on two development kit cameras equipped with automotive image sensors and one automotive MFC that is already available in series production.

6.4.1 Camera Calibration Setup

To demonstrate the repeatability of the calibration and implementation processes on different cameras, this section represents the preparation and initialization setup of the CUM. For confidentiality reasons, only the process on the development kit cameras is shown, i.e., for two cameras.

Two Demo3 kits, consisting of the same sensor headboard and USB 3.0 interface board, are selected. In Fig. 6.6, both cameras are illustrated. As the aim is to prove the repeatability of the calibration and implementation processes, both cameras have the same manufacturing release and revision number, thus diminishing the possible effect that different HW components may have on the model's output. Moreover, both cameras' acquisition setup and internal configuration parameters, like exposure time and Bayer [3] pattern, are set exactly the same.

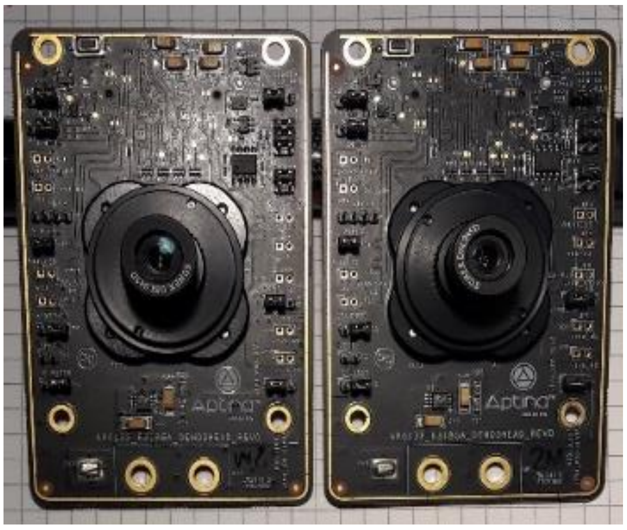


Figure 6.6: Demo Kit3 cameras, sensor and interface board.

As previously mentioned, current automotive cameras have a fixed focusing distance; however, since the focusing distance for the current development kit is adjustable (by screwing/unscrewing the lens), it is essential to ensure that the focusing distance for both cameras is exactly the same before starting the modeling process. This is achieved with the implementation of a defocus detection algorithm that is based on the second image derivative, i.e., by calculating the Laplacian of the image. The method is based on the following paper [62].

The defocus detection method examines the gradient of the image, so for this method to work, a target object that has some texture in it is chosen. In other words, images with pure color are not suitable for this method. The Laplacian highlights regions of an image containing rapid intensity changes. As the first derivative of an image in either x, y, or diagonal-direction represents the rate of pixel intensity shift with respect to neighboring pixels, the 2^{nd} derivative would pass high frequencies that directly correspond to sharp edges. By computing the variance of the obtained Laplacian image

[3] A Bayer filter mosaic is a color filter array (CFA) for arranging RGB color filters on a square grid of photosensors.

in x and y-direction, the spread between pixels that correspond to sharp edges, and others that don't, can be estimated. As a consequence, 2^{nd} image derivatives with higher variance value would directly correspond to a larger gap between pixels that have high frequencies and those that don't, i.e., more focused edges. This concept is applied in order to adjust the focusing distance for both cameras until similar variance value are obtained for images taken from both cameras. For example, Fig. 6.7 (a) represents the 2^{nd} image derivate of a defocused target object (Fig. 6.7 (a)), whereas a correctly focused image is represented in Fig. 6.8.

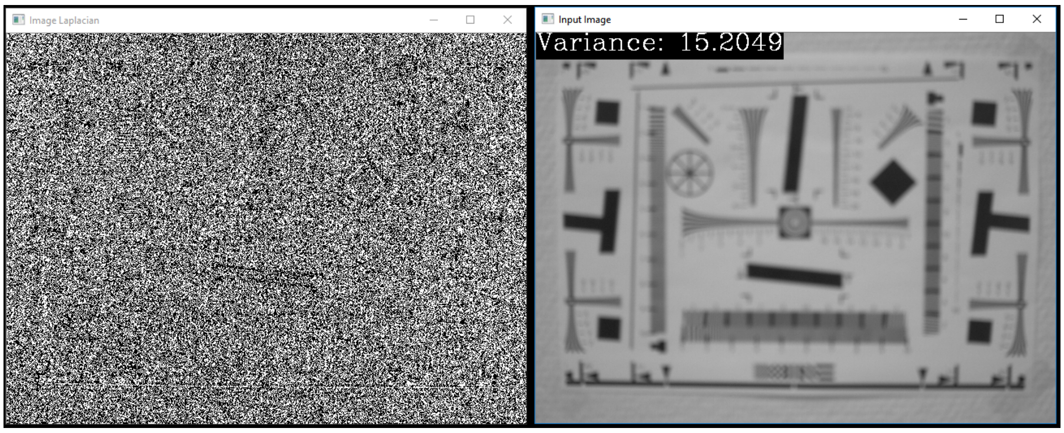


Figure 6.7: (a) 2^{nd} image derivative of a defocused image, (b) Defocused input image.

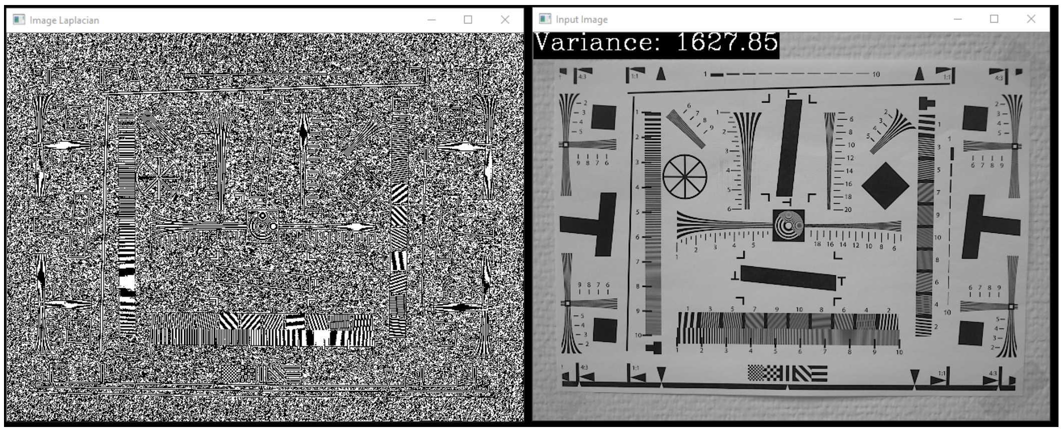
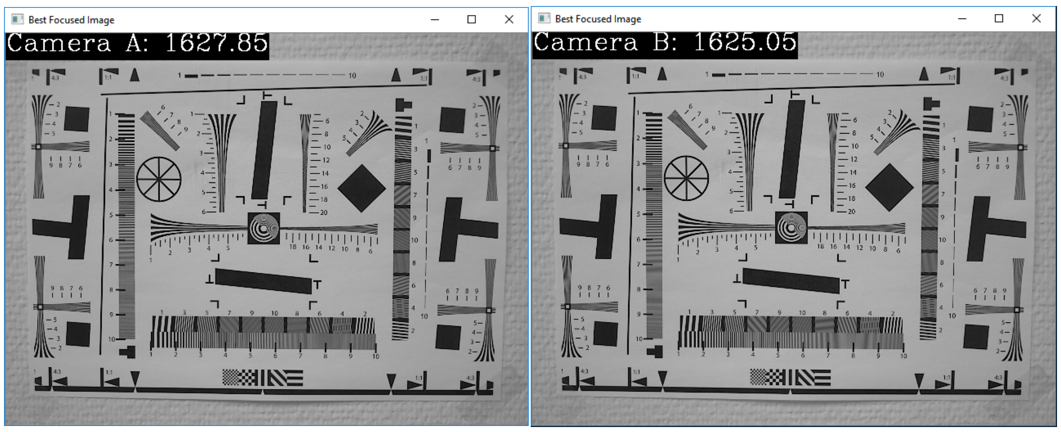


Figure 6.8: (a) 2^{nd} image derivative of a focused image, (b) Focused input image.

As represented in Fig. 6.9, the method is proven to provide consistently reliable results under the same image acquisition conditions. After reaching a low variance value difference ($1627.85 - 1625.05 = 2.8$ in this case), cameras ″A″ and ″B″ can be considered to have the same focusing distance, and hence the modeling process can start as described in Ch. 5 and Ch. 6.



(a) (b)

Figure 6.9: Variance values of the 2^{nd} image derivatives and best focused images for (a) Camera A, (b) Camera B.

6.4.2 Results

Due to time constraints and to the availability period of both cameras, the modeling process was only tested on distortion and vignetting effects.

1. **Distortion Effect**:
   For the calibration of the distortion effect, the setup represented in Fig. 5.2 together with the testbed prototype represented in Fig. 5.1 are used. Table 6.1 illustrates the obtained distortion parameters results for cameras A and B, where ″Ideal″ represents the data sheet parameter values, and Δ_{ideal} represents the difference between ″Ideal″ values and other parameter values obtained for corresponding camera. Similarly, $\Delta_{(A-B)}$ represents the corresponding parameter differences between cameras A and B.

Table 6.1: Calibration results for distortion parameters.

	Focal Length [pxl]		Principal Point [pxl]	
	f_x	f_y	c_x	c_y
Ideal	1466	1466	640	480
Camera A	1484, $\Delta_{ideal} = 18$	1485, $\Delta_{ideal} = 19$	655, $\Delta_{ideal} = 15$	505,$\Delta_{ideal} = 25$
Camera B	1480, $\Delta_{ideal} = 14$	1480,$\Delta_{ideal} = 14$	619, $\Delta_{ideal} = 21$	518, $\Delta_{ideal} = 38$
$\Delta_{(A-B)}$	4	5	36	13

In table 6.1, it can be seen that Δ_{ideal} for both cameras is greater than zero, which is expected. However, by further examining $\Delta_{(A-B)}$ for the focal lengths, the difference value is not zero, but relatively small (less than Δ_{ideal}); in contrast Δ_{A-B} for the principal points cannot still be considered small especially for c_x, where $\Delta_{(A-B)} = 36$ exceeds both Δ_{ideal} values. These observations mainly illustrate that the focal length calibration values may be generalized and used for other cameras but not the principal point parameter values. A shift in the c_x value means that the optical center of the lens system is not perfectly aligned with the center of the image sensor in the x-direction, and may be the result of various factors (please refer to section 5.4). Generalizing the principal points for the distortion model may result in a wrong representation/projection of a 3D object into the image space.

2. **Vignetting Effect**:
   For the parameters' calibration of the vignetting effect, the setup represented in Fig. 5.9, together with the testbed represented in Fig. 5.1, are used. Contrary to the distortion model, Fig. 6.10 shows a negligible difference between pixel intensities obtained from the calibration of cameras A and B. This indicates that the vignetting model **may** be generalized for cameras with the same revision release (same optical module was used).

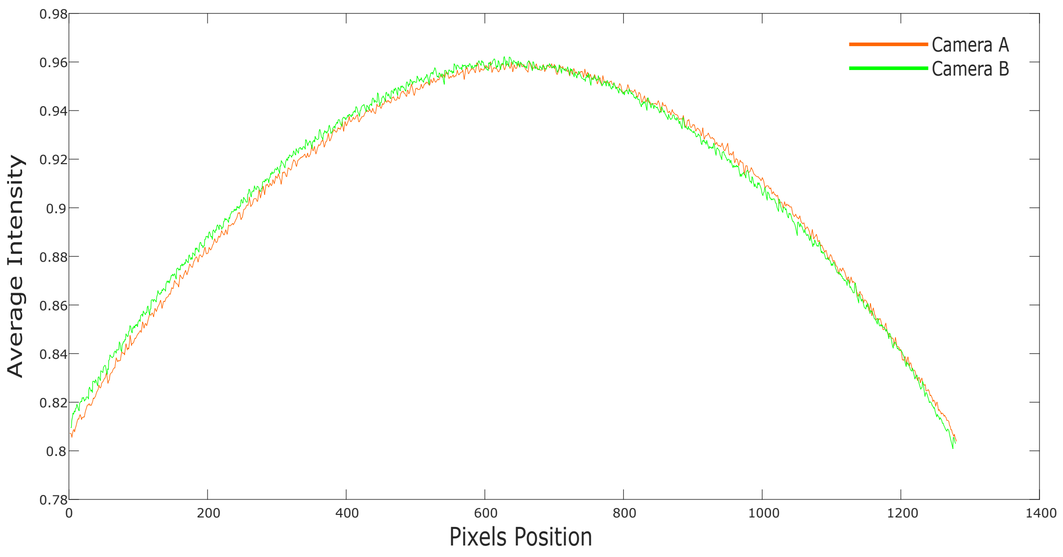


Figure 6.10: Averaged pixel intensity of the vignetting effect for cameras A and B.

To conclude this chapter, a solid conclusion may not be directly drawn regarding what effects may be generalized and benefit from only one calibration process. To reach such a conclusion, more than two cameras should be taken into account (something that is subject to future work) and driven through the entire calibration and implementation processes. The required number of samples necessary to make a valid inference can be determined either by setting a target variance or a confidence level; both of which are highly dependent on previous experience. Despite the mentioned limitations, this chapter still indicates the need for an investigation of separate effects, and paves the way for future strategies related to physical camera model calibration. As a final note, differences that may be obtained throughout the calibration process should not immediately be a cause for alarm, because in the end, these may still lie inside the tolerance and abstraction levels needed for a specific use case. Therefore, George Box's quote proves pertinent yet [7]:

"All models are wrong, but some are useful."

7 Camera Model Applications

7.1 Introduction

This chapter presents several possible applications of the camera model in ADAS/AD function development and virtual testing. The following sections handle different image-based algorithms, like lane and vehicle detection. The developed algorithms cover approaches that rely entirely on classical CV methods as well as others that rely on NN approaches.

7.2 Virtual Testing Setup for Lane Detection Algorithm

7.2.1 Scope

The scope is to demonstrate the usage of the first toolchain represented in Ch. 6, Fig. 6.3. The ability to couple different simulation software running on different operating systems (Linux and Windows), and the integration of an LD algorithm in the simulation toolchain, are illustrated. Additionally, the flexibility in including or excluding any camera effect through the `CameraModel.fmu` configuration is also demonstrated. It is essential to point out that the scope here is not to validate the camera sensor model but to demonstrate that image data generated from the camera model influence the behavior of ADAS/AD functions.

7.2.2 Method Description

For this demonstration, an LD algorithm is created and integrated into a separate FMU: `LaneDetecion.fmu`. The LD pipeline consists of seven main steps:

1. **Threshold & Convert to Binary**:
   In this step a binary image is created first by transforming the input image (7.1 (a)) from an RGB color space into an Hue, Saturation, Lightness (HSL) color space, and applying a threshold operation per channel; second by extracting and accentuating the edges via a *Sobel* operator. The final, obtained binarry image is the result of two logical operations (&, ||) performed on the previously obtained images. The result of this step is represented in Fig. 7.1 (b).

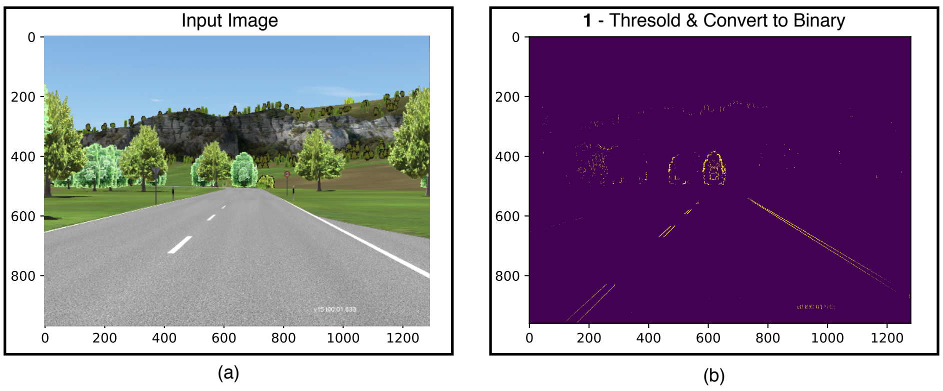


Figure 7.1: LD input and threshold image.

2. **Extract Region of Interest**:
   After the binary image is obtained, a specific ROI (Fig.7.2 (a)) is extracted based on the mounting position and orientation of the camera inside of the vehicle. The camera's mounting position is provided as an FMU input from the `CameraModel.fmu`.
3. **Obtain Warped Image**:
   This step aims to perform a perspective transformation, which maps the points extracted from the ROI image into a bird's-eye view transformation; this is necessary for calculating the lane curvature in later steps. Fig. 7.2 (b) represents the obtained result.

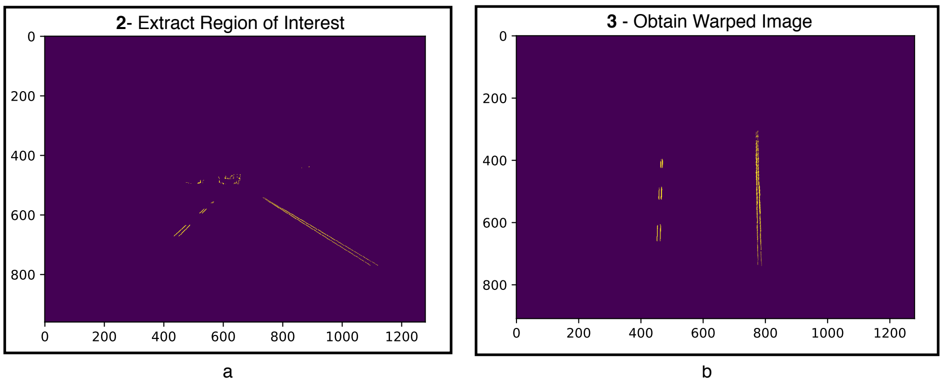


Figure 7.2: ROI implementation and bird's-eye view representation of the cropped image.

4. **Estimate Lane Position**:
   In order to reduce the search region for candidate lanes inside the bird's-eye view image, the histogram along the columns is calculated in this step; then the position of both peaks represented in Fig. 7.3 (a) are used as starting points when it comes to searching for the lines.
5. **Execute Sliding Window**:
   With the points obtained from the previous step, the sliding windows are placed around the line centers, and moved upward step by step following the lines up to the top of the frame as represented in Fig. 7.3 (b). At each window position, active pixels, i.e., non-zero pixels, are identified and saved to be used later on for line estimation.

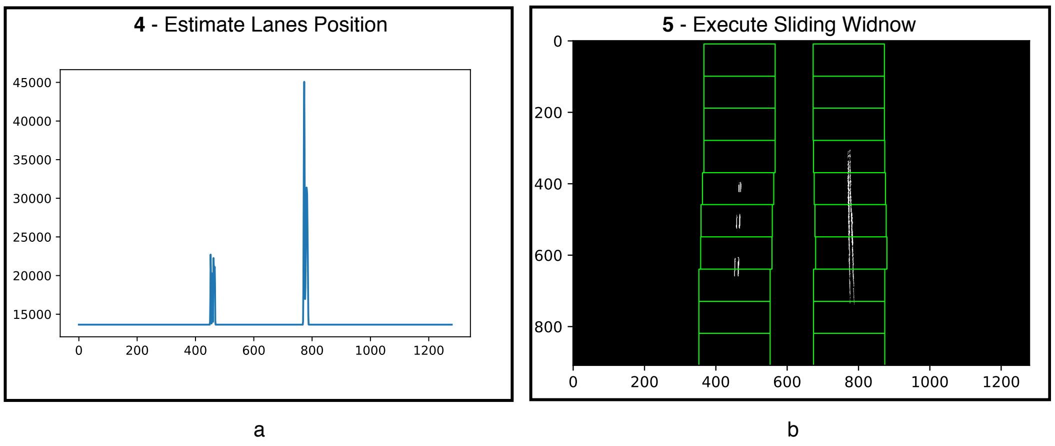


Figure 7.3: a. Column histogram, b. Sliding window representation.

6. **Compute Vehicle Position**:
   After obtaining all pixels belonging to each line, in this step a 2nd degree polynomial is fitted to each line:

$$f(y) = ay^2 + by + c \tag{7.1}$$

   With the obtained equation, the radius of curvature is then calculated using the following equation:

$$R_{curve} = \frac{\left(1 + (2ay + b)^2\right)^{\frac{3}{2}}}{|2a|} \tag{7.2}$$

   Where:

   $a\,,b\,,c$ are the polynomial coefficients.
   y is the position of the current detected point (in the vertical direction).

   Up to now, all calculations are done in pixels, so the last step before displaying the results is to convert them into world space, i.e., to meters. This can be achieved by knowing how many meters per pixel exist in our perspective transformation. Estimating such values can be derived from various camera-related parameters; however, for this demonstration, the horizontal distance between two lines is set to a minimum value of 3.5 m, and the length of the lane considered in the perspective transformation is set to approximately 25 m (verified in the simulation software). Additionally, the vehicle's offset from the middle lane is calculated as the difference between the center point of the detected lanes and the image center (considering the camera is mounted in the middle of the vehicle with no yaw or roll angle).
7. **Recast Image for Display**:
   In the last step (and only for visualization purposes), the bird's-eye view image is transformed back into the original image shape and overlaid, together with additional text information, on the input image. The obtained results are represented in Fig. 7.4; the large curvature indicates that the current detected lanes are straight, whereas a positive center offset indicates a shift in the vehicle's position to the left of the lane center. Likewise, a negative offset would indicate a shift to the right of the lane center.

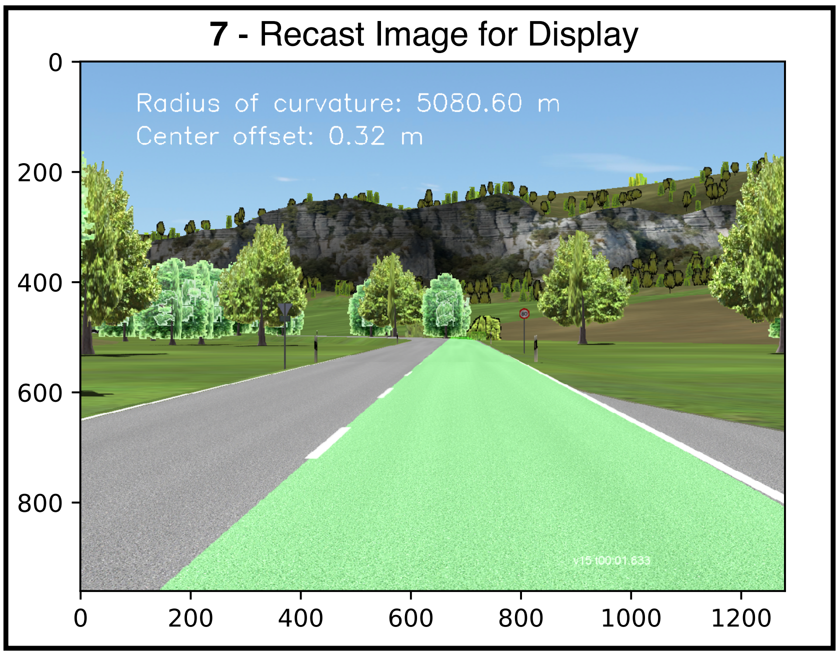


Figure 7.4: LD output with computed lane curvature and center offset.

After successfully generating the `LaneDetection.fmu`, the aim is to integrate it with the `CameraModel.fmu` and simulation software in a co-simulation platform. Due to current limitations of the developed LD algorithm (currently, it cannot handle sharp turns), only high-way scenarios with a minimum radius curvature of 800 m are considered. With the usecase ready, two different simulation software, i.e., CarMaker and VIRES VTD, are coupled to the FMUs, and the algorithm's outputs are saved for further analysis.

7.2.3 Experimental Setup

For the camera model and LD use case evaluation, the experimental setup presented in Fig. 7.5 is used. In Fig. 7.5, the VDS exchange between the simulation environment and the camera model is done via TCP communication. Depending on the host IP address and the port number, the camera model can be configured, through FMU parameters, to receive the VDS either from VTD (Linux-Host) or CarMaker (Windows-Host); both setups run separately, i.e., with either VTD or CarMaker. Through the camera model's FMU parameters, the path for the parametrization file is also set, and the desired effects for implementation are selected, that is, one test run can be configured to include camera blur and vignetting, the other may be set to consider only camera distortions. For the current test scenario, the camera model is configured to include the distortion, blur, and vignetting effects.

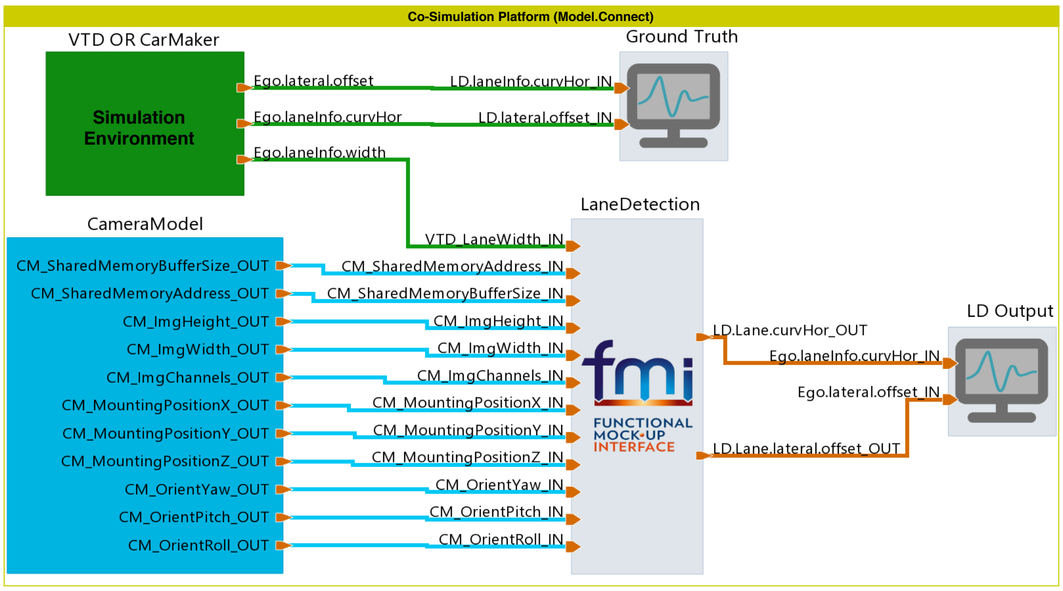


Figure 7.5: Co-Simulation platform for LD and camera model integration.

As described in section 7.2.2, the LD pipeline starts with an input image. In the current toolchain, this image is provided by the camera model. For computation and data transfer efficiency, the camera model writes its output on a global shared memory and provides the memory address (`CM_SharedMemoryAddress_OUT`) and (`CM_SharedMemoryBufferSize_OUT`), the buffer size, as FMU outputs to the LD in order to access the image data. Furthermore, the camera model provides other information, like camera mounting position and orientation, to the LD FMU. By default, the camera model is always writing the non-augmented and augmented images to the shared memory (this behavior is configurable). For the LD algorithm to be aware of where an image starts and ends, the camera model also provides image information, like height width and the number of channels, as standard FMU outputs. The final two blocks in Fig. 7.5, i.e., "Ground Truth" and "LD Output," are two data loggers used for offline data analyses.

7.2.4 Results

For the evaluation process, the vehicle's lateral offset from the center of the driving lane, calculated by the LD algorithm, is compared with the ground truth data obtained from the simulation environment. Using Eq. (7.3), the precision of the calculated offset is obtained for each image frame.

$$Precision_{offset} = 100 - \frac{|GT_{offset} - LD_{offset}|}{GT_{offset}} \tag{7.3}$$

Where:

$Precision_{offset}$ is the precision of the calculated offset in [%].
GT_{offset} is the ground truth vehicle offset from the driving lane's center [m].
LD_{offset} is the center lane offset calculated by the LD algorithm [m].

For the current test run, the LD algorithm running on non-augmented VDS recorded an average precision of 97.45% and 88.96% for the LD algorithm running on the camera model output indicating that the augmentation method is negatively affecting the performance of the LD algorithm. In Fig. 7.6, the calculated offset precision is represented for both the synthetic VTD and the camera model augmented image data. The latter shows a lower precision value, indicating that the LD algorithm operating on augmented image data faces some difficulties in correctly detecting the lane markings. The observed results may reflect the fact that some LD algorithm parameters need to be tuned to better cope with real scenarios; however, this can only be confirmed by bringing a real test scenario to simulation; this, at the time and due to several logistical reasons, proved to be an unfeasible process to perform.

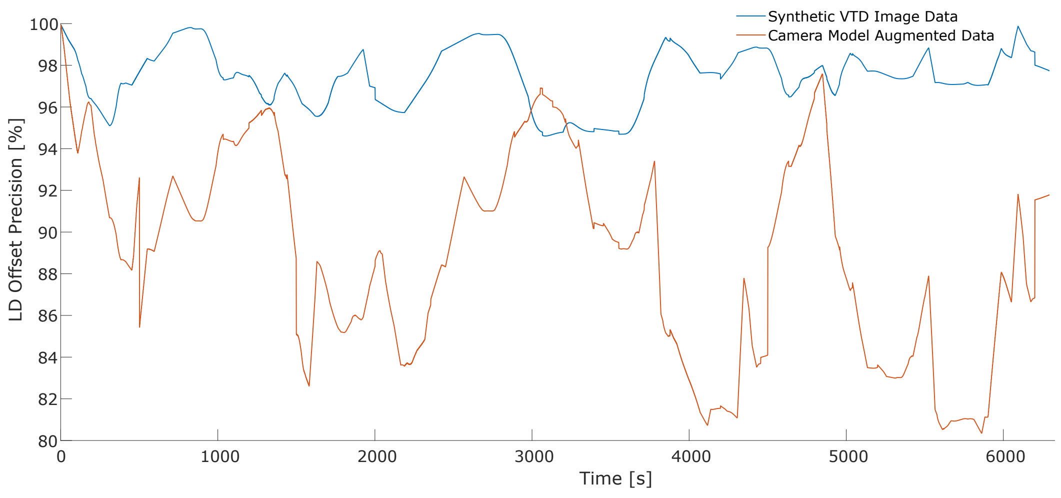


Figure 7.6: Relative precision for the calculated vehicle offset from the lane center.

Similar results (but with slightly different ranges) were also observed when the process was running with image data generated from CarMaker. For a more modular toolchain, all separate ground truth and sensor-related signals can be replaced by OSI messages. Throughout this dissertation, OSI buffer messages were used and implemented but not presented in this toolchain since, currently, the used simulation environments do not officially support OSI.

7.3 Virtual Testing for Vehicle Detection

7.3.1 Scope

In the automotive industry, and especially in ADAS/AD domain, functions like VD undergo a costly and time-consuming process for development and validation before their deployment in a vehicle. In many cases, the testing and validation process of image-based VD algorithms highly rely on previously recorded data, which in some cases is challenging to obtain, especially for some edge use cases that often rely on uncontrollable factors, like weather conditions.

In the following section, the toolchain presented in Fig. 6.4 (Ch. 6) is put into practice. The main aim of this process is to investigate the possibility of testing image-based functions (VD in this case) on virtually generated image data.

7.3.2 Method Description

For the current use case, and in order to illustrate the usage of the developed camera model in ADAS/AD function development and testing, a relatively simple VD algorithm is developed and put to the test using different setup versions of the toolchain presented in Ch. 6, Fig. 6.4.

Fig. 7.7 represents the main block diagram of the VD algorithm. The process consists of two main parts:

1. **Training**:
   This process is done offline and normally used for obtaining a vehicle classifier. A combined dataset of images labeled as vehicle or non-vehicle from Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago (KITTI) [48], BDD100K [63], and **Cars** dataset [64] is created as the main input for the feature extraction process, which relies on color and a Histogram of Oriented Gradients (HOG). The justification behind considering color as a feature for the training process is that virtually generated images fail to reproduce accurate color representations when compared to real recorded scenarios, a gap which may be diminished by the camera model. In the next step, the extracted features are combined and used to train a linear Support Vector Classifier (SVC).
2. **Processing (VD)**:
   The second part illustrates the VD processing pipeline. Starting from an input image, a sliding window approach is implemented and set to start approximately on the lower half of the image with different window sizes and overlap. Cropped images are then obtained and passed through the feature extractor, where color and gradient-based features are obtained. Using the previously trained VD algorithm, the overlapping tiles from each frame are then classified as ″vehicle″ or ″non-vehicle.″ Since several overlapping windows may be classified as vehicles or non-vehicles, and in some cases, they may partially overlap, a heat map to identify vehicle classifications in the same or near

the same locations from several subsequent frames is implemented. As a final step, bounding boxes around high-confidence detections (extracted from the heat map) are generated and overlapped onto the input image.

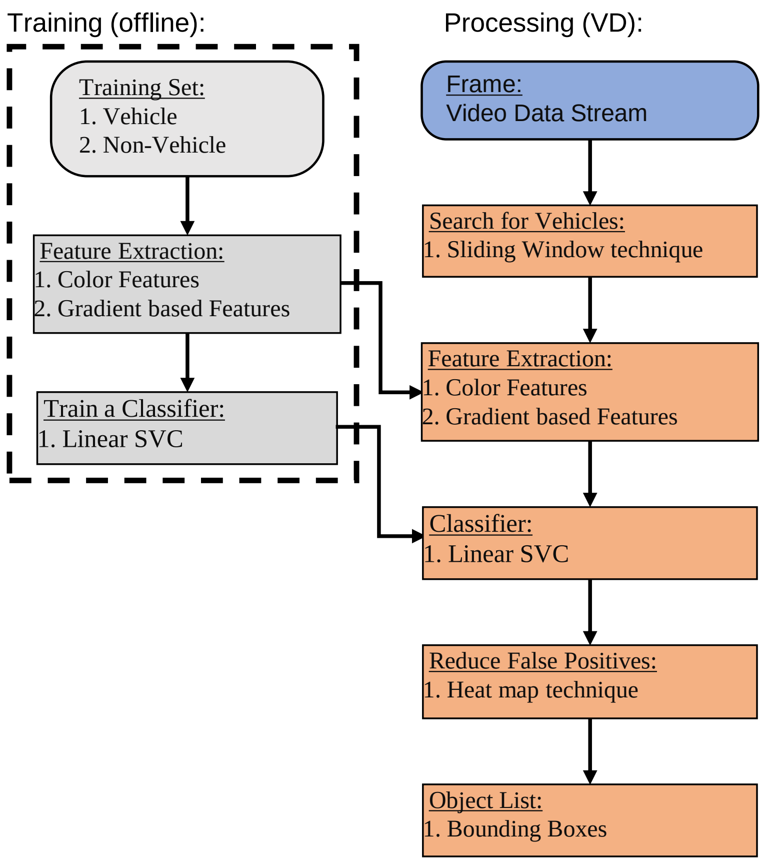


Figure 7.7: VD training and processing pipeline represented as a block diagram.

For a meaningful comparison of the object list generated by the VD algorithm when running on real, synthetic, and camera model VDS, it is imperative to construct virtual driving scenarios that emulate real driving scenarios as accurately as possible. In Fig. 7.8, the green dashed box roughly illustrates the process for obtaining the necessary ground truth data and camera sensor information, like mounting position, orientation, and resolution from the real test run scenario. The obtained information is necessary to generate a virtual scenario that is as close as possible to the real one. To generate the virtual scenario, CarMaker is used as a simulation software and, with the help of **IPG Movie**[1], a synthetic non augmented VDS is obtained (represented as VDS* in Fig. 7.8). The VDS* is then fed to the camera model, which is directly integrated into the simulation software (as the toolchain in Fig. 6.4 suggests) and, as a result, the augmented VDS (VDS**) is obtained and fed to the same VD algorithm (Vehicle Detection Algorithm*). Depending on the selected camera effects that are configured for simulation, VDS** may represent an image that is augmented by one camera effect, or a combination of them; a complete list of possible effects to be implemented is presented in Ch. 5 under section 5.2 or 5.4.

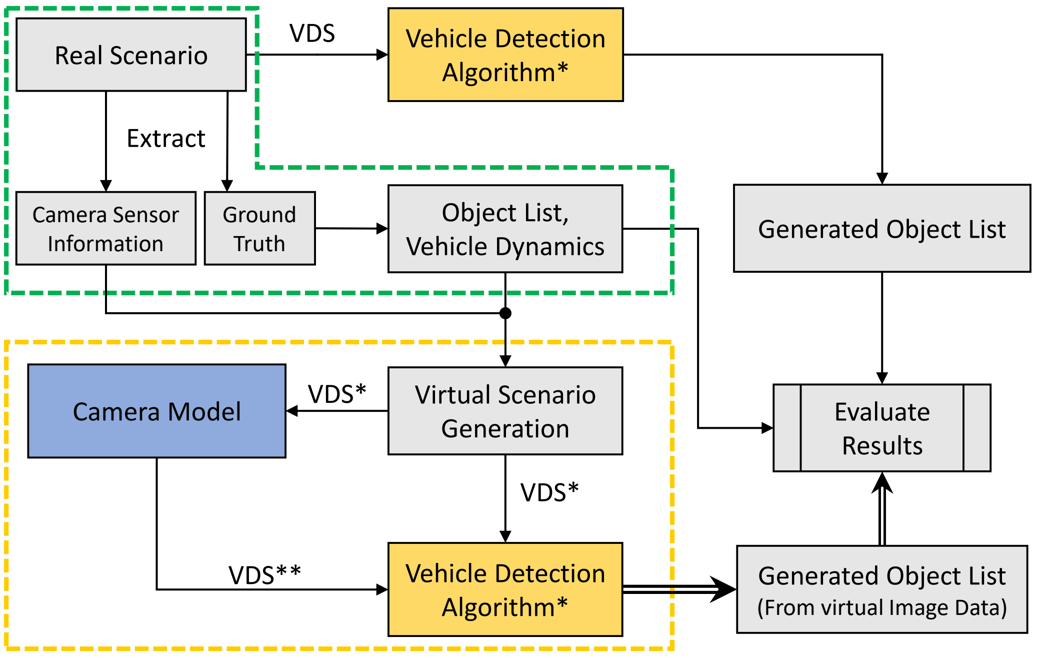

Real Scenario
VDS
Vehicle Detection Algorithm*
Extract
Camera Sensor Information
Ground Truth
Object List, Vehicle Dynamics
Generated Object List
Camera Model
VDS*
Virtual Scenario Generation
Evaluate Results
VDS*
VDS**
Vehicle Detection Algorithm*
Generated Object List (From virtual Image Data)

Figure 7.8: Real to virtual scenario process and simulation toolchain representation.

7.3.3 Experimental Setup

Transferring an entire real scenario into a virtual software is a highly demanding task that necessitates specific measurement equipment and extended planning beforehand. For the purpose of this demonstration, a test vehicle is equipped with a dedicated measurement system to log its position at all times and a camera, of which its mounting position, orientation, and

[1]IPGMovie is a visualization tool that can be used as a camera raw signal interface.

resolution are known. One target vehicle is used, the ADAC target vehicle [65], and its relative position with respect to the ego vehicle is provided at all times. With the recorded ground truth data, the results of replicating the real scenario into a simulation software are represented in Fig. 7.9. The first image on the left is a visual representation of the ego vehicle's track; `Dist_Target_VUT` represents the relative distance between the test target and the front end of the ego vehicle (current position indicated by the green cross on the bottom of the image). The ADAC test target is represented in the middle figure; the reason behind using an ADAC test target instead of a real vehicle is that these measurements were done through an emergency backing assist function test. On the right side of Fig. 7.9, the virtually generated track and a close enough representation of the ADAC test vehicle are shown.

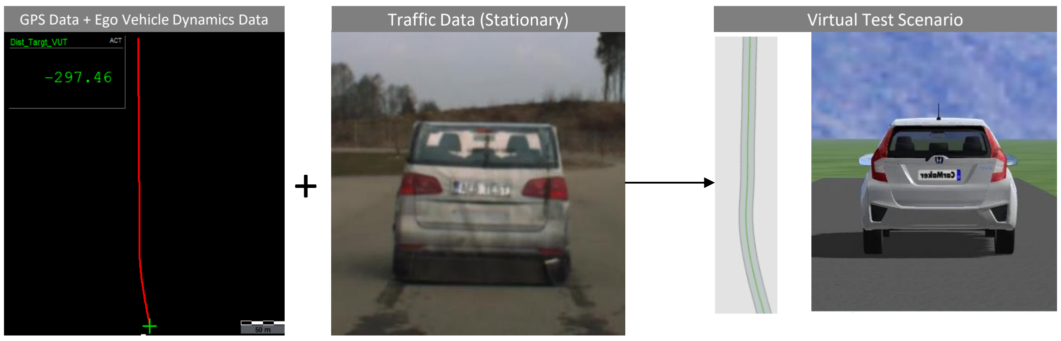

GPS Data + Ego Vehicle Dynamics Data
Dist_Targt_VUT
ACT
-297.46
50 m
+
Traffic Data (Stationary)
Virtual Test Scenario

Figure 7.9: Real and virtual scenario representation.

After creating the virtual test run, the toolchain presented in Fig. 7.8 is adapted to the representation in Fig. 7.10. This means that, for the current test run, both real and virtual scenarios are identical from vehicle dynamics and ground truth object list point of view; from a visualization point of view, i.e., image data, the scenarios are brought to resemble each other as much as possible.

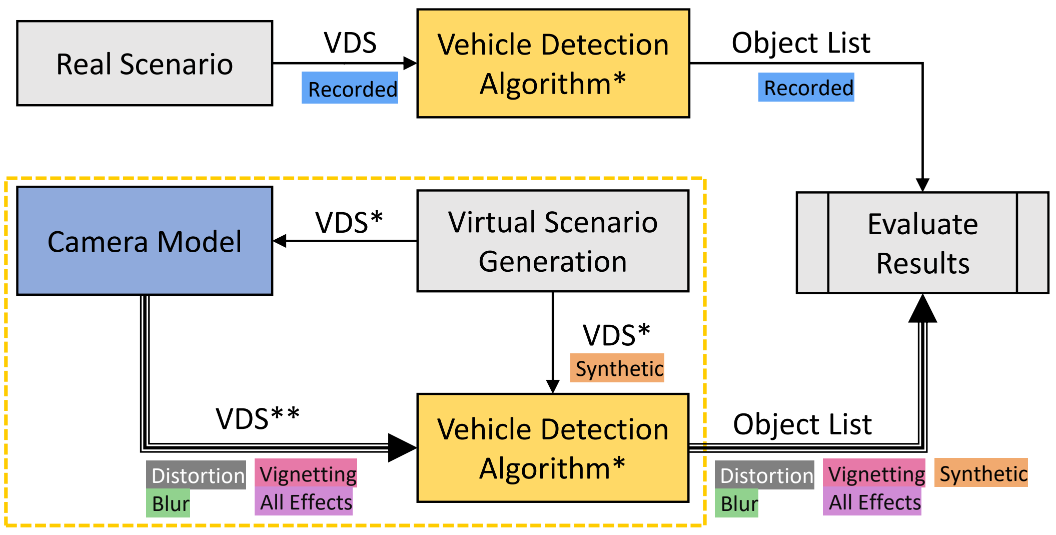


Figure 7.10: Experimental setup for VD testing with synthetic and camera model image data.

For the interpretation of Fig. 7.10 the color codes and notations signifies the following:

- VDS associated with "Recorded" is highlighted in blue, and refers to image data from the real camera under modeling.
- VDS* associated with "Synthetic" is highlighted in orange, and refers to non-augmented data from the simulation software (IPG Movie in this case).
- VDS** associated with "Distortion", "Vignetting", "Blur", and "All Effects", respectively highlighted in gray, pink, green, and violet, refers to augmented image data obtained from the camera model. For example, VDS** "All Effects" is a video data stream augmented with the distortion, vignetting, and blur effects.
- The same concept applies to the output of the VD algorithm, i.e., `Object List`.

7.3.4 Results

Taking into account that the VD algorithm used is based on Support Vector Machine (SVM), in the training process, a separating hyper-plane between both classes (vehicle and non-vehicle) is determined. From the calculated distance to the generated hyper-plane, an input sample is classified either as a vehicle or as a non-vehicle. Based on the calculated distance, a classification-confidence value is obtained for all input samples that are classified as vehicles.

The obtained classification-confidence values are then used to assess the effects of the camera model on the VD algorithm. In the following section, histogram representations illustrating the occurrence rate and the classification-confidence values are represented. Keeping in mind that the aim is to check the performance variations of the VD algorithm when tested on recorded VDS (VDS directly obtained from the camera) and other VDSs obtained from simulation, the object list obtained from the VD algorithm acting on recorded VDS is considered as the new ground truth. Additionally, in all histogram representations, positive detections that co-occurred (at the same frame) in both object lists coming from recorded and simulation use cases are considered for analysis.

In Fig. 7.11, classification-confidence values from *Recorded* and **Synthetic** images are represented. Most positive vehicle detection occurrences for **Synthetic** VDS lies below 50%, contrary to occurrences obtained from **Recorded** VDS. For the current use case, this gap indicates that testing the VD algorithm on data that is directly generated from simulation software, otherwise referred to as **Synthetic** VDS, does not provide a meaningful/accurate notion on how the VD algorithm would behave in reality. Conversely, directly testing on simulation data indicates that the algorithm would behave far worse than it would do on real image data.

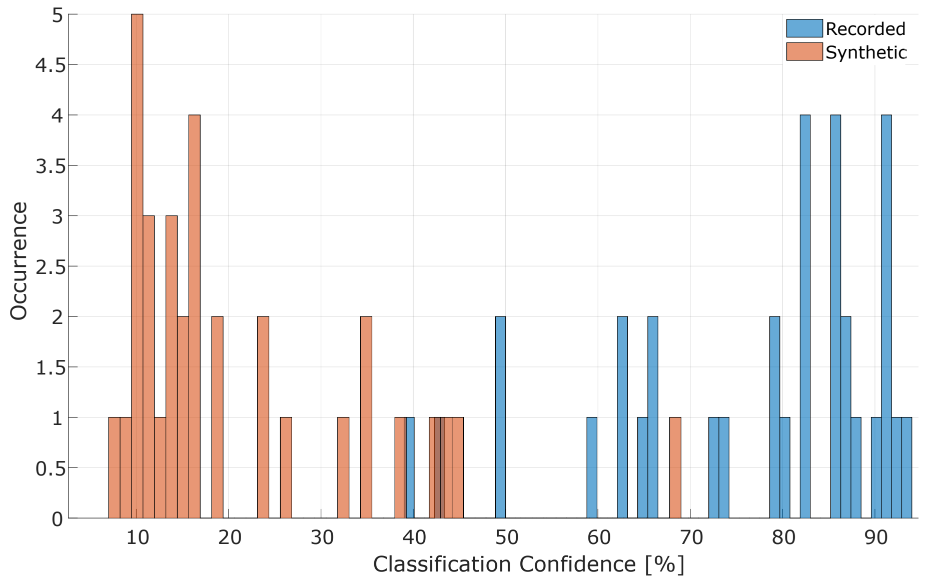


Figure 7.11: Histogram representation of VD object list from recorded and synthetic data.

Comparing classification-confidence values from *Recorded* and *Distorted* images (Fig. 7.12) shows similar results to those obtained in Fig. 7.11. It can be noticed that the distortion effect on its own is not capable of closing the gap between simulation and real data. This behavior is rooted in the fact that the distortion effect, in its essence, does not lead to information loss but information displacement. Additionally, as previously illustrated in 7.7, the VD algorithm relies on color, which is not affected by radial or tangential distortions, in addition to HOG features that, as an ensemble, are not highly sensitive to geometrical distortions, especially when the target lies approximately in the middle of the image where distortions are at their minimum.

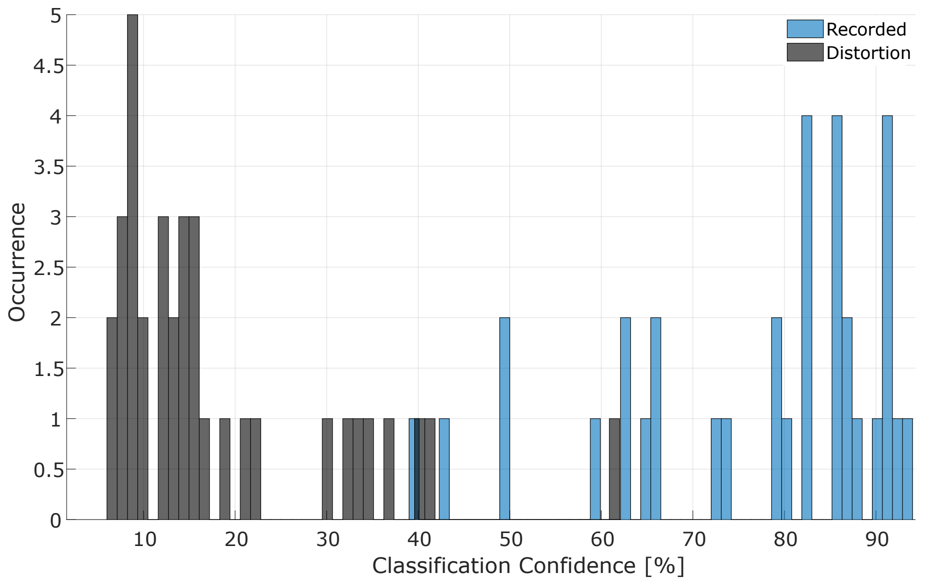


Figure 7.12: Histogram representation of VD object list from recorded and distorted data.

Vignetting, by definition, has a direct influence on pixel intensity values, i.e., color features. In Fig. 7.13, better results can be noticed, where several classification-confidence values obtained from **Vignetting** start to resemble those from **Recorded**. The difference is still relatively big yet better, and a part of this is due to the fact that the VD algorithm does not entirely depend on color features, but also on HOG features. Actually, the color feature vector is smaller than that of the HOG's feature vector.

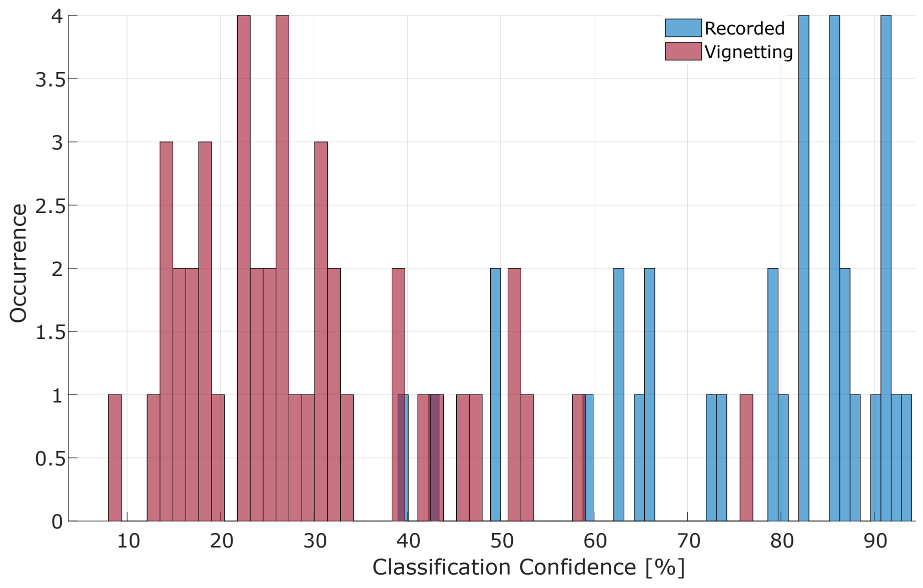


Figure 7.13: Histogram representation of VD object list from recorded and vignetting data.

In the case of the blur effect, Fig. 7.14 shows a better resemblance between both classification-confidence values. The blur model is implemented as a kernel convolution with the synthetic image; thus, it will have an effect on color and structural image features.

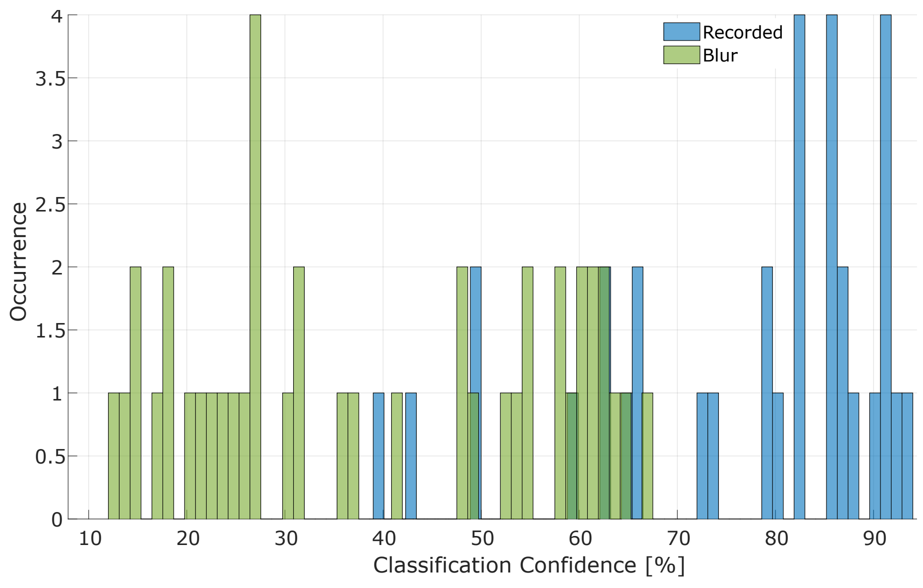


Figure 7.14: Histogram representation of VD object list from recorded and blurred data.

Finally, as presented in Fig. 7.15, when including all effects, i.e., distortion, vignetting and blur, **All Effects** classification-confidence values presented the best resemblance to **Recorded** values. It is observed that the changes made to color and HOG features had a direct impact on the VD algorithm.

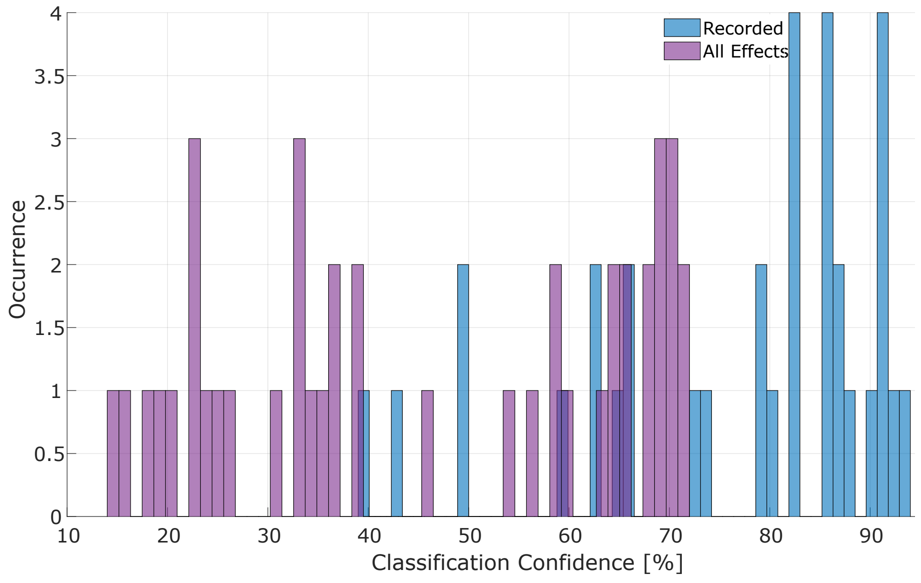


Figure 7.15: Histogram representation of VD object list from recorded and all effects data.

This experiment demonstrated the effects of the developed camera model on an image-based algorithm, namely VD algorithm. Though other effects, like temporal and spatial noise, can still be used for image augmentation, the aim here is not to obtain 100% similar classification-confidence values from simulation and real data, but to show and prove that accuracy can be increased when it comes to such results. Another essential aspect relies upon the nature of the detection algorithm and its functioning principle. In this regard, the distortion model may have yielded better results if the assessment were based on the accuracy of generated bounding box positions and not the classification-confidence values. Finally, it is also observed that one cannot derive a linear relationship between the influence of the implemented camera effects and their effect on the object list generated from the VD algorithm.

7.4 NN Training and Inference

7.4.1 Scope

Classification, detection, segmentation, and regression are considered the four fundamental tasks of autonomous driving. Taking into consideration the ability of DNN in performing these tasks, a natural cohesiveness between DNNs and autonomous driving can be seen. In general, training and testing DNNs for supervised learning relies on a large number of labeled datasets; this amount only increases in the context of autonomous driving, especially when the "driver" cannot take control to provide corrective actions and the system must be designed to a higher Automotive Safety Integrity Level (ASIL) [66].

For autonomous driving tasks, size, and variation of labeled training datasets are critical for performance improvement of DNNs, and several benchmark datasets are already available [67]–[71]. However, considering the vastly dynamic environment autonomous vehicles need to operate in and its everlasting variations, it is practically impossible to gather real datasets capable of capturing all real-world variabilities. One way to tackle these challenges is by bridging the gap between real and synthetic images for the scope of DNN training on virtually augmented images. This can be achieved using the developed camera model and the toolchain proposed in Ch. 6, Fig. 6.5. Using the Virtual KITTI Virtual KITTI (VKITTI) dataset [71], the following section demonstrates the camera model's usage in synthetic image augmentation for the scope of training a car detection DNN, and, by using the KITTI dataset [69], the DNN's behavior on real camera images.

7.4.2 Method Description

For the augmentation pipeline, the previously generated lookup table for vignetting implementation is used. Fig. 7.16 represents the main workflow of this approach. In Fig. 7.16, VKITTI dataset is fed into the camera model, which performs a pixel-wise computation on the original image for vignetting implementation. As a result, "Augmented VKITTI" dataset is produced with the vignetting effect implemented on each image of the dataset.

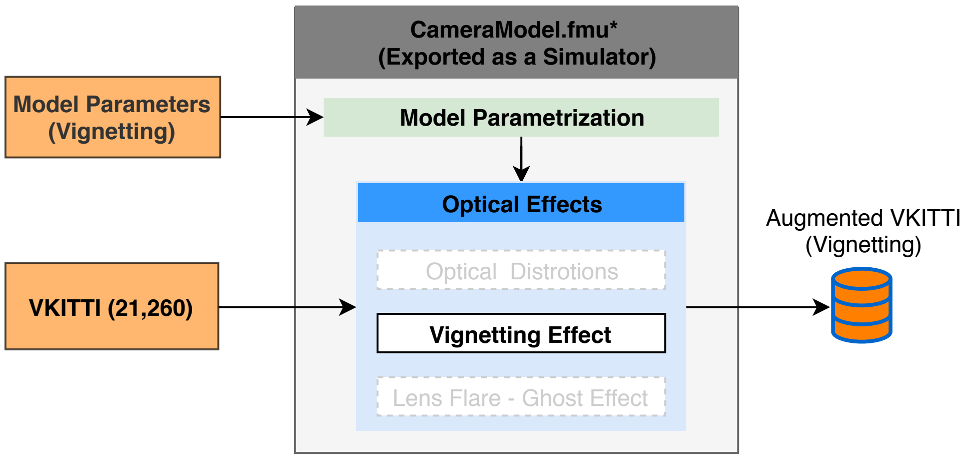


Figure 7.16: Vignetting augmentation pipeline.

Taking into consideration that the primary purpose of the camera model is to demonstrate the effects of augmented synthetic data on DNN's performance, only the "Car" class is considered. Additionally, the process is configured to start from a pre-trained Faster R-CNN [72] with ResNet50 [73] as a feature extractor and weights pre-trained on ImageNet [74]. The model is trained with an open-source implementation [75], using Adam optimizer [76] for both the classifier and the regional proposal network with a fixed learning rate of 10^{-5}.

7.4.3 Experimental Setup

Fig. 7.17 represents the setup used for evaluating the effects of vignetting on DNN training and "Car" detection. In Fig. 7.17, two independent instances for training the DNN on original synthetic images (non-augmented) and images augmented by the vignetting effect are configured; both datasets consist of 21260 images. In Fig. 7.18, the mean average precision of the network, calculated on the validation dataset, is represented, where it can be seen that both networks reached approximately the same precession values.

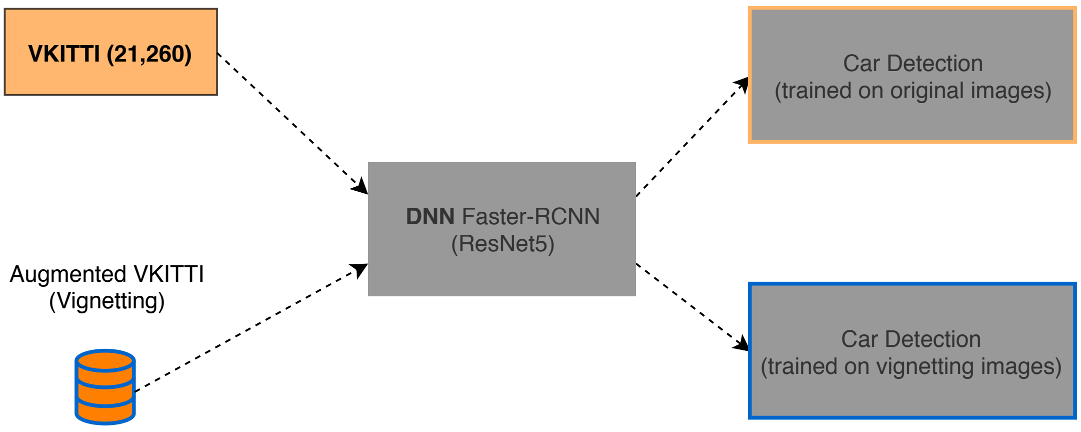

VKITTI (21,260)
Augmented VKITTI (Vignetting)
DNN Faster-RCNN (ResNet5)
Car Detection (trained on original images)
Car Detection (trained on vignetting images)

Figure 7.17: DNN training setup.

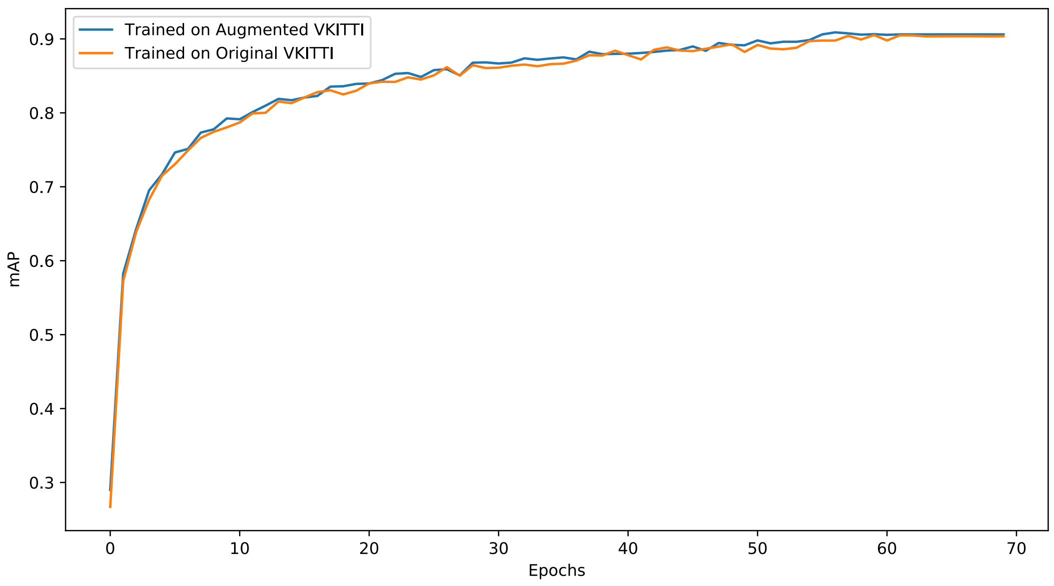


Figure 7.18: Mean average precision on VKITTI validation dataset.

The final step in the experiment setup is to check the trained DNN's performance results on real images, which is attained by performing inferencing using the KITTI dataset for 2D object detection [69] as demonstrated in Fig. 7.19. The same dataset is fed into the trained car detection algorithms, and the results are recorded for further performance assessment and analysis.

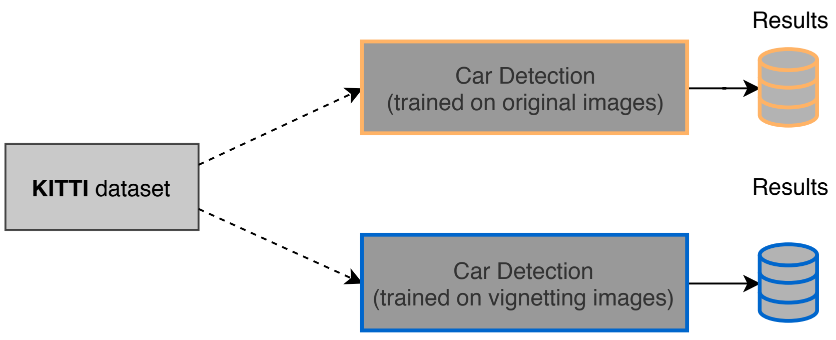


Figure 7.19: Inferencing setup.

7.4.4 Results

To assess the performance differences between DNN trained on ideal synthetic images and DNN trained on augmented synthetic images, the mean Average Precision (mAP) during training, validation, and inferencing is used.

Table 7.1, represents the obtained results after inferencing is applied. DNN trained on the augmented dataset showed a 5.84% increase in precession when tested on the real KITTI dataset. In order to understand the effect of vignetting on the observed precision gain, it is worth pointing out that included in the 5.84% precision increase, 970 true positive detections are unique to the DNN trained on augmented data. Fig. 7.20 represents their distribution over the image space.

Table 7.1: Inferencing results on KITTI dataset.

Augmentation Method	**Precision [%]**	
	Value	*Gain*
No Augmentation	83.12	-
Vignetting	88.96	+5.84

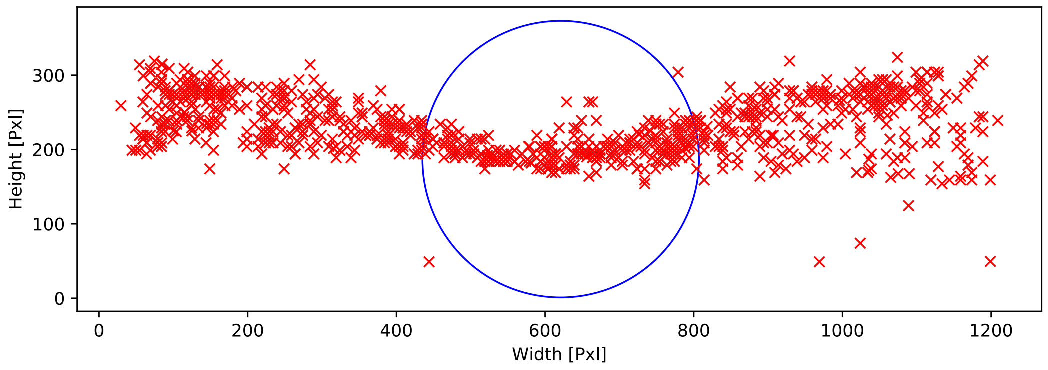


Figure 7.20: Exclusive true positives to DNN trained on augmented images.

In Fig. 7.20, the red crosses represent the center of the bounding boxes, whereas the blue circle (slightly less than the image height in diameter, i.e., 355 pixels) represents a boundary where vignetting, theoretically, should not have a significant effect on light intensity fall-off. By computing the average intensity for the bounding boxes of detected cars outside the blue circle, the distribution represented in Fig. 7.21 is obtained.

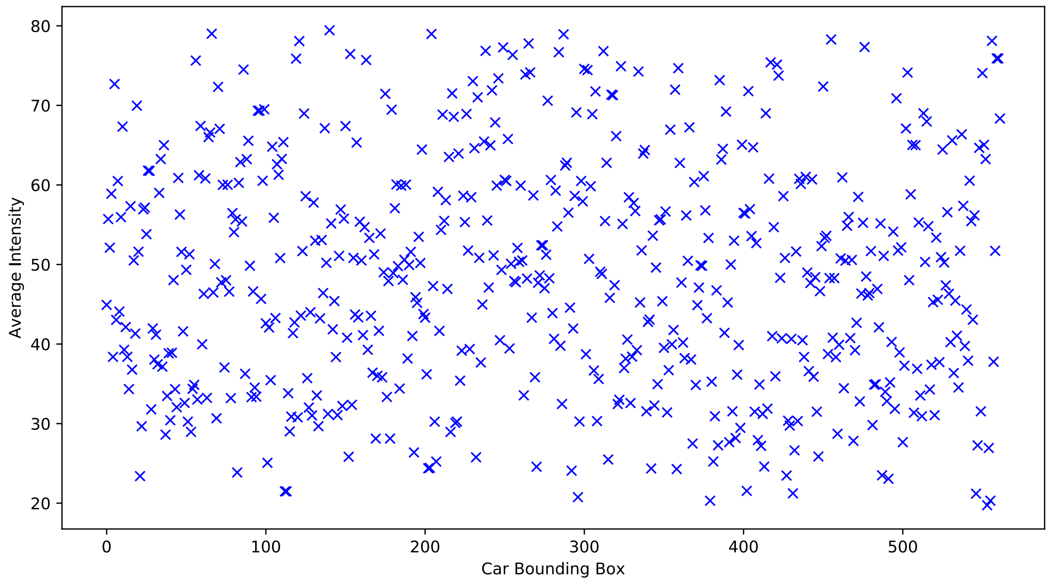


Figure 7.21: Average bounding box intensity distribution.

The pixel intensity range of the original image may vary between 0 and 255. In Fig. 7.21, the maximum average intensity is approximately 80, and the minimum is slightly below 20. The average intensity of pixels behind the bounding boxes is a result of several factors, such as a dark paint job, shadows, or even occlusions. For example, in Fig. 7.22, a vehicle occluded by tree shadows is only detected by the DNN trained on the augmented dataset.

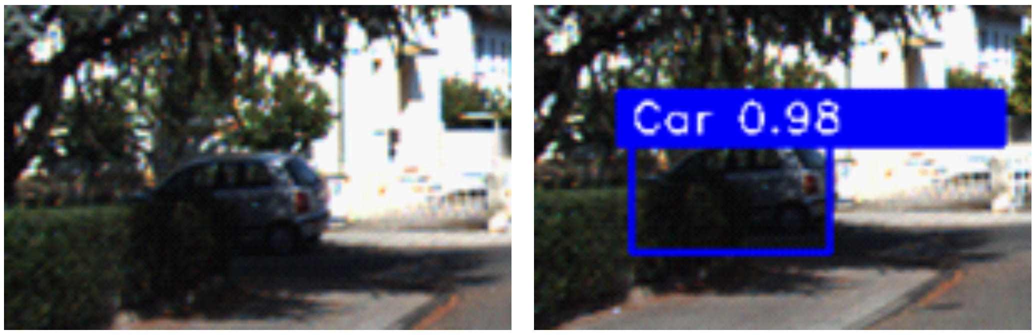


Figure 7.22: Detected car under low illumination by the DNN trained on the augmented dataset (right figure).

By further examining the distribution of bounding boxes with average light intensity less than 60 (considered as a low illumination value), it can be observed that the initial histogram diagram represented in Fig. 7.23 resembles the shape of the light's fall-off graph represented in Ch. 5 (Fig. 5.7). In Fig. 7.23, the missing bins in the middle of the graph can be observed due to the fact that corresponding detections for the current representation are excluded. Fig. 7.23 only provides an abstract representation, for this distribution is highly dependent on the initial number of vehicles that did not lie at the image center but toward image peripherals. To account for this situation, in Fig. 7.24 the relative percentage of detected vehicles to the total number of detected vehicles over the entire image width is represented.

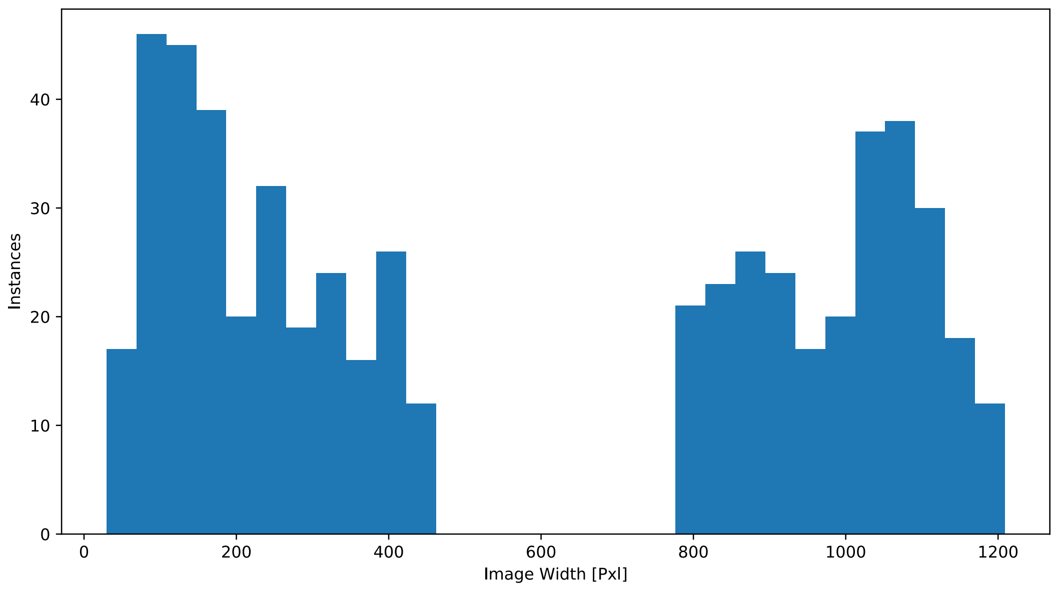


Figure 7.23: Histogram representation of bounding box center of mass.

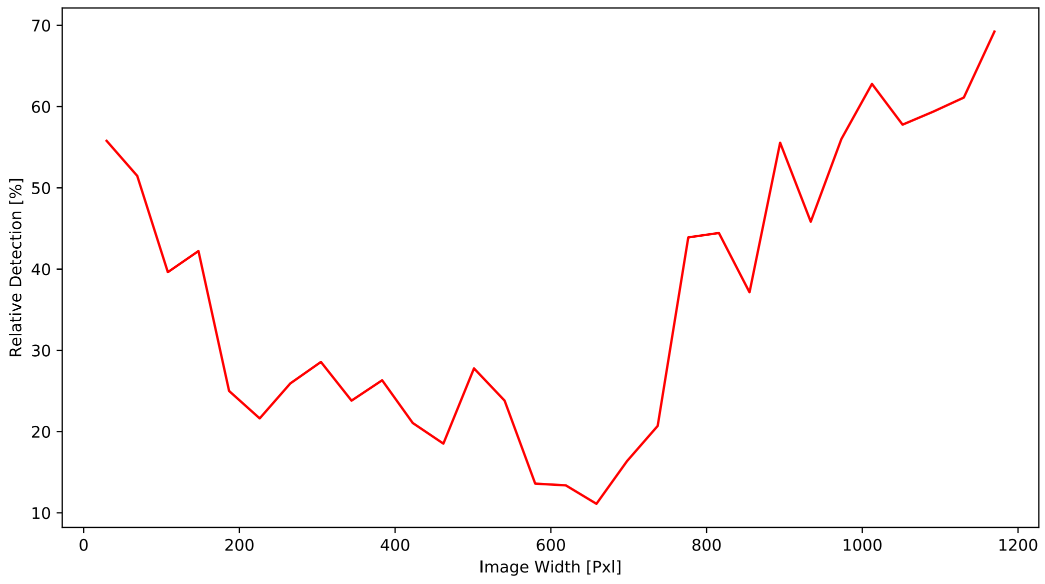


Figure 7.24: Car relative detection over the entire image width.

Fig. 7.24 shows how more cars were detected in low light conditions at both margins of the image than those detected with low light conditions around the center of the image. When considering the detected vehicles in better light conditions (100 Relative Detection [%]), the graph starts to resemble the light fall-off graph in a more intuitive way. Illustrated in Fig. 7.24, the percentage of detected vehicles follows the trend at which the light intensity falls off at the image edges'.

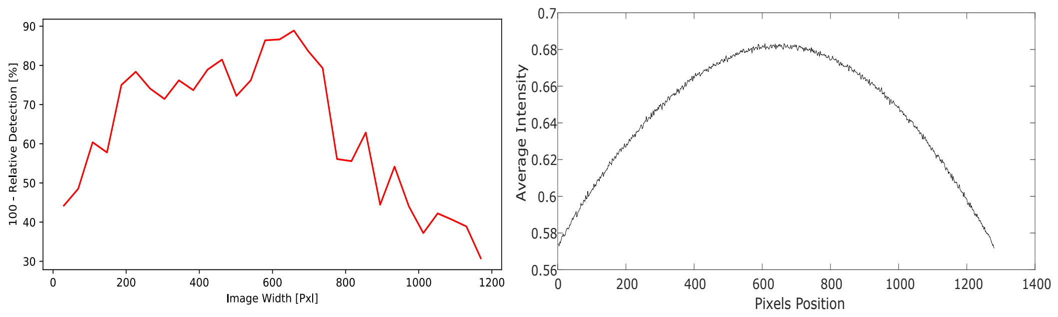


Figure 7.25: Car relative detection (left graph) and light fall-off representaion (right graph).

8 Conclusion and Outlook

In order to achieve the main aim of this dissertation, a physical model for an advanced driver-assistance camera was created, a hybrid test strategy initiated, and co-simulation frameworks with standardized interfaces envisioned.

For the modeling process of an MFC, the system architecture and the camera's interface were illustrated and used as a starting point to identify relevant optical and image sensor effects. Followed by model parametrization, calibration and implementation, this work demonstrated the importance of modularity and interchangeability in modeling and simulation since not all results can be generalized for all cameras, even when dealing with identical cameras with precisely the same hardware components. To sum up this section, the modeling process used was applied to three cameras; however, the same concept is also applicable to other perception sensors like radar or lidar.

With the hybrid development and test strategies, the usage of the camera model was demonstrated in different setups, where its applicability in testing classical CV algorithms and NN image-based algorithms was illustrated. The test strategy showed how the behavior of several detection functions, namely LD and VD algorithms, differ when tested on real, synthetic (non-augmented), and augmented (camera model output) image data. The usage of the camera model is also shown in the case of training DNNs, where DNNs trained on augmented images provided better results than those that were trained on purely synthetic images. More precisely, the correlation between the vignetting effect and performance improvement of DNN was shown. The obtained results can be utilized as springboards for more investigations in this domain where other effects can be put to the test and similar approaches can be applied to other perception sensors like radar or lidar.

Finally, considering the importance of flexible and modular co-simulation platforms, standardized interfaces, like FMI and OSI, are integrated and enabled for the overall system architecture. By encapsulating the camera model and an LD algorithm inside two separate FMUs, performing various test cases with different simulation software (hosted on Linux or Windows) was shown to be possible. OSI, a relatively embryonic interface with promising potential, was also integrated into the camera model.

ADAS and AD function development and testing are hard to achieve with "only" elaborate real tests, results presented in Ch. 7 demonstrated the usage and the importance of camera sensor models. In conclusion, generating modular and interchangeable sensor models and shifting towards simulation are some of the essential pillars for reaching fully autonomous vehicles.

It is also important to point out the importance of defining appropriate abstraction levels throughout the model generation process, as it was demonstrated that one generic sensor model may not be suitable for all use cases. This dissertation aims to lay the groundwork for simulation and virtualization-based approaches. From the author's perspective, these approaches are not limited to camera sensors, but can also be applied to other sensors related to autonomous driving.

Bibliography

[1] EUROPEAN COMMISSION, *EUROPE ON THE MOVE: Sustainable Mobility for Europe: safe, connected and clean: ANNEX 1*, EUROPEAN COMMISSION, Ed. 2018.

[2] W. Team, *How simulation turns one flashing yellow light into thousands of hours of experience*, [Online; accessed 30-June-2019]. [Online]. Available: `https://medium.com/waymo/simulation-how-one-flashing-yellow-light-turns-into-thousands-of-hours-of-experience-a7a1cb475565`.

[3] W. C. Partners, *Beyond the headlights, adas and autonomous sensing*, [Online; accessed 11-June-2019]. [Online]. Available: `http://www.woodsidecap.com/wcp-publishes-adasautonomous-sensing-industry-beyond-headlights-report/`.

[4] Kevin Forsberg and Harold Mooz, "The relationship of system engineering to the peoject cycle," 1991.

[5] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Handbook of Driver Assistance Systems*. Cham: Springer International Publishing, 2016, ISBN: 978-3-319-12351-6. DOI: `10.1007/978-3-319-12352-3`.

[6] Ron Wasserstein, "George box: A model statistician. significance, 7," pp. 134–135, 2010. DOI: `10.1111/j.1740-9713.2010.00442.x`.

[7] George E. P. Box, "Science and statistics," vol. 71, no. 356, pp. 791–799, 1976. [Online]. Available: `http://links.jstor.org/sici?sici=0162-1459%28197612%2971%3A356%3C791%3ASAS%3E2.0.CO%3B2-W`.

[8] L. W. Barsalou, "Abstraction in perceptual symbol systems," *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 358, no. 1435, pp. 1177–1187, 2003, ISSN: 0962-8436. DOI: `10.1098/rstb.2003.1319`.

[9] S. Online, *Zirkus-zebra trickst polizisten aus*, [Online; accessed 27-June-2019]. [Online]. Available: `https://www.spiegel.de/panorama/gesellschaft/rheinland-pfalz-zirkus-zebra-loest-in-bitburg-polizeieinsatz-aus-a-914835.html`.

[10] L. BArtleet, *11 really weird facts about teletubbies*, [Online; accessed 27-June-2019]. [Online]. Available: `https://www.nme.com/blogs/tv-blogs/teletubbies-facts-2226455`.

[11] J. C. V. Guerra, Z. Khanam, S. Ehsan, R. Stolkin, and K. McDonald-Maier, *Weather classification: A new multi-class dataset, data augmentation approach and comprehensive evaluations of convolutional neural networks*. [Online]. Available: `http://arxiv.org/pdf/1808.00588v1`.

[12] A. Torres, *Vehículos que fueron extremadamente sobrecargados*, [Online; accessed 27-June-2019]. [Online]. Available: `https://www.taringa.net/+humor/vehiculos-que-fueron-extremadamente-sobrecargados_hhmv7`.

[13] Harsha Jakkanahalli Vishnukumar, Christian Müller, Björn Butting, and Eric Sax, *Machine Learning and Deep Neural Network – Artificial Intelligence Core for Lab and Real-World Test and Validation for ADAS and Autonomous Vehicles: AI for efficient and quality test and validation*. Piscataway, NJ: IEEE, 2017, ISBN: 9781509064366. [Online]. Available: `http://ieeexplore.ieee.org/servlet/opac?punumber=8318444`.

[14] Edward A. Lee, "Cyber physical systems: Design challenges," 2008.

[15] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, et al., "The functional mockup interface for tool independent exchange of simulation models," in *8th International Modelica Conference*. DOI: `10.3384/ecp11063105`.

[16] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, *Co-simulation: State of the art*. [Online]. Available: `http://arxiv.org/pdf/1702.00686v1`.

[17] A. Artunedo, J. Godoy, R. Haber, J. Villagra, and R. M. d. Toro, "Advanced co-simulation framework for cooperative maneuvers among vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, 15-Sep-15 - 18-Sep-15, pp. 1436–1441, ISBN: 978-1-4673-6596-3. DOI: `10.1109/ITSC.2015.235`.

[18] Harald Waschl, Roman Schmied, Daniel Reischl and Michael Stolz, "A virtual development and evaluation framework for adas—case study of a p-acc in a connected environment," pp. 107–117, 2019. [Online]. Available: `https://doi.org/10.1007/978-3-319-91569-2_6`.

[19] J. Barceló, "Fundamentals of traffic simulation: Microscopic traffic flow simulator vissim," vol. 145, pp. 63–65, 2010. DOI: `10.1007/978-1-4419-6142-6`.

[20] Robert Lajos Bucs, Pramod Lakshman, Jan Henrik Weinstock, et al., "Fully virtual rapid adas prototyping via a joined multi-domain co-simulation ecosystem," pp. 59–69, 2018.

[21] Robert Lajos Bucs, Luis Gabriel Murillo, Ekaterina Korotcenko, and et al., "Virtual hardware-in-the-loop co-simulation for multi-domain automotive systems via the functional mock-up interface," no. 15588262, 2015.

[22] IEEE Computer Society, *Ieee standard system c language reference manual*, Piscataway, NJ, USA. DOI: 10.1109/IEEESTD.2006.99475.

[23] Martin Krammer, Helmut Martin, Zoran Radmilovic, and et al., "Standard compliant co-simulation models for verification of automotive embedded systems," 2015.

[24] IEEE Computer Society, *Ieee standard for standard systemc(r) analog/mixed-signal extensions language reference manual*, Piscataway, NJ, USA. DOI: 10.1109/IEEESTD.2016.7448795.

[25] C. van Driesten and T. Schaller, "Overall approach to standardize ad sensor interfaces: Simulation and real vehicle," in *Fahrerassistenzsysteme 2018*, T. Bertram, Ed., Wiesbaden: Springer Fachmedien Wiesbaden, 2019, pp. 47–55, ISBN: 978-3-658-23751-6.

[26] Pegasus, *Pegasus-osi*, [Online; accessed 11-June-2019]. [Online]. Available: https://www.pegasusprojekt.de/en/home.

[27] J. Chen, K. Venkataraman, D. Bakin, B. Rodricks, R. Gravelle, P. Rao, and Y. Ni, "Digital camera imaging system simulation," *IEEE Transactions on Electron Devices*, vol. 56, no. 11, pp. 2496–2505, 2009, ISSN: 0018-9383. DOI: 10.1109/TED.2009.2030995.

[28] Junqing Chen, "Imaging sensor modulation transfer function estimation: Hong kong, 26 - 29 sept. 2010," pp. 533–536, 2010. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=5641636.

[29] Ali Mosleh, Paul Green, Emmanuel Onzon, and et al., "Camera intrinsic blur kernel estimation: A reliable framework," pp. 4961–4968, 2015. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=7293313.

[30] C. Wittpahl, H. B. Zakour, M. Lehmann, and A. Braun, *Realistic image degradation with measured psf.* [Online]. Available: http://arxiv.org/pdf/1801.02197v1.

[31] A. Carlson, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, *Modeling camera effects to improve visual learning from synthetic data*. [Online]. Available: http://arxiv.org/pdf/1803.07721v6.

[32] ——, *Sensor transfer: Learning optimal sensor effect image augmentation for sim-to-real domain adaptation*. [Online]. Available: http://arxiv.org/pdf/1809.06256v2.

[33] rFpro, *Rfpro*, [Online; accessed 11-June-2019]. [Online]. Available: http://www.rfpro.com/.

[34] J. Horgan, C. Hughes, J. McDonald, and S. Yogamani, "Vision-based driver assistance systems: Survey, taxonomy and advances," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, 15-Sep-15 - 18-Sep-15, pp. 2032–2039, ISBN: 978-1-4673-6596-3. DOI: 10.1109/ITSC.2015.329.

[35] L. Fridman, *Tesla Vehicle Deliveries and Autopilot Mileage Statistics*, Jan. 2019. DOI: 10.5281/zenodo.2530449.

[36] Tesla, *Autopilot: Tesla-support*, [Online; accessed 03-April-2019]. [Online]. Available: https://www.tesla.com/support/autopilot.

[37] ——, *Autopilot: Full self-driving hardware on all cars*, [Online; accessed 03-April-2019]. [Online]. Available: https://www.tesla.com/autopilot.

[38] Fei-Fei Li, Andrej Karpathy, Justin Johnson Andrej Karpathy, and Justin Johnson, "Spatial localization and detection," 2016.

[39] R. Szeliski, "Computer vision: Algorithms and applications," 2010. [Online]. Available: http://szeliski.org/Book/.

[40] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "End-to-end deep reinforcement learning for lane keeping assist," 2016. [Online]. Available: http://arxiv.org/pdf/1612.04340v1.

[41] FMI development group, *Functional mock-up interface for model exchange and co-simulation, version 2.0*, 2014.

[42] TUM, *Tum-osi*, [Online; accessed 11-June-2019]. [Online]. Available: https://www.hot.ei.tum.de/forschung/automotive-veroeffentlichungen/.

[43] S.-A. Schneider and K. Saad, "Camera behavioral model and testbed setups for image-based adas functions," *e & i Elektrotechnik und Informationstechnik*, vol. 135, no. 4-5, pp. 328–334, 2018, ISSN: 0932-383X. DOI: 10.1007/s00502-018-0622-7.

[44] Francis A. Jenkins Harvey E.White, *Fundamentals-of-optics*, Fourth Edition. New York, 2001, ISBN: 0-07-256191-2.

[45] P. K. Sinha, *Image acquisition and preprocessing for machine vision systems*. Bellingham Wash.: SPIE, 2012, vol. vol. no. PM197, ISBN: 0819482021.

[46] Meixian Wu, "Light scattering in worn windscreens: Development of technique for quantitative measurement of visibility degradation," PhD thesis, KTH Royal Institute of Technology.

[47] E. D. Dickmanns, *Ground vehicle guidance by vision*, [Online; accessed 28-June-2019]. [Online]. Available: `http://dyna-vision.de/`.

[48] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[49] P. D. Lin, *Advanced Geometrical Optics*. Singapore: Springer Singapore, 2017, vol. 4, ISBN: 978-981-10-2298-2. DOI: `10.1007/978-981-10-2299-9`.

[50] D.-L. Lin, C.-C. Wang, and C.-L. Wei, "Quantified temperature effect in a cmos image sensor," *IEEE Transactions on Electron Devices*, vol. 57, no. 2, pp. 422–428, 2010, ISSN: 0018-9383. DOI: `10.1109/TED.2009.2037389`.

[51] P. E. Douglas A. Kerr, "Derivation of the "cosine fourth" law for falloff of illuminance across a camera image," no. 4, 2007.

[52] J. Nakamura, *Image sensors and signal processing for digital still cameras*. Boca Raton FL: Taylor & Francis, 2006, ISBN: 0849335450.

[53] Ali Tourani, Asadollah Shahbahrami, and Alireza Akoushideh, "Challenges of video-based vehicle detection and tracking in intelligent transportation systems," 2017.

[54] B. McCleary and A. Ortega, "Photo-response non-uniformity error tolerance testing methodology for cmos imager systems," S. P. Farnand and F. Gaykema, Eds., ser. SPIE Proceedings, SPIE, 2009, p. 724 216. DOI: `10.1117/12.807557`.

[55] B. Deegan, "Led flicker: Root cause, impact and measurement for automotive imaging applications," *Electronic Imaging*, vol. 2018, no. 17, pp. 146–1–146–6, 2018, ISSN: 2470-1173. DOI: `10.2352/ISSN.2470-1173.2018.17.AVM-146`.

[56] A. E. Conrady, "Decentred lens-systems," pp. 384–390, 1919. [Online]. Available: `https://academic.oup.com/mnras/article-abstract/79/5/384/107877`.

[57] D. C. BROWN, Ed., *Decentering Distortion of Lenses: The prism effect encountered in metric cameras can be overcome through analytic calibration.* 1965.

[58] K F Cart, "Integrating sphere theory and applications part ii: Integrating sphere applications," 1997.

[59] Matthias B. Hullin, Elmar Eisemann, Hans-Peter Seidel, and Sungkil Lee, "Physically-based real-time lens flare rendering,"

[60] N. Matsui and M. Shigyo, "Brushless dc motor control without position and speed sensors," in *Conference Record of the 1990 IEEE Industry Applications Society Annual Meeting*, IEEE, 7-12 Oct. 1990, pp. 448–453, ISBN: 0-87942-553-9. DOI: 10.1109/IAS.1990.152224.

[61] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation," *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–33, 2018, ISSN: 03600300. DOI: 10.1145/3179993.

[62] J. L. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez, and J. Fernandez-Valdivia, "Diatom autofocusing in brightfield microscopy: A comparative study," in *15th international conference on pattern recognition*, A. Sanfeliu, Ed., IEEE Comput. Soc, 2000, pp. 314–317, ISBN: 0-7695-0750-6. DOI: 10.1109/ICPR.2000.903548.

[63] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, *Bdd100k: A diverse driving video database with scalable annotation tooling*. [Online]. Available: http://arxiv.org/pdf/1805.04687v1.

[64] Jonathan Krause, "3d object representations for fine-grained categorization," 2013.

[65] Herman Van der Auweraer, "Virtual engineering at work: The challenges for designing mechatronic products," *Engineering with Computers*, vol. 29, no. 3, pp. 389–408, 2013, ISSN: 0177-0667. DOI: 10.1007/s00366-012-0286-6.

[66] I. S. Electrical, electronic components, and general system aspects, *Road vehicles, Functional Safety, Part 3, ISO 26262-3*, Nov. 2011.

[67] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Dec. 2012.

[68] Zhou, B. A. Lapedriza, A. Khosla, A. Olivia, and A. Torralba, *Places: A 10 million image database for scene recognition*, 2017.

[69] A. Geiger, P. Lenz, and R. Urtasun, *Are we ready for autonomous driving? The KITTI vision benchmark suite*, 2012.

[70] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, *A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation*, Dec. 2015.

[71] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, *Virtual worlds as proxy for multi-object tracking analysis," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[72] S. Ren, K. He, R. Girshick, and J. Sun, *Faster R-CNN: Towards real-time object detection with region proposal networks*, 2015.

[73] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition"*, Dec. 2015.

[74] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *ImageNet: A Large-Scale Hierarchical Image Database"*, 2009.

[75] J. Tian, *Keras implementation of faster-rcnn*, [Online; accessed 30-June-2019]. [Online]. Available: `https://github.com/jinfagang/keras_frcnn`.

[76] D. P. Kingma and J. Lei Ba, *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION"*, Jan. 2017.