2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)
September 12-14, 2018, Madrid, Spain

Test Methodology for Vision-Based ADAS Algorithms with an Automotive Camera-in-the-Loop

Fabio Reway
CARISSMA
Technische Hochschule Ingolstadt
Ingolstadt, Germany
Email: Fabio.Reway@carissma.eu

Werner Huber
CARISSMA
Technische Hochschule Ingolstadt
Ingolstadt, Germany
Email: Werner.Huber@thi.de

Eduardo Parente Ribeiro
Department of Electrical Engineering
Federal University of Parana (UFPR)
Curitiba, Brazil
Email: Edu@eletrica.ufpr.br

***Abstract*—In order to correctly perceive its surroundings, advanced driver-assistance systems (ADAS) rely on the data quality of environment sensors, such as cameras, and on the data processing to distinguish multiple classes of traffic participants. Real test drives are important for their testing and validation, but certain test scenarios are difficult to be reproduced or automated, e.g. adverse weather conditions. Therefore, it is essential to bring this system to a controlled virtual environment so that it is possible to determine their correctness and performance under these circumstances before their release. For this reason, Hardware-in-the-Loop testing methods have been increasingly utilized in the industry, with which real hardware is connected to driving simulation software and deficiencies can be identified in a early development phase. This paper presents a test setup with a real automotive Camera-in-the-Loop and a testing method to evaluate a proprietary algorithm for multi-class object detection of an ADAS platform available on the market and validate the specifications described by its manufacturer.**

I. Introduction

The environment sensors – such as LIDAR, radar and camera – are essential for enabling automated driving. Particularly the “eyesight" of a camera and algorithms for processing the captured image data allow advanced driver-assistance systems (ADAS) to differentiate road users such as pedestrians, other vehicles, cyclists and traffic signs. Thus, it is possible for the ego vehicle to perceive its surroundings. However, this system may have its capabilities drastically limited under certain circumstances. The high requirements levels for dynamic range and adverse weather condition (e.g. rain and fog) are a major challenge to its correct operation [1]. In this case, it may insufficiently detect or mistakenly classify the objects associated with the driving scenario. Therefore, the risk of accidents increases, as the vehicle has an unreal or imprecise interpretation of the real world, which can negatively influence the decision-making algorithms of ADAS.

The fatal accidents with automated cars in May 2016 [2] and in March 2018 [3] reinforce the importance of testing environment sensors and driving algorithms more intensively before bringing them to the streets. The first one has happened because the sensors failed to detect a white trailer crossing the highway against a bright sky (Figure 1, left side). In case of the other accident, the car was unable to react properly to a pedestrian pushing a bicycle while crossing the street in the middle of the night (Figure 1, right side).

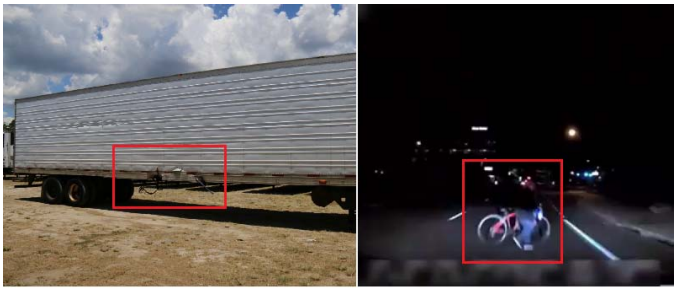

Figure 1. (Left) The white-trailer hit by a partial automated car back in 2016 [2]. (Right) The soon-to-be hit pedestrian victim in the accident in 2018 in which another automated car was involved [3]

Since the performance of the sensors must be identified in these challenging cases, testing them under adverse conditions is required so that the system boundaries are defined and accidents like that can be avoided. Real test drives are an essential part for securing these systems, but they cannot cover all possible scenarios that the sensors will be exposed to [4]. It is estimated that autonomous driving requires billions of kilometers of test drives in order to ensure safety in usage and functional safety [5], suggested by the ISO 26262 [6]. For this reason, despite the high degree of validity of their test results, they are associated with high time and cost expenditures, i.e., difficult to be reproduced. Consequently, simulation has been increasingly employed in industry in the last years, since it allows safety-critical scenarios to be reproducibly carried out under controllable conditions in the laboratory [7].

Outline

This paper is organized as follows: Section II presents an overview of the related work. Section III describes the test setup that includes a real automotive Camera-in-the-Loop and the developed test methodology used to quantify the quality of an algorithm for multi-class object detection running on an ADAS development platform. Here, it is expected that the provided details encourage the reproduction of this method. In Section IV, the obtained results are discussed and compared with the specifications provided by the manufacturer. Section V concludes and presents the contributions of this work.

978-1-5386-3543-8/18/$31.00 ©2018 IEEE

II. Related Work

For the development and automated testing of electronic control units (ECUs) and their software, the Hardware-in-the-Loop (HiL) method has been employed since early 90's [8], [9]. It is used for testing a controller first against its specifications individually and, once the system under test (SUT) has been verified, can be used for testing its integration with other components. The simulation takes part in this process by means of generating the necessary input or stimuli for the SUT in real time. One of the most well established approaches for validating the hardware of vision-based ADAS "in-the-loop" is via recorded video sequences. However, this is an inflexible approach, since the recorded events cannot be altered and, for this reason, the feedback control systems cannot be easily tested under different conditions [10]. For an improved validation of ADAS, the development of test methods is needed so that the hardware of environment sensors can be stimulated inside the laboratory with synthetic data – this can be referred to as "Sensor-in-the-Loop". This approach allows the creation of customized scenarios and, therefore, increases the number of possible covered test cases tremendously.

Different approaches for data injection in environment sensors have been analysed in [11]. The authors illustrate a simplified setup, in which a screen or projector emits light and injects it directly into a camera's optics. In [12], a test bench for vision-based control of unmanned air vehicles is presented – out of the context of driving, but with a similar approach of placing a camera in front of screen. In [13], a similar method is referred to as "Monitor HiL", considered to be the most cost-effective approach to inject synthetic data in a camera ECU.

Besides cameras, authors presented techniques to include even radar sensors in the loop. The authors in [14] stimulate a real radar sensor by feeding radar raw data in the algorithm for object classification in the radar ECU and identify the limitations for creating test drives scenarios with such an approach. The work in [15] demonstrates how to manipulate the reflected radar electromagnetic waves in order to emulate virtual radar targets for a real sensor.

The validity of securing a highly automated driving function in a virtual test domain was addressed in [16]. Using a scenario-based method that combines a scenario description and key performance indicators (KPI), it is possible to adjust and transfer the tests of automated driving from the real environment to a virtual domain [16]. However, this kind of approach is not sufficiently mature yet - especially with regard to the large number of necessary test scenarios to be modelled.

The future ADAS and automated driving functions are based on multiple integrated sensors. However, the current development stage of testing methods or test benches for this kind of systems are either purely virtual or limited to individual components. While testing a single camera with this method is an already well established approach in the industry, such complete testing framework – as shown in this paper – is yet to be explored. This work combines the advantages of virtual testing and HiL testing for multiple sensors. In the future stage of this work, different sensors principles (video, radar and LIDAR) should be stimulated by one virtual scenario. This paper focuses on the first step, which is stimulating multiple real cameras using a single testing environment.

III. Materials and Methods

The first subsection "Test Setup" introduces the software and hardware elements used for carrying out the tests, i.e. it is a description of the mounted test bench. The second subsection "Test Strategy" should clarify how the tests were undertaken and what parameters were considered for the evaluation of the ADAS algorithm.

A. Test Setup

The test setup is composed by four main elements: Environment Model, Automotive Camera, ADAS Platform and the Driving Function. Fig. 2 illustrates the designed arrangement of these elements for a clear understanding. An Environment Model is needed for the vehicle dynamics simulation and scene generation. The automotive camera is used to capture the computer-generated image frames and is connected to the ADAS Platform. In this platform, an algorithm is implemented, with which it is possible to process the acquired data and to classify different classes of traffic participants. Based on this, the model of a driving function is used as the decision-maker that re-engages in the simulation so that the virtual car undertakes a determined action in a closed-loop.

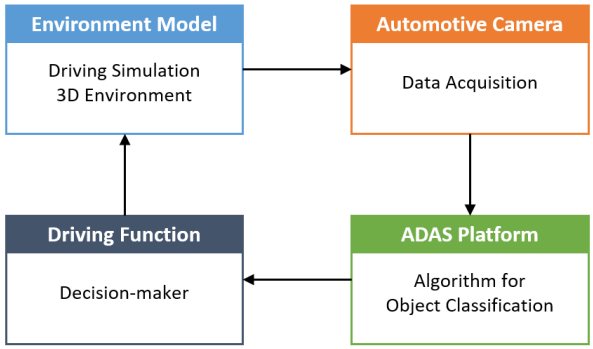

Figure 2. Designed arrangement of the four main elements in the developed Test Setup

Environment Model: It is able to perform the dynamic simulation of the ego vehicle and all traffic participants in a driving scenario. In real time, it is able to generate the synthetic image data that corresponds to the surroundings of the virtual ego vehicle. Some simulation software available on the market – such as Virtual Test Drive from VIRES or CarMaker from IPG – enable a customized construction of a test scenario in a virtual 3D environment. It is possible to generate the image data from any perspective of the vehicle placing the camera in a custom position in the virtual car.

Automotive camera: It is a series-produced camera specifically designed for ADAS applications that utilizes the proprietary Gigabit multimedia serial link (GMSL) technology. It has a resolution of 1928x1208 Pixel (2.3M Pixel) and a frequency rate of 30fps. In this work, two different types of camera lenses were used. The first having a field of view (FOV) of 60° for front and rear views and the other 100° for lateral view. The cameras were connected to the ADAS Platform with a FAKRA-Z connector. The output data is generated in a raw format;

ADAS Platform: It is a series-produced device designed to be the core of a centralized architecture for automated driving applications running an UNIX operating system. It has multiple interfaces for GMSL cameras and standard automotive protocols, such as CAN, FlexRay, LIN and automotive Ethernet. A deep neural network – already trained by its developer and implemented in this platform – is able to process the raw data images from multiple cameras, detect in real-time and classify five different classes of objects: cars, trucks, bicycles, pedestrians and traffic signs.

Driving Function: This driving function was modeled in MATLAB Simulink as an accident avoiding system. Its purpose is to prevent a collision in the simulated test scenario. It takes into consideration the camera detections and also the distance from the ego vehicle to other traffic participants, which can be directly retrieved from the radar model implemented in the environment model software. For simplification purposes, the triggering of the driving function relies on a fixed distance threshold (safety-critical distance) from the ego vehicle to other traffic participants.

The detailed architecture of this test setup is illustrated on Fig. 3. The color of its frames make reference to Fig. 2.

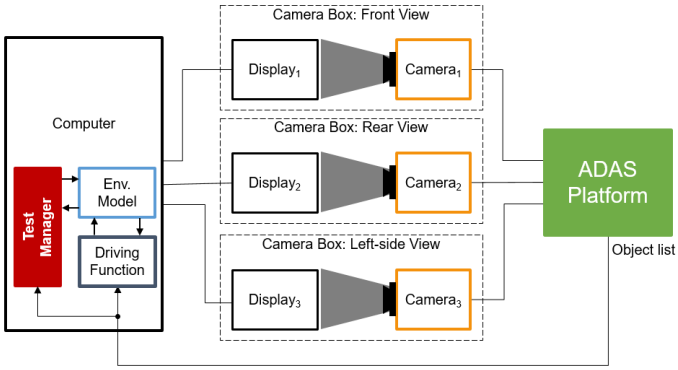

Figure 3. Overall architecture of the test setup.

The automotive camera is fixed in front of a display inside a dark box in which the ambient light is blocked – this is referred to as "camera box". This display has a resolution of 1920x1200 Pixel, a brithness level of 500 cd/m² and contrast ratio of 1000:1. The camera sensor was framed in a specific position so that (1st) it only captures the image on the display and (2nd) so that its focus is at the image on the display. This distance has to be adjusted for different FOV. This can be achieved by moving the metal rods to which the camera is attached to and screwing it again in a different position. The Fig 4 illustrates the structure of the interior of a camera box.

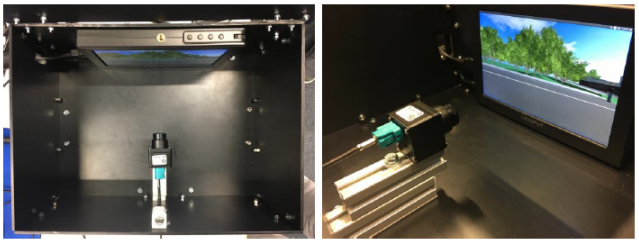

Figure 4. Inside view of the camera box, in which the automotive camera is placed in front a high resolution display. The view on the display corresponds to the left-side lateral view perspective from the perspective of the virtual car.

Three camera boxes were built to be able to take into account different perspectives of the ego vehicle: front, rear and left-side. The three displays are connected via HDMI to a computer, which has a high-performance graphic card capable of rendering the images with the highest possible quality in full-screen mode.

For each real camera, a virtual camera was created in the simulation with the same FOV, so that the specification for its virtual lens is coherent to the one in reality. The front camera was configured with a 60° FOV and placed in the front of the car at a distance of 57 mm from the bottom level of the tires. The rear camera was also configured with a 60° FOV, but placed in the back of the car at a distance of 112 mm from the the bottom level of the tires. Meanwhile, a 100° FOV was set for the side camera, which was placed on the left-side of the vehicle, under the rear view mirror at a distance of 96 mm from the bottom level of the tires. The output image from the camera observed on the ADAS Platform is illustrated in the Fig. 5. One can also note the labeling of different objects in two of the three perspectives done by the deep neural network algorithm mentioned above. The approximate position of each virtual camera is illustrated at the bottom left corner of each perspective view with a red rectangle.

The interface between the ADAS Platform and the Driving Function was built using the UNIX network stack. It works as follows: for every detection, one network packet is sent to the IP address of the computer running the Environment Model and the Driving Function. This packet is composed of a message of 3 bytes. The first byte represents the **Camera ID**, so that it is stated from what perspective a certain object was detected. The second byte contains the **Object Class ID**, that distinguishes the class of this particular object and describes it as either a car, truck, bicycle, pedestrian or traffic signs. The last byte contains the **Object ID** so that multiple objects of the same class can be distinguished when they coexist in the scene. This is referred to as the **Object list** in Fig. 3.

The interface between the Environment Model and the Driver Function is a MATLAB Simulink Library. With this, it is possible for the driving function to access the simulation variables, that means, for example, to read and override the thrust values on the breaking and acceleration pedals in case

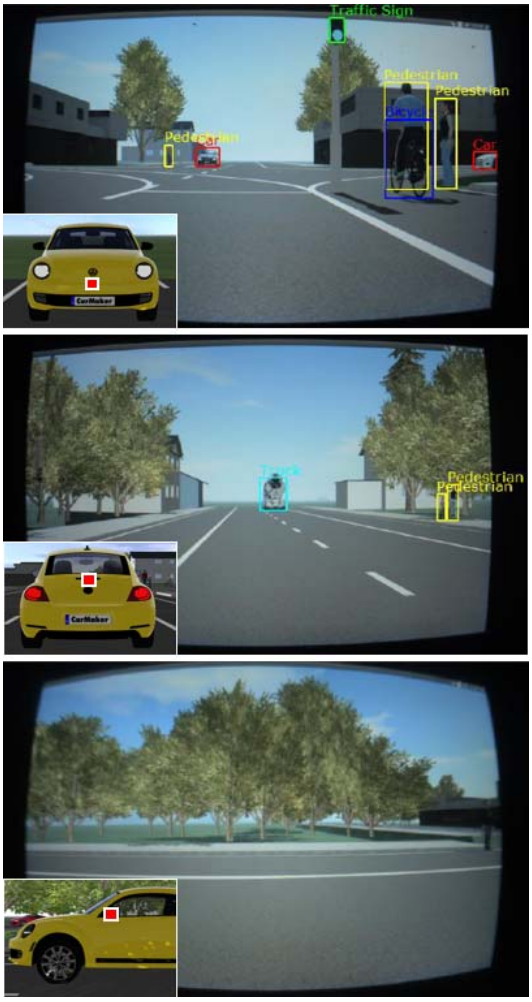

Figure 5. Camera images and approximate position of virtual camera in the ego vehicle – front (top), rear (center) and left-side (bottom) views

if the actual distance of the ego vehicle is shorter than the safety-critical predefined threshold.

The Test Manager (TM) is a flexible testing framework written in MATLAB. This:

1) has a Graphic User Interface (GUI) for automatically creating test sequences;
2) has an interface for controlling the environment model, so that it is possible to modify the simulated conditions in the test scenario;
3) is able to automate the testing process by running sequential test cases;
4) generates a test report, which contains the test results of a test sequence.

This framework was developed so that different interfaces to other environment models or videos of real test drives could be later implemented. This enables the comparison of different input data types within just one software tool.

B. Test Strategy

The test strategy is illustrated on the Fig. 6. First, for preparing the test using the TM GUI, the test scenario has to be loaded and the active cameras in the test setup have to be defined. After that, the true values for object detection and classification have to be set. That means, the number of expected objects to be detected and their respective classes has to be manually defined for each active camera. Then, the desired test sequence has to be created. The possibilities for creating different test cases are explained next on this subsection. After that, the test sequence can be initiated. For each test case, the list of detected objects is stored in the TM. The sequence runs until its last test case, when finally a test report is automatically generated.

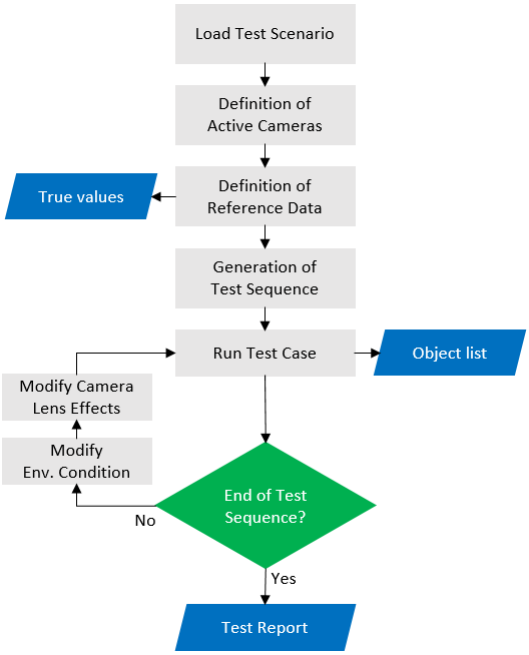

Figure 6. Test strategy

In order to evaluate this algorithm for detection of traffic participants under different conditions, the TM is capable of creating different test cases for a single scenario. The main conditions considered in this work are the environmental ones: **daytime**, **dense fog**, **deep night** and **dusk**. The Fig. 7 illustrates their reproduction in the simulation software. It is also possible to reproduce some unwanted effects on the camera lenses, such as **occlusion**, **dirt**, **image distortion** and **noise**. Using the TM GUI, it is possible to automatically create a test sequence by:

- varying the environment conditions;
- varying the lens effects for a specific camera or for every one of them;
- combining both variations above.

A single test case description is defined as one environmental condition and a single or aggregated lens effects on the virtual cameras. A method was designed and implemented for generically describing a test case, so that this is not specific to certain simulation software. The interface between TM and the Environment Model can parse a test case description and

execute the corresponding commands for applying the corresponding alterations on the environment and on the camera lens. This happens before the execution of each test case.

Figure 7. Scene variation - Environment conditions: daytime (top left corner), dense fog (top right corner), deep night (bottom left corner), dusk (bottom right corner)

The description of a test case consists of a binary array. The first 2 bits in this array represent the environment condition, since the permutation of these is equal to 4 different arrangements, what is sufficient to describe all the environment conditions: **00** for *daytime*, **01** for *fog*, **10** for *deep night* and **11** for *dusk*. Other four bits are needed for individually representing the four described effects on the camera lenses as *on* or *off*. The total length of the array depends on the number of cameras connected to the test setup, but it has a minimum of 6 elements, as shown on the the top of Fig. 8.

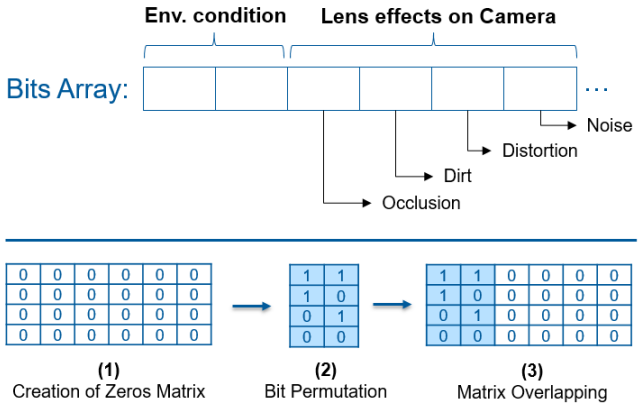

Figure 8. Process for creating a test sequence with different optical effects

For example, if a test sequence needs to be created, in which only the environment conditions will vary and only a single camera is connected to test setup, the process for creating it can be described as:

Step (1): A matrix full of zeros is created. Its size is 4x6: 4 lines for describing each test case, 2 columns containing the bits for environment condition plus 4 columns for describing whether the lens effects are turned on or off.

Step (2): Then, in this case, the permutations of 2 bits are created resulting in a matrix of size 4x2.

Step (3): Finally, the bit permutation matrix obtained in step 2 is merged with the zeros matrix created in step 1, overlapping the first two columns reserved for the environment conditions.

The result is: each line of this matrix represents a different test case, which has a unique environment condition, but no effect on the camera lens. The bottom of the Fig. 8 illustrates the described test sequence generation from Step (1) to (3). Besides generating only a few test cases like in this example, this method can also be escalated. When three cameras are connected to the test setup, it is possible to create up to 2^{14} different test cases for a single scenario.

After the created test sequence is executed, a test report is automatically generated. This contains:

1) general information about the executed test (time duration and computer specifications);
2) a confusion matrix - used as a basis for evaluating the correctness of the algorithm for object classification;
3) a table containing environment condition, camera lens effects, the true values of objects and number of detections for each test case and for each camera.

A confusion matrix (Fig. 9) is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known [17]. In this case, the true values for classification are manually set at the early beginning, before the test sequence is executed. It consist of a set of data of to-be-detected objects for each class that can be classified by the neural network in the ADAS Platform: car, pedestrian, truck, bicycle and traffic sign. This data set is assigned separately for each camera. This table reports the values of:

True Positives (TP): A TP is the matched pair of an object classified during the execution of a test case with an object of the same class that belongs to the data set of true values;

False Negatives (FN): A FN represents the missing classification of an object of any class which exists in the data set of true values;

False Positives (FP): A FP is indicated as an incorrectly or mistakenly classified object during the execution of a test case which is inexistent in the data set of true values;

True Negatives (TN): A TN of a certain class is the sum of all the other instances correctly classified as a different class.

The values for the different classes are aggregated and an overall performance is calculated by the Test Manager at the end of a test sequence. On this basis, the following correctness metrics are defined [18]:

- **Accuracy** (ACC): ratio of correct classifications, calculated by Equation 1;
- **Sensitivity** (or True Positive Rate, TPR): ratio of positives correctly classified, calculated by Equation 2;
- **Fall-out** (or False Positive Rate, FPR): ratio of expected false positives, calculated by Equation 3;
- **Precision** (or Positive Predictive Value, PPV): ratio of positive results that are true positives in fact, calculated by Equation 4;

- **Miss-Rate** (or False Negative Rate, FNR): ratio of false results that are, in fact, positive, calculated by Equation 5;

$$ACC = \frac{TP + TN}{TP + TN + FN + FP} \tag{1}$$

$$TPR = \frac{TP}{TP + FN} \tag{2}$$

$$FPR = \frac{FP}{FP + TN} \tag{3}$$

$$PPV = \frac{TP}{TP + FP} \tag{4}$$

$$FNR = \frac{FN}{FN + TP} \tag{5}$$

Total Population		True Condition		Evaluation
		Condition Positive	Condition Negative	
Predicted Condition	Predicted Condition Positive	**True Positive** ( )	**False Positive** ( )	**Precision** ( )
	Predicted Condition Negative	**False Negative** ( )	**True Negative** ( )	**Miss Rate** ( )
Evaluation		**Sensitivity** ( )	**Fall-out** ( )	**Accuracy** ( )

Figure 9. Confusion Matrix

IV. Results

After describing the test setup and the test strategy in Sections II and III, respectively, the test execution and results are going to be discussed here. The performance metrics described in this work are used to quantify the quality of the algorithm for classifying multi-classes of traffic participants in the ADAS Platform. This is used as basis for comparing its performance under different conditions against its specification described by the developer.

First of all, a test scenario representing a safety-critical situation was designed. In this test scenario, the ego vehicle drives along a straight road and, after a few meters, faces a cross-junction. At this cross-junction, the traffic flow is controlled by traffic light. At the moment when the ego vehicle reaches the crossing point, it intends to make a left turn. The light is green for its lane, allowing it to drive further. The light is red for cars coming from the left on the perpendicular road. However, a car approaching from this direction do not respect the red light and decides to drive through it. Since the ego vehicle is equipped with the left-side surrounding view camera as described in Subsection III-A, in theory, it is able to detect this car, since it finds itself entirely in the ego vehicle's field of vision. The implemented Driving Function for avoiding crash is, in ideal conditions, able to intervene by hitting the breaks and, therefore, avoiding the lateral collision. If the car is somehow not detected by the left-side camera, the cars crash. Fig. 10 illustrates the described scenario, which was also built in the environment model software, but with extra traffic participants (pedestrians, cyclists, trucks and other cars) so that the classification of different classes are also taken into account and the scenario becomes even more complex.

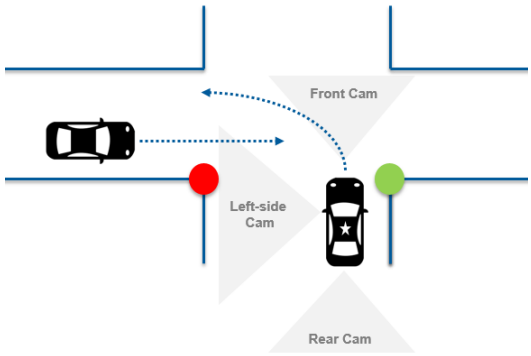

Figure 10. Test Scenario - Cross-junction

Since the ADAS Platform documentation for the deep neural network reports that it is "optimized for the daytime", the main goal is to validate this statement. Besides that, this work aims to quantify how much worse would this algorithm perform under different conditions with the previously described metrics.

Following the test strategy illustrated on the Fig. 6, the test scenario was first loaded by the TM in the environment model software. Then, the three surrounding view cameras (front, rear and left-side) were configured exactly as described on Subsection III-A and set as active in the TM. First, the simulation was executed without any modification on environment condition or camera lenses, so that the true values of each camera for all classes of traffic participants (cars, trucks, bicycles, pedestrians and traffic signs) could be visually identified and recorded.

Since the specification on the documentation only relates to the performance of the algorithm under a specific environment condition, no effects on the camera lenses were considered. Then, the test sequences were strictly generated with environment condition variations. For each, 4 different test sequences (TS) were generated: **Daytime**, **Dusk**, **Deep Night** and **Dense Fog**. Each one of these TS has 500 test cases. One test report was generated for each TS after the execution of these test cases. Therefore, a different confusion matrix was generated for every TS, in which the correctness metrics were calculated for evaluating the deep neural network algorithm under different environment conditions.

The Fig. 11 shows the *Sensitivity*, *Miss Rate* and *Accuracy* obtained for the executed TS. These were calculated as explained in the end of Subsection III-B. The obtained results confirm the specifications of the developer. These numbers show that the algorithm performs better under daytime, having a high percentage of positives correctly classified, low miss-rate and the highest accuracy of all scenarios considered in this work. Besides that, the results show, as expected,

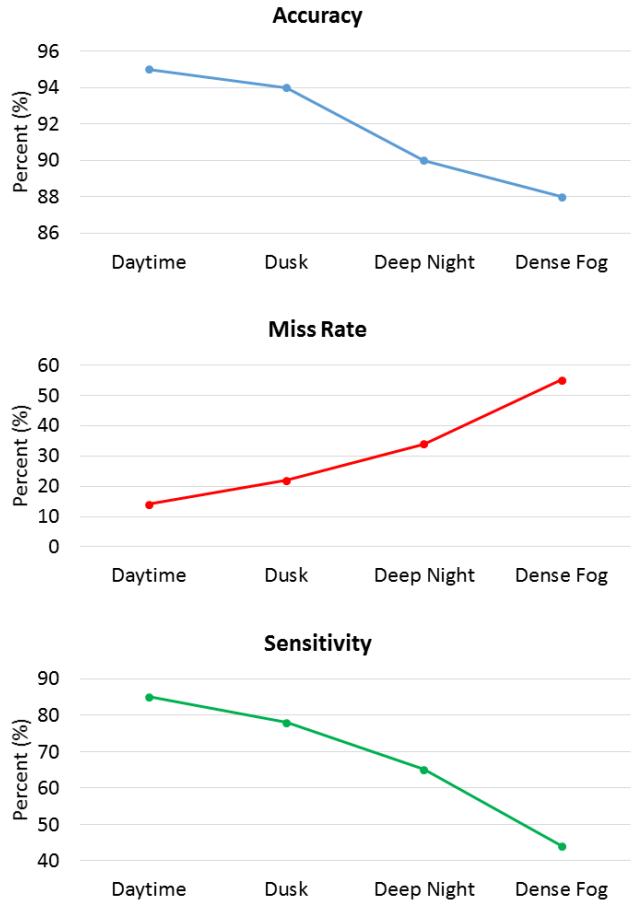

Figure 11. Test results - correctness metrics

that the probability of a correct detection decreases as the complexity of the environment increases. This is observed as the sensitivity and accuracy decrease from the TS "Daytime" to the TS "Dense Fog", which was the hardest scenario to be correctly interpreted.

V. Conclusion

The accidents involving automated cars reinforce the need of testing the environment sensors and algorithms for ADAS more intensively in the laboratory before bringing them to the streets. The current developed technology stage for testing these systems mainly focus on purely virtual approaches or on individual components. This work presented a methodology with multiple camera sensors "in-the-Loop". With this, it is possible to quantify how well an object detection algorithm performs under different environment conditions, such as night and fog or when cameras have unwanted effects on their lenses, such as dirt and distortion. This method was used to validate a deep neural network trained for the classification of multi-classes of traffic participants and implemented on an ADAS Platform available on the market. As expected, the obtained results match the specification described by its developer, showing that the developed testing method is applicable for its purpose.

This was the first step of this work, so that it still needs to be improved in a way that other sensors – such as radar and LIDAR – can be integrated and stimulated by the same simulated test scenario. Since this methodology allows the quantification of the performance of environment sensors with correctness metrics, it is also intended that such a multi-sensor setup can be used as a benchmarking platform for different Hardware and Software for the development of data fusion algorithms and automated driving functions.

Acknowledgment

This work is supported under the FH-Impuls program of the German Federal Ministry of Education and Research (BMBF) under Grant No. 13FH7I01IA.

References

[1] Hasirlioglu, S., Kamann, A., Doric, I. and Brandmeier, T., “Test methodology for rain influence on automotive surround sensors” in 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), November 2016.

[2] National Transportation Safety Board, “Collision Between a Car Operating With Automated Vehicle Control Systems and a Tractor-Semitrailer Truck Near Williston, Florida”, May 2016.

[3] Tempe Police [Online]. Available: https://twitter.com/tempepolice/status/97658509854283366. [Accessed: 09-Apr-2018].

[4] Riener, A., Wintersberger, P., Hempen, T., Brandmeier, T., Lauerer C., Hasirlioglu, S., Reway, F., “A Flexible Mixed Reality Test Environment to Improve Simulation-based Testing for Highly Automated Driving”, Aktive Sicherheit und Automatisiertes Fahren – Methodenentwicklung im Expertendialog, 2016

[5] Maurer, M., Gerdes, J. C., Lenz, B., Winner, H. “Autonomous Driving: Technical, Legal and Social Aspects”. 2016, p. 439-442

[6] ISO/DIS 26262: Road vehicles – Functional safety, International, Organization for Standardization (ISO), 2009.

[7] Winner, H., Hakuli, S., Lotz, F., Singer, C., “Handbook of Driver Assistance Systems”. 2016, p. 160

[8] Hanselmann, H., "Hardware-in-the Loop Simulation as a Standard Approach for Development, Customization, and Production Test of ECU’s", SAE International, 1993.

[9] Boot, R., Richert, J., Schutte, H., "Automated test of ECUs in a hardware-in-the-loop simulation environment", Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design, 1999.

[10] Nentwig, M., Stamminger, M., "Hardware-in-the-Loop Testing of Computer Vision Based Driver Assistance Systems", IEEE Intelligent Vehicles Symposium (IV), 2011.

[11] Feilhauer, M., Haering, J., Wyatt, S., "Current Approaches in HiL-Based ADAS Testing", SAE International", 2016.

[12] Gans, N.R., Dixon, W.E., Lind., R., Kurdila, A., "A hardware in the loop simulation platform for vision-based control of unmanned air vehicles", 2009.

[13] Pfeffer, R., Haselhoff, M., "Video Injection Methods in a Real-world Vehicle for Increasing Test Efficiency", 2016.

[14] Weiskopf, M., Wohlfahrt, C., Schmidt, A., "Integrationslösung zur Absicherung eines realen Radarsensors im Systemverbund mit der Hardware-in-the-Loop Testtechnologie", 2015.

[15] Engelhardt, M., Pfeiffer, F., Biebl, E., "A high bandwidth radar target simulator for automotive radar sensors", Radar Conference (EuRAD), 2016 European, 2016.

[16] Groh, K., Kuehbeck, T., Fleischmann, B., Schiementz, M., Chibelushi, C., "Towards a Scenario-Based Assessment Method for Highly Automated Driving Functions", 2017.

[17] Galarnyk, M., Logistic Regression using Python (scikit-learn) [Online]. Available: https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a. [Accessed: 20-Apr-2018].

[18] Fawcett, T., "An introduction to ROC analysis", 2005.