Generating Critical Driving Scenarios Using Optimization To Test Autonomous Emergency Braking System (AEBS)

Yamen Albdeiwi
dept. Faculity of Computer Science
Lund University
Lund, Sweden
mo4718al-s@student.lu.se
https://github.com/Yamen9418/ProjectInAI-AEB.git

***Abstract*—Automated driving systems are becoming increasingly complex, making it essential that they are tested thoroughly before being deployed. To ensure the safety and reliability of autonomous driving systems, testing is vital, however identifying effective test scenarios is notoriously difficult, especially for the critical scenarios that may lead to collisions or near-collisions. My research focuses on autonomous emergency braking system (AEBS) and uses optimization to generate critical driving scenarios using a multi-objective self-adapting algorithm (pilOPT) and a supplemental tool (modeFrontier). The results show that none of the 26 driving scenarios imported from Euro NCAP detected any failure. As I continued the testing through optimization using the pilOPT algorithm, a bug was found in modeFrontier as well as a faulty behavior in AEBS. A workaround was provided by the modeFrontier team after I reported the bug to them, but I was not able to test it due to the time constraints for this study. This research provides a strong case for integrating optimization with regression testing to uncover defects and misbehavior in the AEBS.**

***Index Terms*—AEB, AEBS, ADAS, LKAS, Automated driving toolbox, Autonomous Driving System, Autonomous Driving Feature, MATLAB, modeFrontier, pilOPT, regression testing, SIL, AEBTestBench.**

I. Introduction

The complexity of automated driving systems is increasing, so they must first be thoroughly tested before they are deployed. Testing autonomous driving systems for safety and reliability is vital, but identifying effective test scenarios is notoriously challenging, especially for the scenarios that could result in collisions or near-collisions. Additionally, testing critical driving scenarios is crucial to ensure that the system can handle rare situations such as Uber's fatal crash in USA 2018 [1], where the Uber's autonomous vehicle hit a cyclist which was not detected by the system. Moreover, due to the complexity of driving scenarios and the uncertainty of the operational environment, traditional requirements driven testing approaches are impeded [2]. Thus, it is necessary to identify the most critical scenarios for testing the autonomous driving systems and to uncover defects and misbehavior. As we have seen in Uber's case, it is a life or death matter.
Furthermore, testing critical driving scenarios helps to avoid relaying on substantial real-world testing or collecting real driving data at scale. Nevertheless, numerous algorithms have been developed in recent years for generating critical scenarios for testing autonomous driving systems, but no comparative studies have been conducted to determine their efficacy and efficiency. Advanced driver assistance systems (ADAS) enhance road safety through features such as adaptive cruise control, autonomous emergency braking, and lane keeping assistance. In this study, I chose autonomous emergency braking system (AEBS), as System under test (SUT).

II. Preliminaries

This section introduces the AEBS under test, the Euro NCAP test protocol prebuilt in MATLAB, as well as the modeFrontier tool.

A. The Autonomous Emergency Braking system Under Test

It is a part of MATLAB's Automated Driving Toolbox, which offers a prebuilt reference application named *Autonomous Emergency Braking System,* which is the AEBS under test in the present study. The AEB is a safety feature that could save lives by warning drivers of impending collisions and automatically apply brakes to prevent collisions or reduce the impact of collisions [3], and it demonstrates a robust approach to the controller design by a model called AEBtestbench which is a system-level simulation test bench model to explore the behavior of the controller for an AEB system. In order to achieve this, the test bench model contains the following modules: [4], as seen in Fig 6

- Sensors and Environment — Subsystem that specifies the road, actors, camera, and radar sensor used for simulation.
- Sensor Fusion and Tracking — Algorithm model that fuses vehicle detections from the camera to those from the radar sensor.
- AEB Decision Logic — Algorithm model that specifies the lateral and longitudinal decision logic that provides most important object (MIO) related information and ego vehicle reference path information to the controller.

- AEB Controller — Algorithm model that specifies the steering angle and acceleration controls.
- Vehicle Dynamics — Subsystem that specifies the dynamic model of the ego vehicle.
- Metrics Assessment — Subsystem that assesses system-level behavior.

Thereafter, a stopping time calculation approach is used in the Braking Controller subsystem to implement the *Forward Collision Warning FCW* and the *Automatic Emergency Braking AEB* control algorithms.

1) Forward Collision Warning FCW: By detecting stopped or slowly moving vehicles ahead of your vehicle, forward collision warning systems alert you to an impending collision. While driving, radar, lasers, or cameras are used to scan for possible collisions, as seen in Fig 1. The FCW algorithm used in the current study uses only camera vision and radar coverage. Whenever there is an imminent collision with a lead vehicle, FCW warns the driver, and the driver is expected to react and apply the brakes in a timely manner τ_{react}. Furthermore, in order to determine whether an FCW alert should be activated, some calculations should be conducted to determine whether the situation is critical or not, as seen in Fig 2:

In the first step, the system determines the *Stopping Time* τ_{stop}, which is the period of time between the moment the ego vehicle applies its brakes a_{brake} and the moment it comes completely to a halt. $\tau_{stop} = v_{ego}/a_{brake}$, where the v_{ego} is the ego vehicle's velocity. The total time elapsed by the ego vehicle prior to colliding with the lead vehicle can be written as follows:

$$\tau_{FCW} = \tau_{react} + \tau_{stop} = \tau_{react} + v_{ego}/a_{driver}$$

An FCW alert is triggered when the lead vehicle's time-to-collision (TTC) is less than τ_{FCW} [4].

Fig. 1. A real output image of Forward Collision Warning System FCW [5]

2) Automatic Emergency Braking AEB: If the driver has not taken any action in time, the next step for AEBS is to apply cascaded braking by activating AEB [3]. The AEB would act independently of the driver to avoid the collision or even mitigate it. The AEB algorithm runs at three different levels seen in Fig .4 as follows:

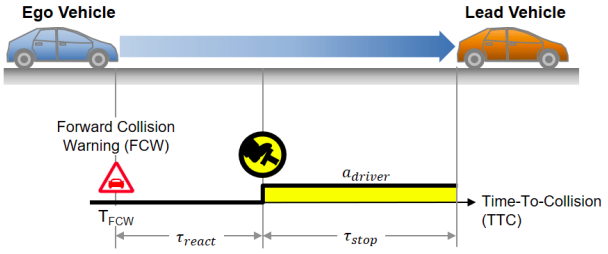

Fig. 2. The time calculation used by the FCW system [4]

- Yellow — First stage partial brake is activated.
- Orange — Second stage partial brake is activated.
- Red — Full brake is activated.

The different levels could be seen in the dashboard panel of the AEBtestbench during the simulation, Fig .3.

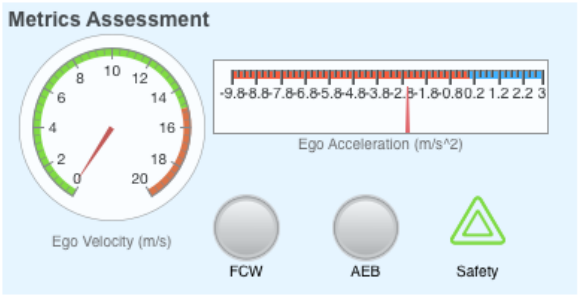

Fig. 3. The dashboard panel of AEBtestbench model

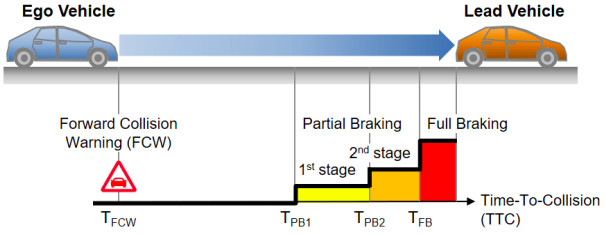

Fig. 4. Multi-stage partial braking followed by full braking [3]

B. The European New Car Assessment Programme

Euro NCAP [1] refers to the European New Car Assessment Programme, which was originally initiated 20 years ago with the intent of raising safety standards in the automobile industry. Tests of the AEBS were conducted with the help of *Driving Scenario Designer DSD*, which offers a library of prebuilt test scenarios representing Euro NCAP test protocols. The library includes different test protocols for different autonomous driving features, such as AEB, emergency lane keeping and lane keep assist systems [6]. In my study, I have adopted 26 different driving scenarios categories designed for the AEBS, of which two examples are shown in Fig .5.

[1]https://www.euroncap.com

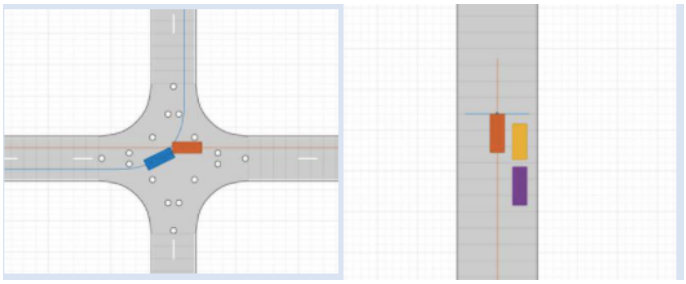

Fig. 5. Two examples of the Euro NCAP driving scenarios depicted in DSD. (a) The Global Vehicle Target GVT follows a straight-line path in the lane adjacent to the vehicle under test (VUT)'s initial position, in the opposite direction to the VUT with a constant velocity of 8.33 m/s. (b) A pedestrian (child) crosses the road from the right side and the two vehicles are stationary on the road that obstructs the visibility of pedestrian for the VUT.

C. modeFrontier Tool

modeFrontier is a software which enables automation of simulation processes and design optimization. The reason why modeFrontier is chosen in this study is that it integrates with the MATLAB node, where scripts are stored and configured, as well as it provides different optimization algorithms. In addition, files and variables can be sent to and received from other application nodes in the workflow [7].

III. Methods

A. Simulating AEBS

In this method, I have used Autonomous Emergency Braking System (AEBS), as well as a system-level simulation test bench (AEBTestBench) provided by MATLAB. My first step was to import the model inside the working directory, and open the AEBTestBench, which consists of six sub-models as stated previously, see Fig 6. The next step is setting up the model by using a function called *helperSLAEBSetup* that takes a scenario as input to create required busses and variables for simulating AEBS model.

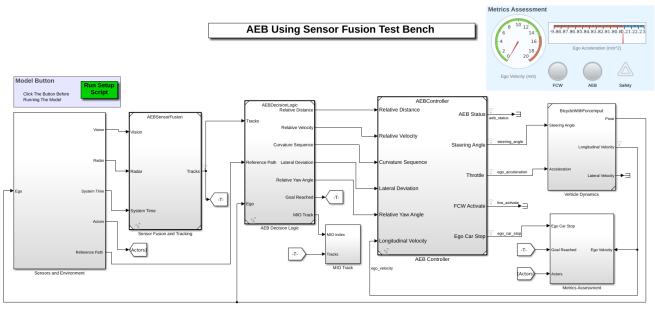

Fig. 6. AEBTestBench Model [3]

1) Dataset: The Dataset in this study is 26 important driving scenario categories imported from Euro NCAP [6]. All these scenarios are prebuilt and designed to test AEBS.

2) Test Requirments Set: Test requirements set is used in this method to describe the test conditions and verify that the AEBS model is robust [8], see Fig 7.

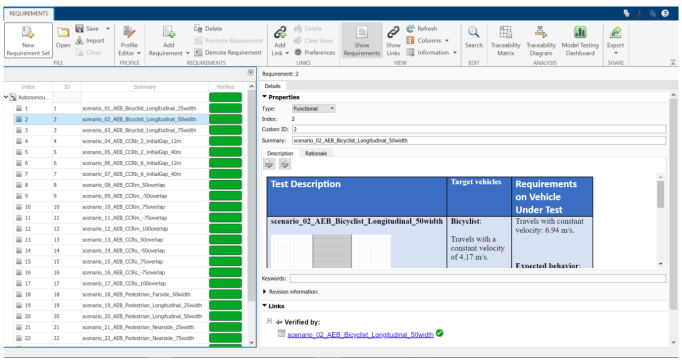

Fig. 7. The Test Requirements Set used in this study

3) Regression Testing: In regression testing, the aim is to ensure that modifications or additions to software, such as features added or modified, have not adversely affected the software features that should remain unchanged. In regression testing, various test cases are run to check for modifications in previous software versions [9]. All simulations have been automated in order to conduct regression testing of system-level functionality. In this step, MATLAB's *Test Manager* has been used to link each test case to the test requirements set. POST-LOAD is used by each test case to run the setup function with an appropriate input [8]. In addition, parallel automate testing has been used to increase the overall automate execution speed.

B. Optimizing Critical Driving Scenarios

In this method, the goal is to optimize driving scenarios to generate critical driving scenarios to test AEBS. I have used modeFrontier tool for automating the optimization process. To begin, I created a MATLAB node and then defined a MATLAB script I would run the optimization on, which I selected *Scenario23* described in Table I, seen in Fig 8.

As for the optimization inputs and outputs I/O, I chose the ego Vehicle's velocity, as well as the pedestrian's velocity with predefined upper and lower bounds as input, and TTC as output. The next step was to choose an optimization algorithm, which I chose a multi-objective self-adapting algorithm which combines the advantages of local and global search, which is called *pilOPT* provided by modeFrontier [10].

Next, I created a workflow of different nodes in modeFrontier as seen in Fig 9 where all the AEBTestBench sub-models were given as input to the MATLAB node in form of support files. Then, I defined the objective for the pilOPT algorithm, which is to minimize TTC. After that, I run the algorithm with 1000 epochs/evaluations, defined a threshold for the TTC, and plot the optimized results to find extreme critical driving scenarios.

IV. Results And Discussion

Since two different test methods were used for this study, the results are separated into two phases.

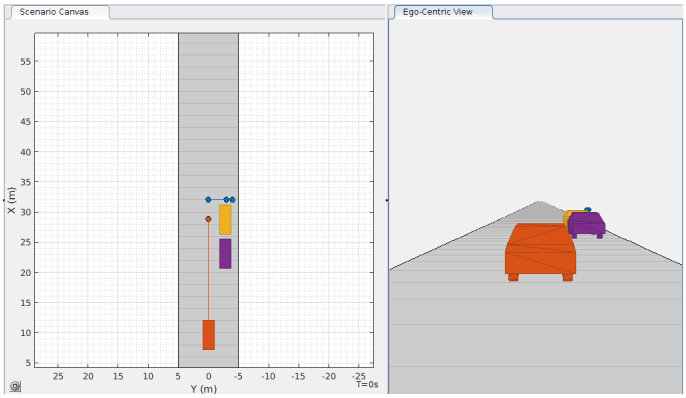

Fig. 8. An illustration of Scenario 23 used to run the optimization on

TABLE I
SCENARIO 23 DESCRIPTION

Expected Behavior	**Target Actor**		
	Child Pedestrian	***Obstructing Vehicles***	***VUT***
Forward collision warning (FCW) and triggering of Autonomous Emergency Braking (AEB) which stops the ego vehicle by applying brakes to avoid collision.	pedestrian crosses the road from the right side (near side) at a constant speed of 1.39 m/s.	The two vehicles are stationary on the road that obstructs the visibility of pedestrian for the VUT.	Travels with constant velocity: 5.55 m/s.

A. The First Phase

When applied to the 26 original Euro NCAP scenarios, AEBS worked well, preventing collisions. In each scenario, the check safety goal is achieved, the collision is mitigated, and the test requirements are met. This is summarized and can be seen in Fig 7. Regression testing of the following sub-models: *Sensor Fusion and Tracking, AEB Decision Logic, and AEB Controller* was applied by using software-in-the-loop (SIL) verification in order to quickly and cost-effectively catch bugs and improve the quality of the code [11].

However, when I ran scenario 23, which is an extremely

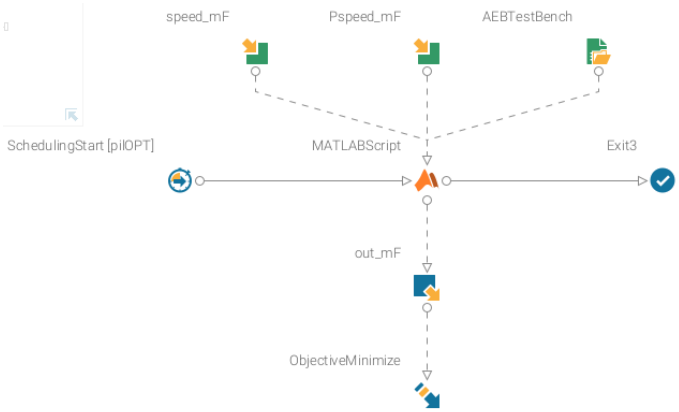

Fig. 9. The Optimization Workflow in modeFrontier

critical scenario, a few times, I got the following plots for the TTC as shown in Fig 10 and Fig 11, respectively. The two diagrams show that AEBS has a potential for misbehavior, shown in a time interval of $[1, 1.5]$ seconds, which was interesting for me to test even further during the optimization in the second phase, to uncover potential defects in the AEBS.

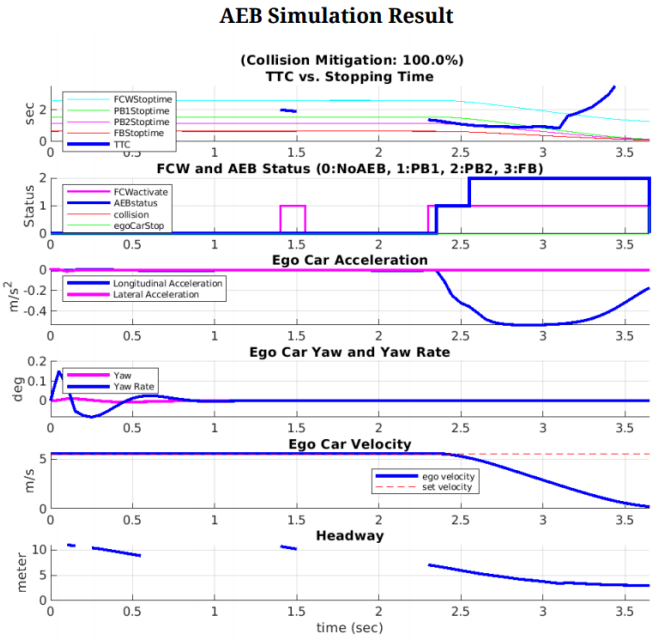

Fig. 10. Simulation Callback Plots after running Scenario 23

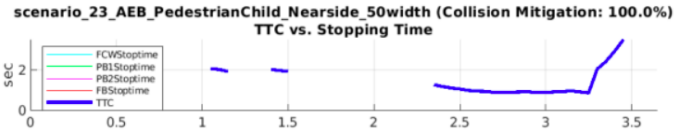

Fig. 11. TTC plot after running Scenario 23

B. The Second Phase

As part of the integration process, I have found a bug in modeFrontier which stuck during the optimization process, and specifically when the script included the simulation command *simout = sim('AEBTestBench.slx')*. I have logged the issue, reported it to modeFrontier's team, and they have confirmed it and provided a workaround using an indirect integration i.e., without employing the Matlab node of modeFrontier but instead using the EasyDriver node of modeFrontier [2]. Unfortunately, the optimization workflow could not be carried out as planned due to the bug, since for each epoch/evaluation for the pilOPT, the TTC signal should be updated by running the simulation command mentioned earlier after changing the input for the algorithm. Thus, I was not able to run the

[2]https://engineering.esteco.com/technology/simulation-process-integration-automation/#

modeFrontier with the simulation command in the script of the MATLAB's node, because modeFrontier's optimization process got stuck in an infinite loop. Once I commented out the simulation command in the script, though, the optimization worked, but the TTC signal did not get updated after each epoch.

My workaround was to run pilOPT on the input parameters, which were the velocity of both the ego vehicle and the pedestrian, and then update the code of scenario 23 manually using the evaluated dataset. This method allowed me to run the simulation and update the simulation's output, which was a new dataset containing the updated TTC signal value.
The manual optimization of the AEBS based on the possible misbehavior in scenario 23, discovered in the first phase, revealed a defect in which the collision could not be avoided, thus the ego vehicle collided with the pedestrian/child as seen in Fig 12 in which the collision mitigation was 41.9%.

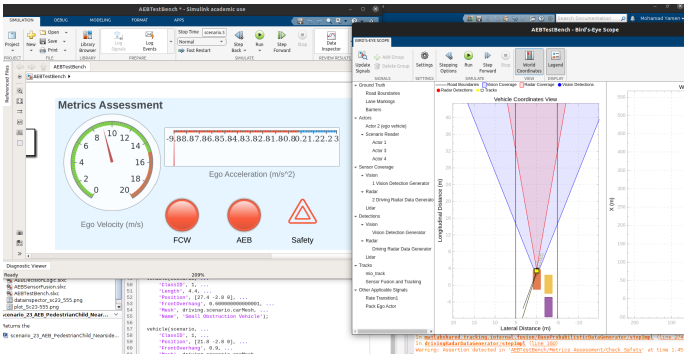

Fig. 12. The optimized Scenario 23-Ego vehicle collided with the pedestrian

By comparing the TTC values between two simulation's runs, one without optimization and one with optimization, we can observe both the tolerance and the difference as seen in Fig 13.

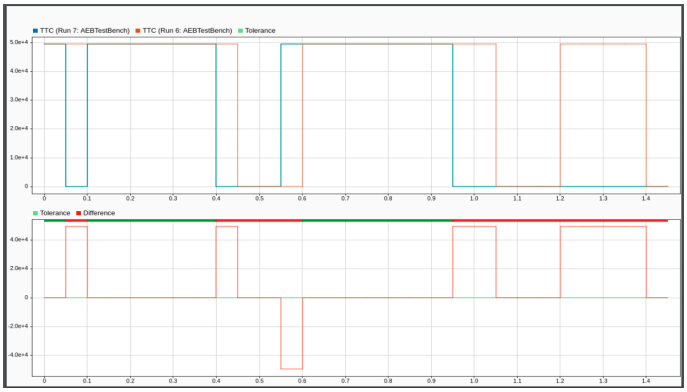

Fig. 13. The outputs of two simulation runs of the TTC are compared. The first one is indicated by a blue line, while the second one has a red line

The difference in the velocity of both the ego vehicle, and the pedestrian for two different epochs, is shown in Table II, and the generated plots for the TTC after running the optimized scenario 23 can be seen in Fig 14.

TABLE II
THE DIFFERENCE IN THE VELOCITY BETWEEN THE TWO SIMULATION'S RUNS

Scenario 23	**Input parameters**	
	Ego vehicle's Velocity	***Pedestrian's Velocity***
Epoch 0	5.55 m/s	1.39 m/s
Epoch 10	15.55 m/s	3.39 m/s

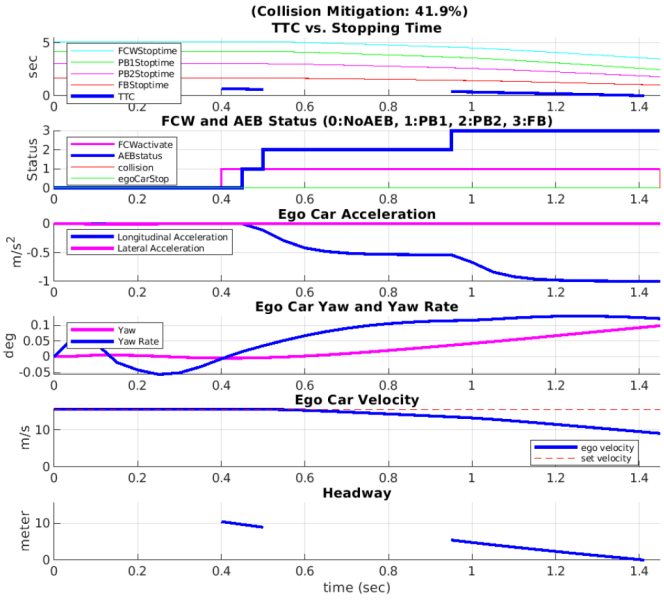

Fig. 14. Simulation Callback Plots after running the optimized scenario 23

V. LIMITATIONS AND OBSTACLES

I had different limitations and obstacles while conducting this study. The initial plan was to use Carla Simulator [3], an open-source simulator that is used for research in autonomous driving systems. Carla, however, requires very high computing power, which I do not possess. Thus, I wanted to test Apollo [4], a high-performance, flexible platform that enables rapid development, testing, and deployment of Autonomous Vehicles [12], but I had the same difficulty as Carla.

On the other hand, as for the Open Surface 3D environment provided by MATLAB, I could not use it because of the limited GPU I had. Furthermore, The modeFrontier installation on my macOS laptop did not work, so I had to transfer all project files to my Linux laptop.

VI. RELATED AND FUTURE WORK

Metamorphic testing (MT) could be used as a main approach to enhance Euro NCAP standards. MT examines the metamorphic relations (MRs) among the inputs and outputs of multiple system under test executions, rather than focusing on the correctness of each execution result of a single test case [13]. Furthermore, testing of other autonomous driving features (ADF) such as emergency lane keeping and lane

[3]https://carla.org/
[4]https://github.com/ApolloAuto/apollo

keep assist systems, could also be considered related work. In addition, a physics-based simulation platform used in the automotive industry for development of Advanced Driver Assistance Systems (ADAS) that are based on sensor technologies such as radar, laser/LiDAR, camera, and GPS, which is called PreScan [14] and provided by MATLAB, could also be considered as another testing environment.

VII. Conclusion

In this paper, I have reported on an optimization of pilOPT algorithm using modeFrontier to the AEBS on MATLAB Simulink, involving the Euro NCAP scenario-based test protocols. When tested with the original 26 Euro NCAP protocol driving scenarios, the AEBS was always able to prevent collisions. However, after applying the optimization on the velocity of both the ego vehicle and the pedestrian, to generate an extremely critical scenario, a defect was revealed in which the ego vehicle collided with the pedestrian and the AEBS could mitigate the collision with 41.9% after being 100%. In addition, a bug was detected in modeFrontier and later confirmed by the modeFrontier's team.

Based on existing testing standards and protocols for the verification and validation of ADAS, this research provides a method for generating critical driving scenarios to be used to test the AEBS or even other ADF.

Acknowledgment

This work was supported by Lund University in Sweden, as a part of a course called *Project in Computer Science*. For his assistance with modeFrontier, I wish to thank *Alexandre Mugnai*, the business development manager for Europe at the ESTECO HQ, as well as my supervisor, *Qunying Song*, for his assistance throughout the project.

References

[1] P. Kohli and A. Chadha, "Enabling pedestrian safety using computer vision techniques: A case study of the 2018 uber inc. self-driving car crash.," 2018.

[2] Q. Song, "Critical scenario identification for testing of autonomous driving systems," English, Ph.D. dissertation, Mar. 2022, ISBN: 978-91-8039-211-2.

[3] W. Hulshof, I. Knight, A. Edwards, M. Avery, and C. Grover, "Autonomous emergency braking test results," in *Proceedings of the 23rd International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, National Highway Traffic Safety Administration Washington, DC, 2013, pp. 1–13.

[4] W. Li and J. Bélanger, "An equivalent circuit method for modelling and simulation of modular multilevel converters in real-time hil test bench," *IEEE Transactions on Power Delivery*, vol. 31, no. 5, pp. 2401–2409, 2016.

[5] P. M. Forte *et al.*, "Exploring combined dark and bright field illumination to improve the detection of defects on specular surfaces," *Optics and Lasers in Engineering*, vol. 88, pp. 120–128, 2017.

[6] A. Kullgren, A. Axelsson, H. Stigson, and A. Ydenius, "Developments in car crash safety and comparisons between results from euro ncap tests and real-world crashes," in *Proceedings of the 26th International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, 2019.

[7] J. Ding, T. Yu, Y. Yang, and T. Q. Bui, "An efficient variable-node xfem for modeling multiple crack growth: A matlab object-oriented implementation," *Advances in Engineering Software*, vol. 140, p. 102 750, 2020.

[8] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements interaction management," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 132–190, 2003.

[9] W. Wong, J. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," in *Proceedings The Eighth International Symposium on Software Reliability Engineering*, 1997, pp. 264–274.

[10] S. Costanzo, Z. Xue, M. Engel, and C. H. Chuang, "Multistrategy intelligent optimization algorithm for computationally expensive cae simulations," in *NAFEMS World Congress 2015 (NWC 2015)*, NAFEMS, 2015, pp. 1–16.

[11] J. Kiesbye *et al.*, "Hardware-in-the-loop and software-in-the-loop testing of the move-ii cubesat," *Aerospace*, vol. 6, no. 12, p. 130, 2019.

[12] M. Lund and T. Jevremovic, "Enhanced geant4 monte carlo simulations of the space radiation effects on the international space station and apollo missions using high-performance computing environment," *Acta Astronautica*, vol. 165, pp. 219–228, 2019.

[13] M. Iqbal, J. C. Han, Z. Q. Zhou, and D. Towey, "Enhancing euro ncap standards with metamorphic testing for verification of advanced driver-assistance systems," in *2021 IEEE/ACM 6th International Workshop on Metamorphic Testing (MET)*, 2021, pp. 37–41.

[14] O. J. Gietelink, D. Verburg, K Labibes, and A. Oostendorp, "Pre-crash system validation with prescan and vehil," in *IEEE Intelligent Vehicles Symposium, 2004*, IEEE, 2004, pp. 913–918.