CarFASE: A Carla-based Tool for Evaluating the Effects of Faults and Attacks on Autonomous Driving Stacks

Mehdi Maleki, Ashfaq Farooqui, Behrooz Sangchoolie
Dependable Transport Systems, RISE Research Institutes of Sweden, Borås, Sweden
{mehdi.maleki, ashfaq.farooqui, behrooz.sangchoolie}@ri.se

***Abstract*—This paper presents CarFASE, an open-source carla-based fault and attack simulation engine that is used to test and evaluate the behavior of autonomous driving stacks in the presence of faults and attacks. Carla is a highly customizable and adaptable simulator for autonomous driving research. In this paper, we demonstrate the application of CarFASE by running fault injection experiments on OpenPilot, an open-source advanced driver assistance system designed to provide a suite of features such as lane keeping, adaptive cruise control, and forward collision warning to enhance the driving experience. A braking scenario is used to study the behavior of OpenPilot in the presence of brightness and salt&pepper faults. The results demonstrate the usefulness of the tool in evaluating the safety attributes of autonomous driving systems in a safe and controlled environment.**

***Index Terms*—Fault Injection, Carla, Autonomous Driving, Safety.**

I. Introduction

Safety- and security-critical automotive systems continue to be integrated into our daily lives e.g., in the form of automated driver assistance systems. These systems could be built and integrated into vehicles throughout the vehicles' development lifecycle or they could be bought as off-the-shelf devices and connected to a regular, non-autonomous vehicle to provide the driver with intelligent assistance. OpenPilot [1] is one such famous device sold by Comma AI.

OpenPilot is an open-source advanced driver assistance system (ADAS) designed to provide a suite of features such as lane keeping, adaptive cruise control, and forward collision warning to enhance the driving experience. The system uses cameras, radars, and other sensors to monitor the road and provide drivers with alerts, warnings, and automated driving functions. The software stack can be installed on a regular computer and connected to any supported vehicle via the CAN link. Furthermore, Comma AI sells its own hardware that can be attached as a plug-and-play device for any of the supported vehicles.

OpenPilot aims to make advanced driver assistance systems more accessible and affordable to consumers, and its open-source nature allows for contributions and improvements from the community. The ramifications of such devices being readily available in the market on safety can be catastrophic. Thus we need methods to validate the safety attributes of such third-party systems.

Fault injection is a well-established method used for measurement, test, and assessment of dependable computer systems in extreme stress or faulty conditions. It involves introducing faults into the system to study the system's reaction and whether it can recover from the impact of the faults. It is typically used to evaluate the dependability attributes of a system. Functional safety standards such as IEC 61508 [2] and ISO 26262 [3] recommend the use of fault injection to prove that malfunctions in electrical and/or electronic systems will not lead to violations of safety requirements. Fault injection could also be used to evaluate security properties of computer systems. In this case, the evaluation method could also be called *attack injection*. This definition is, infact, inline with the one given by Avizienis et al. [4], who consider an attack to be a special type of fault which is human made, deliberate and malicious, affecting hardware or software from external system boundaries and occurring during the operational phase.

Performing fault injection in physical systems is not feasible as it could lead to potentially dangerous situations. Fault injection could instead be done in a simulation environment as it provides several advantages compared to using the real system. Unlike the real system, the simulation can be run faster than real-time, even multiple instances in parallel, thereby speeding up the learning process. This way, dangerous collisions and unforeseen events are avoided and confined to the simulation, providing a safe testing environment. Additionally, once a simulation is obtained, the financial investment needed relates to obtaining powerful computers – which in today's world is relatively cheap. One such simulator for the simulation of driving scenarios is Carla [5].

Carla (Car Learning to Act) [5] is an open-source, highly customizable, and adaptable simulator for autonomous driving research. It is designed to support autonomous driving systems' development, training, and validation. Carla provides a realistic urban environment in which autonomous vehicles can be tested and evaluated in a controlled and repeatable manner. It includes realistic elements such as traffic lights, pedestrian crossings, vehicles, and dynamic weather conditions that can suit the researchers' specific needs. The simulator also allows for easy environmental customization, including adding new

objects and scenarios. Most importantly, a real-world car controller can be connected to the simulator to test the controller's behavior in a hardware-in-the-loop setting. Researchers and engineers widely use Carla in the autonomous driving industry and academia to study, develop and evaluate new technologies and algorithms related to autonomous driving [6]–[8].

In this paper, we present and demonstrate the usefulness of an open source tool called *CarFASE (Carla based Fault and Attack Simulation Engine)*[1] to run fault injection experiments in the simulated environment of Carla to test and evaluate autonomous driving stacks. The application of the tool is demonstrated by running fault injection experiments on OpenPilot. To this end, two different fault models are used in a braking scenario to study the behavior of OpenPilot in the presence of faults.

A. Related Work

Various approaches to assess the safety attributes of ADAS systems using simulation-based fault injection have been proposed in the literature over the past years [9]–[15]. This includes presentations on improved and new methods, tools, and experimental assessments. Thus, the literature within this area is very extensive. For this paper, we will confine our attention to papers that present tools that inject faults into the perception module of ADAS systems and can work in conjunction with either Carla, like simulators, OpenPilot, or both.

AVFI [11] proposes a tool for end-to-end resilience assessment of autonomous vehicles using fault injection. The tool interfaces with Carla and can inject faults into the Imitation learning-based [16] driving already existing in Carla. The paper presents a high-level overview of the injection of data, hardware, and timing faults. However, detailed experimental results and discussions on the validity of the experiments are missing to the best of our knowledge.

DriveFI [17] and Kayotee [12] are two other tools by the same authors as AVFI that extend the initial ideas of AVFI to allow the application of more sophisticated fault injection techniques. DriveFI integrates domain knowledge and Bayesian networks to generate faults. The results are verified on two industry-grade autonomous driving software stacks. Kayotee, on the other hand, is much more sophisticated and is capable of injecting faults into the closed-loop environment's hardware and software components. While both these tools are demonstrated to be quite advanced, they are not publicly available.

Rubaiyat et al. [18] provide a platform for experimental analysis of Openpilot in the presence of faults. To analyze the effects of faults with OpenPilot, they perform a golden run by recording the control outputs of OpenPilot for a given video stream. Then they rerun the same experiment by injecting faults into the video stream. The resulting control outputs are compared to assess the impact of faults on the driving profile. Such an approach helps assess the behavior of OpenPilot but does not consider environmental changes due to changes in the control output. In comparison, CarFASE overcomes this issue by performing the experiments in a simulated environment.

II. CarFASE: Carla-based Fault and Attack Simulation Engine

CarFASE is developed to automate the evaluation of safety attributes of autonomous driving stacks by injecting faults and evaluating the impact in a simulated environment. A core feature of CarFASE is that it provides a straightforward way to automate the experimentation process. This way the users can run comprehensive fault injection experiments automatically. In this paper, we demonstrate the use of CarFASE by studying the impact of perception faults on OpenPilot.

Fig. 1 shows a high-level architecture of CarFASE along with its connection to Carla and OpenPilot. As seen here, CarFASE consists of three main components, *scenario configurator*, *fault library*, and *campaign configurator*. The scenario configurator is responsible for creating a scenario that consists of a number of vehicles, a certain weather condition, the choice of maps, and the trajectory of vehicles. The fault library contains implementations of different fault models. Finally, the campaign configurator is responsible for applying the chosen fault parameters. These parameters are defined in a separate user-defined configuration file. Thus, the user needs to configure the experimentation setup in one file that provides the source for all experiments.

In order to run the experiments, CarFASE interacts with Carla and OpenPilot. In the first step, the scenario configurator sets up the required scenario based on the test requirement. With the scenario in place, CarFASE then starts OpenPilot. In order to connect a vehicle in Carla with the OpenPilot, a *Bridge* is employed. The Bridge translates and allows the exchange of data between Carla and OpenPilot and is officially provided by the developers of OpenPilot. Furthermore, simulation outputs from all these components are collected into a database which can then be used to analyze the experiments.

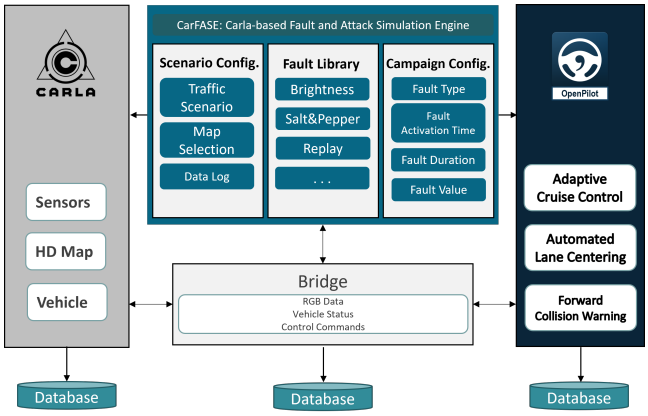


Fig. 1: Architecture of CarFASE showing its connection with Carla and OpenPilot.

[1] *https://github.com/RISE-Dependable-Transport-Systems/CarFASE*

A. Implemented Fault Models

Perception sensors play an important role in the precise functioning of AD stacks. Several studies have already addressed the criticality of faults and attacks on the perception sensors [11], [19], [20]. Authors in [19] provide an in-depth analysis of the failure modes of automated vehicle's camera. Using their work as an inspiration, we demonstrate the usefulness of the CarFASE by implementing fault models which target the RGB camera data of the OpenPilot and study the safety implications of these faults on the vehicle behavior. To this point, two fault models have been implemented and integrated into CarFASE. These models are explained in this section.

1) Brightness Fault: This fault model represents the impact of brighter and darker environments on the RGB camera used on automated vehicles. These environments can emerge due to the angle of sunlight [21], overlighting of the road during night time [22], the shadow of mountain roads, or failures in the camera [19]. The produced image under the above-mentioned circumstances could be either darker or brighter than a regular image. We model the brightness fault using the `PIL.ImageEnhance.Brightness` function provided by Python Imaging Library [23]. Moreover, we model different amounts of brightness using a parameter called *Brightness Factor* (BF).

2) Salt&Pepper Fault: Digital images can be corrupted due to the impulse noise caused during transmission in a noisy channel or stored in a faulty location [24], [25]. Salt&Pepper noise and random-valued noise are two common types of impulse noise [19], [24], [25]. In this paper, we study the impact of *Salt&Pepper* noise. We also use this fault to model the impact of noise from extreme weather events such as salt and sand storms in coastal areas. We implement this fault using a Python code where we define a parameter called *noise percentage* to introduce the amount of noise. Moreover, using this parameter we set the defined percentage of the pixels to one (i.e., white to represent salt) and zero (i.e., black to represent pepper) to model the different amounts of image noise.

III. Experimental Setup

A. Traffic Scenario

To study the safety of OpenPilot to faults, we use a braking traffic scenario. The selection of this scenario is inspired by NHTSA pre-crash scenario topology [26], where traffic scenarios prior to the collision are investigated. The scenario consists of two vehicles, an ego and a lead vehicle, driving on a highway road (see Fig. 2), where the lead vehicle starts to brake at a certain time to have a safe stop. This scenario aims to create a suitable condition to evaluate Adaptive Cruise Control (ACC), and Automated Lane Centering (ALC) functionalities of OpenPilot. In other words, the ego vehicle is our target vehicle which is controlled by OpenPilot, and, the lead vehicle drives with a predefined behavior implemented in Carla.

Vehicles equipped with OpenPilot can use ACC and ALC functionalities at different driving speeds. To investigate the impact of driving speed on the resiliency of these functionalities we consider two different braking scenarios. In these scenarios, the maximum cruise speed for the ego vehicle is set to 40 km/h (scenario 1) and 70 km/h (scenario 2). Accordingly, the trajectory of the lead vehicle is adjusted, allowing the ego vehicle to drive around the defined maximum cruise speed as shown in Fig. 3 and Fig. 4.

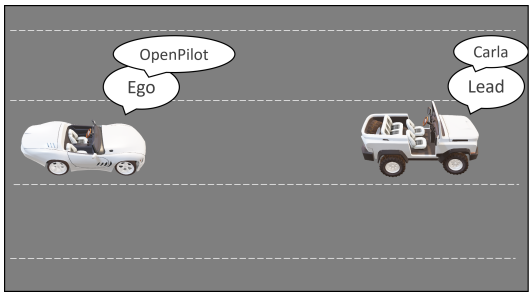


Fig. 2: Traffic scenario on a highway road.

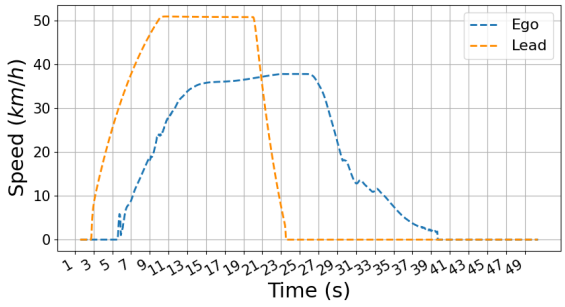


Fig. 3: Speed profiles of the ego and lead vehicles when the maximum cruise speed is set to 40 km/h.

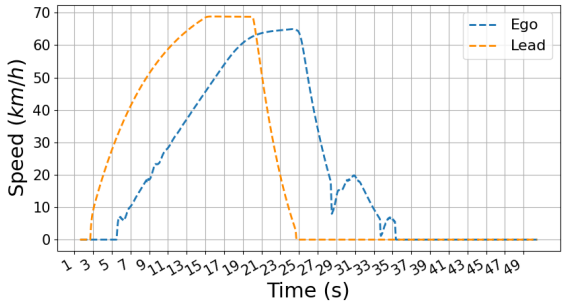


Fig. 4: Speed profiles of the ego and lead vehicles when the maximum cruise speed is set to 70 km/h.

B. Campaign Configuration

We configure the fault injection campaigns considering the traffic scenario and fault models used. Table I provides details

TABLE I: Parameters & values used to setup the fault injection campaigns.

Fault/Attack type	Implementation	Selected valueRange	Selected activationTimes	Selected durationTimes
Brightness	We use brightness factor parameter implemented in the PIL Python Imaging Library [23] (PIL.ImageEnhance.Brightness).	0 to 7.5 with 0.3 steps	20.0s to 30.0s with 0.5s steps	1s to 10s with 1s steps
Salt&Pepper	We use noise percentage parameter implemented in the function to introduce amount of the noise	1% to 10% with 1% steps		

about parameters and values used to configure fault injection campaigns.

For fault activation, we select the time period between 20 s and 30 s with a step of 0.5 s to initiate a fault. This period is selected as the lead vehicle starts to brake from time 20 s, and at time 30 s, the speed of the ego vehicle becomes relatively low (i.e., $\approx$ 12 km/h) as a result of the braking process. We keep the fault active for a range of duration from 1 s to 10 s with a step of 1 s. Here, fault duration refers to the time between the start and the end of a fault. Moreover, this duration is selected as it allows us to evaluate the system from the time that the braking process starts (i.e., time $\approx$ 20 s) until a complete stop (i.e., time $\approx$ 40 s) (see Fig. 3 and Fig. 4).

For the *brightness factor* (BF), we select a value range of 0 to 7.5 with a step of 0.3. Note that, the BF is a coefficient therefore it is unitless. In the selected value range, 0 produces a completely black image, whereas, 7.5 produces a highly brightened image. We have in fact conducted experiments for values greater than 7.5 and observed the same result classification as when 7.5 is selected. Given the values selected and presented in Table I, the total number of experiments for this fault injection campaign becomes 5460 (i.e., 21 *fault activation times**10 *fault duration**26 *BF values*).

When injecting $Salt\&Pepper$ fault, we introduce *noise* values from 1% to 10% (i.e., percentage of pixels affected) with a step of 1%; this results in 10 different noise values. Similarly, here we conducted experiments for values lower than 1% and higher than 10%, where, we observed the same result classification as when 1% and 10% were selected, respectively. Given the values selected and presented in Table I, the total number of experiments for $Salt\&Pepper$ fault injection campaign becomes 2100 (i.e., 21 *fault activation times**10 *fault duration**10 *noise values*).

C. Result Classification

We use lane-invasion and collision incidents to measure the impact of the injected faults on the ACC and ALC functionalities of the OpenPilot. To this end, two sensors are mounted on the ego vehicle to collect the lane-invasions and collision incidents. The outcome of each fault injection experiment is classified into one of the three classes presented here *(i) Negligible:* the injected fault results in neither a lane invasion nor a collision incident, *(ii) Benign:* the injected fault causes a lane-invasion incident, without causing a collision incident. We classify these cases benign as there might be a vehicle on the adjacent lane that collides with this vehicle, or the vehicle might go outside the highway and collide into the barriers on the side of the road, and *(iii) Severe:* the injected fault causes a collision incident, irrespective of whether or not a lane-invasion has occurred.

TABLE II: Fault injection results.

Campaign	Max Speed	Negligible	Benign	Severe	Total
Brightness Fault					
1	40 (km/h)	3622 (66.3%)	1271 (23.3%)	567 (10.4%)	5460
2	70 (km/h)	3411 (62.5%)	751 (13.7%)	1298 (23.8%)	5460
Salt&Pepper Fault					
3	40 (km/h)	1333 (63.5%)	0 (0%)	767 (36.5%)	2100
4	70 (km/h)	979 (46.6%)	0 (0%)	1121 (53.4%)	2100

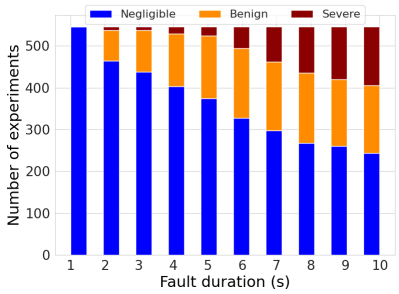
IV. Result and Analysis

In this section, we present the results of the fault injection campaigns. We used the metrics and classification categories described in §III-C to group the outcome of the experiments and summarized them in Table II. Note that for campaigns 1 and 3, we use braking scenario 1 (see §III-A), which is where the maximum cruise speed for the ego vehicle is 40 km/h. For campaigns 2 and 4, on the other hand, scenario 2 is used which is where the maximum cruise speed for the ego vehicle is 70 km/h.

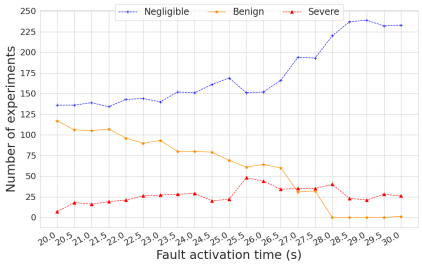
A. Fault Injection Results for the Brightness Campaigns

Table II shows that out of the 5460 experiments conducted in campaign 1, 10.4% resulted in severe, 23.3% benign, and 66.3% negligible outcomes. Note that in this campaign, the faults are injected into the ego vehicle having a maximum cruise speed of 40 km/h. The table also shows that out of the 5460 experiments conducted in campaign 2, 23.8%, 13.7%, and 62.5% resulted in severe, benign, and negligible, respectively.

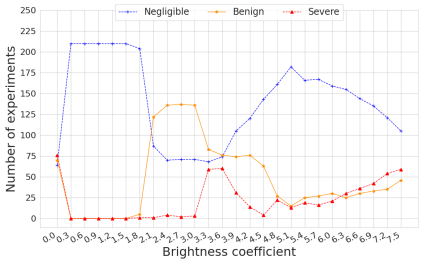
The number of severe outcomes obtained for campaign 2, where the maximum cruise speed of the ego vehicle is 70 km/h, is more than two times higher than the one observed for campaign 1. This is a clear indication of the significant impact of high vehicle's speed in causing crashes in between vehicles. In the following paragraphs, we present and discuss three additional factors that play an important role in the severity of the obtained outcomes. These factors are duration of a brightness fault, the time in which the fault is activated, and the value of the fault.



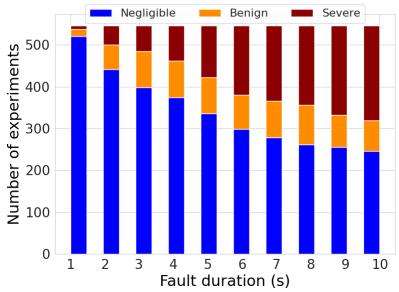
(a) Impact of *fault duration* scenario 1.



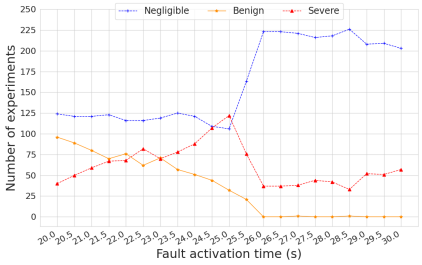
(b) Impact of *fault activation time* scenario 1.



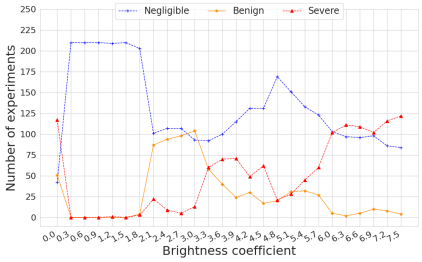
(c) Impact of *brightness coefficient* scenario 1.



(d) Impact of *fault duration* scenario 2.



(e) Impact of *fault activation time* scenario 2.



(f) Impact of *brightness coefficient* scenario 2.

Fig. 5: Classification of the results obtained for the brightness campaigns. Note that, in scenarios 1 and 2, the maximum ego vehicle's cruise speed are 40 km/h and 70 km/h, respectively.

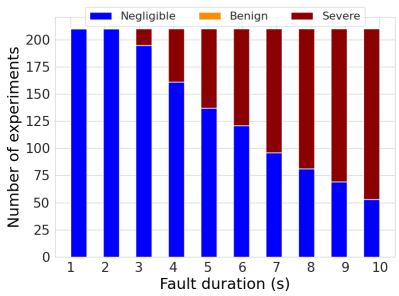
The impact of fault duration on the outcome of the experiments on scenarios 1 and 2 is shown in Fig. 5a and Fig. 5d, respectively. The figures show that for both scenarios, exposing the vehicle to the faults for a longer period of time result in higher number of severe outcomes. Note that we also expected to draw this conclusion as by exposing the camera images to the brightness faults for a longer period of time, we also increase the period of time in which the ACC controller is consuming these faulty images. This in turn directly affects the driving experience, causing a higher number of crashes.

Fig. 5b and Fig. 5e show the influence of the fault activation time on the outcome of the experiments. To present the figures, we divide the results obtained into two sections according to when the faults have been activated: (i) those activated in between time 20 s and $\approx$ 25 s, and (ii) those activated in between time $\approx$ 25 s and 30 s. The figures show that in the first section, later activation times result in higher number of severe outcomes (this is more prominent in scenario 2 which has higher driving speed). In the second section, however, there is little variation in between the severe outcomes obtained for the majority of the activation times. The figure shows that for both scenarios, the maximum number of severe cases has been obtained for when the faults are activated at about time 25 s. This could be explained by taking a look at Fig. 4, which shows that at time $\approx$ 25 s, the ego vehicle has its highest speed while the lead vehicle has reached a complete stop. Therefore, activating a fault at time 25 s is more likely to cause a crash compared to when it is activated after this point as this also means that the ego vehicle has already started to reduce its speed (see Fig. 4).

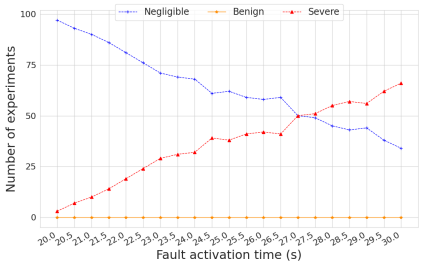
Fig. 5c and Fig. 5f demonstrate the influence of brightness factor (BF) value on the experiment outcome. The figures show that when a fault has turned an image into almost a complete dark image (i.e., when the BF value is 0), a high number of severe outcomes are observed. This is also the case for when the faulty image is too bright (i.e., when the BF value is 7.5). In fact the figures show that with every increase of BF value from around 5.0 to 7.5, a higher number of severe outcomes is observed. The figure also shows that BF values which are lower than $\approx$ 3.0 and higher than 0.0 cause very little to no severe outcomes. In these cases, the images are just slightly darker or brighter than the original images. The variation in the severe outcomes obtained for BF values that are higher than $\approx$ 3.0 and lower than 5.0 worth further investigation and will be part of our focus in the future.

B. Fault Injection Results for the Salt&Pepper Campaigns

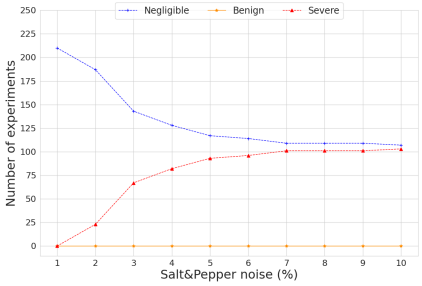
When it comes to the Salt&Pepper fault model, we have conducted 2100 experiments for each of the campaigns. Table II shows that when using scenario 1, 36.5%, 0%, and 63.5% of the experiments resulted in severe, benign, and negligible outcomes, respectively. In scenario 2, however, these numbers were 53.4%, 0%, and 46.6%. The results show that when using this fault model, we did not observe any benign outcomes. In fact, after further investigations, we observed that Salt&Pepper faults do not have any influence on the functionality of ALC (Automated Lane Centering) in the OpenPilot. This is an



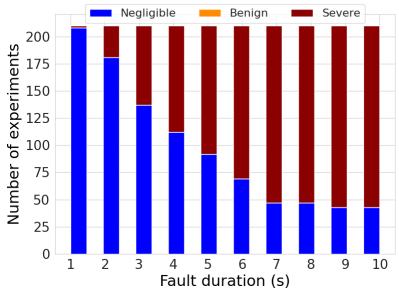
(a) Impact of *fault duration* scenario 1.



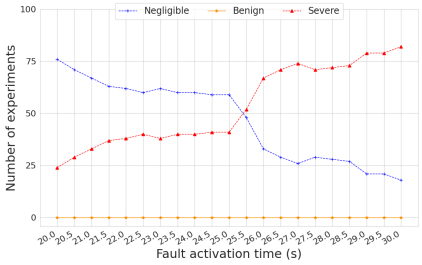
(b) Impact of *fault activation time* scenario 1.



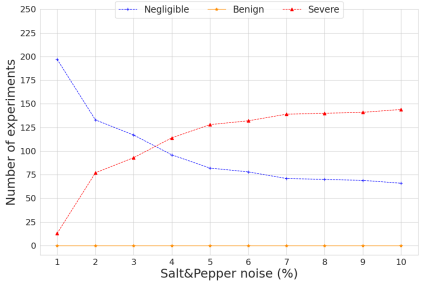
(c) Impact of *Salt&Pepper noise* scenario 1.



(d) Impact of *fault duration* scenario 2.



(e) Impact of *fault activation time* scenario 2.



(f) Impact of *Salt&Pepper noise* scenario 2.

Fig. 6: Classification of the results obtained for campaigns 3 and 4 (Salt&Pepper fault) for two traffic scenarios defined in III-A.

interesting observation that we plan on investigate further in the future.

Table II also shows that similar to the results obtained for the brightness campaigns, a higher percentage of severe outcomes is observed for scenario 2 compared to scenario 1. In fact, for the Salt&Pepper campaigns, the severe outcomes obtained for scenario 2 is around 17 percentage points higher than that obtained for scenario 1. This is yet another clear indication of the significant impact of high vehicle's speed in causing crashes in between vehicles. In the remaining of this section, we present and discuss the impact of duration of a Salt&Pepper fault, the time in which the fault is activated, and the value of the fault in the severity of the obtained outcomes.

Fig. 6a and Fig. 6d illustrate the impact of fault duration on the outcome of the experiments. Fig. 6a shows that except for when the fault duration is only 1 s and 2 s, exposing the vehicles to the faults for a longer period of time result in higher number of severe outcomes. Fig. 6d, however, shows that the increase in the duration of the faults result in higher number of severe outcomes only for fault up to around 7 s. The figure show that the variations in the severe outcomes obtained after exposing the vehicles to more than 7 s of fault is insignificant. The conclusions drawn from these two figures are slightly different than those drawn for the brightness campaigns. This shows that the impact of the fault duration on violating safety requirements varies depending on the scenario as well as the type of fault under focus.

Fig. 6b and Fig. 6e show the influence of the fault activation time on the outcome of the experiments. The figures show that regardless of the scenario, delaying the activation of the fault result in obtaining a higher number of severe outcomes. This shows that as the ego vehicle approaches the lead vehicle, there is a high chance that the injected Salt&Pepper fault will lead to a severe impact.

Fig. 6c and Fig. 6f demonstrate the impact of the noise used to model Salt&Pepper faults on the fault injection outcomes. The figures show that when the noise value is 1%, the number of severe outcomes is significantly low. As the noise value increases, we also observe a higher number of severe outcomes. However, the increase of the noise to values higher than 5% does not result in significant variations in the severe outcome obtained.

V. Discussion and Future Work

CarFASE is developed to evaluate the safety of AD stacks through the use of fault and attack injection in an efficient way in terms of time, cost, and covering broader test cases. The tool offers an exhaustive experimentation possibility to test an AD stack under a variety of test setups such as different fault activation times, fault duration times, and fault values as well as desired traffic scenarios based on the test requirements.

For this study, we integrated CarFASE into OpenPilot AD stack. However, our intention is to make it compatible with other AD stacks such as Autoware and Apollo which offer a high level of automation with several sensors. Moreover, the two fault models presented in this paper are not the only ones implemented and integrated into CarFASE fault library. In fact, we have integrated an attack model into this library

that models replaying of communication messages. We have used this model and performed attacks on the CAN (Control Area Networks) messages in OpenPilot. The results of these experiments are under analysis, which is why they have not been included in this paper and will be presented as part of our future work. Moreover, we plan to extend the fault library by adding Gaussian and Poisson fault models, and attack models such as delay attack.

A note to those individuals who are interested in using CarFASE, there is currently no user interface to configure the traffic scenario or test campaign. The configurations would need to be prepared manually from the relevant scripts explained in §II. Moreover, keep in mind that similar to any other simulation-based fault and attack injector, the usefulness and accuracy of the fault injection results obtained is tightly connected to the representativeness of the AD stack under test.

Although limitations of OpenPilot are clearly pointed out by its developers and it is suggested to deploy it in a specific area, it is not guaranteed that it will function flawlessly. Therefore, CarFASE could be used to evaluate the resiliency of OpenPilot to faults and attacks. In fact, the results obtained from the fault injection campaigns reveal vulnerabilities of OpenPilot to faults in the RGB camera.

VI. Conclusion

In this paper, we presented CarFASE, a carla-based tool developed to evaluate the effects of faults and attacks on autonomous driving (AD) stacks. We used OpenPilot as the AD stack and introduced faults into RGB camera images to evaluate the safety implication of these faults on OpenPilot. To show the usefulness of CarFASE, we run experiments using two fault models, namely, brightness and Salt&Pepper. The faults are injected in a braking scenario, which is used to challenge the functionality of the OpenPilot under a faulty RGB camera when the vehicle in front starts to brake. Moreover, to investigate the impact of driving speed on the fault injection results, we considered two maximum cruise speeds (i.e., 40 km/h and 70 km/h) for the target vehicle.

In total, we conducted four campaigns comprising 15120 experiments. The outcome of experiments was classified using lane-invasion and collision incidents recorded for each experiment. Moreover, the impact of injected faults was analyzed by looking into the fault duration, fault activation time, and value of the injected fault. The obtained results revealed the vulnerability of the OpenPilot to RGB camera faults, causing lane-invasions and collisions. It also showed that driving at a higher speed increases the likelihood of collision incidents when a fault occurs.

Acknowledgment

This work was supported by VALU3S project, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.

References

[1] OpenPilot. https://comma.ai/openpilot. (accessed: 03.04.2023).

[2] I. E. Commission, ""IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems"," 2010. [Online]. Available: https://webstore.iec.ch/publication/5515

[3] ISO, "ISO 26262:2018 Road vehicles – Functional safety," 2018.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[5] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[6] C. Gómez-Huélamo, J. Del Egido, L. M. Bergasa, R. Barea, E. López-Guillén, F. Arango, J. Araluce, and J. López, "Train here, drive there: Simulating real-world use cases with fully-autonomous driving architecture in CARLA simulator," in *Advances in physical agents II*, L. M. Bergasa, M. Ocaña, R. Barea, E. López-Guillén, and P. Revenga, Eds. Cham: Springer International Publishing, 2021, p. 44–59.

[7] M. Hofbauer, C. B. Kuhn, G. Petrovic, and E. Steinbach, "TELE-CARLA: An Open Source Extension of the CARLA Simulator for Teleoperated Driving Research Using Off-the-Shelf Components," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 335–340.

[8] E. Zapridou, E. Bartocci, and P. Katsaros, "Runtime verification of autonomous driving systems in CARLA," in *Runtime verification*, J. Deshmukh and D. Ničković, Eds. Cham: Springer International Publishing, 2020, p. 172–183.

[9] M. Maleki and B. Sangchoolie, "Simulation-based Fault Injection in Advanced Driver Assistance Systems Modelled in SUMO," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2021, pp. 70–71.

[10] M. Malik, M. Maleki, P. Folkesson, B. Sangchoolie, and J. Karlsson, "ComFASE: A Tool for Evaluating the Effects of V2V Communication Faults and Attacks on Automated Vehicles," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 185–192.

[11] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer, "AVFI: Fault injection for autonomous vehicles," in *2018 48th annual ieee/ifip international conference on dependable systems and networks workshops (dsn-w)*. IEEE, 2018, pp. 55–56.

[12] S. Jha, T. Tsai, S. Hari, M. Sullivan, Z. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Kayotee: A fault injection-based system to assess the safety and reliability of autonomous vehicles to faults and errors," *arXiv preprint arXiv:1907.01024*, 2019.

[13] M. Maleki and B. Sangchoolie, "SUFI: A Simulation-based Fault Injection Tool for Safety Evaluation of Advanced Driver Assistance Systems Modelled in SUMO," in *2021 17th European Dependable Computing Conference (EDCC)*, 2021, pp. 45–52.

[14] J. L. C. Hoffmann, L. P. Horstmann, and A. A. Fröhlich, "Integrating Autonomous Vehicle Simulation Tools using SmartData," in *2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2022, pp. 1–8.

[15] T. Munaro and I. Muntean, "Early Assessment of System-Level Safety Mechanisms through Co-Simulation-based Fault Injection," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1703–1708.

[16] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end Driving via Conditional Imitation Learning," no. arXiv:1710.02410, Mar 2018, arXiv:1710.02410 [cs]. [Online]. Available: http://arxiv.org/abs/1710.02410

[17] S. Jha, S. Banerjee, T. Tsai, S. K. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2019, pp. 112–124.

[18] A. H. M. Rubaiyat, Y. Qin, and H. Alemzadeh, "Experimental Resilience Assessment of an Open-Source Driving Agent," in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, dec 2018.

[19] A. Ceccarelli and F. Secci, "RGB cameras failures and their effects in autonomous driving applications," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[20] J. Liu and J.-M. Park, ""Seeing is Not Always Believing": Detecting Perception Error Attacks Against Autonomous Vehicles," *IEEE Trans-*

actions on Dependable and Secure Computing, vol. 18, no. 5, pp. 2209–2223, 2021.

[21] K. Yoneda, N. Ichihara, H. Kawanishi, T. Okuno, L. Cao, and N. Suganuma, "Sun-Glare region recognition using Visual explanations for Traffic light detection," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021, pp. 1464–1469.

[22] H. Fleyeh, "Color detection and segmentation for road and traffic signs," in *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, vol. 2, 2004, pp. 809–814.

[23] Python Imaging Library. https://pypi.org/project/Pillow/. (accessed: 03.04.2023).

[24] R. H. Chan, C.-W. Ho, and M. Nikolova, "Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization," *IEEE Transactions on image processing*, vol. 14, no. 10, pp. 1479–1485, 2005.

[25] J. A.-a. A. Sharadqh, B. Ayyoub, Z. Alqadi, and J. Al-azzeh, "Experimental investigation of method used to remove salt and pepper noise from digital color image," *International Journal of Research in Advanced Engineering and Technology*, vol. 5, no. 1, pp. 23–31, 2019.

[26] Pre-Crash Scenario Typology for Crash Avoidance Research. https://www.nhtsa.gov/sites/nhtsa.gov/files/pre-crash_scenario_typology-final_pdf_version_5-2-07.pdf. (accessed: 07.03.2023).