Efficient and Effective Generation of Test Cases for Pedestrian Detection - Search-based Software Testing of Baidu Apollo in SVL

Citation for the original published paper (version of record):
Ebadi, H., Moghadam, M., Borg, M. et al (2021). Efficient and Effective Generation of Test Cases for Pedestrian Detection - Search-based
Software Testing of Baidu Apollo in SVL. Proceedings - 3rd IEEE International Conference on Artificial Intelligence Testing, AITest 2021: 103-110.
http://dx.doi.org/10.1109/AITEST52744.2021.00030

Efficient and Effective Generation of Test Cases for Pedestrian Detection – Search-based Software Testing of Baidu Apollo in SVL

Hamid Ebadi*, Mahshid Helali Moghadam†, Markus Borg†§, Gregory Gay‡, Afonso Fontes‡ and Kasper Socha§
*Infotiv AB, Sweden
†RISE Research Institutes of Sweden, Sweden
‡Chalmers and the University of Gothenburg, Sweden
§Lund University, Sweden

***Abstract*—With the growing capabilities of autonomous vehicles, there is a higher demand for sophisticated and pragmatic quality assurance approaches for machine learning-enabled systems in the automotive AI context. The use of simulation-based prototyping platforms provides the possibility for early-stage testing, enabling inexpensive testing and the ability to capture critical corner-case test scenarios. Simulation-based testing properly complements conventional on-road testing. However, due to the large space of test input parameters in these systems, the efficient generation of effective test scenarios leading to the unveiling of failures is a challenge.**

This paper presents a study on testing pedestrian detection and emergency braking system of the Baidu Apollo autonomous driving platform within the SVL simulator. We propose an evolutionary automated test generation technique that generates failure-revealing scenarios for Apollo in the SVL environment. Our approach models the input space using a generic and flexible data structure and benefits a multi-criteria safety-based heuristic for the objective function targeted for optimization. This paper presents the results of our proposed test generation technique in the 2021 IEEE Autonomous Driving AI Test Challenge. In order to demonstrate the efficiency and effectiveness of our approach, we also report the results from a baseline random generation technique. Our evaluation shows that the proposed evolutionary test case generator is more effective at generating failure-revealing test cases and provides higher diversity between the generated failures than the random baseline.

***Index Terms*—Search-Based Test Generation, Evolutionary Algorithm, Advanced Driver Assistance Systems, Pedestrian Detection, Automotive Simulators**

I. Introduction

The capabilities of autonomous vehicles have increased remarkably in recent years. A self-driving car is arguably the most tangible example of what the European Commission (EC) defines as an Artificial Intelligence (AI) system [1]. From an AI perspective, the automotive industry has successfully harnessed the disruptive potential of machine learning over the last decade. Driven by the availability of big data and computing power, deep neural networks (DNNs) have enabled new levels of vehicular perception. However, performing effective quality assurance of systems that rely on DNNs requires a paradigm shift [2]. No longer do human engineers explicitly express all logic of the system in source code. Instead, DNNs are trained using enormous quantities of manually annotated data and perform actions probabilistically based on patterns observed in that data. The research community has put substantial effort into making DNN-based systems trustworthy in the automotive AI context, spurring major R&D projects and global safety standardization efforts.

The concept of Trustworthy AI receives particular attention in the EC's AI Strategy [3]. EC defines AI systems as "software (and possibly also hardware) systems designed by humans that, given a complex goal, act in the physical or digital dimension by perceiving their environment through data acquisition, interpreting the collected structured or unstructured data, reasoning on the knowledge, or processing the information, derived from this data and deciding the best action(s) to take to achieve the given goal" [1]. Novel ways to test AI systems, including autonomous vehicles, are urgently needed—and the research community has taken up the challenge [4], [5].

The use of virtual prototyping platforms for automotive software engineering has rapidly grown in recent years [6]. The use of virtual methods allows testing and validation at early development stages, which leads to fewer development cycles and faster time-to-market. Simulation-based testing is required to complement conventional on-road testing due to severe drawbacks in the use of on-road testing [7], i.e., system testing on public roads is costly and does not scale to the quantity of scenarios needed—in addition, it can be dangerous to provoke a critical situation on the road. Testing autonomous vehicles in simulators is fundamental to quality assurance in the automotive sector—as indicated in the evolving standard ISO 21448 Safety of the Intended Functionality [8].

Efficient and effective testing in simulated environments require sophisticated approaches to automatically generating test cases. Several authors have demonstrated that search-based software test generation (SBST) [9] is a feasible approach to generate critical test scenarios in the automotive context [10]–[14], i.e., test scenarios that lead to the violation of safety requirements. SBST formulates test input selection as a search problem, where optimization algorithms attempt to systematically identify the test input that meet goals of interest. Given a scoring function denoting *closeness to the attainment of those goals*—called *objective function*—optimization algorithms can sample from a large and complex set of test inputs as guided by a chosen sampling strategy (a *metaheuristic*—in our case, a genetic algorithm) [9].

In the 2021 IEEE Autonomous Driving AI Test Challenge competition, our contribution—`ScenarioGenerator`—uses SBST to generate test scenarios that cause the Baidu Apollo's autonomous driving platform to fail. While different scenarios can be tested using `ScenarioGenerator`, for the purpose of this work, we assume a scenario with a pedestrian crossing a street with the following high-level safety goal: "The ego car shall not crash into pedestrians on collision course." We refer to any crashes between an ego car and pedestrians as a safety violation or failure.

Our work relies on a test strategy involving the following steps of simulation-based automotive testing using SBST. We:

1) Build a scene in the virtual environment.
2) Define the parameters involved in creating a varied set of test cases.
3) Define ranges for each parameter, representing the test input space to explore.
4) Define an objective function that measures the *quality* of a generated test case, in terms of its potential to demonstrate a safety violation. In our case, lower scores indicate more dangerous scenarios.
5) Apply a genetic algorithm to generate test cases that minimize the objective function, leading to safety-critical scenarios.

To accomplish this, we first import a pre-existing map into the SVL Visual Scenario Editor and create an initial movement path for a pedestrian using fixed waypoints—a set of coordinates (points) showing the initial path of the pedestrian's movements. Then, during the simulation, in the designed scene, the ego car moves forward toward a target and a pedestrian crosses the road from the right.

The proposed evolutionary test case generation formulates the search space using a generic *noise vector* data structure and minimizes a multi-criteria objective function that combines (1) distances between the ego car and other road agents, (2) the distance of the journey taken by the ego car towards the target, and (3), the number of accidents detected. Using the noise vector, as a generic and flexible structure for representing the search space of the problem, facilitates the use of a wide variety pf search algorithms. This paper presents the results of our proposed test case generation technique in the 2021 IEEE Autonomous Driving AI Test Challenge. To provide the comparative results and demonstrate the efficiency and effectiveness of our evolutionary text case solution, we also compare our results to random generation of test scenarios.

The rest of the paper is organized as follows: Section II presents the details of the proposed search-based test case generation approach. Section III elaborates on the empirical evaluation, including the research method, test scenario execution and experiment setup, results, and threats to the validity of the results. Section V presents an overview of related work, and Section VI summarizes our findings in light of the importance of simulation-based testing of autonomous vehicles and potential research directions for future work.

II. Search-based Test Case Generation

This section present how we use an evolutionary search-based technique to generate test cases. Since each scenario takes a few seconds to execute, it is not feasible to try all possible test scenarios. Our approach is to adopt a generic data structure, i.e., a data vector called a "noise vector", to represent the test input domain for producing test scenarios. Each element of this vector represents a parameter that defines a test scenario, e.g., waypoints, illumination, and weather. The values of these parameters do not lie within the same range, so to bind the values within a specific range, the input representation also scales the concrete real values to values within the range $[-1, +1]$. The values in the noise vector are manipulated by the search algorithm to produce test cases. In our approach, we use a genetic algorithm to explore the search space and produce test cases that are judged as more valuable using an objective function based on potential pedestrian collisions.

A. Scenario Creation and Manipulation

We use SVL Visual Scenario Editor as the first step to create a basic scheme of the test scenarios that are going to be executed by SVL simulator. SVL Visual Scenario Editor is a GUI application that can be used to create basic scenarios specifying where agents (pedestrians, vehicles, ego vehicle, etc.) are positioned in a map and the basic scheme of the path that they should take through the map, which is specified in the form of waypoints.

The basic scenario is created and exported from SVL Visual Scenario Editor to SVL simulator. This scenario is then manipulated by `ScenarioGenerator` to produce new test scenarios. In `ScenarioGenerator`, a derived test scenario is specified by a vector of real numbers, the *noise vector*, with values between -1 and $+1$.

B. Scenario specification

A test scenario is defined as a set of parameters used for test scenario generation, i.e., modeling the test inputs, which is shown as follows:

$$\begin{aligned} TS = \langle S_1, S_2, \cdots, S_m \rangle \ , \ & R_{imin} \leq S_i \leq R_{imax} \\ & R_{imin}, R_{imax} \in \mathbb{R} \end{aligned} \quad (1)$$

Where TS indicates a test scenario and S_i denotes a test input parameter. The values of the test input parameters often vary over different ranges. R_{imin} and R_{imax} represent the upper and lower boundaries of the value range for parameter S_i.

For example, the scenario may define a variable S_{tod} representing the time of day. In the base scenario, the time of day may be defined as 12:00. R_{todmin} and R_{todmax} are used to limit the *change* in this value in a generated test scenario (e.g., values of -5 and 5 would allow the time to vary from 7:00 to 17:00). The values of parameters representing the positions of the agents would have different ranges—e.g., the position points in a path that the vehicle takes may change by ±2 (meters).

C. Noise vector

The proposed representation for a test case is a vector, which is defined as follows:

$$noise\_vector = \langle N_1, N_2, \cdots, N_m \rangle \ , \ -1 \leq N_i \leq +1 \quad (2)$$

where each element, N_i, corresponds to a test input parameter, S_i, and the values of components of the noise vector are scaled to values in R using a linear scaling function to create a test scenario, TS.

$$S_i = (N_i + 1) \times (R_{i_{max}} - R_{i_{min}})/2 + R_{i_{min}} \quad (3)$$

This transformation allows the use of a generic representation that can be uniformly manipulated by the test generator without detailed knowledge of each input parameter. All elements of the noise vector fall within the range $[-1, +1]$, and are scaled appropriately using $R_{i_{min}}$ and $R_{i_{max}}$ for that S_i.

Extending the above scenario, a noise vector value of 0.5 for the entry representing the time of day, S_{tod}, would result in the following concrete value in a test case: $S_{tod} = (0.5 + 1) \times (17 - 5)/2 + 5 = 1.5 \times 6 + 5 = 14$, or 14:00.

D. Objective Function

In order to generate valuable test scenarios, we must identify scenarios that are more likely to lead to safety violations. Safety violations can occur then the ego car moves toward its target at a reasonable speed. Specifically, the objectives to be optimized are as follows:

- The total distance[1] of the ego vehicle from other non-ego traffic during scenario execution. This objective should be *minimized*—we want to examine ego vehicle behavior in potentially dangerous scenarios.

$$ego\_agents\_distance = \sum_{agent \in agents} \sum_{s \in (1,\ldots,steps)} d(ego.pos_s, agent.pos_s) \quad (4)$$

- The total distance of the journey. This should be *maximized*, as longer journeys are preferred.

$$journey\_distance = d(ego.pos_1, ego.pos_{finalstep}) \quad (5)$$

- acc : the number of accidents. This should also be *maximized*, as we seek failures in ego vehicle behavior.

Since the aforementioned objectives do not conflict with each other, we merge them to form a single objective function. This function is *minimized*—lower scores are preferred. The objective function that we seek to minimize is defined as:

$$E = ego\_agents\_distance - journey\_distance - 1000 \times acc \quad (6)$$

We put high values on the number of accidents, as we are interested in generating test scenarios leading to crashes.

E. Search Algorithm

It is not possible to execute every possible test scenario that can be defined by an instance of the noise vector. Instead, we seek a systematic means to sample from the space of possible scenarios in *search* of those that could lead to safety violations. This can be done by using an optimization algorithm to sample the space, as guided by the objective function.

The optimization algorithm used to minimize the objective function is a Genetic Algorithm (GA). Genetic Algorithms are modeled on the evolution of a population over time. Initially, a random population of solutions (*noise_vector* instances) is generated. Then, at each generation, a new population is formed based on the best solutions resulting from the previous generations of evolution. This population is formed by:

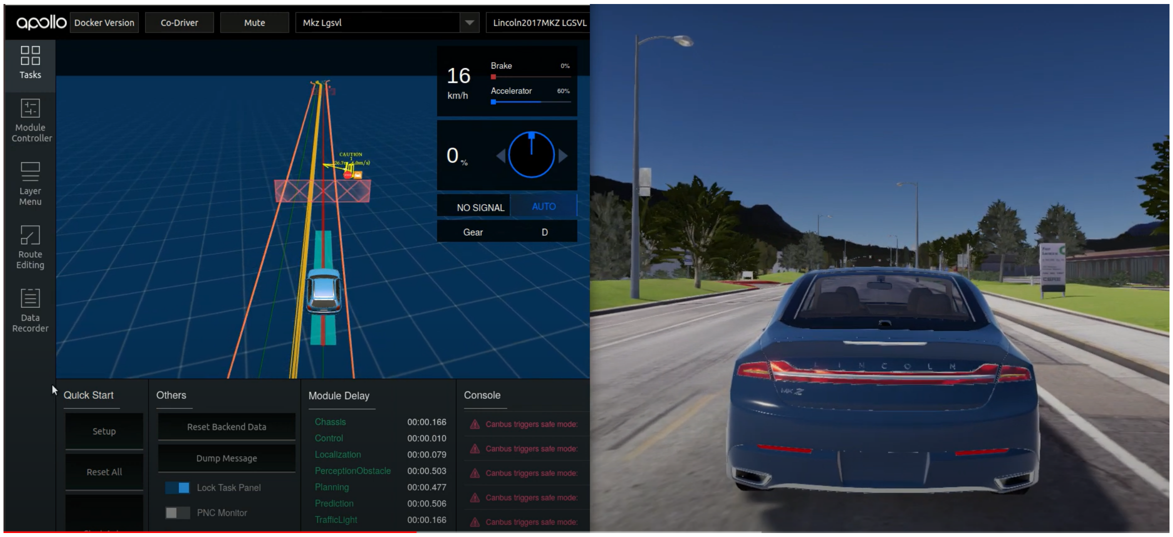
- Identifying good solutions using *tournament selection*, where a subset of the population is selected at random and the best member of the subset is identified.
- Breeding "child" solutions by combining elements of "parent" solutions through *crossover*, where the child solutions are formed by selecting genes (elements) from each parent solution.
- Introducing *mutations* into the population by making small, random adjustments to solutions.

Tournament selection is performed to identify parent solutions, then crossover and mutation are performed at user-set probabilities. Either, or both, may be applied to transform the identified solutions. Finally, the resulting solutions are added to the new population. This process continues until a new population is formed. The objective function is calculated for each member of this population, and the score is stored for that solution. This process is performed each generation, until a user-set number of generations has been exhausted. At the end, the best solutions are returned.

In our case, we have three objectives—$ego\_agents\_distance$, $journey\_distance$, and acc, which have been merged into a single formula. Tournament selection picks the best solution among the solutions in each tournament. The number of individuals participating in each tournament denotes the size of the tournament. In our approach, we omit the crossover operation, as the noise vector contains the values for the parameters of the test scenarios in a certain order, and crossover could violate this ordering. Instead, we apply mutation with a high probability. We use *Polynomial Bounded mutation*, as proposed and implemented in NSGA-II [15]. It is a bounded mutation operation for real-valued parameters and uses a polynomial function for the probability distribution. It uses a parameter, eta indicating the *crowding degree* of the mutation, which is used to encourage diversity in the resulting population. A high eta yields a mutant resembling the original solution, while a small value for eta produces a solution more divergent from the original. The GA algorithm used for generating test scenarios is configured as presented in Algorithm 1.

[1] Euclidean distance

$$d(p1, p2) = \sqrt{(p1_x - p2_x)^2 + (p1_y - p2_y)^2 + (p1_z - p2_z)^2)}$$



Module Delay	
Chassis	00:00.166
Control	00:00.010
Localization	00:00.079
PerceptionObstacle	00:00.503
Planning	00:00.477
Prediction	00:00.506
TrafficLight	00:00.166

```
Canbus triggers safe mode:
Canbus triggers safe mode:
Canbus triggers safe mode:
Canbus triggers safe mode:
Canbus triggers safe mode:
```

Fig. 1: Overview of the experimental setup.

Algorithm 1 GA for Test Scenario Generation

```
Initialize population with solutions from random seeds;
Evaluate the population;
repeat
    1. Select offspring using Tournament Selection with replacement;
    2. Mutate the resulting offspring using Polynomial Bounded mutation operation with a certain probability (mutation rate = 0.95);
    3. Evaluate the offspring using the objective function.
until meeting the stopping criteria (reaching the maximum number of generations or other limitations specified in the test budget);
```

III. Implementation and Empirical Evaluation

We perform an empirical evaluation of the proposed test case generation technique, ScenarioGenerator[2] by running experiments on our experimental setup on a desktop PC with the following specifications:

- Ubuntu version 18.04
- Intel Core i7-10700K CPU @ 3.80GHz × 16
- 32GB RAM
- GeForce RTX 2070 SUPER/PCIe/SSE2
- SVL simulator 2021.1 (linux64) with modular testing setup (3D Ground Truth sensor and Signal sensor publish ground truth perception data to Apollo via CyberRT bridge)
- Baidu Apollo (r6.0.0 branch)

The experiments are simulations that are controlled by a Python scenario runner which uses our test case generation technique for generating the scenarios in the simulation environment. Baidu Apollo is the autonomous driving software platform that controls the ego vehicle. It connects to the simulator through its customized bridge and drives the ego vehicle (Fig. 1).

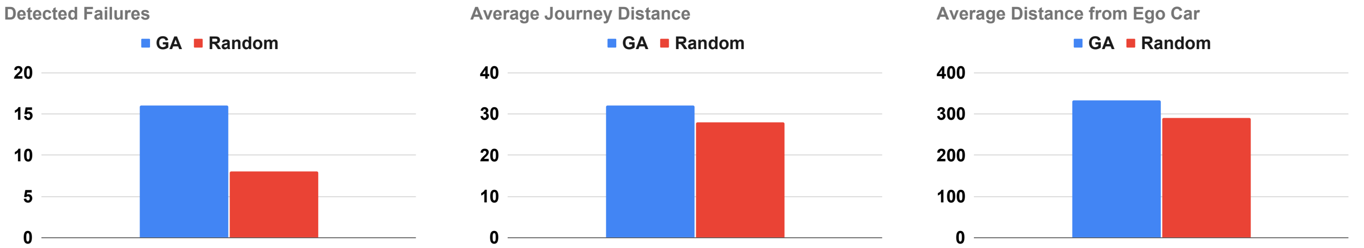
We design a set of experiments to assess the efficiency and effectiveness of the proposed test case generation for testing Apollo in the SVL simulation environment. Pedestrian detection and proper responding is the target use case of Apollo in our experiments. For a comparative analysis, we also report results from a random testing technique as a baseline approach. In random testing, the test cases are generated randomly, which means that the set of noise vector instances are generated by setting the test input parameters to random values within the allowed range. The target is to generate the highest number of diverse valid test cases leading to failures, i.e., collisions between the ego vehicle and pedestrians. We use the following quality criteria for evaluating the proposed test case generation technique:

- **Detected Failures:** The number of test cases that lead to a collision.
- **Failure Diversity:** The dissimilarity between the generated test cases leading to failures. We are interested in generating diverse test cases, as triggering similar failures lead to waste of the test budget, e.g., computation resources. To measure failure diversity, we use the *Euclidean distance* between failing noise vectors.

A. Test Scenario Execution

The testing budget (including, e.g., execution time) is a limited resource. While not as expensive to perform as on-road testing, running test scenarios in simulators also takes time. In our experiments, each scenario takes about 10 seconds to execute and evaluate. Therefore, for the purpose of this competition, we set the limit for the number of simulation executions to 200 in the Genetic Algorithm. This would correspond, for example, to 20 generations with a population size of 10.

[2] Available from https://github.com/ebadi/ScenarioGenerator.



(a) Number of detected failures.

(b) Objective values for the average journey distance during failure-revealing test cases.

(c) Objective values for average distance from ego car during failure-revealing test cases.

Fig. 2: Comparisons between GA and random generation.

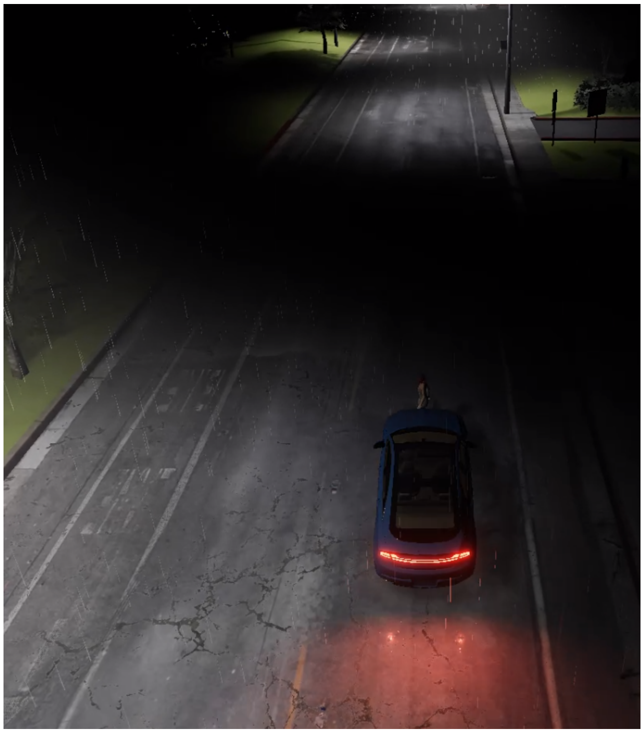


Fig. 3: Collision between pedestrian and the ego vehicle on a rainy night.

In `ScenarioGenerator`, the user-controllable parameters for test scenario creation and manipulation are as follows:

- Initial JSON file created by SVL Visual Scenario Editor.
- Test case generation strategy, which is used for scenario generation. Currently, Differential Evolution, Powell Optimization, Genetic Algorithm, and random generation strategies are supported. Meanwhile, the capability of replaying a scenario is also supported by passing the JSON file and setting the action to *replay*. A specific noise vector in combination with replay action can also be used. In this mode, in addition to all the previous parameters, a specific noise vector is given to be played.
- The ego vehicle destination.
- Acceptable range of changes in the values for the *position of each waypoint* (x, z).
- Acceptable range of changes in the color of each agent (r, g, b).
- Acceptable range of changes in the weather in the simulation (e.g., rain, fog, wetness, cloudiness, road damages).
- Acceptable range of changes in the time of day.
- Acceptable range of changes in the speed of each agent.

In a test case, the generated noise vector is used to impose changes to the position of each waypoint, the color of each agent, the weather, the time of day, and the speed of each agent. The base scenario defines a value for each of these parameters. The user-controllable parameters are used to constrain the range of changes made by the voice vector between minimum and maximum values, as discussed in Section II.

IV. Results and Discussion

This section presents the experimental results and assesses the proposed test case generation compared to the random testing with regard to the quality criteria.

Detected Failures: Fig. 2(a) shows the number of detected failures (test cases leading to collisions) by the GA-based test case generation and random testing. The proposed GA-based technique trigger twice as many failures than random testing on the same configuration and test budget, and consequently, in this regard, works more effectively. Fig. 3 also shows a sample of a generated test scenario leading to a collision between the pedestrian and the ego vehicle.

In order to investigate the characteristics of the detected failures, we can examine the values of two of the objectives in the objective function—$ego\_agents\_distance$ and $journey\_distance$. These can show the characteristics of the detected failures. Fig. 2(b) and (c) show the average values of the two objectives in failure-revealing test cases for both techniques. These average values do not differ significantly between the two approaches. This indicates that the GA reveals more failures, but the failures revealed by the two techniques fall in similar objective ranges. However, both distances are somewhat higher in the GA—i.e., the GA generates tests with slightly longer journey distances and a slightly higher distance from the ego car. These tests may be somewhat more interesting for revealing errors in the ego car functionality, as—for example—a longer distance between the ego car and

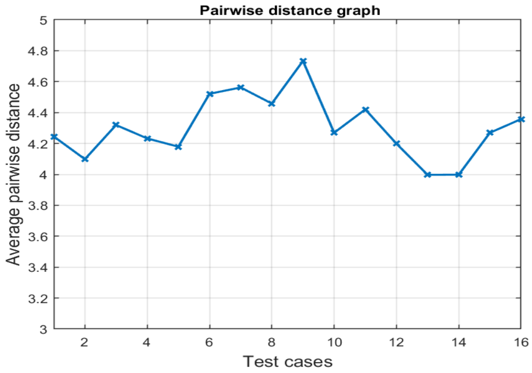


Fig. 4: Diversity of failure-revealing test cases generated by the GA.

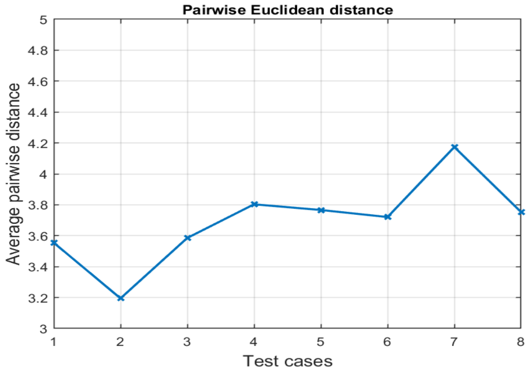


Fig. 5: Diversity of failure-revealing test cases generated by random testing.

TABLE I: Failure diversity in GA and random testing, shown as the range in the average pairwise Euclidean distance for test cases.

	Genetic Algorithm	**Random**
Range of Euclidean Distances	$4.1 - 4.7$	$3.2 - 4.2$

a pedestrian should offer more time to make corrections. In future work, we will examine failing scenarios more closely and discuss them with domain experts.

Failure Diversity: We use pairwise Euclidean distance between the noise vectors to show diversity between the failure-revealing test cases. Fig. 4 and 5 show the average pairwise Euclidean distance for each of the failure test cases generated by GA and random testing respectively. The average pairwise Euclidean distance refers to the average difference between a test case and the other test cases. Table I shows the range of average pairwise Euclidean distance for the failure-revealing test cases from the GA and random testing. In this regard, the GA technique also promotes more diversity between generated failure-revealing test cases than random testing.

A. Threats to Validity

Some of the main sources of threats to validity of the experimental results are as follows:

Internal Validity: During the experiment, we noticed that many of the failures that are captured are not completely reproducible. In fact, the simulation execution often does not produce identical results given identical input parameters and configuration setup. One of the main reasons is that Apollo does not function in a deterministic manner. We tried to mitigate the effects of this by reporting average values from the experiments, and conducting the experiments in a controlled manner, i.e., using the same experimental setup and keeping the user-controllable parameters fixed between executions. Another source of threat is the fact that as the simulator runs a large number of test cases, the simulations become slower and less responsive probably due to performance bottlenecks.

External Validity: We have focused on a single scenario. As we have used a generic data structure consisting of variables scaled in a certain range, i.e., the noise vector with variables within the range $[-1, 1]$, we believe that the representation model and test case generation approach could be used for simulation-based testing of more complex scenes and other use cases. However, the variables in the noise vector might need to be modified (e.g., extended) for different use cases.

V. Related Work

Simulators as a form of digital twins play a key role for different purposes in testing and verification, control and monitoring, and improvement of cyber-physical systems (CPS). For ADAS and autonomous-driving cars, this is even more significant and there is a higher demand for high-fidelity simulators. Simulation-based testing is one of the most effective approaches for system-level testing of ADAS and acts as a suitable complementary solution to on-road testing, since it provides the possibility for early stage testing, capturing critical corner test scenarios and enabling inexpensive testing. Field testing of such systems is expensive, inefficient and even dangerous, in some cases. Recently, various simulators such as those ones using physics-based models, e.g., SVL simulator [16], PreScan [17] and Pro-SiVIC [18] or the ones relying on game engines, e.g., BeamNG [19] and CARLA [20], have been developed to meet the need for realistic simulation of the functions in autonomous driving.

Accordingly, various system-level testing approaches relying on the simulators have been proposed in the recent years. One of the common intended purposes in those studies is generating critical test cases (scenarios) that lead the system to fail. This is a challenging problem, due to the large search space of input parameters in these systems. Covering all possible simulation test scenarios is not feasible in practice. Therefore, in this regard SBST techniques have been widely used to generate effective test simulation scenarios for those systems. In recent studies, multi-objective search algorithms like NSGA-II [10], many-objective algorithms like MOSA [21] using a combination of different objectives based on branch coverage and failure-based heuristics [22], and learnable evolutionary algorithms [23] have been used to generate critical test cases leading to violations of safety requirements in autonomous driving cars. Moreover, there have also been studies focusing on the role of simulators and the type of test data. In [24] a comparison between testing of DNN-based ADAS using real-world and simulator-generated data is conducted and it is also showed that how on-line and off-line testing of these systems can differ and meanwhile complement each other. Markus et al. studied the consistency between the results obtained from two different simulators and investigated whether the obtained results could be mutually reproducible in both simulators [13].

VI. Conclusion and Future Work

Efficient and effective test case generation for use in virtual environments is essential for testing AI-based automotive systems. In this paper, we presented a SBST approach to generate test scenarios that lead to detection of failures and safety violations of the Baidu Apollo pedestrian emergency braking system. We have made three primary observations. First, our results show that the proposed GA-based test case generation is more effective than random testing, i.e., it is more effective in generating failure revealing test cases and provides higher diversity between the generated test cases compared to random testing. Second, unfortunately, many of the captured failures could not be reproduced given the same configuration and user-controlled parameters due to the non-deterministic nature of Apollo. Third, we see great potential in simulation-based testing of different functions of autonomous driving systems using SVL simulator and Baidu Apollo. In future work, we will broaden the scope of the research into additional safety scenarios. We will also extend SBST approaches with machine learning-based techniques (e.g., reinforcement learning) for test case generation in system-level testing of ADAS. We are also interested in the use of Generative Adversarial Networks (GANs) as a technique for enabling the discovery of failure-revealing test cases.

Acknowledgment

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 876852 (VALU3S). Furthermore, this work received support from the ITEA3 European IVVES project (https://itea3.org/project/ivves.html) and the SMILE III project financed by Vinnova, FFI, Fordonsstrategisk forskning och innovation under the grant numbers 2019-05871 and the AIQ Meta-Testbed project funded by Kompetensfonden at Campus Helsingborg, Lund University, Sweden. Additional support was provided under Vetenskapsrådet grant 2019-05275. The authors would like to thank INFOTIV AB for their support and cooperation.

References

[1] "A definition of artificial intelligence: Main capabilities and scientific disciplines," High-Level Expert Group on Artificial Intelligence, Brussels, Belgium, Tech. Rep., 2018.

[3] "Communucation from the commission to the european parliament, the european council, the europan economic and social commitee and the commitee of the regions - artificial intelligence for europe," Eurpean Commision, Brussels, Belgium, Tech. Rep., 2018.

[2] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Lewandowski, S. Gao, Y. Tan, H. Kaijser, H. Lönn, and J. Törnqvist, "Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry," *Journal of Automotive Software Engineering*, vol. 1, no. 1, pp. 1–19, 2019.

[4] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[5] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5193–5254, 2020.

[6] F. Bock, C. Sippl, S. Siegl, and R. German, "Status report on automotive software development," in *Automotive Systems and Software Engineering*. Springer, 2019, pp. 29–57.

[7] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[8] "Road Vehicles - Safety of the Intended Functionality," International Organization for Standardization, Tech. Rep. ISO/PAS 21448:2019, 2019.

[9] P. McMinn, "Search-based software testing: Past, present and future," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 153–163.

[10] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proc. of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 63–74.

[11] ——, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proc. of the 40th International Conference on Software Engineering*, 2018, pp. 1016–1026.

[12] A. Gambi, T. Huynh, and G. Fraser, "Generating effective test cases for self-driving cars from police reports," in *Proc. of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 257–267.

[13] M. Borg, R. B. Abdessalem, S. Nejati, F.-X. Jegeden, and D. Shin, "Digital twins are not monozygotic–cross-replicating adas testing in two industry-grade automotive simulators," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 383–393.

[14] M. H. Moghadam, M. Borg, and S. J. Mousavirad, "Deeper at the sbst 2021 tool competition: ADAS testing using multi-objective search," in *2021 14th Intl. Workshop on Search-Based Software Testing (SBST)*. IEEE, 2021.

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[16] LG Electronics, "SVL Simulator," https://www.svlsimulator.com/, Retrieved July, 2021.

[17] TASS International, ". PreScan Simulator," https://tass.plm.automation.siemens.com/prescan-overview, Retrieved July, 2021.

[18] A. Belbachir, J.-C. Smal, J.-M. Blosseville, and D. Gruyer, "Simulation-driven validation of advanced driving-assistance systems," *Procedia-Social and Behavioral Sciences*, vol. 48, pp. 1205–1214, 2012.

[19] BeamNG GmbH., "BeamNG.research," https://beamng.gmbh/research/, Retrieved July, 2021.

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[21] A. Panichella, F. M. Kifetew, and P. Tonella, "Reformulating branch coverage as a many-objective optimization problem," in *2015 IEEE 8th international conference on software testing, verification and validation (ICST)*. IEEE, 2015, pp. 1–10.

[22] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Testing autonomous cars for feature interaction failures using many-objective search," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2018, pp. 143–154.

[23] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 1016–1026.

[24] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Comparing offline and online testing of deep neural networks: An autonomous car case study," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 85–95.