Optimizing coverage of simulated driving scenarios for the autonomous vehicle

Marc Nabhan, Marc Schoenauer, Yves Tourbier, Hiba Hage

Optimizing coverage of simulated driving scenarios for the autonomous vehicle

Marc Nabhan
CAE & Testing Dept.
Renault SAS
Guyancourt, France
marc.nabhan@renault.com

Marc Schoenauer
LRI-TAU
Inria
Orsay, France
marc.schoenauer@inria.fr

Yves Tourbier
CAE & Testing Dept.
Renault SAS
Guyancourt, France
yves.tourbier@renault.com

Hiba Hage
CAE & Testing Dept.
Renault SAS
Guyancourt, France
hiba.hage@renault.com

***Abstract*—Self-driving cars and advanced driver-assistance systems are perceived as a game-changer in the future of road transportation. However, their validation is mandatory before industrialization; testing every component should be assessed intensively in order to mitigate potential failures and avoid unwanted problems on the road. In order to cover all possible scenarios, virtual simulations are used to complement real-test driving and aid in the validation process. This paper focuses on the validation of the command law during realistic virtual simulations. Its aim is to detect the maximum amount of failures while exploring the input search space of the scenarios. A key industrial restriction, however, is to launch simulations as little as possible in order to minimize computing power needed. Thus, a reduced model based on a random forest model helps in decreasing the number of simulations launched. It accompanies the algorithm in detecting the maximum amount of faulty scenarios everywhere in the search space. The methodology is tested on a tracking vehicle use case, which produces highly effective results.**

***Index Terms*—ADAS, autonomous driving, validation, optimization, machine learning**

I. Introduction

Autonomous vehicles are being hailed as the future of road transportation. Their deployment will hypothetically mitigate human drivers' mistakes, since they are not subject to distraction and fatigue. Their improved perception and decision-making should naturally lead to better execution and more-precise driving. However, self-driving cars face potential risks as well. They represent a highly complex system, from the coordination of its various sensors to their fusion, leading to the command law which ultimately decides the action to be executed by the vehicle. A failure can occur during any stage of this process, which will result in a wrong execution on the road. Typically, two failures are distinguished: false alarms and non-detections. A false alarm can be as dangerous as a non-detection, since in both cases, the car will take a concrete action and might head into a crash. Thus, each component of this whole system should be effectively tested in order to anticipate any kind of failure and eliminate it.

The engineering team at Renault, and every other major car company, rely on real test-driving under various conditions in order to validate the autonomous vehicle. Their main objective is to detect specific system failures during these tests. Then, these failures can be eliminated by updating the system accordingly. The complete test database is conducted at different milestones identified in the design process. Nonetheless, the testing of the autonomous vehicles faces numerous challenges [1]. For instance, real test-driving is long and expensive, as it is necessary to reproduce accurately all the imaginable conditions, e.g. weather and traffic, that an autonomous vehicle can encounter. Plus, studies show that at least 8.8 billion miles of test driving, i.e. 400 years of driving, are needed to demonstrate with 95% confidence that the autonomous vehicle failure rate is lower than the human driver failure rate [2]. Hence, it is almost impossible to validate the self-driving cars using only real test-driving. This is why, due to the increase in computing power nowadays, numerical models and simulations are used to test autonomous vehicle functions [3]. They can be used together with real test-driving to help covering various conditions of the miles needed [4]. Typically, two types of numerical validations are distinguished. On one hand, resimulation is useful to carry out regression tests. Real test-driving data are injected in a numerical model of the command law to try to replicate them in open loop. On the other hand, numerical simulation generates virtual parameterized data, creates new tests which were unavailable in the original test database, and runs these tests directly in a simulation loop. The key advantage of the second method is that it is a closed loop, which makes the optimization more robust.

The work presented here is part of an on-going industrial project that aims to validate the command law using numerical simulations. Optimization algorithms, machine learning and deep learning models are considered and used. While previous studies have used machine learning techniques to optimize the detection of test cases with higher probability of failures [5], the objective of this paper is to present an algorithm designed to detect a maximum number of failures of the autonomous vehicle command law in the space of input parameters of the simulator. The main industrial restriction, however, is that this goal must be achieved by running as few numerical simulations as possible, in order to minimize the computational power. Thus, the final model should be fast and inexpensive, while being efficient in finding the highest number of command law failures caused by scattered input conditions. The remainder of this paper is as follows. Section

II presents the problem statement. Section III details the models and methods used. Section IV presents and discusses the results obtained on a specific use case, namely the simplest case of the autonomous vehicle having to follow another vehicle in the same lane. Finally, Section V concludes the paper.

II. Problem Statement

All driving sequences of the autonomous vehicle can be enumerated, as far as simulation is concerned, as hundreds of use cases, each characterized by input variables of the simulator. Input variables define the scenario, an instanciation of a use case. Thus, the number of possible scenarios for a use case is proportional to the use case dimension and the range or list of values of each input parameter. The outputs of the simulator are the actual values of various criteria measured on the autonomous vehicle during the simulation. These criteria are organized by theme, e.g. safety, testing, and marketability. An example of a use case is the one under experimentation in this article as seen in Fig. 1. It consists of an autonomous vehicle, commonly called EGO, tracking another vehicle in front of it while staying on the same lane. The preceding vehicle is scheduled to perform acceleration and deceleration cycles during the simulation. The input variables of this use case are described in Table I.

After the input values have been set, the simulation is launched, EGO being controlled by the command law to be validated. The software chosen by Renault is SCANeR Studio [6]; it is dedicated to simulation and testing for Advanced Driver-Assistance Systems (ADAS) and Autonomous Driving (AD), and provides various tools to run realistic simulations. Thus, the environment of the autonomous vehicle can be chosen accurately in order to examine the autonomous vehicle command law in different situations. During the simulation, the preceding vehicle performs acceleration and deceleration cycles as described, and the autonomous vehicle response is monitored throughout the scenario using the output criteria. Three output variables are taken into consideration in this work, as detailed below. Each of them fits into one of three test categories linked to the EGO vehicle: EGO position with respect to its environment, EGO dynamics, and EGO position with respect to other vehicles.

First, the lateral lane decentering distance aims at qualifying the lateral position of EGO in its lane, and calculates how far EGO is from the center of its lane. If the maximum decentering distance value reached throughout the simulation is higher

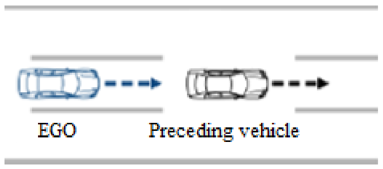

Fig. 1. Use case under experimentation where EGO is tracking a preceding vehicle that is performing acceleration and deceleration cycles on the same lane.

TABLE I
Description of the input parameters of the tracking vehicle use case under experimentation

Input parameters specifications		
Name	***Unit***	***Range/List***
Initial velocity of PV*	km/h	$[60, 110]$
Acceleration cycle value of PV*	m/s^2	$[1, 3]$
Deceleration cycle value of PV*	m/s^2	$[-3, 0]$
Acceleration/deceleration cycles time	s	$[0, 5]$
Initial inter-vehicle distance	m	$[3.5, 200]$
Type of PV*	-	Car, motorcycle, bus
Type of ground network	-	8 types of roads

*PV refers to the Preceding Vehicle in the use case.

(resp. lower) than a fixed threshold (0.4015 meters for a road measuring 3.5 meters for example), then the run is labeled *NG*, i.e. “No Good” (resp. *G*, i.e. “Good”) for this criterion.

Second, the longitudinal deceleration warning qualifies the stress suffered by a passenger due to a longitudinal deceleration in a straight line, and is defined as the worst perception among deceleration and jerk effects computed from lookup tables. It ultimately outputs an integer value that references a state of the passenger's stress; 1, 2, 3 and 4 indicate comfort, dynamic, sport and emergency respectively. Because the goal is that the passenger always feels comfortable while driving EGO, this warning returns *NG* if the output value is higher or equal than 2, and *G* otherwise.

Third, the safety time gap warning monitors the distance between EGO and the preceding vehicle. In France, the threshold of the safety time gap is set by the Highway Code to 2 seconds [7], below which the car is considered to be unsafe. Therefore, this warning detects a failure and returns *NG* if the time gap between EGO and the preceding vehicle takes a lower value than the threshold during the simulation, and *G* otherwise.

Our goal in this work is to identify the maximum number of input values of the simulator resulting in a failure (at least one warning returns *NG*). This can be viewed as an optimization problem, requiring to run the simulator in the optimization loop. Nonetheless, three restrictions surface.

First, one way of detecting all possible failures is to perform an exhaustive grid search on all the input parameters. Let us assume we take into account all the values of the two discrete inputs, and normalize all five continuous inputs between 0 and 1 while setting a step size of 0.1. The resulting number of scenarios to be tested is 3,865,224, knowing that other use cases are defined by a much larger number of inputs. Furthermore, the validation process needs to be executed for all use cases, and should be repeated every time the command law is modified. Thus, grid search is not an option, and developing an optimization algorithm that performs only a handful of scenario simulations is more effective for validating industrial project milestones in terms of meeting time and cost requirements.

Second, a crucial industrial restriction is to minimize the overall computational cost, which translates into launching as few simulations as possible. However, the optimization algorithm requires to have access to the outputs of the scenario in order to find the best candidate while altering the input values. A workaround is to use a reduced model during the course of the algorithm, thus avoiding to use the actual simulation software. This model would output the criteria value needed for the algorithm, and be updated throughout the algorithm iterations with the results of some actual simulations.

Third, the runtime of a simulation of one scenario for this use case is about 5 minutes. Due to technical and computer limitations in the software, a massive simulation plan cannot be run in parallel as of today. Therefore, if thousands or tens of thousands of simulations are required to explore the input search space with the algorithm, hundreds of hours are needed to sequentially test the optimization algorithm for a single use case. A solution is to build a surrogate model of the simulation software which can instantly output the criteria of the scenario. In that way, the optimization algorithm can be tested more quickly until running multiple simulations in a parallel operating mode becomes possible. The surrogate model, the reduced model and the optimization algorithm are each detailed next.

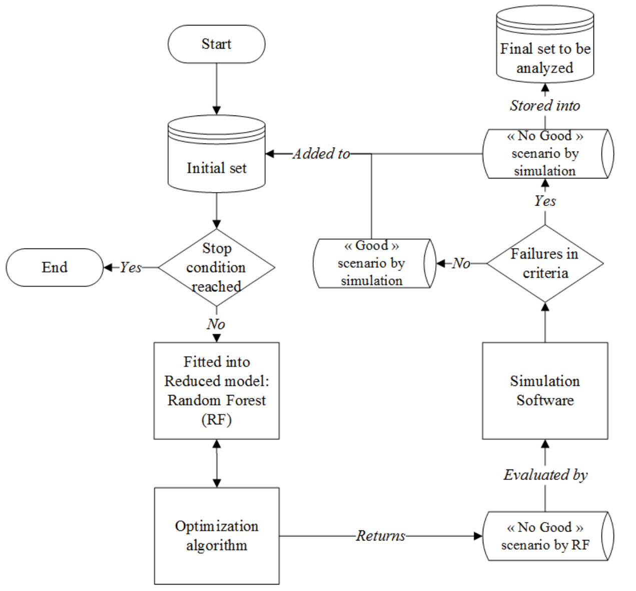

Fig. 2. Flowchart describing the framework of the study where the optimization algorithm is required to detect new faulty scenarios the farthest possible from the faulty scenarios of the initial set while using the simulation software as little as possible.

III. Models & Methods

The flowchart in Fig. 2 explains the structure of our study. In this article, a scenario whose command law failed during the simulation through at least one of the three criteria, qualifies as *NG* for "No Good", while another scenario that was simulated without any failures detected in any of the criteria is *G* for "Good". First, the initial set is fitted into the reduced model designed to be used by the optimization algorithm. The algorithm's goal is to output a scenario that is *NG* while being simultaneously the farthest possible from the *NG* scenarios of the initial set. Then, it evaluates the real criteria values of that scenario by simulation. In fact, the reduced model could output a wrong faulty scenario, because it is based on a small initial set that does not cover the whole search space. Along with the new criteria predicted, the scenario is added to the initial set. If the scenario was indeed found to have a failure in its criteria, then it is stored in a final set for later analysis. This whole process marks a single iteration and is repeated until the algorithm meets its stopping condition. Furthermore, after the completion of the algorithm, the *NG* scenarios that have been found are evaluated to estimate how well they cover the search space, how large are the "holes" left compared to a full grid of the same search space (see Section IV).

A. Surrogate model

For the aim of building a reliable surrogate model, a design of experiments with a maximin decision rule is built [8]. Its goal is to maximize the Euclidian distance between the scenarios, so that the experimental design is composed of scenarios far apart from one another in the input space parameter. It consists of a total of 19,992 scenarios scattered in the input space parameters, and represents as effectively as possible various areas of input combinations that can be injected at the start of the simulations. These scenarios are input to the simulation software and their corresponding outputs are retrieved.

After completed this experimental design of inputs and outputs, we feed them into a neural network for multiple output regression. The inputs of the neural network are the inputs of the use case, normalized between 0 and 1 for the continuous variables, and one-hot encoded for the discrete variables (i.e. transformed into one boolean variable per possible value). The neural network is composed of 4 layers of 300, 200, 100 and 3 neurons respectively. The 3 neurons in the output layer correspond to the 3 warnings in the use case. The loss is the mean-squared error, and we use the Adam optimization algorithm [9]. Moreover, to better analyze the results of the model, its output values are compared to the corresponding thresholds of the criteria in order to determine the classification accuracy of the warnings: *G* or *NG*. These evaluations are then compared to the real values of the dataset to examine if the neural network was able to predict the warning evaluations correctly from the scenario inputs. By counting the number of times the predictions were correct, accuracies are calculated for each criterion. The global accuracies obtained are 98.61%, 96.13% and 96.01% for the training set, the validation set (used to choose the hyper-parameters, namely here the architecture of the network) and the test set respectively. Thus, the neural network makes an acceptable surrogate model of the simulation software. We notice that 24 million parameters were needed in order to reach these accuracies, which reflects the difficulty of this problem. In the following of this paper, this neural network will be used in lieu of the simulation software whenever a simulation is needed or mentioned, thus overcoming the current software limitations.

B. Reduced model

As stated earlier, a reduced model is crucial for the goal of minimizing the use of software simulations during the functioning of the algorithm. First, Sobol sequences, which are an example of low discrepancy sequences, are generated from within the input ranges, being able to fill the space of possibilities more evenly [10]. Then, scenarios are drawn uniformly from this set, are simulated to collect their criteria, and are included in the initial set of the algorithm until a certain number of *NG* scenarios are retrieved. This condition is necessary for the algorithm to function, because it needs initial *NG* scenarios in order to detect others that are far away from the initial ones. The number of *NG* scenarios required in the initial set is fixed by the user, so as to eventually build an initial set of less than a hundred scenarios.

Once such an initial set has been retrieved, the reduced model can be created. We chose it to be a regression random forest model [11]. It is a supervised machine learning algorithm that combines multiple decision trees, and outputs the mean prediction of the individual trees in case of a regression, in order to avoid overfitting. The random forest model is created and fitted into the initial set. Next, it is used by the algorithm while trying to find the best scenario candidate during each iteration. The new scenario is added afterwards to the initial set after being injected to the simulation software, whether it is *G* or *NG*. The random forest is then updated at the beginning of the next iteration by fitting it into the newly expanded initial set.

C. Optimization algorithm

As already mentioned, the goal is to detect scenarios that have failed through at least one of the three criteria, while being the farthest possible from the already known *NG* scenarios from the initial set. For that purpose, we need to compute the distance between a given scenario and all these *NG* scenarios. Then, the aim of the algorithm will be to maximize that distance while making sure that the scenario corresponds to a *NG* scenario. Hence, a function that calculates this distance is developed. It takes a scenario as input, and starts by predicting its criteria using the random forest model (as using the simulation would be too costly). The distance is calculated for each criterion using the following equations, where n is the dimension of the use case.

$$d(x) = \begin{cases} \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}, & \text{if criterion is } NG \quad (1) \\ 0, & \text{if criterion is } G \quad (2) \end{cases}$$

If a criterion returns *NG*, the Euclidean distance between the normalized input scenario x and all the scenarios of the initial set y that are *NG* for that same criterion is calculated in (1). If a criterion returns *G*, then the distance equals zero (2). Hence, *G* scenarios are penalized and *NG* scenarios are favored, since the algorithm will be pushing to maximize the distance. All the distance values obtained by the *NG* criteria are gathered at the end of the function, and their minimum value is identified as the final return value of the function.

The optimization algorithm used here is CMA-ES, the Covariance Matrix Adaptation Evolution Strategy, the state-of-the-art of derivative-free continuous global optimization algorithms [12]. It requires an initial guess, the objective function (the distance function described above), and various parameters to be tuned e.g. the input bounds, the initial step size and some tolerance. The initial guess is drawn uniformly outside the initial set, and its criteria are predicted by the random forest model. If a starting scenario is *G* through all its criteria, the distance computed will be zero, and the algorithm will pass to another starting point. That is why *NG* scenarios are needed in the initial set as mentioned previously. Subsequently, the CMA-ES algorithm will perform multiple iterations until detecting the best candidate scenario with the maximal distance value. More precisely, it goes through a cycle of proposing a scenario, predicting its criteria using the reduced model, and calculating its distance. Because the distance of the best scenario is not zero, at least one of its criteria reported a failure and the scenario is *NG* according to the reduced model. The algorithm continues until it meets its stopping condition. Several stopping conditions have been tried during this study and are presented, along with post-analysis and evaluation of the solution, in the next section.

IV. Results & Discussion

In this study, the preceding vehicle type is fixed as "car", and the road type as "normal" road whose track width measures 3.5 meters. The values of all five remaining inputs range between their corresponding intervals listed in Section II. Hence, the optimization problem dimension is reduced from 7 to 5. The algorithm stopping condition is triggered when some fixed distance is reached. As a matter of fact, the distance value returned by the objective function is globally decreasing along the iterations, until it reaches the stopping condition. For instance, Fig. 3 shows the evolution of the objective function throughout the algorithm iterations that successfully returned a *NG* scenario later evaluated by simulation. We notice that the decrease of the objective function is accompanied by numerical noise, which is the highest during the first iterations and begins to attenuate as the iterations go by. This can be explained due to the fact that the random forest model is recomputed using only tens of scenarios during the first iterations. Therefore, its accuracy is initially poor, but hopefully increases along the iterations. Plus, for each *NG* criterion, the distance function considers only the scenarios from the initial set that have the same *NG* evaluation for that same criterion. Thus, the distance value returned by the function is dependent of the initial scenarios considered, which in turn are determined by the predictions of the random forest model. Nonetheless, as the iterations go by, the reduced model is fitted into an ever-expanding set of scenarios with correct predictions by simulation. Therefore, it is updated continuously and is gaining in accuracy. That is

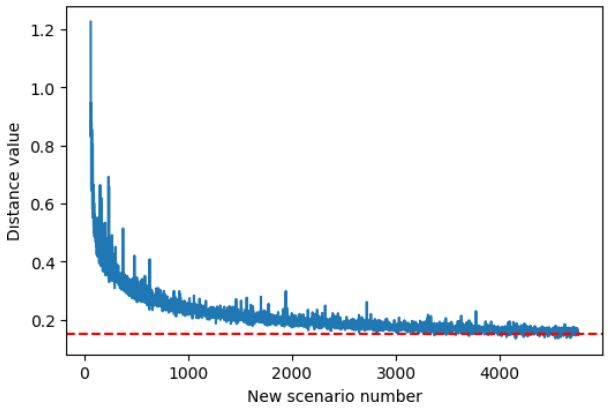

Fig. 3. Evolution of the distance value during the algorithm iterations that resulted in a faulty scenario after evaluation by simulation for a test with a stopping condition of reaching the minimal value of 0.15, a hundred times.

why the numerical noise is reduced throughout the iterations. Several values of the stopping condition have been tried out corresponding to precision values equal to 0.3, 0.25, 0.2 and 0.15. Furthermore, in order to account for the numerical noise, the stopping condition was triggered only after the threshold value has been passed several times, e.g. 10, 50, 100, 250 or 500 times, to make sure that we globally reached the stopping condition distance value.

In order to evaluate the quality of the resulting set of scenarios, it is compared to the *NG* scenarios of a full grid over the search space. Each normalized input is discretized into 10 values, resulting in a total of 100,000 scenarios. These scenarios are actually simulated and their criteria are retrieved. The total number of *NG* scenarios found in the grid is 46,722 scenarios. Next, the idea is to take each *NG* scenario from the grid, and to calculate its distance with all the *NG* scenarios of the final set. If at least one of these distances is less than some fixed precision, it means that the algorithm was able to predict a *NG* scenario that is close to that grid scenario at this precision. The number of such instances can be viewed as a discovery rate up to the given precision. If the algorithm manages to predict scenarios close to all the grid scenarios within the precision distance, it has succeeded in exploring the whole search space and detecting all the failures.

To visualize the evolution of the discovery rates according to the number of simulations performed during a test, checkpoints are considered throughout the iterations. Each checkpoint takes place after a fixed number of simulations, and reports the corresponding proportion of the final set obtained. When the algorithm ends, discovery rates are calculated for each checkpoint according to the final set. These rates are then illustrated according to the number of simulations at each checkpoint. Furthermore, in order to effectively validate this stochastic algorithm, 11 runs are conducted for each stopping condition, and statistical significance is tested. For that matter, 11 initial sets are generated, so as to they all contain 30 *NG* scenarios. Each set has a different seed for the random function, meaning that the uniform draw of the scenarios will also be different. Therefore, each of the 11 runs has a unique initial set. After the 11 runs are conducted, final sets are compared to the grid scenarios and discovery rates are computed. Then, the mean and standard deviations of the discovery rates of the 11 runs are computed. Finally, the evolution of the mean discovery rate w.r.t the number of simulations is visualized, along with vertical error bars equal to twice the standard deviation.

Such plots for four experiments are shown in Fig. 4. To visualize the difference between various stopping conditions, each test example stems from a different precision value set as stopping condition: 0.3, 0.25, 0.2, and 0.15. Plus, the threshold distance required to calculate the discovery rate when compared to the grid is equal to the corresponding precision of the stopping condition for each test example in this article. Theoretically, if the distance value returned by the function represents how far away from the initial set stands the new scenario proposed by the algorithm, and this distance is decreasing along the iterations until reaching a given precision, then the algorithm managed to explore the search space with respect to this precision. Therefore, if we assimilate the search space as a full grid, then we are evaluating whether the *NG* scenarios introduced by the algorithm are close to the *NG* grid scenarios by that same precision used by the algorithm as stopping condition. In that way, the discovery rate of the algorithm towards the grid is assessed. Hence, we can see in Fig. 4 that the mean discovery rate is increasing quickly during the first iterations, and continues to increase more slowly but steadily as the iterations go by, until reaching values almost equal to 90% and higher in all examples; it means that the algorithm succeeds in exploring the input search space. We also notice that the standard deviation values are the highest during the quick increase, corresponding to maximum values of less than 4% for all test cases, then begin to decrease throughout the iterations due to the reduced model that is gaining in accuracy.

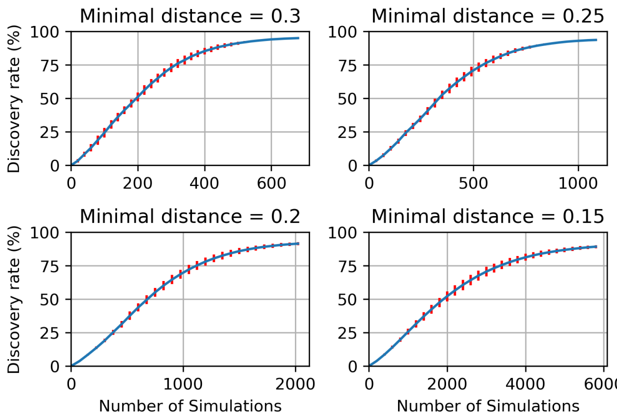

Fig. 4. Variation of the mean discovery rates of the algorithm according to the number of simulations where the vertical error bars are equal to twice the standard deviation values. Each curve is the result of 11 runs stemming from different initial sets. Four stopping conditions are represented: minimal distance equal to 0.3 (top left), 0.25 (top right), 0.2 (bottom left) and 0.15 (bottom right).

The main idea to reflect upon, however, is that, because of the slower increase at high discovery rates, the highest the target discovery rates, the more we need simulations. Plus, if we compare the number of simulations between all test cases, we notice that the lower the precision of the stopping condition, the more simulations are needed in order to reach the same mean discovery rate. For instance, if we look at the algorithm performances in Table II, 721 simulations are required in average for a precision of 0.3 to reach discovery rates almost equal to 90% and higher, against 1,161, 2,180 and 6,186 for precisions of 0.25, 0.2 and 0.15 respectively. Thus, because the key restriction of our study is to minimize the use of software simulations, a compromise should be made between attaining the best discovery rates, refining the search of failures, and calling the simulation software as little as possible for the aim of finding the maximum amount of failures.

V. Conclusion

This paper has introduced a methodology to improve the coverage of the simulated scenarios in order to validate the command law of an autonomous vehicle. Its aim is to detect the maximum number of failures of its system during realistic virtual simulations. It has been experimentally validated on a tracking vehicle use case. Three criteria have been taken into consideration to determine whether there is a failure or not: the lateral lane decentering distance, the longitudinal deceleration and the safety time gap. The proposed algorithm has been designed to explore the input search space of the use case scenarios in order to detect the faulty ones.

A crucial goal of our study was to minimize the number of actual simulations needed, which is more efficient for industrial project monitoring purposes. Thus, a reduced model based on a random forest model is fitted into a small initial set and used by the algorithm as a proxy to detect faulty scenarios during each iteration. The simulation software only evaluates a posteriori the optimal scenarios for the distance objective that have been predicted faulty by the reduced model.

The algorithm iterates until reaching its stopping condition, which corresponds to a given precision for the distance. Because the search space is bounded and continuous, the success of the algorithm is measured by how well it managed to scan the search space effectively: this is achieved by comparing it to scenarios scattered on a full grid. The results show that the algorithm manages to attain high discovery rates. For instance, 6,186 simulations are required in average to reach a discovery rate of almost 90% with a precision of 0.15. The scenarios obtained are evaluated whether they are close within a maximal distance of 0.15 from each scenario of a full grid of 100,000 scenarios. Thus, the methodology presented in this article is able to reduce the number of simulations needed to explore the input search space and detect directly faulty scenarios by comparison to the full grid which requires 100,000 simulations. Nonetheless, a compromise arises between reaching very high rates, which translates into detecting most failures accurately, and the constraint to launch as few software simulations as possible.

For future studies, this algorithm will be tested on other use cases. This will help proving that the algorithm is able to generalize and achieve high discovery rates whatever the use case. Furthermore, other algorithms are being developed under the same methodology, and will be complementary to find the maximum number of failures for the aim of extending this methodology for complete validation of the command law of the autonomous vehicle and ADAS.

TABLE II
Average algorithm performances on 11 runs for each example of stopping condition

Stopping condition minimal distance	**Average algorithm performances**				
	Number of iterations	***Number of simulations***	***Time***	***Mean discovery rate***	***Maximum standard deviation***
0.3	1,968	721	2h49	94.99%	3.58%
0.25	3,113	1,161	5h37	93.62%	3.67%
0.2	5,835	2,180	8h41	91.48%	3.43%
0.15	16,621	6,186	33h23	89.22%	3.89%

References

[1] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," in *SAE Int. J. Trans. Safety*, vol. 4, no. 1, 2016, pp. 15–24.

[2] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" in *Transportation Research Part A: Policy and Practice*, vol. 94, 2016, pp. 182–193.

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

[4] H. J. Vishnukumar, B. Butting, C. Müller, and E. Sax, "Machine learning and deep neural network - artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation," in *IntelliSys*, 2017, pp. 714–721.

[5] H. Beglerovic, M. Stolz, and M. Horn, "Testing of autonomous vehicles using surrogate models and stochastic optimization," in *ITSC*, 2017, pp. 1–6.

[6] AVSimulation, *SCANeR Studio*, Boulogne-Billancourt, France.

[7] G. Breyer, "Safe distance between vehicles." Conference of European Directors of Roads (CEDR), 2010.

[8] M. Johnson, L. M. Moore, and D. Ylvisaker, "Minimax and maximin distance designs," in *Journal of Statistical Planning and Inference*, vol. 26, no. 2, 1990, pp. 131–148.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[10] J. Santiago, M. Claeys-Bruno, and M. Sergent, "Construction of space-filling designs using wsp algorithm for high dimensional spaces," in *Chemometrics and Intelligent Laboratory Systems*, vol. 113. Elsevier, 2012, pp. 26–31.

[11] L. Breiman, "Random forests," in *Machine Learning*, vol. 45, no. 1, 2001, pp. 5–32.

[12] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.