Testing of Autonomous Vehicles Using Surrogate Models and Stochastic Optimization

Halil Beglerovic*, Michael Stolz and Martin Horn

***Abstract*—Advancement in testing and verification methodologies is one of the key requirements for the commercialization and standardization of autonomous driving. Even though great progress has been made, the main challenges encountered during testing of autonomous vehicles, *e.g.*, high number of test scenarios, huge parameter space and long simulation runs, still remain. In order to reduce current testing efforts, we propose an innovative method based on surrogate models in combination with stochastic optimization. The approach presents an iterative zooming-in algorithm aiming to minimize a given cost function and to identify faulty behavior regions within the parameter space. The surrogate model is updated in each iteration and is further used for intensive evaluation tasks, such as exploration and optimization.**

I. Introduction

Autonomous driving is one of the most anticipated emerging technologies in the automotive industry today. The big economic impact of autonomous vehicles, predicted by various studies, is evident as the major automotive companies are adjusting their portfolio in order to prepare for the future. For the commercialization and standardization of autonomous vehicles, rigorous criteria on safety and reliability have been set. As addressed by Winner *et al.* [1], autonomous vehicles are expected to drive over 200 million kilometers to prove that they are at least not worse than human drivers on highways. This verification approach is unfeasible and can not be used for development or system approval. However, current methodologies used for verification of autonomous driving components mainly rely on real world and proving ground testing. In order to extend testing on real roads, Wachenfeld *et al.* [2] proposed a redundant parallel system running alongside the driver, which uses the sensor information in order to generate adequate maneuvers. The maneuvers generated by the autonomous systems are not executed. Instead, they are verified and evaluated directly against the behavior of the driver. Even with these methodologies, the complexity of various traffic scenarios, together with numerous variations in the environment, pose clear limitations in conducting exhaustive real world tests. Other issues, that arise when conducting real world testing, are setup costs and repeatability, as it is hard to recreate identical scenarios multiple times. In addition, the need to cope with challenging environment influences and various scenario types is increasing the complexity of algorithms dealing with autonomous driving, *i.e.*, decision-making, perception, predicting human influence etc. This increased complexity presents new challenges for testing because as the complexity of systems goes up so does the effort to test those systems.

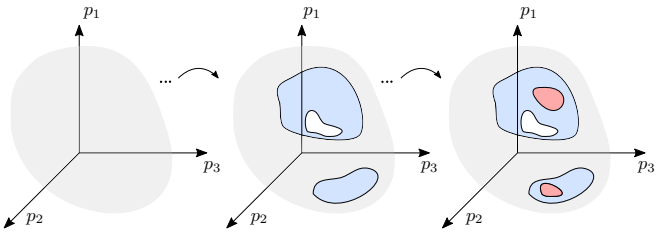

Fig. 1. Iteratively locating a faulty behavior region inside the parameter space

A powerful tool that can address some limitations of real world testing is simulation. Simulation allows reuse of existing scenarios and ensures repeatability in each simulation run. Several research groups have addressed the state-of-the-art topics on autonomous vehicle testing using simulation or mixed virtual and real setup. Stellet *et al.* [3] focus on the taxonomy and high level approach, problem statement and requirements regarding the testing of autonomous vehicles. They propose a systematic methodology for testing, elaborating on the need for properly defined metrics, references and scenarios. Huang *et al.* [4] present an overview on current methodologies, tools, platforms and proving grounds used for testing of autonomous vehicles. Zofka *et al.* [5] [6] [7] and Jemma *et al.* [8] focus on developing new simulation frameworks and validation methodologies for Advanced Driver Assistance Systems (ADAS). Sieber *et al.* [9] conducted research on driver perception and reactions when avoiding collisions, and discussed the implications on the development of ADAS systems.

Kapinski *et al.* [10] state that current verification methodologies can not prove that a hybrid system satisfies formal specifications, as it is not possible to predict their behavior. Automated vehicles, in particular their complex decision-making algorithms, exhibit such hybrid behavior. As it is not possible to test an autonomous vehicle in all possible scenarios under all possible conditions, existing methods for testing rely on the engineer's expertise knowledge to find critical situations with faulty behavior. Usually, full system simulations are time-consuming, and it is not affordable to have many runs, leaving most of the search space not covered. In addition, without an a priori knowledge of all possible interactions between

Halil Beglerovic is with AVL List GmbH, Hans-List-Platz 1, 8020 Graz, Austria, halil.beglerovic@avl.com

Michael Stolz is with Virtual Vehicle Research Center, Inffeldgasse 21a, 8010 Graz, Austria, michael.stolz@v2c2.at

Martin Horn is head of Institute of Automation and Control at Graz University of Technology, Inffeldgasse 21b, 8010 Graz, Austria, martin.horn@tugraz.at

components, it is hard to determine parameters or initial states which will yield faulty behavior.

In this paper, we propose an iterative approach that guides the testing towards faulty behavior regions in the parameter space as shown in Fig.1. In order to identify the faulty behavior regions, appropriate cost functions, which can be minimized by various optimization methods, must be defined. As we are interested in the region around the worst behavior, and we want to avoid false positives in the form of local minima, global optimization methods are needed. One drawback of global optimization algorithms is that they require many function evaluations (simulation runs) to find the global optimum. To overcome this limitation in the proposed approach, we create a surrogate model with inexpensive evaluations on which the optimization algorithms can run. With each new iteration a better model of the system is built around the faulty region. The advantage of this method is that it can handle black box systems with appropriate feedback cost function. In summary, the main contribution of this paper is the development of an approach based on surrogate modeling and stochastic optimization used for the testing of autonomous vehicles.

The paper is structured as fallows. Section II summarizes the related work with focus on optimization based testing. Section III recapitulates the formal problem statement, while the main approach is explained in section IV. Surrogate modeling and optimization are addressed in section V. The test scenario and evaluation is explained in section VI. Finally, a summary and conclusion are given in sections VII.

II. Related work

Tuncali *et al.* [11] have proposed an approach based on stochastic optimization techniques to automatically generate test cases that lead to collisions. They start by sampling the parameter and input space and generate an initial state configuration. After selecting a proper robustness criteria and using the parameter spaces as an input, the framework is able to minimize a given cost function by applying various optimization techniques and iteratively find faulty behavior regions. The proposed framework is called S-TaLiRO [12]. Kapinski *et al.* [10] have given a great overview and comparison of other similar tools: Breach [13], RRT and S3CAM, that utilize similar optimization methodologies to falsify embedded control system. In the article, they give an in depth state-of-the-art overview and problem statement for testing and verification of ECUs, focusing on current and emerging approaches.

Another method has been proposed by Abdessalem *et al.* [14]. They conducted research on a visual emergency breaking ADAS system detecting pedestrians. The main idea was to use neural networks to model the ADAS behavior and use the model instead of the real simulation to get an evaluation and confidence level output. By using genetic optimization methods, they were driving the system to faulty behavior. In addition, by utilizing the neural network model, they were able to reduce the total number of simulation evaluations leading to faster discovery of faulty behavior. However, one limitation of this method is that an expert knowledge in neural network construction is necessary to build a satisfactory model. Also, the network needs to be trained and validated beforehand with a significant amount of real simulation data, which presents a time-consuming and sometimes unfeasible task.

The main difference between our proposed method and the research described above is that we build a surrogate model during the simulation runs by applying an iterative zooming-in approach. No training, nor a priori knowledge of the system, nor data preparation is needed. In addition, since we are using the surrogate model as the input for the optimization algorithms, the complexity of the real simulation will not directly impact the optimization speed.

III. Problem statement

For the problem statement we are using the notation presented in Kapinski *et al.* [10] and modify parts to better fit our proposed algorithm. Firstly, we denote the model we want to test with $\boldsymbol{M}$. The model $\boldsymbol{M}$ is not necessarily a simulation model; it can also represent a real system running on the vehicle or on a HIL/VIL setup. Furthermore, it is also not limited in complexity as it can represent the whole autonomous driving system or any part of it. Next, we define the parameter space $\boldsymbol{P}$ which contains all the environmental (external) or model (internal) parameters and it represents an infinite search space. Usually, a new set of inputs $\boldsymbol{U}$ is introduced; however, we are going to assume that the input signals can be parameterized and the shape can be varied by changing the adequate parameters, in which case the parameters space $\boldsymbol{P}$ contains $\boldsymbol{U}$, *i.e.*, $\boldsymbol{P} \supseteq \boldsymbol{U}$. Finally, the testing criteria is denoted with ψ.

In general, each model $\boldsymbol{M}$ exhibits certain behavior during the simulation or real world trial. This behavior is denoted as $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{p})$ of the model $\boldsymbol{M}$ with respect to the set of parameters $\boldsymbol{p} \in \boldsymbol{P}$. $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{P})$ represents the behavior of $\boldsymbol{M}$ in respect to all possible variations of parameters in the parameter space $\boldsymbol{P}$. If some behavior $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{p})$ satisfies the criteria ψ, the system is working correctly, and we can write $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{p}) \models \psi$. In contrast, if ψ is not satisfied, we write $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{p}) \not\models \psi$.

As we are not able to test the whole parameter space, either because we do not know all the parameters or because of the complexity of the problem, we need to define $\hat{\boldsymbol{P}}$ a subset of the parameter space $\hat{\boldsymbol{P}} \subseteq \boldsymbol{P}$, on which the test is going to be carried on. To test a system, we need to determine whether $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{p}) \models \psi$ holds. However, this task turns out to be very challenging. The main problems are the curse of dimensionality - testing becomes exponentially more complex with each new introduced parameter, the evaluation complexity and evaluation time. One possible solution, proposed by [10] [11] [14], is to find a set of parameters $\boldsymbol{p} \in \hat{\boldsymbol{P}}$ such that the $\boldsymbol{\Phi}(\boldsymbol{M}, \boldsymbol{p}) \not\models \psi$ *i.e.*, a testing instance where the evaluation criteria is not satisfied. By limiting our search for a specific set of parameters $\boldsymbol{p} \in \hat{\boldsymbol{P}}$, we can vastly improve the speed and avoid exploring regions of the search space $\hat{\boldsymbol{P}}$ that are of no interest.

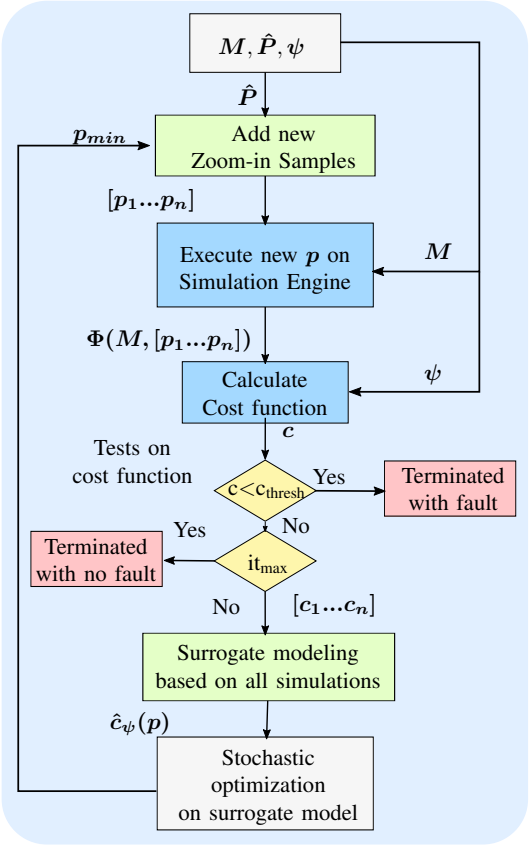

Fig. 2. Workflow of the proposed method

In order to evaluate the behavior $\boldsymbol{\Phi(M,p)}$, we need to introduce some kind of cost function, based on the evaluation criteria $\boldsymbol{\psi}$, which is going to generate numerical evaluations based on the behavioral performance. We can denote such a cost function with $\boldsymbol{c_\psi(\Phi(M,p))}$. By selecting an appropriate cost function $\boldsymbol{c_\psi}$, it is possible to guide the testing towards regions where the behavior is not satisfactory and where the evaluation criterion $\boldsymbol{\psi}$ is not satisfied.

It is also important to note that the selection of an appropriate evaluation criteria $\boldsymbol{\psi}$, cost function $\boldsymbol{c_\psi(\Phi(M,p))}$, parameter space $\boldsymbol{\hat{P}}$ and initial state $\boldsymbol{p_0 \in \hat{P}}$ is a non-trivial task and all of them come with their own challenges. Nevertheless, the outcome of the testing will heavily depend on the quality of the chosen values.

IV. Concept Overview

In the previous section, we gave an overall problem statement and mentioned that a cost function $\boldsymbol{c_\psi(\Phi(M,p))}$ is needed in order to evaluate the behavior of the model $\boldsymbol{M}$. However, because of the complexity of the model or long simulation duration, it is not always feasible to run the simulation and evaluate the cost function directly. This limitation is a big problem if we want to find a global minimum of the cost function, as all the methods searching for a global minimum require many function evaluations. In this paper, we propose an approach that relies on a surrogate model which approximates the cost function $\boldsymbol{c_\psi(\Phi(M,p))}$ with a new function $\boldsymbol{\hat{c}_\psi(p)}$. The new function $\boldsymbol{\hat{c}_\psi(p)}$ takes a parameter set $\boldsymbol{p}$ as input and outputs an approximated numerical value $\boldsymbol{\hat{c} \approx c_\psi(\Phi(M,p))}$ and it is improved iteratively, giving a good representation of the real cost function in the region of interest.

Fig.2 gives an overview of the proposed method. As an input to the algorithm, we need to provide a search space $\boldsymbol{\hat{P}}$ and the *Zoom-in Sampler* is going to make an initial sampling of the space and invoke the *Simulation Engine*. The number of initial samples is defined with the parameter n_s and together with the zooming factor z_f, zooming iteration number z_{in} and sample randomness factor r_f present the only input parameters for the algorithm. *Zoom-in Sampler* will make an initial n_s x n_s grid on the parameter borders and scale it down using the z_f for each iteration. The randomness factor r_f moves the sampling points in the vicinity of the original position, if the simulation is run for more than one time, leading to a better overall approximation. If r_f is set to zero, the *Zoom-in Sampler* samples the space in the exact same values each time. Parameters selection for the proposed algorithm, surrogate modeling and optimization will be discussed later in more detail. The cost function $\boldsymbol{c_\psi(\Phi(M,p))}$ is going to be evaluated for each parameter set $\boldsymbol{[p_1 ... p_n] \in \hat{P}}$ generated by the sampler and an output vector of numerical values $\boldsymbol{[c_1 ... c_n]}$ is going to be computed. For each iteration, we can evaluate the numerical values $\boldsymbol{min\{[c_1 ... c_n]\} < c_{thresh}}$ and decide if a faulty region has been reached. Usually, the cost function $\boldsymbol{c_\psi(\Phi(M,p))}$ can be modeled in such a way that a negative value represents a faulty behavior, *i.e.*, $c_{thresh} = 0$; however, that is not mandatory and any kind of value for c_{thresh} can be used. If the faulty behavior has not been found and if the maximum number of iterations has not been reached, the numerical evaluations of the cost functions $\boldsymbol{[c_1 ... c_n]}$ are passed to the *Surrogate modeling* block. The surrogate model is iteratively extended using the values of $\boldsymbol{[c_1 ... c_n]}$ and provides the approximation function $\boldsymbol{\hat{c}_\psi(p)}$ to the *Stochastic Optimization* where various optimization algorithms can be used. The evaluation of the approximated function $\boldsymbol{\hat{c}_\psi(p)}$ is computationally much cheaper than the original function and is suitable for the extensive evaluations by the optimization algorithms. *Stochastic Optimization* block outputs p_{min} representing the most likely location for the global minimum of the approximated function $\boldsymbol{min\{\hat{c}_\psi(p)\}}$. In the next iteration, the *Zoom-in Sampler* reduces the size of the new sampling set $\boldsymbol{[p_1 ... p_n]}$ using the r_f coefficient and moves the center to the value of p_{min}, effectively zooming into the region with the lowest value of the cost function. A new evaluation is done with the new parameters and a better model of the approximated function $\boldsymbol{\hat{c}_\psi(p)}$ is built until the algorithm reaches a faulty behavior or the maximum number of iterations. After several zooming-in iterations, the size of the sampling set n_s x n_s is significantly reduced and the model accuracy does not increase. In order to further minimize the number of function evaluations, only the most likely location for the global minimum is considered for iterations higher than z_{in} without using the randomness factor r_f.

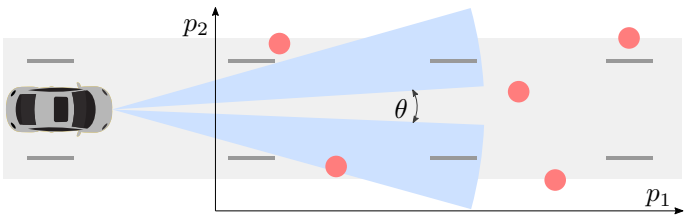

Fig. 3. Scenario setup

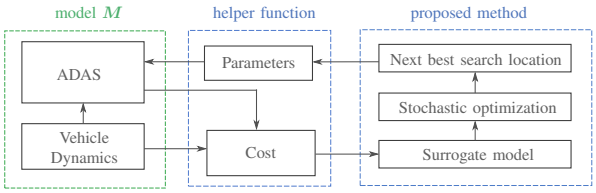

Fig. 4. Simulation setup using MATLAB

V. Surrogate Modeling and Optimization

For the surrogate modeling, we decided to use the RBF - Radial Basis Function approximation. The main idea of RBF is to find an estimation $\hat{f}(x)$ of a real system or process $f(x)$ by sampling the function $f(x)$ in samples $x_i \in X_s$ and assigning a radial basis symmetrical kernel function ϕ for each sample. By adding the influences of each kernel, the approximation can be constructed. The equation of the estimation is given as:

$$\hat{f}(x) = W^T \Phi = \sum_{i=1}^{n_s} w_i \phi(\|x - x_i\|)$$

where $w_i \in W$ represent the weights corresponding to each kernel function. The approximated function has the following properties: $\hat{f}$ is equal to $f(x)$ for $x \in X_s$ and the approximation is worse as the distance from the sample increases. Weights w_i can be obtained by solving the system of linear equations

$$W = \Phi^{-1} Y$$

where $Y = f(x), \forall x \in X_s$ and $\Phi = \phi(\|x_i - x_j\|)\ , i, j = 1...n_s$ where Φ is also called the Gram matrix. For the introduction of RBF, we used notation consistent with literature and the Gram matrix Φ should not be confused with the system behavior $\boldsymbol{\Phi(M, p)}$ from the problem statement.

There are many possibilities when choosing the kernel function for the RBF surrogate modeling. However, in our use case, we tested the Gaussian (ϕ_g) and Multiquadric (ϕ_{mq}) kernel.

$$\phi_g = e^{\frac{-\|x_i - x\|^2}{2\sigma^2}},\ \phi_{mq} = \sqrt{\|x_i - x\|^2 + h}$$

When building a model with a Gaussian or Multiquadric kernel, the user needs to manually assign the coefficients σ or h. The selection of those coefficients is not a trivial task, especially when new zoomed-in sampling points are added in each iteration.

In order to overcome this limitation, we decided to use the Kriging models [15] as they provide a way to use optimization tools in order to find the appropriate coefficients. The Kriging model also uses the Gaussian kernel function ϕ_g but a different coefficient $\gamma = \frac{1}{2\sigma^2}$ and norm p (usually $p \in [1, 2]$) are computed for each parameter. Even though Kriging models are more complex and use more resources for construction, the effort clearly pays off since no tuning is needed. Furthermore, the coefficients γ show which parameter has the higher influence on the cost function. The higher the value of γ for a particular parameter, the higher the influence on the cost function. This can be useful in cases with high number of parameters where the parameters with lower influence could be fixed and excluded from the search.

An additional benefit of using Kriging models is that, instead of searching for the global minimum of the model, we can use a probabilistic approach in order to find the most likely location of the global minimum [15]. The equation that predicts the improvement of sample x is given below:

$$E[I(x)] = (y_{min} - \hat{y})\Phi\Big(\frac{y_{min} - \hat{y}}{\hat{s}}\Big) + \hat{s}\phi\Big(\frac{y_{min} - \hat{y}}{\hat{s}}\Big),$$

where y_{min} is the current minimum, $\hat{y}$ is the model approximation in point x, Φ is the cumulative distribution function, ϕ is the probability density function and $\hat{s}$ is the root mean squared error in point x. The equation $E[I(x)]$ is equal to zero in the sampling points ensuring that by iteratively maximizing $E[I(x)]$ we will eventually find the global minimum.

Beside the most likely global minimum search, the Kriging models require another optimization step for finding adequate parameters γ and p. The value of p was fixed to $p = 2$, as proposed in [15], leading to a simpler optimization task for finding γ. It is important to state that the limitation of all surrogate models is that the cost function needs to be smooth in order to achieve the best modeling results and save computation time.

For the optimization tasks, we use the Differential Evolution (DE) genetic optimization algorithm [16] and Particle Swarm optimization (PSO) algorithm [17] because of their straightforward and simple implementation. It is important to mention that the overall execution time of the proposed algorithm depends strongly on the parameters, i.e number of agents and number of iterations, as the optimizers are used several times for each iteration.

VI. Case Study

In order to validate the proposed method, a simple highway scenario was used. The testing is done for an emergency brake assist ADAS system. The scenario consists of a passenger car driving on the highway and encountering an obstacle in its path. The goal of the ADAS system is to avoid collisions by braking, *i.e.*, no evading maneuvers are used. To prove that our system is able to detect faulty behavior inside the parameter space, we introduce an error in the sensor's field of vision.

A. Scenario description

A detailed overview of the scenario can be seen in Fig.3. The vehicle starts from a still stand and accelerates with constant acceleration until it reaches a maximum velocity of $100\frac{km}{h}$, moving along the x axis. Simulation duration is 10 seconds at which the car reaches $128\ m$. The parameter space $\hat{P}$ is represented by the obstacle's (x, y) coordinates, leading to a 2D search space. Static obstacles are placed on the vehicle's path within the following boundaries: $\hat{P}_{min} = [25,\ -12]$, $\hat{P}_{max} = [165,\ 12]$. The distance at which an obstacle is detected is fixed to $d_{sensor} = 20\ m$ and the sensor's detection angle is $\alpha = 30°$ ranging from $[-15°, 15°]$. An error $\theta = 1,5°$ is introduced ranging from $[1°,\ 2.5°]$, and a search for the worst case crash is going to be conducted.

B. Simulation setup

For the simulation setup Matlab and Simulink [18] are used. The vehicle dynamics and ADAS model represent the model $\boldsymbol{M}$ from the problem statement. The simulation setup overview can be seen in Fig.4. In order to use the proposed approach, only a helper function needs to be available to run the simulation model with parameters $p \in \hat{P}$ and receive back the evaluated cost function $\boldsymbol{c : c_\psi(\Phi(M, [p_1 ... p_n])}$.

C. Cost function selection

As discussed before, the selection of the cost function $\boldsymbol{c_\psi}$ is not a trivial task. Nicolao *et al.* [19] have introduced an approach where they propose a risk assessment evaluation based on the probability that a collision with a pedestrian will occur. Ferrara *et al.* [20] have used a collision cone approach where they explore necessary and sufficient conditions for a collision to occur. Some good practices that can be considered when constructing a cost function are: smoothness - as it will enable a better approximation when using surrogate modeling and convexity - this will ensure quicker convergence to the global minimum. For our case study, we are going to use a cost function based on the time to collision ttc between the vehicle and obstacle. Additionally, the vehicle speed v_{veh} is added in order to give a higher cost value to the obstacle that is out of the sensor range and poses no threat to the vehicle. Crashes that occur at higher speed have lower evaluation value then crashes at lower speeds. Similarly, test runs with no crash at all will lead to higher valued cost. We used

$$\boldsymbol{c_\psi} = \begin{cases} min(ttc + v_{veh}),\ ttc = \frac{d - d_{min}}{v_{veh} + \varepsilon} & , no\ collision \\ -v_{veh} & , collision \end{cases}$$

where ε is a user defined value used to evaluate ttc when the velocity is equal to zero. By minimizing the cost function, we aim to find the most severe crash conditions.

D. Evaluation

In order to evaluate the results obtained from our algorithm, we have simulated the same scenarios using the optimization algorithms DE and PSO directly on the simulation model, without building the surrogate. The aim of this paper has been to reduce the overall number of function calls as they can have a long execution time or could be expensive regarding computation or other resources. Since we are dealing with stochastic optimization algorithms, we can not be certain on the number of function calls needed to find a global minimum. Therefore, we have conducted our experiment for 100 runs and gathered the average values for comparison. In addition, the functions and parameter spaces are equal for all methods used. Because the optimization algorithms do not save states between evaluations, we built the surrogate model for every test run, *i.e*, the samples were not saved between the test runs, even though, in a real scenario, that would be beneficial. The number of function evaluations was limited for all algorithms and was set to 100 evaluations per test run. The optimization algorithms were limited to use 5 agents and 19 iterations with an additional 5 evaluations for the initialization. Coefficients in the DE algorithm were set to $c = 0.5$ for the mutation and $f = 0.5$ for the crossover. For the PSO, the parameters were set to $c_1 = 1.05$ for cognitive, $c_2 = 1.05$ for social and $w = 1 \rightarrow 0.1$ for the particle speeds. The parameters were chosen taking in account advice given in [21] and [17], respectively.

TABLE I
EVALUATION RESULTS

	f-avrg	f-best	g-best	found	a-f-call	time
Kri	-5.27	-11.43	111.2, 0.24	97%	42.67	1785s
R_g	-2.70	-13.78	106.7, 0.20	70%	57.70	689s
R_{mq}	-3.29	-12.04	112.7, 0.23	84%	51.24	901s
PSO	-5.08	-12.19	99.18, 0.21	86%	53.70	623s
DE	-2.37	-13.03	92.20, 0.20	67%	59.90	345s

Our proposed method was running using $n_s = 3$ leading to 9 samples per zooming iteration for iterations below $r_{in} = 4$ and was limited to maximum 100 function evaluations. The zooming-in factor was set to $\gamma = 0.35$ and the random factor was set to $\eta = 0.1$. In general, a higher value of n_s leads to better surrogate model but it will also lead to higher number of required real function evaluations per iteration. The zooming-in factor γ should be chosen depending on the cost function. If the cost function is convex then a lover value is better as it will lead to faster convergence and a good interpolation around the faulty behavior. However, if the cost function is unknown and may have many local minima, a higher value is recommended as the model will have better overall approximation of the cost function. If several test runs are possible, it is better not to sample the model at the same location and reasonable values for the random factor are $\eta \in [0, 0.5]$.

For the surrogate model optimization tasks, we used the PSO optimization algorithm. However, because we are not calling the objective function and the computation is reasonably inexpensive, we used a higher number of agents (15) and iterations (15).

The experiment was repeated for 100 times within the parameter space $\hat{P}_{min} = [25,\ -12]$, $\hat{P}_{max} = [165,\ 12]$. The

testing criteria is $\psi : ttc \leq 0$ that collision has occurred. Table I presents the obtained results. The columns of the table are respectively: average evaluated global minimum; the best global minimum; position of the best global minimum; percentage of successfully found crashes from all test runs, *i.e.*, Krigin model found crashes in 97 out of 100 runs; average number of simulation evaluations per test run; and computation time for all test runs. The rows show the results for the Kriging, RBF with Gaussian and Multiquadric kernel, DE and PSO optimization algorithms.

We can conclude that the Kriging model managed to find test cases with crashes with higher probability while using less simulation evaluations. The higher computation time is directly related to the two optimization steps needed in each iteration; however, in our setup, one simulation run lasted on average 0.127s, and the benefits of a reduced number of function calls could be seen on simulations with longer duration.

VII. Summary and Conclusion

In this paper we introduced an iterative testing approach for autonomous driving, focusing on finding faulty behavior inside the parameter space. Because of the simulation duration or complexity, it is not always feasible to run the optimization algorithms directly on the system. We proposed an approach where a computationally inexpensive surrogate model of the system behavior is built, and optimization algorithms are then applied on the surrogate and not the real system. For the surrogate modeling we used the Radial Basis Function approximation, and we have explored models with different kernel functions. For the optimization tasks the Differential Evolution and Particle Swarm Optimization were implemented. The testing evaluation was conducted on a highway scenario and an Emergency Breaking Assist ADAS. The scenario consisted of a passenger car driving in a straight line and an obstacle position was varied. An error in the sensor's field of vision was introduced and the task of the algorithms was to find the test case with the worst crash evaluation.

The aim of the research was to show that we can reduce the number of real system evaluations, if we first build a surrogate model and then run the optimization algorithms on the surrogate and not on the real system. In the test runs, the maximum number of real system evaluations was fixed for all algorithms and the simulations were repeated for 100 times and the average outcomes were compared. In the end, we have shown that the Kriging model produced the best results leading to a lower number of real system evaluations and a good approximation inside the faulty region.

Acknowledgment

The project leading to this application has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No. 675999, and partially supported by the Austrian COMET K2 - Competence Centers for Excellent Technologies Programme.

References

[1] H. Winner, S. Hakuli, F. Lotz, and C. Singer, *Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort*. Springer, 2016.

[2] W. Wachenfeld and H. Winner, "Virtual assessment of automation in field operation - a new runtime validation method," in *10. Workshop Fahrerassistenzsysteme.*, Walting im Altmühltal, September 2015.

[3] J. E. Stellet, M. R. Zofka, J. Schumacher, T. Schamm, F. Niewels, and J. M. Zllner, "Testing of advanced driver assistance towards automated driving: A survey and taxonomy on existing approaches and open questions," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sept 2015, pp. 1455–1462.

[4] W. Huang, K. Wang, Y. Lv, and F. Zhu, "Autonomous vehicles testing methods review," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2016, pp. 163–168.

[5] M. R. Zofka, S. Klemm, F. Kuhnt, T. Schamm, and J. M. Zllner, "Testing and validating high level components for automated driving: simulation framework for traffic scenarios," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016, pp. 144–150.

[6] M. R. Zofka, R. Kohlhaas, T. Schamm, and J. M. Zllner, "Semivirtual simulations for the evaluation of vision-based adas," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 121–126.

[7] M. R. Zofka, F. Kuhnt, R. Kohlhaas, C. Rist, T. Schamm, and J. M. Zllner, "Data-driven simulation and parametrization of traffic scenarios for the development of advanced driver assistance systems," in *2015 18th International Conference on Information Fusion (Fusion)*, July 2015, pp. 1422–1428.

[8] I. B. Jemaa, D. Gruyer, and S. Glaser, "Distributed simulation platform for cooperative adas testing and validation," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2016, pp. 77–82.

[9] M. Sieber and B. Frber, "Driver perception and reaction in collision avoidance: Implications for adas development and testing," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016, pp. 239–245.

[10] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems*, vol. 36, no. 6, pp. 45–64, Dec 2016.

[11] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, "Utilizing s-taliro as an automatic test generation framework for autonomous vehicles," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2016, pp. 1470–1475.

[12] A. Donzé, *Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 167–170.

[13] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, *S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 254–257.

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

[15] A. Forrester, A. Sobester, and A. Keane, *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[16] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[17] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[18] MATLAB, *version 9.1.0.4 (R2016b)*. Natick, Massachusetts: The MathWorks Inc., 2016.

[19] G. D. Nicolao, A. Ferrara, and L. Giacomini, "Onboard sensor-based collision risk assessment to improve pedestrians' safety," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 5, pp. 2405–2413, Sept 2007.

[20] A. Ferrara and C. Vecchio, "Collision avoidance strategies and coordinated control of passenger vehicles," *Nonlinear Dynamics*, vol. 49, no. 4, pp. 475–492, 2007.

[21] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, Feb 2011.