Utilizing S-TaLiRo as an Automatic Test Generation Framework for Autonomous Vehicles

Cumhur Erkan Tuncali, Theodore P. Pavlic and Georgios Fainekos

***Abstract*— This paper proposes an approach to automatically generating test cases for testing motion controllers of autonomous vehicular systems. Test scenarios may consist of single or multiple vehicles under test at the same time. Tests are performed in simulation environments. The approach is based on using a robustness metric for evaluating simulation outcomes as a cost function. Initial states and inputs are updated by stochastic optimization methods between the tests for achieving smaller robustness values. The test generation framework has been implemented in the toolbox S-TaLiRo. The proposed framework's ability to generate interesting test cases is demonstrated by a case study.**

I. Introduction

Driver Assistance Systems (DAS) like lane keeping, adaptive cruise control, pedestrian/obstacle collision avoidance, automatic lane change, emergency braking systems and many more are being used in high-end modern automotive systems. Prototype autonomous vehicles have already driven more than a million miles on the roads. However, ensuring safe operation of these vehicles will require intensive testing in various scenarios, which will be a major challenge. As addressed by Bengler *et al*. [1], testing autonomous vehicles is a challenging problem which can not be efficiently handled with conventional approaches. According to Maurer and Winner [2], considering the pace of functional growth in DAS, lack of efficient testing can affect time-to-market for more advanced systems like fully autonomous vehicles.

Because tests involving autonomous vehicles would comprise test cases which may lead to collisions or near collisions, performing many tests with actual vehicles ending with collisions would not be economically efficient and practical. An alternative for testing these systems is using computer simulations for the vehicles and their surroundings. However, manually creating test scenarios with large number of different environmental settings and road conditions would be difficult and highly time consuming. Furthermore, in order to extract the limits of the systems under design, engineering teams would like to discover the behaviors on the boundary between safe and unsafe operations. Creating test cases manually for detecting the boundary conditions which barely cause collisions like fender-benders may be a challenging job. We believe that automatic test generation frameworks utilizing simulations are crucial for the future of autonomous vehicle testing. Such frameworks would produce large number of tests generated in an intelligent way and help engineering teams to discover unforeseen scenarios that could lead to failure.

Although discovering potential problems by testing in the simulation environments would be beneficial, due to the inevitable differences in the simulation environments and the real-world, some test cases that do not fail in simulations may fail in the real-world or vice versa. So, instead of getting pass/fail results from the tests, using a metric that indicates how close each test result is to a failure case, similar to phase and gain margins in control theory [3], would be more useful. With availability of such a metric, engineering teams can run large number of tests in faster simulation environments and after (automatic) analysis of the test results, they can repeat some scenarios in the real world using the methods described in [4] or in much more accurate simulators that require more computation power/execution time.

In this paper, we address the problem of creating an automatic test generation framework for autonomous vehicle systems with a focus on collisions. Because of the dynamics and possible physical limitations on the motion, an autonomous vehicle cannot be expected to be collision free for every possible situation. For instance, it may not be possible to avoid a collision with a vehicle cutting in front at a very short distance or with a vehicle approaching very fast from a side. Our main focus is to find the conditions on the boundary between safe scenarios and collision scenarios. Our approach is based on running simulations, using simulation results to compute a robustness value that shows how close a system trajectory gets to an unsafe set of states and utilizing optimization methods to seek smaller robustness values by changing initial states and inputs for the system for the next test case.

II. Related Work

State of the art in testing advanced DAS is discussed by Stellet *et al*. [5]. They categorize the main principles for a testing framework as: (i) derivation of test criteria and metrics, (ii) establishing a reference system as the ground truth information and (iii) generation of test scenarios. The use of robustness metric in our approach falls into the category (i) per their taxonomy. Simulation environment itself can be considered as the category (ii), and the optimization engine for S-TaLiRo that is used to create test trajectories can be considered as the category (iii) based on the discussions in that work. Our robustness metric definition can be a candidate to quantitatively measure safety of autonomous

This work has been partially supported by awards NSF CNS 1446730 and NSF CNS 1350420

C.E. Tuncali, T.P. Pavlic, and G. Fainekos are with School of Computing, Informatics and Decision Systems, Arizona State University, Tempe, AZ 85281, USA
(e-mail:{etuncali, tpavlic, fainekos}@asu.edu)

vehicles against collisions. S-TaLiRo provides methods for automatic, high throughput testing of fully autonomous vehicles inside simulations that can cover complex real-world traffic situations.

A vehicle in the loop (VIL) test setup is presented by Bock *et al*. [6]. They discuss the advantages of using simulators for testing DAS. Their approach is based on having a human driver in a simulator and using conventional methods for generation of test cases.

Althoff *et al*. [7] propose an online formal verification approach for autonomous vehicles. The approach proposed in that work is based on reachability analysis for the ego vehicle, *i.e*., the vehicle under control, and other participants on the road. They compute the reachable sets and occupancy of the ego vehicle for reference trajectories and claim that the reference trajectory is safe if occupancy of the ego vehicle does not exit drivable area nor intersect with the other participants on the road. They assume that other participants obey the traffic rules. However, in [7], they are only considering verification of planned trajectories and executing the trajectories that are deduced to be safe. In contrast, the test results with our approach give an idea on how the mistakes of other vehicles in the traffic are handled by the ego vehicle which is a valid concern for traffic environments consisting of both autonomous and non-autonomous vehicles.

Our approach is complementary to the VIL testing, online verification and testing with real vehicles [4], as it will suggest important and challenging test-cases for existing methods.

III. Problem Definition

We denote the set of autonomous Vehicles Under Test (VUT) by $\mathbf{V} = \{v_1, \ldots, v_p\}$, surroundings for the VUT by $\mathbf{S} = \{s_1, \ldots, s_q\}$, and the set of dummy actors by $\mathbf{D} = \{d_1, \ldots, d_r\}$. In more detail, the vehicles in the set $\mathbf{V}$ are simply Cyber-Physical Systems (CPS) which are to be tested either partially, *e.g*., controller only, or as a whole. The surroundings $\mathbf{S}$ consist of the environmental settings like road network, weather and road conditions, traffic rules and special zones. The set $\mathbf{D}$ of dummy actors, *i.e*., the mobile or immobile objects in the surroundings, may contain dummy vehicles, pedestrians, obstacles, etc., optionally with controllers that can give them mobility. The dummy actors physically interact with the VUT and they are typically used to trick the VUT in various test scenarios.

The initial state vectors for the VUT in $\mathbf{V}$, the surroundings $\mathbf{S}$ and dummy actors $\mathbf{D}$ are respectively denoted by $\mathbf{x}_{0,\mathbf{V}}$, $\mathbf{x}_{0,\mathbf{S}}$, and $\mathbf{x}_{0,\mathbf{D}}$. The range of initial values for the i^{th} state of the k^{th} entity in a set $\mathbf{V}$, $\mathbf{S}$, or $\mathbf{D}$, is denoted by $R_{v_k^i}$, $R_{s_k^i}$, $R_{d_k^i}$, respectively. These ranges define the domains for the initial state vectors. For instance, $\mathbf{x}_{0,\mathbf{V}} \in \prod_{k=1}^{|\mathbf{V}|} (R_{v_k^1} \times \ldots \times R_{v_k^{\sigma_k}})$, where $\prod$ and $\times$ denote Cartesian product, $|\mathbf{V}|$ is the number of elements in $\mathbf{V}$ and σ_k is the number of states for $v_k \in \mathbf{V}$. We denote the concatenation of the initial state vectors $\mathbf{x}_{0,\mathbf{V}}$, $\mathbf{x}_{0,\mathbf{S}}$, and $\mathbf{x}_{0,\mathbf{D}}$, *i.e*., the initial state for the overall setup, by $\mathbf{x}_0$.

Abbas *et al*. [8] parameterize input signals $u(t)$ over a bounded time domain R, by the parameter vectors $\lambda = [\lambda_1 \ldots \lambda_m]^T \in \Lambda$, $\tau = [\tau_1 \ldots \tau_m]^T \in R^m$, where Λ is a compact set, $\tau_i < \tau_j$ for $i < j$, such that for all $t \in R$, $u(t) = \mathfrak{U}(\lambda, \tau)(t) \in \mathbb{R}$. The function $\mathfrak{U}(\lambda, \tau)$ returns a function which is parameterized by λ and τ. For instance, $\mathfrak{U}$ could represent the space of functions parameterized using splines [9]. We slightly modify the notation used in that work to allow inputs to be functions of the states of the system, not only signals over time. Hence, we allow the set R in the above notation to be the bounded domain of any variable which can be a state of the system or time. We use the notation $\mathbf{u_V}$, $\mathbf{u_S}$, $\mathbf{u_D}$ for the vectors of input functions for the entities in $\mathbf{V}$, $\mathbf{S}$, $\mathbf{D}$, respectively. Each entry of these input vectors is a tuple $(\lambda, \tau, \mathfrak{U})$, *i.e*., parameterization vectors and a choice of interpolation function, corresponding to an input for an entity of $\mathbf{V}$, $\mathbf{S}$, or $\mathbf{D}$. We denote the input functions for the overall simulation setup by $\mathbf{u}$.

We define a simulation over $\mathbf{V}$, $\mathbf{S}$ and $\mathbf{D}$ as a function $\Sigma_{\mathbf{V},\mathbf{S},\mathbf{D}} : \bar{X}_0 \times \bar{\Lambda} \times \bar{R} \mapsto \mathbb{R}^{n \times k}$ where $\bar{X}_0$ defines the domain for the initial states, $\bar{\Lambda}$ and $\bar{R}$ define the domain for the λ and τ parameters of all the input functions respectively, k is the number of simulation steps and n is the total number of outputs generated by the elements of $\mathbf{V}$, $\mathbf{S}$ and $\mathbf{D}$.

The problem we target is to compute:

$$(\mathbf{x}_0^*, \bar{\mathbf{u}}^*) = \underset{\mathbf{x}_0 \in \bar{X}_0, \bar{\lambda} \in \bar{\Lambda}, \bar{\tau} \in \bar{R}}{\arg\min} \mathcal{R}\big(\Sigma_{\mathbf{V},\mathbf{S},\mathbf{D}}\big(\mathbf{x}_0, \bar{\mathbf{u}}(\bar{\lambda}, \bar{\tau})\big)\big) \quad (1)$$

where $\mathcal{R} : \mathbb{R}^{n \times k} \mapsto \mathbb{R}$ is defined as a robustness (cost) function, $\bar{\lambda}$ is the vector of λ parameters and $\bar{\tau}$ is the vector of τ parameters for all the inputs. The vector $\bar{\mathbf{u}}$ contains all input functions, and it is obtained by applying the interpolation function $\mathfrak{U}(\lambda, \tau)$ for each input to the corresponding parameter vectors λ, τ from the selected vectors $\bar{\lambda}, \bar{\tau}$.

In other words, for a given simulation function, a set of vehicles under test, a set of dummy objects, surroundings information and constraints on state space and input space, we are seeking the particular inputs and initial states for the simulation that would minimize a robustness function.

Careful selection of a robustness function is important for (quickly) finding initial states and input signals which lead to critical operating points of the system under test like boundaries between safe and unsafe behavior.

IV. Solution Overview

A simplified overview of the architecture of our approach is illustrated in Fig. 1. The main components of the vehicular systems testing framework that we propose are the optimization engine of S-TaLiRo, a simulation engine, a robustness evaluation function and a simulation configuration.

A. S-TaLiRo

S-TaLiRo [10] is a MATLAB [11] toolbox for systematic testing of hybrid systems, *i.e*., the systems that exhibit continuous and discrete dynamics. It uses a robustness metric that represents how far a system trajectory is from falsifying formal system requirements. In particular, negative robustness

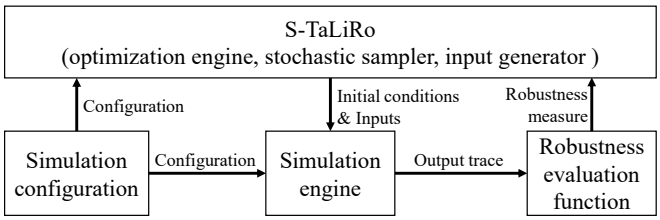

Fig. 1. An Overview of the Framework Architecture

values mean a requirement is falsified, *i.e.*, conditions have been found under which the system does not satisfy a requirement. S-TaLiRo uses one of various global optimization methods for minimizing the robustness function [10] and, thus, for seeking a falsifying system trajectory. We utilize S-TaLiRo for solving the problem defined in Section III, basically for intelligently sampling initial states and input functions that will be applied to the simulations.

First, a sample space is created from the user defined input and/or initial state configuration. Then, an initial states vector and an input functions vector are sampled from the generated sample space. The simulation of the vehicular systems is executed for a predefined amount of time with the selected initial states and inputs. As illustrated in Fig. 1, the simulation engine returns the simulation output trajectory which consists of states and/or outputs of the simulated system(s) for each time step of the simulation. The output trajectory obtained from the simulation is supplied to the robustness evaluation function that returns a real-valued robustness measure as an evaluation of how close the simulation results are to an unsafe set of states. The obtained robustness measure is used by the optimization engine and the stochastic sampler in S-TaLiRo for generation of the inputs and initial states for the next simulation with an attempt to obtain smaller robustness values. This cycle of input generation, simulation and robustness evaluation continues until either a negative robustness value is achieved or the maximum number of simulations is reached which we use as the termination conditions for the optimization problem given in (1).

B. Simulation Configuration

Simulation configuration is used to parameterize a wide range of classes of systems and scenarios. The automated test generation proceeds by sampling points from this parameterized space as explained above. A test scenario can be described in a simulation configuration with a focus on a function with different types of systems and environments.

Referring to the definitions given in Section III, a simulation configuration is basically a structure describing the sets $\mathbf{V}$, $\mathbf{D}$, $\mathbf{S}$, and the initial state and input spaces $\bar{X}_0$ and $\bar{\mathcal{U}} = \bar{\Lambda} \times \bar{R}$. The space for the initial states is described by supplying the ranges for each initial state, *e.g.*, $R_{v_i^{\sigma_k}}$, and the space for the inputs is described by supplying the parameterization, *i.e.*, $(\lambda, \tau, \mathfrak{U})$ as detailed in Section III. The simulation configuration can further constraint the relations between the initial states and inputs.

A typical configuration contains environmental parameters, the number of vehicles in the simulation and parameters for each vehicle. Some examples to the environmental parameters could be wind, road incline, lane width, number of lanes, inputs and states of the environment. Vehicle-related parameters can be mass, tire-friction, ranges of initial states and inputs, function handles that describe dynamics of these vehicles or controllers for the vehicles. These are only some examples and the actual parameters must be completely defined by the user in accordance with the user supplied simulation engine and the robustness evaluation function.

C. Simulation Engine

The simulation engine can be a MATLAB function, a SIMULINK [11] model or any external simulator like WEBOTS [12] or CARSIM [13] that can be wrapped by a MATLAB function. The simulation engine must be able to accept inputs described in the configuration, and it must return the computed states and/or outputs for each time step of the simulation. Because the simulation configuration is available to the simulation engine, the user can freely parameterize the simulation engine in the desired level of detail in accordance with the testing purposes.

Here, we recall the definition of a simulation function given in Section III as $\Sigma_{\mathbf{V},\mathbf{S},\mathbf{D}} : \bar{X}_0 \times \bar{\mathcal{U}} \mapsto \mathbb{R}^{n \times k}$. In summary, the simulation engine first initializes the models and/or controller functions of the VUT in the set $\mathbf{V}$, surroundings $\mathbf{S}$ and the dummy actors $\mathbf{D}$ with given initial states. Then, it executes these models/controllers with respect to the given inputs while considering the interactions of the entities in the above sets with each other.

D. Robustness Evaluation Function

The robustness evaluation function is used as a cost function in the optimization engine of S-TaLiRo. It can either be supplied by the user or used from robustness computation implementations with respect to temporal logic specifications available in S-TaLiRo [14].

In a simulation setup, as the number of variable parameters increase, the space created by these parameters can be very large. Testing every combination over such a large space and finding falsifying behaviors is infeasible in most cases. Because S-TaLiRo is based on optimization over robustness, the obtained robustness values from different simulations are expected to guide the search towards a failure as opposed to completely random selection of test cases. It should be noted that the choice of the robustness evaluation function plays an important role for better guidance.

A robustness evaluation function must return smaller robustness values as we approach to the most interesting failing system behavior that we seek for. If a negative robustness value is obtained, S-TaLiRo immediately stops and returns the related trajectory as a falsifying trajectory. Otherwise, the search for smaller robustness value over trajectories continues until a given maximum test count is reached.

In this work, we mainly focus on testing autonomous vehicles against collisions in an environment where some

vehicles may follow trajectories that can lead to dangerous situations. Furthermore, we seek the conditions, *i.e.*, initial states and input signals, that lead to near collisions. Hence, we design a robustness evaluation function so that the boundaries between safe and unsafe behavior can be reached by minimizing the robustness function. Here, we will propose a robustness evaluation function that can be applicable to a wide range of setups for testing autonomous vehicles with a purpose of detecting collisions and/or the situations where vehicles exit a predefined drivable area.

For a collision instance between two vehicles with velocities $\vec{v}_1$ and $\vec{v}_2$ at the time of collision, we compute the severity of the collision as $\|\vec{v}_1 - \vec{v}_2\|$, where $\|\cdot\|$ is the Euclidean norm. When a collision involving a VUT is detected in a simulation output trajectory y, we compute $v_{coll,y}$ as the collision severity at the moment of the first collision experienced in y.

If there is no collision involving a VUT in a simulation output trajectory, we use a safety measure called Time-To-Collision (TTC) [15]. The TTC is the time required for two vehicles to collide when they are on a collision path. Being on a collision path for two vehicles means that they will collide if they continue their current motion. In particular, the TTC for two vehicles that are not on a collision path is infinite. We use the looming points approach described by Ward *et al.* [16] for collision path and TTC computations. The minimum TTC experienced between any two vehicles during a simulation trace y is denoted by $ttc_{min,y}$.

Note that we can use collision severity and TTC metrics for testing against a vehicle exiting the drivable area. Considering the boundaries of drivable areas as stationary objects, *e.g.*, a wall, TTC or collision severity with these objects can be computed in a similar manner by taking the velocity of the objects as a zero vector.

Because we search for the boundary between safe and unsafe operations, we can consider a collision where vehicles barely touch each other with zero difference in velocities as the boundary case that we seek. Hence, a collision with high relative velocity between the vehicles must have a larger robustness value compared to a collision with low relative velocity. In addition, a simulation trace with no collision must have larger robustness value compared to a simulation result involving a collision. Our proposed robustness function $\mathcal{R}(y)$ for a simulation trace y is given below:

$$\mathcal{R}(y) = \begin{cases} v_{coll,y} - v_\epsilon & , \textit{collision detected in } y \\ ttc_{min,y} + v_{coll,max} & , \textit{otherwise.} \end{cases} \tag{2}$$

where $v_{coll,max}$ is the maximum possible relative collision velocity and v_ϵ is a user defined nonnegative real-valued number representing the minimum collision severity of concern. Whenever the framework achieves a collision with a severity smaller than v_ϵ, the robustness value will be negative and the search will be terminated. In particular, setting v_ϵ to zero means that we are seeking the collisions with the vehicles barely touching each other. However, in this case, the search will continue until the maximum number of simulations is reached and the detected minimum robustness value will be returned.

We assume that we know maximum possible velocity for all the objects in the simulation which is denoted by v_{max}. The maximum collision velocity in a simulation can be experienced between two vehicles traveling at the maximum speed in opposite directions. Hence, $v_{coll,max} = 2v_{max}$.

V. Case Study

As a case study, we use the simulation engine with the simulation configuration described below and the robustness function in (2). S-TaLiRo is configured to use the simulated annealing method [17].

A. Simulation Configuration for the Case Study

Our case study consists of two VUT in the set $\mathbf{V}$ and a dummy vehicle in the set $\mathbf{D}$ on a straight two-lane road that is described in $\mathbf{S}$. The inputs to the simulation are the target speed functions for the VUT and target speed and lateral position functions for the dummy vehicles. Target speed functions are defined over time, and the target lateral position function is defined over the longitudinal position state of the dummy vehicle.

One of the VUT is following the other on a straight target trajectory on the right lane of a two-lane road. The dummy vehicle has a trajectory which starts on the left lane of the road and changes to the right lane after a distance chosen by the testing algorithm. The target position of the dummy vehicle inside a lane is varying over the course of the simulation and the lane change position is also sampled from a predefined longitudinal position range.

The shape and dimensions of the vehicles are described in the configuration as the critical points, *e.g.*, corners, of the vehicles. These points are used to detect collisions and also used in the looming points method [16] to check collision paths and to compute the TTC values.

B. Simulation Engine for the Case Study

For the simulation of the VUT, we use a vehicle dynamics model from the literature [18], [19]. To accurately represent the dynamics of the VUT, we use relatively complex dynamical models that are costly to compute during simulation. However, it is not computationally practical to use dynamical models of similar complexity for the dummy vehicles that are merely meant to generate reasonable test trajectories to challenge the VUT. Furthermore, the actual controllers on the dummy vehicles will be out of the control of the tester, and so all that is necessary for the dummy vehicles is to capture the salient features of realistic vehicles in simulation. Consequently, for the dummy vehicles, we use simpler kinematic models and controllers. The kinematic model we use for the dummy vehicle in our case study is described by Walsh [20]. We have implemented our simulation engine for the case study as a MATLAB function.

C. Sensor Setup

We describe the sensors on the vehicles by their orientation, range, maximum sensing angle and position with respect to the center of mass of the vehicle. In our case study, we use a sensor setup for side collision avoidance. The vehicles under test have one distance sensor in front with a range of $40\,\text{m}$ and 10° sensing angle and one distance sensor on the left side with a range of $3\,\text{m}$ and 45° sensing angle. The sensor placement and orientation is illustrated in Fig. 2. The rectangle in the figure represents the top view of the vehicle where the tip of the arrow on the rectangle is towards the front of the vehicle. This sensor configuration is used to test the framework's ability to detect possible collisions resulting from the corresponding blind spots.

D. Vehicle Controller with Collision Avoidance

We have implemented a controller with basic forward and side collision avoidance capabilities by merging two controllers from the literature. For steering control, we used the Stanford's Racing Team's approach [21] for the DARPA Grand Challenge 2005. For the longitudinal control, we used the model predictive convoy controller from Liu and Ozguner [22]. A reactive planner generates a target path based on the input target speed and the forward and side distance sensor data. The generated target path and the input target speed are supplied to the controller, which generates force and steering inputs. We use this controller and the simulation setup only for demonstrating our framework, and we do not claim any performance or accuracy guarantees for the controller or the simulator.

We describe the target speed for the VUT as an input signal chosen by the testing algorithm in a predetermined range $[5, 15]\,\text{m/s}$ over the simulation time. The target lateral position for the VUT is the midpoint of the right lane. However, because the VUT controller has collision avoidance capabilities, the target lateral positions for the vehicles are updated during run-time based on the sensor data.

E. Motion Controller for the Dummy Vehicle

The target trajectory for the dummy vehicle is described by two input functions for S-TaLiRo. One input function is the target speed for the vehicle. We define the target speed as a signal with a predefined number of control points, *i.e.*, the parameter τ described in Section III, equally distributed over the simulation time. We set the lower and upper limits, *i.e.*, the domain for the parameter λ described in Section III, for the target speed at each control point. Piecewise cubic Hermite interpolating polynomial (pchip) interpolation [23] function that is available in MATLAB [11] is used for interpolation between the control points, *i.e.*, the $\mathfrak{U}$ described in Section III. Thus, the target speed for the dummy vehicle is a signal interpolated between the values chosen by the test algorithm from a given range.

The other input function for the dummy vehicle describes its target lateral position. We describe this input with respect to the vehicle's longitudinal position state instead of the simulation time. We use 4 control points for this function where the first and the last control points are located at positions $0\,\text{m}$ and $300\,\text{m}$ in the longitudinal axis. The locations for second and third control points are chosen by the test algorithm between these positions with a constraint for the distance between two consecutive control points to be at least $5\,\text{m}$. The value ranges for the control points are the limits of the left lane for the first two control points and the limits of right lane for the remaining control points. This describes a trajectory that starts at the left lane and then changes to the right lane. The lateral position inside the lanes varies between the selected values by the test algorithm.

Fig. 2. Sensor Placement

For the dummy vehicle, as opposed to the VUT, we have implemented a PID controller for tracking the target speed instead of the costly model predictive controller. As discussed in subsection V-B, the controller for the dummy vehicle is only used for roughly following the trajectory proposed by the tester that will be used to challenge the VUT.

F. Experimental Results

As stated in subsection V-C, we intentionally created a blind spot in the sensor setup for the VUT. During our experiments, the framework successfully detected failure cases caused by this weakness. Fig. 3 illustrates a near collision where the VUT (at the bottom-right of the figure) avoids a side collision in the first place and then collides with the dummy vehicle when returning back to its lane. In this case, the VUT first avoids the side collision by changing its lateral position and slowing down. However, after avoiding the side collision it loses track of the dummy vehicle because of the blind spot. Hence, the VUT does not continue slowing down although it should have. Furthermore, it starts making the maneuver to return to its lane. As a consequence, it barely touches the dummy vehicle at its rear-right corner. The final parts of the vehicle trajectories are displayed as traces behind the vehicles in Fig. 3. The second VUT, *i.e.*, the one on bottom-left of the figure, is following the VUT that had a collision. This VUT is far from the collision scene, and it is not affected by the collision. There are additional collisions detected by the framework, and all of them are returned to the user for further analysis; however, this was the collision with the minimum robustness value returned by S-TaLiRo, which makes it an interesting case at a boundary between safe and unsafe operation. The sampled input parameters and the generated input function as the target lateral position of the dummy vehicle leading to the collision is given in Fig. 4. The τ parameters are defined

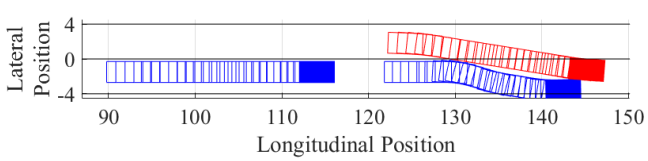

Fig. 3. History of the Vehicles Before a Collision Instance

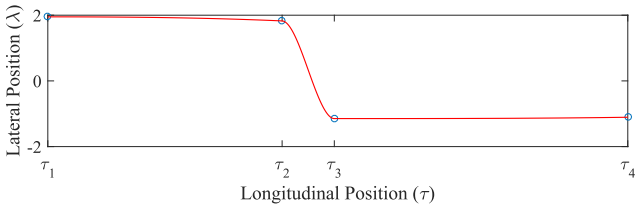

Fig. 4. The Input for Target Lateral Position of the Dummy Vehicle

over the longitudinal position state of the dummy vehicle, and the λ parameters are the target lateral positions for the corresponding τ parameters. For this case study, the (τ, λ) samples that led to the collision of concern are $(0.0, 1.95)$, $(121.2, 1.83)$, $(148.2, -1.15)$, $(300.0, -1.11)$.

We have run our experiments on a Windows® PC with an Intel® Core™ i7-4790 CPU and 16GB RAM. A simulation of 32 s of the described test setup takes 18 s physical time on our setup. The above failure condition was detected in 100 simulations. Even though the convergence to the global minimum robustness value is guaranteed with simulated annealing [17], our stochastic approach provides no guarantee on the number of simulations required to achieve the global minimum. In general, the execution time of one simulation in the proposed framework depends on the complexity of the vehicle ODEs and controllers. The overall worst case execution time for the framework grows linearly with the maximum number of simulations chosen by the user.

VI. Conclusions

We proposed an approach for automatic simulation based testing of autonomous vehicle controllers that is guided by a robustness metric. We believe that the optimization over robustness for test guidance is a promising approach for testing cyber physical systems in general [24].

As a future work, we plan to extend the capabilities of our framework by extracting the conditions leading to unsafe behavior from the simulations and use them for training a model for estimating the probability of future collisions.

The trajectory generation for dummy vehicles in our framework is based on the boundaries described by the user. The generated trajectories are then tracked by the user-supplied controllers. Nagy *et al*. [25] propose a method for generating trajectories for mobile robots that can be easily tracked by a real vehicle. Although the final trajectories followed by the controllers are realistic in our framework, utilizing the approach described in that work can allow using the trajectories directly without the need for a controller to track them. As another future work, we plan to incorporate such a method for trajectory generation.

References

[1] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems: Review and future perspectives," *Intelligent Transportation Systems Magazine, IEEE*, vol. 6, no. 4, pp. 6–22, 2014.

[2] M. Maurer and H. Winner, *Automotive systems engineering*. Springer, 2013.

[3] K.-W. Han and C.-H. Chang, "Gain margins and phase margins for control systems with adjustable parameters," *Journal of guidance, control, and dynamics*, vol. 13, no. 3, pp. 404–408, 1990.

[4] 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, 2015.

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

[6] T. Bock, M. Maurer, and G. Farber, "Validation of the vehicle in the loop (VIL); a milestone for the simulation of driver assistance systems," in *Intelligent Vehicles Symposium, 2007 IEEE*. IEEE, 2007, pp. 612–617.

[7] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *Robotics, IEEE Transactions on*, vol. 30, no. 4, pp. 903–918, 2014.

[8] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 95, 2013.

[9] M. Egerstedt and C. Martin, *Control Theoretic Splines: Optimal Control, Statistics, and Path Planning*. Princeton University Press, 2009.

[10] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 254–257.

[11] MATLAB, *version 9.0.0 (R2016a)*. Natick, Massachusetts: The MathWorks Inc., 2016.

[12] O. Michel, "Webots™: Professional mobile robot simulation," *arXiv preprint cs/0412052*, 2004.

[13] Mechanical Simulation, "CarSim," 2016. [Online]. Available: http://www.carsim.com/

[14] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[15] J. C. Hayward, "Near-miss determination through use of a scale of danger," *Highway Research Record*, no. 384, 1972.

[16] J. Ward, G. Agamennoni, S. Worrall, and E. Nebot, "Vehicle collision probability calculation for general traffic scenarios under uncertainty," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 986–992.

[17] H. Abbas and G. Fainekos, "Convergence proofs for simulated annealing falsification of safety properties," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2012, pp. 1594–1601.

[18] K. Zhang, J. Sprinkle, and R. G. Sanfelice, "A hybrid model predictive controller for path planning and path following," in *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*. ACM, 2015, pp. 139–148.

[19] E. Narby, "Modeling and estimation of dynamic tire properties," Master's thesis, Linkopings Universitet, Linkoping, 2006.

[20] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J.-P. Laumond, "Stabilization of trajectories for systems with nonholonomic constraints," *Automatic Control, IEEE Transactions on*, vol. 39, no. 1, pp. 216–222, 1994.

[21] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *American Control Conference*, 2007, pp. 2296–2301.

[22] P. Liu and U. Ozguner, "Predictive control of a vehicle convoy considering lane change behavior of the preceding vehicle," in *American Control Conference (ACC), 2015*. IEEE, 2015, pp. 4374–4379.

[23] F. N. Fritsch and R. E. Carlson, "Monotone piecewise cubic interpolation," *SIAM Journal on Numerical Analysis*, vol. 17, no. 2, pp. 238–246, 1980.

[24] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-guided approaches for verification of automotive powertrain control systems," in *2015 American Control Conference (ACC)*. IEEE, 2015, pp. 4086–4095.

[25] B. Nagy and A. Kelly, "Trajectory generation for car-like robots using cubic curvature polynomials," *Field and Service Robots*, vol. 11, 2001.