Runtime Verification of Autonomous Driving Systems in CARLA

Eleni Zapridou[1][0000−0002−5025−6835], Ezio Bartocci[2][0000−0002−8004−6601], and Panagiotis Katsaros[1][0000−0002−4309−5295]

[1] Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
{zapridou,katsaros}@csd.auth.gr
[2] Vienna University of Technology (TU Wien), Wien, Austria
ezio.bartocci@tuwien.ac.at

Abstract. Urban driving simulators, such as CARLA, provide 3-D environments and useful tools to easily simulate sensorimotor control systems in scenarios with complex multi-agent dynamics. This enables the design exploration at the early system development stages, reducing high infrastructure costs and high risks. However, due to the high-dimensional input and state spaces of closed-loop autonomous driving systems, their testing and verification is very challenging and it has not yet taken advantage of the recent developments in theory and tools for runtime verification. We show here how to integrate the recently introduced `rtamt` library, for runtime verification of STL (Signal Temporal Logic) specifications, with the CARLA simulator. Finally, we also present the obtained results from monitoring quantitatively interesting requirements for an experimental Adaptive Cruise Control system tested in CARLA.

Keywords: autonomous driving · simulation · Signal Temporal Logic · runtime verification.

1 Introduction

Controllers design for autonomous driving systems is based, to a large extent, on high-fidelity simulators, such as CARLA [7], for their validation in urban driving scenarios with traffic intersections, pedestrians, street signs, street lights etc. CARLA is a versatile simulator that supports multiple approaches of autonomous driving, including a system decomposition into perception, planning and control, as well as the training of autonomous systems with machine learning (ML) components. CARLA is continuously developed towards a richer set of environment models, driving scenarios and ML use cases, but little is done for providing adequate means of model evaluation. CARLA and similar simulators, in their basic configuration, can export simulation traces to be further post-processed. However, this is not enough to effectively validate simulated systems.

Recent advances in runtime verification render it a promising perspective, for a multitude of reasons. First, closed-loop reachability analysis of control systems is characterized by theoretical limitations [1], which render them inapplicable

to real-scale industrial problems [9]. Second, runtime verification of simulation traces allows validating systems in realistic urban driving scenarios with complex dynamics between the traffic agents and the environment. Third, the runtime verification of STL (Signal Temporal Logic) properties [11] enables systematic design space exploration (e.g. through property falsification and parameter synthesis) [4]. These analysis capabilities also open prospects for testing and verifying the robustness of autonomous driving systems with ML components.

We present here the first step towards this perspective, i.e. the runtime verification of STL properties for autonomous driving systems in CARLA. Specifically, we have extended CARLA by integrating the `rtamt` library [12], for online runtime verification of STL properties. `rtamt` supports the qualitative and quantitative (property robustness) verification of STL specifications. This new tool combination allowed (i) the design space exploration of an Adaptive Cruise Control (ACC) system, with respect to various PID (proportional-integral-derivative) control parameters and (ii) the quantitative verification of performance requirements that are necessary for an ACC system [14]. Our experimentation with the CARLA model took place on a set of diversified scenarios, which guarantee that the ACC system is effectively validated under realistic urban driving conditions.

Section 2 presents the CARLA simulator, its configuration and integration with the `rtamt` library. Section 3 discusses the experimental results from our ACC system. In Section 4, we review the related work and in the last section we summarize the achievements and the future research prospects.

2 Online runtime verification of STL properties in CARLA

CARLA is a driving simulation environment [7] built on top of the Unreal Engine 4 game engine. It features a variety of digital assets for urban driving scenarios, including a sensor suite, various actors placed/moving on the map and the capability to control them, as well as the simulated environmental conditions.

A simulation is composed of: (i) the CARLA Simulator that computes the physics and renders the scene and all actor properties, (ii) client scripts written using a Python API, to spawn actors, attach sensors to them and, then, retrieve the sensor data, process them and compute the parameters needed by the controller (throttle, brake and steering). Computed values are then sent back to CARLA Simulator, thus forming an ever-running client-server interaction loop.

A client script consists of two parts:

- In the first part, connection with the CARLA Simulator is established, sensors are attached to the controlled vehicles and the actors of the simulated scenario are spawned or destroyed.
- The second part (Figure 1a) contains the control algorithm and the client-server interaction loop. The client retrieves the simulated world (`get_world()`) and the sensor signals, and then responds (via the `carla.VehicleControl` object) with the vehicle control signals that are subsequently applied to the `Vehicle` object (method `apply_control()`).

The simulated world, maintained by the server, has its own clock and time. The time-step can be fixed or variable depending on the user preferences. CARLA runs, by default, in asynchronous mode: the server runs the simulation, without waiting for the client, which receives sensor data via a callback function that is called whenever there is new sensor data. In synchronous mode, the client imposes total control over the simulation and its information: the server waits for a client tick (`world.tick()`), before updating to the next simulation step.

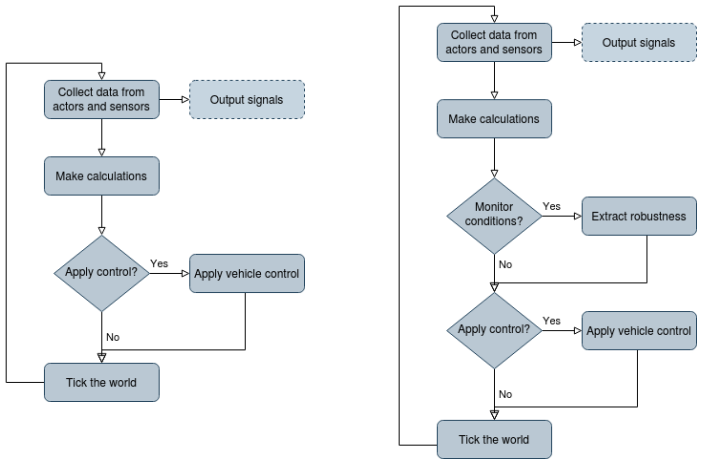


Fig. 1: CARLA (a) main simulation loop, and (b) online monitoring

The time-step duration affects the computation of physics. As time progresses and more variables are involved, the simulation may become imprecise. Therefore, physics must be computed within time-steps not greater than 0.1 sec. An important consideration is the reproducibility of simulation scenarios. This is achieved using functions of the Python API that enable and stop a simulation recorder. Then, the obtained file can be playback.

To inspect the actor behavior (e.g. location, velocity, acceleration), there are client functions to access values at the latest tick. If the simulation runs in synchronous mode (Figure 1a), data is stored in signals and it is reliable. For online monitoring (Figure 1b), we use the synchronous mode with fixed time-step, ensuring that the simulation time and physics will be in synchrony and the sensed signals are reliable. There are, however, challenges regarding the properties that can be monitored, since online monitoring is restricted to a single pass through the simulation trace. To assess the performance of closed-loop autonomous systems, we need to be able to determine the satisfaction/violation of a property specification based on a robustness degree function, i.e. a means to indicate how far is the monitored trace from satisfying or violating a specification. This allows to interpret the performance of the system design under various parameters, scenario events (e.g. street signs) and environment perturbations, as opposed to the binary pass/fail answer of qualitative verification, which is not very informative.

We rely on the `rtamt` library [12] that generates online robustness monitors from bounded-future STL specifications [11,4] that can express temporal properties for continuous signals. Monitoring takes place through evaluating the (equivalent) past STL specification, according to its quantitative semantics, interpreted in discrete or dense time [6]. For a signal given as a sequence of (time, value) pairs, `rtamt` computes at different instants how far is the signal from satisfying or violating the property. When using discrete-time monitors, sensing of inputs and output generation are done at a periodic rate, whereas dense-time monitors compute the min and max of a numeric sequence over a sliding window.

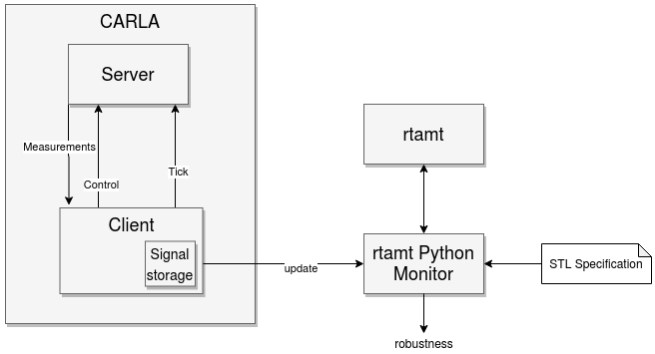


Fig. 2: STL runtime verification of CARLA simulations with the `rtamt` library

Figure 2 shows how an `rtamt` monitor is integrated with a CARLA simulation in synchronous mode and fixed time-step. The (time, value(s)) pairs with data, for every car with the autonomous system enabled, are stored by the client at each step. This requires monitoring events, like that the host (ego) vehicle is behind another car. If these conditions hold true, the monitor's `update` function computes the robustness measures from the signals. This is a non-intrusive solution, since all computations take place, while the simulated time is frozen.

3 Experimental results for an ACC system

3.1 An experimental ACC system

ACC systems extend the functionality of conventional cruise control systems by the capability to adjust the host vehicle's velocity and assuring a safety distance to the preceding vehicle through controlling its throttle and/or brake. A key part of ACC is the range sensor, which is used to measure the distance from the preceding vehicle. We use the CARLA obstacle detector that detects obstacles, including vehicles, located within a specified distance from the host vehicle, towards its traveling direction. The ACC system of the host vehicle is enabled, when the preceding vehicle is too close (distance below a fixed threshold r) or if it is moving slowly. In this case, the ACC controls the throttle and the brake, in order to keep the distance higher or equal to the safety distance, which is

dynamically calculated, as in Def. 1. In the absence of a preceding vehicle, the velocity of the host vehicle is controlled by the CARLA server.

Definition 1. *[13] A longitudinal distance between a car c_h that drives behind another car c_p, where both cars are driving at the same direction, is safe w.r.t. a response time ρ, if for any braking of at most $a_{brake_{max}}$, performed by c_p, if c_h will accelerate by at most $a_{accel_{max}}$ during the response time, and from there on will brake by at least $a_{brake_{min}}$ until a full stop, then it won't collide with c_p.*

Let v_h, v_p be the longitudinal velocities of the cars. Then, the safe longitudinal distance between c_h and c_p is ($[x]_+ := max\{x, 0\}$):

$$SD = \left[v_h \cdot \rho + \frac{1}{2} a_{accel_{max}} \cdot \rho^2 + \frac{(v_h + \rho \cdot a_{accel_{max}})^2}{2 \cdot a_{brake_{min}}} - \frac{v_p^2}{2 \cdot a_{brake_{max}}} \right]_+$$

In our case, ρ is equal to the time elapsed between two simulation steps, since we the simulation runs in synchronous mode this is fixed and set to 0.05 seconds.

For $a_{accel_{max}}, a_{brake_{min}}$ and $a_{brake_{max}}$ we parameterized the simulation model using values from vehicle specifications by car manufacturers. The values used are: $a_{accel_{max}} = 5.4\, m/sec^2, a_{brake_{min}} = 2.9\, m/sec^2, a_{brake_{max}} = 9.8\, m/sec^2$.

The ultimate aim of ACC is to allow the host vehicle traveling with a velocity v_h at most equal to v_p, lower than the road speed limit (imposed by CARLA), while minimizing the distance d from the preceding vehicle, without violating SD. The available time, before the two vehicles get closer than SD is:

$$time_{safe} = \frac{d - SD}{v_h - v_p}$$

Thus, the target velocity v_{tar} and target acceleration a_{tar} for the host vehicle to have in next simulation sample (time-step) are:

$$a_{tar} = \frac{min\{v_p, speed_{limit}\} - v_h}{time_{safe}}$$

$$v_{tar} = v_h + \rho \cdot a_{tar}$$

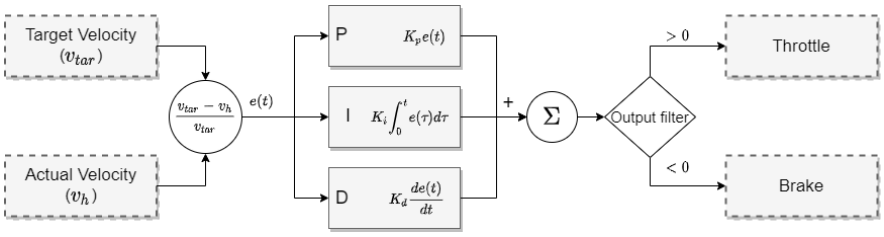


Fig. 3: PID controller for the ACC system

The ACC function is driven by the PID controller of Figure 3, which computes the needed throttle and brake based on v_{tar} and the actual velocity (v_h). The PID

controller continuously calculates an error value $e(t)$, as the difference between the desired set point (in our case v_{tar}) and a measured process variable (v_h), and applies a correction based on three terms, namely the proportional, the integral, and the derivative (denoted respectively by P, I and D). In practice, the PID controller applies a responsive correction to the controlled function.

3.2 Design space exploration w.r.t. PID parameters

When designing a PID controller, it is important to understand how to improve the system's performance. As a rule of thumb, by increasing the proportional gain (K_P) we achieve a proportional increase of the control signal for the same level of $e(t)$. In this way, the system reacts more quickly, but it tends to exceed its target more (overshoot). The parameter of derivative control (K_D) affects the capability of damping, which is important to decrease overshoot. The value of integral parameter (K_I) affects the capability to limit the steady error, but the system may become more sluggish and oscillatory, if it is not properly tuned.

Our runtime verification approach can help to adjust the gains K_P, K_I, and K_D with the aim to eventually achieve a satisfactory overall response. We present here the experimental results obtained for two different sets of parameter values:

$$K_{P_A} = 1,\ K_{I_A} = 1,\ K_{D_A} = 0.0005$$
$$K_{P_B} = 2,\ K_{I_B} = 0.01,\ K_{D_B} = 0.4$$

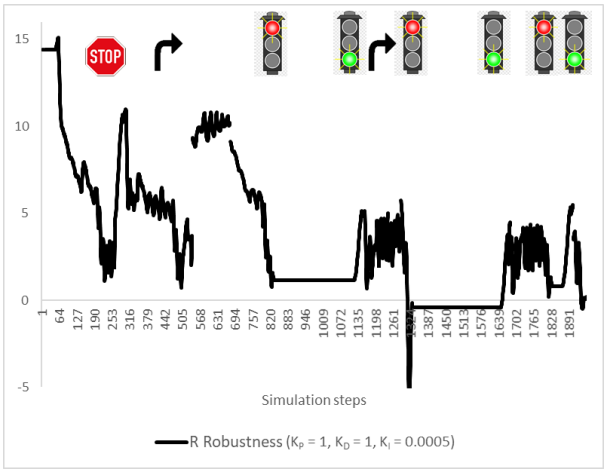
Since the primary aim of the ACC system is to keep the safe distance from the preceding vehicle, we evaluate the requirement,

$$R := d - SD > 0$$

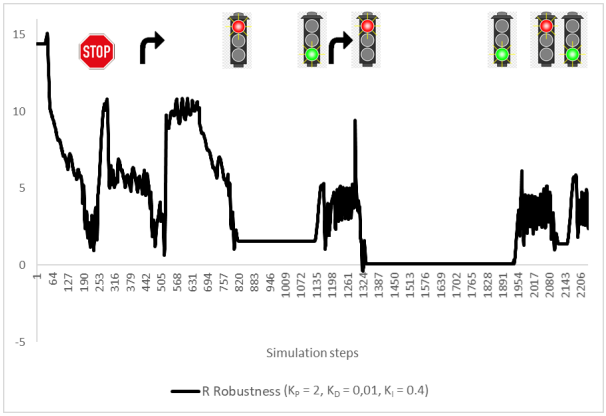
in order to find the parameter set that seems to be more appropriate. For the runtime verification of R and for all other experiments that are reported henceforward, we employed the `rtamt` library to generate discrete-time monitors that were integrated with the CARLA simulation of our ACC system.

Figure 4 shows the robustness of R, when monitoring similar driving scenarios with the two mentioned sets of parameters (A and B). These scenarios refer to the route of two cars moving one behind the other in the CARLA urban environment with junctions, stop signs and traffic lights. While following the same route, the scenarios differ slightly with respect to the duration of the red traffic lights.

In Figure 4a, no robustness value is shown in specific simulation steps, when the preceding vehicle was too far and the ACC of the host vehicle was disabled. For parameters $K_{P_A}, K_{I_A}, K_{D_A}$, R is strongly violated, when the cars stop at the second traffic light, and it is violated, for more than 300 steps, while waiting for the green traffic light. For $K_{P_B}, K_{I_B}, K_{D_B}$ (Figure 4b), R is slightly violated at the same instant, but the system adjusts quickly and the host vehicle stops at the traffic light maintaining the safety distance. By simulating more similar scenarios, we found that R is violated for $K_{P_A}, K_{I_A}, K_{D_A}$ at the second traffic light and every next time the vehicles stop for a long period. The problem vanishes for $K_{P_B}, K_{I_B}, K_{D_B}$ that was chosen for the experiments reported hereafter.



(a)



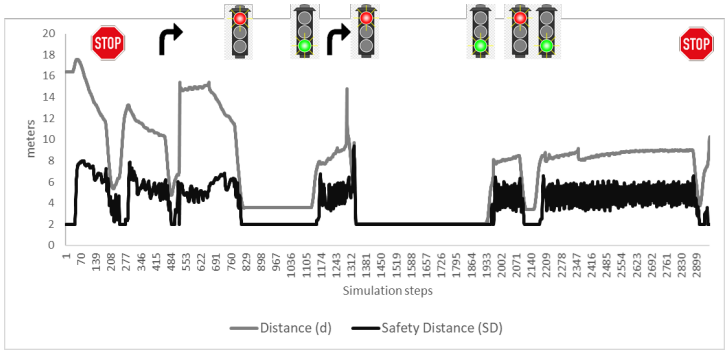
(b)

Fig. 4: Robustness of R for (a) $K_{P_A}, K_{I_A}, K_{D_A}$ and (b) $K_{P_B}, K_{I_B}, K_{D_B}$

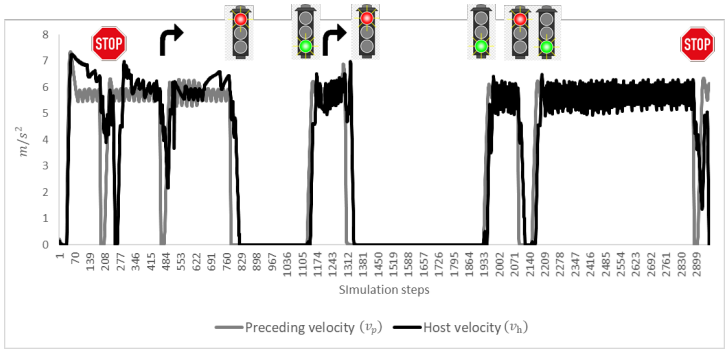
A more detailed view of the same scenario, extended by additional steps, is shown in Figure 5. These graphs show the distance between the two vehicles and how it compares with the safety distance (Figure 5a), and their velocities (Figure 5b). We observe that in a stop signal, the host car stops when the preceding has already left and similarly in a traffic light, the vehicles stop and start moving with a time difference. In Figure 5a, we see that d gets closer to SD, as the PID controller adjusts by using more historic data.

3.3 Requirements for an ACC system

According to [14], any ACC system has to fulfill the following performance requirements, in order to be used in public roads:



(a)



(b)

Fig. 5: Distance (a) between host and preceding vehicle and (b) their velocities

R_1. Relative difference $v_p - v_h$ should not be too high, except when the preceding vehicle is out of range or the host vehicle has decelerated, due to a stop signal or a red traffic light, while the preceding vehicle is moving ahead.

R_2. Acceleration of the host vehicle (a_h) must be greater than or equal to G, except if the preceding vehicle's acceleration (a_p) is less than this limit,

$$G := -0.25 \cdot \frac{T_{hw}+1}{m}$$

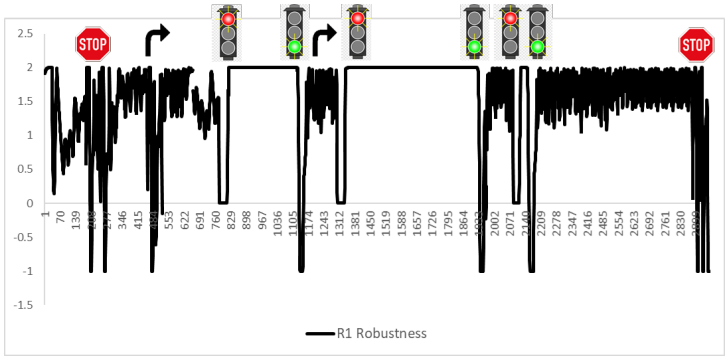
with m the vehicle's mass and T_{hw} a constant time headway, i.e. the distance between the two vehicles in time expressing the degree to which the safety distance varies proportionally to the vehicles velocity (spacing policy).

The STL property for the first mentioned requirement is:

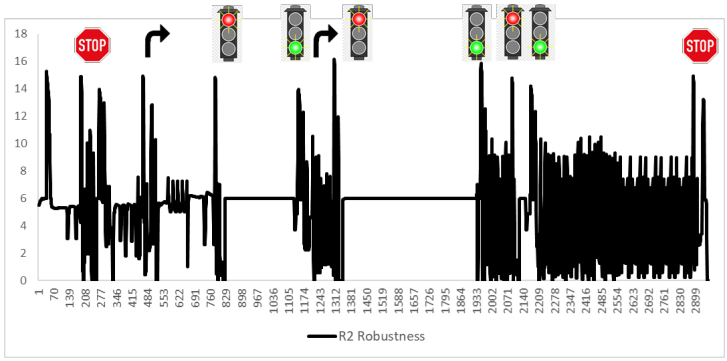
$$R_1 := v_p - v_h < c \vee d > r \vee \mathit{stopped}_h$$

where c is a threshold for the relative difference between velocities, r is the distance threshold under which the ACC system is enabled and $\mathit{stopped}_h$ is `true`, when the host vehicle decelerates, due to a stop signal or a red traffic light. We set the threshold for the difference between the velocities $c = 1.5\, m/s$.

The STL property for the R_2 requirement is ($\boxminus$ is the *historically* operator):



(a)



(b)

Fig. 6: Robustness of requirement (a) R_1 and (b) R_2

$$R_2 := a_h \geq G \vee \boxminus_{[0:t]} a_p < G$$

where $[0 : t]$ is the time interval for the host vehicle to adjust its acceleration back to normal, after an extreme deceleration of the preceding vehicle.

Figure 6a shows the robustness of R_1 for the scenario of Section 3.2. We observe violations after stop signals and green traffic lights. In stop signals, when the host vehicle stops, the preceding has already moved ahead, causing an excessive difference in velocities. When a traffic light turns to green, the host vehicle needs more steps to start accelerating and the velocities difference is also increased. These performance perturbations, due to usual driving incidents, do not invalidate our system. R_1 is satisfied and robustness raises as time proceeds, showing that the ACC adjusts to acceleration changes of the preceding vehicle.

Figure 6b shows the robustness of R_2 for the same scenario. In this experiment, we have set $T_{hw} = 1.5\,sec$, $t = 3$ and $m = 1200\,kg$ (mass of "Seat Leon" that is the host vehicle in our experiments). The ACC fulfills the requirement, although the robustness in some steps is low (host vehicle's acceleration is close to G). By increasing/decreasing T_{hw} we could find the lowest time headway that still fulfills R_2 or a higher time headway, for which the robustness is increased.

4 Related work

In contrast with other systems, automotive control systems exhibit complex behaviors that are difficult to anticipate at design time. Their performance requirements typically arise out of test driving scenarios.

The VerifAI toolkit [8] analyzes simulation traces (also from CARLA) of systems with ML components. It has a wider scope from our work, aiming to address the absence of specifications for perception components, the analysis of ML components and environment modeling (e.g. distribution assumptions made by ML components). It works through offline monitoring system-level properties in Metric Temporal Logic (MTL) [2]. Monitors can output falsifying traces and a feedback to direct sampling to find falsifying scenarios. Such an analysis may be also possible in our case, since `rtamt` can be easily used for offline monitoring.

Through simulation-based analysis, we can also identify behaviours that can be then captured as requirements. In [10], a set of automotive behavior classes is identified that control engineers typically want to avoid (ringing, excessive overshoot, slow response time, steady state error etc.) and a library of signal templates for STL is proposed, such that it will be easier to specify STL requirements that exclude them. These requirements are easier to be checked over simulation traces produced by a Simulink model of the system under design, whereas for our ACC requirements (adopted from [14]) in Section 3.3, we advocate their validation over realistic driving scenarios generated by CARLA.

Worth to mention are the S-TaLiRo [3] and Breach [5] tools, for sensitivity analysis and falsification testing over Simulink traces. Sensitivity analysis of model robustness to STL requirements is based on uniformly varying model parameters, whereas falsification looks for an input signal that violates a requirement. Another interesting prospect is the requirements-driven testing [15].

5 Conclusion and future research prospects

We presented an approach for integrating CARLA simulations with runtime monitors generated by the `rtamt` library[1]. Our proposal enables the validation of autonomous driving control by online monitoring STL specifications over realistic driving scenarios. We believe that this is a means for design space exploration and has the potential to uncover undesired behaviours or check important performance requirements. We showed the results from applying our approach, in order to find appropriate parameters for the PID control of an experimental ACC system and for checking it against important performance requirements.

Our work is a first step towards additional contributions that will allow testing and verification of autonomous driving systems with ML components. CARLA already supports the simulation of such systems, but we need to further develop our approach towards automating property falsification, parameter synthesis, sensitivity analysis and systematic scenario testing.

[1] The CARLA client scripts for our ACC system with the integrated `rtamt` monitors can be accessed online at http://depend.csd.auth.gr/software/carla

References

1. What's decidable about hybrid automata? Journal of Computer and System Sciences **57**(1), 94 – 124 (1998)
2. Alur, R., Henzinger, T.A.: Logics and models of real time: A survey. In: Real-Time: Theory in Practice. pp. 74–106. Springer (1992)
3. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 254–257. Springer (2011)
4. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications, pp. 135–175. Springer (2018)
5. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Computer Aided Verification. pp. 167–170. Springer (2010)
6. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Formal Modeling and Analysis of Timed Systems. pp. 92–106. Springer (2010)
7. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. pp. 1–16 (2017)
8. Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Ravanbakhsh, H., Vazquez-Chanlatte, M., Seshia, S.A.: Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In: Computer Aided Verification. pp. 432–442. Springer (2019)
9. Jin, X., Deshmukh, J.V., Kapinski, J., Ueda, K., Butts, K.: Powertrain control verification benchmark. In: Proc. of 17th Int. Conf. on Hybrid Systems: Computation and Control. pp. 253–262. HSCC 14, Association for Computing Machinery, New York, NY, USA (2014)
10. Kapinski, J., Jin, X., Deshmukh, J., Donze, A., Yamaguchi, T., Ito, H., Kaga, T., Kobuna, S., Seshia, S.: St-lib: A library for specifying and classifying model behaviors. In: SAE Technical Paper. SAE International (2016)
11. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166. Springer (2004)
12. Nickovic, D., Yamaguchi, T.: RTAMT: online robustness monitors from STL. CoRR **abs/2005.11827** (2020), `https://arxiv.org/abs/2005.11827`
13. Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a formal model of safe and scalable self-driving cars. CoRR **abs/1708.06374** (2017), `http://arxiv.org/abs/1708.06374`
14. Takahama, T., Akasaka, D.: Model predictive control approach to design practical adaptive cruise control for traffic jam. International journal of automotive engineering **9**, 99–104 (2018)
15. Tuncali, C.E., Fainekos, G., Prokhorov, D., Ito, H., Kapinski, J.: Requirements-driven test generation for autonomous vehicles with machine learning components. IEEE Transactions on Intelligent Vehicles **5**(2), 265–280 (2020)