2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Safety Analysis of Automated Driving Platforms Using Digital Twin Simulation and Runtime Monitoring

Tasneem A. Awaad, Hanya A. Elged, Mohamed A. Abu-Bakr, Sama Y. Fathy, Sara H. Ahmed, M. Watheq El-Kharashi, Ain-Shams University, Cairo, Egypt

Mohamed AbdElSalam, Siemens EDA, Cairo, Egypt

***Abstract*—The future of the automobile industry lies in Autonomous Vehicles (AV). Despite the efforts of major companies to completely automate driving, ensuring AV's safety remains challenging. The objective of our work is to provide a compositional simulation interconnect framework to verify the safety of Autonomous Driving Platforms (ADP). This objective is achieved by translating high-level safety requirements from ISO 26262 and ISO 21448 into verifiable properties and building a comprehensive digital twin comprising of a Runtime Verification (RV) monitor for property checking together with a car scenario simulator and ADP-under-test. We demonstrate our framework using Apollo an open-source autonomous driving platform developed by Baidu as a case-study.**

Keywords—Autonomous Driving Platforms, Digital Twin Simulation, formal analysis, runtime verification, Safety

I. INTRODUCTION

The quick evolution of autonomous vehicles, driven by the utilization of ADPs, particularly Advanced Driver Assistance Systems (ADAS) software stacks, characterizes both current and future vehicles. Industry players such as Waymo and Baidu have demonstrated the capability of fully automated driving, especially in specific geographic locations and under restricted weather and illumination conditions. Simultaneously, major software companies like Uber are actively engaged in developing their automated driving frameworks.

The urgency for safety requirements on both traditional software and Machine Learning (ML) components within ADAS software stacks is increasingly evident to prevent catastrophic events. In the automotive domain, safety requirement elicitation, also known as functional safety requirement analysis, is well-defined by two domain-specific standards: ISO 26262, addressing safety requirements related to component malfunctions, and ISO 21448, outlining limitations in achieving the intended functionality of ML-based components.

Our work aims to fulfil a crucial objective: safety analysis using comprehensive digital twin simulation. The digital twin is built using a compositional simulation interconnect fabric to connect an ADP to a car scenario Simulator and a Runtime Verification monitor. We also established a safety analysis method that translates high-level ISO requirements into properties for verification during simulation, ensuring a proactive approach to safety assessment. In essence, our work aligns with the industry's pressing need for robust safety measures in the domain of automated driving.

The paper is organized as follows. Section II gives some background on ISO Standards for safety and a brief introduction to ADPs and tools we used to build our compositional simulation interconnect framework for safety analysis. Section III discusses related work in the field of digital twin simulation and safety analysis of AVs. Section IV presents the workflow of the proposed safety framework. Section V presents the results of our case-study with conclusions and future work discussed in Section VI.

II. BACKGROUND

A. *ISO 26262 & ISO 21448*

ISO 26262 [1] is an international standard for the functional safety of electrical and electronic systems in road vehicles. It is derived from the broader IEC 61508 standard and focuses specifically on automotive applications. It utilizes Automotive Safety Integrity Levels (ASILs) to classify risks and determine the necessary safety measures and its objective is to ensure that safety-related systems perform their intended functions correctly and at the right time, minimizing the risk of accidents due to system failure.

On the other hand, ISO 21448 [2] provides a framework for identifying and mitigating risks associated with the intended functionality of ML-based systems. This includes addressing potential hazards that could arise from the limitations of ML algorithms. The standard also ensures that ML systems are designed and tested to meet safety requirements, reducing the likelihood of accidents due to functional insufficiencies. By integrating these principles, ISO 21448 helps ensure that ML-based automotive systems are safe, reliable, and capable of handling real-world complexities.

B. Autonomous Driving Platforms

Autonomous driving refers to vehicles or systems that operate with minimal or no human intervention. ADPs provide the infrastructure and technology enabling these vehicles to function. These ADPs integrate key components such as localization, perception, prediction, planning, and control. Apollo [3] developed by Baidu, is a leading open-source autonomous driving platform that we use in our case-study. The same can be applied to other ADPs, e.g. Autoware built on the Robot Operating System (ROS), Nvidia Drive, Intel Mobileye, Tesla Autopilot and Self Driving and Alphabet Waymo.

C. Sensor/Scenario Simulators

Sensor/Scenario Simulators provide a physics-based simulation platform to prototype, test, and validate advanced driver assistance systems. Unlike real-world conditions, simulated conditions can be fully quantified and controlled. In our case-study we used CARLA [4] open-source simulator, other simulators (open-source or proprietary) can be used as well in our framework for safety. Usually, these simulators have Python or C++ interface to connect with other tools for co-simulation.

D. Runtime Monitoring

DejaVu [5] is a monitoring tool designed in Scala, for the synthesis of runtime monitors for past-time first-order LTL specifications. Starting with a user-defined formula/property, the tool parses the formula and generates an abstract syntax tree, which is then traversed and translated into a monitor program. An event trace is fed into the monitor that returns a verdict. It supports property checking against event streams using inputs event stream files (CSV) and property files written in temporal logic (QTL) as shown in Fig. 1.

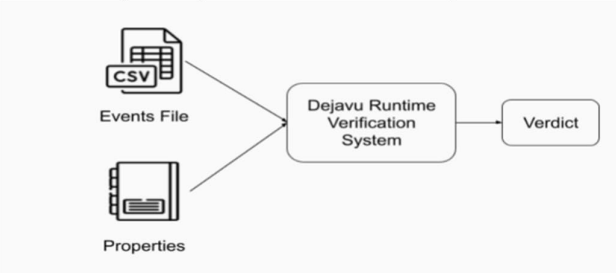

Fig. 1. DejaVu runtime monitor.

Events Trace: The events trace file lists events in CSV format, each representing an occurrence as follows.

```
detect, o1, d1
```

Property Specification: Properties are written in first-order past-time linear temporal logic in QTL files (e.g.,

```
prop checkObstacleDistance :
Forall object . Forall d .
detect(object , d) -> d <= R        # R is safe distance
```

TP-DejaVu enhanced DejaVu by managing two-phase Runtime Verification (RV). The first phase, implemented in Scala, performs operational RV with arithmetic, string, and Boolean manipulations. The second phase uses DejaVu for monitoring against first-order specifications.

Operational Specification: The operational specification file initializes variables and performs manipulations.

```
initiate
no_of_objects: int := 0
on detect(object: int, distance: int, timestamp: int)
```

```
no_of_objects: int := no_of_objects + 1
output objects_detected(no_of_objects, timestamp)
```

Declarative Specification: The declarative specification checks conditions against events output by the operational file as follows.

```
prop objectsCheck :
forall no_of_objects . forall timestamp.
objects_detected(no_of_objects, timestamp) -> timestamp < 10 or
no_of_objects >= 5
```

E. Compositional Simulation Interconnect framework

To connect the ADP with the sensor/scenario simulator and Runtime monitor, a compositional simulation interconnect framework is needed [6]. The framework enables interoperability between the various components of a digital twin in general, synchronizes the network communication and enables data transfer via various protocols (e.g. Ethernet). A client connection is feasible if it complies with the electronic design automation standards supported: SystemC TLM 2.0, JModelica FMI 2.0, and Inter-Process communication. Such clients include sensor/scenario simulators, mechatronic system simulators exported as co-simulation FMUs, dashboard SW, cloud services and various computational and AI models at different abstraction levels, including C/C++, SystemC TLM, Python, Robot Operating System, virtual platforms, HW emulation, FPGA prototyping platforms and embedded system boards. The presumption is that each of these external simulators and foreign models runs in their own processes and supports a third-party API. That application-specific API, as shown in Fig. 2, can be coupled with a TLM fabric portal interface – we call this a Gateway – which gives access to the interconnect fabric backplane for transactional communication purposes as well as mutual time advancement coordination. Each third-party simulator or foreign model is assumed to be a client process that hangs on the common backplane interconnect. The backplane is the keeper of time and is responsible for all time advance operations. To ease the automation of the interconnect fabric. The framework automation flow [7] starts with a description of the clients' connection using a Digital Twin Description language and a builder that parses the description and generates the backplane server and gateways needed to connect different digital twin components' remote clients.

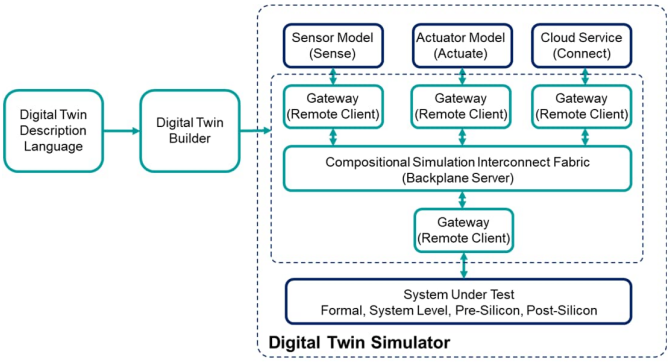

Fig. 2. Compositional simulation interconnect framework.

III. RELATED WORK

There have been contributions to Safety Runtime Verification in the process of running Temporal Logic properties at runtime and other contributions in the field of AV safety in general. In [8], Donal Heffernan et al. applied ISO 26262 to define functional safety requirements for an E/E gearbox controller. The functional safety statements were mapped to Past-time Linear Temporal Logic (ptLTL) and verified using a runtime monitor. Similarly, [9] also focused on applying safety runtime monitoring on a prototype of a vehicle built with Raspberry Pi and Arduino Uno. There is also more work done on vehicles simulated on various platforms. For example, [10] showed how to integrate the RTAMT [11] library, for runtime verification of Signal Temporal Logic (STL) specifications, with the CARLA simulator. Also, in [12], Kosuke Watanabe et al. used STL and the Breach monitoring tool to monitor and detect the undesirable interactions between the Cooperative Pile-up Mitigation

System (CPMS) and False-Start Prevention System (FPS) ADAS features on an AV modeled on Unity. Moreover, some research contributed to making the safety verification process easier without doing the actual testing. For example, in [13], Lina Marsso et al. focused on creating two formal models (written in the LOTOS New Technology (LNT) language) for AVs that are more comprehensive than existing models and can be used to generate relevant critical scenarios for testing AVs. In [14], Paul Rau et al. developed a structured framework for deriving scenarios necessary for the Safety of the Intended Functionality (SOTIF) analysis and applied this framework to a highly automated chauffeur system. Authors in [15] also presented a safety verification framework for AVs. The framework is based on new longitudinal and lateral safe distances, lane changes, overtaking and how to face new traffic participants. Finally, in [16], Chejian Xu et al. provided the first unified platform SafeBench to integrate different types of safety-critical testing scenarios, scenario generation algorithms, and other variations such as driving routes and environments.

Compared to related work, our methodology aims to build a comprehensive digital twin of ADPs with sensor/scenario simulators and runtime monitoring tools with safety properties extracted from ISO standards 26262 and 21448. Furthermore, we rely on runtime monitors as an assertion-based verification tool for ADPs.

IV. Work Flow

In this section, we detail the user workflow for verifying the safety of ADPs and the associated safety properties used as a proof of concept. As shown in Fig. 3, the workflow typically consists of four stages: setting up the chosen ADP with the scenario simulator, translation of ISO high-level safety requirements into formal logic properties, building a comprehensive digital twin framework, and finally checking any property violations using TP-DejaVu.

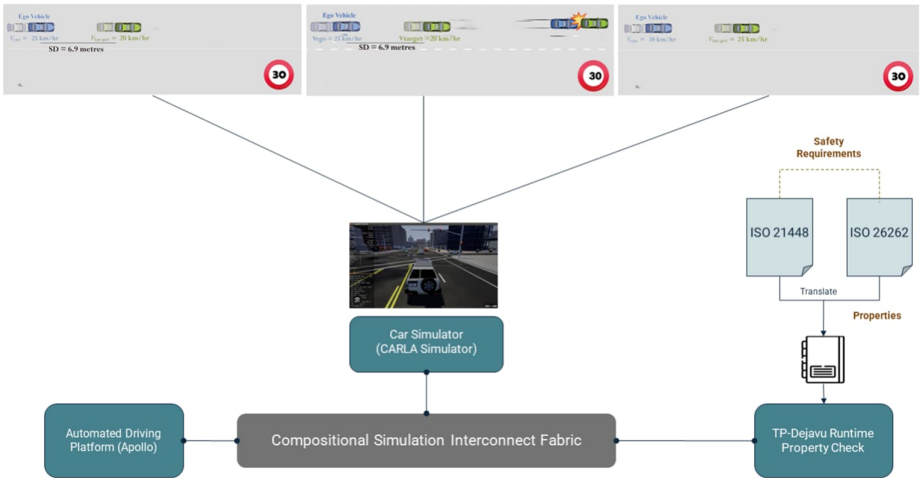

Fig. 3. Safety analysis workflow of ADPs.

ADP failures could result from human errors while coding different components of the ADP software stack or because of intrusions/attacks on any of these components. Our success criteria are to be able to catch these failures using the runtime monitor based on a set of well-defined safety requirements during a comprehensive digital twin simulation of the ADP. Since Apollo is a mature autonomous driving software, error injection is employed during simulation to test for violations of the safety properties we are checking.

A. *Safety Properties*

In this section, we will discuss three crucial safety properties for autonomous vehicles: Safe Distance, Road Speed Limit, and Collision Rate. The Safe Distance property will be explained in detail, highlighting the structure of the operational and declarative files used for verification. For the Road Speed Limit and Collision Rate properties, a brief overview will be provided, focusing on their importance and the key aspects of their implementation without diving into the specifics of the operational and declarative files.

1) Safe Distance:

Reference ISO standard: ISO 26262. **Component:** Adaptive Cruise Control (ACC). **Background:** ACC systems adjust the host vehicle's velocity to maintain a safe distance from the preceding vehicle by controlling the throttle and brake. A crucial part of ACC is the range sensor, which measures the distance to the preceding vehicle. We used the CARLA obstacle detector to identify obstacles, including vehicles, within a specified range. The ACC system activates when the preceding vehicle is too close (below a threshold r), adjusting the throttle and brake to maintain a safe distance.

Safe Distance Calculation: The safe longitudinal distance (d_{safe}) between a host vehicle (*ch*) and a preceding vehicle (*cp*) is determined using the following equation:

$$d_{\text{safe}} = v_h \cdot \rho + \frac{1}{2} a_{\text{accel,max}} \cdot \rho^2 + \frac{(v_h + \rho \cdot a_{\text{accel,max}})^2}{2a_{\text{brake,min}}} - \frac{v_p^2}{2a_{\text{brake,max}}},$$

where:

- v_h and v_p are the velocities of the host and preceding vehicles, respectively.
- ρ is the response time (0.01 seconds in our simulation).
- $a_{accel,max}$ = 5.4 m/s², $a_{brake,min}$ = 2.9 m/s², and $a_{brake,max}$ = 9.8 m/s².

Event Trace (CSV Format): The log records events such as:

```
detectLeadingVehicle(egoSpeed, leadingVehicleSpeed, distance)
```

Operational Specification:

```
initiate
   P: double := 0.01
   AccMax: double := 5.4
   AccBrakeMax: double := 9.8
   AccBrakeMin: double := 2.9

on detectLeadingVehicle(vEgo: double, vLeading: double, distance: double)
   t0: double := vEgo * P
   t1: double := 0.5 * AccMax * P*P
   t2: double := vEgo + P * AccBrakeMax
   t3: double := 2.0 * AccBrakeMin
   t4: double := vLeading * vLeading
   t5: double := 2.0 * AccBrakeMax
   SD: double := t0 + t1 + (t2 * t2)/t3 -  t4/t5
   distanceCheck: bool := distance >= SD
   output checkSafeDistance(distanceCheck)
```

Declarative Specification: This property verifies that the output from the pre-evaluation step (distanceCheck) is always true.

```
prop safeDistanceCheck :
forall distanceCheck .
checkSafeDistance(distanceCheck) -> distanceCheck = "true"
```

2) Road Speed Limit:

Reference ISO standard: ISO 26262, ISO 21448**. Component:** Perception, Planning, and control**. Background:** Speed management ensures safe mobility by setting speed limits, reducing speeding, and mitigating speeding-related crashes. This property verifies that the host vehicle respects the speed limits imposed by CARLA.

3) Collision Rate: **Reference ISO standard:** ISO 21448**. Component:** Perception, planning, and control. **Background:** Collision rate analysis is crucial for ensuring the safety and reliability of autonomous vehicles. The property tracks the number of collisions and verifies whether the collision rate is below a specified threshold.

B. Scenario Generation

After implementing the safety properties in TP-DejaVu, we generate scenarios to verify that TP-DejaVu runtime monitoring is functional as expected. The scenarios test Apollo in normal mode and abnormal mode through error injection by code modifications to mimic the effect of a programmer bug or intrusion/attack.

Table I. Scenario generation.

Scenario	**Safety Requirements Monitored (Safe Distance, Speed Limit, Collision Rate)**		
	Description	***Error Injection***	***Expected Properties To be Violated***
I	The host vehicle follows a leading vehicle on a road with a speed limit of 30 kilometres per hour.	No	None
II	Same scenario as normal mode, but the host vehicle exceeds the road speed limit.	Apollo/modules/planning/reference line/ reference line.cc - Line 827: The modification is done in the method ”Reference-Line::GetSpeedLimitFromS” to return the speed limit multiplied by two.	Speed Limit
III	Same scenario as normal mode, but the host vehicle follows the leading vehicle very closely.	Apollo/modules/planning/tasks/deciders/ speed decider/speed decider.cc - Line 173: ”IsFollowTooClose” to always return false. Apollo/modules/planning/tasks/optimizers/ path time heuristic/ dp st cost.cc - Line 113:”GetObstacleCost” to return the obstacle cost as zero. Apollo/modules/planning/tasks/optimizers/ piecewise_jerk speed/piecewise jerk speed nonlinear_optimizer.cc - Line 286: ”SetUpStatesAndBounds” to increase the speed upper bound by removing the subtracted 8. Apollo/modules/planning/tasks/optimizers/ piecewise_jerk speed/piecewise jerk speed optimizer.cc - Line 114:” Process” to increase the speed upper bound by removing the subtracted 8.	Safe Distance
IV	Same scenario as normal mode, but the host vehicle follows the leading vehicle very closely and collides with it. The leading vehicle is also controlled to suddenly break after 120 metres.	Same error injections as Scenario III	Safe distance and collision rate
V	Same scenario as normal mode, but the host vehicle follows the leading vehicle very closely and collides with it, and exceeds the speed limit. The leading vehicle is also controlled to suddenly break after 120 metres.	Same error injections as Scenarios II and III	Safe distance, speed limit and collision rate

V. EXPERIMENTAL RESULTS & DISCUSSIONS

In this section, we demonstrate the experimental results of verifying the three safety properties across five different scenarios. These results were obtained using Carla simulator version 0.9.13 and Apollo version 8.0.0. The scenarios were run on a machine with an AMD Ryzen 7 6800H with Radeon Graphics CPU, 16 GB of ram, and an RTX 3050 mobile GPU. Table II illustrates the results for the first scenario. In this scenario, all three safety properties were monitored over a simulation period of 135 seconds, with no violations detected. The Apollo autonomous driving platform operated without any errors, showcasing its inherent safety and reliability.

Fig. 4 contains tables that summarize the results for scenarios II, III, IV, and V, above each table, a bar graph is included to visualize the results. In scenario II, errors were intentionally injected into the planning module of Apollo to induce speed limit violations. This resulted in 227 violations over a simulation period of 120 seconds. In scenario III, violations were observed only for the Safe Distance property. These violations were induced by injecting errors into the planning module of the Apollo stack. In scenario IV, violations were observed for both

Safe Distance and Collision Rate properties. These violations were induced by injecting errors into the safe distance property in the planning module of the Apollo stack and instructing the leading vehicle to suddenly brake. In scenario V, errors were intentionally injected into the planning module of Apollo in addition to instructing the leading vehicle to suddenly brake to induce violations for all three properties over a simulation period of 136 seconds. Note that the violations for the collision rate property do not indicate the number of collisions, in fact, there was only one collision in both scenarios IV and V. The reason for this is that the property is checked at each tick of the simulation, so all the events would be recorded as violations until the vehicle travels a sufficient distance without collisions.

Table III. Property checking results.

Property	**Scenario I**		
	#Events	***#Violations***	***Sim. Time (sec)***
Safe Distance	1,644	0	135
Speed Limit	7,255	0	135
Collision Rate	7,255	0	135

Fig. 4. Scenario II, III, IV, V results.

VI. CONCLUSIONS

In this paper, we successfully demonstrated the feasibility and effectiveness of integrating a digital twin of an autonomous driving platform within a car scenario simulator, alongside a runtime monitor for safety analysis. By translating high-level safety requirements from industry standards ISO 26262 and ISO 21448 into checkable properties, we established a pipeline for real-time safety verification during digital twin simulation.

Future research should aim to develop a more comprehensive list of properties that incorporate a wider range of environmental factors for safety compliance of existing or newly developed autonomous driving platforms across various real-world scenarios.

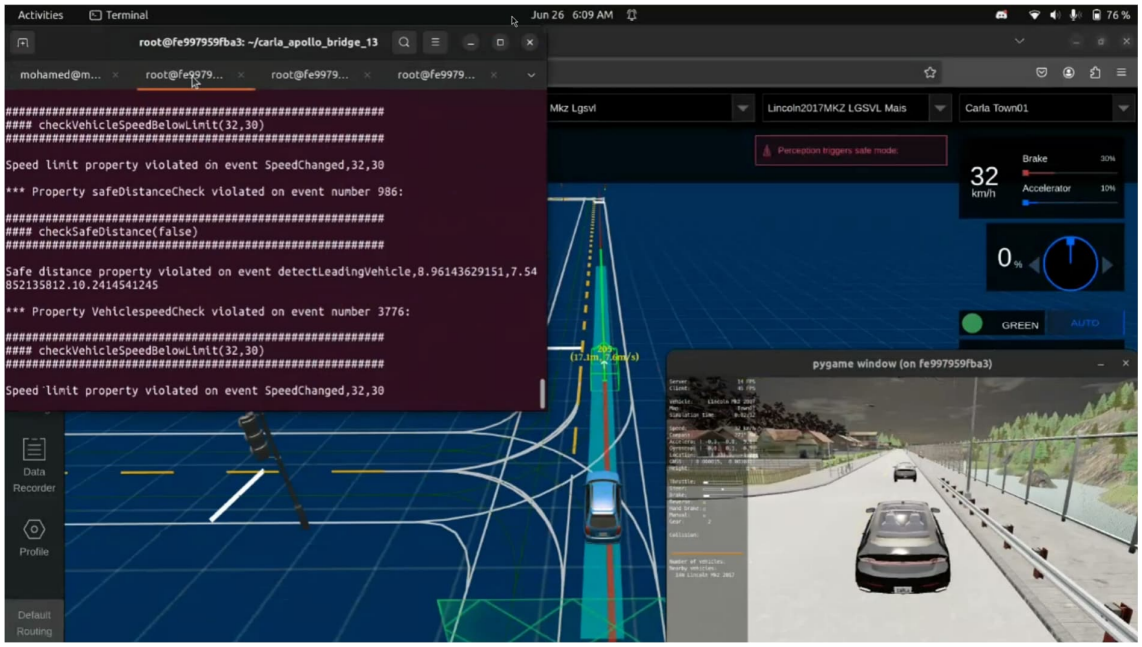

Fig. 5. Digital twin simulation run (Apollo, CARLA, DejaVu-TP).

REFERENCES

[1] ISO. 2018. ISO 26262: 2018 - Road vehicles – Functional safety. Standard. International Organization for Standardization.

[2] ISO. 2019. ISO/PAS 21448: 2019 - Road vehicles — Safety of the intended functionality. Standard. International Organization for Standardization.

[3] ApolloAuto, Apollo: An open autonomous driving platform, Accessed: 2024-06-30, 2024. [Online]. Available: https://github.com/ApolloAuto/apollo.

[4] CARLA - an open urban driving simulator. [Online] Available: https://carla.org/, Accessed: 2024-06-30.

[5] DejaVu Runtime Monitor. [Online] Available: https://github.com/havelund/dejavu.

[6] Anastasios Temperekidis, Nikolaos Kekatos, Panagiotis Katsaros, Weicheng He, Saddek Bensalem, Hisham AbdElSabour, Mohamed AbdElsalam, and Ashraf Salem. “Towards a Digital Twin Architecture with Formal Analysis Capabilities for Learning-Enabled Autonomous Systems”. In MESAS NATO conference for modelling and simulation of autonomous systems, Prague, Czech Republic, Oct. 2022.

[7] Tasneem Awaad, Mohamed Ellethy and Mohamed Abdelsalam. Exploring Software-Defined Vehicles through Digital Twin Simulation with Extensible Prototyping FPGA: A Tool Perspective, DVCon Japan, Tokyo, 29 August 2024.

[8] P. F. D. Heffernan C. MacNamee, “Runtime verification monitoring for automotive embedded systems using the iso 26262 functional safety standard as a guide for the definition of the monitored properties,” Institution of Engineering and Technology, 2014.

[9] S. Shankar, U. V. R, S. Pinisetty, and P. Roop, “Formal runtime monitoring approaches for autonomous vehicles,” in Proceedings of the 2nd Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis, 2020.

[10] E. Zapridou, “Runtime verification of autonomous driving systems in carla,” Bachelor’s Thesis, Aristotle University of Thessaloniki, 2020.

[11] N. O. Vic, Rtamt, GitHub, Available: https://github.com/nickovic/rtamt, 2020.

[12] K. Watanabe, E. Kang, C. W. Lin, and S. Shiraishi, “Invited: Runtime monitoring for safety of intelligent vehicles,” in Proceedings of Institute of Electrical and Electronics Engineers Inc., 2018.

[13] L. Marsso, R. Mateescu, L. Muller, W. Serwe, and U. G. Alpes, “Formally modeling autonomous vehicles in lnt for simulation and testing,” in Mars 2022-5th Workshop on Models for Formal Analysis of Real Systems, 2022.

[14] J. B. P. Rau C. Becker, “Approach for deriving scenarios for safety of the intended functionality,” in ESV, 2019.

[15] H. Wu, D. Lyu, Y. Zhang, et al., “A verification framework for behavioral safety of self-driving cars,” IET Intelligent Transport Systems, vol. 16, no. 5, 2022.

[16] C. Xu, W. Ding, W. Lyu, et al., “Safebench: A benchmarking platform for safety evaluation of autonomous vehicles,” in Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks, NeurIPS, 2022. [Online]. Available: https://safebench.github.io.