Simulation Environment for Validation of Automated Lane-Keeping System

Jiri Vlasak*†, Michal Sojka*, Zdeněk Hanzálek*
*Czech Institute of Informatics, Robotics and Cybernetics, †Faculty of Electrical Engineering,
Czech Technical University in Prague
Jugoslávských partyzánů 1580/3, 160 00 Praha 6, Czech Republic
{jiri.vlasak.2,michal.sojka,zdenek.hanzalek}@cvut.cz

***Abstract*—Automated Driving Systems (ADS) need to be validated in a wide range of conditions to ensure the safety of their operation. It is impossible to validate everything in a real environment, and simulation is the only viable alternative to cover testing under all needed conditions. We present the architecture of a simulation environment based on the CARLA simulator, aimed at validating the Automated Lane-Keeping System (ALKS), the first ADS with available legislation for its approval. We propose to simplify the development and deployment of such a complex software simulation environment through the use of the Nix package manager. We also propose how the example scenarios distributed with CARLA can be extended to make them suitable for validation of ALKS.**

***Index Terms*—Automated Driving System; CARLA simulator; Robot Operating System; Automated Lane-Keeping System.**

I. Introduction

The development of Automated Driving Systems (ADS) requires a lot of validation and testing activities to ensure safe operation. Automated Lane Keeping System (ALKS) is the first ADS for which there is a legal document specifying the requirements for its approval: United Nations (UN) Regulation No. 157 [1]. Compared to similar regulations issued in the past, this document contains only general requirements without specific detailed instructions on what and how to test. It is up to the approving organization to develop precise procedures that will enable it to assess the safety of the ALKS implementation.

To this end, many approval bodies are preparing procedures and technical equipment for the approval process, and similarly, car makers are working on their internal validation and testing procedures. This work is a first step toward the same goal, but on a smaller scale. As an academic institution, we are developing an ALKS-like function to drive a real car under a set of limiting conditions. In particular, we limit the functionality to a subset of weather conditions and omit some required functionality like detection of approaching emergency vehicles. At the same time as the ALKS function, we are developing a simulation environment allowing to test out the ALKS implementation in a virtual environment before deploying it in the real vehicle. The goal is to close the loop between development and validation, gain experience, and provide feedback to other organizations working with real ALKS implementations.

In this paper, we outline the architecture of our initial simulation environment, which is based on open-source software, namely the CARLA simulator. Since the software environment for simulation and validation integrates software components from different sources, version conflicts between different components often occur. We propose to solve these integration problems using the Nix package manager. We publish our initial implementation for use by others. Additionally, we analyze which ALKS validation scenarios, as required by current legislation, can be reused from CARLA and related tools and how they need to be extended for ALKS validation.

This paper is structured as follows. We review related work in Section II. Then, in Section III we describe the architecture of our simulation environment followed by the analysis of simulation scenarios in Section IV. We conclude in Section V.

II. Related work

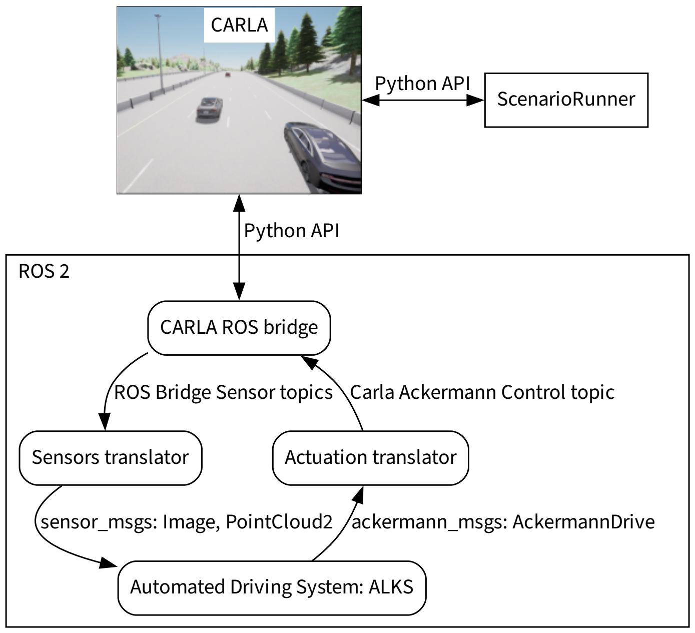
Based on the definitions and taxonomy from SAE International [2], this paper targets verification of ALKS, in the scenario-based simulation environment.

Requirements for the ALKS are given by UN Regulation No. 157 [1] (regulation in short). The regulation introduces the required behavior of the ALKS (Dynamic Driving Task – DDT) within predefined conditions (Operational Design Domain – ODD). However, the regulation lacks exact parameters for the test scenarios.

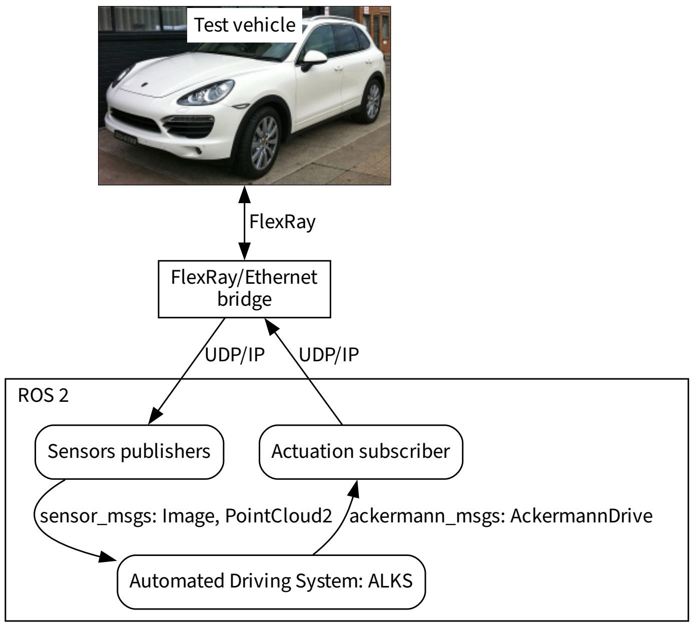
To overcome the problem of missing parameters, Tenbrock et al. [3] present a methodology for finding relevant scenarios in real-world data and extracting the scenarios parameters. They apply their methodology to the highD dataset [4] extracting more than 340 scenarios into OpenSCENARIO format to be later used in CARLA or esmini simulators. They named the extracted dataset ConScenD.

In this work, we use CARLA [5] – an open-source simulator for driving automation systems testing – along with the ScenarioRunner [6] to execute the scenarios. The CARLA simulator already includes basic scenarios like lane-keeping, vehicle following, or cut-in and cut-out situations. We aim at extending these scenarios to cover all parameters from the regulation.

Riedmaier et al. [7] present a state of the art survey on scenario-based approaches to safety assessments of driving automation systems. They discuss two approaches for scenario



(a) ALKS in simulation environment.



(b) ALKS in test vehicle.

Fig. 1. Interfacing of Automated Driving System into simulated and real environments.

generation: knowledge-based scenario generation and data-driven scenario extraction. By this distinction, basic scenarios included in CARLA area knowledge-based, and ConScenD scenarios are data-driven. The authors also discuss the difference between testing- and falsification-based scenario selection. The former approach aims at a subset of scenarios and generalize the results. The latter aims at finding a violation of the safety requirements. Furthermore, the authors identify formal verification as an alternative to scenario-based testing. They propose formal verification for the planning module and the scenario-based testing for the whole system.

Weissensteiner et al. [8] introduce a simulation framework for scenario-based virtual validation of ALKS with its necessary subsystems, including the interfaces between these subsystems. They test the simulation framework on ALKS in two Operational Design Domains using AVL Model.CONNECT, CarMaker, and CARLA.

III. Simulation environment

In this section, we describe the architecture of our simulation environment (Figure 1a). It is based on an open source simulator CARLA [5] and the goal is to test the unmodified implementation of ALKS software developed for a real vehicle (Figure 1b).

A. Architecture

The simulation environment depicted in Figure 1a uses the CARLA simulator as the main simulation engine and its module ScenarioRunner. ScenarioRunner is a CARLA client application written in Python and we use it to initialize the scenario and to control the vehicles (other than the ego vehicle) in the scenario.

We develop the ALKS in the Robot Operating System (ROS) [9] so we also use *CARLA ROS bridge* [10] as an interface between ROS and CARLA. CARLA ROS bridge is a ROS package providing the so called ROS node that acts as a CARLA client and translates data from CARLA to ROS and vice versa. In the ROS terminology, it acts as a ROS publisher for data from CARLA and as a ROS subscriber for data that go in the other direction. CARLA ROS bridge is accompanied by related ROS packages as CARLA Ackermann control and CARLA spawn objects. CARLA Ackermann control allows controlling the ego vehicle in the simulation via well known AckermannDrive ROS messages. CARLA spawn objects is used to manage vehicles simulated in CARLA from ROS. Particularly, we use it to spawn the ego vehicle.

The CARLA ROS bridge publishes data from simulated sensors in CARLA-specific messages. To use them with the ALKS implementation for the test vehicle, we convert them to the format expected by the implementation. This is implemented in the *Sensors translator* ROS node. Similarly, the messages in the other direction are converted from vehicle-specific to CARLA-specific format by the *Actuation translator* ROS node.

The Automated Driving System node in Figure 1a implements the Automated Lane-Keeping System (ALKS). It is a collection of multiple cooperating ROS nodes with specific purposes such as perception, maneuver decision, trajectory planning, and trajectory execution. The internal architecture of the *Automated Driving System* block is out of the scope of this paper.

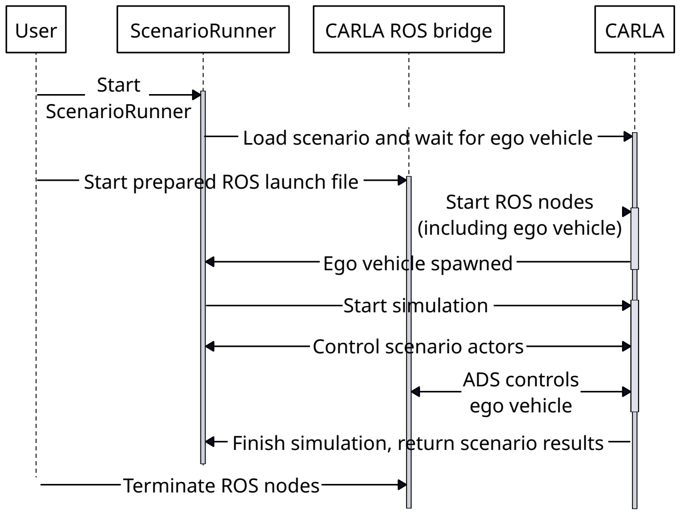


Fig. 2. Sequence diagram of running scenarios in the simulation environment.

B. Execution

To execute the simulations, we run the individual components as shown in the sequence diagram in Figure 2. Currently, we run them manually. Automation is planned for the future.

First, we start the ScenarioRunner with arguments telling it how to connect to the CARLA simulator, which scenario to load, and that the ScenarioRunner should not control the ego vehicle. After the ScenarioRunner prepares the scenario and waits for the ego vehicle, we start all ROS nodes from Figure 1a via a ROS launch file. This step includes spawing the ego vehicle itself. After all required ROS nodes are running and the ego vehicle is spawned, the simulation starts automatically. Individual simulation time-steps are triggered by the ScenarioRunner. ScenarioRunner also controls the movement of simulation vehicles except the ego vehicle, which is controlled by the ALKS ROS node, as shown in Figure 1a. When end conditions of the scenario are satisfied, the simulation ends, and the ScenarioRunner finishes. Then, we manually stop the remaining ROS nodes.

C. Reproducible development environment

The testing environment presented above allows to simulate selected scenarios for ALKS validation. However, setting up the environment on one's computer requires significant effort. For Linux, which we target for our development, CARLA is officially supported only on Ubuntu 18.04. Using it with newer Ubuntu versions works with a few undocumented tweaks. Using it with non-Ubuntu distribution is more difficult. For advanced use of CARLA, such as adding new maps or vehicle models, building CARLA from source is required. This is even more complex, as it needs installing packages from unofficial sources, easily leading to broken systems. At least, this is our experience with students trying to build CARLA themselves. Building CARLA from its source code requires about 130 GB of disk space and hours of compilation time. Any error during the build process multiplies the needed time.

Some of the problems related to building and installing CARLA can be mitigated by using Docker and pre-built CARLA images. However, using Docker brings other challenges that need to be overcome such as making the GPU (Graphics Processing Unit) available inside the Docker containers.

While the above mentioned problems can be resolved with some effort, we argue that the effort is better spent elsewhere than repeatedly trying to reproduce commands from CARLA documentation. Therefore, we propose to use the Nix package manager [11] to manage the software stack for ADS testing and validation. Nix revolutionizes the software building and deployment process by providing strictly controlled environment for software builds, which makes it easy to build complex software stacks reproducibly, i.e., results are bit-by-bit equivalent, in a way that works the same on any Linux distribution. This is achieved by the following features of Nix: (1) Every build command can access only explicitly specified dependencies, the rest of the system is "invisible" to it and (2) any piece of software, called *store object* in Nix terminology, is identified by a hash of all inputs (dependencies) and the commands used to build it. Going into details is out of scope of this paper, but an interested reader can refer to [12], which describes similar problems that we experienced with CARLA, explains how Nix helps to solve them, and why is the Nix solution better than using Docker. The main problem of building CARLA "the Nix way" is the fact that CARLA, and the Unreal Engine, which CARLA is based on, try to achieve reproducible build by its own imperfect way, i.e., by using a custom build system and by downloading prebuilt versions of some, but not all dependencies.

The result of our work-in-progress effort is available in a GitHub repository [13]. Currently, it provides two main functionalities:

- An environment for building CARLA from source. Such an environment contains all the needed dependencies like libraries and compilers in correct versions.
- CARLA client libraries and Python bindings packaged as Nix expressions, allowing their compilation and use on any Linux distribution. This can be used to develop CARLA clients even if binary CARLA packages are unavailable, for example due to the fact that your distribution provides only newer Python versions than those required by CARLA binary packages.

The instructions for how to use the repository are provided in its README file.

IV. Validation scenarios

The UN Regulation No. 157 [1] defines in Annex 5 so called test scenarios, which should be used to validate ALKS implementations. We are interested in simulating those scenarios during ALKS development before testing them in real world.

The ScenarioRunner for CARLA comes with several example scenarios. Here, we analyze which of those example scenarios can be used for validation of ALKS and how they need to be modified or extended for ALKS testing. The

TABLE I. ALKS TEST SCENARIOS AND HOW THEY MAP TO CARLA SCENARIORUNNER EXAMPLES.

ALKS Test Scenario	Similar ScenarioRunner Examples	Extensions	Notes
1. Lane Keeping	FollowLeadingVehicle{5,7}	§ 1 to 5	§ 10
2. Avoiding a collision with a road user or object blocking the lane	StationaryObjectCrossing5, DynamicObjectCrossing5	§ 3 and 6	
3. Following a lead vehicle	FollowLeadingVehicle{5,7}	§ 1 to 5	
4. Lane change of another vehicle into lane	CutInFrom_left_Lane, CutInFrom_right_Lane	§ 7	
5. Stationary obstacles after lane change of the lead vehicle	ChangeLane1	§ 3 and 6	
6. Field of View test	N/A	§ 8	§ 11
7. Lane changing	OtherLeadingVehicle{1,2,4,5,6}	§ 9	§ 12
8. Avoiding emergency manoeuvre before a passable object in the lane	FollowLeadingVehicle{5,7}	§ 1	

(§1) Use different vehicle types of other vehicles, i.e., leading, cutting-in, in the scenario. Namely use: passenger car, powered two-wheeler (PTW), and other vehicle. (§2) Test for different the lead vehicle speeds of (constant, realistic speed profile, braking) and different steering (still, swerving, different lateral positions in lane). (§3) Test different roads segments (straight, various curvatures). (§4) Make scenario timeout longer (5 minutes) and update end conditions. (§5) Extend these scenarios with another vehicle driving close but in an adjacent lane. (§6) Test multiple stationary objects (passenger car, PTW, pedestrian, partially or fully blocked lane, multiple obstacles). (§7) Parameterize scenario with different Times to Collision (TTC), distances, and relative velocities to test scenarios where collision can be avoided as well as scenarios where collision is unavoidable. This includes different longitudinal speed (constant, accelerating, decelerating) and different lateral velocity (constant, accelerating, decelerating) of cutting-in vehicle. Examples of simulations when ALKS should avoid the collision for cut-in, cut-out, and deceleration scenarios are depicted in Appendix 1 of Annex 5 of the UN Regulation No. 157 [1]. (§8) New test scenarios will need to be created. These should contain stationary objects (pedestrian, PTW) on the outer edge of adjacent lanes and within the ego lane and PTW approaching from different directions to the ego vehicle.
(§9) Test different situations when Lane Change Maneuver (LCM) is either possible or impossible due to other vehicles (passenger car, PTW) approaching from different sides of the ego vehicle. (§10) To test the functionality of cruise control without a leading vehicle, we need to create a new scenario because the ScenarioRunner cruise control examples are not positioned on the highway. (§11) These tests target real vehicle sensors. In simulations, properties of simulated sensors can be configured to be arbitrarily good or bad. (§12) Only for ALKS implementations capable of lane change procedure (LCP).

results are summarized in Table I, where each line represents one ALKS test scenario and which of the ScenarioRunner examples are most suitable for simulating it (if any). Each ScenarioRunner example has a name, e.g., *FollowLeadingVehicle* and can be parameterized by several parameters. The particular set of parameters is identified by the number after the scenario name; the values of the parameters are specified in the XML file of the example. If multiple parameter sets are suitable, we denote it with their numbers in curly braces, e.g., *FollowLeadingVehicle{5,7}*.

In addition to ALKS test scenarios from the UN Regulation 157, many other scenarios will need to be created to test full functionality of the ALKS implementation. These include, e.g., scenarios for testing the implementation of the minimum risk maneuver, which should stops the vehicle in a safe way if the driver is not responding to the request to take over vehicle control.

V. CONCLUSION

In this paper, we present an initial version of the simulation environment to validate an implementation of the Automated Lane-Keeping System developed for a real vehicle using the Robot Operating System. The simulation environment is based on the CARLA simulator.

Building the CARLA simulator from source code to add custom vehicles or sensors is a time-consuming task. To avoid this effort, we propose to use the Nix package manager and publicly share the repository with our ongoing work.

Furthermore, we analyzed which CARLA example scenarios are suitable for implementing tests specified in Annex 5 of UN Regulation No. 157 [1].

The goal of our future work is to complete Nix packaging of the CARLA simulator with related tools and develop ALKS validation scenarios in it. The result should be an easy-to-use simulation environment in which it is possible to automatically test the Automated Lane-Keeping System provided as a Robot Operating System packages.

ACKNOWLEDGMENT

This work was supported by the Technology Agency of the Czech Republic under the project Certicar CK03000033.

REFERENCES

[1] ECE/TRANS/WP.29/GRVA, *Proposal for the 01 series of amendments to UN Regulation No. 157 (Automated Lane Keeping Systems)*, https://unece.org/sites/default/files/2022-05/ECE-TRANS-WP.29-2022-59r1e.pdf, May 2022.

[2] SAE International, *J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International*, Apr. 2021. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/ (visited on 08/02/2022).

[3] A. Tenbrock, A. König, T. Keutgens, and H. Weber, "The ConScenD Dataset: Concrete Scenarios from the highD Dataset According to ALKS Regulation UNECE R157 in OpenX," in *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, Jul. 2021, pp. 174–181. DOI: 10.1109/IVWorkshops54471.2021.9669219.

[4] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, ISSN: 2153-0017, Nov. 2018, pp. 2118–2125. DOI: 10.1109/ITSC.2018.8569552.

[5] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, ISSN: 2640-3498, PMLR, Oct. 2017, pp. 1–16. [Online]. Available: https://proceedings.mlr.press/v78/dosovitskiy17a.html (visited on 08/04/2022).

[6] "CARLA ScenarioRunner." (2023), [Online]. Available: https://carla-scenariorunner.readthedocs.io/ (visited on 02/17/2023).

[7] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "Survey on Scenario-Based Safety Assessment of Automated Vehicles," *IEEE Access*, vol. 8, pp. 87 456–87 477, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2993730.

[8] P. Weissensteiner, G. Stettinger, J. Rumetshofer, and D. Watzenig, "Virtual Validation of an Automated Lane-Keeping System with an Extended Operational Design Domain," *Electronics*, vol. 11, no. 1, p. 72, Jan. 2022, Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2079-9292. DOI: 10.3390/electronics11010072. [Online]. Available: https://www.mdpi.com/2079-9292/11/1/72 (visited on 08/04/2022).
[9] "Robot Operating System." (2023), [Online]. Available: https://www.ros.org/ (visited on 02/17/2023).
[10] "CARLA ROS Bridge." (2023), [Online]. Available: https://carla.readthedocs.io/projects/ros-bridge/ (visited on 02/17/2023).
[11] "Nix package manager." (2023), [Online]. Available: https://nixos.org/ (visited on 02/17/2023).
[12] A. Brooks. "Taking off with Nix at FlightAware." (Nov. 2022), [Online]. Available: https://flightaware.engineering/taking-off-with-nix-at-flightaware/ (visited on 12/09/2022).
[13] "Carla simulator Nix packaging." (2023), [Online]. Available: https://github.com/CTU-IIG/carla-simulator.nix (visited on 02/17/2023).