Highly-Modular and Immersive Human-in-the-Loop Driving Simulators Using the CARLA Simulation Environment

Patrick Rebling, Lars Beeh, Philipp Nenninger, Reiner Kriesten
Institute of Energy Efficient Mobility
Karlsruhe University of Applied Sciences
Karlsruhe, Germany
e-mail: {patrick.rebling | lars.beeh | philipp.nenninger | reiner.kriesten}@h-ka.de

***Abstract*—Driving simulators are a common industry tool for verifying driver assistance systems with human involvement. However, there is considerable variation in the hardware and software specifications of these simulators. Consequently, the development of such simulators is often a lengthy process due to the need to create custom software or the high cost of commercial solutions. The goal of this project is to integrate the simulation software Car Learning to Act (CARLA) into highly modular and immersive driving simulators. This will result in the creation of an open source, reconfigurable hardware abstraction that will facilitate the easy and rapid construction of driving simulators that prioritize modularity and extensibility.**

Keywords-simulation; testing; human-computer interaction; automotive.

I. INTRODUCTION

In past years, the development of autonomous driving has been accompanied by a series of optimistic assumptions [1][2]. However, despite significant progress, the road to fully autonomous vehicles capable of seamlessly handling all possible driving situations remains an ongoing challenge [3]. One of the most prominent challenges is the proliferation of mixed traffic scenarios, in which the road is shared by different entities, including automated and autonomous vehicles, cars with human drivers, as well as vulnerable road users, such as cyclists and pedestrians. Understanding, predicting, and replicating human driving behavior in these complex and dynamic environments has emerged as a central but challenging facet of autonomous driving research. The need to address this challenge is not only rooted in safety concerns, but extends to the broader goals of gaining public acceptance [4] and trust [5] in Artificial Intelligence (AI), particularly in the area of self-driving cars [6].

This is the domain in which the Human-In-The-Loop (HITL) methodology is applicable, which is particularly suited to understanding human behavior in complex driving situations without endangering the test subjects. However, there are several challenges associated with HITL driving simulator software. Achieving a high level of realism is critical, as visual, auditory, and tactile feedback must be convincing to ensure realistic driver responses, which requires high-quality graphics and precise input/output synchronization. Ensuring minimal latency between the driver's actions and the simulator's responses is essential, as any delay can disrupt the driving experience and affect the accuracy of the data. The integration of different subsystems can be complex due to different communication protocols and data formats, especially for hardware-related input and output devices, such as realistically behaving force feedback (FFB) motors, different visualization systems, and different driver positions and therefore visualization angles. Designing an intuitive and user-friendly interface is essential for efficient control and quick adjustments. Overcoming these challenges improves the test and verification processes for driver assistance systems in driving simulators, leading to safer and more reliable automotive technologies. The software must be open and adaptable to different driving simulators and scalable for future enhancements without extensive redevelopment.

However, to the best of our knowledge, there is no freely available open source framework that considers all requirements and provides a highly modular hardware abstraction of components for immersive simulators, including realistic steering wheel behavior and easily configurable input and output devices based on the Car Learning to Act (CARLA) simulation environment [7]. The objective of this research is to address the above issues by developing a solution that enables driving simulators to be quickly built and deployed, and that allows them to be connected all over the world.

Section II presents related work to this paper, including highlighting new elements and the need for such a modular, open-source framework for easily integratable driving simulators. In Section III, the approach and implementation of the modular framework is presented. Section IV concludes this paper and highlights areas for further research and development.

II. RELATED WORK

Modularizing software for abstraction is a common practice in software development. The use of Hardware Abstraction Layers (HALs) [8] allows the development of easily reconfigurable software. A popular HAL technology is virtual machines, which simulate operating systems on different host systems [9]. Robot Operating System (ROS) [10] is also widely used for hardware abstraction, as it allows software to be developed in a modular fashion so that hardware-related components can be easily replaced. For example, in the automotive industry, the AUTomotive Open System ARchitecture (AUTOSAR) [11] standard is a common way of abstracting hardware. Furthermore, [12] have developed a HAL for embedded systems with time-triggered hardware access. Simulation environments, such as CARLA on the open source side, or SILAB [13] on the commercial side, often provide a naturally modular architecture with a focus on software interfaces for testing autonomous driving functionality and testing driver assistance systems. In 2008, [14] presented an approach based on MATLAB/SIMULINK for customizable vehicle dynamics in HITL simulators. In the context of driving simulators, [15] presented a modular software-based architecture for integrating developed software for model-based testing of automated driving functions, but focused on specific simulator configurations and lacking open source features. [16] developed modularization in terms of interchangeability of hardware mock-up modules. For example, Realtime Technologies [17] commercially offers its RDS-Modular simulator mock-ups, which can be assembled from predefined modules to meet customer requirements.

To the best of our knowledge, there is currently no freely available open source framework that comprehensively addresses all potential requirements while providing a highly modular hardware abstraction for components used in immersive simulators. This includes features, such as realistic steering wheel behavior and easily configurable input and output devices, all integrated into the CARLA simulation environment. This also allows simulators based on the same simulation environment and framework presented in this paper to be connected all over the world. Thus, the next chapter presents an approach to address these challenges by developing a solution that enables rapid construction and deployment of driving simulators.

III. Approach

The overall concept of our approach for integrating the CARLA simulation environment into a driving simulator is shown in Figure 1.

To achieve a high degree of modularity and extensibility, the processing of driver input, control of environmental variables, display of images, FFB for the steering wheel, and vehicle control are divided into several separate ROS nodes as shown in Figure 2. ROS is a state-of-the-art framework for automated and autonomous driving research, as it enables highly modular software design (see, for example, [18]–[20]). Therefore, a simulator that enables testing of automated driving functions with HITL should also be based on ROS. The nodes communicate via ROS messages with the CARLA-ROS-Bridge and with each other or, if needed, as clients directly with the CARLA server via the API. The vehicle control interface to external driver assistance systems is formed by six ROS topics, allowing testing of, for example, Cruise Control (CC), Lane Keeping Assist System (LKAS), and Lane Change Assist (LCA), and can be easily extended to include further functionality. The following subsections provide a more detailed explanation of each node.

A. Input handling

The *Input Handling* node is responsible for handling all user input from the steering wheel, buttons, pedals, and keyboard. The Python library Pygame [21] is used because it is platform-independent and generally compatible with all game controllers and other input devices. A configuration file contains the assignment of inputs to various program functions, such as activating a turn signal, changing gear, or changing weather conditions. This gives users the flexibility to change assignments and use hardware with different numbers of buttons and axes. A ROS message is issued when an input event occurs. In addition, a CANopen interface is available to publish Controller Area Network (CAN) messages within the ROS environment and to control the ego vehicle via CAN.

B. Time and weather control

The *Weather Controller* node is responsible for controlling the weather and time of day within the simulation. A configuration file contains an expandable list of preset time and weather conditions. At the user's request, the system switches to the next or previous preset by passing the corresponding parameter values to the CARLA-ROS-Bridge, where they are applied to the simulation.

C. Display control

Camera sensors are added to the simulated vehicle at the start. The number of cameras M is equal to the number of monitors or video projectors used in the driving simulator. This value is specified in a configuration file which also contains information about other parameters, such as the resolution, the width of the monitor b, the width of the side monitor frame k, and the distance between the screens and the driver's head d as described in Figure 3. The *Display Controller* node uses these values to calculate the angle f (see (1) and Figure 3) and the horizontal rotation angles o_i (see (2) and Figure 3) of the camera sensors, ensuring that the simulated scene is displayed in a realistic manner.

$$f = 2 \cdot \arctan\left(\frac{\left(\frac{b}{2} - k\right)}{d}\right) \tag{1}$$

where b is the width of a single monitor, d is the distance from driver's head to the monitor surface, and k is defined by the frame width of the monitor. While f remains constant for all camera sensors with identical monitors, the rotation angles o_i of these cameras vary by a factor i, where i is calculated based on whether the number of displays M is an even or odd integer. Thus, i represents a list of numbers for the partial rotation of each monitor:

$$\begin{gathered} o_i = i \cdot 2 \cdot \arccos\left(\frac{d}{\sqrt{\left(\frac{b}{2}\right)^2 + d^2}}\right) \quad \text{with} \\ i = \begin{cases} \pm n \text{ and } n \in \mathbb{N}_0, n \leq \frac{M-1}{2}, & M \bmod 2 = 1 \\ \pm \frac{n}{2} \text{ and } n \in \mathbb{N}, n \leq \frac{M}{2}, & M \bmod 2 = 0 \end{cases} \end{gathered} \tag{2}$$

For example, three monitors result in $i = \{-1, 0, 1\}$. The position of the camera sensors is identical to the position of the driver's head in the vehicle coordinate system, as specified

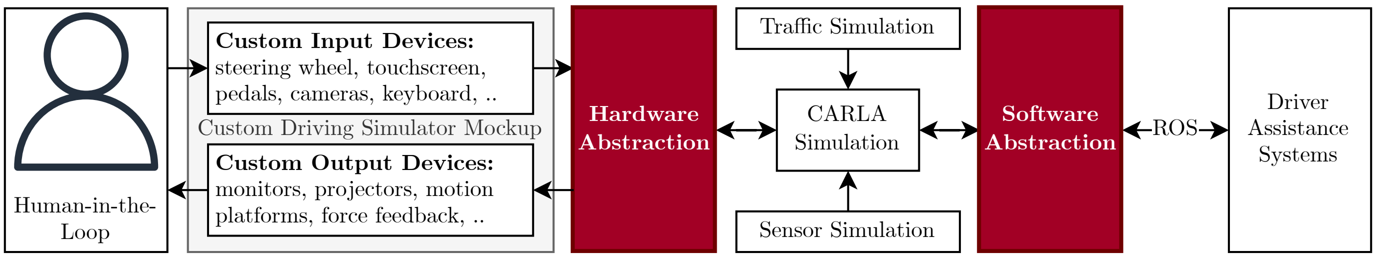
Human-in-the-Loop
Custom Input Devices: steering wheel, touchscreen, pedals, cameras, keyboard, ..
Custom Driving Simulator Mockup
Custom Output Devices: monitors, projectors, motion platforms, force feedback, ..
Hardware Abstraction
Traffic Simulation
CARLA Simulation
Sensor Simulation
Software Abstraction
ROS
Driver Assistance Systems

Figure 1. Abstract concept of the open simulation software for seamless and fast integration into custom driving simulators for HITL tests. The two abstraction modules are presented in this paper.

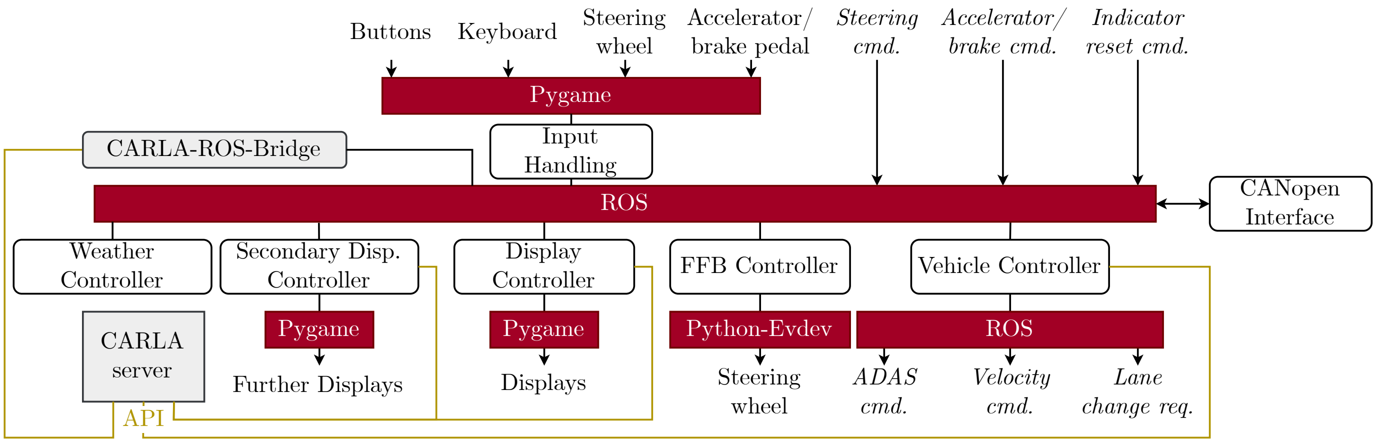

Figure 2. Integration of CARLA into a static driving simulator. Tasks are distributed across multiple ROS nodes (white) that communicate via ROS messages. Hardware interaction is facilitated by the use of the Pygame and Python-evdev libraries. All interfaces are shown in red.

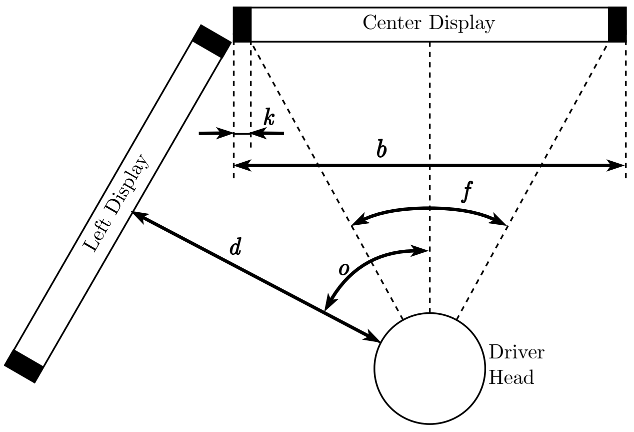

Figure 3. Geometrical description of the display setup of driving simulators with distance from driver head to display d, frame width k, display width b, horizontal field of view f and horizontal display rotation o.

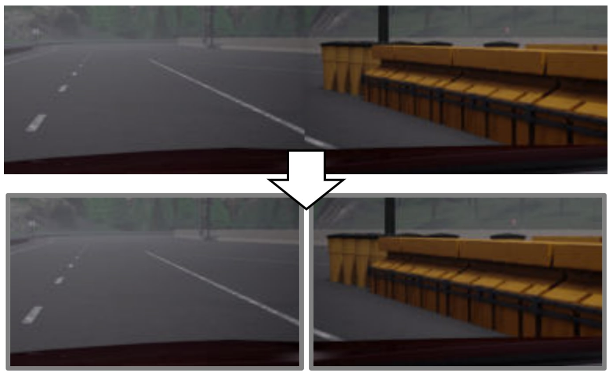

Figure 4. Example of offset compensation for simulators with displays. An automatically calulated offset based on the user-defined parameters resulting in a seamless image on the custom hardware setup.

in the configuration file. For reasons of runtime efficiency, the images from the camera sensors are not received from the server via ROS, but via API. They are then displayed side by side in a Pygame window with the total resolution of the screens as shown in Figure 4. Taking into account the monitor frames, rotations and distances, the image from the camera sensors will appear smooth and without shifting on the driver's monitor.

A secondary display shows important information, such as current vehicle speed, vehicle lighting status, and current gear. The *Secondary Display Controller* can be used to integrate additional displays for mirrors and user interfaces.

D. *Vehicle control*

The *Vehicle Controller* node is responsible for controlling the simulated vehicle's steering, accelerator, brake, shift, and lighting functions. As given in the example programs including the values [22], the steering wheel angle is transmitted nonlinearly to the vehicle wheels (see (3)), with slight modifications to map the steering angle non-linear from -1 to 1. This results in less sensitive steering behavior at small steering angles, which helps prevent the vehicle's lateral dynamics from becoming unstable.

$$f(x) = 0.509 \cdot \tan(1.1x) \tag{3}$$

where x is the raw steering angle received from hardware steering wheel. A similar principle applies to the simulator's accelerator and brake pedals. Gears can be shifted manually by the driver or automatically. While brake lights and rear lights are activated and deactivated according to the vehicle's operating status, most other lights are controlled by the driver, such as high beams or interior lighting. State machines are used to ensure that the functionality of the indicators is consistent with real-world expectations, and to control the switching between parking, low beam, and fog lights. By way of illustration, the state machine for the indicators is shown in Figure 5. It can be seen, for example, that the left-hand indicator is deactivated when the right-hand indicator is activated, and that the hazard warning lights can be temporarily deactivated using the standard indicator switches, but are reactivated when the indicator is deactivated.

Six ROS topics facilitate external control of the vehicle by driver assistance systems, in particular CC, LKAS, and LCA. The node provides the necessary information on whether the driver has activated or deactivated the above systems and whether a lane change has been initiated by activating an indicator. In case of a successful lane change, the indicator can be reset via an external message. The node also provides the desired speed of the vehicle, which is the current speed of the vehicle when CC is engaged. This speed can be increased or decreased by the driver. The actual positions of the accelerator and brake pedals are then overwritten by the external values, except in cases where the driver applies more force to the accelerator than CC requires, such as when overtaking. CC is disengaged in the event of a collision or when the brake pedal is depressed. When LKAS is activated, the actual steering wheel angle is not overwritten by the external angle. Instead, the steering wheel is rotated to the correct position using force feedback effects as described in Subsection III-E. In the event of a collision, lane keeping is disabled.

E. Force feedback

The *FFB Controller* node has two tasks: generating force feedback and controlling the steering wheel angle when LKAS is enabled. These are achieved by using the Python-evdev library [23], which is based on the generic Linux input event interface evdev. This usually ensures compatibility with game controllers.

The inclined steering axis of a car results in steering resistance due to the centripetal force acting on the wheels. Without compensation, the wheels and steering wheel will automatically align in the center position [24]. The force, which is proportional to the square of the vehicle's velocity v, can be simulated with an autocentering effect. The strength s of the effect is calculated as follows

$$s = p \cdot v^2 \tag{4}$$

where p is a user defined constant.

Figure 6 shows the control loop required to turn the steering wheel to the desired angle. To avoid the nonlinearity inherent in the nonlinear transmission of the steering wheel angle y to the wheel angle y^*, the inverse nonlinearity is applied to w^* and the resulting steering wheel angle w is used as the setpoint. The controller output r is the strength of the constant effect. Note that this strength is not unlimited. It is reasonable to expect that small disturbances may occur in the motors or other components of the steering wheel, which are represented as z. When other force feedback effects are disabled or compensated, the wheel behaves like an integrator.

IV. Conclusion and Future Work

This paper presents a highly modular, open source software architecture designed to increase the flexibility and adaptability of driving simulators by enabling seamless integration of different hardware configurations and providing a plug-and-play experience for researchers and developers. By supporting the interchangeability of displays, input devices, and other peripheral components, the architecture promotes a user-centric approach that can accommodate different setups without requiring extensive reconfiguration. An additional feature of the proposed framework is its ability to replicate realistic steering wheel behavior during active Advanced Driver Assistance Systems (ADAS) operations, including scenarios where the steering wheel autonomously rotates to reflect real-world conditions. This enhancement improves the immersive quality of the simulator and provides a more accurate representation of ADAS functionalities and their impact on driver control.

The modularity and hardware-agnostic design of the proposed system makes it an ideal choice for HITL simulations where the flexibility to integrate different input and display devices is essential. This level of modularity supports different research needs, facilitates the testing and evaluation of autonomous driving systems across multiple hardware setups, and allows for easy upgrades or changes to the simulation configuration. Furthermore, integration with ROS allows driving data to be captured for further investigation in specific scenarios. The ScenarioRunner for CARLA [25] enables efficient, scenario-based testing using OpenDrive [26] and OpenScenario [27] data. By incorporating an architecture that is compatible with industry-standard hardware and adaptable to future advances, the framework provides a robust foundation for the continued development of HITL simulation environments and is used in our multi-simulator framework at the Karlsruhe University of Applied Sciences [28].

Future developments will focus on extending the capabilities of the system through full integration with the D-BOX motion platforms, providing enhanced physical feedback for even greater immersion. This addition will allow the simulator to

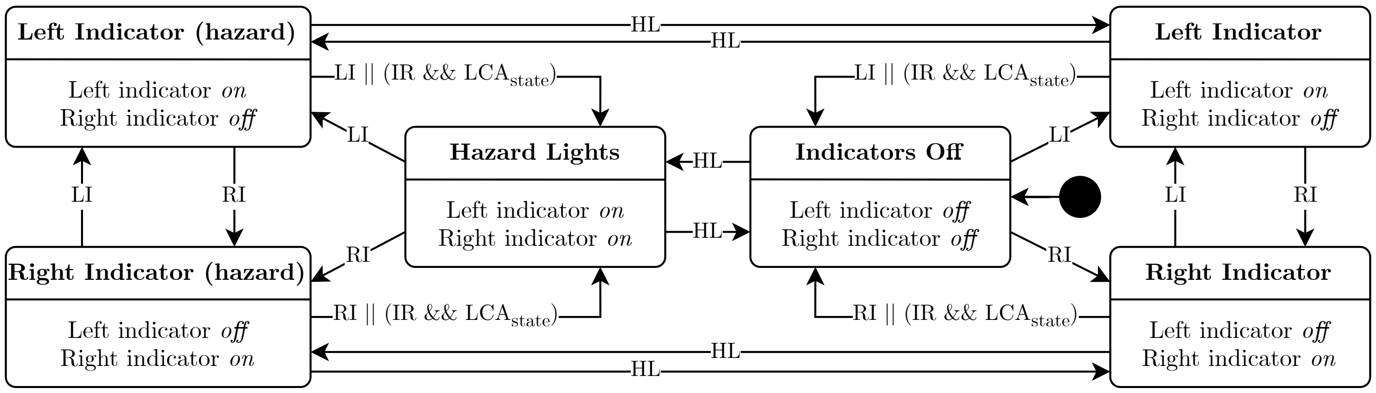

Figure 5. Indicator System State Machine. Events (rising edges): Hazard Light (HL), Left/Right Indicator (LI/RI), and Indicator Reset (IR). For IR to be effective, the state Lane Change Assist (LCA) must be active.

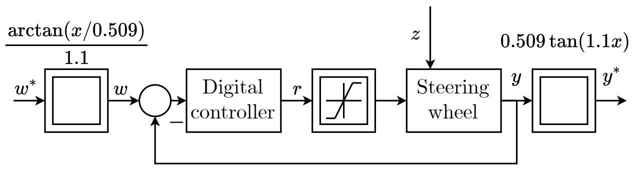

Figure 6. Control loop for steering angle x control according to external setpoints. The nonlinear steering transmission that would otherwise be present in the loop is avoided by using inverse nonlinearity.

convey realistic vehicle dynamics and road conditions to the user, increasing the fidelity of the simulation. In addition, we plan to develop an open interface to support custom motion platforms, allowing researchers and developers to use a variety of motion systems within the simulator framework. By enabling compatibility with a wide range of motion platforms, the simulator will offer increased adaptability, positioning it as a versatile tool for both research and industry applications in autonomous and assisted driving. Support for Windows devices is also provided by adding a DirectInput mode instead of the Linux-specific evdev library. The framework will be made publicly available at https://git.ieem-ka.de/public-repositories/carla-sim.

Acknowledgment

This work was developed in the project KIIWI (reference number: 16DHBKI060) which is funded by the German Federal Ministry of Research, Technology and Space (BMFTR).

References

[1] E. Musk, K. Swisher, and W. Mossberg, *Interview with Elon Musk at Code Conference 2016*, English, 2016. Accessed: Aug. 4, 2025. [Online]. Available: https://www.youtube.com/watch?v=wsixsRI-Sz4.

[2] K. Bimbraw, "Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology," *ICINCO 2015 - 12th International Conference on Informatics in Control, Automation and Robotics, Proceedings*, vol. 1, pp. 191–198, Jan. 2015. DOI: 10.5220/0005540501910198.

[3] B. Padmaja, C. V. K. N. S. N. Moorthy, N. Venkateswarulu, and M. M. Bala, "Exploration of issues, challenges and latest developments in autonomous cars," *Journal of Big Data*, vol. 10, no. 1, p. 61, May 2023, ISSN: 2196-1115. DOI: 10.1186/s40537-023-00701-y.

[4] S. Kelly, S.-A. Kaye, and O. Oviedo-Trespalacios, "What factors contribute to the acceptance of artificial intelligence? A systematic review," *Telematics and Informatics*, vol. 77, p. 101 925, Feb. 2023, ISSN: 07365853. DOI: 10.1016/j.tele.2022.101925.

[5] N. Gillespie, S. Lockey, C. Curtis, J. Pool, and Ali Akbari, "Trust in Artificial Intelligence: A global study," The University of Queensland; KPMG Australia, Brisbane, Australia, Tech. Rep., Feb. 2023. Accessed: Aug. 4, 2025. [Online]. Available: http://doi.org/10.14264/00d3c94.

[6] L. J. Molnar et al., "Understanding trust and acceptance of automated vehicles: An exploratory simulator study of transfer of control between automated and manual driving," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 58, pp. 319–328, Oct. 2018, ISSN: 13698478. DOI: 10.1016/j.trf.2018.06.004.

[7] 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*, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., ser. Proceedings of Machine Learning Research, vol. 78, PMLR, 2017, pp. 1–16. Accessed: Aug. 4, 2025. [Online]. Available: https://proceedings.mlr.press/v78/dosovitskiy17a.html.

[8] A. J. Massa, "The Hardware Abstraction Layer," in *Embedded software development with e-Cos*, ser. Bruce Perens' Open source series, Upper Saddle River, NJ: Prentice Hall, 2003, ISBN: 978-0-13-035473-0.

[9] Y. Li, W. Li, and C. Jiang, "A Survey of Virtual Machine System: Current Technology and Future Trends," in *2010 Third International Symposium on Electronic Commerce and Security*, 2010, pp. 332–336. DOI: 10.1109/ISECS.2010.80.

[10] M. Quigley et al., "ROS: An open-source Robot Operating System," in *IEEE International Conference on Robotics and Automation*, vol. 3, Jan. 2009. Accessed: Aug. 4, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:6324125.

[11] S. Voget, "AUTOSAR and the automotive tool chain," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, event-place: Dresden, Germany, Leuven, BEL: European Design and Automation Association, 2010, pp. 259–262, ISBN: 978-3-9810801-6-2.

[12] G. Simmann, V. Veeranna, and R. Kriesten, “Design of an Alternative Hardware Abstraction Layer for Embedded Systems with Time-Controlled Hardware Access,” English, SAE International, Jul. 2024. DOI: 10.4271/2024-01-2989.

[13] H. Krueger, M. Grein, A. Kaußner, and C. Mark, “SILAB—A Task Oriented Driving Simulation,” 2005. Accessed: Aug. 4, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:11019771.

[14] M. Cipelli, W. Schiehlen, and F. Cheli, “Driver-in-the-loop simulations with parametric car models,” *Vehicle System Dynamics*, vol. 46, no. sup1, pp. 33–48, Sep. 2008, ISSN: 0042-3114. DOI: 10.1080/00423110701882280.

[15] M. Fischer et al., “Modular and Scalable Driving Simulator Hardware and Software for the Development of Future Driver Assistence and Automation Systems,” de, in *New Developments in Driving Simulation Design and Experiments*, A. Kemeny, S. Espié, and F. Mérienne, Eds., ISSN: 0769-0266, Paris, Frankreich, Sep. 2014, pp. 223–229. Accessed: Aug. 4, 2025. [Online]. Available: https://elib.dlr.de/90638/.

[16] F. De Filippo, A. Stork, H. Schmedt, and F. Bruno, “A modular architecture for a driving simulator based on the FDMU approach,” *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 8, no. 2, pp. 139–150, May 2014, ISSN: 1955-2513, 1955-2505. DOI: 10.1007/s12008-013-0182-3.

[17] R. Technologies, *RDS-Modular*, 2024. Accessed: Aug. 4, 2025. [Online]. Available: https://www.faac.com/realtime-technologies/.

[18] S. Kato et al., “Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems,” in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, Porto: IEEE, Apr. 2018, pp. 287–296, ISBN: 978-1-5386-5301-2. DOI: 10.1109/ICCPS.2018.00035.

[19] A.-M. Hellmund, S. Wirges, O. S. Tas, C. Bandera, and N. O. Salscheider, “Robot operating system: A modular software framework for automated driving,” in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, Brazil: IEEE, Nov. 2016, pp. 1564–1570, ISBN: 978-1-5090-1889-5. DOI: 10.1109/ITSC.2016.7795766.

[20] M. Maarssoe et al., “ADORe: Unified Modular Framework for Vehicle and Infrastructure-Based System Level Automation:” in *Proceedings of the 11th International Conference on Vehicle Technology and Intelligent Transport Systems*, Porto, Portugal: SCITEPRESS - Science and Technology Publications, 2025, pp. 571–581, ISBN: 978-989-758-745-0. DOI: 10.5220/0013405200003941.

[21] P. Shinners, *Pygame*, 2000. Accessed: Aug. 4, 2025. [Online]. Available: https://www.pygame.org/.

[22] CARLA, *Python API examples manual_control_steeringwheel.py*, Nov. 2021. Accessed: Aug. 4, 2025. [Online]. Available: https://github.com/carla-simulator/carla/blob/master/PythonAPI/examples/manual_control_steeringwheel.py.

[23] G. Valkov, *Python-evdev Introduction*, 2022. Accessed: Aug. 4, 2025. [Online]. Available: https://python-evdev.readthedocs.io/en/latest/index.html.

[24] C. Smith, *Here’s Why The Front Wheels Automatically Return To Center*, Jan. 2019. Accessed: Aug. 4, 2025. [Online]. Available: https://www.motor1.com/news/299470/why-front-wheels-return-center/.

[25] CARLA Team, *ScenarioRunner for CARLA*, 2024. [Online]. Available: https://github.com/carla-simulator/scenario_runner.

[26] Association for Standardization of Automation & Measuring Systems, *ASAM OpenDrive*, 2024. Accessed: Apr. 28, 2025. [Online]. Available: https://www.asam.net/standards/detail/opendrive/.

[27] Association for Standardization of Automation & Measuring Systems, *ASAM OpenSCENARIO XML*, 2024. Accessed: Apr. 28, 2025. [Online]. Available: https://www.asam.net/standards/detail/openscenario/.

[28] P. Rebling, R. Kriesten, and P. Nenninger, “Towards the Interpretation of Customizable Imitation Learning of Human Driving Behavior in Mixed Traffic Scenarios,” Detroit, Michigan, United States, Apr. 2024, pp. 2024–01–2009. DOI: 10.4271/2024-01-2009.