2025 IEEE/SICE International Symposium on System Integration (SII)
January 21-24, 2025, Munich, Germany

One Stack, Diverse Vehicles: Checking Safe Portability of Automated Driving Software

Vladislav Nenchev

Abstract— **Integrating an automated driving software stack into vehicles with variable configuration is challenging, especially due to different hardware characteristics. Further, to provide software updates to a vehicle fleet in the field, the functional safety of every affected configuration has to be ensured. These additional demands for dependability and the increasing hardware diversity in automated driving make rigorous automatic analysis essential. This paper addresses this challenge by using formal portability checking of adaptive cruise controller code for different vehicle configurations. Given a formal specification of the safe behavior, models of target configurations are derived, which capture relevant effects of sensors, actuators and computing platforms. A corresponding safe set is obtained and used to check if the desired behavior is achievable on all targets. In a case study, portability checking of a traditional and a neural network controller are performed automatically within minutes for each vehicle hardware configuration. The check provides feedback for necessary adaptations of the controllers, thus, allowing rapid integration and testing of software or parameter changes.**

I. Introduction

An Automated Driving (AD) software stack is typically developed focusing on one Vehicle Hardware Configuration (VHC) initially. In the later stages of development these must be ported to operate across a wide range of hardware including different sensor types, actuators, and computing platforms. Traditionally, this process has been manual, iterative and, thus, error prone. Further, an emerging business model in the automotive domain is function-on-demand, where customers initially purchase a vehicle without certain features and later upgrade the functionality as needed. This is often facilitated through over-the-air software updates, which pose additional safety challenges due to the complexity of interacting software and hardware components and the high configurability of modern vehicles.

The safety of an AD function can be compromised by the performance of sensors, actuators, and computing devices [1]. Extensive simulation and real-world testing are typically used to ensure safe operation in a wide range of scenarios. While software testing against selected test cases has proven useful to prevent regression and supports debugging during development, it is non-exhaustive by nature. Formal methods may provide guarantees of completeness to a certain level in the context of AD [2], but they are mainly based on abstract models derived manually from the software deployed in the vehicle. Automatic portability checking methods at code-level become increasingly important for managing hardware diversity and covering the demand for reliable AD systems with a large Operational Design Domain (ODD) including after-sale (e.g. over-the-air) updates and feature releases.

In this paper, formal models of the vehicle and its surroundings are used to automatically check the portability of Adaptive Cruise Control (ACC) software to different VHCs. The derived models encompass common safety-relevant aspects of sensors, computing platforms, and actuators in AD. Given the main functional safety requirement of ACC [3] – to maintain an appropriate safe distance from relevant front objects – a safe set within the ODD is obtained for each VHC. This allows using the method proposed in [4] to check if the closed-loop execution of the ACC code operates safely for each VHC. A case study demonstrates how the check provides essential feedback for conventionally implemented and Deep Neural Network (DNN)-based ACC software for three different VHCs.

The proposed portability check comprehensively covers the entire ODD, but relies on a formal model that may not reflect all behaviors of the real system. Although real-world datasets provide a more accurate representation of behaviors, they may fail to encompass all relevant scenarios within the ODD. Therefore, this approach serves as a complementary method to evaluations based on real-world datasets. It supports rapid integration and testing of software or parameter variations, thus enhancing the efficiency of deploying AD functions across a diverse vehicle fleet.

The remainder of the paper is organized as follows. After summarizing related work (Sec. II), preliminaries for ACC are provided (Sec. III). Then, in Sec. IV, models and safe sets for diverse configurations are obtained to enable portability checking. Finally, the portability of two ACC implementations is checked for three VHCs (Sec. V), followed by a discussion and conclusions (Sec. VI).

II. Related work

Testing is common practice for checking features and functional safety, and is also often applied to show the portability or compatibility of software (updates). However, software compatibility testing [5] is traditionally non-functional, i.e., it ensures that the software application can be executed on the hardware (possibly including a different operating system, communication network etc.). Functional portability testing checks if the software operates according to specified functional requirements that is traditionally implemented as a combination of unit, simulation and manual

V. Nenchev is with the Department of Electrical and Computer Engineering, University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, 85579 Neubiberg, Germany. He was formerly with the BMW Group, Munich. vladislav.nenchev@unibw.de

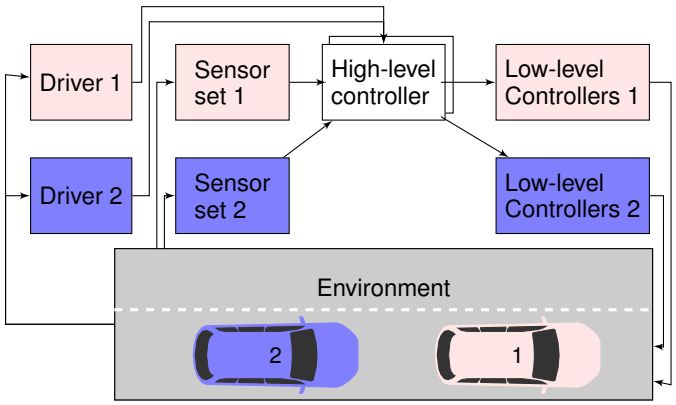

Fig. 1: A common automated driving architecture for vehicles 1 and 2. Differing components are colored in red and blue, respectively. Low-level controllers are configuration-specific, while each vehicle has an instance of the same high-level controller implementation.

real-world tests. Functional testing is often complemented by an automatic search for specification counter-examples, e.g., using reinforcement learning [6], sampling promising operation regions [7]. or extracting abstract logic models from code [8]. Control software executions can also be checked with respect to monitors, which encode formal specifications [9]. Although these methods can identify corner cases that might otherwise be missed, they are not guaranteed to scale effectively for all automotive systems, nor do they offer portability statements for the entire ODD of an AD function.

By using model checking [10], counter-example-guided searching [11] or reachability analysis [12] formal verification of traditionally designed controllers can be achieved with some completeness guarantees. Automatic proofs of closed-loop safety have also been obtained for DNN-based controllers, e.g., [13], [14]. However, these methods do not operate directly with the deployed code, but employ conservative approximations or a model of the controller.

Porting software to new configurations was addressed by deriving a hardware-specific model for formally checking portability and auto-generating low-level code for different OS kernels and drivers [15]. A case study for code-level safety verification based on set invariance was presented for ACC [4]. In contrast to [4], the models employed in this work capture not only additional relevant aspects of the vehicle's basic kinematics and its environment, but also important points of variation for VHCs such as the ego vehicle's dynamics and sensing properties.

To the best of the author's knowledge, this paper is the first to provide an automatic portability check for AD software based on a formal model of safety-relevant configuration-specific functional behavior.

III. Preliminaries

A typical architecture for an AD system [16] is presented for two vehicles with different VHCs in Fig. 1. Each vehicle is equipped with different sensors and actuators, necessitating distinct low-level controllers. Both AD vehicles are assumed to use the same software stack for the high-level controller. For longitudinal guidance, the high-level controller ACC generates an acceleration a, which is provided to the low-level controllers such as the engine control unit, powertrain, transmission, and brake controllers.

In Fig. 1, Vehicle 1 serves as a relevant front object for Vehicle 2. Let h denote the current headway between the two vehicles. The time headway t_h represents the duration until a collision occurs between Vehicle 1 as a front object for Vehicle 2, assuming Vehicle 1 suddenly stops while Vehicle 2 continues moving at its current speed v; this is expressed as $t_h = h/v$. ACC can be activated or deactivated by the driver, who also specifies a desired velocity v_d and a desired time headway t_{h_d}. The desired time headway t_{h_d} to the front object indicates the relative distance that must ultimately be maintained. For simplicity, it is assumed that the driver input interface is the same in all vehicles. The driver inputs form a parameter vector $p_t = [v_d, t_{h_d}]^T$, bounded by

$$P = \{p_t \in [p_{min}, p_{max}]\}. \tag{1}$$

In set speed operation, the ego vehicle's velocity has to eventually reach the driver's desired velocity. The front object is not a concern and only the vehicle's internal actuator limitations need to be respected. In terms of portability, this can be checked using simple testing. In time gap (keep distance) operation, the sensors gather information about relevant front objects, which the ACC uses to determine the desired acceleration a automatically. The ACC ensures that the headway to the front object remains above a minimum value h_{min}, while also keeping the current time headway above a minimum time headway $t_{h_{min}}$. Assuming a bounded velocity of the ego vehicle and the front object, this corresponds to the ODD:

$$O = \{h, v, v_T | h \geq h_{min}, h/v \geq t_{h_{min}}, \\ v_{min} \leq v \leq v_{max}, v_{T,min} \leq v_T \leq v_{T,max}\}. \tag{2}$$

IV. Checking Functional Portability

The goal of this work is to automatically check the safe functional portability of an ACC (denoted by the high-level controller in Fig. 1), implemented in a general purpose programming language (e.g., C/C++), for different VHCs. For that, a model of a VHC n, denoted by σ_n, has to capture safety-relevant behavior. Let the set of all VHC models be $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$. Then, safe functional portability is ensured, when the controller implementation provides only controls a for every relevant platform model $\sigma_n \in \Sigma$, such that the closed-loop operation always remains in (2) for all parameters (1). Testing the portability of keep distance behavior for different VHCs is non-trivial because the ODD O may vary for each hardware setup. In addition, due to, e.g., the internal actuator limitations of the vehicles, not the whole area of O is safe.

First, a formal model is developed that incorporates configuration-specific effects that can compromise safe portability. The workflow depicted in Fig. 2 is employed to check

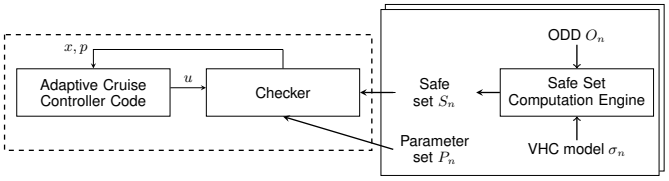

Fig. 2: Portability checking for each VHC n: based on the ODD O_n and the model σ_n, the safe set S_n is computed. Then, S_n is used with the parameter set P_n to check the safe closed-loop operation of the ACC software.

portability, using the framework introduced in [4]. A safe set S_n within the ODD O_n is computed for each VHC model σ_n, where every trajectory originating from this set remains contained within it at all times. To minimize excessive simulation and ensure completeness, checking portability of the high-level controller ACC for each VHC is formulated as a containment problem with respect to the safe set S_n and parameter set P_n. Next, we focus on modeling vehicle configurations.

A. Vehicle hardware configuration modeling

To use model checking for functional portability, effects of the VHC (and low-level software) that have an impact on ACC safety have to be included in a representative model.

1) Vehicle dynamics: As ACC is concerned with the longitudinal motion of the vehicle, a dynamical longitudinal vehicle model is required to describe the vehicle's motion along a path. For that, force-balance equations based on Newton's second law of motion are employed. The vehicle is assumed to move on flat terrain, and exact feedback linearization is used to compensate non-linearities from the aerodynamic drag and rolling resistances [17]. Remaining unmodelled dynamical effects are captured by an additive bounded disturbance $w \in W, W = [w^n_{min}, w^n_{max}]$ to the AD vehicle's velocity. Thus, the vehicle model is given by

$$\dot{v} = c_{n,1}a + c_{n,2}w, \dot{v_T} = a_T, \dot{h} = v_T - v, \qquad (3)$$

where $c_{n,i} \in (0, 1]$ denote configuration-specific parameters. The AD vehicle acceleration a and the object acceleration a_T are bounded with $U = \{u_t = [a_t, a_{T,t}]^T \in [a_{min}, a_{max}] \times [a_{T,min}, a_{T,max}]\}$. As the linear model (3) is a simplified representation of a vehicle's behavior, additional important sensor and actuator effects are considered next.

2) Sensor effects: To perceive its surroundings, a modern AD vehicle uses a mix of cameras, LiDAR, radar, audio, ultrasound, GPS, and inertial measurement. Object detection effects are of main interest for checking functional portability of ACC software.

Different sensors have varying abilities to detect objects based on distance and angle. Current sensor models for testing AD systems are categorized into low-fidelity (ideal), medium-fidelity (probabilistic), and high-fidelity (physical) models [18]. Low-fidelity and some medium-fidelity models provide both object lists and perception data, while high-fidelity models only offer raw measurement data. This paper considers the former type of models, as it examines high-level ACC controllers. Each sensor setup has a maximum distance at which objects can be reliably detected. For each VHC n, a specific maximum detection distance $h_{max,n}$ is estimated, which is used to define a configuration-specific object detection range. The general ODD in (2) is extended by $h \leq h_{max,n}$, yielding a configuration-specific ODD O_n.

The ability to classify objects also varies by sensor type. For instance, a radar may correctly identify a vehicle but fail to recognize a pedestrian in the same position. This object-dependent perception effect is considered by augmenting the parameter set (1) by an additional parameter for object classification $c \in \{\text{car}, \text{pedestrian}\}$, yielding a configuration-specific parameter set P_n.

Measurement uncertainty typically occurs within a limited range around the actual value. An additive disturbance w affecting the AD vehicle's velocity v has already been introduced in (3). A uniform distribution is assumed to account for measurement uncertainties. Given the dynamics (3), the disturbance w also acts indirectly on the headway. This approach helps to evaluate the performance of different sensor setups across various vehicle configurations by treating all potential errors as equally likely. It is important to note that the formal approach presented in this paper is best used in combination with evaluations of portability based on representative real-world data.

3) Actuator effects: When ACC is deployed in the vehicle, there is a significant time delay in executing the input signal by the actuators. This is often caused by the delays in vehicle communication buses or by the dynamics of low-level controllers and actuators. As many AD functions are safety critical, they have to maintain a fixed cycle time t_{cycle} for obtaining a new measurement from the environment through the sensors and providing an updated control a. If a VHC has lower onboard computing capabilities, but the same software is to be deployed, it is possible that an updated control is produced in a following cycle. All these effects accumulate to a substantial control delay τ_n that may compromise safety in a critical situation, e.g., when braking is required instantaneously to avoid a collision.

Considering the aforementioned effects, a combined dynamical model is developed. The overall state at the discrete time t is given by $x_t = [v, v_T, h]^T$, which contains all variables relevant for the ODD (2). The continuous differential equations (3) are converted into discrete time difference equations using exact discretization with an equidistant sampling time $t_s = t_{cycle}$. A zero-order hold is used at a time instant t for the duration of t_s for a and a_T, which are denoted by a_t and $a_{T,t}$ in the discrete time domain. Without loss of generality, the overall control delay is $\tau_n = k_n t_s$ for a given fixed worst-case $k_n \in \mathbb{N}$ for VHC n. Let $a_{t-\tau_n}$ denote the control with τ_n time delay. Thus, the overall model for

each VHC σ_n is

$$x_{t+1}=Ax_t+B_n^a a_{t-\tau_n}+B^f a_{T,t}+E_n w_t, B^f=\begin{bmatrix}0 & t_s & \frac{1}{2}t_s^2\end{bmatrix}^T,$$
$$A=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ -t_s & t_s & 1\end{bmatrix}, B_n^a=\begin{bmatrix}c_{1,n}t_s\\ 0\\ -\frac{c_{1,n}}{2}t_s^2\end{bmatrix}, E_n=\begin{bmatrix}c_{2,n}t_s\\ 0\\ -\frac{c_{2,n}}{2}t_s^2\end{bmatrix}. \quad (4)$$

B. Safe set computation

As described in the previous section, each VHC n may have a different ODD represented by a convex polytope over states and inputs, i.e., $O_n = (O \cap \{h \leq h_{max,n}\}) \times U$. To check the safe operation of a high-level controller (Fig. 1), control signals that keep a subset of the ODD invariant need to be identified. Note that $a_{T,t}$ is considered as an unknown disturbance for the model. The dynamical model with delay (4) is represented equivalently by a higher-dimensional auxiliary system without delay using the Krasovskii approach, which adds states for delayed inputs [19]. Then, the maximal Robust Controlled Invariant Set (RCIS) is computed for the discrete-time auxiliary system with ODD O_n using, e.g., [20], and the corresponding safe set S_n is represented by N_S inequalities, i.e., $S_n = \{x | A_x^n x \leq B_x^n\}, A_x^n \in \mathbb{R}^{N_S \times 3}, B_x^n \in \mathbb{R}^{N_S}$, where A_x^n and B_x^n are matrices defining the set.

C. Checker

A controller implementation is safe with respect to σ_n and the ODD O_n when it operates only within their safe set S_n. The Checker in Fig. 2 is a Bounded Model Checker (BMC) that executes the controller code to verify that it cannot produce a control signal that leads to leaving the safe set. Since the parameters v_d, t_{h_d} and c can only take finite values, checking is done for each feasible combination thereof in parallel.

For a state $x_t \in S_n$, there exists at least one control action a_t, front object acceleration $a_{T,t}$ with $[a_t, a_{T,t}]' \in O_u$ and a disturbance $w_t \in W$, such that the next state x_{t+1} remains within S_n. Thus, the safe portability of the ACC controller is checked as follows. The continuous variables x_t, $a_{T,t}$ and w_t are chosen by the BMC within their admissible bounds using `assume` statements and provided to the to-be-checked controller code that produces an output a_t. Checking that the next state resulting from x_t, $a_{T,t}$, w_t and a_t with (4) is within the safe set is evaluated using an `assert` statement. The BMC also implicitly checks for implementation and security flaws such as integer overflows and pointer errors.

For a DNN-based controller, the Checker (Fig. 2) is based on a three-step approach: first, a finite number of representative states within the ODD are selected to test integration. Next, the original DNN is replaced with a simpler version and BMC is used to verify that all deployment code works correctly. Finally, a dedicated DNN verifier is used to check if the original DNN produces safe a_t for a x_t in the safe set and admissible values of $a_{T,t}$ and w_t. Safe functional portability is confirmed only if all three verification steps are successful. The interested reader is referred to [4] for further details.

TABLE I: Estimated parameter values for the vehicle hardware configuration models.

Parameter	Description	VHC 1	VHC 2	VHC 3
c_1	friction/drag coef	0.95	0.9	0.85
c_2	friction/drag coef	0.1	0.2	0.2
$t_{cycle}[s]$	cycle time	0.2	0.1	0.2
k	time delay coef	1	1	2
$h_{max}[m]$	max sens. range	220	200	200
$a_{min}[m/s^2]$	min acceleration	-4	-4	-4
$a_{max}[m/s^2]$	max acceleration	2	2.2	2.1
$w_{min}[m/s^2]$	min disturbance	-0.05	-0.1	-0.1
$w_{max}[m/s^2]$	max disturbance	0.05	0.1	0.1

V. Case study

The proposed approach is applied for checking the safe portability of two adaptive cruise controllers for three VHCs. CBMC 5.95.1 [21] is used as a bounded model checker with a timeout of 1 hour. The analysis was performed on a workstation with an AMD Ryzen Threadripper PRO 7955WX (16-Core) CPU with 128 GB RAM.

A. Vehicle hardware configurations

All three vehicle configurations are assumed to be equipped with a typical set of AD sensors. The parameters $v_t, v_{T,t} \in [1, 130]\, km/h$, $h_{min} = 5\, m$, $t_{h_{min}} = 0.9\, s$ are common for all configurations. In addition, the boundaries of the front object acceleration are assumed as $a_{T,min} = -1\, m/s^2$ and $a_{T,max} = 0.5\, m/s^2$. Without loss of generality, a fixed desired ego velocity $v_d = 130\, km/h$ and a desired time headway $t_{h_d} = 1.8\, s$ are assumed in the case study.

A small set of reference maneuvers is defined according to [22] and the model parameters are tuned to reproduce the corresponding driving behavior using local optimization methods such as [23]. The estimated values of configuration-specific parameters and bounds are summarized in Table I. The safe sets S_1 of VHC 1 and S_2 of VHC 2 are computed based on the maximal RCIS of (4) augmented by an additional state for a one step actuation delay within their respective ODDs O_1 and O_2. Similarly, the safe set S_3 of VHC 3 is obtained by using two additional states for a two-step actuation delay. The maximal number of iterations for safe set computation with [20] is 100. Under the assumption that all delayed input states and w_t are zero, a 3-dimensional slice of the safe set polytopes is illustrated in Fig. 3. Additionally, a 2-dimensional slice with $v_T = 20\, m/s$ is presented. Note that S_2 has a larger volume than S_1 and S_3 mainly due to the shorter cycle time t_{cycle}.

B. Controller implementations

The first considered type of ACC is based on a Model Predictive Control (MPC) [24], using the model (4), but not accounting for the additive disturbance w_t. Such controllers are commonly used in automated driving and other industrial applications. The front object is assumed to maintain $a_T = 0$ throughout the optimization horizon with $N = 5$ samples.

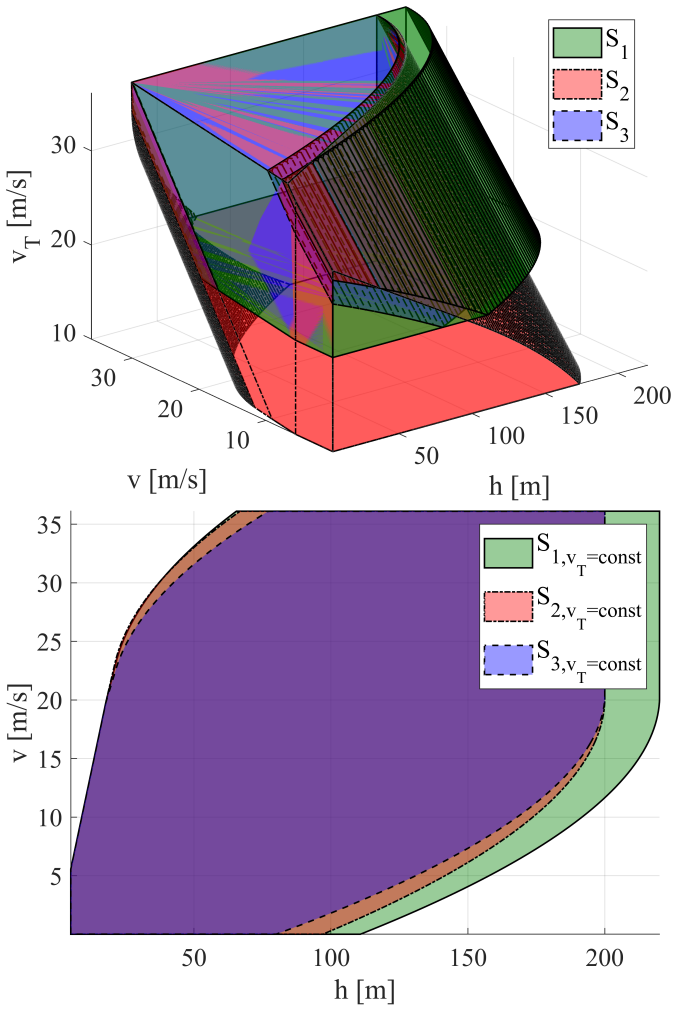

Fig. 3: Slices of the robust controlled invariant sets: 3D for $w_t = 0$ and zero delayed inputs (upper plot), and 2D for additionally $v_T = 20\,m/s$ (lower plot).

Starting from the initial state $x_{\tilde{t}}|_{\tilde{t}=0}$, the following quadratic program is solved at each state x_t:

$$\min_{a_{\tilde{t}}} \sum_{\tilde{t}=0}^{N} (\|v_{\tilde{t}} - \min(v_d, h_{\tilde{t}}/t_{h_d})\| + \|a_{\tilde{t}}\|),$$
$$\text{s.t. } \forall \tilde{t} \in [0, N], x_{t+1} = Ax_t + B_n^a a_t + B^f a_{T,t};$$
$$\forall \tilde{t} \in [1, N], a_{T,\tilde{t}} = 0; a_{\tilde{t}} \in U; x_{\tilde{t}} \in O; x_{\tilde{t}}|_{\tilde{t}=0} = x_t.$$

The controller is implemented using the Multi-Parametric Toolbox [20]. An explicit solution with 348 state feedback controllers was exported to C, which translates to 5521 lines of C code.

The second considered type of ACC is based on a Neural Network Controller (NNC) [25]. This approach combines imitation learning from recorded demonstrations with the optimization of a reward function that includes safety, efficiency, and comfort metrics to maximize cumulative rewards through simulations. Deep deterministic policy gradient is used to learn an actor network together with a critic network. The focus is on verifying the actor with an input x_t and an output a_t. The actor has one hidden layer with 30 neurons, using the rectified linear unit activation function for all layers. The deployment code for this NNC consists of 1672 lines.

TABLE II: Computation times for RCIS in $[min]$ and falsification/verification in $[min]/[min]$ for controllers for different VHCs: '-' left: checking successful, '-' right: counter-example found.

operation	VHC 1	VHC 2	VHC 3
RCIS computation	7.5	29.5	12.1
RCIS inequalities	3642	9460	4199
MPC checking	– / 3.2	– / 7.0	5.2 / –
NNC checking	16.4 / –	– / 35.4	30.1 / –

C. Results

Table II shows results from the RCIS computation for the corresponding VHCs and the portability check of the controllers. The MPC, which was designed to explicitly consider the model (4) and the operation set, was verified for VHC 1 and 2, but not for VHC 3 due to the higher control delay. Interestingly, the NNC was not verified for VHC 1 and 3, but it was verified for VHC 2. This suggests that the control delay plays a more important role than the maximum sensing range for the safety of this configuration. All falsifying scenarios denote an insufficient ego vehicle deceleration, while the front object decelerates with a_{min} and the time headway is not large enough.

D. Discussion

The experimental results demonstrate several benefits of our approach. The portability of controllers were either confirmed or a counter-example, which hints for necessary adaptations of the corresponding VHC or software, was provided (Table II). This allows an evaluation of vehicle hardware effects on AD safe portability, and the possible trade-offs that can be made between the available hardware, the desired ODD and the implemented software. For example, the confirmed portability of NNC for VHC 2 and the discovered counter-example for VHC 1 suggest that safety can be maintained with a smaller control delay even with a lower maximum sensing range. Thus, the method can be used to gather feedback for potential VHCs allowing safe operation with a lower cost or a larger ODD. The proposed portability check is suitable for time-invariant and deterministic controllers. Note that the obtained VHC safe sets can be used as customized supervisors for legacy ACC. Even though ACC was in focus, the presented check is not limited to the considered ODD and model (4). A similar approach can be adopted for other cyber-physical systems. With automated checking possible within minutes, the solution can be integrated into a continuous integration process.

Despite the promising results in the case study, several limitations must be noted. The portability check provides either a compliance proof or a counter-example based on the

employed model and specification. The safe portability check is complete across the ODD; however, it is based on a formal model that, in general, does not describe all behaviors of the actual system. Real-world data may not cover the whole ODD completely, potentially leading to gaps in evaluation. Consequently, the presented approach should be viewed as complementary to assessments using real-world data. An alternative remedy might be to complement the formal model with statistical validation methods in addition to traditional discrepancy analysis techniques [26]. Further, defining the operation and parameter sets for model checking becomes more challenging with an increasing model and specification complexity. Finally, high fidelity models may render RCIS computation infeasible or automated checking intractable. Specifically, larger actuation delays greatly increase the complexity of RCIS computation. In addition, large disturbance sets may lead to an empty RCIS.

VI. Conclusions

A portability checking approach based on formal modeling of safety-critical aspects of vehicle hardware configurations was proposed for adaptive cruise control software. By computing a safe set for these models, the portability of the controller implementation was automatically checked using an existing verification framework. This approach was successfully applied to both a conventionally implemented controller and a neural-network-based controller. The results highlight the potential of formal methods to streamline the adaptation process for automated driving systems, providing a scalable and reliable solution to what is typically a labor-intensive and error-prone integration process.

While the focus was on high-level controller code, low-level software (such as drivers and OS routines) has not been considered. Since the latter are often handwritten for each hardware and may cause additional safety flaws, these should be considered for portability in the future. Further, modeling additional effects from sensors and actuators, as well as other automotive controllers will be studied.

References

[1] T. Gordon and M. Lidberg, "Automated driving and autonomous functions on road vehicles," *Vehicle System Dynamics*, vol. 53, no. 7, pp. 958–994, 2015.

[2] N. Mehdipour, M. Althoff, R. D. Tebbens, and C. Belta, "Formal methods to comply with rules of the road in autonomous driving: State of the art and grand challenges," *Automatica*, vol. 152, p. 110692, 2023.

[3] *Intelligent Transport Systems - Adaptive Cruise Control Systems - Performance Requirements and Test Procedures*, ISO Standard Std. 15 622, 2018, ISO Standard 15622:2018.

[4] V. Nenchev, C. Imrie, S. Gerasimou, and R. Calinescu, "Code-level safety verification for automated driving: A case study," in *Formal Methods (FM'24)*, ser. Lecture Notes in Computer Science (LNCS). Springer, 2024.

[5] A. Haghighatkhah, A. Banijamali, O.-P. Pakanen, M. Oivo, and P. Kuvaja, "Automotive software engineering: A systematic mapping study," *Journal of Systems and Software*, vol. 128, pp. 25–55, 2017.

[6] A. Favrin, V. Nenchev, and A. Cenedese, "Learning to falsify automated driving vehicles with prior knowledge," *IFAC-PapersOnLine*, 2020, iFAC World Congress 2020 (IFAC'2020), Berlin.

[7] G. Chou, Y. E. Sahin, L. Yang, K. J. Rutledge, P. Nilsson, and N. Ozay, "Using control synthesis to generate corner cases: A case study on autonomous driving," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2906–2917, 2018.

[8] V. Nenchev, "Automated behavior modeling for verifying safety-relevant modules," in *Proc. of IEEE Int. Conf. on Robotic Computing (IRC)*, 2021.

[9] ——, "Model checking embedded adaptive cruise controllers," *Robotics and Autonomous Systems*, vol. 167, p. 104488, 2023.

[10] J. Lygeros, D. N. Godbole, and S. Sastry, "A verified hybrid controller for automated vehicles," *Proceedings of 35th IEEE Conference on Decision and Control*, vol. 2, pp. 2289–2294 vol.2, 1996.

[11] O. Stursberg, A. Fehnker, Z. Han, and B. H. Krogh, "Verification of a cruise control system using counterexample-guided search," *Control Engineering Practice*, vol. 12, pp. 1269–1278, 2004.

[12] A. Alam, A. Gattami, K. H. Johansson, and C. J. Tomlin, "Guaranteeing safety for heavy duty vehicle platooning: Safe set computations and experimental evaluations," *Control Engineering Practice*, vol. 24, pp. 33–41, 2014.

[13] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1749–1767, 2023.

[14] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems," in *Proc. 22nd ACM Int. Conf. on Hybrid Systems: Computation and Control*, ser. HSCC'19. New York, NY, USA: ACM, 2019, p. 147–156.

[15] R. Martins Gomes, B. Aichernig, and M. Baunach, "A framework for embedded software portability and verification: from formal models to low-level code," *Software and Systems Modeling*, vol. 23, p. 289–315, 2024.

[16] G. R. Widmann, M. K. Daniels, L. Hamilton, L. Humm, B. Riley, J. K. Schiffmann, D. E. Schnelker, and W. H. Wishon, "Comparison of lidar-based and radar-based adaptive cruise control systems," *SAE Transactions*, vol. 109, pp. 126–139, 2000.

[17] R. Rajamani, *Vehicle Dynamics and Control*, ser. Mechanical Engineering Series. Springer US, 2011.

[18] B. Schlager, S. Muckenhuber, S. Schmidt, H. Holzer, R. Rott, F. M. Maier, K. Saad, M. Kirchengast, G. Stettinger, D. Watzenig, and J. Ruebsam, "State-of-the-art sensor models for virtual testing of advanced driver assistance systems/autonomous driving functions," *SAE Int. J CAV*, vol. 3, no. 3, pp. 233–261, 2020.

[19] M.-T. Laraba, S. Olaru, S.-I. Niculescu, F. Blanchini, G. Giordano, D. Casagrande, and S. Miani, "Guide on set invariance for delay difference equations," *Annual Reviews in Control*, vol. 41, pp. 13–23, 2016.

[20] J. Holaza, L. Galčíková, J. Oravec, and M. Kvasnica, "A software package for mpc design and tuning: Mpt+," in *62nd IEEE Conf. on Decision and Control (CDC)*, 2023, pp. 5682–5689.

[21] E. Clarke, D. Kroening, and F. Lerda, "A tool for checking ANSI-C programs," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, ser. Lecture Notes in Computer Science, K. Jensen and A. Podelski, Eds., vol. 2988. Springer, 2004, pp. 168–176.

[22] T. Menzel, G. Bagschik, and M. Maurer, "Scenarios for development, test and validation of automated vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1821–1827.

[23] H.-J. Wu, V. Nenchev, and C. Rathgeber, "Automatic parameter tuning of self-driving vehicles," in *IEEE Conf. on Control Technology and Applications (CCTA)*, 2024.

[24] G. Naus, J. Ploeg, M. Van de Molengraft, W. Heemels, and M. Steinbuch, "Design and implementation of parameterized adaptive cruise control: An explicit model predictive control approach," *Control Engineering Practice*, vol. 18, no. 8, pp. 882–892, 2010.

[25] M. Zhu, Y. Wang, Z. Pu, J. Hu, X. Wang, and R. Ke, "Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving," *Transportation Research Part C: Emerging Technologies*, vol. 117, p. 102662, 2020.

[26] R. Donà, S. Vass, K. Mattas, M. C. Galassi, and B. Ciuffo, "Virtual testing in automated driving systems certification. a longitudinal dynamics validation example," *IEEE Access*, vol. 10, pp. 47 661–47 672, 2022.