CeCar: A platform for research, development and education on autonomous and cooperative driving

Carsten Thomas, Joachim Wegener, Frank Bauernöppel, Thomas Baar, Heide Brandtstädter

CeCar: A platform for research, development and education on autonomous and cooperative driving

Carsten Thomas*, Joachim Wegener**, Frank Bauernöppel*, Thomas Baar*, Heide Brandtstädter*

* HTW Berlin, University of Applied Sciences,
Berlin, Germany
[name.surname]@htw-berlin.de

** Expleo Group,
Berlin, Germany
joachim.wegener@expleogroup.com

***Abstract* – In this paper, we introduce CeCar as an affordable model-car based platform supporting research, development and education in the field of autonomous and cooperative driving. We present the application-oriented use cases and key platform requirements, and explain the logical and technical architecture of the CeCar platform, alongside with details on the underlying modularity concept. Subsequently, we introduce CeCar application scenarios for the areas research, development and education, and provide relevant application examples. Further, we discuss the CeCar platform concept in comparison with other model-car based education and research platforms, and outline planned future work on the CeCar platform.**

Keywords – autonomous driving; cooperative driving; systems engineering; education.

I. INTRODUCTION

Autonomous and cooperative driving of passenger cars and trucks has been a research area since many years. Initial work goes back to the early nineties, with the Carnegie Mellon NAVLAB project and others spearheading the development [1]. Now, almost 30 years later, the technology is about to enter everyday life, with first products already available on the market and many others in the pipeline. Consequently, autonomous and cooperative driving is currently one of the most vibrant research and development topics. Research and development projects require adequate and affordable platforms for developing novel functionalities – a combination that is difficult to satisfy. In addition, the market calls for engineers with sufficient background in related technologies. Universities currently struggle to fulfill this demand, amongst other reasons also because they lack affordable means to perform education in this field.

Autonomous and cooperative driving brings together many individual technologies and concepts, which at some point have to be integrated and tested in real-life scenarios. For doing this, engineers often rely on full-scale testbeds, i.e. commercially available vehicles that they equip with the required sensors, computation means and actuation devices. Even though engineers are trying to reduce cost of such full-size development platforms, they are inducing significant effort for realization and operation.

As an alternative, developers use simulation to develop algorithms and to perform initial experiments. In such environments, developers often focus on selected aspects of autonomous driving, such as obstacle recognition and autonomous decision-making. Yet, some powerful end-to-end simulation platforms for autonomous driving research and development do exist, e.g. CARLA [2] and AutonoVis-Sim [3].

In some areas, neither full-size-car testbeds nor simulation-based approaches are appropriate. Simulators often concentrate on specific aspects and lack the multi-aspects fidelity required to perform end-to-end experimentation and validation of autonomous driving solutions. Full-size experimental cars excellently cover these multiple aspects, but are prohibitively expensive concerning realization and operation.

We propose to use model cars as an affordable yet powerful platform for performing research and development in the field of autonomous and cooperative driving. In this paper, we introduce a modular platform that provides core capabilities for this purpose, including driving, self-protection, positioning, environment sensing and communication. On top of these core capabilities, researchers, developers and students can build their own solutions. The modularity of the platform ensures that we can tailor individual vehicles to specific applications and use cases, whilst keeping overall architecture and the larger part of pre-available functionality intact. On this basis, we can equally support the diverse challenges of research, development in commercial projects, and education with one single modular platform.

As an affordable and accessible experimentation platform, the CeCar platform lends itself to implementing novel solutions for automated and cooperative driving as part of research activities. Due to its good degree of representativeness, the platform also supports prototyping and validation of concepts and algorithms as part of commercial development work (but without taking credit on those activities for the final product-related verification work). In education, the platform allows students to work on challenging multi-disciplinary development projects that reflect key elements of the state-of-the-art in industry, and to live through all project phases as part of their curriculum.

II. CECAR USE CASES

With CeCar as a multi-purpose experimental platform, we target a wide range of use cases. An initial set of use cases covers basic car functionalities that are necessary as a functional foundation (Figure 1).

This basic set includes use cases for functionalities such as driving, communicating, self-monitoring and self-protecting, which are relevant for all potential applications. Other use cases also cover basic functionalities, but are not relevant for all applications. An example for these is the use case on position data

acquisition, because (absolute) position data are not relevant in some application scenarios.

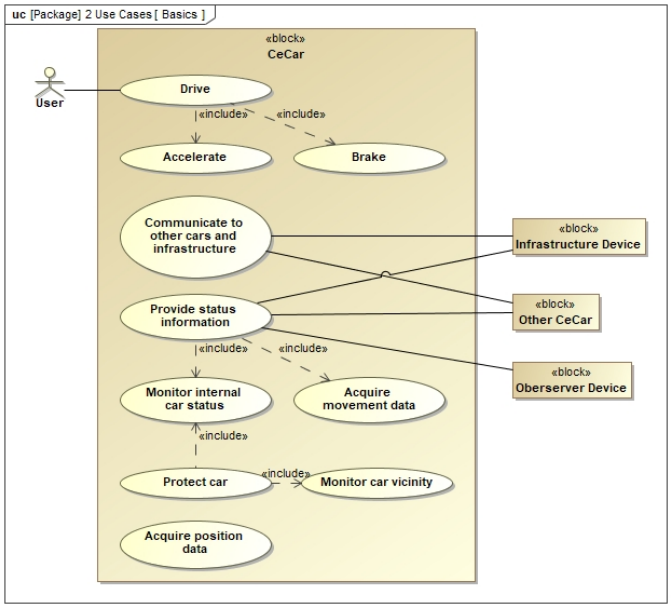

Figure 1: Basic use cases

On top of these basic use cases, we defined a set of higher-level use cases, supporting driver assistance scenarios and automated driving scenarios. The use cases on driver assistance (Figure 2) include well-known applications like speed control, adaptive cruise control and traffic-sign assistance. The use cases on automated driving include applications where a single car operates autonomously, and applications where a car cooperates with other cars and/or with the wayside installations like traffic lights (see Figure 3 for examples).

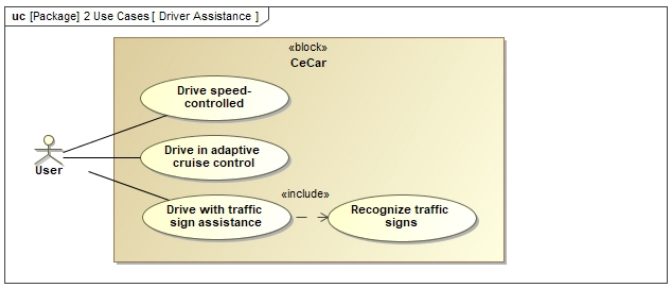

Figure 2: Driver assistance use cases

These sets of use cases cover a wide range of intended CeCar applications. Somewhat orthogonal to this view on use cases, we also had to consider aspects relevant for the use of CeCar as white-box, extensible, representative and affordable experimental platform for research, development and education. This resulted in specific requirements on modularity, size and cost, and in pre-settings on selected technical solutions.

III. CeCar Architecture

We developed the CeCar architecture based on an earlier model car platform named VeloxCar that the Expleo team designed in the context of the EU-funded research project AMASS [4]. To cover the full set of use cases and requirements, we defined a modular architecture that we can adapt and extend for specific applications.

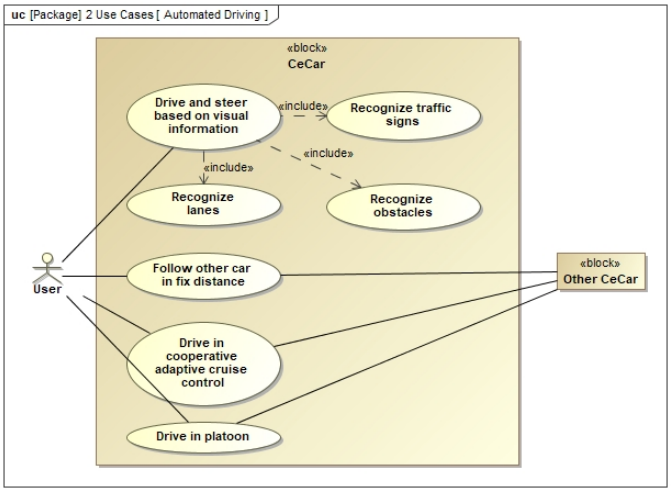

Figure 3: Automated driving use cases (examples)

A. Logical system architecture

The basic logical system architecture of CeCar (Figure 4) is similar to the VeloxCar system architecture. We use encoder and IMU data to compute direction and speed information that feeds into *Direction Control* and *Velocity Control*, and is shared via the *Communication* function with external systems. External systems (including but not limited to a human operator connected via a Control Device) may provide driving commands. We pre-process such driving commands in the *OpMode Control* function according to the operation mode of the CeCar.

The operation mode depends on (1) the CeCar configuration, (2) the current health status of the car, and (3) situational data (influencing the situational adequacy of certain operations). The car configuration is related to the general availability of certain functionality (see the discussion on modularity in section IV), and determines the ability of a configured car to support specific use cases. The health status of the car influences the momentary availability of functionality; the operation mode thereby also reflects the degradation status of the car. Situational data are internal and external states that influence the situational adequacy of certain operations. For example, the Self-Protection function processes car direction and car speed (internal data), and the distance of objects to the car (external data) to compute the risk of the car hitting the detected objects. As a result, the Self-Protection function yields protection commands, that influence the operational mode of the car and define if a certain operation (continued driving into the current direction) is appropriate.

We extend this basic logical architecture with additional functionality as required to support additional use cases. E.g., for the “Drive and steer based on visual information” use case, we add a *Visual Perception* function, and a cascade of video processing, path planning and path execution functions (Figure 5).

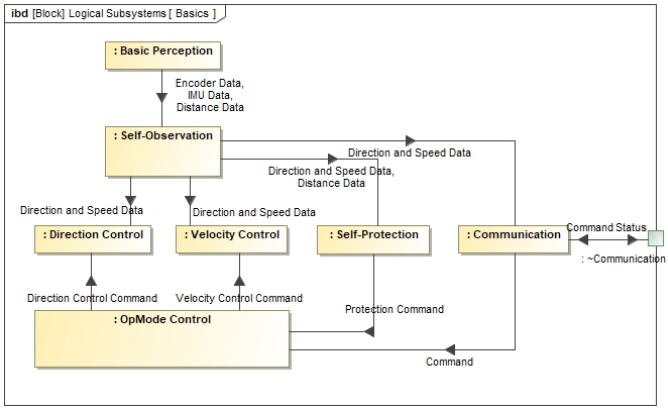

Figure 4: Basic logical system architecture (simplified)

To protect the modularity of the approach, we strive to define purely additive architecture extensions, i.e., extensions that keep existing functions and their interfaces untouched. In cases where such additive extension is not possible, we must introduce new versions of existing functions, which puts additional strain on configuration control for CeCar.

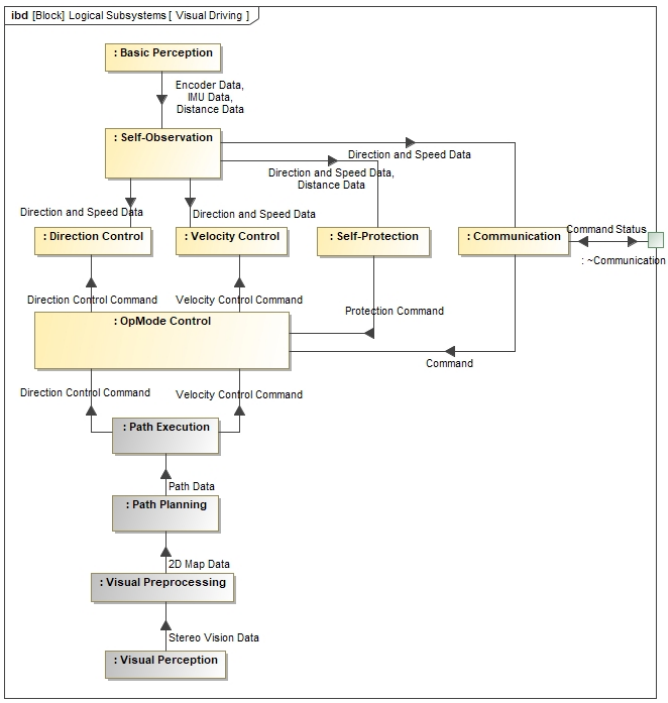

Figure 5: Logical system architecture for vision-based driving (simplified, additional functions in grey color)

B. Technical system architecture

We based CeCar on a commercially available 1/8-scale model racecar kit. To improve driving characteristics of the vehicle, especially to increase its controllable velocity range, we replaced the built-in motor and steering control. We separated low-level and high-level computation onto two control boards, and added multiple sensors, including wheel encoders, accelerometer, ultrasonic and optical distance sensors. Further, we implemented provisions for mounting advanced sensors like lidar and stereo camera. For powering the drive motor, we use a high-current lithium polymer accumulator, while we insert a multiple-voltage power bank to support the controller boards and other components.

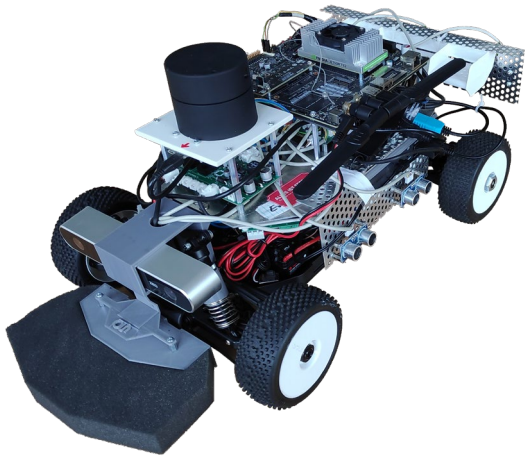

Figure 6: The CeCar platform

In addition, we modified the mechanical construction of the commercial racecar kit. The added electronic components had increased the car weight significantly, so we adapted the suspension system to this higher weight. In addition, we added mounting points for sensors on top of the model car (lidar) and at its front (stereo camera), as well as at all side panels (ultrasonic and optical distance sensors).

For computation, CeCar features two control boards: One real-time control board (the Real-time Control Unit – RCU) – an STM32-based board running FreeRTOS – encapsulating the lower-level control tasks, and one NVIDIA Jetson TX2 board (the Master Control Unit – MCU) for higher-level control, vision-based navigation and similar tasks.

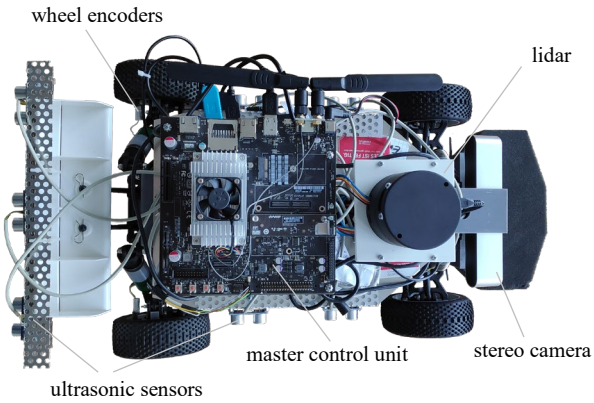

Figure 7: CeCar hardware concept

We decided to use the Robot Operating System (ROS) to implement the MCU software. ROS [5] is a software framework for building robots and clusters of robots, which comprises basic functionality such as hardware abstraction, device drivers and communication, as well as predefined software modules for commonly used functionality, and associated tools. A ROS system consists of a number of independent nodes, each of which communicates with other nodes using a publish-subscribe messaging model. ROS excellently supports software modularity

and allows us to tap into a vast pool of preexisting software modules for sensor data preprocessing and vehicle control. For CeCar, we implemented the MCU software as a set of ROS nodes, thereby mapping relevant parts of the logical system architecture to the MCU ROS node architecture.

The MCU links to the RCU via MAVLink. The RCU hosts all timing-critical software components, including lower-level control functions like *Direction Control* and *Velocity Control*, and high-speed perception functions like *Wheel Encoder Processing* and *Optical Distance Sensor Processing*. The RCU also hosts one part of the *OpMode Control* function, which is necessary to handle potential failures of the MCU or of the communication between RCU and MCU.

The MCU carries all higher-level functions and all functions that require high computational performance. This includes, e.g., the *Visual Preprocessing*, *Path Planning* and *Path Execution* functions introduced in Figure 5, but also the *Communication* function that enables data exchange with other CeCar vehicles, with infrastructure devices such as traffic lights, and with human-machine interfaces. It as well hosts the *Self-Observation* function (that creates status information on the car itself and on its environment by merging all available sensor data), and the *Self-Protection* function (that implements safety features based on status information).

Since the higher-level control software hosted on the MCU is realized in ROS, we use the ROS publish-subscribe mechanisms for car-internal communication. For communication between individual cars, and between cars and infrastructure devices, we originally focused on ROS-compatible communication frameworks such as Coaty [6]. In the future, we will turn to ROS2 and its built-in communication features for car-to-car and car-to-infrastructure communication.

IV. Modularity

To cater for the very different application scenarios, we have put special emphasis onto modularity. We clustered the CeCar functionality into modules that we allocated into two layers: The *Capability Layer* implementing all basic functionalities that developers and students can readily use, and the *Application Layer*, hosting all specific functions developed by the users of the platform.

A. Capability Layer

The Capability Layer contains several capability sets, each of these containing RCU and MCU software modules and associated hardware: The first set, *Foundation*, provides the basic platform capabilities, like driving, steering, self-protection, and communication.

Other capability sets provide additional features that developers can use as pre-available building blocks. Examples are the *Positioning* capability set which includes the lidar hardware and associated mapping and positioning software, and the *Vision* capability set that contains the stereo camera hardware and basic video processing software functions.

B. Application Layer

Using the features available through the capability layer, users can build their own solutions and place them into the *Application Layer*. Applying the CeCar platform, researchers, developers and students have done this already for various autonomous and cooperative driving functionalities, including cooperative adaptive cruise control with connected cars and vision-based autonomous driving. In Section V, we will present some of these CeCar application cases in more detail.

The modularity concept also supports steady evolution of the platform and its growth into additional application areas such as platooning, ad-hoc cooperation, and AI based driving.

C. Product line engineering aspects

There are multiple configurational dependencies between Application Layer solutions and capability sets (hardware modules, RCU software modules and MCU software modules) contained in the Capability Layer. There are "requires" and "contains" relationships, but also "excludes" and other rather complex dependencies.

To systematically analyze and describe system variants and the resulting subsystem / component configuration dependencies, developers often use product line engineering methods and tools. For CeCar, we applied the concepts of product line engineering, but did not yet formally document the configurational dependencies in a variant model. We plan to do this as one of the steps to make the CeCar experimental platform accessible to a larger group of researchers and developers. Likely, we will apply the SysML-based variant modeling method described in [7] for this purpose.

V. Platform Application

A. Application for Research

In the frame of the AMASS research project [4], the Expleo team designed a model car platform named VeloxCar and used that to design, implement and demonstrate algorithms for cooperative driving, applying novel specification approaches like contract-based design [8]. Since then, we have adapted and extended the platform, amending the platform capabilities and improving its modularity.

We now use the VeloxCar and CeCar platform in several other projects focusing on autonomous and connected driving research, including the nationally funded projects CrESt [9] and SiReSS [10]. Two aspects of the CeCar platform are specifically advantageous for this type of application: Firstly, the application of ROS as the foundation for communication allows applying all features and tools of the ROS framework, including ROS-bag playback for development and testing, and ROS 2D and 3D simulators and visualizations for "model-car-in-the-loop" or even "model-car-fleet-in-the-loop" experiments. Secondly, the provision of a communication layer above the individual ROS-based model cars allows to easily implement cooperative scenarios with V2V (vehicle-to-vehicle) and V2I (vehicle-to-infrastructure) communication.

In addition, the modular design approach lends itself easily to stepwise functionality enhancements, and allows research teams to focus solely on the core aspects of their research, re-

using the platform functionality as a strong and proven foundation for their solutions and related experiments and demonstrations.

One example of such research application is the planned demonstration of system-of-systems reconfiguration capabilities in the SiReSS research project. The SiReSS project aims at developing methods for negotiating shared objectives and optimal configurations within systems-of-systems, for applications in, e.g., car platooning scenarios. Using CeCar as a demonstrator platform, the project team can implement their reconfiguration method as Application Layer solution, consisting of SiReSS-specific ROS nodes and re-using all required capability sets provided by the CeCar platform.

As a white-box platform with fully accessible and understood dynamic behavior, CeCar also lends itself to another type of research work: Rigorous validation of underlying control models and formal verification of important safety properties. One typical example is the verification of the *Velocity Control* function. It has to be shown that the car is able to stop always in time under all possible circumstances. The *Velocity Control* function implements a control loop taking into account the current velocity as well as the car's weight and its dynamics during acceleration and deceleration. The physical model underlying the control loop had to be validated rigorously in numerous tests. Once we were sure the physical entity car behaves as predicted by the physical model used in the control loop, we could start to analyze the decision-making algorithms used within the control loop. Prior to applying the theorem prover KeYmaera [11] to formally verify certain safety properties of the algorithms controlling the car, we had to translate our control model (mainly specified in MATLAB) into a model understood by the theorem prover. One of the biggest challenges was to master the complexity of the resulting model in the notion of Hybrid Automata [12]. In this research application, CeCar serves as a very accessible example system to experiment and demonstrate the applicability of novel engineering techniques to real-world problems.

B. *Application for Development*

As for research work, we can use CeCar also for development work in a commercial environment. We use the platform primarily for pre-development and pre-validation of algorithms, before we bring these onto full-size test cars for validation in the real environment. Due to the affordability of the platform, this approach allows to scale and parallelize development activities, without the need to invest into many full-size test vehicles.

This approach works well for camera-related algorithms. We prototype algorithms on the CeCar platform that we can apply on full-size cars with few adaptations. Key for representativeness in this field and for correlation of results between model environment and real world is the use of qualitatively similar camera sensors on CeCar and on the full-size test cars.

Also for prototyping of connected-car algorithms, the CeCar provides a good and affordable testbed. This group of algorithms is not very dependent on hardware and size aspects; therefore, correlation between model-car test results and real-world test results is high.

In this context, the white-box nature of the CeCar platform is important. We sufficiently understand the dynamics of the model car and its "drive train", and capture that in simulation models. When re-using algorithms that we pre-develop on the CeCar platform on the full-size cars, we are therefore able to adapt their parameters according to the differences between the dynamic model of the CeCar platform and the dynamic model of the full-size car.

On the contrary, representativeness is difficult to achieve for environmental sensing algorithms, which strongly depend on the quality and physical positioning of sensors (e.g., ultrasonic sensors, radar, or lidar). Here, the model car and its sensors deviate strongly from the full-size test cars, and hamper comparability and portability of algorithms.

C. *Application for Education*

Curricula at universities often contain project phases, to allow students to apply their acquired knowledge in multidisciplinary settings, and to expose them to the real-world challenges of executing complex development projects in settings with time pressure and mutual dependencies between individuals and teams.

At HTW Berlin, we use the CeCar platform in a master student's course where groups of 5 to 10 students develop a complex technical system over a period of up to 3 semesters. In the first cycle, a student team has extended the core CeCar functionality by adding self-protection and positioning features to the capability layer. In a second cycle, a different team has added vision capability, and is currently working on a vision-based automated driving function using traditional (i.e., non-machine-learning) obstacle detection and route-finding algorithms.

There are several advantages connected to this setting:

- Students do real-world development, using a platform as basis for their work, which reflects the current state-of-the-art in industry. The nature of the platform allows them to focus on low-level technology aspects (real-time computation, communication protocols, sensor integration etc.) as well as on high-level aspects (system architecture, cooperation and reconfiguration concepts etc.).
- Reusing an existing platform design as starting point for own developments, and being forced to document own solutions properly such that others in turn can build on that is a very typical setting for development in industry, and therefore helps to prepare students for their future professional life.
- Building their own solutions on top of an existing platform enables the students to develop very attractive demonstrable products within the restricted timeframe of the course.

Ideally, these projects require very different professions and specializations, such that the students have a chance to contribute all knowledge that they have acquired in their studies, plus additional expertise they might have. The projects require skills from the core computer engineering curriculum, like hardware development and real-time software development, but often also things like control systems development, web-centric user interface design, and design and manufacturing of 3D parts.

In parallel to such group work in projects, we allow individual students to perform bachelor and master theses focusing on specific questions in automated and collaborative driving. Through their thesis works, these students also contribute to the continuous growth of CeCar capabilities.

D. *Affordability and Representativeness*

The total cost of a single CeCar is in the range of a few thousand Euro. In comparison, a full-size experimental car has an initial cost that is likely 10 times higher, and in addition induces significant operational cost.

Despite the low cost, the CeCar platform is very representative of highly automated cars, with respect to dynamic features, available sensors, and computing architecture. Representativeness is limited in areas that relate to model scale, detailed placement of sensors, and certain aspects of vehicle dynamics. In consequence, CeCar is very useful for experimental exploration of algorithms for (highly) automated driving that do not depend on such topics. Specifically for studies on cooperative driving – that usually require several cars to participate in experiments – application of the CeCar platform permits considerable savings. Additional advantages arise from the fact that due to the scale of cars, experiments can be performed inside of buildings or on small test tracks, and require neither protected test circuits nor lengthy authorization processes for the use of public streets.

VI. Discussion

Model car concepts for experimentation and education already exist elsewhere. Probably the most well-known is the MIT RACECAR. The MIT team developed the original version of this model car around 2015, based on a commercial 1/10-scale model racecar kit, and equipped with a Jetson TK1 processor board, stereo camera and lidar sensor [13]. The team promotes the MIT RACECAR for use in education, with students developing software for autonomous driving and performing competitions on this basis. The platform has technologically progressed over time, with alternative versions relying the more powerful Jetson AGX Xavier board and the capable and affordable Jetson TX Nano board announced in 2019 [14].

Other universities have developed similar platforms. In [15], the author presents a concept for a standardized experimental platform that was developed at Chalmers based on a systematic literature review and in-depth analysis of miniature car concepts that have been successful in self-driving car challenges. The software architecture of the platform is modular and uses standardized software interfaces to support adaptation for new or changed application scenarios. On the hardware side, the platform relies on a standardized sensor set that is expected to match a large variety of use cases.

Another example is the MOPED platform developed at SICS, that is fully oriented towards applications for education [16]. Aiming to be representative w.r.t. industrial development challenges, the platform software is intentionally distributed onto three Raspberry Pi controller boards, two of them operating under the AUTOSAR automotive operating system. Whilst this platform is conceptually extensible, it provides only low-level control functions for speed and direction control, such that users of the platform must develop all higher-level functionality on top.

A similar platform is described in [17]. Here, the authors aim to provide a hands-on platform that allows building self-driving model vehicles with minimal effort. Whilst in some areas, for instance on accurate positioning, they excellently provide the required functionality, they stay silent on other required functionality (like vision-based perception and associated processing functions).

Different to these existing model car platforms, we propose a truly modular system that strictly differentiates between the Capability Layer providing the basic features of the platform, and the Application Layer hosting the use-case-specific solutions built on top. Consistent with this modular approach, we separate the control algorithms influencing the core vehicle dynamics, host them on a specific real-time-capable control unit, and keep them stable over time. Thereby we encapsulate the dynamic behavior of the model car, which enables us to capture it in vehicle dynamics models and to rely on this behavior when developing higher-level control functionality.

The CeCar platform supports the same basic objectives as other existing model car experimental platforms, such as representativeness, accessibility, and extensibility. With its pre-defined and managed approach on modularity, CeCar lends itself to stepwise extension and adaptation, and can therefore address the needs of three different application areas in parallel: research, development, and education. Core technical features of the platform, such as its 1/8 scale and the computational power of the MCU enable integration of further hardware components and support functionality growth for the coming years.

VII. Summary and Future Work

CeCar is a very affordable platform for autonomous and cooperative driving. Due to its modularity and the full accessibility of its vehicle dynamics, engineers can easily tailor it to support very different research, development and education objectives.

We plan to increase the usefulness and coverage of the platform further by placing the development data and documentation online (in GitHub or alike), and by formally documenting product line aspects of the CeCar platform using SysML. Following such step, we expect that the existing community of users will start to grow rapidly and that users will make new capability sets and solutions available to the community.

With the lifetime of ROS ending, we plan to port the CeCar platform to ROS2. Whilst this is requiring considerable effort, it will allow redefining and simplifying the CeCar concepts for V2V and V2I communication, and the replacement of Coaty by ROS2-native mechanisms.

VIII. Acknowledgements

We are grateful for the contributions that researchers and students at Expleo, HTW Berlin and other universities have provided for the CeCar platform and its VeloxCar predecessor.

The work presented in this paper is funded by the Berlin Institute for Applied Research (IFAF).

IX. REFERENCES

[1] D. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, Denver, CO, USA, 1988.

[2] 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*, 2017.

[3] A. Best, S. Narang, L. Pasqualin, D. Barber and D. Manocha, "AutonoVi-SIm: Autonomous Vehicle Simulation Platform with Waether, Sensing and Traffic Control," in *2018 IEEE/CVS Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[4] Technalia, "Assurance and Certification of CPS," 2019. [Online]. Available: https://www.amass-ecsel.eu/. [Accessed 21 June 2019].

[5] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, "ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[6] Siemens, "Coaty - The lightweight open-source framework for Collaborative IoT," 2018. [Online]. Available: https://coaty.io/. [Accessed 21 June 2019].

[7] T. Weilkiens, Variant Modeling with SysML, 2016.

[8] B. Zarrouki, V. Klös, M. Grabowski and S. Glesner, "Fault-Tolerance by Graceful Degradation for Car Platoons," in *Workshop on Autonomous Systems Design*, Schloss Dagstuhl, Germany, 2019.

[9] TU Munich, "CrESt - Collaborative Embedded Systems," 2018. [Online]. Available: https://crest.in.tum.de/. [Accessed 21 June 2019].

[10] IFAF, "SiReSS - Sicherheitsrelevante Rekonfigurierende Systems of Systems," 01 April 2019. [Online]. Available: https://www.ifaf-berlin.de/projekte/siress/. [Accessed 21 June 2019].

[11] J.-D. Quesel, S. Mitsch, S. Loos and A. Platzer, "How to model and prove hybrid systems with KeYmaera: a tutorial on safety," *International Journal on Software Tools for Technology Transfer,* vol. 18, no. 1, pp. 67-91, February 2016.

[12] T. Baar and S. Staroletov, "A Control Flow Graph Based Approach to Make the Verification of Cyber-Physical Systems Using KeYmaera Easier," *Modeling and Analysis of Information Systems,* pp. 465-480, 2018.

[13] MIT Aero Astro, "Students' autonomous robots race in MIT tunnels," 6 April 2015. [Online]. Available: https://news.mit.edu/2015/students-autonomous-robots-race-mit-tunnels-0406. [Accessed 21 June 2019].

[14] MIT Aero Astro, "MIT RaceCar Platforms," 24 May 2019. [Online]. Available: https://mit-racecar.github.io/icra2019-workshop/platform. [Accessed 21 June 2019].

[15] C. Berger, "From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation," *Journal of Software Engineering for Robotics,* pp. 63-79, May 2014.

[16] J. Axelsson, A. Kobetski, Z. Ni, S. Zhang and E. Johansson, "MOPED: A Mobile Open Platform for Experimental Design of Cyber-Physical Systems," in *40th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2014.

[17] B. Vedder, J. Vinter and M. Jonsson, "A Low-Cost Model Vehicle Testbed with Accurate Positioning for Autonomous Driving," *Journal of Robotics,* pp. 1-10, November 2018.