Virtual Verification of Cause-Effect Chains in Automotive Cyber-Physical Systems

Ricardo Gonzalez de Oliveira[1,3], Indrasen Raghupatruni[1], Arne Hamann[2], and Achim Henkel[1]

[1] Robert Bosch GmbH: Sys. Eng. BBM Tech. Strategies and Enabling
[2] Robert Bosch GmbH: Corporate Research
[3] University of Luxemburg: Faculty of Science, Technology, and Medicine

Abstract. The technical complexity of automotive Cyber-Physical Systems (CPS) traditionally demands high development and validation efforts. Due to the new technologies entering the automotive market, such as Highly Automated Driving (HAD) ($\geq$ SAE L3) and connected infotainment, the overall system complexity is currently increasing significantly, challenging traditional system development methods and requiring new approaches for validation and verification (V&V). In parallel, new Electric/Electronic (E/E) architecture patterns are emerging in the automotive industry, distributing the functionalities across several multi-core Electrical Control Units (ECU) connected via Ethernet-based in-vehicle networks. This distributed approach leads to complex inter- and intra-ECU timing relations challenging the concept of freedom from interference according to the ISO 26262, and adding another dimension of effects analysis during V&V in the context of ISO PAS 21448 and the upcoming ISO TR 4804. This work enhances a cyber-physical functional simulation tool to include timing effects in distributed cause-effect chains and multi-technology-communication networks (incl. Ethernet and CAN). The resulting simulation allows the system designer to evaluate the impact of timing properties on a given distributed vehicle function, enabling an early validation of the system, avoiding rework during later stages of the development process resulting from wrong design choices.

1 Introduction and Motivation

The future of mobility will be different, driven by the megatrends coming to the automotive market, such as autonomous driving, connectivity, and shared & services [1]. The current status of the automotive Electric/Electronic (E/E) architecture results from the historical growth of the automotive systems, and it is not designed for handling such increasingly advanced functionalities [2]. According to [3], and [4], these new functionalities require stronger co-engineering between different fields such as control, software, and network, leading to not always clear cross-cutting concerns (CCC) between the domains.

Today's automotive E/E architecture systems can range up to 120 decentralized electrical control units (ECUs), and 100 million lines of code [5]. Experts expect to cope with the megatrends, the number of lines of code will increase by a factor of 10,000, with the coming functionalities varying from real-time-systems to interactive apps, transforming the car to a software-defined vehicle. To handle the megatrends reliably, cost-effectively, and with the complexity under control, new automotive E/E architecture patterns are emerging, using state of the art solutions from the information technology world, such as multi-core devices and high-speed Ethernet links.

Figure 1 depicts the progress of the automotive E/E architecture and the patterns, where the left column presents the evolution from a domain-centric approach to a more integrated approach leading to fusion of ECUs and domains, ultimately leading to centralized computers handling the core functionalities of the vehicle. The right column presents a space of solutions for the domain integration. Moreover, the picture shows the evolution from central gateways to a zone gateway architecture. The latter enables the vehicle segmentation in zones, having the legacy communications such as, controller area networking (CAN) and local interconnect network (LIN), at the zone and a high-speed Ethernet link to the centralized vehicle computer as a cross zone communication.

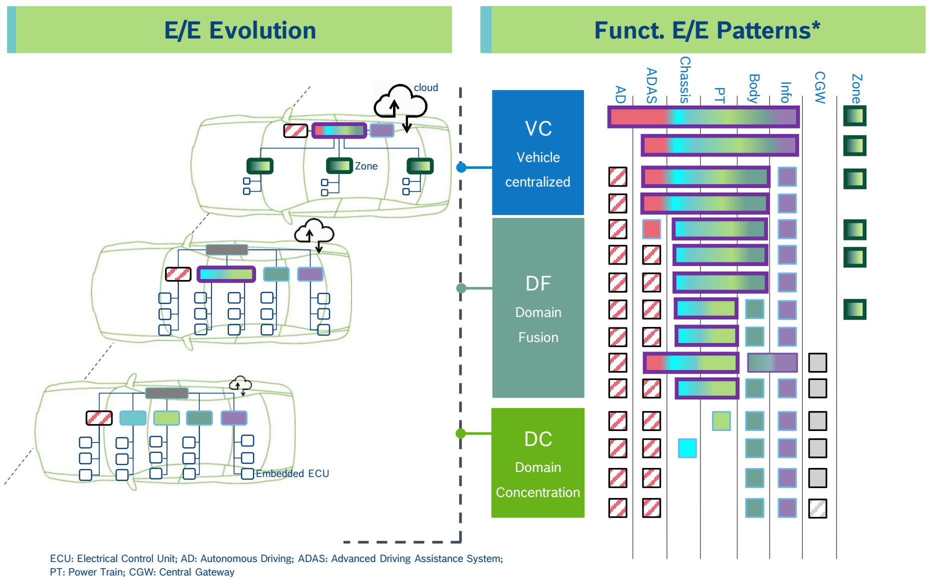

Fig. 1: Automotive E/E architecture evolution and patterns[1] [6]

Those new solutions for the automotive E/E architecture challenge the concept of freedom from interference according to the ISO 26262 [7], where such systems have mixed-criticality processes sharing the runtime environment, mak-

[1]shared housing concept not shown.

ing not trivial the relation between the Intra-/Inter-ECU[2] functionalities leading to complex cause-effect chains. On the Intra-ECU scope, tasks running in multi-core processors could take longer to be executed than expected [8]. On the Inter-ECU scope, data sent by communication channels could present additional delays and jitter [9].

In order to evaluate the impacts of the timing behavior of the runtime environment, this work enhances a functional simulation tool to include a timing model of the Inter- and Intra-ECU performance to enable the system designer to evaluate the impact of the timing properties, allowing an early verification of the system, avoiding rework during later stages of the development process resulting from wrong design choices.

2 Methodology to extract the Timing Behavior of Automotive Cyber-Physical Systems

To present the complexity of the runtime environment timing behavior in automotive cyber-physical systems summarized in the Introduction, this section will explain the timing properties presented in automotive software solutions. Moreover, later will be shown an approach to extract the timing behavior using simulation tools.

2.1 Timing Behavior in Automotive Cyber-Physical Systems

Automotive software systems typically consist of several dozen functionalities that are scheduled with static priority preemptive scheduling policies, typically based on OSEK compliant operating systems. Scheduled entities are called tasks that are repeated cyclically with a fixed period. Each task contains several processes that contain the functional code. Processes are assigned to tasks according to the continuous dynamics of the physical process they control. In the simplest case, one functionality (e.g., one controller) is realized using a single process. However, more complex functionalities are realized using several communicating processes distributed over several tasks, which, in turn, might be executed on different ECUs communicating via in-vehicle networks.

As shown in Figure 2 it is obvious that the scheduling leads to both a sampling jitter (variable time between task release and process execution) as well as a response time jitter of a process (variable time from process start to end), a well-known and well-studied effect in real-time research. Note that for control this means working with stale data (sampling jitter) and applying the control decisions too late (response time jitter). Both effects might thus lead to decreased performance of the control software.

Obviously, in the case that a functionality is realized using several communicating processes (running on the same ECU or even distributed over network)

[2]This work, will make the separation between the computations inside the ECU as Intra-ECU functions, and the ones related to distributed ECUs through the in-vehicle networks as Inter-ECU functions.

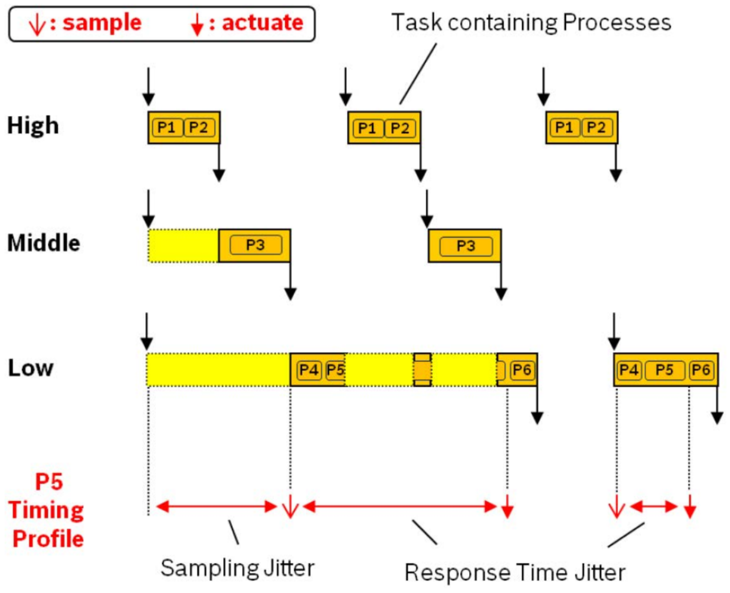

Fig. 2: Sample schedule of a system consisting of three tasks with different priorities. Preemptions by higher priority tasks lead to sampling jitter and response time jitter for process P5 influencing its functional behavior

additional timing effects related to communication need to be considered (compare [10] for examples) increasing the sampling and response time jitters, and, thus, also the impact on the functional behavior.

The mentioned timing affects have practical implication as has been discussed in [12] and [13]. However, both, control engineering and real-time systems engineering often assume idealized system abstractions that mutually neglect central aspects of the other discipline. Control engineering theory, on the one hand, usually assumes jitter free sampling and constant input-output latencies disregarding complex real-world timing effects. Real-time engineering theory, on the other hand, uses abstract performance models [11] that neglect the functional behavior, and derives worst-case situations that have little expressiveness for control functionalities in automotive systems. As a consequence, there is a lot of potential for a systematic co-engineering between both disciplines, increasing design efficiency and confidence. In this paper, we discuss a possible approach for such a co-engineering and show the effectiveness on a real-world example.

2.2 Extracting Timing Behavior and Methodology for Cause-Effect Chains Verification

As described in the Introduction Section, the automotive systems evolve from a domain concentration approach to a vehicle centralized approach, relying on heterogeneous in-vehicle networks to connect the embedded ECUs, sensors, and actuators. Such solutions, with functionalities distributed across the E/E architecture, raises the awareness of cause-effect chains. A cause-effect chain can be understood as the path from an input (e.g., sensor) across the required software elements until the desired output (e.g., actuator) [8]. The Intra-ECU software tasks and Inter-ECU communications' timing properties affect such chains' performance, as described in the subsection before, meaning there are necessary methods to extract such timing behaviors in order to verify the effects.

To verify real-time systems, *response time* is a metric to evaluate the system's performance, giving the latencies and jitters for the processing and forwarding of data through the software devices. There are different techniques to capture those values, such as using hardware tools in a testbed or using simulation environments. In both cases, it is possible to record the timestamps of event changes or states, leading to a statistical distribution of when a task is processed, or a message arrives. Figure 3 illustrates the distribution of a frame's response time in a non-deterministic communication channel, illustrating that most of the events are captured by a testbed setup and the long tail is detected mostly by long simulation runs. Moreover, it is possible to use worst-case analysis or computed bounds (e.g., network calculus [14]) to determine the maximum bounds from the distribution. However, such metrics tend to be pessimistic, where a testbed or a simulation environment could never record such values [15].

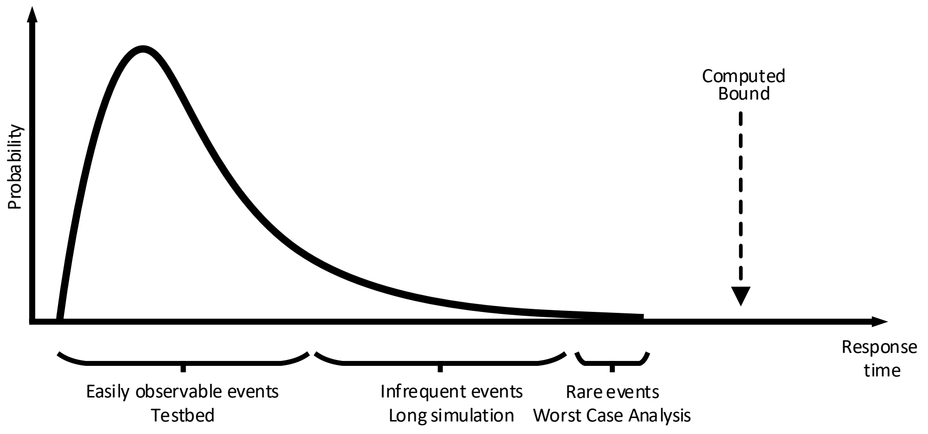

Fig. 3: Metrics for timing behavior and techniques to verify [15]

Figure 4 illustrates an approach to verify the effects of the E/E architecture timing behavior on a given automotive functionality. It starts defining the functional and non-functional requirements for the specific use-case, later leading to the system architecture describing the E/E architecture elements and connections. In order to evaluate the timing behavior, tools for the in-vehicle networks simulation (e.g., RTaW [16]) and embedded multi-core systems (e.g., chronSIM

[17]) can be used. The results from the timing simulator tools are then used to validate the response times and to create timing behavior models that will be included in the functional simulation tool. For a further investigation of the response times, error injection can be done to evaluate how additional effects (e.g., frame drops and transient loads) can affect the timing behavior. Enhancing the functional simulation tool to reproduce the additional delays and jitters from the response time gives the designer a better understanding of how the E/E architecture runtime environment impacts the vehicle functionality cause-effect chains and comes with an argument for the verification of the use case.

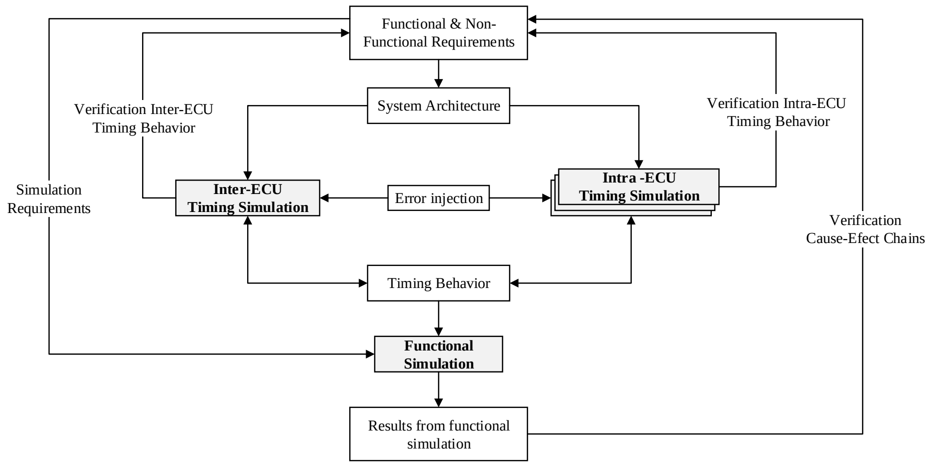
Functional & Non-Functional Requirements
System Architecture
Verification Inter-ECU Timing Behavior
Verification Intra-ECU Timing Behavior
Simulation Requirements
Inter-ECU Timing Simulation
Error injection
Intra -ECU Timing Simulation
Verification Cause-Efect Chains
Timing Behavior
Functional Simulation
Results from functional simulation

Fig. 4: Methodology for the verification of cause-effect chains in automotive functionalities

To evaluate the methodology, Section 3 illustrates an automotive function to be verified, and Section 4 will provides evidences using a simulation setup.

3 Use Case Example and Architecture

The methodology discussed in section 2 is demonstrated in detail using a Lane Keeping Assist (LKA) system as the use case, which is an advancement of the lane departure warning driver assistance system. LKA system supports to correct the course of a vehicle gradually deviating out of its lane due to driver inactivity. Often, an advance warning is given to the driver to correct the steering angle and in case driver does not respond in time, the system applies a small amount of steering to prevent the vehicle from leaving its lane. The paper discusses this use case in the below section 3.1 especially to show the influence of non-functional requirements on the functional behavior of LKA.

3.1 LKA feature and test scenario

LKA systems in advanced vehicles typically use an array of sensors to detect different types of the lane markings in various weather conditions and are rewarded according Euro NCAP, based on a standard set of tests performed on a test track. In this paper we discuss a simplified LKA system with only one camera sensor fitted to behind the windscreen to detect the lane markings and provide the processed inputs to the Highly Automated Driving (HAD) motion control for optimal steering actuation. Test cases to demonstrate the LKA functionality in simulation are performed on a curvy two-lane road with dotted makings as shown in Figure 5.

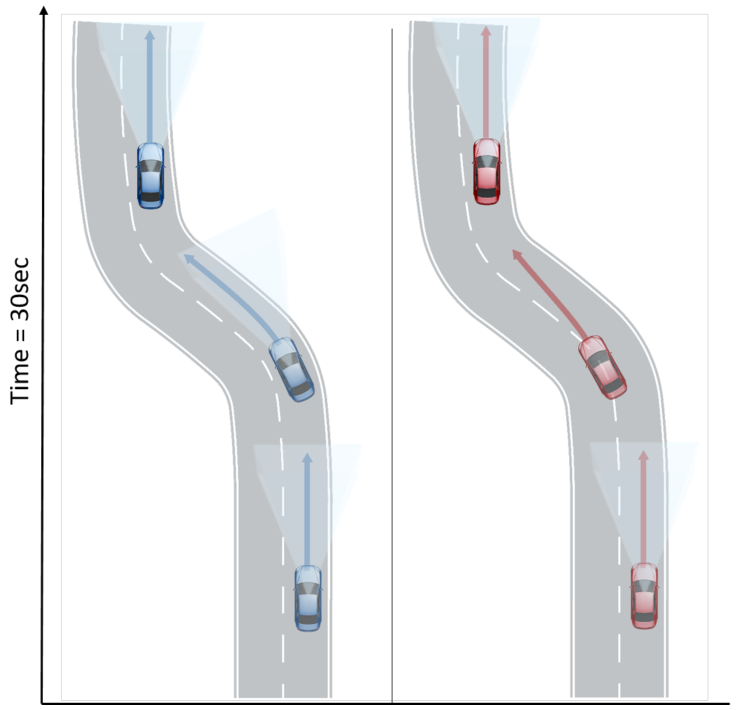

Fig. 5: LKA test scenario. The left figure shows a correct behavior, with the system following the center of the lane. The right picture illustrate a wrong behavior with the system not following the center of the road. The scenario has a total time of 30s with the vehicle in a constant velocity of 50km/h

3.2 System Architecture and Cause-Effect Chain

Figure 6 illustrates the system architecture of the simplified LKA system discussed in this paper. As mentioned in section 3.1, the system use only one camera sensor mounted behind the windscreen of the vehicle. This camera sensor is interfaced to the High Autonomous Driving (HAD) Motion Controller with automotive grade 100BASE-T1 Ethernet protocol according to IEEE802.3 via an Ethernet Switch, which also interfaces other sensors or control units referred to as Other 1-4 in Figure 6. Steering Controller responsible for the lateral dynamics of the vehicle is interfaced to the HAD Motion Controller with CAN-FD protocol according to ISO 11898-1:2015. The HAD Motion Controller is interfaced also to other actuators referred to as Other 5-8 in Figure 6 via the same CAN-FD channel.

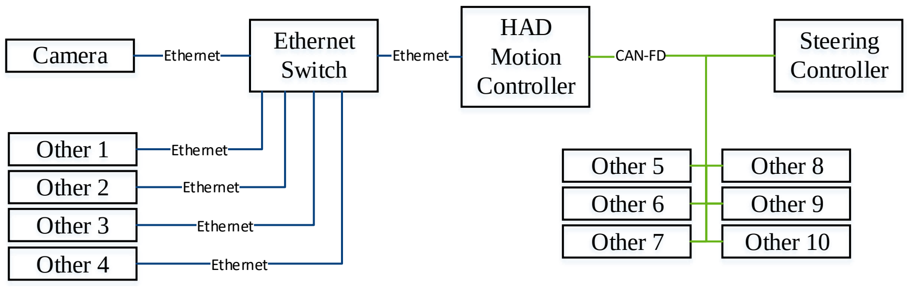

Fig. 6: LKA system architecture overview

It is important to understand these interfaces in the system architecture as the rest of the paper illustrates with the support of simulation, how the latencies and jitters caused due to these communication channels and computations time influence the overall cause-effect chain of the LKA system. Typically, the cause-effect chain in this use-case is $Sens- > Think- > Act$ with Camera sensor, HAD Motion Controller and Steering Controller respectively as shown in the Figure 7 below with a random distribution of latency and jitter between the modules of the cause-effect chain.

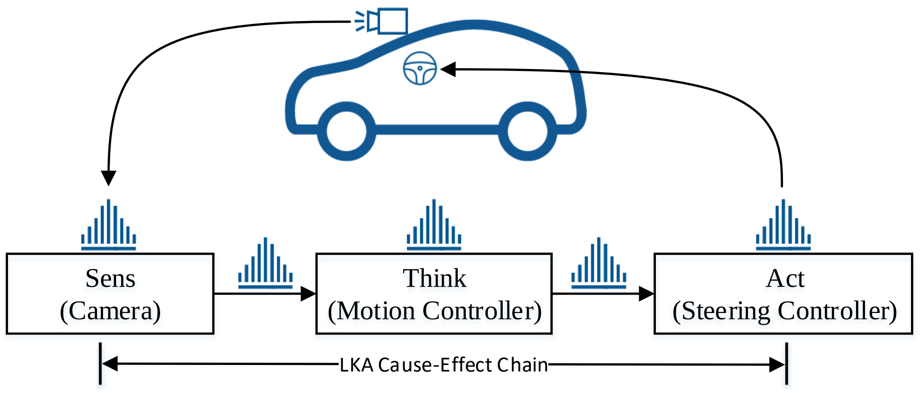

Fig. 7: LKA cause-effect chain

The test cases in simulation are assumed to pass if the vehicle maintains the lane using solely the LKA functionality for a simulation time of 30sec with varied latency and jitter as discussed in Section 4.

4 Simulation Results

To evaluate the timing behavior impact on the LKA use case scenario, it is necessary to build a functional simulation environment capable of simulating the cause-effect chain elements (camera, motion controller, and steering controller) and the environment (car and road). As this work focuses on integrating a novel approach to simulate the response time from the software devices, an of-the-shelf simulation for the LKA system is used [18]. To model the timing behavior, a timing orchestrator block is added to the simulation environment in Matlab/Simulink. The block is responsible for providing the triggers and durations in the simulation for the response time of the LKA software components.

Using the timing simulation tools approach presented in Section 2.2, the camera's timing behavior, the Ethernet network, the HAD motion controller, the CAN-FD network, and the steering controller are generated. The figure 8 presents in details the response time distribution of the camera data. The camera system generates and sends the images in $30Hz$ in a burst of Ethernet frames. In the picure it is possible to observe that the delay for all the frames to reach the HAD motion controller is around $10ms$ with a probability deviation up to $10.20ms$. The computed bound is $10.24ms$. Moreover, the HAD controller has a cycle time of 0.1ms$\pm$0.01 to process the data and send the steering angle by the CAN-FD Bus. The CAN frame of the steering angle has a delay of $0.2ms$ up to $0.5ms$. The response time of the steering controller was neglected in this work.

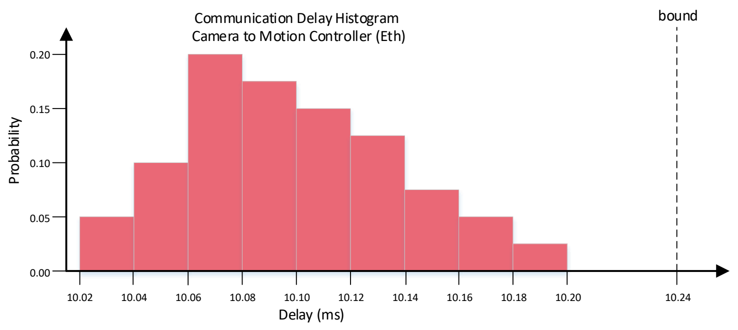

Fig. 8: Transversal network time: Camera eth. frames to HAD motion controller

As a result of the simulation, the figure 9 illustrates two simulation outcomes: One assuming a perfect technology with no delays and jitter from the response time of the systems, and another using the response time values described before (nominal response time). As can be seen from the figure, the one with the additional timing behavior follows the center of the lane with almost no relative deviation from the perfect technology, however, with a more oscillatory control signal for the steering angle.

As a further investigation, the system elements' response time was increased to see which values the functionality will not be able to follow the center of the lane. This investigation comes from the fact that transient or permanent faults on the system could increase the response time (e.g., Babbling Idiot [19]). With an increase of $6ms$ on the camera Ethernet frame jitter and $10ms$ on the steering angle CAN frame jitter compared to the nominal response time, the system is unstable. Figure 10 shows the results.

5 Conclusion

In this paper we discussed a methodology to extract different timing behaviors of the automotive cyber-physical systems using state of the art timing simulation tools. With a Lane Keep Assist system as use case we then demonstrated the effects of timing propagation on the functionality, especially the lateral maneuver in a simulation environment. Using a novel timing orchestrator we then analyzed the maximum transportation delay allowed to keep the vehicle in the desired lane. The feasibility of the approach shall be validated in the future by comparing with the real vehicle behavior as well and the usage of timing orchestrator in a co-simulation environment shall be explored with multiple control units.

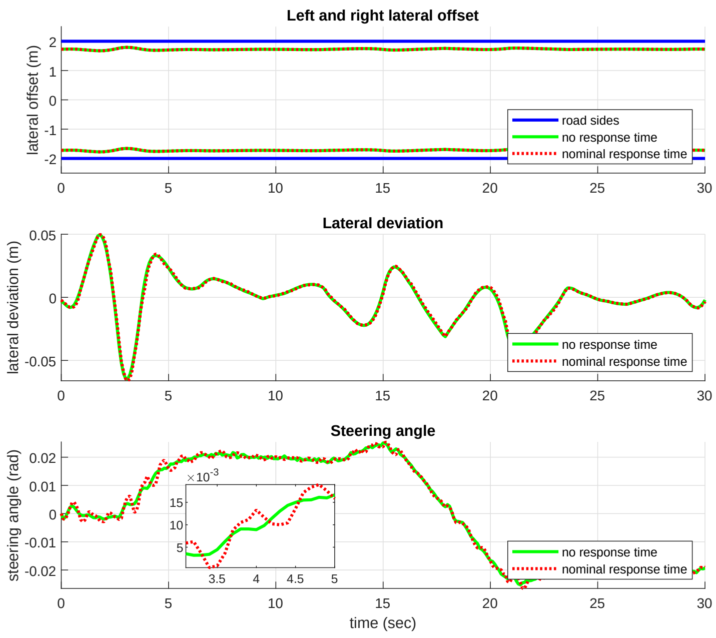

Fig. 9: Comparison of LKA performance with no delays/jitter (perfect technology) and delays/jitter from the processing units and communication networks. Zoom at the steering angle signal between 3-5s to illustrated the slight more oscillatory behavior due to the response time delays

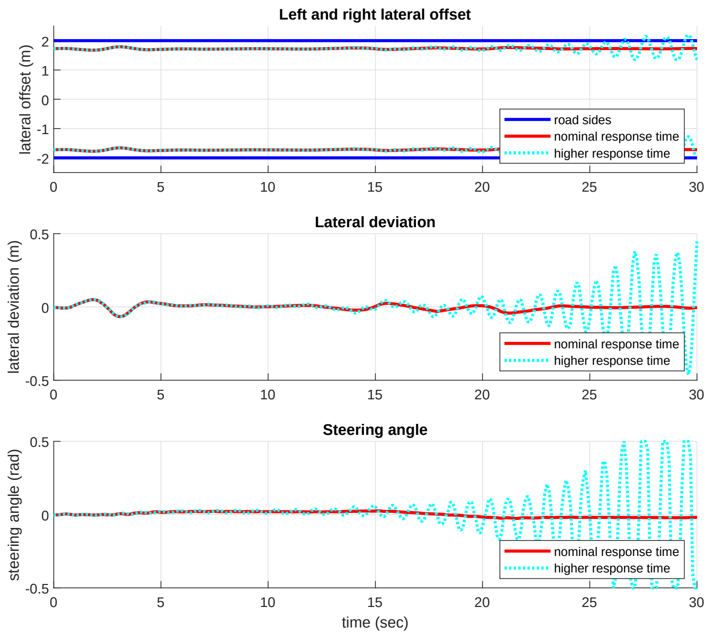

Fig. 10: Comparison of LKA performance with nominal response time and increased response time. With an additional delay of $6ms$ for the camera Ethernet frames and $10ms$ for the steering angle CAN frame, the functionality is unstable

References

1. C. Ehlers: Mobility of the future connected, autonomous, shared, electric. In: 30th International AVL Conference Engine & Environment, pp. 175-177, Graz (2018)
2. A. Kampmann, B. Alrifaee, M. Kohout, A. Wstenberg, T. Woopen, M. Nolte, L. Eckstein, and S. Kowalewski: A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland (2019)
3. P. Derler, E. A. Lee, M. Trngren, and S. Tripakis: Cyber-physical system design contracts. In: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems, pp. 109118. (2013)
4. M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann: Engineering Automotive Software. In: Proceedings of the IEEE, vol. 95, pp. 356-373. (Feb 2007).
5. ETAS: Developing vehicles of the future. Entering new worlds. https://www.etas.com/download-center-files/DLC_realtimes/RT_2019_2020_en_6_rgb_02-2020.pdf
6. R. G. de Oliveira, C. Kerstan, and A. Henkel: Keynote: Service-Oriented Architecture. Chances and Challenges. In: Automotive Ethernet Congress, Virtual. (2021)
7. ISO: Road vehicles - functional safety. International Organization for Standardization, Geneva, Switzerland, ISO 26262. (2018).
8. R. Mnzenberger and O. Schmidt: From Assisted to Autonomous Driving and Beyond. Taking control of system timing challenges in embedded automotive systems. Whitepaper (2019)
9. N. Finn: Introduction to Time-Sensitive Networking. In: IEEE Communications Standards Magazine, June (2018)
10. A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst: Communication centric design in complex automotive embedded systems. In: 29th Euromicro Conference on Real-Time Systems (ECRTS). (2017)
11. R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst: System level performance analysis - The SymTA/S approach. In: IEE Proceedings-Computers and Digital Techniques 152 (2), pp. 148-166. (2005)
12. D. Ziegenbein and A. Hamann: Timing-aware control software design for automotive systems. In: Proceedings of the 52nd Annual Design Automation Conference. (2015)
13. S. Lampke, S. Schliecker, D. Ziegenbein, and A. Hamann: Resource-aware control-model-based co-engineering of control algorithms and real-time systems. In: SAE International Journal of Passenger Cars-Electronic and Electrical Systems. (2015)
14. J.-Y. Le Boudec and P. Thiran: Network Calculus. Springer. (2001)
15. N. Navet, S. Louvart, J Villanueva, S. Compoy-Martinez, and J. Migge: Timing verification of automotive communication architectures using quantile estimation. (2013)
16. N. Navet, J. R. Seyler, and J. Migge: Timing Verification of Realtime automotive Ethernet networks: What can we expect from simulation? In: SAE 2015 World Congress. (2015)
17. S. Anssi, K. Albers, M. Drfel, and S. Grard: chronVAL/chronSIM: A Tool Suite for Timing Verification of Automotive Applications. In: RTS2012, Toulouse (2012) hal-02191852
18. Matlab: Automated Driving Toolbox. Design, simulate, and test ADAS and autonomous driving systems. https://nl.mathworks.com/products/automated-driving.html#vcontrol
19. I. Broster and A. Burns: The Babbling Idiot in Event-triggered Real-time Systems. (2013)