EasyChair Preprint

№ 7301

Exploring System-Level Coordination of Vehicular Electronics: a Case Study for Traction Control

Md Rafiul Kabir and Sandip Ray

January 5, 2022

Exploring System-level Coordination of Vehicular Electronics: A Case Study for Traction Control

Md Rafiul Kabir and Sandip Ray
Department of ECE, University of Florida, Gainesville, FL 32611. USA
kabirm@ufl.edu, sandip@ece.ufl.edu

***Abstract*—In current practice, exploring the computation and software level of individual ECUs of an automotive system does not seem feasible enough for a system-level understanding of vehicular electronics. Exploring vehicular system-level use cases requires exercising the communication and coordination of the constituent ECUs. We are developing a prototype environment, VIVE, to enable early exploration of system-level coordination. VIVE enables extensible use case definition, as well as smooth and seamless addition of new, compute, sensor, or actuation functionality. This solution is flexible and configurable in such a way that enables the user to exercise inter-component and inter-system interactions. In this paper, we demonstrate the utility of such a prototyping environment in the exploration of a *traction control* use case.**

I. Introduction

Modern automotive systems are complex, distributed cyber-physical systems consisting of hundreds of Electronic Control Units (ECUs), a variety of sensors and actuators, several in-vehicle networks, and several hundred Megabytes of software code. A key challenge with automotive system validation is to explore and exercise system-level usage scenarios. Any system-level usage scenario (*e.g.*, the response of the vehicle to the successful pressing of a brake or turning of steering wheel) entails coordination of multiple components through a variety of networks; innocent optimizations to the process with an inaccurate mental picture can lead to subtle safety and security vulnerabilities. On the other hand, it is difficult to exercise such scenarios early in the system design life-cycle as all the hardware vehicular components have been manufactured at a later stage. These scenarios after all require the interaction of different ECUs, sensors, and actuators, and a significant component of the functionality is realized through software; correspondingly, exploring them requires a mature platform in which the ECUs, sensors, and actuators have been integrated and the relevant software can run. Obviously, errors and vulnerabilities found at that stage can be costly to fix, resulting in delays in production timeline as well as brittle patches, workaround, and point fixes.

In recent work, we proposed a solution for this problem by defining a prototyping environment, VIVE, for automotive system-level usage scenarios [1]. The key idea is to abstract details of ECU functionality and instead focus on the communication and coordination among various components. Correspondingly, VIVE includes a detailed simulator for in-vehicle networks (*e.g.*, CAN) providing a flexible interface to integrate ECUs, sensors, and actuators. VIVE enables extensible use case definition, as well as smooth and seamless addition of new, compute, sensor, or actuation functionality. We showed the efficacy of VIVE as an exploration platform for several automotive use cases *e.g.*, anti-lock braking system (ABS), right turn, return-to-center (RTC), traction control system (TCS), cruise control, and direct tire pressuring monitoring system (dTPMS) and indirect tire pressuring monitoring system (iTPMS). We also showed the efficacy of this platform for optimization by using existing optimization techniques like simulated annealing targeting two critical parameters *i.e.*, congestion and latency.

In this paper, we do a deep dive on a system-level use case study: *traction control*. The basic functionality of the system is fairly standard [2], [3]; the idea is to adjust the torque based on the calculated slip rate during vehicle acceleration. However, from our perspective, the goal is to comprehend and exercise the variety of interactions across different ECUs through CAN to realize the functionality and scope for potential optimization. We show how VIVE enables realistic, early exploration of this use case without requiring the constituent ECUs to be available in their fully elaborated (silicon or RTL) implementation *a priori*.

II. Related work

Nowadays, virtual prototyping has become a feasible method to explore the possibilities of development in vehicular electronics. There are a few closely depicted research works to mention. A graphical processing unit (GPU) virtualization of a digital cluster has been addressed in [4], which is mentioned as Virtualized Automotive Display (VADI) system. They can manage multiple in-vehicle execution software. In [5], a comprehensive discussion is provided on the exploits of automotive virtualization to integrate multiple ECUs into a few Domain Controller Units (DCUs). An integration of VP to the V-model of automotive software development in [6], has been introduced that enables verification and validation on SoC, ECU, and system level. The primary concept and control schemes were addressed in [2], [7], that described system functionality in details. However, that was not conceptualized using any kinds of exploration platform. Our solution is complementary to the existing practice but stands differently by taking a system-level approach while abstracting individual vehicular components.

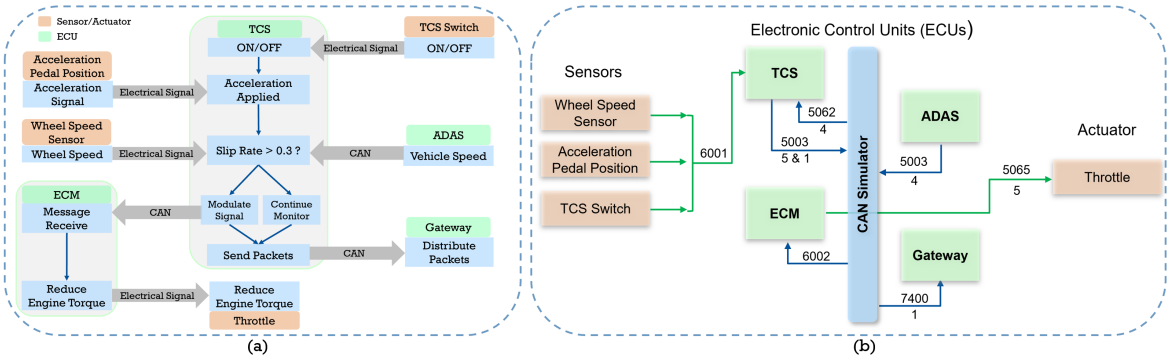

Figure 1. Traction Control system (a) platform's functionality flow diagram and (b) platform's primary system design

III. ViVE Architecture

Any automotive system compromises sensors, actuators, and ECUs with an in-vehicle communication network (*e.g.*, electrical signal, CAN, LIN, etc). A more detailed exposition of the ViVE architecture is available in a previous paper [1]. Here we provide a brief exposition to make the paper self-contained.

A. In-vehicle Communication

ViVE models the vehicular components in a simulated environment to exercise system-level use cases (*e.g.*, ABS, Right turn, etc) that involve coordination of computation, communication, sensor, and actuarial functions. Since ViVE focuses primarily on exploring the coordination of components, the central focus of the architecture is a simulator for in-vehicle communication. There are two types of communication shown here: (1) Electrical signal, and (2) In-vehicle communication network, *e.g.*, CAN. A reconfigurable CAN simulator is modeled to implement the CAN bus by using the available CAN library to build the CAN frame as byte array messages. The simulator uses Transmission Control Protocol (TCP) sockets to realize interaction among all the ECU processes where the socket programming is based on a standard client-server model.

B. Vehicular Electronic Components

Aside from the communication, the other automotive components involved in an use case are incorporated into ViVE as follows.

- *ECU*: Unlike traditional simulation platforms, a complete software model of the ECU is not required for ViVE. Instead, it bears the computational attributes of an ECU to simulate the functionality as relevant to the use cases by taking inputs from corresponding sensors or other ECU computational blocks.
- *Sensors*: ViVE provides an interface that can be used to connect a physical sensor or simply a software process to generate synthetic computational data representing the behavior of automotive subsystem sensors (*e.g.*, brake pedal sensor, angle sensor, etc).
- *Actuators*: Actuators are mostly functioning as the outputs to ECUs for actuarial activities, similarly represented by software processes.

Note that, the goal of this platform is to enable the user to get a realistic idea of the interaction of different components and subsystems through automotive use-cases. Therefore, all these vehicular components are modeled as continuously running computation blocks. For example, the angle sensor implemented provides simulated steering wheel angle data continuously giving inputs to the ECU associated with that use case (*e.g.*, right turn).

IV. Traction Control with ViVE

Fig. 1 provides an overview of the Traction Control System (TCS) and its implementation in ViVE. The components include TCS ECU, advanced driver-assistance systems (ADAS) ECU, engine control module (ECM) ECU, gateway, acceleration pedal position sensor, wheel speed sensor, TCS switch, and throttle. In ViVE we realize the ECUs and sensors as indigenous processes.[1] The coordination entails data communication through the CAN simulator that uses TCP sockets; each computation process is correspondingly hooked with port numbers (*e.g.*, 5003, 6001, etc) and arbitration ids (*e.g.*, 1, 4 and 5) *i.e.*, shown in Fig. 1 (b).

Table I
Byte Array Messages

Components	Data
Acceleration Pedal Position	[0] or [1]
Wheel Speed Sensor	simulated values from [0] to [60]
ADAS	simulated values from [0] to [60] (CAN)
TCS (final output)	[0] or [1] (CAN)
TCS (for gateway)	[0] or [1] (CAN)
ECM	[0] or [1]

[1] ViVE also enables the integration of actual ECUs as well as Raspberry PIs as ECU placeholders to develop a software-only feature. These more elaborated integrations can be used at advanced stages of system development when detailed implementations of individual hardware and software components are available. For this paper, we stick to the indigenous process model since we focus on exercising system-level scenarios *early* in the design. Nevertheless, the replacement of a component by a more elaborated one is a matter of simple component swap in ViVE: the interfaces themselves remain unchanged.

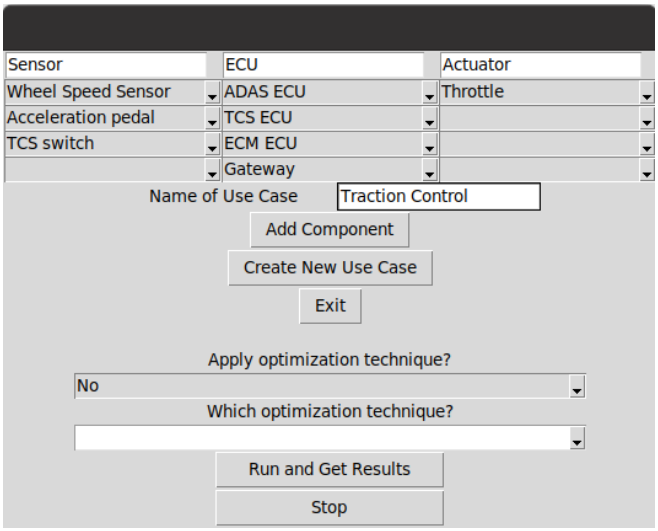

Figure 2. Traction Control simulation window

The user initiates actuarial actions by pressing the TCS switch (by default it is *ON* unless pressed to turn *OFF*) and acceleration pedal position, which is enabled through the VIVE GUI. Fig. 2 shows the initial simulation window from which a use case can be initialized. The window enables multiple GUIs denoting the TCS switch, acceleration pedal position and final simulation output. Table I shows all the byte array messages sent by the components (indicating both CAN and non-CAN messages). Here, the acceleration pedal position has only binary message options instead of intermittent values because the traction control function (active or not active) does not vary based on a long-range of values, rather on the *pressed or not pressed* status of the acceleration pedal. The TCS ECU only needs to take this binary input into consideration alongside TCS switch.

The VIVE action log shows the sequence of actions induced by the use case. The sequence involves (1) the acceleration pedal position sensor sending the pedal position signal to TCS ECU, (2) calculation of slip rate from simulated speed values, (3) communication of a byte array message containing CAN frame from TCS to ECM ECU if a high slip rate (greater than 0.3) is computed, and (4) the communication of torque reduction data to the throttle for applying traction that allows the car to accelerate in a more controlled manner. In addition, the TCS ECU sends a CAN message to the gateway ECU for distributing packets. The functionality of the use cases has been currently implemented till the gateway to focus on the system level operations; the functionality of the instrument cluster (*e.g.*, TCS active notifications in the head-up display) are not implemented. The wheel speed sensor is a continuously running process; consequently, the slip rate value continually changes resulting in engaging or disengaging of traction control which can be visualized in VIVE in real-time. The speed values are ranged between 0 to 60 mph for simulation purposes, not conceptually limited to this range for any use case.

Note that the TCS exploration involves interactions involving multiple ECUs (*i.e.*, four ECUs for this particular use case), along with sensors and actuator under implemented network simulator. The use of VIVE for this use case demonstrates that the interactions can be explored, customized, and optimized early in the platform design time frame. This is in stark contrast with related work [4], [6], which focus primarily on prototyping environment at ECU and SoC levels. We report that the current trend of analyzing through this ECU level exploration, does not provide a more feasible high-level scenario for the respective vehicular subsystems compared to the system-level exploration we demonstrated.

V. Conclusion

Exploring and exercising system-level use cases early in the design is a critical requirement for system-level validation of vehicular electronics. To achieve this, we are building VIVE with the explicit goal to realistically simulate in-vehicle communications. In this paper, we demonstrated the utility of VIVE in the exploration of an illustrative system-level automotive use case. The fact that we can explore coordination and communication among components naturally shows the value of such a system-level prototyping framework.

Obviously, VIVE is an early work in progress. In future work, we plan to harden VIVE with more realistic implementations, implement more use cases, and explore corner cases resulting from their interactions.

References

[1] M. R. Kabir, N. Mishra, and S. Ray, "Vive: Virtualization of vehicular electronics for system-level exploration," in *24th IEEE International Conference on Intelligent Transportation (ITSC 2021)*, 2021.

[2] J. H. Park and C. Y. Kim, "Wheel slip control in traction control system for vehicle stability," *Vehicle system dynamics*, vol. 31, no. 4, pp. 263–278, 1999.

[3] L. Austin and D. Morrey, "Recent advances in antilock braking systems and traction control systems," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 214, no. 6, pp. 625–638, 2000.

[4] C. Lee, S.-W. Kim, and C. Yoo, "Vadi: Gpu virtualization for an automotive platform," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 1, pp. 277–290, 2015.

[5] M. Strobl, M. Kucera, A. Foeldi, T. Waas, N. Balbierer, and C. Hilbert, "Towards automotive virtualization," in *2013 International Conference on Applied Electronics*. IEEE, 2013, pp. 1–6.

[6] M. Safar, M. A. El-Moursy, M. Abdelsalam, A. Bakr, K. Khalil, and A. Salem, "Virtual verification and validation of automotive system," *Journal of Circuits, Systems and Computers*, vol. 28, no. 04, p. 1950071, 2019.

[7] K. Chun and M. Sunwoo, "Wheel slip control with moving sliding surface for traction control system," *International journal of automotive technology*, vol. 5, no. 2, pp. 123–133, 2004.