ADAS Virtual Prototyping using Modelica and Unity Co-simulation via OpenMETA

Masahiro Yamaura[1] Nikos Arechiga[1] Shinichi Shiraishi[1]
Scott Eisele[2] Joseph Hite[2] Sandeep Neema[2] Jason Scott[2] Theodore Bapty[2]

[1]Toyota InfoTechnology Center, U.S.A., Inc., U.S.A., {myamaura, narechiga, sshiraishi}@us.toyota-itc.com
Institute for Software-Integrated Systems, Vanderbilt University, U.S.A., {eiseles, jhite7, sandeep, jscott, bapty}@isis.vanderbilt.edu

Abstract

Automotive control systems, such as modern Advanced Driver Assistance Systems (ADAS), are becoming more complex and prevalent in the automotive industry. Therefore, a highly-efficient design and evaluation methodology for automotive control system development is required. In this paper, we propose a closed-loop simulation framework that improves ADAS design and evaluation. The proposed simulation framework consists of four tools: Dymola, Simulink, OpenMETA and Unity 3D game engine. Dymola simulates vehicle dynamics models written in Modelica. Simulink is used for vehicle control software modeling. OpenMETA provides horizontal integration between design tools. OpenMETA also has the capability to improve design efficiency through the use of PET (Parametric Exploration Tool) and DSE (Design Space Exploration) tools. Unity provides the key functionality to enable interactive, or closed-loop ADAS simulation, which contains sensor models for ADAS, road environment models and provides visualization.

Keywords: ADAS, Efficient Design, Game Engine, Modelica, Simulink

1 Introduction

The number of installations of Advanced Driver Assistance Systems (ADAS) is rapidly growing in the automotive industry. In the case of Toyota cars, Toyota Safety Sense, which is a type of ADAS package, will be available in most passenger cars released by Toyota Motor Corporation in Japan, North America, and Europe by the end of 2017 (Toyota Motor Corporation, 2014). This emerging market of ADAS poses difficult system design problems. That is, we cannot use a traditional development methodology that considers only a target vehicle. We need to derive a new methodology which allows us to take its environment into account, e.g., road, other vehicles, pedestrians, etc. With the announcement that the majority of cars will contain an ADAS, it is apparent that the design space of future cars will be vast. Moreover, the complexity of these systems is also increasing along with their extended features, e.g., communication with other vehicles, cooperation with a navigation system, etc.

The above problems imply that a highly-efficient design and evaluation methodology for ADAS development is required. Van Waterschoot and van der Voort have recognized this same need for efficient design when looking at ADAS as a human factors problem (van Waterschoot *et al*, 2009). Simulation-based verification and validation can be a key technology in such a methodology as shown by Gruyer *et al.* (Gruyer *et al*, 2011). More precisely, closed-loop simulation including vehicle dynamics and road environments is essential.

Our work addresses the need for closed-loop simulation by using Simulink to model software components and Modelica to model physics components. Simulink is currently the state-of-the art tool for developing and analyzing automotive software models. Modelica is well suited to describe and simulate physics which includes vehicle dynamics. However, the task of describing complex conditions around the vehicle, such as traffic events, pedestrian activity and weather activity is complex and results in simulations not amenable to interactive simulation. Our work uses Unity to model complex environmental conditions. Unity is a video game development tool which is well suited to describe complex road situations. Our proposed framework consists of a co-simulation-based solution for ADAS development challenges by using OpenMETA to integrate Simulink, Modelica and Unity, and provide some features which aid in the design of complex systems.

Generally, game engines provide sophisticated virtual reality environments, and can be used to allow users to collaborate. These game engine advantages support valuable features in ADAS development such as gamified and crowd-sourced vehicle testing, and virtual dealership, which are described below.

Section 2 provides a background on existing design tools. Section 3 describes our tool framework. Section 4 describes our case study and Section 5 presents our conclusions and possible directions for future work.

2 Background

2.1 Existing Tools

Modelica is a multi-physics, multi-domain, acausal modeling language. Dymola, developed by Dassault Systèmes, is a powerful tool as an editor and simulator of Modelica models (Dassault Systèmes, 2015).

Simulink is a graphical modeling tool produced by MathWorks. It provides a graphical modeling editor, a customizable set of block libraries, and solvers for simulations (Mathworks, 2015). Designers can edit models by deploying blocks from the libraries and adding causal connections between blocks. Simulink is widely used in the automotive industry for vehicle control system software design, such as ADAS systems, engine control systems, transmission control systems, etc. This is the reason why we selected Simulink for control system software modeling.

The OpenMETA toolchain was developed by Vanderbilt University in conjunction with the Adaptive Vehicle Make program of DARPA (Sztipanovits *et al*, 2014; Sztipanovits *et al*, 2015). OpenMETA is a tool infrastructure with the goal of enabling development of cyber-physical systems. This is accomplished by providing horizontal integration between external software tools. A model in OpenMETA references component models which exist in external tools such as Dymola, Creo, or ADAMS. In order to interface with an external tool, an interpreter is created for OpenMETA which transforms the model into a format the external tool can use. Typically OpenMETA is setting parameters which are then used to provide inputs into a detailed model that exists in the external tool. Once the interface is in place, then any parametric changes made to the model in OpenMETA will also appear in the external tool. Additionally, if the internals of any tool specific model are changed, as long as the interface remains, the models will function as they would if the model had been generated entirely in the external tool.

In the case of Simulink integration, a Simulink model is wrapped as a C library which is referenced in OpenMETA. The representation of this library in OpenMETA includes the interfaces exposed in the original Simulink model.

Additionally, OpenMETA has other features for highly-efficient design, such as the Parametric Exploration Tool (PET) and Design Space Exploration (DSE). Automotive control software generally has many parameters that should be calibrated in the development phase. PET enables a designer to explore the interactions between parameters in an automated fashion and then displays the results in a way which allows the designer to make tradeoffs and select the parameter set which is most suited to the design criteria. Since OpenMETA has this feature, there is interest in applying OpenMETA to ADAS development. In this paper, we focus more on utilizing the PET in OpenMETA to calibrate system parameters with the simulation models.

A growing challenge with the design of complex systems is that each system may be designed using a variety of architectures and each architecture is comprised of components which also have variations as shown in Figure 1. This information is represented in OpenMETA through the use of design spaces. A design space is a set of design containers which contain a family of components which share a common interface. Alternate design architectures are similarly represented except rather than alternative components there are alternative architectures where the architecture is in turn made up of components. This results in an explosion in the number of configurations that could be considered. OpenMETA provides a tool, which is called DSE, to list all of the candidate design configurations and simulate those that meet the static

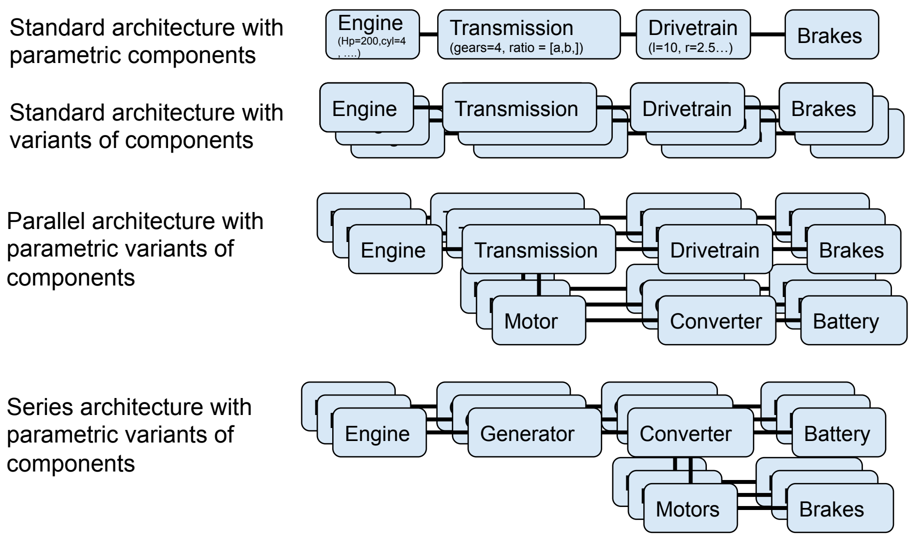

Standard architecture with parametric components
Engine (Hp=200,cyl=4 ,....)
Transmission (gears=4, ratio = [a,b,])
Drivetrain (l=10, r=2.5...)
Brakes
Standard architecture with variants of components
Engine
Transmission
Drivetrain
Brakes
Parallel architecture with parametric variants of components
Engine
Transmission
Drivetrain
Brakes
Motor
Converter
Battery
Series architecture with parametric variants of components
Engine
Generator
Converter
Battery
Motors
Brakes

Figure 1. An example of multiple architectures with component alternatives.

constraints which are specified by the designer. OpenMETA also provides a visualization tool that allows a designer to compare the designs which meet system requirements from DSE simulation results.

Although existing tools mentioned above allow us to build a vehicle model with cyber components and physical components, the proposed integration with Unity allows us to provide a far richer, virtual reality testing environment, including traffic, pedestrians, sensor models and effects of weather and the environment, such as those due to fog, heavy rain, and icy road conditions. These effects are important, because they can potentially compromise the correct functionality of ADAS systems.

2.2 Unity

Instead of commercial tools of ADAS simulation, e.g., PreScan, CarMaker, etc. we use Unity (Unity Technologies, 2015), which is a 3D game engine for video game development, for road environment modeling, and sensor modeling. The primary reason that we selected a game engine was to leverage the capability to involve users from all over the world in our ADAS evaluation tests. This concept is referred to as "gamified and crowd-sourced virtual testing". Generally, many situations would need to be considered in order to evaluate an ADAS implementation such as driver input, other vehicle behaviors, road geometry and so on. By using a game environment populated by human and virtual users, the ADAS software can be tested more extensively than with traditional static test scenarios.

In addition to providing a multi-user platform, Unity has a user friendly GUI editor, 3D physics engine, animation engine, 3D model import, and scripting in C# or JavaScript. These features help a simulation designer model cities which contain road models and other dynamic objects, such as vehicles, pedestrians, motorcycles, and bicycles. Although other 3D game engines are available, Unity was selected because of its large asset library, multiplatform support, and large community support.. Many assets are available through the Unity Asset Store which accelerates development and would not be available in other tools. For example, road editors, vehicle physics, and car traffic simulators are available as ready-to-use assets with Unity.

3 Methodology

3.1 Simulation Architecture

We integrated the four tools for the simulation framework: Dymola, Simulink, OpenMETA and Unity. The architecture is shown in Figure 2. OpenMETA integrates the Dymola and Simulink models. The Dymola model has some vehicle physical components, including the engine, transmission, driveshaft, and differential from Vehicle Dynamics Library. The Vehicle Dynamics Library is developed by Modelon (Modelon, 2014). Other components, such as wheels, are modeled in Unity. The interfaces between Unity and Dymola are wheel torques and wheel rotation speeds. The Dymola simulation sends wheel torques to the Unity simulation, and the Unity simulation sends wheel rotation speeds to Dymola. These interfaces are implemented by UDP socket communication. The Modelica Device Drivers (Bernhard et al, 2015) library provides UDP communication blocks used in Dymola.

The Unity model also has road environments, other vehicle models, and sensor models for ADAS systems. The sensor data in Unity is also sent to the Simulink controller via UDP. The list of UDP interfaces is shown in Table 1. In the Unity model, some assets from Unity Asset Store were used for modeling. For example, EasyRoad3D Pro is used for road building,

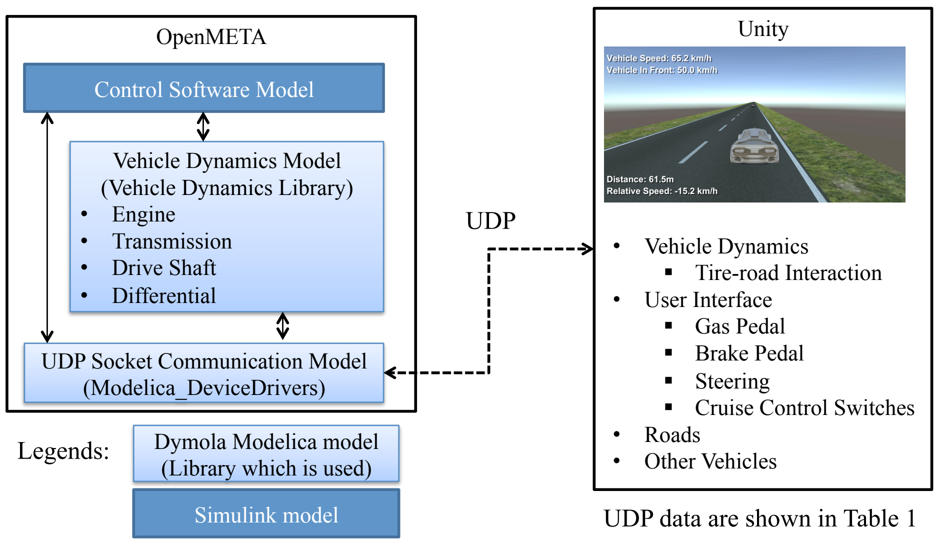


Figure 2. Simulation architecture

and the Urban Construction Pack for building city models.

Table 1. UDP Communication data list

Direction	*Data*
Dymola → Unity	Wheel torques
	Vehicle status (ADAS status, etc.)
Unity → Dymola	Wheel rotation speeds
	Sensor data (Millimeter wave radar, etc.)
	User operation (Pedals, steering, etc.)

4 Case Study

In this paper, we show simulation results with PET in OpenMETA to illustrate the advantages of this simulation framework. The PET is a highly-efficient methodology for calibrating control software parameters.

As the first case study for this simulation toolchain with PET, we decided to model the Adaptive Cruise Control (ACC) system. The ACC is one of the ADAS systems. This system helps mitigate driver fatigue by assisting accelerator operations. Toyota's ACC system has 2 modes: constant speed control mode and vehicle-to-vehicle distance control mode. Constant speed control mode is the same as a conventional cruise control system. While this mode is active the system works to maintain a target velocity. Vehicle-to-vehicle distance mode works with sensors, such as millimeter wave radar sensor that detects the presence of lead vehicles. Upon detecting a vehicle, the ACC adjusts the speed in order to maintain a safe following distance. The control flow is shown in Figure 3. The driver can choose the following distance: Long, Middle or Short. Actual distances are determined based on the velocity of the vehicle.

For the ACC case study in this paper, we defined following scenario as shown in Figure **4**. There is an ACC installed in host vehicle A, which has initial speed of v_0 and a lead vehicle B, which has constant speed v_{front}. Vehicle A is initialized with the ACC active and velocity set point v_{set}. The initial distance between two vehicles was set so that vehicle A accelerates to the speed v_{set}. After some time vehicle A senses vehicle B and transitions to vehicle following mode. Vehicle A decelerates to maintain the distance between two vehicles d_{set}. In this paper, we used values in Table 2.

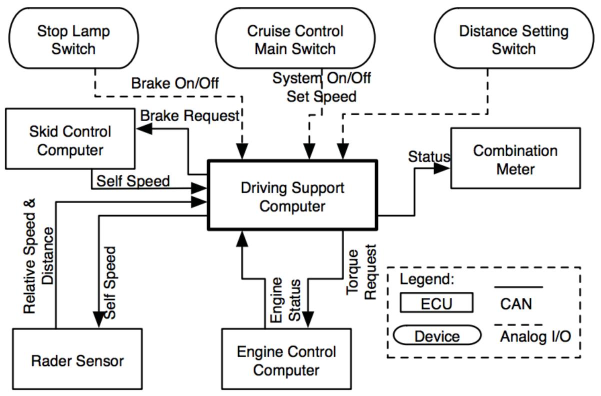
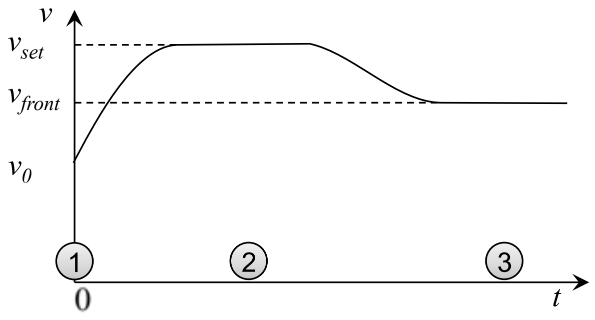


Figure 3. Adaptive cruise control diagram (Shiraishi *et al*, 2011)



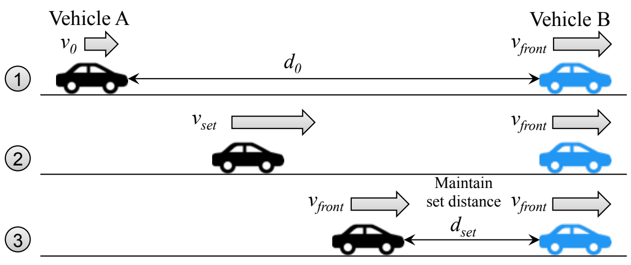
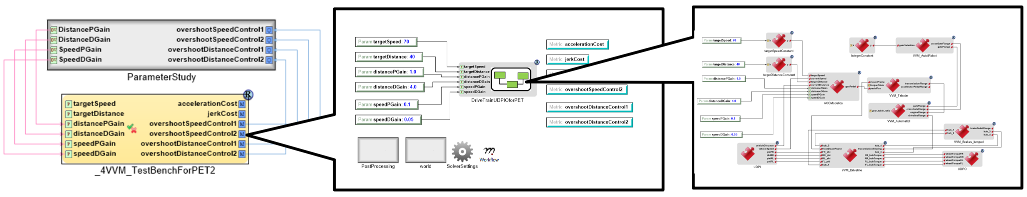


Figure 4. Case study of ACC system

Table 2. Parameters used in the simulation

Parameters	*Values*
v_0	0 km/h
v_{set}	70 km/h
v_{front}	50 km/h
d_0	150 m
d_{set}	40m



PET Test Bench

- Target parameter range
- Metrics

Test Bench

- Simulation settings
  - Simulation Time
  - Solver
  - Input parameters
  - Metrics

Model Integration

- Controller model
- Vehicle dynamics model

Figure 5. OpenMETA models for PET

4.1 Controller Model

The controller[1] developed for this experiment is bimodal. The first mode utilizes vehicle-to-vehicle distance as the desired value and the second mode uses a velocity set point as the desired value. Both controllers were implemented using PD controllers. The controller behavior is to maintain a safe following distance if there is a lead vehicle. Otherwise, it keeps vehicle speed to the set speed. The PD gains are calibrated by using PET which is discussed below. The ACC is always activated during the simulations so that the simulation does not need any driver inputs.

4.2 Unity Model

For the ACC case study, we added a long straight road, two vehicles and a millimeter wave radar sensor model. The sensor model obtains a distance between two vehicles and their relative speed. These signals are communicated to Dymola via UDP and are used as inputs to the controller. The sensor model was built by using "Ray" class in Unity scripting C# API which is often used in shooting games.

4.3 Parametric Exploration Tool and Test Bench

To run the simulation from OpenMETA with PET, designers have to set up some Dymola parameters, such as simulation time, solver, etc. in the OpenMETA Test Bench as shown in Figure 5. Additionally, metrics, which are used for evaluations of the models, need to be described in the Test Bench. The metrics in the ACC case study are velocity and gap distance overshoot. Settling time and rise time are also major metrics of this kind of system, but have not been included in this case study.

Next step toward parameter design is building a PET test bench. PD gains, which are speed control P, D gains and distance control P, D gains, are target parameters to be calibrated. In the PET test bench, designers need to assign parameters, their ranges, and testbench outputs or metrics. The PET test bench is also shown in Figure 5.

4.4 Simulation Results

We ran the PET of the ACC case study with parameters which are shown in Table 2. The result of the PET simulation results can be visualized as a "Constraint Plot" in the OpenMETA dashboard, which is shown in Figure 6. The horizontal axis and vertical axis of this plot are the PD gains mentioned above. The plots show boundaries, which represent which combinations of parameters that meet the overshoot requirements. Thresholds are adjustable in the dashboard. The threshold values in Figure 6 are *overshoot* < 0.

A designer can find PD gains by clicking on a point in a plot. We picked gains which are close to the boundary which meets both overshoot requirements. The graphs in Figure 7 are Dymola simulation results using gains selected by examining the constraint plot shown in Figure 6. The upper graph in Figure 7 represents distance between two vehicles; the lower graph in Figure 7 represents the velocity of vehicle A. The red line represents the target value and the blue line represents the current value. This plot shows that there is no overshoot in either graph, demonstrating that the PET was useful in selecting design parameters.

[1] The ACC controller model in this paper is *not* a real ACC model

5 Conclusions

This paper described the methodology of integrating OpenMETA and Unity. The foundation of an integrated simulation framework, which includes PET for ADAS evaluation, has been built. As a result, designers may calibrate control software parameters more efficiently. Next steps include developing high quality and multi-fidelity models which would allow for greater flexibility in the design process and developing additional case studies including the addition of driving scenarios such as: curves, intersections, etc. Additionally, it is also planned to consider driver-in-the-loop simulation which incorporate user input from Unity clients. These simulations would allow for some interesting applications. One such application is developing a

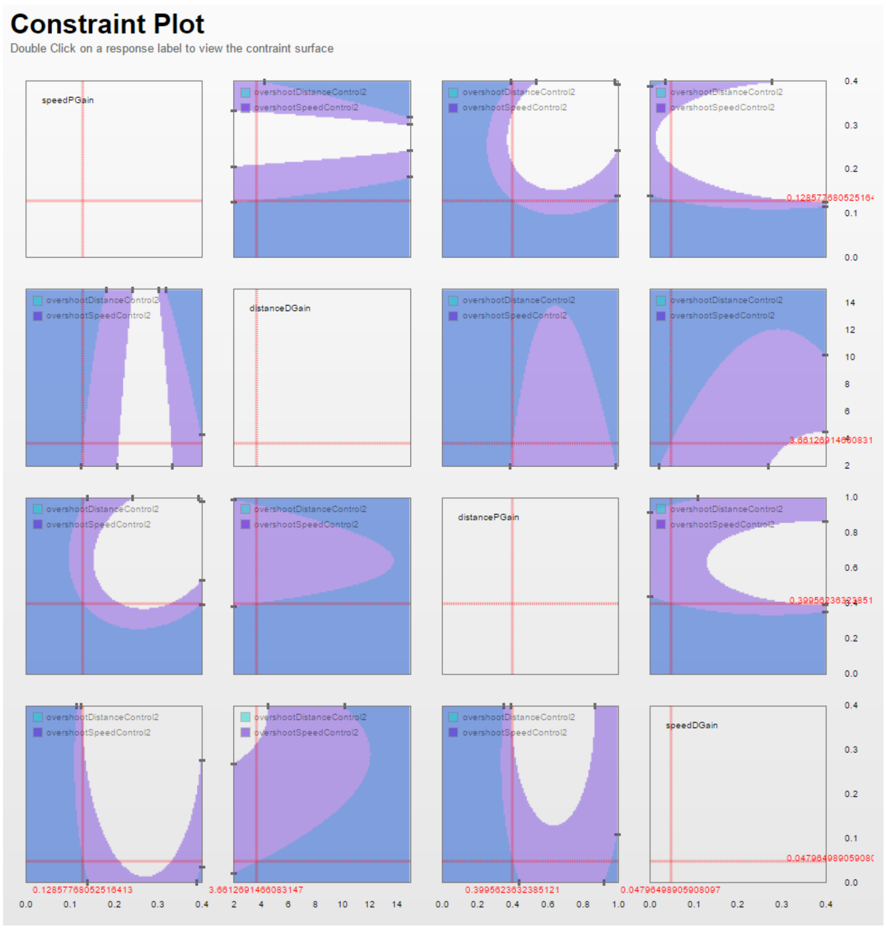


Figure 6. PET result: Constraint Plot

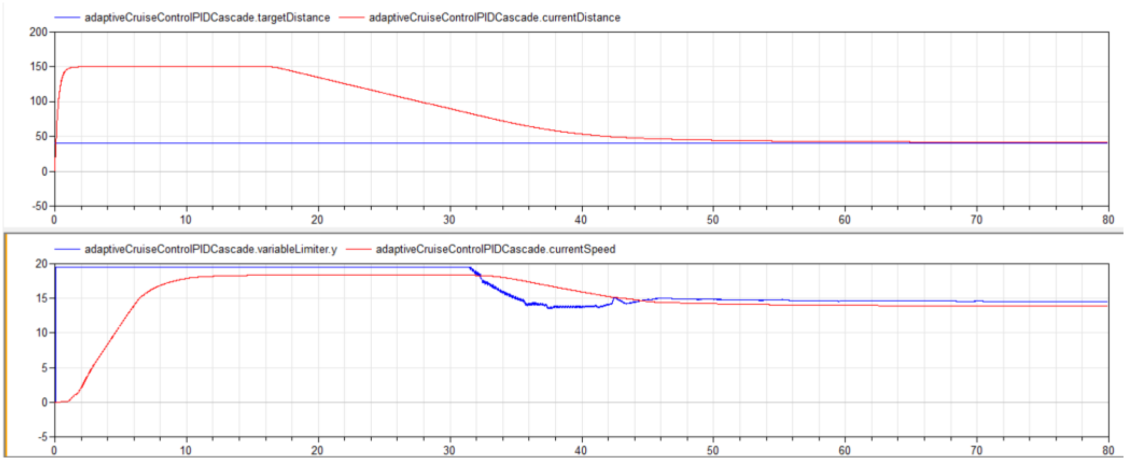


Figure 7. Waveform results of PET

virtual dealership concept in which customers participate in the design of their vehicles, and test-drive their creations. This test environment would also be used to crowd-source vehicle testing allowing for improvements to systems like ADAS and resulting in designs that have better performance and reliability.

References

Bernhard Thiele, Tobias Bellmann, tbeu, Modelica_DeviceDrivers, 2015 `URL: https://github.com/modelica/Modelica_DeviceDrivers`

Dassault Systèmes AB, Dymola 2015, 2015 `URL: http://www.3ds.com/products-services/catia/products/dymola`

D. Gruyer, S. Glaser, S. Pechbert, R. Gallen, and N. Hautiere, Distributed Simulation Architecture for the Design of Cooperative ADAS", *Presentation at First International Symposium on Future Active Safety Technology toward zerotraffic-accident*, September 2011.

MathWorks, Simulink 8.6, 2015 `URL: http://jp.mathworks.com/products/simulink/`

Modelon, Vehicle Dynamics Library 1.9, 2014 `URL: http://www.modelon.com/products/modelica-libraries/vehicle-dynamics-library/`

Shinichi Shiraishi and Mutsumi Abe. Automotive System Development Based on Collaborative Modeling Using Multiple ADLs. *Presentation at ESEC/FSE 2011 Industrial Track*, Sep. 2011.

Janos Sztipanovits, Ted Bapty, Sandeep Neema, Larry Howard, and Ethan Jackson. OpenMETA: A Model- and Component-Based Design Tool Chain for Cyber-Physical Systems. *From Programs to Systems. The Systems perspective in Computing*, pp. 235-248, 2014. doi: 10.1007/978-3-642-54848-2_16

Janos Sztipanovits, Ted Bapty, Sandeep Neema, Xenofon Koutsoukos and Jason Scott. The META Toolchain: Accomplishments and Open Challenges. *Vanderbilt University Institute for Software-Integrated Systems Technical Report*, 2015

Toyota Motor Corporation (2014). "2014 Toyota Safety Technology Media Tour", `URL: http://www.toyota-global.com/innovation/safety_technology/media-tour/` .

Unity Technologies, Unity 5.3.0, 2015 `URL: https://unity3d.com/unity`

Boris van Waterschoot and Mascha van der Voort. Implementing Human Factors within the Design Process of Advanced Driver Assistance Systems (ADAS) Engineering Psychology and Cognitive Ergonomics Vol. 5639, pp. 461-470, 2009. doi: 10.1007/978-3-642-02728-4_49