presented at the 6th IFAC Symposium Advances in Automotive Control, July 12-14, 2010, Munich, Germany

Automated simulation of scenarios to guide the development of a crosswind stabilization function

Klaus-Dieter Hilf*. Ingo Matheis**
Jakob Mauss. Jochen Rauh***

**Daimler AG, D-71059 Sindelfingen, Germany (e-mail: {klaus-dieter.hilf, jochen.rauh}@daimler.com).*
***QTronic GmbH, AltMoabit 91d, D-10559 Berlin, Germany (e-mail: {ingo.matheis, jakob.mauss}@qtronic.com)*

Abstract: Mercedes-Benz has recently added a crosswind stabilization function to the Active Body Control (ABC) suspension for the 2009 S-Class. For this purpose the ABC uses the yaw rate, lateral acceleration, steering angle and velocity sensors of the Electronic Stability Program ESP to vary the wheel load distribution via the ABC spring struts, depending on the direction and intensity of the crosswind. This function has to distinguish between vehicle reactions caused by crosswind, by driver interaction, and by road unevenness. The effects of the crosswinds can be compensated in this way, or reduced to a minimum in the case of strong gusts. For developing this function Mercedes Benz used the test case generator TestWeaver to generate thousands of different driving and crosswind scenarios. The scenarios have been executed using a co-simulation of: (i) a dynamic vehicle model (based on the in-house tool CASCaDE), (ii) a road and crosswind model implemented in C and (iii) a MathWorks/Simulink model of the crosswind stabilization function. This simulation-based approach helped considerably to validate and iteratively improve the safeguarding algorithms of the stabilization function through all design phases.

Keywords: Rapid Control Prototyping; Systems for Vehicle Dynamics Control; Lanekeeping.

1. INTRODUCTION

Nowadays an increasing number of automotive functions is realized using software, resulting in a steadily growing complexity of automotive controllers.

For validation and test of complex controllers, traditional methods based on hand-written test scripts do not scale well. Testing the controller in real life by trying to expose the system under test to all relevant situations is very time consuming or even not feasible without excessive effort. New methods and tools supporting a much higher degree of automation are required here, to meet shorter time-to-market and high quality demands. In this paper, we present one such method based on fully automated generation, execution and validation of useful test cases. We also report how the corresponding tool, TestWeaver, has been used to validate and iteratively improve the safeguarding algorithms of the crosswind stabilization function of the 2009 S-Class. The paper is structured as follows: in the next section, we describe our simulation-based validation and test environment. Section 3 presents the executable model of the system under test, consisting of the stabilization and safeguard functions, road, wind and vehicle models. Section 4 describes the automated test and validation process. We conclude with a brief assessment of the presented approach.

2. VALIDATION AND TEST ENVIRONMENT

The entire validation and test environment runs on a standard PC, without any real vehicle hardware in the loop. Section 3 describes how a realistic system simulation model was built. Such a pure 'virtual' setup can be easily duplicated, e.g. to parallelize and hence speed-up development within a team. Another advantage is that, without real vehicle hardware (such as ECUs) in the loop, there is no real-time requirement for running the models: Simulation can be suspended at a specified event to inspect all variables of the simulated vehicle. Simulation can also be arbitrarily fast, resulting in increased test throughput. In our case, the simulation runs about 10 times faster than real time. Thus, in just 3 days of simulation, about one month of street driving, with a huge number of differing situations, can be simulated and analyzed on one PC.

For automated validation (see Fig. 1), the simulation of the system under test is driven by a sequence of inputs generated by the test case generator TestWeaver. The inputs control the road and wind properties, acceleration and brake pedals, steering, and may also be used to activate dynamically (simulated) component faults, e.g. of sensors and actuators. Selected outputs of the simulation (such as car speed, gear rates, key variables of the controller) are observed by TestWeaver and stored together with the inputs in a data base, labeled 'state DB' in Fig. 1.

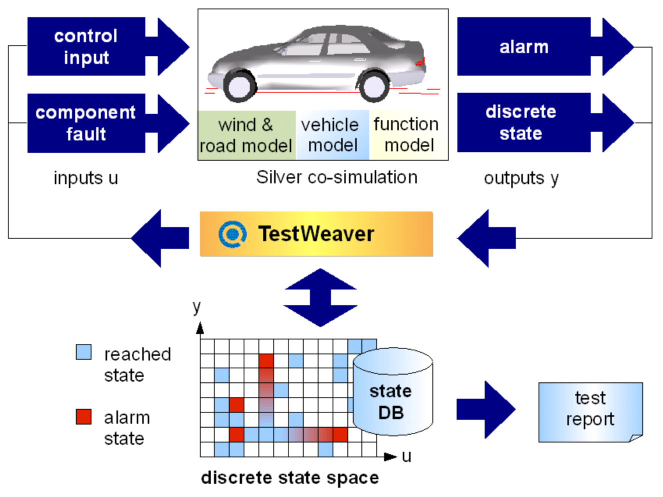


Fig. 1. Setup for simulation-based validation and test.

The test case generation, execution and validation does not require any user interaction and is interleaved: a new test case depends on the outcome of all previously generated tests. TestWeaver generates tests not randomly (this does not help much), but in a reactive, informed way, trying to worsen actively scenarios that are already sub-optimal until system behavior is really bad, i.e. a bug or flaw has been found. Here, a 'bad' scenario is by definition a scenario where an output variable reaches a value classified as 'bad' in the test specification, see below. TestWeaver also attempts to maximize the coverage of the system state space, i.e. to reach every reachable state in at least one of the generated scenarios. As indicated in Fig 1, state space is here the space spanned by all inputs and outputs that connect the system under test to TestWeaver. Maximizing state coverage is non-trivial, because TestWeaver can only control the inputs directly, not the outputs. For example, TestWeaver cannot set the speed of the car (an output of the model), but it can learn that pushing the acceleration pedal (an input of the model) for a while leads to high vehicle speed. To guide scenario generation, TestWeaver stores each state reached during simulation into a state data base, together with the sequence of inputs that leads into this state. Thereby TestWeaver successively learns how to control the system under test. TestWeaver uses this knowledge to drive the system into states not reached before (to maximize state coverage) and to worsen scenarios locally by automated variation of those already generated scenarios that got worst scores.

Technically, an input or output is a model fragment implemented in C, Simulink, Modelica or Python as part of a model or sub-model and that connects to TestWeaver using TCP/IP to either retrieve input values during simulation or report output values.

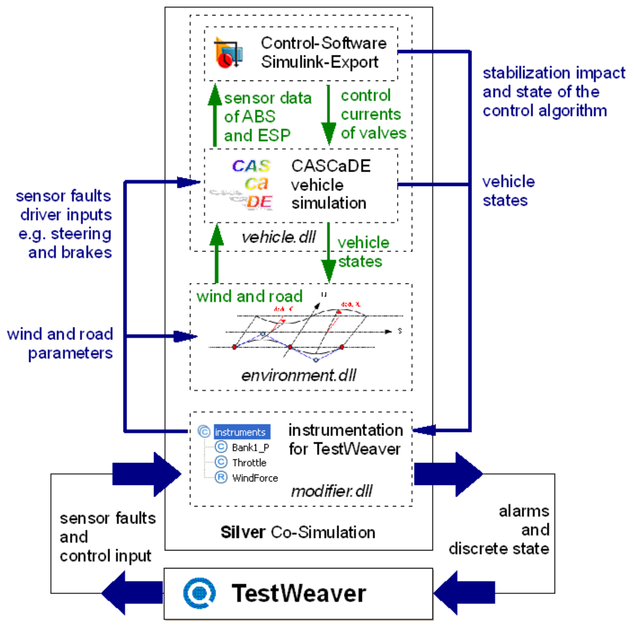
For testing a system with TestWeaver no test scripts need to be specified. Instead, a test or development engineer provides a very compact test specification with the following information:

- names of input variables, allowed set of discrete values, and classification of these input values on a good-bad scale (to support fault injection)
- names of output variables and classification of output values on a good-bad scale (to support automated validation of generated scenarios during execution)
- templates for reporting reached coverage in the state space and other test results
- general specification data, such as maximal duration of generated scenarios, upper-bounds for injected faults per scenario, command used to start the simulation, etc.

TestWeaver reports the test results using HTML. Report templates use SQL (a standard for data bases) to define the content of the tables. All scenarios generated by TestWeaver can be replayed by the test engineer on demand for detailed investigation and debugging. More details can be found in (Brückmann et al. 2009), (Gäfvert et al. 2008), (Junghanns et al. 2008), (Rink et al. 2009).

3. SYSTEM MODEL

This section describes the executable system model used for automated validation by TestWeaver. Simulation has been implemented here as a co-simulation of several sub-models using the co-simulation tool Silver (Rink et al. 2009). In Silver, a sub-model contains either a numerical solver, or uses a numerical solver provided by Silver. In both cases, a Silver sub-model is a DLL (dynamic link library) that implements a certain API, such as the standard FMI (ITEA 2 2010) or the proprietary Silver module API. For the application presented here, the modules and their mutual connections as well as the embedding in the Silver Co-Simulation are shown in Figure 2.



Control-Software Simulink-Export
sensor data of ABS and ESP
control currents of valves
stabilization impact and state of the control algorithm
CASCaDE vehicle simulation
vehicle.dll
vehicle states
vehicle states
sensor faults driver inputs e.g. steering and brakes
wind and road
environment.dll
wind and road parameters
instrumentation for TestWeaver
Instruments
Bank1_P
Throttle
WindForce
modifier.dll
Silver Co-Simulation
sensor faults and control input
alarms and discrete state
TestWeaver

Fig. 2. Integration of CASCaDE-simulation into TestWeaver.

The CASCaDE vehicle model has been exported as DLL that implements the Silver API and uses a CASCaDE solver for numerical integration (shown as vehicle dll). A second sub-model was created to model crosswind and the road, called the environment dll in Figure 2. The wind stabilization function has been developed using MATLAB/Simulink and was included into the vehicle dll also comprising the CASCaDE vehicle model. A third sub-model called modifier dll contains all instruments (inputs u and outputs y in Fig. 1) used by TestWeaver to control simulated crosswind, road and vehicle and to observe and assess model behavior.

3.1 *Crosswind Stabilization Function*

The stabilization function (Keppler et al. 2010) is based on a disturbance observer which measures the difference between predicted and actual vehicle behavior. From the calculated deviation a disturbing moment around the vertical axis of the inertia system is derived.

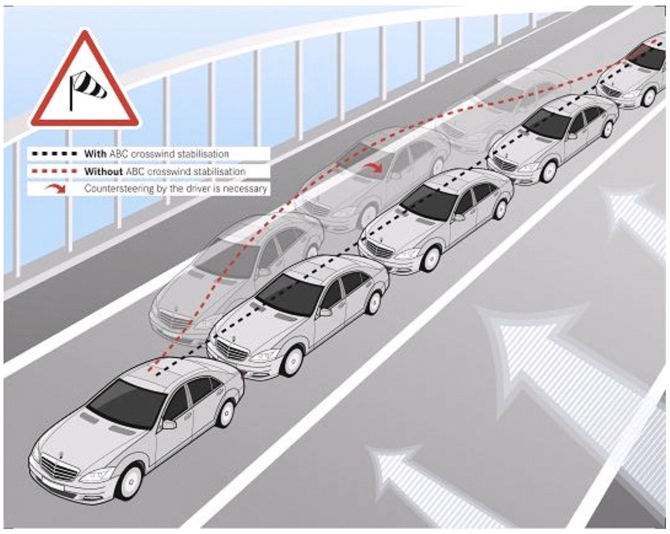

With ABC crosswind stabilisation
Without ABC crosswind stabilisation
Countersteering by the driver is necessary

Fig. 3. Driving with and without stabilization function.

If the safeguard functions determine that this moment is caused by crosswind, a path correction is induced by performing a diagonal wheel load actuation (warp mode) called Active Body Control crossover with the hydraulic struts of the ABC suspension. Through the elastokinematic design of the axle, changes in the toe angles are generated, resulting in an asymmetric side force. This leads to a steering reaction of the car compensating the lateral offset induced by the crosswind. The intervention of the system is scaled to compensate the disturbing moment up to a designed degree.

For simulation purposes the controller developed in Simulink was exported using the RealTime Workshop. In the CASCaDE (Rauh et al. 2008) simulation environment, used here for vehicle dynamic simulation, the subsystem-interface was used to couple efficiently the inputs and the outputs of the control system with the vehicle model.

3.2 *Road and Wind Model*

The system model also includes configurable road and wind models. During simulation, TestWeaver controls key control signals of this model in order to test the system under a great range of differing road and wind conditions.

The bank angle of the road is modeled as superposition of two Bezier splines - capturing large and small scale variations of the bank angle. One such spline is shown, together with its control points, in Fig 4. Control points are dynamically generated by TestWeaver in front of the vehicle on demand during simulation. Similarly, the local road inclination is modeled by two Bezier splines for large and small scale variations. Again, control points are dynamically generated on demand by TestWeaver. The road generated by TestWeaver is constrained in a way that the acceleration of the driver does not exceed a certain threshold during driving.

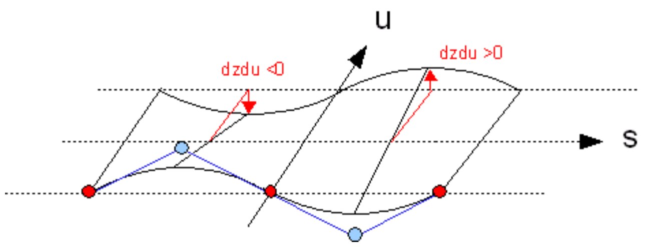


Fig. 4. Bank angle of road modeled using Bezier splines.

Speed and direction of the wind is modeled and controlled in a similar manner. In addition, the wind model provides a couple of parameters for varying statistical properties of the wind, such as shape of and delay between wind gusts.

The road and wind models have been implemented in C and compiled as a DLL that directly runs in Silver. The dynamic control of the road and wind model during simulation (as opposed to using predefined static road and wind profiles) gives TestWeaver better chances to increase the state coverage of the total system, including road, wind, vehicle and controller states: this way TestWeaver can better synchronize differing road and wind events with differing states occurring in the controller and vehicle model.

3.3 *Vehicle model*

The CASCaDE (Rauh et al. 2008) simulation model describes the vehicle dynamics of a car. All important aspects like steering, propulsion, braking system and suspension are modeled in appropriate depth and detail for vehicle dynamics analysis. A model of the hydraulic suspension system ABC with a simple representation of the hydraulic lines, valves, cylinders and the suspension struts is included. The detailing is adapted to the problems examined here. The original control software of this active suspension system is also embedded as exported c-code and linked with the model. The module receives sensor-information created by the simulation and outputs the control currents for the valves, thereby performing the desired wheel-load changes.

The vehicle dynamics behavior and especially the steering effect based on wheel load variation – the elastokinematic effect used here for crosswind stabilization – were validated from measurements. The aerodynamic characteristics were parameterized from extensive wind tunnel measurements and validated from bypass measurements at a crosswind test facility.

The ESP-algorithm is not included in the simulation model. Since crosswind impact is generally not strong enough to cause an ESP-intervention in the S-Class, a car featuring a strong directional stability, the influence of the ESP-system can be neglected in the study reported here. Only the ESP sensors used by the stabilization function are represented in the model. For other investigations the ESP could also be included.

This simulation model (including the stabilization function from 3.1) was converted into a dynamic link library (DLL) with an open interface implementing the communication with Silver. Driver inputs, current tire patches and wind is fed to the vehicle simulation. Vehicle and controller states are reported back to TestWeaver for scenario assessment and state coverage measurements (see Figure 2).

4. TEST OF THE STABILIZATION FUNCTION

It is not possible to test all possible driving situations in real life. Disregarding the great effort in time and expenses which make extended test drives undesirable, even on test tracks, only a limited number of road profiles is available, so all possible road excitations can never be covered. Furthermore, the possibilities to create different wind profiles for real life testing are very limited. In virtual test drives, however, every combination of road and wind excitation can be generated. Therefore, TestWeaver was chosen as a promising approach to cover the necessary test range with acceptable effort.

The main focus of the investigations was safeguarding against control impacts due to an erroneous crosswind detection. Since the observer bases the detection only on ESP-sensor data, and no direct wind-sensor is implemented, an asymmetric unevenness of the road, leading to lateral acceleration and yaw rate, could be interpreted as crosswind. To avoid the crosswind stabilization to respond to this excitation, other controller subsystems are designed to differentiate between vehicle reactions due to crosswind and reactions due to driver- and street-interaction or sensor faults.

The first focus was on trying to provoke the crosswind stabilization function to perform steering impacts due to driver and street interaction, thus detecting holes in the safeguarding mechanisms. Since basic features of safeguarding rules implemented were specified, and already sufficiently tested, the range of feasible driving- and environment situations in which the function had to be tested in this approach could be restricted to situations not already reliably and adequately covered. Thus scenarios not respecting these well-known limits set by the safeguarding mechanisms, for instance, on steering wheel angle or velocity, were not investigated and excluded in advance from the situations possibly chosen by TestWeaver. By taking into account this beforehand knowledge the design range TestWeaver had to cover to guarantee the reliability of the system was reduced to the regions not verified so far, allowing TestWeaver to work more efficiently.

Finding categories of suited street excitations was an iterative approach. Too high excitations were easily detected by the safeguard mechanisms implemented so far. Too small excitation did not lead to a relevant wind force estimation and, thus, to no reaction of the system. After choosing a promising range from evaluating the TestWeaver results, TestWeaver found several categories of impacts which the controller was not safeguarded against.

The mechanism included at the examined design stage only used the difference in spring travel between left and right wheel with the standard sensors being available in the ABC suspension system. The failure scenarios found with TestWeaver showed that a certain type of street unevenness did not lead to a high enough difference in spring travel. Reducing the critical limit of difference spring travel allowed was not an appropriate solution - this would reduce the percentage of time the system is active. The relevant scenarios were nonetheless marked by a high individual spring travel. From this observation a new safeguarding module was added, combining individual and difference spring travel.

After this element was included in the controller, a re-run of the critical scenarios showed that the unevenness was now detected. New runs with TestWeaver proved that the protection against false crosswind recognition was complete. The proportion of time the system was active was not reduced. Thus, this new criterion was implemented and approved in the test runs.

In a second approach TestWeaver was additionally used to create sensor faults of different classes: sudden offsets or linear drifts on the different sensor signals used by the observer and the safeguarding mechanism. Here TestWeaver was used during the design phase of the detection module inside the controller. Current versions were immediately exported, linked with the vehicle system simulation and tested with TestWeaver. The effectiveness of new measures or chosen limits was investigated before a first version was tested in the vehicle.

5. CONCLUSION

We reported how a closed-loop vehicle simulation in combination with the test case generator TestWeaver has been used to support and guide the development of a crosswind stabilization function. The validation reported has been conducted by a single engineer (a novice TestWeaver user at that time) within about three weeks. In that time, about 100.000 different driving scenarios, each 45 sec. long, have been generated, executed and validated. The setup has been changed and extended during the investigation to explore also the effect of sensor faults. The coverage achieved this way would have been hard, if not impossible, to achieve with comparable effort using a less automated approach, e. g. based on hand-written test scripts, driving a real car on the road, or using the Daimler crosswind test facility.

To summarize, the presented approach seems extremely well suited for the validation of complex automotive controllers during all stages of development. The main benefit is in the high test coverage that can be achieved with low work effort for engineers, based on a compact high-level specification of the validation task.

REFERENCES

Brückmann, H. et al. (2009). Model-based development of a dual-clutch transmission using rapid prototyping and SiL. In *International VDI Congress Transmissions in Vehicles 2009,* Friedrichshafen, Germany.

Gäfvert, M. et al. (2008). Simulation-based automated verification of safety-critical chassis-control systems. In *Proceedings of AVEC '08,* Kobe, Japan.

ITEA 2 (2010). Functional mock-up interface for model exchange 1.0, Specification, released 26.01.2010. http://www.functional-mockup-interface.org

Junghanns, A., Mauss, J. and Tatar, M. (2008). TestWeaver - a tool for simulation-based test of mechatronic designs. In *6th International Modelica Conference,* pp. 341 – 348, Bielefeld, Germany.

Keppler, D., Rau, M., Ammon, D. et. al. (2010). Realisierung einer Seitenwind-Assistenzfunktion für Pkw. In *AAET – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel,* Braunschweig, Germany (in German).

Rauh, J. and Mössner-Beigel, M. (2008). Tyre simulation challenges. *Vehicle System Dynamics,* volume 46, supplement 1, pp. 49-62.

Rink, A., Chrisofakis, E., Tatar, M. (2009). Automatisierter Test für Softwaremodule. *ATZelektronik,* volume 6, pp. 36-40. (in German).
English: http://www.qtronic.de/doc/ATZe_2009_en.pdf