Integration of a Damper Control Algorithm into an ADAMS/Car Full Vehicle Model

16th European Mechanical Dynamics User Conference, Berchtesgaden

Kai Sorge, Continental AG, Hannover
Harald Wilhelm, AUDI AG, Ingolstadt

Introduction

Today's high level cars are equipped with active and semi-active suspension components. The components are activated by electronic control units (ECUs). On these units algorithms are installed which determine driving states and road conditions from sensor input and compute appropriate responses for the active or semi-active components.
Replacing passive by electronically controlled active components offers changes of the driving behavior without changing the mechanical components simply by adjusting parameters of the control algorithm. Therefore, the task of the test driving engineers has become much more complex, and the need to give insight by means of simulation increases.
In this paper we present the usage of ADAMS/Car to simulate the full vehicle behavior of a car which is equipped with controllable dampers. The focus is the integration of a control algorithm which is available in Matlab/Simulink into the conventional ADAMS/Car vehicle model.

Components of the Semi-Active Suspension System

Continental Teves has developed several air spring-damper units for the AUDI AG. Theses systems consist of struts with air springs and integrated dampers. The air springs require a compressor, valves and air supply lines. The spring travel and other data required to detect the driving conditions are measured by sensors and processed by the ECU. The ECU controls the system by switching on the compressor and valves. Our new systems also contain controllable dampers made by Mannesmann-Sachs. They act in a much higher frequency range then the level control and require additional acceleration, steering and brake pressure sensors.

Development Environment for the Control Algorithm

Matlab/Simulink[1] offers a flexible, transparent environment for developing controllers. The graphical user interface represents algorithms by blocks connected by arrows. The arrows represent vectors with data. The blocks manipulate the data and are available from an extendable library. When used with a rapid prototyping system like DSpace, special blocks are available to realize read and send signals like PWM and CAN. When simulations are started Matlab/Simulink is automatically identifying the structure of the system, i.e. the sequence in which the blocks and operations have to be processed.
The Matlab/Simulink control algorithm serves several purposes:

- the algorithm is tested for its basic functionality with Simulink single and double track models,
- it is tested in a car by transferring the code to a DSpace Autobox, which replaces the ECU in the vehicle,
- the code should be converted to source code suitable to be flashed into the ECU memory,
- the code should be integrated into an ADAMS/Car full vehicle model for analyzing the influence of individual suspension components.

The first two items are state of the art. Code transfer to ECUs is not practice, but very likely to become a standard the more converters are improved and processor performance increases. The last point is discussed in detail below.

Linking Control Algorithms to ADAMS/Car

There are two ways to connect Simulink based control algorithms to an ADAMS/Car model. The first method is a co-simulation of the programs. ADAMS/Control provides a Simulink block ("plant model") which can be inserted into any Simulink model. When running the Simulink model containing the controller, adams is started up automatically. There is a clear procedure available to link the Simulink input/output to Adams variables.

---

[1] Matlab, Simulink and Real-Time Workshop are registered trade marks of The MathWorks,Inc.

This method is convenient when frequent changes of the controller occur. However, the user needs to have access to all software packages, Matlab/Simulink, ADAMS/Car and ADAMS/Control on the same machine. Another disadvantage is losing the usual ADAMS/Car simulation control.
The second method is creating a C-Code from the Simulink model using the Real-Time Workshop module. The resulting code can be linked to a ADAMS subroutine. This requires some subroutine code and more effort to update controller models. On the other hand, the resulting model is an ordinary ADAMS/Car model compatible to any conventional vehicle model.
If the controller is developed by a supplier, the subroutines can also be made available as object code avoiding the full disclosure of code.

Integration of the Damper Control

The ADAMS/Car model required to test the damper algorithm must contain the ADAMS/Car subsystems: front and rear suspensions, chassis, steering, power train, front and rear wheels and brakes. Subroutines are parts of subsystems. The control algorithm is called through a REQSUB subroutine which is residing in the chassis subsystem. REQSUB subroutines are designed to compute output variables at output steps. By calling the control algorithm here, the output step time increment will determine the controller increment. In this case the controller is called every 10 milliseconds. The computed damper adjustment electric currents are stored in a common block and also written to the output of the user subroutine.
Most sensor input values are not directly available. They can be computed using the system function SYSFNC when the appropriate marker identification numbers are passed to the REQSUB subroutine. If these markers are not part of the chassis subsystem, they can be transferred from other subsystems using ADAMS/Car communicators (see fig. 1).

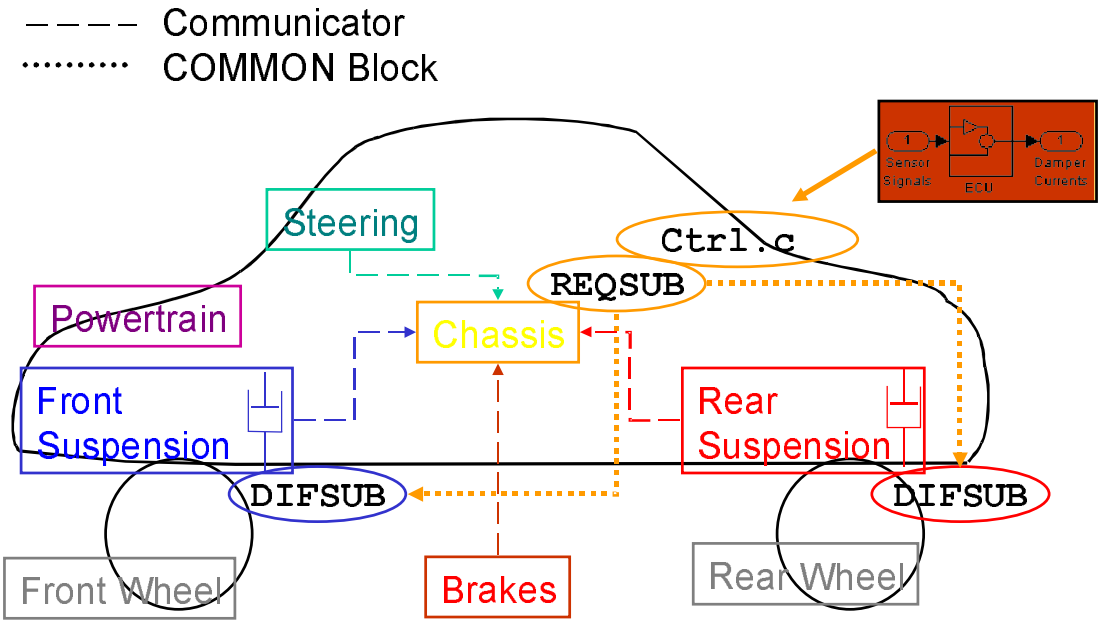

Figure 1: Structure of the ADAMS/Car model: Communicators transfer data required for sensor inputs to the controller. The REQSUB call in the chassis subsystem is invoking the call of the controller code. Calls to a DIFSUB subroutine are defined for each strut. The results of the REQSUB subroutine are accessed through a COMMON block.

The ADAMS user subroutines have to be written using the programming language FORTRAN, the Matlab Real-Time Workshop creates C-Code. With most compilers, a procedure to combine these languages is available (the authors have worked on a NT-PC using Visual Fortran/C).
A conventional damper computes the damper force evaluating a spline with the relative velocity of the corresponding strut. Now the damper force is computed using a DIFSUB subroutine (see the next section for more details). This subroutine takes into account the dynamic behavior and the electric currents provided by the REQSUB routine via a COMMON block. A reference to the integrated value of the DIFSUB subroutine replaces the call of the spline evaluation in the single component force definition of the damper.

Modeling the Controllable Damper

The Mannesmann-Sachs CDC damper contains a hydraulic valve which changes its diameter depending on the applied current. The current can continuously be adjusted. Fig. 2 shows the static force-velocity curves for a set of currents. This so called static behavior can be represented by a three dimensional field.

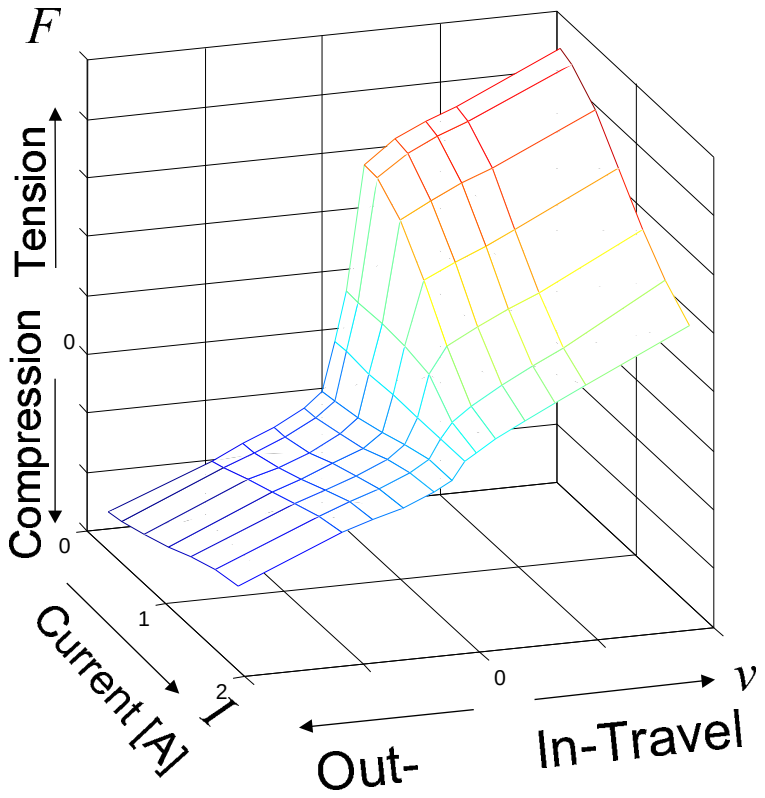

Figure 2: Static damper field

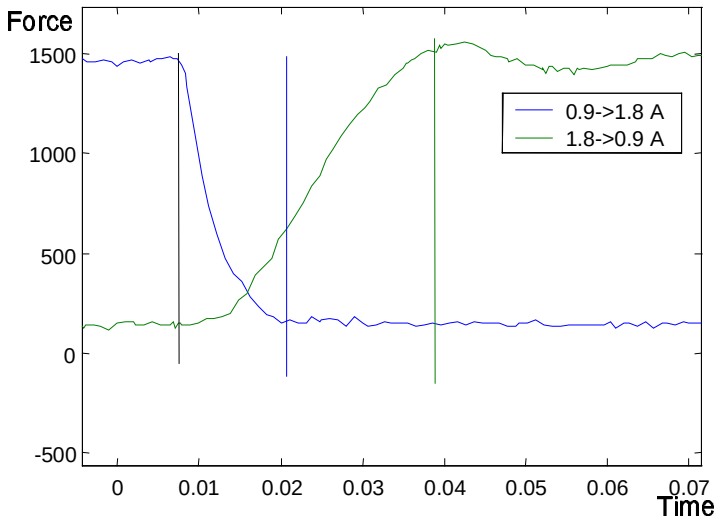

Figure 3: Example for the dynamic behavior of the force. The time required to reach the static force level depends on force level and velocity .

When changing the current I at constant velocity v, the force value F_{dyn} does not change instantaneously (see fig. 3). It takes some time until the static force value F_{stat} is reached. The transition

time depends on the force level and on the direction of change. Switching from a high force level to a small force level is faster than in the reverse direction, because some piston travel is required to build up the required pressures. The force level can be represented by a first order differential equation:

$$\dot{F}_{dyn} = F_{stat} - f(F_{dyn}, v)$$

The differential equation is added to the subsystem by implementing an ADAMS DIFSUB user subroutine.

Converting the Control Algorithm into a C-Code

The Simulink model can be converted to a C-Code by using the Matlab Real-Time Workshop. The code can be used to run the algorithm as a stand alone application. The major purpose of this tool is to generate code which can be loaded and run in a real system for prototyping and hardware-in-the-loop simulators.
The generated C-Code contains a header file, in which the user parameters are defined, initialization routines, routines which are called at full integration steps and subroutines which are also called at intermediate steps requested by the time step control of a continuous system integrator.
The damper control is defined as a fixed time step discrete system, so there is no need to pay attention to subroutines required when continuous states are present.
In order to conveniently call the subroutine from the ADAMS REQSUB subroutine, two small subroutines have still to be defined.
The subroutine calls to be performed at the initial time step can be collected in a subroutine ctrl_init:

```
void ctrl_init(){
    ECU(); /* name coincides with simulink model, set up
            data structure rtS */
    MdlInitializeSizes();
    MdlInitializeSampleTimes();
    MdlStart();
}
```

At every time step the subroutine ctrl_step is called:

```
void cdc_step(real_T*u,real_T*y){
    int i;
    real_T *uu,*yy;
    uu=ssGetU(rtS); /* get input vector pointer*/
    for(i=0;i<10;i++)
      uu[i]=u[i]; /* store input vector into simulink structure*/

    (*rtS->mdlInfo->t)+=0.01; /* time increment*/

    MdlOutputs(i+1);    /* create output*/
    MdlUpdate(i+1);     /* update discrete states */

    yy=ssGetY(rtS); /* get output vector pointer */
    for(i=0;i<4;i++)
      y[i]=yy[i];  /* store results into output value pointer y*/
}
```

See the user's manual "Simulink-Writing S-functions" for more background information about what the individual routines do.
ECU() and the Mdlxxx- routines are automatically created. No manual changes are required. If the Simulink model changes, you have to run the Real Time Workshop, to compile the subroutines again and to link the results to create a new acarsolver.exe. Matlab and the compilers (C, Fortran) can reside on separate machines.

Simulation results

After having created

- DIFSUB subroutines for the damper force,
- REQSUB subroutine to compute the electrical damper valve currents,
- communicators to provide sensor data,
- differential equation systems elements in the subsystems chassis (for calling REQSUB) and the axles (for calling DIFSUB),

and having modified the damper to use the damper force computed by the DIFSUB routine instead of the usual force-velocity curve, the model has been completed and tested with some simple maneuvers.

Braking

A pitch excitation has been provoked by simulating a short brake action (fig. 4). Setting the damper current to a fixed value of 1800mA, which corresponds to the softest setting, the pitch motion is not sufficiently damped. Applying the control algorithm, the motion is quickly damped (fig. 5). Then the dampers return to the soft, comfortable setting again (fig. 6).

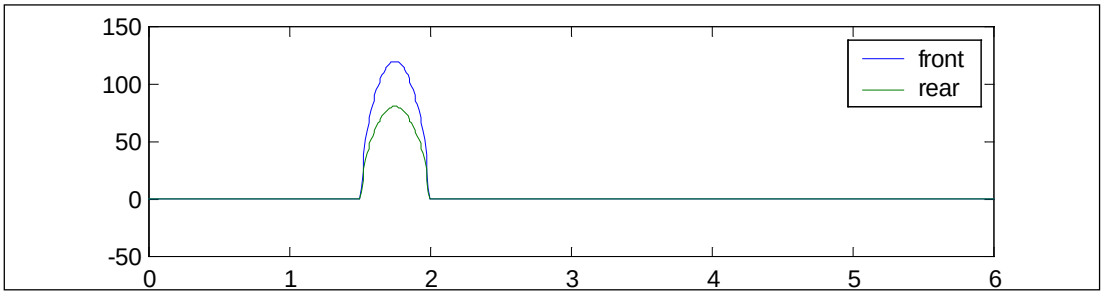

Figure 4: Brake pressure [bar] vs. time [s] for brake test.

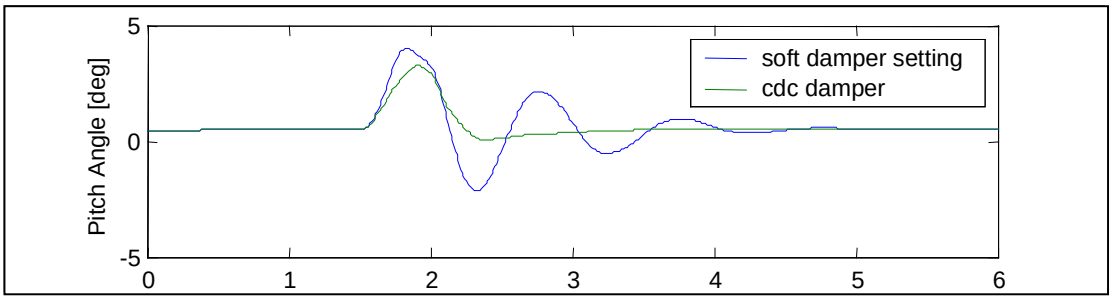

Figure 5: Pitch angle for brake test

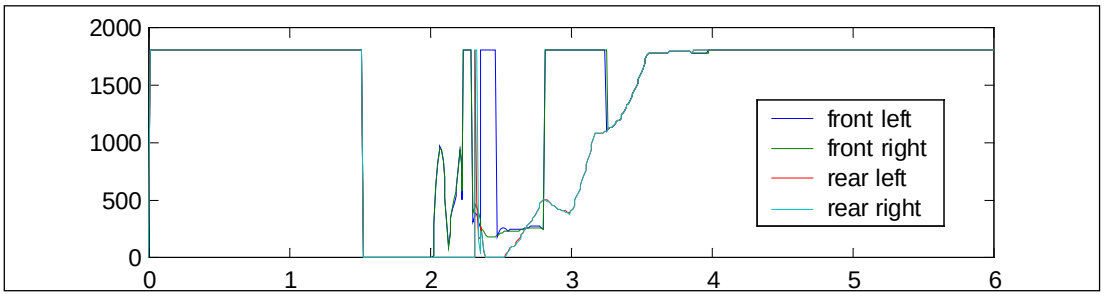

Figure 6: Damper currents [mA] for brake test with activated control. The values range from 0 mA (hard) to 1800 mA (soft).

Step Steer

When applying a step steer (90 degrees steering angle in 0.5 seconds), the soft damper is not sufficiently damping the rolling excitation. The controlled damper performs slightly better than the damper with the hard setting, because it applies less damping while approaching the final roll angle (fig. 7, 8).

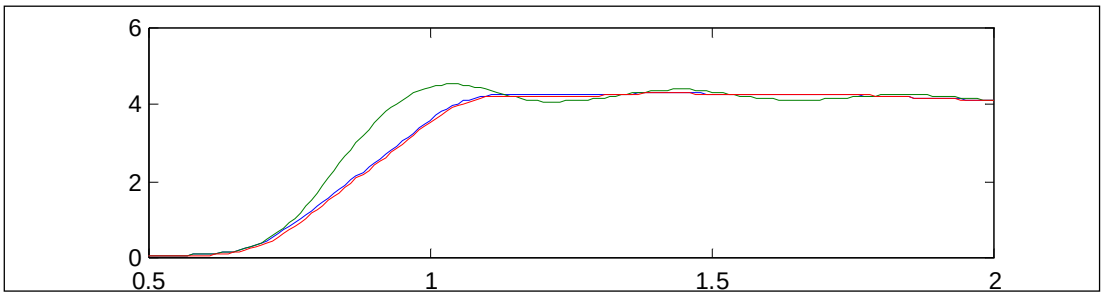

Figure 7: Step steer. Roll angle [deg] vs. time[s]. The soft setting (green) is not sufficiently damped. The controlled damper (blue) reacts slightly faster than the stiff damper (red).

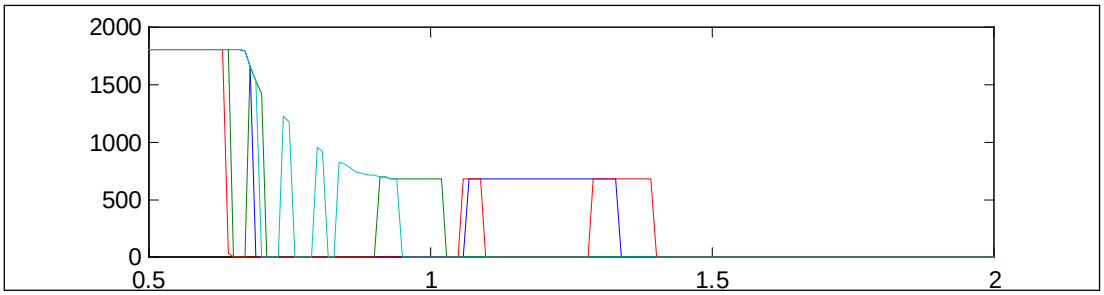

Figure 8: Damper currents for the step steer test with activated control.

Sinusoidal Road

On a sinusoidal road with a wave length of 24 meters and 24 mm wave height, the excitation frequency is slightly above the eigenfrequency of the vehicle at a velocity of 100 km/h. Therefore, the soft damper setting is not able to sufficiently damp the pitch oscillation of the vehicle. It is also visible that the sky hook control achieves lower pitch oscillations than the damper with the stiff setting (see fig. 9).

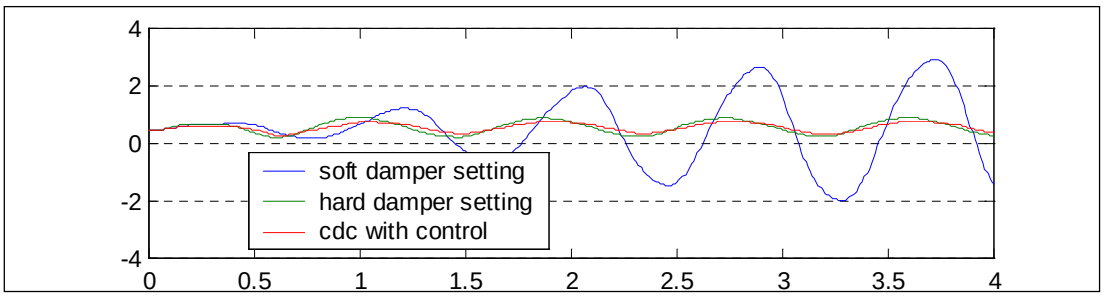

Figure 9: Sinusoidal road: pitch angle [deg] vs. time[s]. Due to the sky hook control, amplitudes remain smaller when the damper force is controlled.

Summary

It has been demonstrated that the integration of a damper control into an ADAMS/Car model can be achieved by calling C-Code describing the control algorithm from ADAMS user subroutines. The C-Code can be generated from a Simulink representation by the Real-Time Workshop extension of Simulink.

This method allows to remain in the ADAMS/Car environment to define and run full vehicle simulations. Examples have been shown which demonstrate the benefit of a controllable damper system. It not only offers the increased comfort of soft dampers while still providing stiff behavior in

curves or braking conditions, it also contains sky hook control algorithms to reduce oscillations of the car body.
This project has also shown that exchanging control algorithms as binary subroutine code is a practicable way to enable the car manufacturer AUDI AG to integrate code developed by the supplier Continental Teves into his simulation environment.