Otto–von–Guericke–Universität Magdeburg

Praktikumsbericht

Modelling and control of an active hydro–pneumatic suspension

von

Florian Knorn
(* xxx Dez. 1981 in Berlin)

1. April 2006

Eingereicht an die: Otto–von–Guericke–Universität Magdeburg
Fakultät für Verfahrens– und Systemtechnik
Prüfungsamt
Universitätsplatz 2
Postfach 4120, 39016 Magdeburg
Deutschland

Betreuer: Prof. Jens C. Kalkkuhl

DaimlerChrysler Research & Technology
REI/AR, Vehicle Dynamics Systems
Hanns–Klemm–Str. 45,
71034 Böblingen
Germany

Table of contents

Acknowledgments

I count myself in nothing else so happy
As in a soul remembering my good friends.

— Henry Bolingbroke in William Shakespeare's
"The Tragedy of King Richard the Second"

A number of lucky coincidences have led to the opportunity to do my internship in the prestigious DaimlerChrysler AG. I would like to kindly thank Prof. Robert Shorten from the Hamilton Institute in Maynooth, Ireland for recommending me to Prof. Jens Kalkkuhl who was my supervisor here at the Vehicle Dynamics Systems group of DaimlerChrysler Research & Technology.

It was a great honour for me to become part of his impressive group and field of work. I enjoyed working together with Jens and the many other talented, friendly and supportive people, and I fully appreciate the valuable time they spent with me and the many things they taught me. I am very grateful for the six months here in Böblingen as they allowed for many insights into the interesting work of a state of the art research group in a large and renown corporation, working on the very future of cars.

My fascination and love for cars has been greatly satisfied; I could not state it better than with the words of Prof. Shorten from a private conversation with me: "For an engineer, it hardly gets more exciting than with cars!"

Also, I am deeply indebted to Dr. Mehmet Akar and Carlos Villegas, two former collegues from the Hamilton Institute, who helped me a lot from afar with many technical and control theoretical issues.

Part of this great experience were also the following people, colleagues and friends. I would like to thank them for the great time they made me have here in the "Schwoobeländle" (titles omitted, alphabetical order): Adrian Thomys, Anders Witt, Anne Ruhnke, Annekatrin Geier, Antonietta Angoretti, Arnaud Sablé, Avshalom Suissa, Brad Schofield, Carlos Rafael Da Cunha, Christian Arnold, Christophe Gosteaux, Daniela Wildenstein, Daniel Goldbach, Daniel Keppler, Diego Gonzalez, Henning Everth, Jörg Schwarzmeier, Julia Frommelt, Julika Steen, Katrin Thyrassa, Lawrence Louis Gilbert, Magnus Rau, Magnus Lewander, Parshant Jagdish Narula, Rebecca Quattelbaum, Stefan Zanger, Stephan Zschäck, Stephanie Rapphahn, Theresa Krippl, Ulrike Eilers, Verena Wuchenauer, ...

Finally I would like to express my deep gratitude toward my family as well as my closer friends for their never ending support and love.

Introduction

The company

Just about every one knows and associates the name "Mercedes" to cars. It is close to being an international synonym for high quality, solid yet luxurious and innovative cars, "Made in Germany".

However, the DaimlerChrysler Corporation is much more than just *Mercedes*. Its product range extends from small consumer cars to large commercial vehicles, and contains many fascinating brands such as *Maybach*, *Dodge*, *Jeep*, or *smart* as well as *Evobus*, *Setra*, or *Sterling Trucks* just to name a few.

This large multinational corporation employs close to 400,000 people, produces in 17 countries and sold in 2004 more than 4 million cars and 700,000 commercial vehicles. Continuing research and innovation lay the base of its future and reputation, offering the customers the very best in terms of design, quality, safety and also enjoyment of using one of its products. In the past year alone, more than 5.7 billion Euro have been invested in research.

The DaimlerChrysler Research and Technology is divided into many departments. In Böblingen / Hulb, close to Stuttgart, I worked at the "Active Safety and Driver Assistance Systems" sub–department (REI/AR), which is part of the "Research E/E and Information Technology" department (REI). In particular, I helped out in the Vehicle Dynamics Systems group of Avshalom Suissa, supervised by Prof. Jens Kalkkuhl.

This group has several projects running at the moment, for instance vehicle load estimation, drive–by–wire systems or one vehicle imitating the dynamic behaviour and driving feeling of another (that is part of "rapid prototyping). I would now like to briefly introduce the project I was working on.

The internship

The focus of my internship lay on improving a controller that was deployed in an Active Hydro–Pneumatic suspension (AHP). This is a relatively new suspension design with the particularity that it works without the classical mechanical parts of a suspension, such as steel springs and dampers. Instead a *hydraulic* system is used, which consists, roughly speaking, of a plunger cylinder, a flow resistance, a hydro–pneumatic capacitor and a strong hydraulic pump together with a fast response servo valve.

At the heart of each AHP suspension strut, there is a force controller, which is responsible for tracking a certain desired force (calculated by other, higher level or "outer loop" controllers). The current controller, which is of the PI–

type, does not allow for sufficient performance in the context of its application, and it was my task, to come up with a "better" one.

Figure 1: *The test car PEGaSOS.*

In the bigger picture, our test vehicle "PEGaSOS"[1] is part of the CEmACS[2] project, an interesting public project spread across and funded by several institutions, such as DaimlerChrysler Research & Technology (Germany), the Hamilton Institute at NUI Maynooth (Ireland), Lund University (Sweden), Glasgow University (Scotland) and SINTEF (Norway)[3].

Among many other things, one aspect of the project is developing a test vehicle that can be used as a simulator for the dynamics of other (and potentially not, or not–yet–existing) vehicles. Using steer–by–wire, four–wheel–steering and the AHP suspension, PEGaSOS will allow to realistically imitate the handling characteristics and driving feeling of another vehicle that is given only by its specific vehicle dynamics (reference) model.

The report

The structure of this "Praktikumsbericht" will roughly be as follows:

Chapter 1 introduces the set–up of the system as well as a rough overview of the control structure involved. We shall not go into the technical details of

[1] **P**rüfstand zur **E**ntwicklung und **Ga**nzheitlichen **S**imulation **O**ptimierter Fahrzeug**S**ystemdynamik — testbed for the development and wholistic simulation of optimised vehicle system dynamics

[2] **C**omplex **Em**bedded **A**utomotive **C**ontrol **S**ystems

[3] **S**tiftelsen for **IN**dustriell og **TE**knisk **F**orskning — Foundation for Scientific and Industrial Research (at the Norwegian Institute of Technology)

the hard– and software running on the car as it is less interesting for the scope of this report.

The second chapter then will establish a mathematical model of the system. Here, we will focus particularly on simplicity as this will facilitate the later control design. The model will also be validated briefly with experimental data.

In the third chapter, we design different types of new non–linear controllers based on our mathematical model. Particular attention is put on their robustness with respect to parameter uncertainties.

We shall finally test and compare in Chapter 4 the performance of the different controllers. Extensive simulations are to shed some light on the desired robustness with regard to parameter changes.

After some concluding remarks the Appendix holds some of the SIMULINK models and MATLAB files created and used during the internship, as well as additional plots and files.

CHAPTER 1

The existing system

We introduce the physical set–up of the AHP and the concept of the existing cascaded control system. The current force controller is described and we show an example of its performance.

1.1 Introduction

This first chapter serves as a presentation of the set–up of PEGaSOS's suspension and the control system related to it.

After introducing the physical make–up of the system we shall take a closer look at the cascaded control systems involved. Here, the innermost control task is to set and track a certain pressure in the cylinder, which translates into a force then exerted by the piston. The reference force (pressure) is the result of several higher (outer) control loops that, for instance, compensate the cars tendency to roll in corners or pitch with changing longitudinal acceleration, and, of course, to track a reference model when imitating another vehicle.

It is important to note that everything discussed here concerns a single wheel only. In the car however, (but for the pump) four of these systems are required, one for each wheel.

1.2 Set–up of the system

So let's start by introducing the different parts of the AHP.

1.2.1 Components

The set–up of the hydro–pneumatic system is shown in Figure 1.1 on the following page. Here, one suspension strut is made up of a

- cylinder (index "z"), which is connected directly to the wheel[1]
- hydraulic capacitor (index "s") or "gas spring", consisting of two chambers: one connected to the oil circuit, the other, separated by a membrane, contains a gas
- laminar resistance between the cylinder and the capacitor

[1] The cylinder is so–called plunger cylinder, as it is connected to the oil system on only one side of the piston.

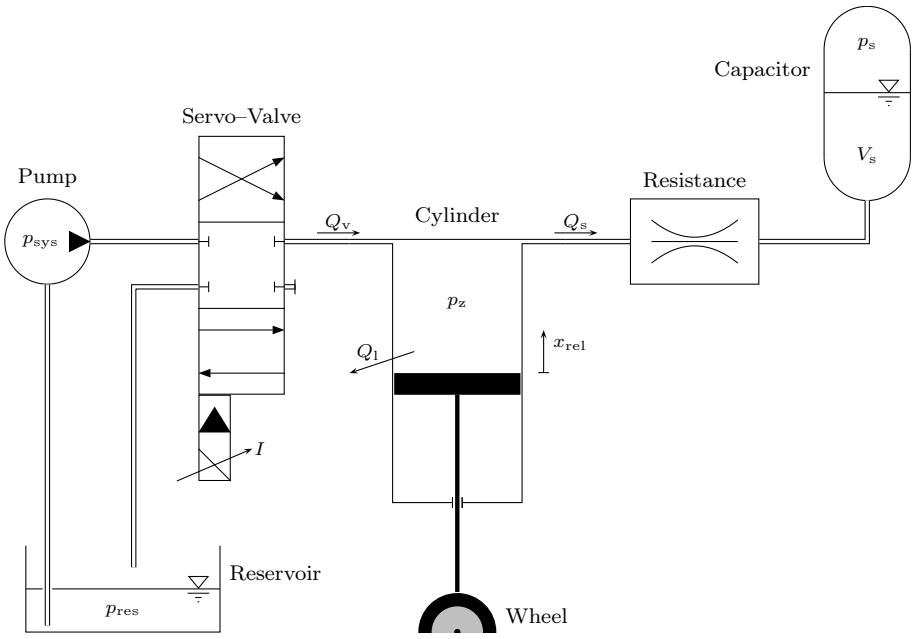

Figure 1.1: *Basic set–up of the system and some of the state variables.*

– hydraulic pump connected to the car's engine

– 4/3 servo valve which controls the in– and outflow of oil to and from the system

At this point, we have already made a number of simplifications. For instance, one might also consider the resistances of the other conducts, or assume the resistance not to be laminar.

1.2.2 Signals

As we are dealing with a hydraulic system, the major types of variables are pressures and flows. Most of the signals — and the interconnection and –action of the different parts of the system — are shown in Figure 1.2 on the next page.

We have the pressures in cylinder and capacitor (p_z and p_s), but also the system pressure p_{sys} and the pressure in the reservoir p_{res}. All these pressures are measured by suitable sensors and are available for use in controllers.[2]

A few comments on p_{sys}: As the pump is connected directly to PEGaSOS's engine, p_{sys} is not always constant. The nominal pressure output of 180 bar is guaranteed from around 2000 r.p.m. on, and we can safely assume that this pressure level is available under normal conditions. However, it *can* drop in cases (which should be prevented when tests are run). This is why we technically consider p_{sys} to be a disturbance, as we do not have any influence

[2] Concerning p_s, I should rather say *should*, as I was to find out that the responsible sensors for the two front wheels were not functioning, and the ones for the rear were anything but calibrated.

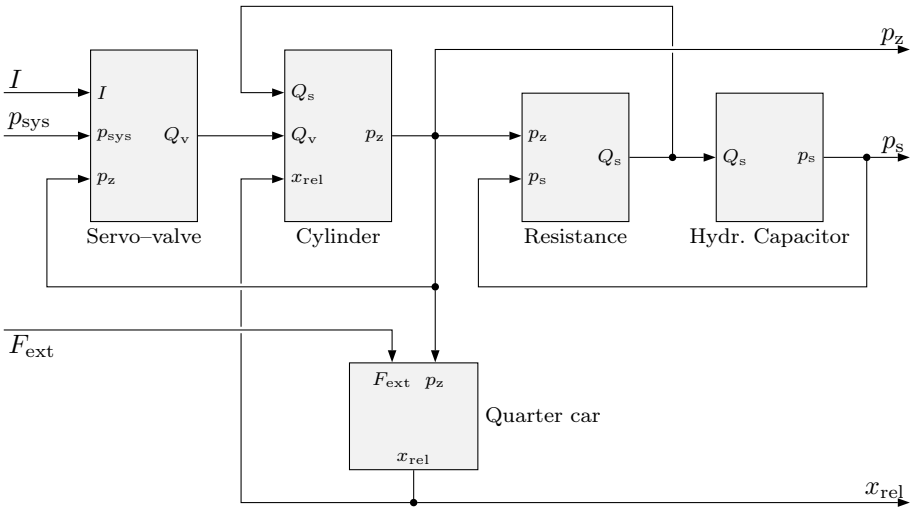

Figure 1.2: *Signals in the system.*

on it (in the scope of the system we consider here), although we can measure it.

The reservoir pressure p_{res} to the contrary is always constantly low, roughly equal to atmospheric pressure, so usually around 1 bar.

The flow variables in the system are Q_v and Q_s, respectively corresponding to the (directed) flows of oil from valve to cylinder and from cylinder to capacitor (the amounts of oil in those shall be called V_z and V_s). Additionally, we allow for some leaking of oil, Q_l, which is positive for oil leaving the system. This could be oil leaving the system through some worn out fittings, especially in the valve, where it may very well be that a certain amount of oil does not flow into the system, but directly into the reservoir for instance.

We then have the control current for the valve I, which, for positive I, injects oil into the system, and for negative I allows oil to leave it. The position of the plunger is x_{rel}; it is zero at the neutral (middle) position, positive if it is "above" that position (cf. figure), negative when it is "below" it. We also allow for some external force $F_{ext}(t)$, which could result from the car running over a bump on the road for instance.

As mentioned above, a certain pressure in the cylinder translates (via the effective surface of the plunger) into a force. This force, diminished by some friction, will accelerate the body of the car sitting on top of the cylinder. The

Inputs	Disturb.	States	Outputs
I	p_{sys}	x_{rel}	x_{rel}
	F_{ext}	$\dot{x}_{rel}$	p_z
	Q_l	p_z	p_s
		p_s	

Table 1.1: *Overview of key variables.*

movement of the body then results in a movement of the piston relative to the body of the car, which is also measured by a sensor. The position of the piston is called x_{rel}; it is zero in the neutral (middle–) position of the plunger, and positive if above it, i. e. when the wheel moves in– or upwards.

Before moving on to the different controllers, we finish this section on the description of the basic set–up of the system with Table 1.1 which shows an overview of the variables that are important from a systems theoretical point of view.

1.3 Outer control loops

The cascaded control system of PEGaSOS' AHP suspension contains two "layers". Several (parallel) components calculate a desired force (or reference force) on the outer layer, which is then to be set and tracked by the inner loop.

For an overview, the different control loops involved in the AHP are shown in Figure 1.3. Double lines signify the flow of more than one signal. Again, all this is on a per–wheel basis.

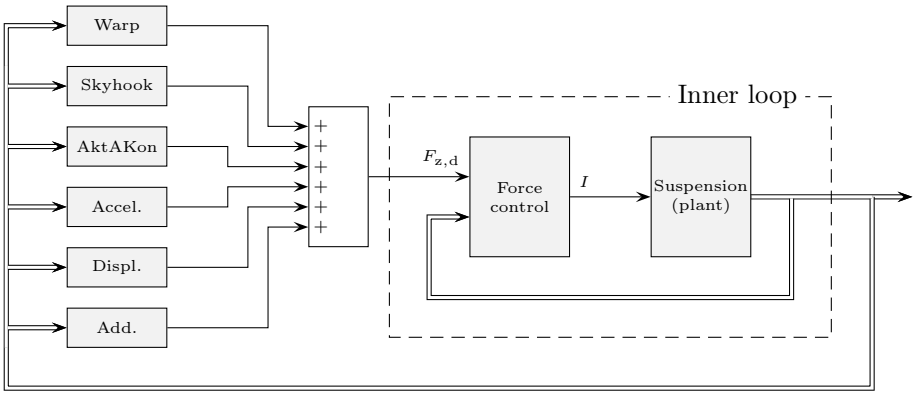

Figure 1.3: *Basic set–up of the cascaded control system for one wheel.*

1.3.1 Components

Good conceptual introductions to the topic can be found in [1], [2] and [6]; for reasons of confidentiality however we are limited here to only superficial descriptions.

Warp control

This controller takes care of the warping of the car. Warping is generally understood to be the cars (elastic) "twisting" along the longitudinal axis of the car. If you increase the force exerted by diagonally opposite wheels, the car's chassis will be slightly (and usually not noticed by the driver) twisted, or "warped".

A number of interesting and useful things can be done with warping, but this is an ongoing topic of research and thus strictly confidential.

Skyhook

Riding comfort is an essential feature of modern consumer cars. As its name suggests, the Skyhook control loop tries to eliminate jerks, jolts or vibrations resulting from bumpy or uneven roads, as if the car was not riding on the rough ground, but "hooked" into the (presumably more comfortable) skies.

AktAKon

This abbreviation stands for **Akt**ive **A**ufbau**Kon**trolle — active body control. Unfortunately I cannot say anything beyond this, as it is also confidential.

Acceleration feed–forward

In addition to AktAKon, this feed–forward component is used to compensate the cars tendency to pitch when accelerating or breaking, or roll when cornering. Part of this taken care of by the AktAKon, but this component is used to further speed up the compensation.

Displacement control

Each of the above parts does not care about the actual position of each of the cylinder pistons. The range within which these are allowed to move is obviously limited, and this control loop's task is to watch that the pistons stay close to the neutral (middle–) position, so that they have room in both directions to move, if necessary.

Additional force control

For adding custom forces to the other ones, this simple controller is set–up mainly for experimental reasons.

1.3.2 Functionality

As shown in Figure 1.3 on the preceding page, each controller basically "votes" for a certain force to be exerted. These reference forces are then simply added up and fed into the inner loop, which takes care of the executive part, i. e. actually tracking the resulting reference force. Obviously, the sketch above is of rather simplified nature — the reference values for each of the outer controllers for instance are not shown (they are supposed to be inside each controller).

Unfortunately, we cannot give an in–depth description of, for example, the actual signals flowing in the outer loop. This is different, however, for the inner loop, which will be presented in the next section.

1.4 Inner loop

We shall now take a quick look at the existing controller in the inner loop. By its structure it is a PI–controller with a PT_1 pre–filter (first order delay with corner frequency of 8 Hz, corresponding to a time constant of about 20 ms). Its input is the difference between current and reference force exerted by the

cylinder, and its immediate output is a desired oil flow into (or out of) the system.

As the actuator is the valve, which takes a specific control current and "translates" it into the wanted oil flow, an inverse valve model is used to determine the necessary control current needed.

The inverse valve model (termed valve compensation) is nothing but a simple inversion of the valve model proposed in Subsection 2.2.4 on page 10, i. e. (2.4) is solved for I. However, instead of using measured pressure differences, as done there, this inverse model uses a *constant* pressure difference of 100 bar. Resulting deviations are automatically compensated by the controller's inherent robustness. Note that the valve compensation is also assumed to be part of the controller; for that reason we let I be the controller's output in Figure 1.3.

The controller gains from $F_{z,d} - F_z$ to I are (including the inverse valve model) $k_P = 0.30$ mA/N for the proportional component and $k_I = 0.03$ mA/N for the integral part.

So in the Laplace–domain, its controller equation is

$$\begin{aligned} u &= k_p e' + k_i \frac{e'}{s} \\ e' &= \frac{1}{0.02s + 1}(F_{z,d} - F_z) \end{aligned} \tag{EPC}$$

with s being the Laplace variable.

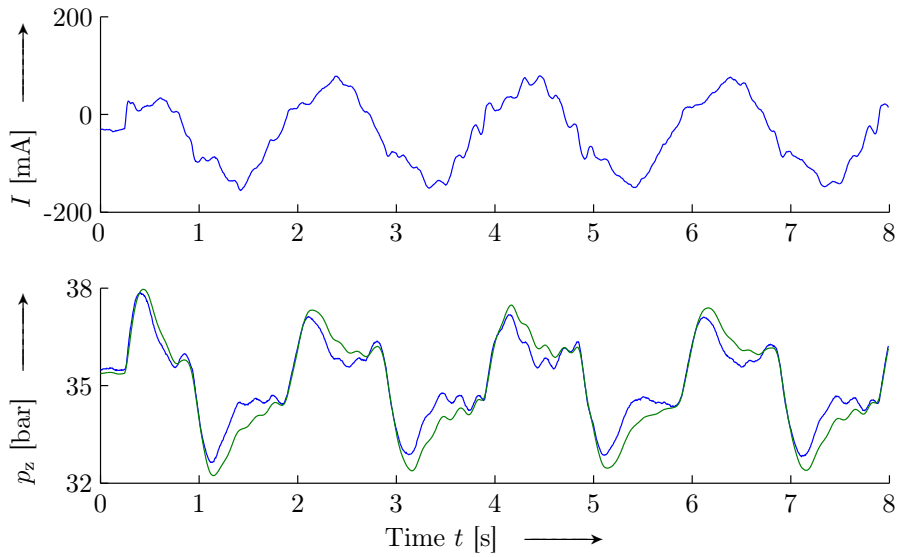

Figure 1.4: *Comparison of reference pressure (——) with the actual cylinder pressure (——) when using the existing force controller.*

To close this chapter we show the performance of the existing PID–controller using some real data from the car. The task was to make the car pitch in a sinusoidal motion, i. e. x_{rel} was to describe a constant sine wave. The resulting

desired pressure signal is shown in Figure 1.4 on the preceding page along with the actual measured response.

Although the achieved tracking performance is sufficient for normal driving operations (i. e. keeping the car level, etc.), it can certainly be improved by more advanced control concepts. These should be able to reduce the visible phase lag for instance. We shall investigate these possibilities in Chapter 3, but we first need to get a mathematical model of the system.

CHAPTER 2

The model

Each of the system's components will be modelled, then a model for the whole system shall be derived. Finally, we'll give the values of all parameters involved and validate the model.

2.1 Introduction

In this chapter, we would like to build a general model of the AHP. We will assume a mass sitting on top of the suspension to be able to also simulate the motion of the plunger in the cylinder.

We will introduce mathematical descriptions of the different parts to model their behaviour, and then connect them together to find a model of the system as a whole.

All models here focus on simplicity, i. e. we try to keep things as simple as possible (but still sufficiently precise). That will later allow a relatively inexpensive controller design and implementation.

At the end of the chapter the values for most of the parameters will be given together with a number of friction models.

A more detailed description of the system and in depth physical background can be found in the excellent Diplomarbeit by Magnus Rau, [7].

2.2 modelling of the components

We shall now give mathematical models for each of the components introduced in Section 1.2 on page 1.

2.2.1 Cylinder

Again, focusing on simplicity, we take the approach that the oil inside the cylinder is slightly compressible. A simple linear assumption then would be that the current pressure is proportional to the current amount of oil in the cylinder relative to some nominal oil volume.[1] The proportionality factor here corresponds to the bulk modulus of elasticity of the oil used.

$$p_z = E\frac{V_z}{V_{z0}}$$

[1] That is, the volume of oil in the cylinder when it is in neutral (middle) position.

The total amount of oil in the cylinder corresponds to the integral over all the flows involved, that is Q_v, Q_s, Q_l but also over the change of volume resulting from the piston movement, so

$$p_\mathrm{z} = \frac{E}{V_\mathrm{z0}} \left[\int_0^t \left(Q_\mathrm{v} - Q_\mathrm{s} - Q_\mathrm{l} + \dot{x}_\mathrm{rel} A_\mathrm{z} \right) \mathrm{d}\tau \right]$$

with A_z being the surface of the cylinder. We thus have

$$p_\mathrm{z} = \frac{E}{V_\mathrm{z0}} \left[x_\mathrm{rel} A_\mathrm{z} + \int_0^t \left(Q_\mathrm{v} - Q_\mathrm{s} - Q_\mathrm{l} \right) \mathrm{d}\tau \right] \tag{2.1}$$

2.2.2 Hydraulic capacitor

A simple way of modelling this element would be to assume a polytropic state change of the gas in the chamber. When oil is forced into the capacitor, the gas gets compressed, which results in an increase of pressure. This compression is assumed to be adiabatic[2] relative to some initial or nominal state (p_a,V_a):

$$p_\mathrm{s} = p_\mathrm{a} \left(\frac{V_\mathrm{a}}{V_\mathrm{s}} \right)^\kappa \tag{2.2}$$

2.2.3 Laminar resistance

This is probably the simplest element in the system. Quite intuitively, the flow through the valve is proportional to the difference in pressure on both sides:

$$Q_\mathrm{s} = \frac{1}{R_d} \left(p_\mathrm{z} - p_\mathrm{s} \right) \tag{2.3}$$

with R_d being the laminar resistance of the restrictor and the used conduits.

2.2.4 Servo valve

The valve to the contrary is a slightly more involved element. We limit ourselves to a very superficial description of it. In a good approximation, the valve can be seen as a restrictor with variable diameter. With increasing control current, the valve opens proportionally and reduces its resistance. This works in two directions: for positive I the system is connected to the pump, and due to the high pressure level there, oil is forced *into* the system. A negative control current results in the connection of the system to the reservoir, and with the very low pressure there, oil flows *out* of the system.

[2] That is no heat exchange with the environment takes place — an assumption justifiable by the fact that the compression and decompression of the gas usually happen very fast.

Inside the valve we have sharp, rectangular edges. For that reason, it is common to assume that the flow is proportional to the square root of the respective pressure difference. We also Concentrate all valve related gains into a single gain k_v, the overall gain of the valve. It describes what oil flow a certain control current ultimately results in (for a certain pressure p_z).

Saturating I at $\pm I_\mathrm{s}$ (as the opening fraction of the valve is, of course, physically limited), the model equations would then be

$$Q_\mathrm{v}(I, p_\mathrm{z}) = \begin{cases} k_\mathrm{v}\,\mathrm{sat}(I)\sqrt{p_\mathrm{sys} - p_\mathrm{z}} & \text{for } I \geq 0 \\ k_\mathrm{v}\,\mathrm{sat}(I)\sqrt{p_\mathrm{z} - p_\mathrm{res}} & \text{for } I < 0 \end{cases} \tag{2.4}$$

Obviously, because of the square root, this model is only valid when the cylinder pressure is smaller than the pump pressure, and larger than the reservoir pressure, i. e. $p_\mathrm{res} \leq p_\mathrm{z} \leq p_\mathrm{sys}$. This, however, is very unlikely to happen under normal operating conditions.

Even using rather expensive high performance (aerospace industry) servo valves with impressive dynamic behaviour, one could — in order to make the model dynamically more precise — include for instance a PT_2 term to account for the small but existent dynamics in the valve. Obviously, the control piston in the valve has a weight which already introduces delays through its inertia, left alone a number of other factors causing further small delays.

Analysing real data, and in accordance with the manufacturer's recommendations, a deadbeat PT_2 with (double) time constant $T_0 = 0.005$ s, corresponding to a corner frequency of 200 rad/s or about 31.8 Hz, gave best results.

2.2.5 Quarter car

As mentioned earlier, the pressure in the cylinder translates into some force that pushes the piston outward. To determine the piston's movement we assume a mass ("quarter car") sitting on the cylinder. The most classical approach then would be to relate the acceleration of the mass to the sum of the forces acting on it:

$$m\ddot{x}_\mathrm{rel} = F_\mathrm{ext}(t) + mg + F_\mathrm{fr}(\dot{x}_\mathrm{rel}) - A_\mathrm{z} p_\mathrm{z} \tag{2.5}$$

with m being the effective mass weighing down onto the suspension, g the gravitational acceleration and F_fr representing all effects of friction involved.

As friction forces in the cylinder can have significant effects on the pressures and thus must not be neglected, we now propose a number of friction models. However, as they are not directly important in our later controller design, we limit ourselves to only stating them:

(i) No friction: $F_{\mathrm{fr}\,1}(x_2) \equiv 0$

(ii) Coulomb friction[3]: $F_{\mathrm{fr}2}(x_2) = -\mu_{\mathrm{r}} A_{\mathrm{z}} p_{\mathrm{z}0} \sin\left(\tan^{-1}(k_0 x_2)\right)$

(iii) Coulomb and viscous friction: $F_{\mathrm{fr}3}(x_2) = -\operatorname{sign}(x_2)\big(F_{\mathrm{c}1} + d_{\mathrm{v}1} \cdot |x_2|\big)$

(iv) Coulomb, viscous and stiction friction:
$F_{\mathrm{fr}4}(x_2) = \dfrac{F_{\mathrm{c}2}}{\pi/2} \tan^{-1}(-k_1 x_2) + \dfrac{F_{\mathrm{m}2}}{\pi/2} \tan^{-1}(-k_2 x_2) - d_{\mathrm{v}2} x_2$

A detailed description of these models can be found in [5] or in any physics textbook; model (iv) was taken from [7]. Plots of these models are shown in Figure 2.1.

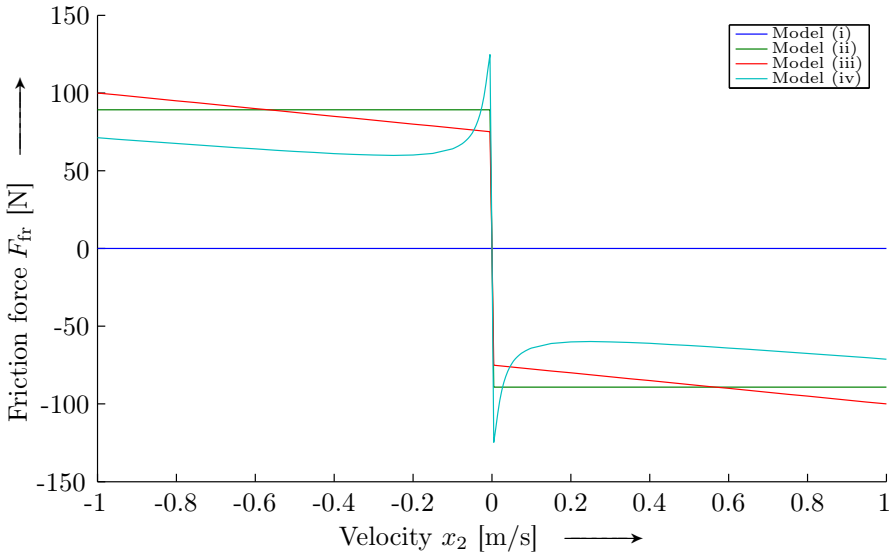

Figure 2.1: *Comparison of the different friction models suggested.*

2.3 modelling of the system as a whole

Now that we have got mathematical model of each of the components involved, we will combine them to derive a compact, non–linear state space model of the system.

2.3.1 Desired System

We aim at creating a system with the following four state variables

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_{\mathrm{rel}} \\ \dot{x}_{\mathrm{rel}} \\ p_{\mathrm{z}} \\ p_{\mathrm{s}} \end{pmatrix}$$

[3] The "classical" and most basic model would probably be $F_{\mathrm{fr}}(x_2) = -\operatorname{sign}(x_2) F_{\mathrm{c}}$, however, for simulation purposes (numerical issues) we prefer to use this continuous and smooth function, which approximates the classical curve very well.

with single input

$$u = I$$

and multiple output

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_{\mathrm{rel}} \\ p_{\mathrm{z}} \\ p_{\mathrm{s}} \end{pmatrix}$$

2.3.2 Putting it all together

In order to obtain the system above, we start off by transforming (2.5) on page 11 into

$$\ddot{x}_{\mathrm{rel}} = \frac{1}{m}\left(F_{\mathrm{ext}}(t) + mg + F_{\mathrm{fr}}(x_2) - A_{\mathrm{z}} x_3\right) \tag{2.6}$$

Differentiating (2.1) on page 10 with respect to the time t and using (2.3), we get the dynamic equation of the cylinder pressure p_{z}

$$\dot{p}_{\mathrm{z}} = \frac{E}{V_{\mathrm{z0}}}\left[A_{\mathrm{z}}\dot{x}_{\mathrm{rel}} + Q_{\mathrm{v}} - \frac{1}{R_{\mathrm{d}}}\left(p_{\mathrm{z}} - p_{\mathrm{s}}\right) - Q_{\mathrm{l}}\right] \tag{2.7}$$

Concerning the pressure in the capacitor p_{s} we can write, by differentiating (2.2) on page 10 with respect to t,

$$\dot{p}_{\mathrm{s}} = p_{\mathrm{a}} V_{\mathrm{a}}^{\kappa}\left(-\kappa\right)\dot{V}_{\mathrm{s}}\, V_{\mathrm{s}}^{(-\kappa-1)}$$

Using the simple relation $\dot{V}_{\mathrm{s}} = -Q_{\mathrm{s}}$ and (2.2) again

$$\begin{aligned}\dot{p}_{\mathrm{s}} &= p_{\mathrm{a}} V_{\mathrm{a}}^{\kappa}\kappa\,\frac{1}{R_{\mathrm{d}}}\left(p_{\mathrm{z}} - p_{\mathrm{s}}\right)\left(\frac{p_{\mathrm{a}} V_{\mathrm{a}}^{\kappa}}{p_{\mathrm{s}}}\right)^{-\left(1+\frac{1}{\kappa}\right)} \\ &= \frac{\kappa}{R_{\mathrm{d}} V_{\mathrm{a}}\, p_{\mathrm{a}}^{\frac{1}{\kappa}}}\left(p_{\mathrm{z}} - p_{\mathrm{s}}\right) p_{\mathrm{s}}^{\left(1+\frac{1}{\kappa}\right)}\end{aligned} \tag{2.8}$$

With (2.4) on page 11 and (2.6)–(2.8) we can now establish the system as desired above:

$$\dot{x}_1 = x_2 \tag{2.9a}$$

$$\dot{x}_2 = K_1(t) + \varphi(x_2) - a_1 x_3 \tag{2.9b}$$

$$\dot{x}_3 = a_2 x_2 - a_3 x_3 + a_3 x_4 - K_2 + b(x_3, u)\,\mathrm{sat}(u) \tag{2.9c}$$

$$\dot{x}_4 = a_4\left(x_3 - x_4\right) x_4^{\bar{\kappa}} \tag{2.9d}$$

$$y_1 = x_1 \tag{2.9e}$$

$$y_2 = x_3 \tag{2.9f}$$

$$y_3 = x_4 \tag{2.9g}$$

---

with the following substitutions:

$$K_1(t) = \frac{F_{\text{ext}}(t)}{m} + g \qquad K_2 = \frac{E}{V_{\text{z0}}} Q_{\text{l}} \qquad \varphi(x_2) = \frac{F_{\text{fr}}(x_2)}{m}$$

$$a_1 = \frac{A_{\text{z}}}{m} \qquad a_2 = \frac{E A_{\text{z}}}{V_{\text{z0}}} \qquad a_3 = \frac{E}{R_{\text{d}} V_{\text{z0}}}$$

$$a_4 = \frac{\kappa}{R_{\text{d}} V_{\text{a}} p_{\text{a}}^{\frac{1}{\kappa}}} \qquad \bar{\kappa} = 1 + \frac{1}{\kappa}$$

and

$$b(x_3, u) = \begin{cases} \dfrac{E k_{\text{v}}}{V_{\text{z0}}} \sqrt{p_{\text{sys}} - x_3} & \text{for } u \geq 0 \\ \dfrac{E k_{\text{v}}}{V_{\text{z0}}} \sqrt{x_3 - p_{\text{res}}} & \text{for } u < 0 \end{cases}$$

Based on the above system and parameters we created for simulation and validation purposes a SIMULINK model, which is described in more detail in Appendix A on page 33.

2.4 Parameters and Validation

To complete this chapter on modelling, we shall present the values for all of the parameters mentioned earlier and introduced so far, and validate them showing an example of how well our mathematical model actually allows to describe the system.

2.4.1 Parameters

A considerable amount of time has been spent on estimating some of the parameters in the system — a common problem in real live applications is that many parameters cannot be found in textbooks, calculated or measured directly with sufficient precision. Also, most of PEGaSOS' suspension is custom built, thus there are no manufacturing manuals or technical detail sheets to consult.

The identification was done using, of course, the above model together with measurements taken from the car. MATLAB's `fminsearch` was then set–up to fine–tune some of the parameters in question, the objective being the minimisation of the quadratic error between the system states and the measured behaviour of the real system (to be more precise, a linear combination of the integrals over the squared deviations of p_{z} and x_{rel}). A more detailed explanation of the process can be found in Section B.2 in the Appendix.

With the parameter estimation done, we can now show all the relevant parameters together with their respective values and units in Table 2.1 on the facing page.

Note that it is impossible to give a function or representation for $F_{\text{ext}}(t)$, as it depends on many outside factors, especially on the particular driving manoeuvre. If the car is standing still however, $K_1(t)$ would disappear. For these reasons $K_1(t)$ does not appear in the table. Please also note the comment in Subsection 1.2.2 on page 2 concerning p_{sys}.

Par.	Value	Unit	Par.	Value	Unit
m	365	kg	g	9.81	$\mathrm{m/s^2}$
E	$2.00 \cdot 10^8$	Pa	V_{z0}	$8.16 \cdot 10^{-5}$	$\mathrm{m^3}$
Q_l	≈ 0	$\mathrm{m^3/s}$	A_z	$10.2 \cdot 10^{-4}$	$\mathrm{m^2}$
R_d	$2.08 \cdot 10^9$	$\mathrm{Pa/(m^3/s)}$	p_a	$4.26 \cdot 10^6$	Pa
V_a	$1.13 \cdot 10^{-4}$	$\mathrm{m^3}$	κ	1.36	–
k_v	$5.90 \cdot 10^{-7}$	$\mathrm{(m^3/s)/(Pa^{\frac{1}{2}}A)}$	I_s	1.00	A
p_{sys}	180	bar	p_{res}	1	bar
K_2	≈ 0	Pa/s	a_1	$2.80 \cdot 10^{-6}$	m
a_2	$2.50 \cdot 10^9$	Pa/m	a_3	$1.18 \cdot 10^3$	$\mathrm{s^{-1}}$
a_4	$7.76 \cdot 10^{-}11$	$\mathrm{(m^2/s)/(Pa^{1+\frac{1}{\kappa}})}$			
μ_r	$2.5 \cdot 10^{-2}$	–	p_{z0}	$35.0 \cdot 10^5$	Pa
F_{c1}	75	N	d_{v1}	25	N/(m/s)
F_{c2}	150	N	F_{m2}	100	N
d_{v2}	20	N/(m/s)	k_0	$55.1 \cdot 10^5$	s/m
k_1	1937	s/m	k_2	-50.0	s/m

Table 2.1: *Values for the different parameters in the system.*

2.4.2 Simulation

Together with these parameters, we are now able to simulate and run our model of the AHP. In Figure 2.2 below we show the effect of three different stimulations on the system, that is two impulses of different width, amplitude and sign, as well as a sinusoidal control current input.

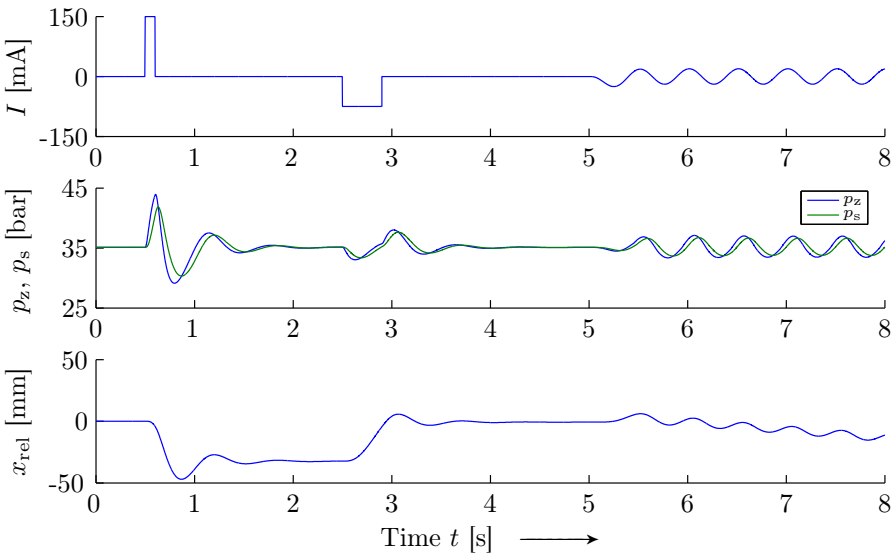

Figure 2.2: *Comparison the different system states for a certain input.*

Looking at the effect of the first impulse, we can see (in both cylinder and capacitor) that at first pressures build up to a certain level. Oil is forced into the system, but at the beginning (as the piston is not moving), most of it has to go into the capacitor. The increase of p_z and p_s starts to create a force that is greater than that generated by the weight of the car — the plunger is accelerated outwards. When it starts moving, the available volume in the cylinder increases, oil car flow back from the capacitor, and pressures decrease.

Once the plunger (and quarter car mass) are moving they have to be stopped again when the oil input ceases. Here, the opposite of the above description is happening (i. e. oil is taken from the capacitor resulting in a lack of pressure, which in turn results in a deceleration, as the weight force of the car is bigger than that generated by the piston).

There is not too much to discuss for the sinusoidal part, just that the plunger has a slight outward moving trend. This should be due to some stiction friction related effects at the beginning of the sine.

Generally, the capacitor pressure is slightly "dragging behind" p_z, and its amplitude stays below it as well. This behaviour can be understood intuitively considering the set–up of the system and the resistance between both elements.

2.4.3 Validation

To finish off this chapter on modelling, we shall quickly give two examples of how well our model and choice of parameters allows to describe the real system, for it is important to get an idea of the quality of the results from our theoretical and experimental system analysis.

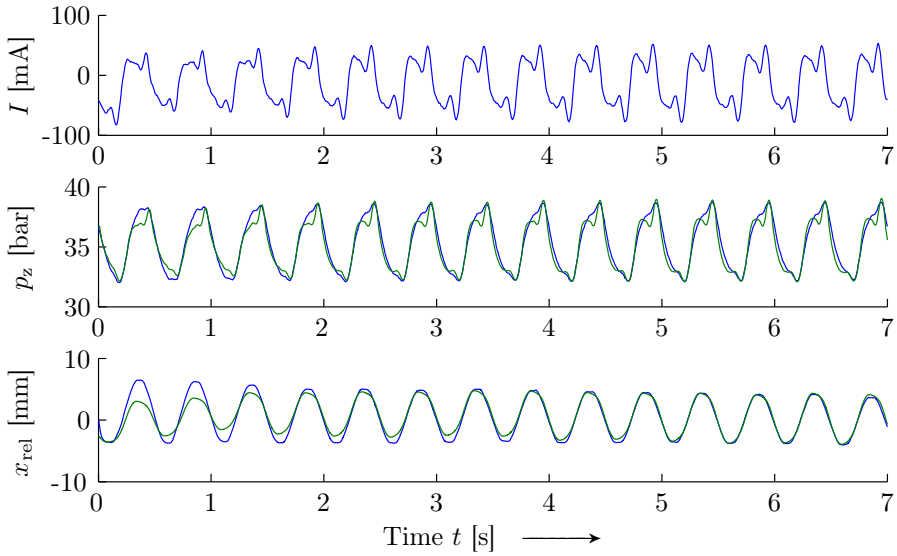

Figure 2.3: *Comparison of simulated values (——) with measurements taken in the car (——) for the input current shown in the first plot, using friction model (ii) was used.*

Standing still, on a flat surface, we made the car track a sinusoidal pitching reference, i. e. all four wheels were to describe (all in parallel) a sinusoidal

vertical movement. We then recorded the displacement x_{rel}, cylinder pressure p_{z}, control current I, as well as reservoir and system pressures p_{res} and p_{z}.

In the first plot of Figure 2.3 on the preceding page, we show the recorded control current in the upper subplot. This current was also fed into our validation model `iotest.mdl`, see Section A.3 on page 36 for more details.

Using friction model (iv)[4] and also including the PT_2 term in the valve model (as suggested in Subsection 2.2.4 on page 10) we get the result shown in lower two subplots of Figure 2.3 on the preceding page: After a few transient seconds, we can see that both x_{rel} and p_{z} are in good accordance with their measured counterparts.

However, as the controller design in the next chapter will only focus on the dynamic equation for p_{z}, we shall separately validate this equation in particular. We did so by not only using the measured current I as an input to our model, but also using the measured x_{rel}. That way, we are independent of the first two state equations and whatever friction model chosen.

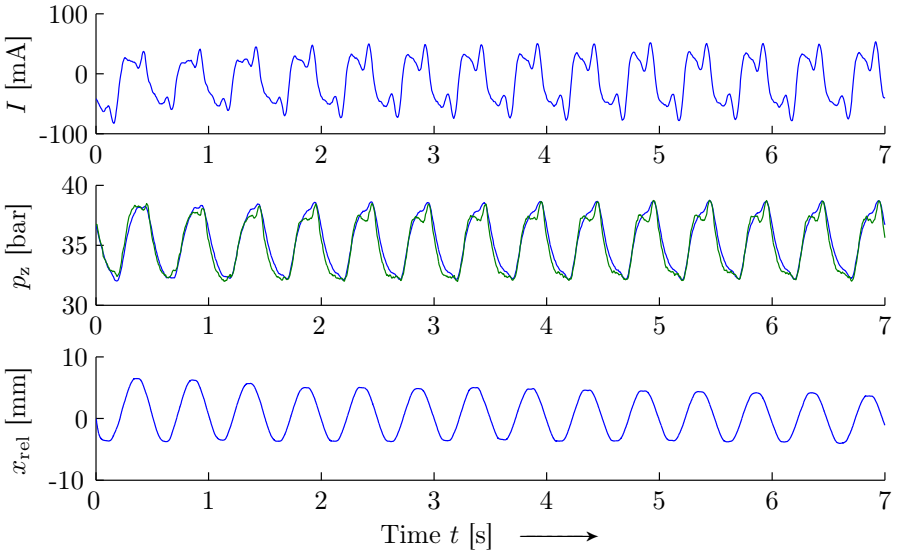

Figure 2.4: *Comparison of simulated values (——) with measurements taken in the car (——) for the input current shown in the first plot and the measured displacements from the third plot.*

As we can see in Figure 2.4 above we have, here as well, a quite good agreement between simulation and measurements, especially when considering the simplicity of our model.

This demonstrated, we can now close this second chapter and move on to designing different types of controllers based on the model we derived above.

[4] The corresponding plots for models (i) to (iii) are shown in Section C.1 on page 47.

CHAPTER 3

Controller design

In this chapter, we design different types of controllers, starting with an input–output linearising controller, followed by several sliding mode and one adaptive controllers.

3.1 Introduction

Based on the model (2.9) on page 13 of the AHP we will design different types of controllers.

We will begin with an input–output linearisation based controller, followed by a simple sliding mode controller. We will then refine it by using a continuous approximation of the switching laws as well as adding some integral action.

The last controller will be adaptive (Lyapunov function based). The performance of these controllers is then demonstrated in the last chapter.

Note that in this chapter we only focus on $x_{3,\mathrm{d}}$, denoting a desired or reference pressure (along with its derivative $\dot{x}_{3,\mathrm{d}}$). In the software already running in the test vehicle the outer control loops ask for a certain cylinder *force* to be created. This objective is however equivalent with tracking a specific pressure, as $x_{3,\mathrm{d}} = p_{\mathrm{z,d}} = F_{\mathrm{z,d}}/A_{\mathrm{z}}$.

3.2 Input–output linearising controller

For an excellent introduction to the topic, the reader is referred to [8]. Going right into *medias res*, we find without any problems that the system has relative degree $\nu = 1$, as, from (2.9c) and (2.9f),

$$\dot{y}_2 = \dot{x}_3 = \underbrace{a_2 x_2 - a_3 x_3 + a_3 x_4 - K_2}_{f(\boldsymbol{x})} + \; b(x_3, u)\,\mathrm{sat}(u) \tag{3.1}$$

i. e. we need to derive the output vector only once to "see" the input. Now, assuming that we have perfect knowledge of x_2, x_3, x_4, all the parameters involved and the (state dependent) control coefficient $b(x_3, u)$ it is straightforward that

$$\begin{aligned} u &= \frac{1}{b(x_3, u)}\big(-f(\boldsymbol{x}) + v\big) \\ v &= \dot{x}_{3,d} - k_p \cdot (x_3 - x_{3,\mathrm{d}}) \end{aligned} \tag{IOL}$$

will lead to direct action of the new input v on $\dot{x}_3$ (as long as we do not saturate u). For v we could then set up a for example a simple P–controller. A strictly positive controller gain $k_p > 0$ will result in an exponentially decaying tracking error

$$e := x_3 - x_{3,\mathrm{d}} \tag{3.2}$$

This can easily be seen by inserting (IOL) into (3.1). We then have

$$\dot{e} + k_p e = 0$$

which represents asymptotically stable error dynamics. Therefore, if initially $e(0) = 0$ and $\dot{e}(0) = 0$, perfect tracking is achieved (as then $e(t) \equiv 0$ for $t \geq 0$).

If the initial error is non zero, it will decay exponentially to zero — the higher k_p, the faster the convergence and the better the tracking (but usually the higher the actuator costs also).

As simply and elegant this approach is, it only works in the ideal case where on the one hand we have a perfect model together with *perfectly correct* parameters, and on the other hand *exact* measurements of three of the four the state variables. As this rather utopic precondition will certainly not be met in a real application we must introduce some robustness against uncertainties and imprecision.

3.3 Sliding mode control

In light of the above limitations, it is imperative to design a more robust controller. A classical approach would be a sliding mode controller.

3.3.1 Basic set–up

Focusing on (2.9c) on page 13, one usually can have a guess of the $f(\boldsymbol{x})$ term, let's call it $\hat{f}(\boldsymbol{x})$. Additionally, one should be able to bound the estimation error by some $F > 0$, that is

$$\left| f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) \right| \leq F \tag{3.3}$$

One then introduces the so called sliding surface s, and sets, for instance,

$$s(\boldsymbol{x}, t) = e \tag{3.4}$$

again e being the tracking error (3.2). The next step in the classical sliding mode controller design would be to demand $\dot{s} = 0$, from which follows

$$\dot{s} = \dot{e} = \dot{x}_3 - \dot{x}_{3,\mathrm{d}} = f(\boldsymbol{x}) + b(x_3, u)\, u - \dot{x}_{3,\mathrm{d}} \stackrel{!}{=} 0$$

The best approximation of the continuous control law that can achieve $\dot{s} = 0$ would be

$$\hat{u} = -\hat{f}(\boldsymbol{x}) + \dot{x}_{3,\mathrm{d}}$$

In order to guarantee the sliding condition without exactly knowing $f(\boldsymbol{x})$ one adds to $\hat{u}$ a term that is discontinuous across the surface $s = 0$, such as

$$\begin{aligned} u &= \frac{1}{b(x_3,u)}\left[\hat{u} - k_s \operatorname{sign}(e)\right] \\ k_s &= F + \eta \end{aligned} \tag{3.5}$$

where $\operatorname{sign}(e)$ is the signum function. With this set–up one can easily show (see for example [8]) that

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}e^2 \leq -\eta|e| \tag{3.6}$$

which corresponds to a decreasing tracking error along all system trajectories.

It is clear that if we actually want to implement the controller, we need to derive a suitable and correct F.

3.3.2 Bounding the uncertainties

We shall now determine F. This bound does not have to be constant — to the contrary, if possible, F should depend on the state vector $\boldsymbol{x}$ to give us a tighter bound on the estimation error.

We suppose that for each parameter in the system we can estimate or calculate a nominal or mean value p. The real value of the parameter must lay within a certain radius, that is we demand $p - \Delta p \leq p \leq p + \Delta p$, where Δp is the maximum deviation allowed from the estimated value p. With this notation we can write

$$\begin{aligned} F(\boldsymbol{x}) &\leq \left| \hat{f}(\boldsymbol{x}) - f(\boldsymbol{x}) \right| \\ &\leq \left| f(\boldsymbol{x}, p_1 + \Delta p_1, \ldots, p_3 + \Delta p_3) \quad - \quad f(\boldsymbol{x}, p_1, \ldots, p_3) \right| \end{aligned}$$

where $p_i = a_2, a_3, K_2$. A classical approach would be to use a first–order Taylor series expansion of $f(\boldsymbol{x}, p_1 + \Delta p_1, \ldots, p_3 + \Delta p_3)$ around the p_i

$$\begin{aligned} F(\boldsymbol{x}) &\leq \left| f(\boldsymbol{x}, p_i) + \sum_{i=1}^{3} \left| \frac{\partial f}{\partial p_i} \right| \Delta p_i \quad - \quad f(\boldsymbol{x}, p_i) \right| \\ &\leq \left| \sum_{i=1}^{3} \left| \frac{\partial f}{\partial p_i} \right| \Delta p_i \right| \end{aligned}$$

to be able to bound $F(\boldsymbol{x})$. Thanks to the multi–linear nature of $f(\boldsymbol{x})$, we find

$$F(\boldsymbol{x}) \leq \left| |x_2|\Delta a_2 \; + \; (|x_3| + |x_4|)\Delta a_3 + \Delta K_2 \right| \tag{3.7}$$

The maximum deviations for the different coefficients and terms are combinations of other "elementary" parameters, for each of which we can bound our amount of uncertainty. Using the laws of error propagation, which boil down to a similar Taylor series expansion approach (see for example [9]), we can then

determine Δa_2, Δa_3 and ΔK_2:

$$\begin{aligned}\Delta a_2 &\leq \left|\frac{\mathrm{d}a_2}{\mathrm{d}E}\right| \Delta E + \left|\frac{\mathrm{d}a_2}{\mathrm{d}A_\mathrm{z}}\right| \Delta A_\mathrm{z} + \left|\frac{\mathrm{d}a_2}{\mathrm{d}V_{\mathrm{z}0}}\right| \Delta V_\mathrm{z} \\ &\leq \frac{A_\mathrm{z}}{V_{\mathrm{z}0}} \Delta E + \frac{E}{V_{\mathrm{z}0}} \Delta A_\mathrm{z} + \left|-\frac{EA_\mathrm{z}}{V_{\mathrm{z}0}^2}\right| \Delta V_{\mathrm{z}0}\end{aligned}$$

$$\Delta a_3 \leq \frac{1}{R_\mathrm{d} V_{\mathrm{z}0}} \Delta E + \left|-\frac{E}{R_\mathrm{d}^2 V_{\mathrm{z}0}}\right| \Delta R_\mathrm{d} + \left|-\frac{E}{R_\mathrm{d} V_{\mathrm{z}0}^2}\right| \Delta V_{\mathrm{z}0}$$

$$\Delta K_2 \leq \frac{Q_\mathrm{l}}{V_{\mathrm{z}0}} \Delta E + \frac{E}{V_{\mathrm{z}0}} \Delta Q_\mathrm{l} + \left|-\frac{EQ_\mathrm{l}}{V_{\mathrm{z}0}^2}\right| \Delta V_{\mathrm{z}0}$$

This done, we should note that we assumed a perfectly correct inverse valve model. We are safe to do so, as any resulting deviations from it should also be compensated by the robustness of the controller. However an important aspect, that will almost always occur in real applications, needs to be addressed before moving on.

3.3.3 Reducing chattering

In real systems, the switching resulting from a change of sign of s usually cannot be done indefinitely fast (as required). This results in the problem called chattering, where the controller output seems to be "quivering" about some value. Mechanical parts of the system could be damaged by such behaviour or significant structural vibrations can occur. Also, unwanted high frequency dynamics may be excited, which could lead to instability of the system. In any case, the controller will not function properly, not at all, or at least not in a satisfactory and efficient way.

To reduce this problem, one possible approach would be to use a saturation function instead of the signum function, i. e. introduce a finite slope around the origin, that way preventing the sudden and abrupt switching.

This translates into a certain boundary layer around the tracking signal, that is a "tunnel" within which the real output is kept by the control actions. The downside of this is a reduced tracking precision, but this disadvantage is made up for by the benefit of a significantly reduced and also controllable chattering. Usually a good trade off between precision and general performance of the controller can be achieved.

For those reasons, it is recommendable to replace $\mathrm{sign}(e)$ in the control law (3.5) on the previous page with $\mathrm{sat}(e/\varepsilon)$, where ε is the tolerated deviation from the reference signal (also called boundary layer width) and

$$\mathrm{sat}(x) := \begin{cases} x & \text{if } |x| \leq 1 \\ \mathrm{sign}(x) & \text{otherwise} \end{cases}$$

So, the new control law now reads

$$u = \frac{1}{b(x_3, u)} \left[\hat{u} - k_s \,\mathrm{sat}\left(\frac{e}{\varepsilon}\right)\right] \tag{SM1}$$

3.3.4 Extension

An important — and in our circumstance necessary — extension to the sliding mode controller (SM1) would be to include integral action, a so–called I–component. This will allow for compensation of a drifting K_2, or a non–zero current neutral position of the valve (due to mechanical problems one may need to apply a constant but non–zero current to the valve to block any flow of oil).

We suggest two possibilities for integral action, the first simply adding an integral component directly to the control law

$$u = \frac{1}{b(x_3, u)} \left[\hat{u} - k_\mathrm{s} F(\boldsymbol{x}) \operatorname{sat}\left(\frac{e}{\varepsilon}\right) + k_\mathrm{i} \int_0^t e \mathrm{d}\tau \right] \tag{SM2}$$

the second including an integral part in the sliding surface, as suggested in [8]: $s = e + \lambda \int_0^t e \, \mathrm{d}\tau$, for $\lambda > 0$. This introduces an additional term $-\lambda e$ into the controller equation, which would then be

$$\begin{aligned} u &= \frac{1}{b(x_3, u)} \left[\hat{u} - \lambda e - k'_\mathrm{s} F(\boldsymbol{x}) \operatorname{sat}\left(\frac{s}{\varepsilon}\right) \right] \\ s &= e + \lambda \int_0^t e \, \mathrm{d}\tau \end{aligned} \tag{SM3}$$

Note all the integrators in the above discussion should be set to zero when the controller is turned on (we used a little reset signal connected to all integrators in the Simulink model which sends an impulse when the controller is turned on).

3.3.5 Potential problems

The actual implementation of the controllers however may involve further problems. For instance, until now, we have considered a *continuous* plant and controller. In reality, the plant of course is also continuous, but as we use digital equipment in the car, the sensors data is *sampled* data. So the controller only gets "snapshots" of the system's state every $T_\mathrm{s} = 5$ ms.

In a way, this can be interpreted as a dead–time (of the order of magnitude of the systems sampling rate T_s) introduced to the system, and this is not and cannot easily be dealt with directly using these types of controllers.

Also, the in some cases extremely large controller gain resulting from F (its order of magnitude is roughly $10^8 \ldots 10^{10}$!) can cause the system to become unstable, or at least behave very "chattery".

A simple but rather crude and unsophisticated solution to this problem would be to just manually fix a relatively small and constant F. We chose, however to use some scaling factors k_s and k'_s to attenuate the effect of the large $F(\boldsymbol{x})$.

With these considerations we close our discussion on the design of a sliding mode based controller and move on to a different control concept.

3.4 Adaptive control

A very good introduction to this discipline of control theory can be found in the exhaustive book by Krstić *et al.*, [4]. Our design procedure roughly follows one of their suggestions.

Although we will hardly go into any details, the reader should be familiar with basic stability theory of non–linear systems, especially Lyapunov functions. If not, he may be referred to the excellent book by H. K. Khalil, [3].

Similar to the previous two controller types, we only focus on the dynamic equation for the cylinder pressure, (2.9c) on page 13, although we now consider the first three state dependent terms to be known.

3.4.1 Basic set–up

Let's recall (2.9c) and transform it into

$$\dot{x}_3 = a(t) - \vartheta + b(x_3, u)\, u \tag{3.8}$$

where $a(t) := a_2 x_2 - a_3 x_3 + a_3 x_4$ is supposed to be known and ϑ is the unknown parameter we would like to have adapted (corresponding to K_2).

We will start by introducing a Lyapunov function that guarantees (asymptotic) stability and convergence toward zero of both tracking and parameter error. We then use its derivative to determine suitable control and parameter adaptation laws:

$$V = \frac{1}{2} e^2 + \frac{1}{2\gamma_\vartheta} e_\vartheta^2 \tag{3.9}$$

with, again, $e := x_3 - x_{3,\mathrm{d}}$ being the tracking error, e_ϑ the parameter error $e_\vartheta := \vartheta - \hat{\vartheta}$ and $\gamma_\vartheta > 0$ the so–called adaptation gain (as we shall see later).

This Lyapunov function could intuitively be interpreted as a combined measure of error in both tracking and parameter adaptation. Clearly, we want it to converge toward zero, corresponding to perfect tracking and parameter estimation, or, at least, have it never increase.

A sufficient condition for that would be that its derivative with respect to the time t has to be be non–positive, i. e.

$$\dot{V} \overset{!}{\leq} 0 \tag{3.10}$$

We can now use this condition to derive the controller structure and parameter adaptation laws. Differentiating (3.9) with respect to the time t, we get

$$\dot{V} = \dot{e}\, e + \frac{1}{\gamma_\vartheta} \dot{e}_\vartheta\, e_\vartheta$$

There, $\dot{e}_\vartheta = \dot{\vartheta} - \dot{\hat{\vartheta}} \simeq -\dot{\hat{\vartheta}}$ when we assume that the unknown parameter ϑ is quasi constant. Replacing $\dot{x}_3$ in $\dot{e} = \dot{x}_3 - \dot{x}_{3,\mathrm{d}}$ by (3.8), we find

$$\dot{V} = \Big[a(t) \underbrace{-\vartheta + \hat{\vartheta}}_{=-e_\vartheta} - \hat{\vartheta} + b(x_3, u) u \;-\; \dot{x}_{3,\mathrm{d}} \Big]\, e - \frac{1}{\gamma_\vartheta} \dot{\hat{\vartheta}}\, e_\vartheta$$

As indicated above, adding and removing $\hat{\vartheta}$ allows us to rearrange the equation, and factorising by e_{ϑ} leads us to

$$\dot{V} = \left[a(t) - \hat{\vartheta} + b(x_3, u)u \; - \; \dot{x}_{3,\mathrm{d}}\right] e - \left(e + \frac{1}{\gamma_{\vartheta}}\dot{\hat{\vartheta}}\right) e_{\vartheta} \tag{3.11}$$

To meet our request (3.10) we could ask for two things

(i) the first term in (3.11) should be strictly negative,

(ii) the second term should be zero.

Letting

$$u = \frac{1}{b(x_3, u)}\left[\dot{x}_{3,\mathrm{d}} - a(t) + \hat{\vartheta} - k_{\mathrm{a}} \cdot e\right] \tag{ADa}$$

with $k_{\mathrm{a}} > 0$ we have gained a suitable control law that would fulfill (i), as long as $e \neq 0$ (otherwise perfect tracking is already achieved).

Concerning (ii), as e_{ϑ} is unknown, we have no other choice but to require the term in parentheses to disappear, resulting in

$$\dot{\hat{\vartheta}} = -\gamma_{\vartheta} e \tag{ADb}$$

with $\gamma_{\vartheta} > 0$ being the adaptation gain. This equation represents the parameter update law for $\hat{\vartheta}$.

It can easily been seen that with these two laws we are indeed able to meet (3.10), as we now have

$$\dot{V} = -k_{\mathrm{a}}\, e^2 \leq 0$$

The inherent integral action of this controller should also allow for sufficient compensation of any drifting oil leak flow constant or the like.

3.4.2 Potential problems

A number of important comments should be made at this point.

Initial conditions

In the above controller, we see a new state (that is $\hat{\vartheta}$) appearing. In simulation and real application, we need to set some sensible initial conditions for it, as well as for the reference signal.

For reasons of tracking performance and transient behaviour it is important to chose these initial conditions wisely. The unknown parameter should obviously be chosen close to its real value; the reference signal however should be set as close to the *current* or initial state of the system as possible. Otherwise, a very large control action might ponder on the system right when the controller is turned on (this is also the case for the sliding mode controllers!).

Once an initial transient is overcome and tracking works well, the reference can then be taken to where the system is to go.

Saturation

Large control action could lead to saturation of the actuator, as we have mentioned earlier. In that case, the integrators in the controllers would continue to build up to higher and higher values, which result in large control actions even if the tracking error has long decreased again.

So it is recommendable to include some mechanism in the system that would, for example, hold these integrators in case of saturation. These are so–called anti–wind–up mechanisms. Another way would be to not simply feed e into the integrators, but instead use $(e - k_{\text{aw}} \cdot [I - \text{sat}(I)])$ with some suitable $k > 0$. Whenever saturation of an actuator occurs, the integrators input would be reduced by some amount proportional to the amount of saturation.

Such a mechanism is indispensable for instance when PEGaSOS' engine is running at idle speed: Then, the pressure generated by the pump usually drops below what is necessary for holding the car's weight, and the car begins to sink slowly (usually just one wheel is "sacking" in). In that corner, the respective controller with integral action will soon open the valve to the maximum (i. e. I saturates), but there is nothing the valve nor the cylinder can do, as there is insufficient system pressure.

During that time, the integrators are accumulating more and more tracking error. When the driver then revs up the engine again, and enough pressure is provided to hold the car, the controller keeps saturating the valves, even though the cylinder has already "caught up".

In that case, without an anti wind–up mechanism, the piston could keep moving until it hits the physical end of the cylinder, possibly causing damage to the suspension strut!

Malfunctioning sensors

As mentioned earlier, an actual problem I was having in testing these controllers was, that (to everyones surprise) the p_{s} sensors were not working. To overcome this problem, Dr. Akar and I developed sliding mode controllers that do not require those measurements. As these are ongoing topics of research, and potential subjects of publications, we decided for now not to give away any details on these.

With these remarks we close this chapter on controller design; the controllers created above shall now be tested in simulation.

CHAPTER 4

Simulations

In this last chapter we set the parameters of the different controllers developed earlier, and use our simulation to test their performance, especially subjecting them to various parameter disturbances.

4.1 Introduction

We tested the controllers in different situations such as modifying the valve's response to a certain control current and other parameter changes. In various plots we shall compare and comment on the reactions and performance of each controller.

It is very important to run these tests, as on the one hand we only have a rather simplified model of the system, and on the other hand we either do not know the correct parameters, or the parameters might drift (for instance, the properties of the hydraulic spring strongly depend on the temperature).

Again, to everyone's surprise we were lacking reliable p_s measurements, which unfortunately kept me from actually testing the controllers in the car. This would certainly have been an integral part of this report. For that reason, this chapter only presents results from simulations.

The good news is, however, that the malfunctioning sensors will be taken care of in the next overhaul of the car. Besides, controllers have been developed that can do without this information.

To facilitate readability throughout the chapter, we will use the following abbreviations for the different controllers:

- **SM2**: Sliding mode controller (SM2) on page 23
- **SM3**: Sliding mode controller (SM3) on page 23
- **AD**: Adaptive controller (ADa) on page 25
- **EPC**: Existing PID–controller (EPC) on page 6

4.2 Set–up of the tests

Before looking at the results of the simulations, we shall describe the testing conditions, and also list our choice of parameters for three of our controllers.

4.2.1 Test conditions and disturbances

It is certainly interesting to see how the controllers work under nominal conditions, i. e. when each parameter in the plant really has the value we assumed it to have (of course, this is only possible in simulations).

But with regard to a real application of the controller, it is also necessary to check what effect various types of parameter changes have; this is testing the robustness of the controllers, for which they have been designed.

There are several disturbances we inflicted on the system. We change valve parameters, such as adding a constant[1] current offset to I or changing the valve gain. We also manipulated some plant parameters like a_2, a_3 and $\bar{\kappa}$.

In all the simulation runs we used friction model (iv).

4.2.2 Controller parameters

Certainly, our particular choice of parameters is a preliminary one, and should be refined especially in conjunction with real measurements.

However, as proper experiments were not possible, only limited time has been spent on fine tuning the controllers here, so there is certainly room for further improvements, especially when tuning can be done in the car.

So here is the list of parameters chosen.

- **SM2**: $k_\mathrm{s} = 0.5, \quad k_\mathrm{i} = 50000$
- **SM3**: $k'_\mathrm{s} = 0.5, \quad \lambda = 5$
- **AD**: $k_\mathrm{a} = 8000, \quad \theta(0) = 0, \quad \gamma_\theta = 10000$

We left out the η parameter introduced in the design of the sliding mode controllers, as its value is negligible before the large values of F (or already there "included").

Let's now take a look at the results from our simulations.

4.3 Nominal conditions

In this section we present a test of the performance of the controllers under nominal conditions, meaning that controller and plant use exactly the same parameter values (for instance, the inverse valve model is perfectly the inverse of the valve in the plant, or $f(x)$ does perfectly cancel the corresponding parts in the plant).

The relevant plots from our simulation are shown in Figure 4.1 on the next page. We can see that when EPC is used there is again the noticeable phase lag between reference and actual value, as already observer in the measurement data at the end of Chapter 1. Also, the amplitude of the oscillations is not matched very well, and there is a (only very slowly decreasing) pressure offset for the constant parts. However, the impulses are followed relatively good, with only little overshoot.

[1] Actually, not a *constant* offset was added from $t = 0$ on, but a ramp offset current, saturated at ± 50 mA, was used, so that the offset starts at zero, but reaches ± 50 mA within about 0.3 s. This generates a smoother transition.

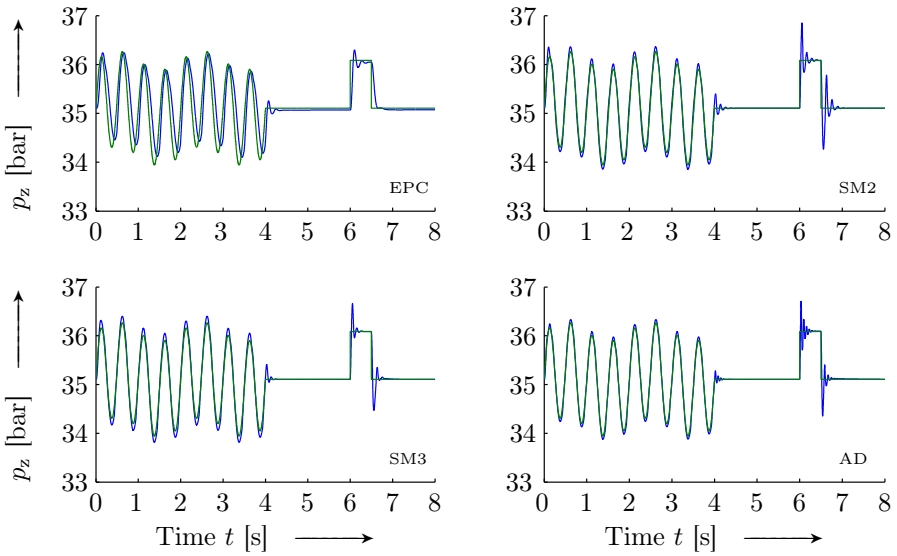

Figure 4.1: *Comparison of simulated performance of each controller, looking at reference pressures $p_{z,d}$ (——) and actual pressures p_z (——) for each controller (using friction model (iv), nominal conditions).*

Looking at SM2, we see the phase lag disappear. Additionally, the amplitude of the sinusoidal part is tracked much better. However, there is a much more pronounced overshoot.

Similar comments hold for SM3, with the difference there here the overshoot at the step changes is slightly less pronounced.

The adaptive controller AD follows the sine part best, but shows a fast and only lightly damped oscillating overshoot at the impulse.

All three new controllers however show no constant error offset, thanks to their I–components, and also feature much better tracking characteristics. The cost of that performance on the downside are stronger overshoots.

But how do the controllers react to plant parameter changes?

4.4 Parameter perturbations

As mentioned earlier, it is more interesting and important to investigate the impact of parameter changes on the controllers performance.

In Figure 4.2 on the following page we added an offset of +50 mA at valve level in the plant. We can see that the integral part of the EPC is not strong enough and the resulting constant tracking error is only slowly decreasing.

Each of the new controllers to the contrary can compensate rather quickly for the offset. Once it is taken care of (this takes less than a second), their performance is similar to that of the nominal case.

Another perturbation tested here is a changed valve gain. Surprisingly, even a large change of +20% does not seem to have a strong impact on any of the controllers, see Figure 4.3 on the next page.

We also ran tests with various other perturbations, but as the biggest concern was regarding the precision of the valve model, we moved those results to Appendix C on page 47.

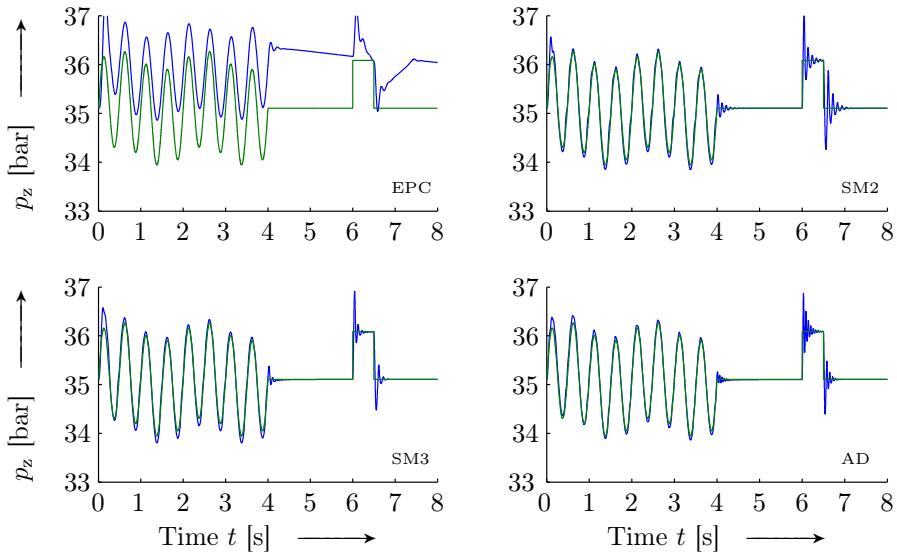

Figure 4.2: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance: +50 mA offset in* I*.*

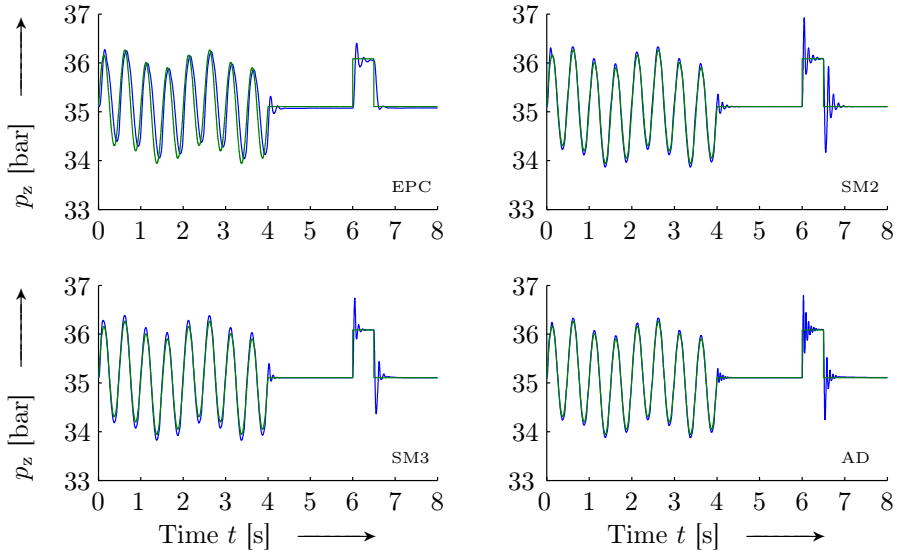

Figure 4.3: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance:* k_v *changed by +20%.*

With the successful tests of three of our non–linear controllers we shall now close this last chapter and move on to the conclusion of this report.

Conclusion

Let us close this report with a few concluding remarks on what has been achieved sofar.

In the first Chapter, we introduced the active suspension and the set–up of its different components. The "bigger picture" has been briefly explained, showing the structure of the control system as well as discussing the function of each part in the outer loop. We then introduced the existing PI controller in the inner loop and showed some real measurements of its performance.

The following chapter dealt with getting a model of the system. After modelling each component of the AHP one by one, we joined them up and established a model of the system as a whole. We then validated the model briefly, showing that on the one hand our complete model allows for a reasonably good fit of the experimental data collected, but also, on the other hand, that the subsystem, describing just the pressures in cylinder and capacitor seems to give good and consistent results as well.

Mainly focusing on the dynamic equation for p_z, Chapter 3 was concerned with deriving suitable controllers for the cylinder pressure (and hence the resulting force exerted by the cylinder). Here, we designed an input–output linearising controller, several sliding mode controllers and a Lyapunov function based adaptive controller. Particular importance was attributed to including integral action in the controllers.

Unfortunately it was not possible to test the controllers properly due to the malfunctioning differential pressure sensors which are necessary to determine p_s. For that reason, the last chapter showed only results from simulations.[2] We were able to get good results (and certainly better results compared to the existing controller), especially with changed valve parameters.

Of course it would be interesting to also evaluate the performance of the other controllers in the car. Also, a next step would be to evaluate our force controllers together with the outer loop controllers, for example by again testing PEGaSOS' ability to imitate other car's dynamic behaviours — one of the ultimate goals of the CEMaCS project. A professional tuning of the controllers on the test track would be the first step in this direction.

However, these open points are being addressed right now in the continuing work in Böblingen and will certainly yield to interesting new results, that may very well be published in the near future.

[2] However, controllers that do not need of knowledge of p_s were developed and successfully tested in the car (for reasons of confidentiality they are not discussed in this report).

APPENDIX A

Simulink models

The following models have been moved to the Appendix in order no to clutter up the body of the document.

A.1 Simulation

The main SIMULINK model is shown in Figure A.1 on the next page. Let us comment on some of its "ingredients".

A.1.1 Main model

As our system model (2.9) on page 13 is a (non–linear, non–autonomous) four–dimensional first–order system of differential equations, the rough structure of the SIMULINK model is clear. In the center, there are four integrator blocks, with appropriate initial conditions (they are set externally for a more concise layout).

At the left hand side of the model we have a number of sources corresponding to external signals, disturbances or simple constants.

Three other, larger blocks contain subsystems, namely the different friction models (cf. Subsection 2.2.5 on page 11), the servo–valve and the controller (more precisely the controllers, with a convenient way of switching between them).

Clearly, the model of the plant is a continuous model; in the car however, as the sensor equipment is digital, the values are sampled (with $T_s = 5$ ms). We imitate this setting by sampling the (continuous) system output with T_s.

The input to the controller is some artificial reference signal generated by the two signal generators. To be closer to the real situation in the car, we also sample the controller output, as it may be continuous (the adaptive controller for instance introduces a continuous signal). This sampled signal is fed into our valve model, and the resulting oil flow then acts on the plant. As noted earlier, the controller block also contains the inverse valve model.

A.1.2 Valve model

The valve model shown in Figure A.2 on page 35 is quite straightforward, it incorporates a saturation and the delay term suggested in Subsection 2.2.4 on page 10. If for some reason the cylinder pressure should be higher than the pump pressure (or lower than the reservoir pressure), the signum blocks come into play — they allow for correct (and, together with the absolute value blocks, mathematically sound) behaviour in this situation.

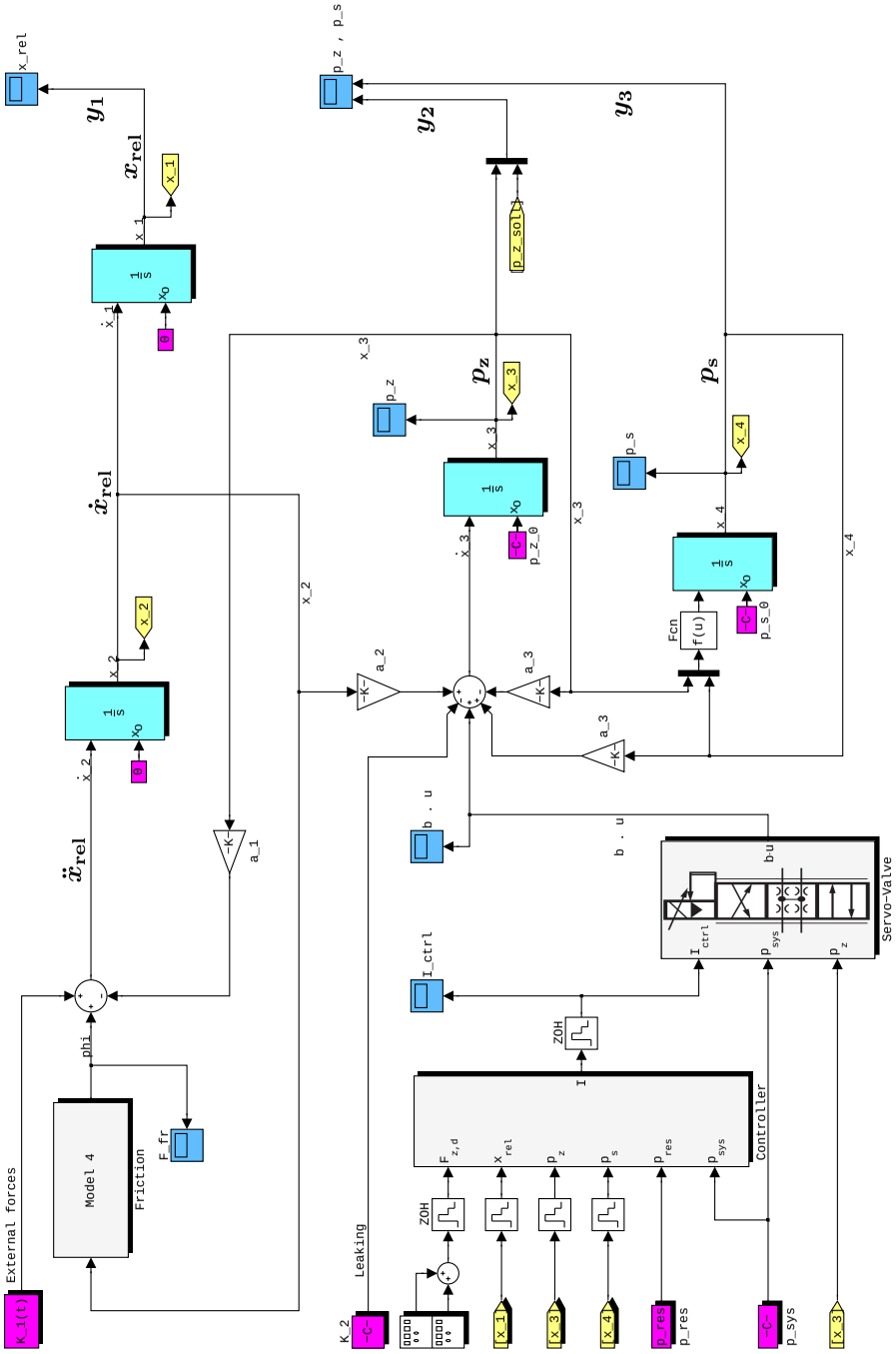

Figure A.1: *Main simulation* SIMULINK *model.*

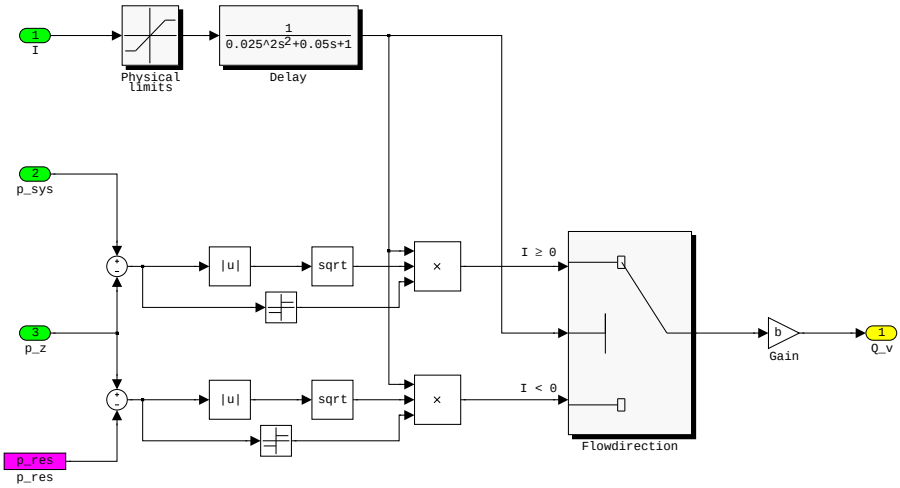

Figure A.2: *Valve block subsystem.*

A.2 Car software

For reasons of completeness we also show the SIMULINK model which is running on the car computer (more precisely, the model which is then compiled via SIMULINK's real–time workshop and then transferred to the car).

Confidentiality dictates however that we must not give away any further details than a top level view of the model, shown in Figure A.3 below.

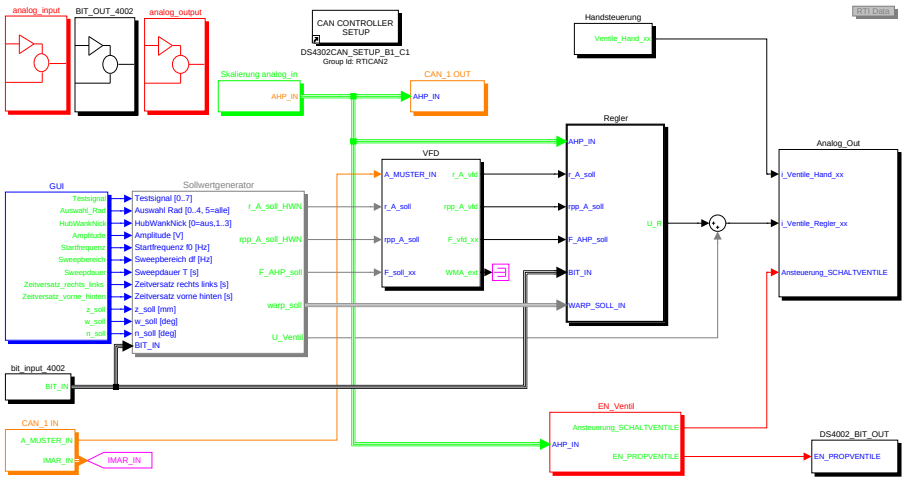

Figure A.3: SIMULINK *model of the car software.*

A.3 Estimation and Validation

Together with `para_ident`, listed on page 41, we used the SIMULINK model shown in Figure A.4 below not only to simulate the system and compare measured with real output, but also to let MATLAB automatically estimate any parameter wanted.

The m–file `convert_data` is needed to convert the measurements into a format suitable for this model and technique.

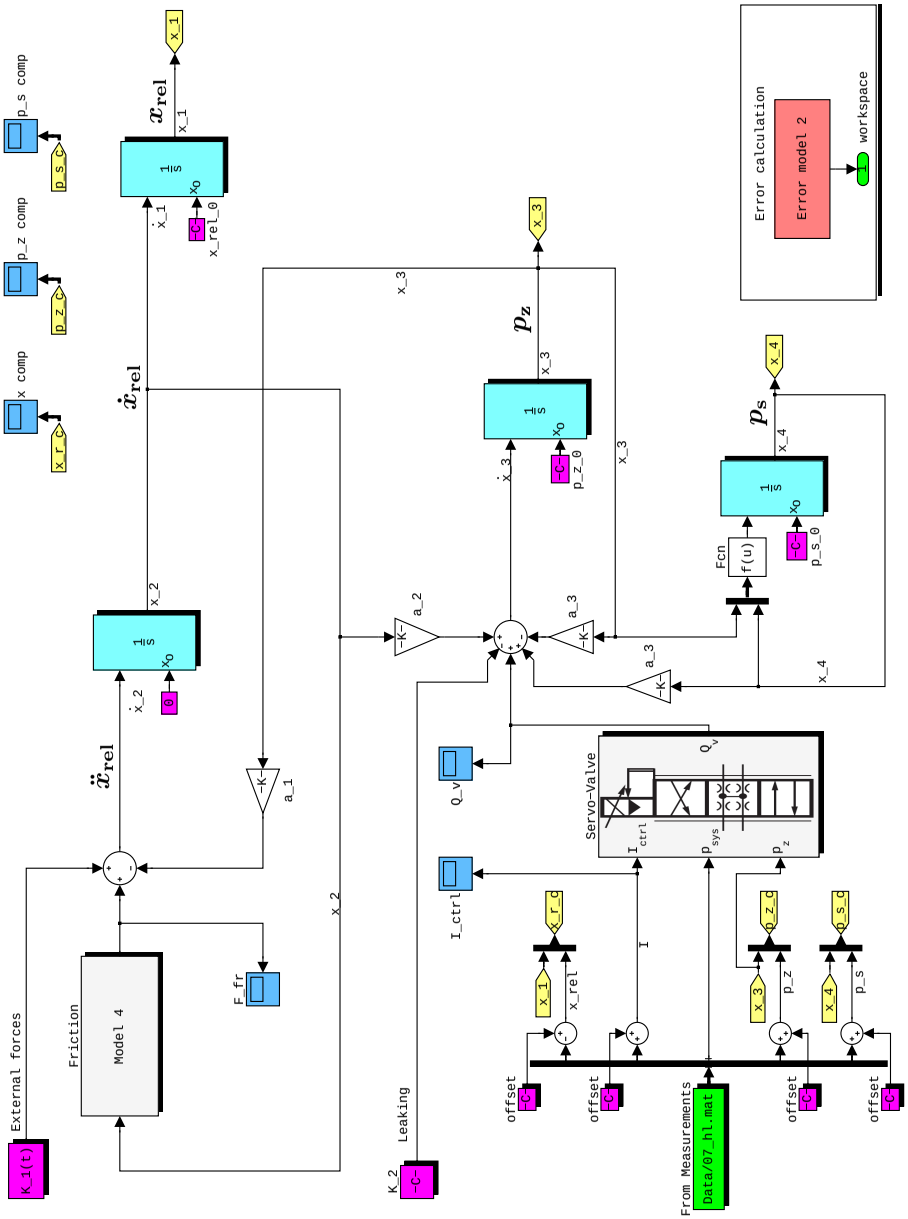

Figure A.4: `iotest.mdl` *used for parameter estimation and validation.*

APPENDIX B

Source codes & other resources

We continue with a listing of some of the m–files which have been written in the course of this internship, as well as a "cookbook" which gives a number of step–by–step guides for the work with PEGaSOS.

B.1 Data treatment

The following function was written to convert data recorded in the car (using the ControlDesk software by dSPACE GmbH) to a format that is compatible with `iotest.mdl`.

Listing B.1: `convert_data.m`

```
function convert_data( filename , corner )
%CONVERT_DATA  Convert ControlDesk .mat files for iotest.mdl
%
%   CONVERT_DATA('filename') extracts measurements from
%    filename.mat and stores the relevant signals in a file for
%    usage in iotest.mdl.
%
%    You must pass at least the filename (without the .mat), but
%    you may also specify which corner you want the data to be
%    extracted for:
%      1 – front left  (default)
%      2 – front right
%      3 – rear left
%      4 – rear right
%
%    Example: CONVERT_DATA('test1',3);
%
%    The orginial .mat file should come from measurements taken
%    in dSPACE ControlDesk.  If you rename things in the
%    accompanying kraft_regler.mdl, you must edit the eval–block
%    below so that the correct signals are fetched!

%    Last change: Florian Knorn, 03/21/2006, 13:50
%    florian@knorn.org

if nargin < 1 || nargin > 2
    error('Please give the name (without .mat) of datafile');
end

if ¬ischar(filename)
    error('Please give the name (without .mat) of datafile');
end

```

```
34 if nargin == 1
35     corner = 1;
36 end
37
38 if corner < 1 || corner > 4 || (corner-ceil(corner)) ~= 0
39     error('Something''s wrong with the corner ...');
40 end
41
42 % sometimes it's "1", sometimes it's "vl" . . .
43 cornerstr = {'vl','vr','hl','hr'};
44 cornerstr = cornerstr{corner};
45
46 % explode the struct into single variables
47 eval(['explode_struct(''',filename,''');']);
48 eval(['load ',filename,'_temp;']);
49
50 % create empty empty matrix
51 all=[];
52
53 % edit names here !!
54 eval(['all = [X;',...                                          % Time
55  'New_controller__x_rel__Out1_1_',num2str(corner),';',...% x_rel
56  'Labels__i_PropV_',cornerstr,';',...                      % I
57  'New_controller__p_sys__Out1 ;',...                       % p_sys
58  'New_controller__p_z__Out1_1_',num2str(corner),';',...  % p_z
59  'New_controller__p_s__Out1_1_',num2str(corner),'];']);  % p_s
60
61
62 % save variables
63 eval(['save ',filename,'_',cornerstr,' all']);
64
65 % remove temporary file
66 eval(['!del ',filename,'_temp.mat;']);
67
68 % finish
69 disp( sprintf('Time + %i variables saved to %s.',...
70     length(all(:,1))-1,[filename,'_',cornerstr]) );
```

The data conversion function uses a subroutine called `explode_struct`. When data is recorded in the car, ControlDesk saves everything into one large structure. This structure is hard to handle, `explode_struct` "explodes" this large structure and saves all the recorded signals into separate variables.

Listing B.2: `explode_struct.m`

```
1  function explode_struct(file_in, file_out)
2  %EXPLODE_STRUCT  Convert struct to single variables
3  %
4  %   EXPLODE_STRUCT('file_in','file_out')  loads file_in.mat and
5  %    "unpacks" all X and Y variables from any struct init to
6  %    single variables and stores them into  file_out.
7  %
8  %    You may also omit the second argument, by default
9  %    'file_in_temp' will then be used as output file name.
10
11 %    Last change: Florian Knorn, 03/14/2006, 11:50
12 %    florian@knorn.org
13
14 % error checking
```

```
15 if nargin < 1 || nargin > 2
16     error('Please give at least an input filename');
17 end
18 if nargin == 1
19     if ¬ischar(file_in)
20         error('The input filename should be a string');
21     else
22         file_out = [file_in,'_temp'];
23     end
24 else
25     if ¬ischar(file_out)
26         error('The output filename should be a string');
27     end
28 end
29
30 % That done, let's load the file
31 eval(['load ',file_in]);
32
33 % Figure out the names of all the structs in the file
34 wers = whos; % struct of all the variables in the file
35 structs={};
36 for i=1:length(wers) % go through all varibales
37     if strmatch(wers(i).class,'struct') % if struct
38         structs{length(structs)+1} = wers(i).name; % store name
39     end
40 end
41
42 % Now, recursively go through each struct found in the file
43 for cs_i = 1:length(structs) % current struct index
44     eval(['cs = ',structs{cs_i},';']); % current struct
45
46     % store x-data
47     X = cs.X.Data;
48
49     % store y-data
50     for i = 1:length(cs.Y)
51         varname = cs.Y(i).Name; % get variable name
52
53         % the varible names represent some sort of hiearchy,
54         % correspoding to the blocks in simulink.  these levels
55         % are seperated with slashes (/).  using a reg. expr.,
56         % try to fetch only the lowest 3 levels (otherwise the
57         % names may be too long for matlab).
58         [von,bis] = regexp(varname,'[^/]+/[^/]+/[^/]+$');
59
60         % if no match has been found, only look for lowest 2
61         if isempty(von)
62             [von,bis] = regexp(varname,'[^/]+/[^/]+$');
63         end
64
65         % make the resulting string suitable for a matlab
66         % variable name (remove slashes etc.).  note that
67         % cleanup(something) is defined below!
68         varname = cleanup(varname(von(end):bis(end)));
69
70         % now assign the variable its value
71         %disp([varname,'= cs.Y(',num2str(i),').Data;']);
72         eval([varname,'= cs.Y(',num2str(i),').Data;']);
73     end % for each variable in Y
74
75     eval(['clear ',structs{cs_i},';']); % clear curr. struct
76
```

```
77 end % looping through different structs in file
78
79 % clear all "temporary" variables so they don't get saved
80 clear varname i wers file_in von bis structs cs cs_i;
81
82 % save the rest ;-)
83 eval(['save ',file_out]);
84 disp(sprintf('All varibles saved to file ''%s''.',file_out));
85
86 % ------------------------------------------------------------------
87 % cleans up & shorten messy names for use as variable names
88 function clean = cleanup(messy)
89 cleaner = strrep(strrep(messy,',','_'),'\n','');
90 cleaner = strrep(strrep(cleaner,'[','_'),']','');
91 cleaner = strrep(strrep(cleaner,' ','_'),'/','__');
92 cleaner = strrep(cleaner,'"','');
93 cleaner = strrep(cleaner,':','');
94 cleaner = strrep(cleaner,'=','');
95 cleaner = strrep(cleaner,'-','');
96 cleaner = strrep(cleaner,'Beschleunigungsaufschaltung',...
97     'Beschl_Auschf');
98 clean = strrep(cleaner,'Verpannungsregelung','Versp_Reg');
```

B.2 Parameter estimation

The following two programs were written for the parameter estimation. Their usage is explained in the comment block at the top of `para_ident`.

Listing B.3: `para_ident.m`

```
% This function tries to estimate some parameters of the AHP
% model, using iotest.mdl and Simulink for the simulation—–
% and Matlab's fminsearch for the estimation part (in con–
% junction with some real measurements taken from the car).

% Usage:
%  1) Get some data from the car.
%  2) Use convert_data to create a file that can be loaded into
%     the iotest.mdl. You may have to check that convert_data
%     is looking for the correct signal names !
%  3) Copy the created file (ending on ..._hl for example) into
%     the ./Data folder and select this file in the
%     "From Measurements" block, do not close the simulink model.
%  4) Put the variable names you want to have estimated in the
%     cell called "identpool" below (line 36).
%  5) Choose an appropriate error model in iotest.mdl.
%  6) Make sure, you're at the top level of the simulink model,
%     then run this file.

%  #) To stop the process you may just close the error–evolution
%     window.
%
%     You may observe how things are going by using the scopes
%     at the top of the simulink model.

%    Last change: Florian Knorn, 04/01/2006, 9:11
%    florian@knorn.org

% startup – clean up workspace and reload variables.
clear all; clc; close all;
parameters;          % model parameters
parameters_car_ctrl; % car controller parameters
iotest_init;         % initialise iotest.mdl

% the following parameters will be estimated . . .
identpool = {'F_c2','F_m2','d_v2'};

% initial condidtions of the system
x_0 = [0 0 p_z_0 p_s_0];

% time span of simulation / identification
tspan = [3 10];

% parameters to estimate
parastr = [];
for i = 1:length(identpool)
    parastr = [parastr,' ',identpool{i}];
end

% create initial parameter values and store them
eval(['param0 = [',parastr,'];']); param1 = param0;

% prepare plotting
ph = plot(0,nan); set(gca,'YScale','log');
```

```
55 xlabel('Iteration'); ylabel('Error measure');
56 title('Evolution of estimation error');
57 set(gcf,'NumberTitle','off',...
58     'Name',get(get(gca,'title'),'String'));
59
60 % store all variables into struct "p"
61 vars = whos;
62 for i=1:length(vars)
63     eval(['p.',vars(i).name,' = ',vars(i).name,';']);
64 end
65 clear vars;
66
67 % call the optimizer
68 finalpara = fminsearch(@para_ident_costf,param0,[],p);
```

MATLAB's `fminsearch` is run on the cost function `para_ident_costf`, which, itself, launches a simulation run of `iotest.mdl` and calculates the estimation error. The optimiser then tries to modify the parameters as to lower the costs.

The costs here are chosen to be the integral (sum) over the squared deviations between measured and simulated value of either x_{rel}, p_{z}, p_{s}, or a combination of them, depending on the choice in the error model block, see Figure A.4 on page 36.

Listing B.4: `para_ident_cost.m`

```
1  function cost = para_ident_costf(est,p)
2  % Cost-Function. Used in conjunction with para_ident.m!
3
4  %    Last change: Florian Knorn, 03/02/2005, 14:00
5  %    florian@knorn.org
6
7  % get iteration number from plot ;-)
8  x = get(p.ph,'XData');
9  fprintf('\n\n___  Step %i  ___________________',x(end)+1);
10
11 % assign variables and display their values
12 for i=1:length(p.identpool)
13     assignin('base',p.identpool{i},est(i));
14     eval(['fprintf(''\n',p.identpool{i},...
15             ' = %+e'',est(',num2str(i),'));']);
16 end
17
18 % run simulation
19 [t,notused,y] = sim('iotest',p.tspan);
20
21 % store parameters temporarily
22 p.param1 = est;
23
24 % calculate costs
25 cost = sum(y.^2);
26
27 % display of error evolution
28 set(p.ph,'XData',[x,x(end)+1],...
29     'YData',[get(p.ph,'YData'),cost]);
```

B.3 Cookbook

The following PEGaSOS cookbook[1] has been written for other people that will get a chance to work with PEGaSOS and it's AHP.

C O O K B O O K F O R P E G A S O S A N D A H P

by Florian Knorn

florian@knorn.org

2006 / 03 / 26

Contents:

1. Abbreviations
2. Windows network settings
3. Matlab / Simulink
4. Car
5. dSPACE ControlDesk
6. General Stuff

[1] The name was suggested by Dr. van Tran.

```
------------------------------------------------------------------------
         1.  Abbreviations (for your information)
------------------------------------------------------------------------

AHP = Aktive Hydropneumatik
EPC = Existing PID Controller (Regler vom aktuellen Flash-Stand).
IDF = Intermediate Data File
PEGASOS = Prüfstand zur Entwicklung und Ganzheitlichen Simulation
      Optimierter Fahrzeugsystemdynamik
PPC = PowerPC
RTI = Real-time interface
RTW = Real-time Workshop


------------------------------------------------------------------------
         2.  Windows network settings (if necessary)
------------------------------------------------------------------------

1. go to Start -> Einstellungen -> Netzwerkverbindungen

2. right-click on Local Area Connection, choose Eigenschaften

3. Double-click on Internet Protocol (TCP/IP)

4. Enter manual ip adresses:
   IP-Adresse: 10.71.1.100,    Subnetzmaske: 255.255.255.0



------------------------------------------------------------------------
         3.  Matlab / Simulink
------------------------------------------------------------------------

STARTUP:
1. cd C:\Pegasos\aktuell

2. add to path (folders and subfolders) C:\Pegasos\aktuell\dspace_ahp

3. cd C:\Pegasos\aktuell\dspace_ahp\ahp

[4. Make sure RTI1005 is the current board. If not, execute "rti1005"]

5. run pegasos_startup to load everything; you may edit this script
   to suit your own needs, like have it load your need, additional
   parameters and such.

EDITING & COMPILING:
6. change whatever you want to change

7. to compile: cd ..\rtw

8. push Ctrl+B (in simulink), or go to Tool -> Real-Time Workshop ->
   -> Build Model [if you're not connected to the car, ignore the
   LOADING FAILED warning.]
```

```
------------------------------------------------------------------------
         4.  Car
------------------------------------------------------------------------

1. Turn on engine

2. Turn on PPC (key)

3. Turn on AHP (switch)



------------------------------------------------------------------------
         5.  dSPACE ControlDesk
------------------------------------------------------------------------

STARTUP:
1. load experiment Kraftregler.cdx
   (from C:\Pegasos\aktuell\dspace_ahp\Experimente)

2. goto Platform -> Change connection

3. select Network Connection, using IP 10.71.1.3, and click ok

UPLOAD NEW SOFTWARE:
4. push alt + 1 and alt + 2

5. at the left, switch to Platform (dark greenish icon), at the
   bottom switch to file selector

6. navigate to rtw folder (C:\Pegasos\aktuell\dspace_ahp\rtw)

7. drag and drop the Kraft_regler.sdf file to the ds1005 board and
   confirm the upload

8. if upload and everythings else went well, you can switch to
   animation mode (hit F5)


EDITING:
*  edit things in EDIT MODE.
*  play with things in either TEST MODE (offline mode),
*  or online: RUN MODE


MEASUREMENTS:
*  everything that is in a plotter (!) is recorded.
*  check out the View -> Controlbars -> Instrument selector

1. paste a CaptureSettings Control block into your current Layout

2. click on Settings -> Acquisition

3. Select Stream To disk, and choose a file
```

```
4. dont store things in the Pegasos folder, but elsewhere. choose
   sensible names. Ideally, use Adrian's "Messprotokoll"-Tool.

5. when you click on Start, it'll start capturing, until you push
   Stop.  Caution, everytime you push Start and don't change the
   filename, you will overwrite things.

6. the recorded data is in the .idf format, convert it to .mat
   using ControlDesks converter from Tools -> Convert IDF-File

7. use Daniel's GAI to inspect the mat-file, use Florian's
   convert_data to convert it for use with iotest.mdl. Add the GAI
   folder to Matlab's path (for convenience), then just simply
   run "gai".



------------------------------------------------------------------------
         6.  General Stuff
------------------------------------------------------------------------

PROBLEMS (most likely "Lost connection bla bla bla"):
1. Close control desk
2. Turn off AHP (switch)
3. wait a couple of seconds
4. Relaunch the DSP service (use shortcut on desktop, or run
   services.msc, and hit "neu starten" on "DSP Net Service")
5. turn AHP (swtich) back on
6. elaunch ControlDesk.

If this doesnt work, repeat procedure, but also turning off the PPC
(key). Note that the PPC will keep running about 45 sec after key
has been turned, so make sure you wait that time and watch the
corresponding LED go out. If problems persist, turn off EVERYTHING
(including laptop), and restart ;-)


TURNING THINGS OFF:
1. Shut down ControlDesk
2. turn off engine
3. Wait until car sinks down fully
4. Turn of AHP (switch)
5. Turn the PPC off (key)
```

APPENDIX C

Additional plots

We shall close this report with a number of additional plots, similar to those already shown earlier.

C.1 Validation

The following plots are analogue to Figure 2.3 but use the other three friction models. See Subsection 2.4.3 on page 16 for more details.

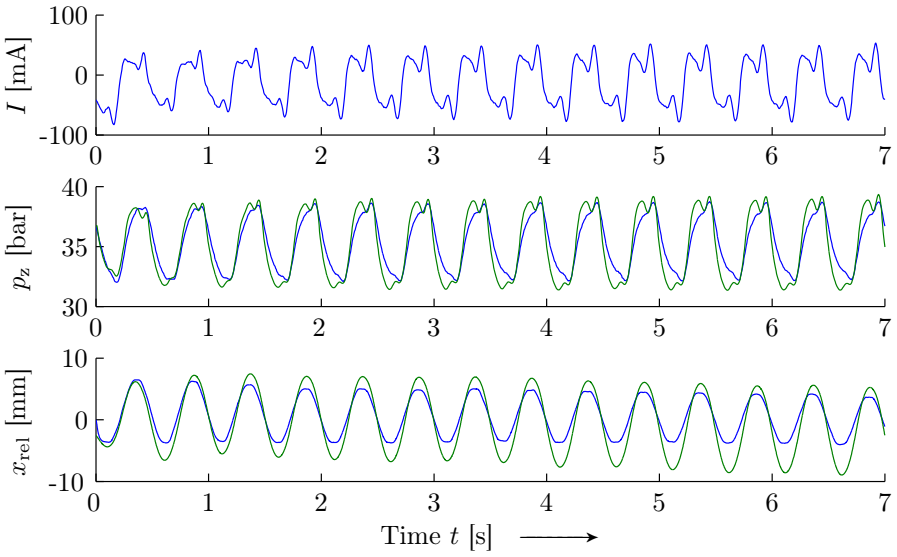

Figure C.1: *Comparison of simulated values (——) with measurements taken in the car (– – –) for the input current shown in the first plot, using friction model (i).*

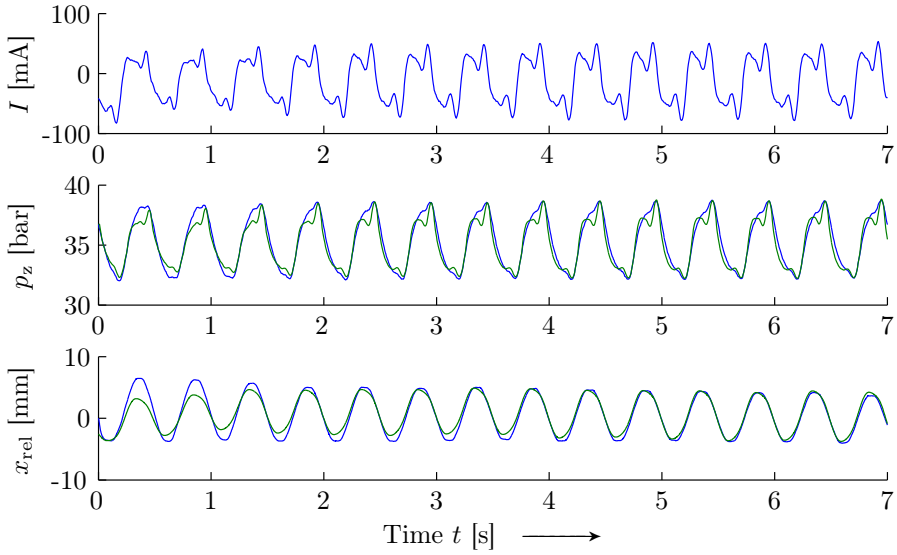

Figure C.2: *Comparison of simulated values (——) with measurements taken in the car (——) for the input current shown in the first plot, using friction model (ii).*

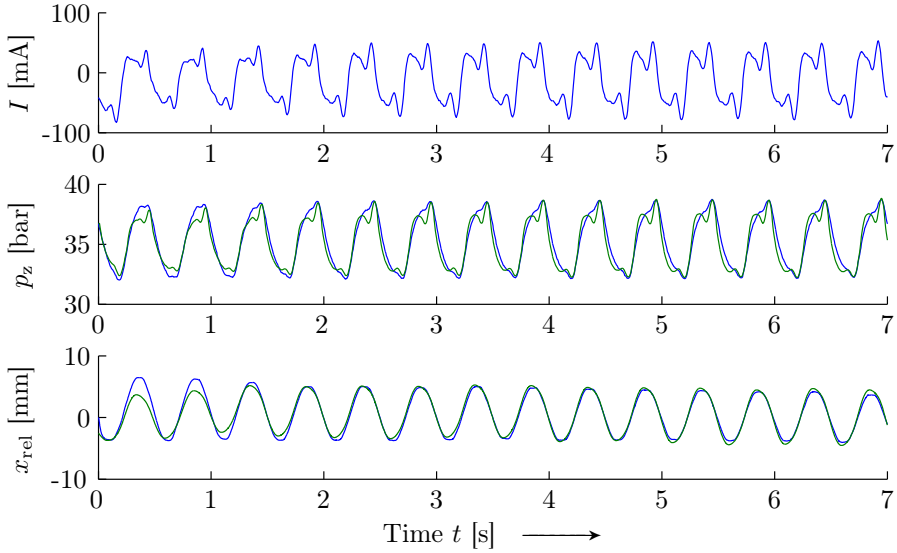

Figure C.3: *Comparison of simulated values (——) with measurements taken in the car (——) for the input current shown in the first plot, using friction model (iii).*

C.2 Controller testing

The remaining figures show further "challenges" for the different controllers, that is various intentional parameter changes for further testing of their robustness (in each case, the perturbation is mentioned in the caption of the figure).

Generally, the controllers do well, and there is not a single major problem occurring. This underlines their rather consistent and robust nature, and underlines again the need for real live testing, as the controllers seem to have passed the "qualification round".

In further tuning, one may try to reduce the overshooting and oscillatory tendencies (the effect of these should also to be evaluated in the car).

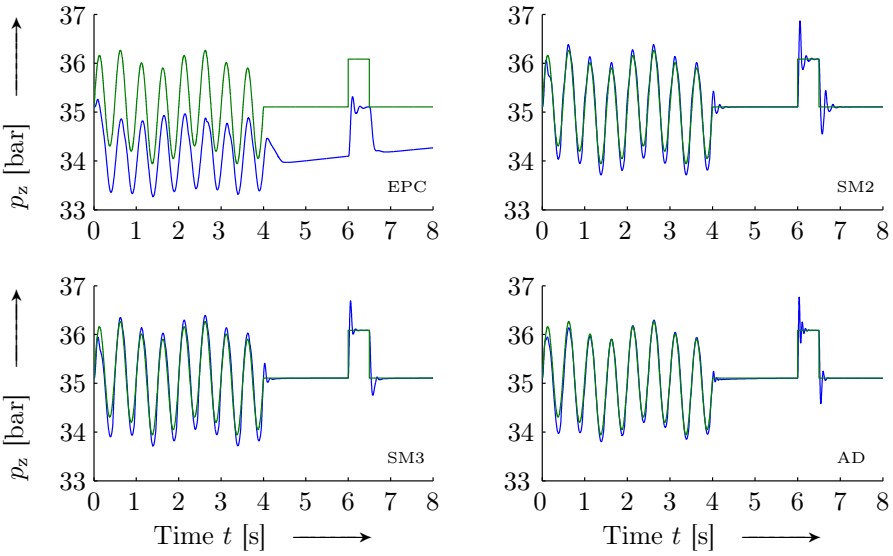

Figure C.4: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance:* -50 *mA offset in* I*.*

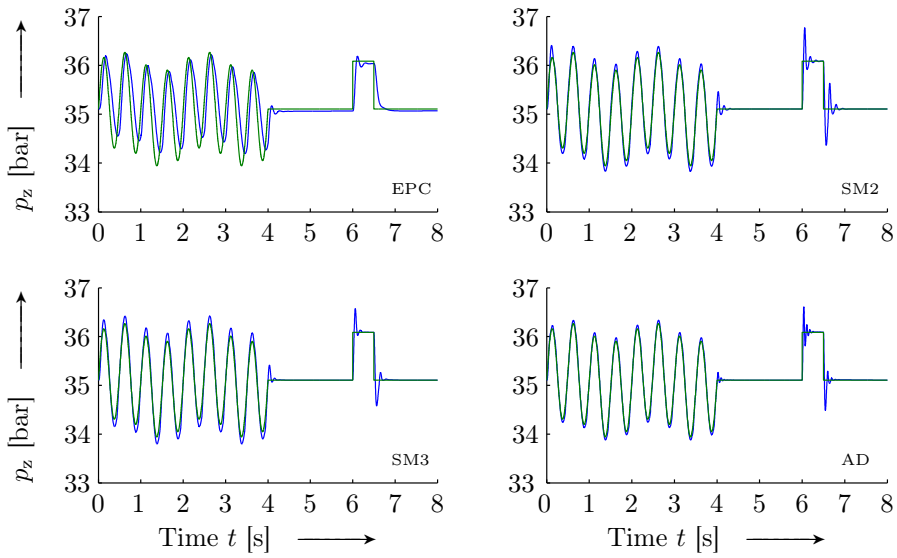

Figure C.5: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance:* k_v *changed by* -20%*.*

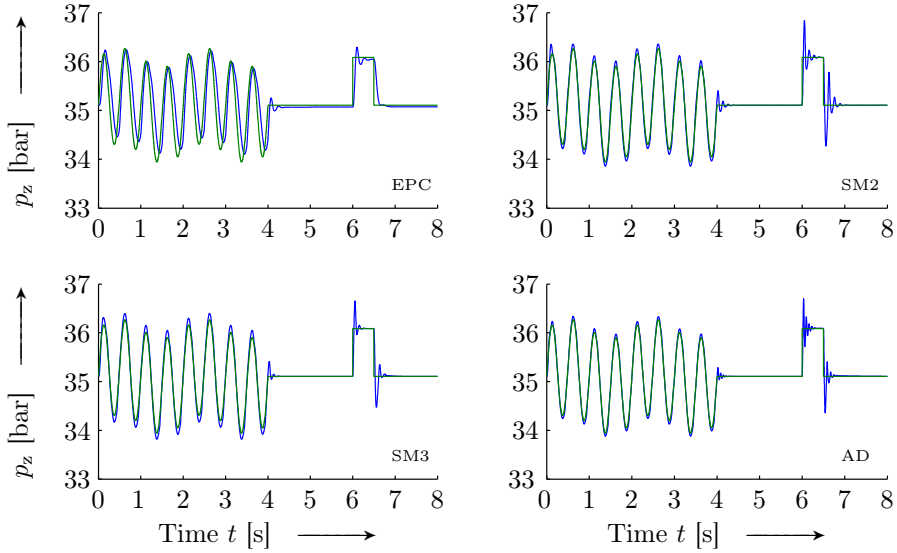

Figure C.6: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance:* a_2 *changed by* $+20\%$*.*

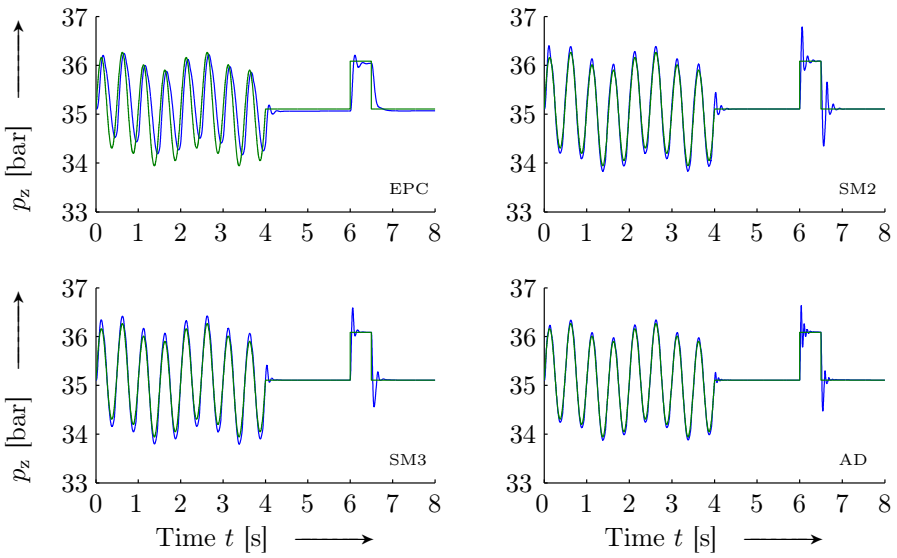

Figure C.7: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance:* a_3 *changed by* $+20\%$*.*

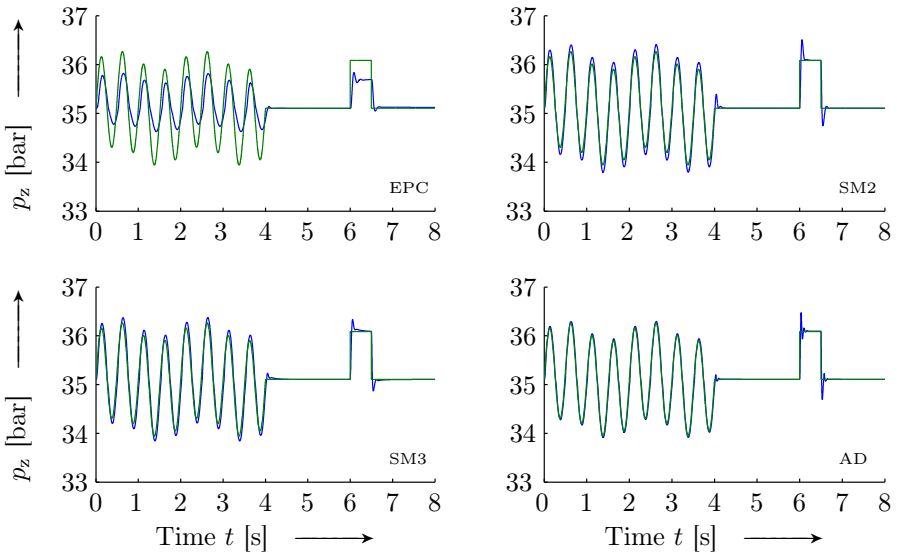

Figure C.8: *Comparison of the performance of the controller, where* $p_{z,d}$ *(——) and* p_z *(——). Plant disturbance:* $\bar{\kappa}$ *changed by* -20%*.*

Bibliography

[1] Bernd Acker, Wolfgang Darenberg, and Heinz Gall. Active suspension for passenger cars. In *IAVSD '89: Proceedings of the 11th Symposium of the International Association for Vehicle System Dynamics*, pages 15–25, August 1989.

[2] Frank Frühauf and Rüdiger Rutz. Innovisia — an active suspension for a coach. *Automatisierungstechnik*, 46:120–127, March 1998.
http://www.oldenbourg.de/cgi-bin/roabstracts?A=659

[3] Hassan K. Khalil. *Nonlinear Systems*. Macmillan Publishing Company, New York, NY, USA, 1992.

[4] Miroslav Krstić, Ioannis Kanellakopoulos, and Petar V. Koktović. *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[5] Henrik Olsson, Karl J. Åström, Carlos Canudas de Wit, Magnus Gäfvert, and Pablo A. Lischinsky-Arenas. Friction models and friction compensation. *European Journal of Control*, 4:176–195, December 1998.
http://www.control.lth.se/~kja/friction.pdf

[6] Michael Pyper, Wilhelm Schiffer, and Walter Schneider. *ABC — Active Body Control*. verlag moderne industrie / DaimlerChrysler AG, Landsberg, Germany, 2003.

[7] Magnus Rau. Modellierung, Simulation und Auslegung eines hydropneumatischen Federbeins mit schnell verstellbarer Dämpfung. Diplomarbeit, 2001.

[8] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice–Hall International, Inc., London, UK, 1991.

[9] Heribert Stroppe, Heinz Langer, and Peter Streitenberger. *Physik für Studenten der Natur– und Ingenieurwissenschaften*. Hanser Fachbuchverlag, Leipzig, Germany, 1999.