Universal Controllers for Architecture Simulation

Alexander Pollok*[a,b] Francesco Casella†[b]

[a]Institute of System Dynamics and Control, DLR German Aerospace Center, Germany
[b]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy

Abstract

For optimization studies of dynamical systems, it is common practice to model and tune local controllers for miscellaneous subsystems. For instance, a model of a chemical plant may contain a valve motor model, and a model of a PID controller may be included to control the motor. The associated controller tuning effort is ultimately wasted. The actual controller will be retuned anyway after finalization of the system design, or will be structurally different.

For this reason, control algorithms are needed that just provide the functionality of the actual control algorithm that will be designed in a later phase of the system design. These temporary algorithms need to have low tuning requirements, and it must be possible for non-control-specialist to generate them. On the other hand, they only need to function inside a simulation environment.

Several mainstream control approaches are reviewed, and boundary layer sliding mode control is proposed as a suitable approach for this kind of task. This class of controllers can be used without any tuning effort, and is able to compete with tuned PID-controllers in terms of tracking performance. An end-user friendly implementation of a universal controller in the equation-based and object-oriented modelling language Modelica is presented. Several examples are shown to demonstrate the performance of the proposed approach.

Keywords: Modelling, Modelica, Sliding Mode, Modelling aids, Optimization, Local Controller

1 Introduction

In industrial projects, typical phases are modelling, architecture optimization, and control systems design. For larger projects, the overlap between the responsible groups of people can be small to nonexistent. Also, not every modelling expert is also a control expert.

For preliminary studies on an architecture level, equation-based object-oriented modelling languages (EOOML) like Modelica are well suited, since individual components can be connected and rearranged quickly and flexibly.

However, additional efforts can arise when the system architecture also includes controlled components, which are also to be sized. An example for this might be pumps that control the mass flow through a pipeline. The actual pump characteristics are parameterized, to be determined during the architecture optimization. During the optimization, each function evaluation corresponds to a simulation of the architecture. For these simulations, the pump needs a controller model that controls the pump in such a way that the target mass flow is reached. This holds, even if the function evaluation is only dependent on the steady state behavior of the system.

A typical workflow for the development of such controllers looks like this: A predefined PID-controller-model from a standard library is included. The controller output is connected to the valve input. Two additional elements are created and connected to the controller model to retrieve or define controller target and actual value. The PID controller is set to P-mode with a k_p of 1, and the system is simulated. The dynamic behavior of the controlled valve is checked, and the controller gain is adjusted for correct order of magnitude and correct sign, if necessary. This can take a few iterations. Subsequently, the controller is set to PI-mode, and a small k_i value is defined to assure zero steady state error. Again, this can take a few iterations until the correct order of magnitude is found. If the dynamical behavior stays insufficient, or if the modelling expert is sufficiently motivated, a few experiments with added derivative action (PID-mode) might follow. Alternatively, optimization tools might be used, shifting the bulk of the effort into the creation of the optimization setup.

This workflow takes some time until the results are acceptable. Also, the resulting controller might not be robust against model changes, making additional effort during the optimization phase necessary. If an architecture contains several controlled components, the necessary effort grows accordingly. This effort is ultimately wasted, since the controllers will be redeveloped by actual control experts anyway, as soon as the architecture is finalized. This is illustrated in Figure 1.

In this paper, preliminary controllers are developed that control a wide range of system models without any

*alexander.pollok@dlr.de
†francesco.casella@polimi.it

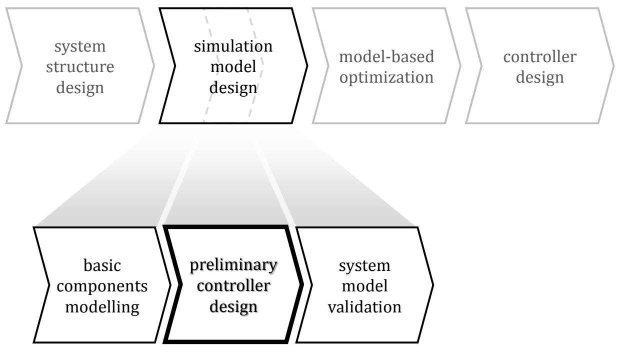


Figure 1. typical workflow for the development of controlled technical systems

tuning effort, or any control systems expertise on the side of the modeller. They are easy to set up, and perform well inside a simulation environment. This is achieved using the class of first-order boundary-layer sliding mode controllers, and implemented in the equation-based modelling language Modelica. The paper is structured as follows: In Section 2, the most important control approaches are reviewed and categorized according to their suitability. The best candidate is identified, then modified to suit the needs of modelling experts in Section 3. Section 4 presents several examples to demonstrate the performance of the resulting control approach. Limits of application of the proposed control approach are discussed in Section 5. The paper is concluded in Section 6.

2 Review of Feedback Control Approaches for SISO systems

2.1 PID

Principle: The controller output is computed as a sum of three components: One component is proportional (P) to the control error, the second component is proportional to the derivative of the control error (D), and the third component is proportional to the time integral (I) of the control error.

Advantages: PID-controllers are the most widely used controllers in the scope of object-oriented modelling. They are easy to understand, offer reasonable performance for most SISO-systems, as well as zero steady state error and simple addition of anti-windup measures. As PID is very commonly used in control algorithms, such a controller would be most representative for the eventual controller implemented with the system.

Disadvantages: PID-controllers require the user to tune 3 parameters, either by hand or using optimization. As soon as a good tuning is found, it might not be suitable for different parameterizations on architecture level. It basically implies control design effort in a stage were just representative functionality is required. The effort will be wasted.

Variants: The concept of PID-controllers has been generalized into Fractional PIDs, as shown in Vinagre et al. (2007). Here, fractional differential operators are used instead of the usual integer operations (integral and derivative action). Fractional differential operators require infinite memory during simulation, but can be approximated in EOOML, see Pollok et al. (2015). However, the number of tuners increases from 3 to 5, therefore the usability for one-size-fits-all-control is even more limited.

2.2 Native Model Inversion

Principle: EOOML are not causal. That means that there is no inherent direction of computation (as opposed to, for instance, a block diagram). Nothing prevents the modeller from defining the nominal output of a system, leaving the computation of the system input to the solver. This can be used for control: The controlled variable is equated to the target value, and the virtual controller output is computed during simulation.

Advantages: The dependency of the controller on the specific system is largely encapsulated in inverse model equations that a simulation environment like Modelica is able to generate automatically. This has enabled automatic control system generation in eg. Aircraft design. No additional effort is necessary on the modellers side. Also, perfect tracking of the controlled variable is possible. The design effort is not wasted, as, once configured appropriately, the control laws can continuously evolve with the system and eventually be implemented.

Disadvantages: Bounds on the controller output cannot be implemented. Also, model inversion usually fails or becomes quite involving, if the system has a high relative degree, gets big, or if there is no unique solution. This affects all but the simplest models. The control architecture must be configured appropriately at the first time, still requiring control expertise.

Variants: Many modern control approaches are based on model inversion. Examples are nonlinear dynamic inversion as used in Thümmel et al. (2005) or incremental nonlinear dynamic inversion as used in Acquatella et al. (2012). However, these approaches lose much of the simplicity of direct inversion, and are therefore not suited for a modelling expert.

2.3 LQR/LQG

Principle: Linear Quadratic Regulators (LQR) represent a static state-feedback-law that optimizes a quadratic cost function (H_2-norm) for LTI-systems. Since they use state-feedback, they require a complete state vector to compute the controller output. Typically, LQRs are combined with a Linear Quadratic Estimator (LQE, or Kalman Filter), to get an estimate of the state vector based on measured variables. The combination is known as Linear Quadratic Gaussian Control (LQG) Kalman et al. (1960).

Advantages: These control laws are more capable in terms of robustness and MIMO systems, thus reducing control design effort in case of systems with strong interaction between different control inputs. For simulation studies, the exact state vector is known to the solver; therefore LQR based control is possible without any need for an estimator. LQR offers guaranteed robustness properties, in contrast to LQG Doyle (1978). The tuning variables of the LQR/LQG approach are design specifications, a skilled control systems engineer can anticipate the effect of the tuning variables on the system behavior.

Disadvantages: Since the computed controller is only optimal for one fixed system, the control design effort is wasted as soon as the system architecture is modified. The computation of the LQR matrix requires a solution of the ricatti-equation, making the use of external tools like Matlab necessary. LQR is based on the assumption that the system is LTI, which is often not the case for complex real world applications. There is no treatment of actuator limitations, as well as no guaranteed zero steady state error. Retrieving the actual state vector in the correct shape might not be that straightforward for the modeller, since environments for EOOML usually encapsulate this information from the user; for this reason alone, the use of LQG in EOOM environments might make sense.

Variants: While the computation of LQR and LQG controllers is straightforward, there are possibilities to extend the method. In Skogestad and Postlethwaite (2007) a variant of an LQG controller is described, where the system model is extended by artificial integral elements. Since the output of these integrators is controlled as well, zero steady state error of the native states can be guaranteed.

2.4 Sliding Mode Control

Principle: In Sliding Mode Control (SMC), a desired subspace (sliding surface) of the system state space is defined in a way that exhibits desirable dynamics. Nonlinear control laws are used to drive the system state onto the sliding surface in finite time. Typically, this takes the form of a bang-bang control-law, where the controller tries to drive the system state into the direction of the sliding surface with maximum authority.

Advantages: SMC is robust against matched uncertainties. As soon as the sliding surface is reached, plant deviations don't lead to deviations in plant dynamics at least for low-order systems. No tuning based on experiments or complex calculations is necessary, only the desired dynamics have to be defined. No control expertise is needed on the side of the modeller. The same controller can be used even if the system is modified later.

Disadvantages: As soon as the sliding surface is reached, chattering occurs. The controller output then changes significantly with each time step. Simulations using implicit solvers and clean event-detection get stuck.

Variants: Several approaches have been described to alleviate the chattering effect: In Filtered SMC as described in Edwards and Spurgeon (1998), the controller output is first-order filtered. In Boundary SMC as described in Utkin et al. (2009), the hard nonlinearity at the sliding surface is replaced by a smooth transition inside a small boundary layer. A number of second Order SMC are described, see for example Bartolini et al. (1998). Here, both the sliding variable (roughly translatable as the distance of the state to the sliding surface) as well as its derivative are driven towards zero in finite time.

2.5 Others

H-infinity synthesis methods have been left out, but their structure, advantages and disadvantages are very similar to LQG-based methods.

Still, the approaches mentioned here don't constitute a complete list. Naturally, it is difficult to review each and every control strategy that has been published. Examples of approaches that have been left out are model predictive control, data driven control system design, adaptive control as described in Åström and Wittenmark (2013) or Intelligent Control techniques like fuzzy control or neural networks Zilouchian and Jamshidi (2000).

However, to the best knowledge of the authors, most of these approaches are either concerned with only a subset of controllable systems, are overly complicated for the task at hand, or have other relevant disadvantages.

3 Universal Controllers for equation-based simulation environments

3.1 Concept

With direct inversion and sliding mode control being the only exceptions, all of the other approaches considered require either manual tuning or external design software. Direct inversion cannot be used for all but the simplest of systems. Second order sliding mode controllers are overly complex, first order sliding mode controllers produce chattering effects, making them unusable for the implicit solvers typically used in EOO modelling environments.

However, modified variants of first order sliding mode control are available to alleviate this problem. Two of them, filtered sliding mode and boundary layer sliding mode were implemented in Modelica and used to regulate the states of a deflected double-integrator back to zero, with first-order decay as sliding surface. For comparison, a conventional first order sliding mode controller was also implemented[1]. The controller output was limited to ± 5. The system behavior for all three controlled system was nearly identical, but the associated controller outputs showed stark differences. While the standard controller exhibited the expected amount of chattering, it was significantly reduced for the filtered sliding mode controller. The Boundary layer variant showed no chattering. These results can be seen in Figure 2. Using implicit solvers, boundary layer sliding mode control simulated roughly 6000 times faster than filtered sliding mode control, probably as a result of generating no state events (compared to 92.942 state events for filtered sliding mode control). That makes the boundary layer variant of first-order sliding mode control a promising candidate for the control of simulated architectures. A commonly cited drawback for boundary layer sliding mode controllers is their sensitivity to noise Young et al. (1996). However, this is not a problem in the context of simulation studies using implicit solvers, where noise is minimal to nonexistent.

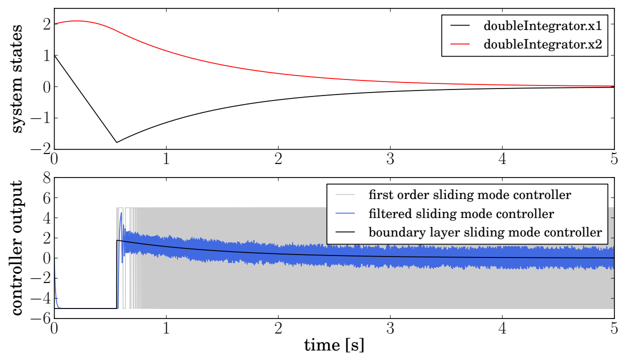


Figure 2. Comparison of different First Order Sliding Mode Controller variants

3.2 Boundary Layer Sliding Mode Control

In sliding mode control, the principle is to force the system state to stay on a defined subspace with beneficial properties. A sliding variable s is defined based on the system output and its derivatives. To enforce first-order decay of a SISO-system, the sliding variable can for instance be defined as

$$s = w_1 \cdot (y - t_{target}) + w_2 \cdot \frac{d}{dt}(y - t_{target}) \qquad (1)$$

with w_1 and w_2 as weighting parameters, determining the time constant of the first order behavior. In the simplest case, the control law takes the form of

$$u = u_{max} \cdot sgn(s) \qquad (2)$$

The idea of boundary layer sliding mode control is to replace the hard discontinuity at $s = 0$ with a linear approximation:

$$u = \begin{cases} -u_{max} & \text{if } s < -l \\ u_{max}\frac{s}{l} & \text{if } -l < s < l \\ u_{max} & \text{if } l < s \end{cases} \qquad (3)$$

with the layer width l. During simulation, this generates state events every time the layer thresholds are crossed, slowing down the simulation. For this reason, a formulation based on the tangens hyperbolicus was used instead:

$$u = u_{max} \cdot tanh\left(\frac{s}{l}\right) \qquad (4)$$

This has the advantage of being solver-friendly, since no state events are generated during simulation. Also, the tangens hyperbolicus is C^∞-continuous, which can be exploited by Dassl and other DAE-solvers that use polynomial expansion.

3.3 Implementation

For implementation in Modelica, usability aspects have to be considered as well as technical aspects. Therefore the controller model is designed to ensure applicability for a wide range of problems, while having a simple interface to the end user.

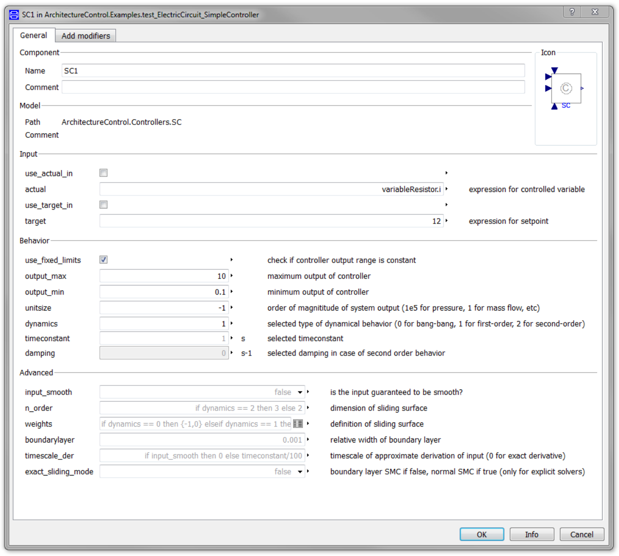
- For input of controlled variable and set point, the user can choose between conditionally defined connectors and input fields, where arbitrary expressions can be included.
- Minimum and maximum controller output can be set as fixed parameters, or can be defined adjustable using conditionally defined connectors.
- Three types of behavior can be selected, corresponding to different structures of the sliding surface: Zero-order dynamics corresponds to perfect matching between set point and actual value whenever possible. For first-order dynamics, a target time constant for control error decay can be entered. For second-order dynamics, both time constant and a damping value are used. In every case, the unused values are greyed out as to not confuse users.

[1]Standard First Order Sliding Mode Controllers cannot be simulated using Dassl Petzold et al. (1982), the standard solver for many Modelica environments. For this reason, an explicit solver (RK4) was used for this comparison.

- The relative width of the boundary layer is dependent on the order of magnitude of the controlled variable. If a pressure variable is controlled, typical values are in the order of 10^5. This makes the boundary layer much smaller in comparison. Therefore the user can also enter a unit size variable to compensate. This variable can also be used to change the sign, if the controller output goes in the wrong direction, thereby making it the only tuning decision which is necessary, albeit a binary one.

- For users that know what they are doing, more options are available. These are grouped in an "Advanced"-tab. Here, the input can be flagged as smooth, so that exact derivatives are used for the calculation of the sliding variable instead of approximate derivatives. Also, the definition of the sliding surface can be overwritten. Finally, the boundary layer formulation can be replaced with a standard sliding mode controller, which might make sense if an explicit solver is used.

The user interface of the controller model is shown in Figure 3. The shortened Modelica code listing can be seen in the Appendix in Section A.



SC1 in ArchitectureControl.Examples.test_ElectricCircuit_SimpleController

General | Add modifiers

Component

Name	SC1
Comment	

Model

Path	ArchitectureControl.Controllers.SC
Comment	

Input

use_actual_in	☐		
actual	variableResistor.i		expression for controlled variable
use_target_in	☐		
target	12		expression for setpoint

Behavior

use_fixed_limits	☑		check if controller output range is constant
output_max	10		maximum output of controller
output_min	0.1		minimum output of controller
unitsize	-1		order of magnititude of system output (1e5 for pressure, 1 for mass flow, etc)
dynamics	1		selected type of dynamical behavior (0 for bang-bang, 1 for first-order, 2 for second-order)
timeconstant	1	s	selected timeconstant
damping	0	s-1	selected damping in case of second order behavior

Advanced

input_smooth	false	is the input guaranteed to be smooth?
n_order	if dynamics == 2 then 3 else 2	dimension of sliding surface
weights	if dynamics == 0 then {-1,0} elseif dynamics == 1 the	definition of sliding surface
boundarylayer	0.001	relative width of boundary layer
timescale_der	if input_smooth then 0 else timeconstant/100	timescale of approximate derivation of input (0 for exact derivative)
exact_sliding_mode	false	boundary layer SMC if false, normal SMC if true (only for explicit solvers)

Figure 3. User Interface of SimpleController parameters

4 Examples

4.1 Low-Order-Systems

As a first test, the controller is used to control three different low-order-systems. The first system is a static unity-gain, the next system is a first-order system with a time constant of three seconds, the last system is a second-order system with an angular frequency of 0.5 rad/s and zero damping.

All controllers use the standard values: First-order target behavior with a time constant of one second. Output limits are set to ± 5. All systems were initialized at state vectors of zero, the target output was defined as a step at $t = 1s$. The results are shown in Figure 4.

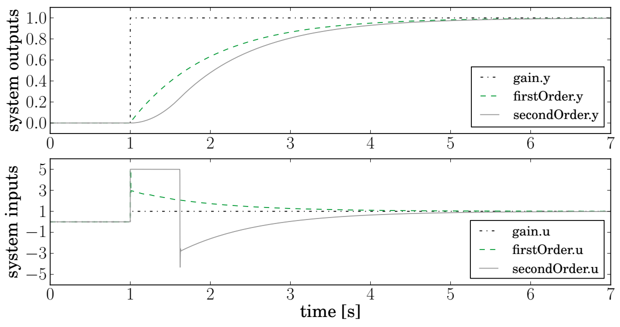


Figure 4. Dynamic behavior of several low order systems controlled by boundary layer sliding mode controller

The first-order system is forced on the reference behavior almost exactly. A small overshoot in the system input/controller output is visible, caused by the time constant of the approximate derivative used for the controller formulation.

The behavior of the second-order system is significantly limited by the controller limits. At 1.62 seconds, this limitation is no longer relevant, and the system reverts to first-order behavior. Again, a small overshoot occurs at the controller output, caused by the approximate derivative.

Interestingly, the static gain doesn't show any dynamic behavior; instead the system output perfectly matches the target. This can be explained by the feedthrough behavior of the controller, which instantly compensates the set point change. Since no control error is being generated, no control error has to fade away with a first-order behavior. In contrast to this, a control error was being generated for the first- and second-order systems, when the controllers couldn't track the set point perfectly due to controller output limitations.

4.2 Thermal Network

To estimate controller performance on systems with a high relative degree, a model of a thermal network was created. The network consists of 9 thermal masses, connected in a two-dimensional grid. At one corner, a fixed temperature boundary condition of -10 egrees C is applied, at the opposite corner, a temperature is measured. This measured temperature is also the control variable, with a set point of 0 degrees C. At one of the remaining corners, a disturbance heat flow is applied, taking the shape of a saw tooth function. The last corner is reserved for the actuator: a variable heat flow, with a range of 0 to 10 Watts. The lower limit of 0 Watt corresponds to a pure heating element, without any cooling capability. It also introduces a strong nonlinearity to the system. All thermal masses were initialized at 20 degrees C. The system can be seen in Figure 5.

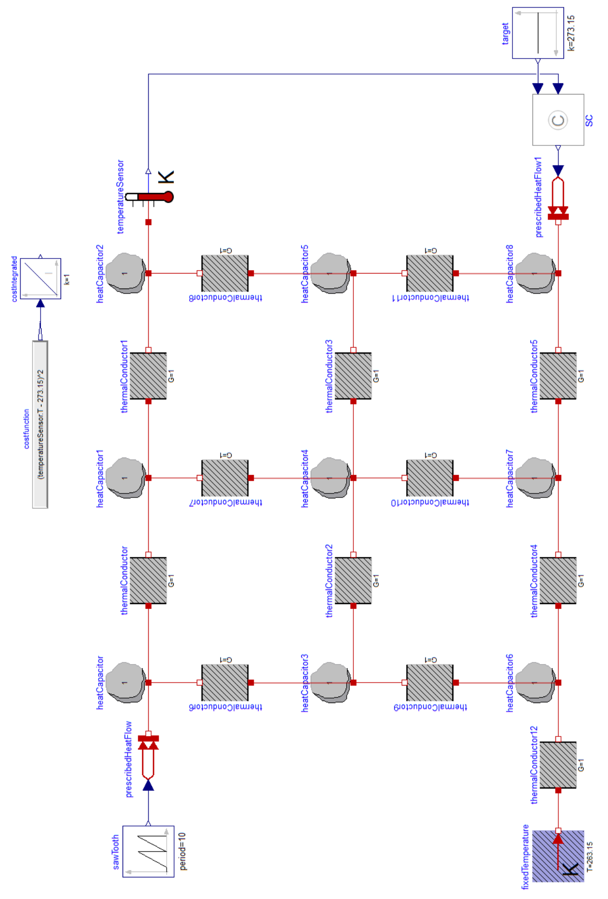


Figure 5. thermal network test model

The system was instantiated two times, using different controllers. One time, the proposed boundary layer sliding mode controller was used, with a selected second-order behavior, a time constant of 1 second and a damping of 1.

For comparison, the system was instantiated with a PID-controller, including anti-windup functionality. The 3 parameters of the PID were optimized using the Modelica Optimization library Pfeiffer (2012). As an optimization goal, the integrated quadratic control error (IQCE) for a simulated time of 100 seconds was used. The optimization achieved an IQCE of 3016.3.

The sliding mode controller achieved an IQCE of 3014.8 without any optimization. The associated temperatures and controller outputs can be seen in Figure 6.

5 Discussion

In the previous Sections it was shown that Boundary Layer Sliding Mode Controllers are a very easy to use tool, suitable for a wide range of simulation tasks. Performance wise, this class of controllers can compete with other control approaches. At the same time, configuration efforts for the modelling expert are minimal.

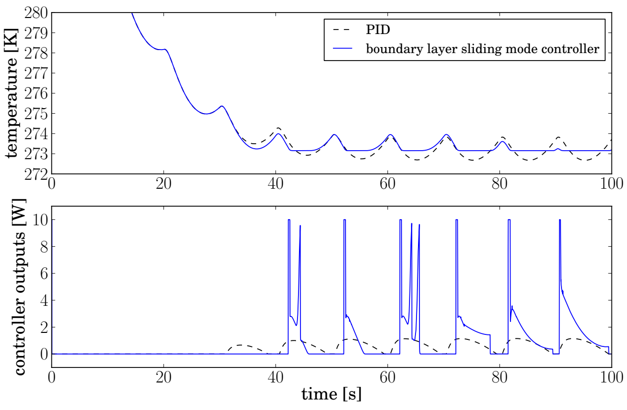


Figure 6. thermal network test results

There are however situations, where the proposed controller is not adequate. For instance, this applies to MIMO-systems with strong coupling. If the water level in n water tanks has to be controlled using 2 valves, and there is no clear one-to-one assignment between the tanks and valves, the proposed controller will probably fail. But as long as the influence of each valve on a single tank is strong, and the influence of the respective valve on all other tanks is weak, a number of boundary layer sliding mode SISO-controllers can be used. For a rule of thumb: If a system can be controlled by a number of PID-controllers, the proposed approach has a good chance of regulating the system in a satisfactory manner.

Another limitation is given by systems that include a hard dead time. Several numerical experiments were done were hard delay-elements were introduced into otherwise simple and good-natured systems. It showed that the introduction of these delay-elements, however small, completely prevented any form of convergence. However, as long as physical dead-times are approximated by first-order elements (this is done for instance in the dynamic pipe model of Modelica.Fluid) the proposed approach should work out fine.

Controllers based around sliding mode concepts don't have a well-defined gain. Small deviations in behavior of the controlled system can however result in large changes in controller output. In this sense, they behave similar to linear controllers with large gain. As such, they can be classified as aggressive controllers. If simulation models are not robust, there may be situations where a boundary layer sliding mode controller prevents the numerical solver from finding simulation results. For the same system, a cautiously tuned PI-controller will probably give better results.

Nevertheless, for a large class of simulation models, the proposed control concept will deliver good results, taking up only very little of the modelling experts time. As long as modelling experts keep the mentioned limitations in mind, the proposed controller can greatly improve productivity.

6 Conclusion

For modelling and simulation projects of complex technical systems, often many local controllers have to be modelled, without them being used in the final architecture. We identify boundary layer sliding mode control as a suitable approach, offering good performance without any tuning effort for many - but not all - systems. The major drawback of this class of controllers - sensitivity to measurement noise - is irrelevant in the context of simulation models.

Acknowledgements

We thank Trey and Matt for inspiration.

References

Paul Acquatella, Wouter Falkena, Erik-Jan van Kampen, and Q Ping Chu. Robust nonlinear spacecraft attitude control using incremental nonlinear dynamic inversion. In *AIAA Guidance, Navigation, and Control Conference*, page 4623, 2012.

Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

G Bartolini, A Ferrara, and E Usai. Chattering avoidance by second-order sliding mode control. *IEEE Transactions on Automatic control*, 43(2):241–246, 1998.

John Doyle. Guaranteed margins for lqg regulators. *IEEE Transactions on Automatic Control*, 23(4):756–757, 1978.

Christopher Edwards and Sarah Spurgeon. *Sliding mode control: theory and applications*. Crc Press, 1998.

Rudolf Emil Kalman et al. Contributions to the theory of optimal control. *Bol. Soc. Mat. Mexicana*, 5(2):102–119, 1960.

Linda R Petzold et al. A description of dassl: A differential/algebraic system solver. *Scientific computing*, 1:65–68, 1982.

Andreas Pfeiffer. Optimization library for interactive multicriteria optimization tasks. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*, number 76, pages 669–680. Linköping University Electronic Press; Linköpings universitet, 2012.

Alexander Pollok, Dirk Zimmer, and Francesco Casella. Fractional-order modelling in Modelica. In *Proceedings of the 11th International Modelica Conference*, 2015.

Sigurd Skogestad and Ian Postlethwaite. *Multivariable feedback control: analysis and design*, volume 2. Wiley New York, 2007.

Michael Thümmel, Gertjan Looye, Matthias Kurze, Martin Otter, and Johann Bals. Nonlinear inverse models for control. In *Proceedings of the 4th International Modelica Conference*, pages 267–279, 2005.

Vadim Utkin, Jürgen Guldner, and Jingxin Shi. *Sliding mode control in electro-mechanical systems*, volume 34. CRC press, 2009.

Blas M Vinagre, Concepción A Monje, Antonio J Calderón, and José I Suárez. Fractional pid controllers for industry application. a brief introduction. *Journal of Vibration and Control*, 13(9-10):1419–1429, 2007.

K David Young, Vadim I Utkin, and Umit Ozguner. A control engineer's guide to sliding mode control. In *Variable Structure Systems, 1996. VSS'96. Proceedings., 1996 IEEE International Workshop on*, pages 1–14. IEEE, 1996.

Ali Zilouchian and Mohammad Jamshidi. *Intelligent control systems using soft computing methodologies*. CRC Press, Inc., 2000.

A Modelica Code of Simple Controller

Listing 1. Modelica Code of SimpleController

```
model SC
  ...
equation
  ...
  x[1] = actual_in_internal
    - target_in_internal;
  for i in 2:n_order loop
   x[i] = td1[i-1].y;
   x[i-1] = td1[i-1].u;
  end for;
  if exact_sliding_mode then
    y = if x*weights > 0 then output_max
    else output_min;
  else
    y = mean + authority*Modelica.Math.tanh(
         x*weights/(boundarylayer*unitsize));
  end if;

  annotation (...);
end SC;
```