On Simulation of Simulink® Models for Model-Based Design

Rohit Shenoy, Brian McKay, and Pieter J. Mosterman

The MathWorks, Inc., 3 Apple Hill Dr., Natick, MA 01760, USA
[rohit.shenoy|brian.mckay|pieter.mosterman]@mathworks.com

Summary. Computational modeling is increasingly being used to support the design of engineered systems. Rather than relying on a physical prototype of the system, it allows the use of a model that can be analyzed with computational methods. The arguably most prolific computational analysis in systems design is numerical simulation. This chapter discusses the use of numerical simulation of computational models for the design of embedded systems, in partical embedded control systems.

DRAFT

1 Introduction

To remain competitive and reduce cost, industry increasingly relies on computational models [23]. Designing computational models and using numerical simulation is an alternative to building hardware prototypes for testing purposes. Computational modeling [1, 5, 7, 9] has a number of advantages over traditional engineering methods, such as:

- Computational models tend to be less expensive to produce and easier to modify.
- The ease of modification enables answering 'what-if' questions by facilitating rapid exploration of design options, a task that is time-consuming and expensive with physical prototypes. Additionally, some experiments require multiple simulations with different parameters which can be performed through Monte Carlo simulations. This is impossible with physical prototypes because prohibitively expensive.
- In the case of complex systems such as aircraft, power grids, and communication networks, simulation enables unstable and emergency modes to be tested more safely, without risk to infrastructure or human life.
- Simulation of such unstable and emergency modes is also an order of magnitude less expensive. For example, testing an aircraft to failure would cost much more than simulating such an experiment, which, if the failure would lead to a crash and loss of life, would not be possible otherwise.

- Modern computational packages allow engineers to employ designs that have been used in simulation and use them in the eventual product. This is a major time-saving feature.

Building dynamic models in software is part of Model-Based Design [3], which has all the merits listed above and is being adopted by engineers in many fields, in particular, the aerospace and automotive industry. It can be applied to applications such as feedback control, communications, signal and image processing, and financial modeling. Model-Based Design with Simulink® [33] enables design, analysis, and simulation of processes in a safe software environment. Consider the example of a robotic arm designed for use in a production plant, shown in Fig. 1. Creating a physical prototype would be expensive and time-consuming. Testing it could be dangerous if the robot malfunctioned. The alternative is to model the mechanical, electrical, and electronic components of the robot in software, and then use the model to test the algorithms through simulation.

Fig. 1. A robotic arm.

Model-Based Design enables continuous testing as algorithms and computational models are created and refined. The integration of testing into the design process can identify problems sooner than approaches that do not use modeling and simulation. Furthermore, tests can be designed earlier in the overall design process, which is particularly advantageous if carried out in parallel with other design tasks. Finally, Model-Based Design with Simulink provides the capability to generate computer code such as C or C++ from algorithms and computational system models, and use that code for implementation and testing. In particular, hardware-in-the-loop simulation which uses a real-time computational model of a physical system will be discussed, as

well as the systematic testing process of a designed system [23]. These aspects help bridge the gap between the software and hardware design.

This article outlines workflow and software tools for computational modeling and numerical simulation of dynamic systems with MATLAB® [20] and Simulink. It will not concentrate on the detailed methodological and technological aspects of numerical simulation of systems with combined continuous-time and discrete-event behavior, so-called *hybrid dynamic systems* [21]. Instead, one of the key goals of the article is to explain the selection criteria for modeling techniques based on simulation, design, and testing needs. Section 2 examines a robotic arm as a case study and several examples with the goal of understanding different modeling techniques. Section 3 discusses how simulation can be used in control system design. Section 4 presents the modeling spectrum spanning first-principles modeling from equations, physics-based modeling and data-driven modeling. Section 5 applies the different modeling approaches to the robotic arm of the case study. Section 6 discusses further usages of simulation in control system design. Section 7 gives an overview of the use of testing as a simulation tool in the design of control systems. Section 8 presents the conclusions of this work.

DRAFT

2 The Case Study Example

A robotic arm application, shown in Fig. 1, is used as a case study because it is relatively generic and so will reduce the need for discussing overwhelming engineering theory. The robotic arm is a commercial off-the-shelf product.[1] It consists of a turntable base, an 'upper arm', a 'forearm', and a 'hand.' Two revolute joints connect the turntable to the upper arm, and a single revolute joint provides the connection to each other part. Revolute joints have one degree of rotational motion about a fixed axis, such as a wheel spinning on an axle. Each joint is connected to a DC motor which serves as the joint actuator, and a potentiometer, which serves as a position sensor. The motor of a joint moves the bodies connected at the joint relative to each other. The sensor measures and reports how far the bodies have moved. Each actuator/sensor pair provides an input/output point, which can be used to manipulate the device through the use of feedback control [2, 10].

2.1 Feedback Control

Consider the task of moving the upper arm in the robotic arm to a specific position from its current position. The steps are as follows:

- The position sensor reports its current position.
- The position is compared to the desired position and it is decided in which direction the motor should turn.

[1] It can be acquired through Lynxmotion, www.lynxmotion.com.

- The motor starts turning in the desired direction.
- At a regular time interval, for example, every 0.01 seconds, the sensor value is read and compared to the desired position.
- The speed and direction of the motor are then adjusted depending on whether the required position is being approached or has been passed.
- These steps are repeated until the required position is reached plus or minus some acceptable threshold.

In this simple feedback-control algorithm, information is returned regarding the current state of the robotic arm position, which enables making a decision on what should happen in the next time step. Feedback control theory and strategies can be arbitrarily complex [2, 13, 26, 42], but the basic process outlined above provides the context to understand how any feedback-control application can be taken and the algorithm designed, tested, and simulated. Figure 2 shows the basic feedback-control loop. The plant, the device to be controlled, is the upper arm with its motor. The sensor is the position sensor. The controller is the algorithm that adjusts the motor speed and direction on the basis of the difference between desired and actual position.

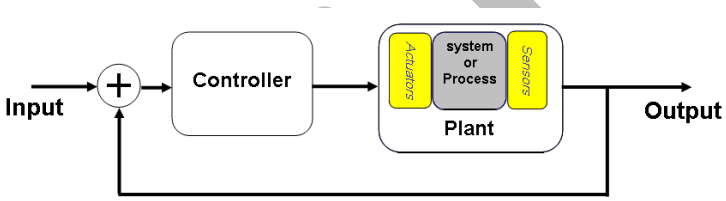


Fig. 2. Elements of embedded control systems.

Now, consider the entire arm. By controlling the motion of each arm segment about the four joints, the hand can be moved. This is a multiple-input, multiple-output (MIMO) problem of great interest to control engineers because many systems have multiple inputs and multiple outputs. Two examples of this are:

- Washing machines must control water temperature and level, and motor speed based on the type of clothes and size of the load.
- Car engines must adjust intake and exhaust levels to meet performance and/or efficiency requirements.

In general, the use of computational modeling and simulation as described in this paper has a prolific application in industry.

DRAFT

3 Designing with Simulation

A 'design' as used hereafter is defined as a software component or algorithm that functions either alone or with other components or algorithms. To study the behavior of a new design, a model is needed that includes not only the design but other components, systems, or algorithms as well with which the design interacts. Once a computational model of the entire system is available, simulation can be used to test and refine the design before going to a final implementation.

Consider the robotic arm. The design is the control algorithm that will be embedded on a microprocessor connected to the robotic arm. The entire system that must be modeled includes the algorithm, the actual robot mechanics, electrical circuitry, and any other devices and components that the robot will interact with. Once this has been modeled accurately, the feedback-control algorithm discussed in the previous section can be designed. As the algorithm is designed, testing of it in combination with the robot can be started in a simulation. Since simulated rather than actual hardware is used, it can be quickly experimented with algorithms as they are developed without costly damage. This would not be possible with a physical robot because hardware would be likely to be damaged during experimentation with different designs. With actual hardware, a small fleet of robots would be needed to run the same tests.

4 Obtaining Computational Models

Here, focus is on the parts of the computational model other than the control algorithm. In the case of the robotic arm this is the plant, which consists of the actual mechanics, electrical circuitry, and dynamic effects such as friction. Computational models of physical devices and phenomena can be obtained by one or more of the following methods (this is not an exhaustive list):

1. Derive mathematical relationships from published or otherwise well-accepted theory, also called first-principles modeling. To an extent, this area pertains to phenemenological modeling [6, 16].
2. Derive mathematical relationships using data gathered from the devices being modeled. The data can then be used in a data-fitting approach [4, 17].
3. Use modeling software that derives computational models from physical information such as mass, inertia, resistance, and capacitance (also considered first principles). This is more closely affiliated with structural modeling approaches such as Modelica [12, 39].

The first method is typically the most time consuming, but is widely accepted because the engineer has visual access to the equations that describe the system. The second method relies on techniques such as curve fitting,

DRAFT

fuzzy logic, neural networks, and nonlinear optimizations to produce mathematical estimates of a systems behavior. While this method can be effective at obtaining models quickly, the engineer must consider several test scenarios to ensure that the computational models capture all the system dynamics. The third method is typically the most practical to implement, since the engineer needs to know only physical properties, not mathematical relationships.

4.1 Modeling From First Principles

First-principle modeling methods offer the capability of testing a design through simulation without committing to specific hardware. This enables development engineers to explore design options more thoroughly and discover problems early, saving time and money.

Before looking at obtaining the computational model of the robotic arm, consider the double pendulum in Fig. 3(a) that is similar to the robotic arm. It contains one less linkage and no turntable. The kinematics equations could be derived based on first principles (Method 1). The dynamics, on the other hand are much more difficult to model. The dynamics will require friction models that are difficult to come by, certainly by derivation from first principles (see, e.g., [18]). To do so would require knowledge much beyond that of multibody dynamics.

The following equation represents the kinematics of the double pendulum in Fig. 3(a)

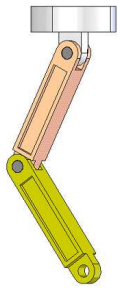
$$\alpha = \int \frac{-L_2 sin(\alpha) + mw_2(-sin(\alpha-\gamma))sin(\gamma) - nc(-sin(\alpha-\gamma))cos(\alpha-\gamma)\alpha^2 - ncos(\alpha-\gamma)r^2}{1 - nesin^2(\alpha-\gamma)} \tag{1}$$

Note that, in spite of the apparent complexity, Eq. (1) does not yet include a friction model. The corresponding Simulink model for the generalized coordinates is given Fig. 3(b). Clearly, the equations for this relatively simple system can be complex. The more complicated robotic arm will have an even more complicated formulation.

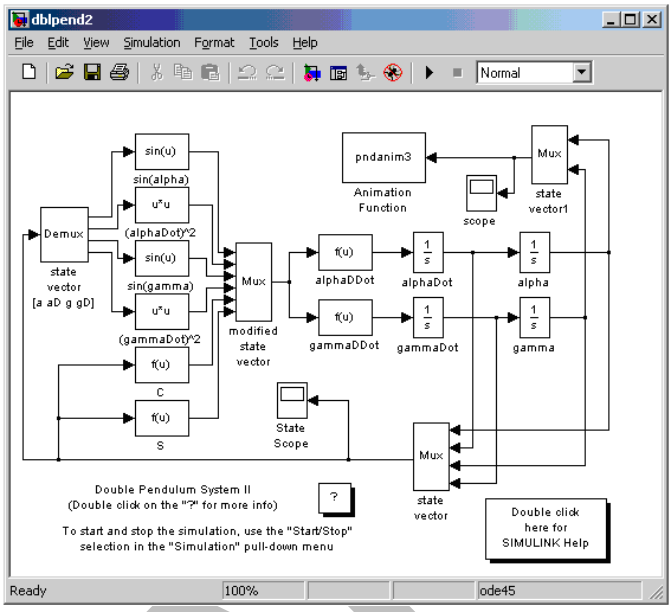
To further compound matters, where Eq. (1) uses a generalized coordinate system with only two variables, an implementation with Cartesian coordinates would have 12 variables. It is obvious how this relatively simple device can become very complicated to model and analyze in detail.

The alternative is to use Method 3 for first-principles modeling. The pendulum can be considered a multi-rigid-body system that can be modeled in SimMechanics [32], a modeling tool that works within Simulink and provides multibody mechanical modeling capabilities. The user needs only information regarding mass, inertia, dimension, and the position and orientation of joint axes of motion to create a SimMechanics model. With this information the model compiler can automatically derive the system of differential and algebraic equations (DAE) that capture the model behavior [40].

DRAFT



(a) Multi-body model



(b) Simulink® model

Fig. 3. A pendulum.

Users can assemble bodies and joints to model their systems, or model their systems in SolidWorks® [37] and automatically extract SimMechanics models. Figure 4 shows the SimMechanics model with its one-to-one correspondence to the physical components. This model can then be integrated with standard Simulink blocks that can be used to model effects such as friction or a control algorithm in continuous-time, sampled time, or scheduled task form.

Computer Automated Multiparadigm Modeling is increasingly become important to negotiate the complexity of modern engineered systems [24]. It relies heavily on domain specific formalisms and tools. In this vein, tools to model physical systems other than SimMechanics that generate the mathematical equations from higher-level component representations in different domains are available to the modeler as well. Simulink integrates tightly with four tools for component-level first-principles modeling of physical systems (Method 3):

- SimMechanics models three-dimensional mechanical systems.
- SimPowerSystems [15] models electrical circuitry and power flow.
- SimDriveline [30] models one-dimensional rotational motion.
- SimHydraulics™ [31] models hydraulic systems.

DRAFT

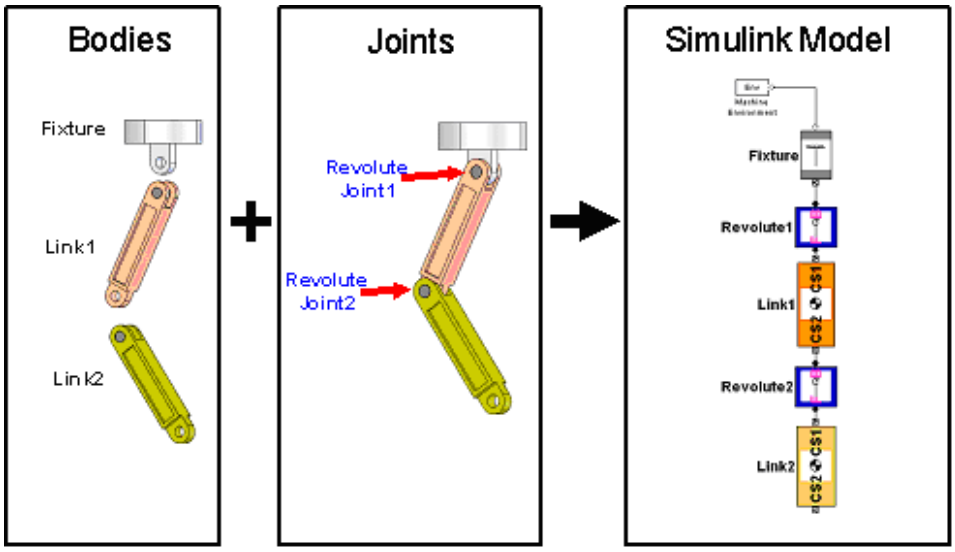


Fig. 4. Elements of a multi-body pendulum model.

DRAFT

These tools can be used in conjunction with equations derived in Simulink to model various types of systems for a variety of purposes.

4.2 Using Data and Simulation to Obtain and/or Tune Models

Often, one does not have the luxury of using Method 3 exclusively and must use Method 1 to model some components. Simulink provides an environment where all the tools for modeling physical systems that are referenced above can be mixed with equations represented with standard Simulink blocks. Method 1 and Method 3 assume that the behavior of the system is well-understood. In an ideal world this is true, and one could model complex behaviors such as friction very accurately. In the real world, however, it must be dealt with manufacturing tolerances, lack of information about component behavior, and other uncertainties regarding the systems being modeled. This is where Method 2 is used, which relies on data collected from the components to either tune existing models built with Method 1 and Method 3, or generate linear models for 'black box systems.' As mentioned earlier, there are a variety of data-fitting techniques but the basic principle is as follows:

- Apply some disturbance or other input to the system such as moving the double pendulum in Fig. 3(b) in a measured way.
- Measure any change in the system with sensors, this is the system output (here, the pendulum angle versus time).
- Use the input/output data pairs to

- tune parameters in existing models, using tools such as Simulink® Parameter Estimation [11] (considered 'grey-box' modeling), or
- generate models using an appropriate tool such as the System Identification Toolbox [38] or the Fuzzy Logic Toolbox [14] (considered black-box modeling).

The black-box techniques such as fuzzy logic are particularly useful in characterizing phenomena such as chemical processes for which equations might not be easily derived. These modeling methods can be applied to many systems and processes. Both types of data modeling (grey-box and black-box) are particularly interesting because they rely on simulation, to tune parameters and generate models, respectively. In the case of grey-box modeling, Simulink® Parameter Estimation simulates a model hundreds or thousands of times while using optimization techniques to adapt parameter values until the actual system output matches the measured system output. This is referred to as *parameter tuning* as well. In the case of black-box modeling, the techniques vary slightly depending on the method. The System Identification Toolbox [38] enables multiple simulations to generate a model that will give the desired measured output from the associated input. Once the parameters have been tuned so the model reflects the observations of reality accurately, the model can be used in the larger system simulation. Models generated from data can be incorporated into larger system simulations as well.

Another approach to modeling with data is transfer function estimation. This is the process of taking experimental data and converting it using spectral estimation techniques to compute the frequency response of a system [4]. The Signal Processing Toolbox [28] has many functions that aid in the estimation of a transfer function. Additionally, a linear parametric model can be fit to the experimental frequency response function using the modeling tools in the System Identification Toolbox.

In some cases, linear models may not describe the model accurately and the underlying equations of motion may not be that well known. In this case a nonlinear black-box neural network can be created. These types of models can be created using the Neural Network Toolbox [25].

DRAFT

5 The Robotic Arm Model

A computational model of the robotic arm is obtained using the different approaches.

5.1 The Mechanical Model of Arm

The computational model of the robotic arm consists of the mechanical model, the motor models, and the actual controller feedback loop. To model the mechanics, the robot is first modeled with a Computer-Aided Design (CAD)

tool, in this case SolidWorks (Fig. 5) using the engineering drawings provided by the manufacturer (Fig. 6). The SimMechanics model is then automatically extracted from the SolidWorks model.

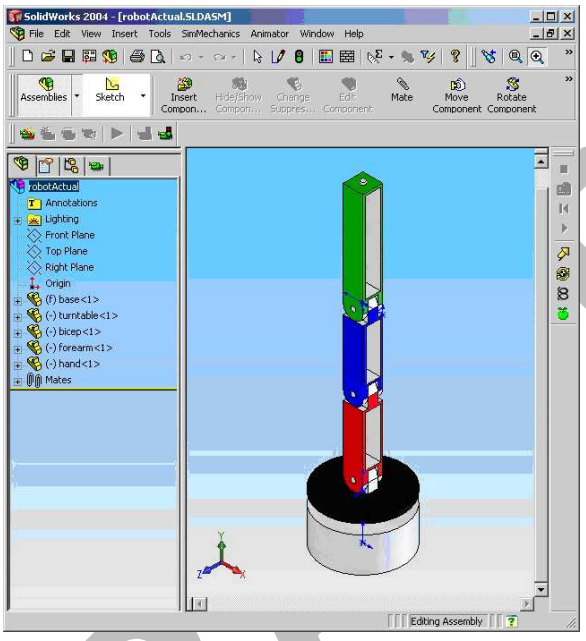


Fig. 5. CAD model of a robotic arm.

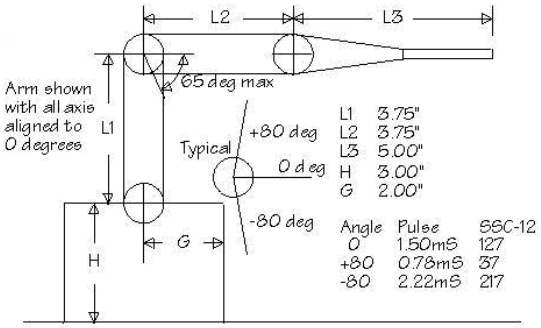


Fig. 6. Engineering drawing of a robotic arm.

SimMechanics models are composed of bodies connected to each other by joints that have user-specified ranges of motion. The dimensions of the bodies

and the axes of motion for the joints are specified in a Cartesian coordinate space (x, y, z). The robot in Fig. 5, has five bodies: three arm segments, the turntable, and the base. The base is fixed to a reference point, the other dimensions and axes are specified relative to this ground position. Each body is connected to the adjacent body by a joint as described in Section 2.

5.2 Electromechanical Model of the Motor and Coupling

Once the kinematics of the mechanical model are available, the dynamics for the motors and the couplings that connect the motors to the joints must be modeled. The joints are assumed to be frictionless (which is a coarse approximation of reality, but sufficient for this application). A pulse-width modulation scheme regulates the voltage to the electric motor and controls the motion of the linkage. The switching circuitry is modeled using an H-Bridge, a digital device that can switch the direction of current flow. The motor itself consists of an inductor, a resistor, and a model of back electromotive force (EMF). The component blocks for the motor and H-Bridge are from SimPowerSystems. The remaining components in the motor model are gearing, inertia, and other mechanical components, all modeled in SimDriveline. Since each joint has one motor, the model can be made a reference to a library component and a separate motor model connected at each joint. An individual motor model is shown in Fig. 7. The motor models can then be coupled to the SimMechanics model of the robotic arm. Standard sensor and actuator interface blocks are necessary when moving from one modeling formalism to another, such as from SimMechanics to SimPowerSystems, or to standard Simulink. The blocks translate physical properties such as force and torque into the time-based signals of Simulink.

5.3 Tuning the Motor Parameters with Data

The DC motor model shows a relationship between current and torque. Torque causes the motor shaft to spin in accordance with a relationship to the back EMF. The remaining parameters include shaft inertia, viscous friction (damping), armature resistance, and armature inductance. Values for those parameters must be accurate for the motor model to behave similar to the actual motor. While manufacturers may provide the values, one should assume those to be averged values with added manufacturing tolerances. It is necessary to estimate these parameters as precisely as possible for the model to ascertain whether it is an accurate representation of the actual DC servo motor system. Table 1 lists the model parameters and their initial values.

When a series of voltage pulses is input to the motor, the motor shaft turns in response. If there is a discrepancy between the model parameters and those of the physical system, however, the model response will not match that of the actual system. Figure 8 shows the response of the model using the initial parameter values listed in Table 1 together with the actual response

DRAFT

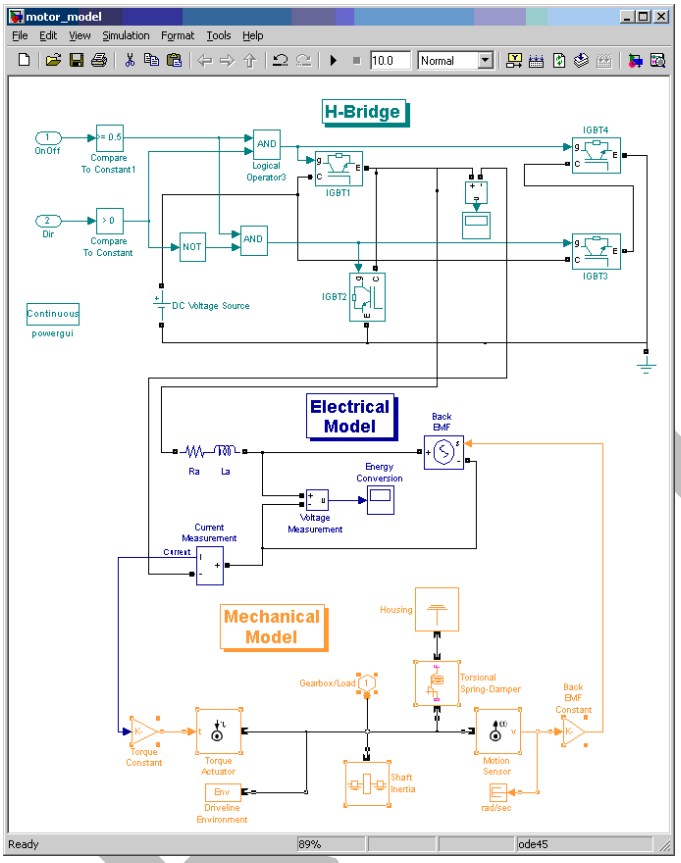


Fig. 7. Model of a DC motor and the signal conditioning hardware.

Table 1. Estimated parameter set.

Name	Parameter	Value	Unit
Viscous Friction	B	0.008	Nms/rad
Shaft Inertia	J	$5.7e^{-7}$	kgm^2
Motor Constant	Km	0.0134	Vs/rad
Armature Inductance	La	$6.5e^{-5}$	H
Armature Resistance	Ra	1.9	Ohm

of the motor. To obtain a more accurate response, the parameters must be re-estimated. This is where Simulink® Parameter Estimation plays a pivotal role.

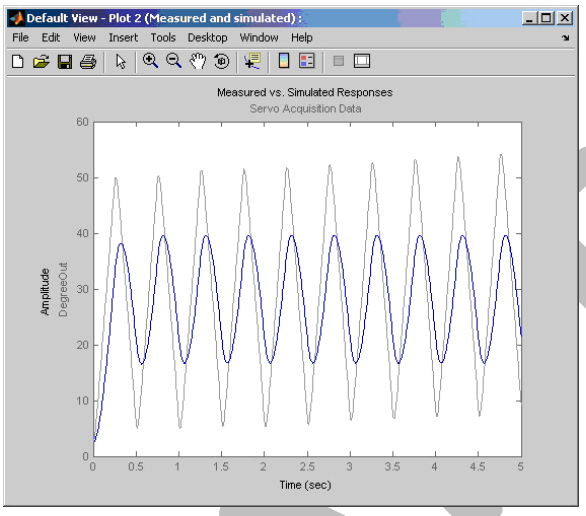


Fig. 8. Actual response and model response *before* fine-tuning the parameter estimates.

Input/output data from the motors can be applied to tune parameters in the model until the computational model mimics the behavior of the real robotic arm with sufficient precision. The typical workflow is as follows:

- Connect a voltage source to a motor on the robotic arm.
- Input a voltage to the motor and read the resulting position sensor values, and save these as input/output datasets.
- Repeat this with different types of input such as steps, ramps, and frequency sweeps.
- Select the parameters to tune.
- Some datasets will be used to tune the parameters while others can be used to validate the tuned values.
- Using the input/output data sets, tune the parameters until the input produces an output that matches the actual robotic arm with the desired precision.
- For validation, simulate the model with an input from one or more datasets that were not used for tuning.
- Compare the computational output to the output from the validation dataset. If they match within reasonable tolerances, the model is sufficiently tuned for design.

DRAFT

Figure 9 shows the comparison of the outputs from the computational model to the robotic arm outputs after tuning. Note that the results overlap closely. At this point, the model can be used to design the controller, for example, by employing classical, modern, and robust control design approaches [13, 19, 26, 29, 42].

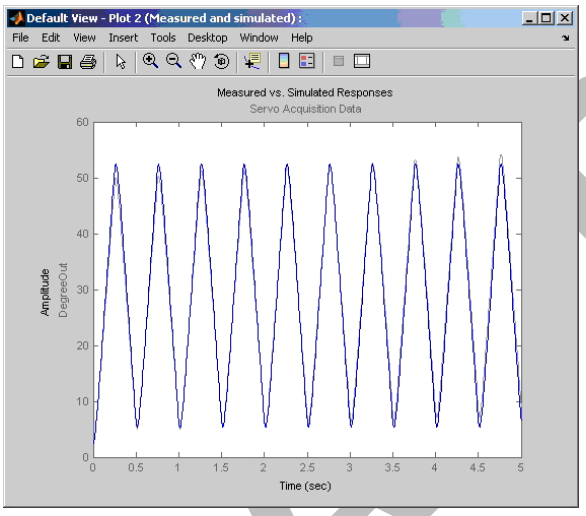


Fig. 9. Actual response and model response *after* fine-tuning the parameter estimates.

5.4 Generating the Motor Models from Data

In some cases, modeling the individual components of the overall system in detail may not be of great interest. Though it is of value when, for example, designing the DC motor configuration, for the design of the motor controller it may not be critical. In case of designing the controller, the entire motor model can often be treated as a black box. A linear model of the motor can be estimated using the same datasets used in Section 5.3. This approach works well when the system being modeled is relatively linear like the motor used here. Using the System Identification Toolbox, a model can be identified using a similar workflow as shown in Section 5.3.

- Connect a voltage source to one motor on the robotic arm.
- Input a voltage to the motor and read the resulting position sensor values, and save these as input/output datasets.
- Repeat this with other input such as steps, ramps, and frequency sweeps.
- Some of the inputs can be used to identify the model while others are used to validate the model.

DRAFT

- Using the input/output data sets, identify the model until the input produces an output that matches the robotic arm.
- For validation, use input from a data set that was not used for identification.
- Compare the computational output to the output from the data set. If they match within required tolerances, the model is sufficiently accurate for design.

6 Using Computational Models for Control Design

The preceding sections have investigated how to obtain accurate dynamic models of physical devices. As stated earlier, one important reason to develop accurate computational models of physical processes and devices is to facilitate the design of control laws for regulating or controlling their behavior.

6.1 Designing Controllers Through Modeling and Simulation

The most basic approach to designing a control system need not rely on computational models, but rather can be developed through paper and pencil analysis, design, and iteration on the real systems that will be controlled. This approach normally requires a lot of experience, and is typically complex to implement. Additionally, it is more costly to implement because of the resources needed to mitigate the risks of damaging the system and actually making the physical system available for design work. Some benefits of using computational models and simulation to design controllers are:

- Ability to use specialized simulation-based control algorithm development tools.
- Ability to evaluate hardware and system constraints such as actuator effort and response time in a safe, low cost environment.
- Increased innovation through the ability to experiment with many possible solutions before implementation.

The actual process to design controllers in a modeling and simulation environment varies depending on the tools available. This section describes the approach used to design controllers in a simulation environment and also studies the use of some specialized tools that aim to simplify the design process. The typical process, which will be described in detail in the subsequent subsections, is:

- Model the dynamics of the device or process being controlled (see Section 5 for the modeling of the robotic arm).
- Obtain a linear representation of this 'plant' model about the relevant operating points.

DRAFT

- Use linear control design techniques to tune the controllers to meet performance requirements.
- Validate the linear control design on the nonlinear computational model.

An alternative to this process is to use tools that depend on simulation and optimization-based techniques. For the design of a controller for the robotic arm, a set of tools called Simulink® Control Design [34], Control System Toolbox [8] and Simulink® Response Optimization [35] are used. A single workflow-based graphical user interface (GUI) serves as a task manager and portal to this set of tools.

In the case of the robotic arm, the control task is to move the hand to a desired point in space. This is done by manipulating the arm segments of the robot arm. The exact control design goals are as follows:[2]

- Robot arm position control:
  - Design joint angle controllers for the turntable, bicep, forearm, and hand joints.
  - Design pre-filters to balance the bandwidths of the responses to reduce the impact of off-diagonal closed-loop responses.
- Joint angle loop control requirements:
  - The bandwidth is less than 50Hz.
  - The gain margin is greater than 20db.
- Closed loop position control step-response requirements:
  - The overshoot is less than 10%.
  - The rise time is less than 1.5 seconds.
  - The cross-coupling is less than 10%.

The model developed in the previous sections is used as a starting point. It will need to be linearized to investigate how controllers can be designed to meet the specified requirements.

6.2 Linearizing Models for Control System Design

With the robotic arm, the first step is to obtain a linear representation of the nonlinear model. The linearized model is then used to compute pertinent open-loop and closed-loop response plots that are used directly in control design. To obtain the linearized model, the following steps need to be taken:

- Specify the control structure in Simulink, with feedback loops, compensators and pre-filters as shown in Fig. 10.
- Select the eight compensators to be tuned which include four pre-filters and four feedback controllers.
- Select the closed-loop inputs and outputs which map the desired position to actual measured position.

[2]Detailed definitions and descriptions of these terms can be found in the on-line documentation of the Control System Toolbox [8], http://www.mathworks.com.

DRAFT

- Specify or compute the operating points for the linear analysis of the model.

Linear models are then automatically extracted using the information from these steps. These linear models are used to setup the control design task in the GUI.

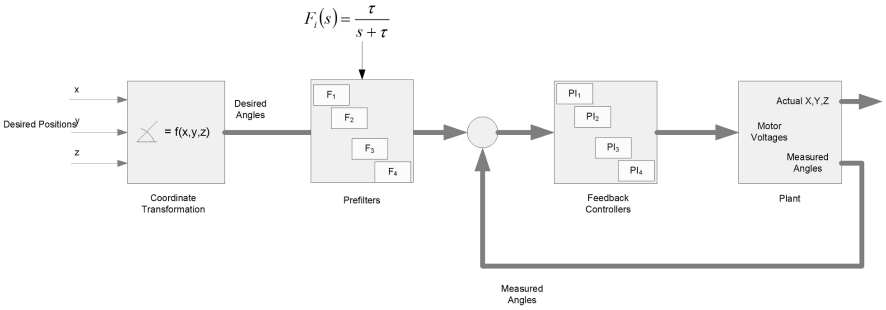


Fig. 10. Feedback loop structure.

The open-loop and closed-loop linearization results are highly dependent on the operating point–for example the states of the integrators of the model at different operating points. Trim or equilibrium operating points are a special type of operating point that engineers find very useful. A basic description of equilibrium conditions is that, over time, the operating point remains steady and constant. In Simulink and other block diagram simulation tools there are two commonly used approaches to specifying equilibrium conditions of a model of the physical system. The first method is that the users employ their intuitive knowledge about the system to pick an equilibrium condition. This can be a rather time-consuming and difficult process because of the large number of operating points that must be specified in a complicated model.

The second option is to employ an approach known as trim analysis. The approach uses optimization to solve for a set of operating points that satisfy the equilibrium conditions. Simulink® Control Design provides trim analysis capabilities to obtain initial conditions for various operating points. Another alternative is to use 'simulation snapshots' to specify operating points close to the region where the control effort is desired.

In the case of the robotic arm the model was linearized at a number of operating points for different positions of the arm. A single operating point was selected for the design and the other operating points were used to verify the control system of the robot arm in different configurations.

DRAFT

6.3 Designing A Controller

Once a linearized open-loop model has been obtained, a typical next step is to select a control system structure and tune the individual compensators. In the case of designing a controller for the robotic arm, the control structure is specified in the earlier step to help the tool determine the linear representation. The robotic arm controller configuration consists of four feedback loops with pre-filters as shown in Fig. 10. For such a multi-loop system, several input/output combinations have to be linearized to attempt to design multiple controllers such that the overall multi-loop system meets controller performance requirements. Multi-loop controller design can be approached in a number of ways:

- Sequential loop closure where the designer first tunes one loop with the others open, and then sequentially closes the other loops.
- Use traditional MIMO design techniques such as H-infinity to tune the loops simultaneously.
- Simultaneously tune the coupled SISO control loops.

For the robotic arm, the GUI is used with the linearized model to setup a control design task to design the controllers. The steps are:

- Select design plots such as root locus, Bode and Nichols charts for each loop that needs to be designed.
- Select closed loop analysis plots for viewing.
- Use design plots (Fig. 11) to graphically shape all loops and edit the compensator structure, while viewing loop interactions and closed loop responses in real time.

Because the tools can compute loop interactions while tuning all loops, the approach is to use visualization to see how changing one compensator affects all the other responses that are of interest. The graphical design tools can be employed in this way to design a set of compensators to try to best meet the performance requirements. Additionally, optimization techniques within these graphical tools can be exploited to help tune existing controllers, while trading off between multiple design requirements.

6.4 Tuning Controller Designs Using Optimization Techniques

The described graphical design tools simplify tuning a multi-input multi-output (MIMO) controller by enabling simultaneous tuning of loops. However, the process can still be quite complex; especially for systems with many tightly coupled loops. Because a computational model is available, this process can be simplified by leveraging optimization techniques to automate multi-loop tuning. Using Simulink® Response Optimization, performance bounds and requirements can be specified on the design and closed loop response plots

DRAFT

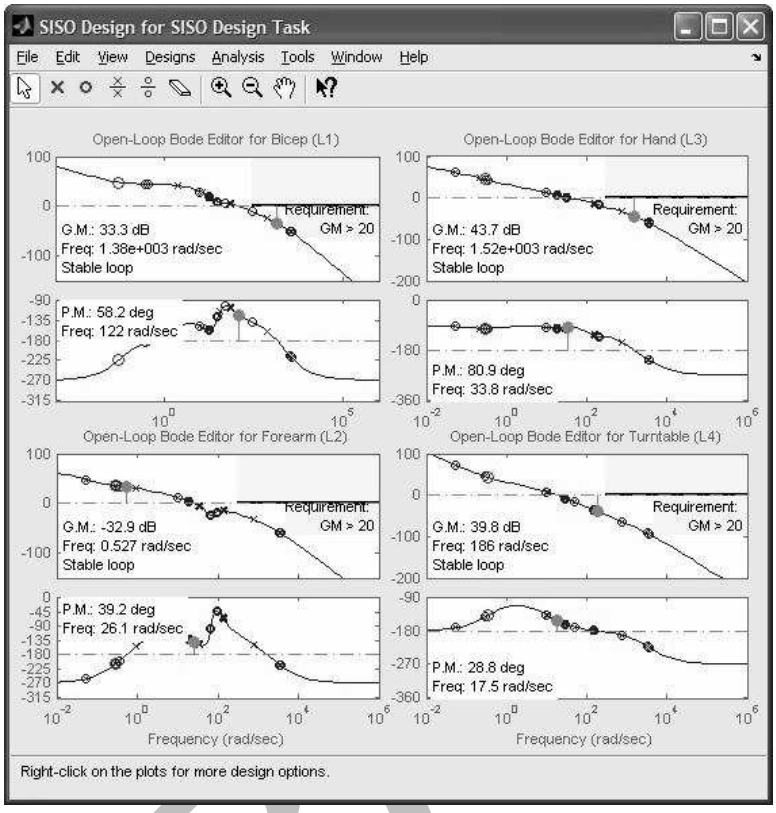


Fig. 11. SISO design plots.

and then several compensators can be tuned simultaneously through optimization. This is done by graphically specifying requirements in either the time domain; as overshoot, settling time, etc., or in the frequency domain for example as gain/phase margin, bandwidth, or pole zero locations.

For the robotic arm the performance requirements listed above are specified for the controller by using a combination of both time-domain and frequency-domain plots. The optimization is then run to obtain a valid solution. Figure 12 shows the final responses obtained and how these fit within the performance envelopes that are defined. Note that although the system being tuned is a MIMO system, the problem is configured as eight coupled SISO controllers that are tuned simultaneously. In the event that the optimization is not able to meet all requirements, the requirements can be relaxed, or, alternatively, a different control strategy can be evaluated using all the methods that have been discussed. Once satisfactory control performance is obtained, the design can be validated on a full nonlinear simulation by exporting the controller gains directly to the Simulink model.

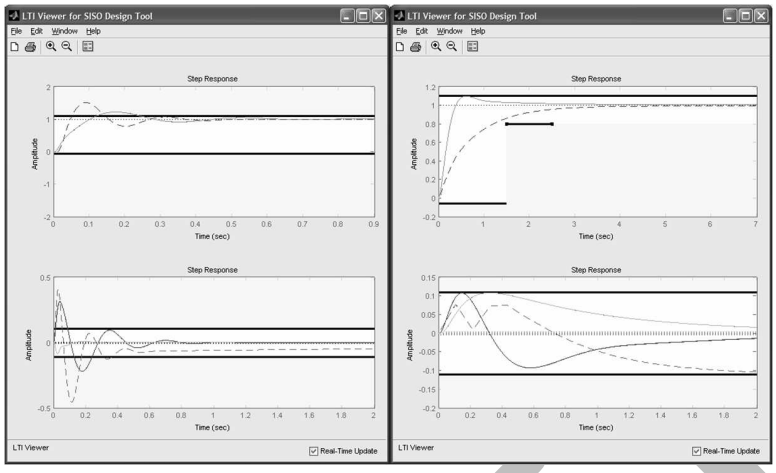


Fig. 12. Closed loop responses before (left) and after (right) optimization.

7 Testing with Model-Based Design

One aspect of Model-Based Design is the testing of designs while they are under development. By testing early in the design phase, errors and deficiencies may be recognized and rectified early in the design phase, before the cost of correction becomes too high, or worse, the error makes it into the final implementation. Testing early and often is a good principle, and it is imperative to do so in a systematic manner.

7.1 Requirements-Based Testing Through Simulation

Requirements-based testing, which can contribute to systematic testing, refers to defining test vectors for each requirement, and creating corresponding checks to verify that the requirements are met. Essentially a test harness is created to verify that the design algorithm meets the requirements. This workflow is shown in Fig. 13.

The test vectors typically consist of a series of inputs designed to exercise the computational model through a range of expected and unexpected behavior. In case of the latter, this enables testing of dangerous modes of operation, and the corresponding design of fail-safe elements. Often this kind of failure-mode testing in simulation is impossible to do with a physical prototype or implementation of the system. In addition to test vectors, verification blocks in Simulink can be used to check model output data ranges and ensure that requirements are met. Once a test harness with test vectors and verification blocks has been obtained, a coverage report can be generated to understand how well the algorithm was exercised. Using Simulink® Verification and Validation [36], coverage metrics are collected as the tests execute to

DRAFT

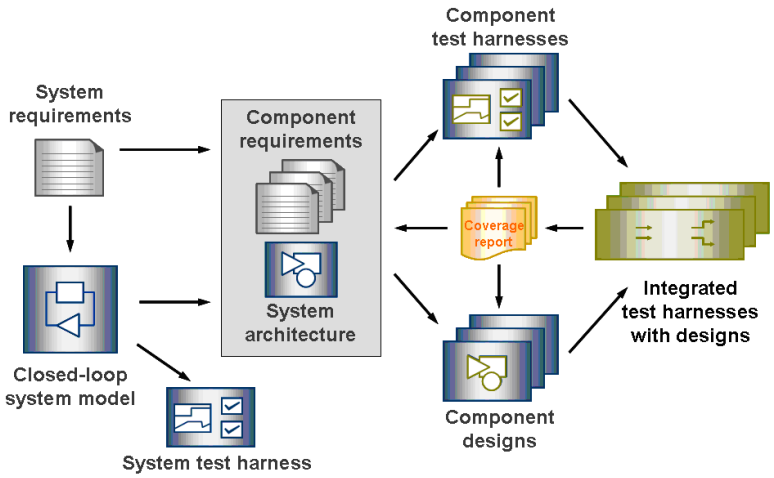


Fig. 13. An overview of Model-Based Design.

quantify which elements of the design have been excited and which have not. Using coverage-based test verification, the following features and benefits are achieved:

- Measure how well the model has been exercised.
- Identify what additional test vectors are needed to exercise the model more thoroughly.
- Identify and remove unnecessary elements in the design.
- Ensure that the requirements, design, and tests are consistent and complete.

7.2 Simulation with Hardware and Implemented Designs

In addition to the use of computational models for simulation during the design stages, models can also be used in real-time simulations for verification and validation. In such a scenario, computer code is generated from the computational model and downloaded to dedicated computers to simulate the model in real time. This is often referred to as *rapid prototyping*, of which there are two variations. The first is *rapid controller prototyping*, which typically involves placing the controller algorithm on a processor such as a commercially available PC-based processor, and interfacing the processor with the plant, here the robotic arm. The entire system is then simulated in real time [22].

The second variation is *hardware-in-the-loop* (HIL), which refers to interfacing the processor on which the controller runs with a combination of real hardware and computational models, and running in real time on dedicated processors. HIL enables the integration of difficult-to-model hardware as part

DRAFT

of the simulation environment. For example, actuators can be highly nonlinear, and so if a model of such an actuator is used, the analysis may not be sufficiently precise. HIL allows the model of the physical system minus the actuator to be connected to the real actuator and hence enables the full nonlinear behavior to be validated.

In the case of the robotic arm, rapid controller prototyping is used with xPC Target [41]. Code is generated from the controller algorithm using Real-Time Workshop® [27]. Next, using a commercially available PC running the real-time kernel of xPC Target, the controller algorithm is downloaded onto the PC and commercially available I/O boards are used to connect the PC to the robotic arm. The control algorithm running on the PC is then used to control the robotic arm and validate the algorithms. At this point, the controller algorithm can be tuned in real time, while connected to the robotic arm. Alternatively, the algorithm can be modified and code regenerated to test the new algorithm. This process is repeated till the algorithm operates satisfactorily.

Reasons to perform rapid controller prototyping include:

- Quick algorithm testing and retesting using hardware that has been tested in simulation.
- Testing control algorithms with fixed-step solvers in real time, which is closer to real-world implementations.

To briefly study an example of HIL, consider a flight-control application with the flight controller implemented on the actual flight-control box, a dedicated computer that will go into the production aircraft. The flight-control box could then be integrated with a cockpit, a human pilot, and a flight-simulator package. The cockpit would enable the pilot to provide realistic inputs, while the flight-simulator could be running the aircraft dynamics computational model (that have been modeled using a combination of the three modeling methods) to validate the behavior of the controller.

The flight simulator could be mounted on a motion-simulator platform to generate the appropriate forces and torques. Models of wind and turbulence and atmospheric effects can be included to test the behavior of the controller in 'dangerous' situations without risk to the pilot or the aircraft. This can be implemented through the use of xPC Target and models generated with Simulink and the physical modeling products such as SimMechanics. This approach is very useful in safety-critical applications such as those found in aircraft and other vehicles, where testing with the real hardware is expensive, time-consuming, and often heavily regulated.

7.3 Other Uses of Rapid Prototyping in the Design Process

To design a control system, rapid prototyping tools can be used as shown in Fig. 14 for other stages of the process including the actual computational modeling. At the start of the control design process, an engineer may have

DRAFT

a rather inaccurate model or no model at all. So, at first a skeleton control system is developed to stabilize a system and to get the desired behavior to experiment with. Once this is achieved, experiments can be designed and performed to acquire responses of the system at various operating conditions. The acquired data can then be exploited to enhance the plant model, and to design a new control system using the more accurate plant model. Simulation of the combined control system and plant model then allows studying the performance of the system and the control system can be optimized using the full nonlinear plant simulation model. Finally, the control system can be implemented on a rapid prototyping system. If the system does not meet the performance of the control system as obtained in simulation, the model is further refined as well as the design of the control system to try to achieve improved performance.

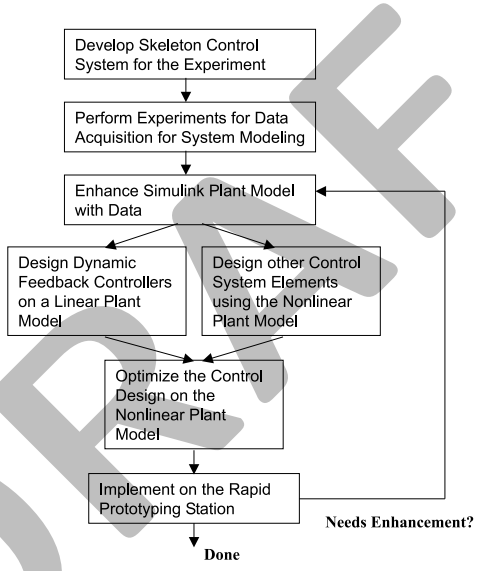


Fig. 14. Rapid prototyping control design process.

DRAFT

8 Conclusions

Through the use of advanced computational modeling and numerical simulation capabilities available in Simulink, an accurate model of the robotic arm can be quickly obtained and fine-tuned with measured data. Control algorithms for the robot arm were then designed and tested rapidly and effectively. Using requirements-based testing and rapid controller prototyping each requirement was systematically tested and the entire set of requirements

was formally verified. When the control algorithm was finally implemented its proper behavior to control the actual robotic arm was established with confidence. It was thoroughly analyzed to deliver the required performance, even under fault conditions.

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks and SimBiology, SimEvents, and SimHydraulics are trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.

References

1. Rob Aberg and Stacey Gage. Strategy for successful enterprise-wide modeling and simulation with COTS software. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. Providence, Rhode Island, August 2004. CD-ROM.
2. Karl J. Åström and Björn Wittenmark. *Computer Controlled Systems: Theory and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.
3. Paul Barnard. Graphical techniques for aircraft dynamic model development. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*. Providence, Rhode Island, August 2004. CD-ROM.
4. Julius S. Bendat and Allan G. Piersol. *Random Data: Analysis & Measurement Procedures*. Wiley-InterScience, Hoboken, NJ, 2000.
5. Arno P. J. Breunese, Theo J. A. de Vries, Job van Amerongen, and Peter C. Breedveld. Maximizing impact of automation on modeling and design. In *ASME Dynamic Systems & Control Div. '95*, pages 421–430, San Francisco, CA, 1995.
6. Jan F. Broenink. *Computer Aided Physical Modeling and Simulation: A Bond Graph Approach*. PhD dissertation, University of Twente, Enschede, The Netherlands, 1990.
7. F.E. Cellier, H. Elmqvist, and M. Otter. Modelling from physical principles. In W.S. Levine, editor, *The Control Handbook*, pages 99–107. CRC Press, Boca Raton, FL, 1996.
8. Control System Toolbox. *Control System Toolbox User's Guide*. The MathWorks, Natick, MA, March 2006.
9. S.J. Culley and A.P. Wallace. The modelling of engineering assemblies based on standard components. In John Sharpe and Vincent Oh, editors, *Computer Aided Conceptual Design*, pages 113–129, Lancaster, United Kingdom, April 1994. ISBN 0-901800-37-6.
10. Richard C. Dorf. *Modern Control Systems*. Addison Wesley Publishing Co., Reading, MA, 1987.
11. Simulink Parameter Estimation. *Simulink Parameter Estimation User's Guide*. The MathWorks, Natick, MA, March 2006.
12. Hilding Elmqvist *et al.* Modelica$^{\text{tm}}$–a unified object-oriented langauge for physical systems modeling: Language specification, December 1999. version 1.3, http://www.modelica.org/.
13. Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, Englewood Cliffs, NJ, 2002.

DRAFT

14. Fuzzy Logic Toolbox. *Fuzzy Logic Toolbox User's Guide*. The MathWorks, Natick, MA, March 2006.
15. Hydro-Queébec TransÉnergie Technologies. *SimPowerSystems User's Guide*. The MathWorks, Natick, MA, March 2006.
16. D.C. Karnopp, D.L. Margolis, and R.C. Rosenberg. *Systems Dynamics: A Unified Approach*. John Wiley and Sons, New York, 2 edition, 1990.
17. Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, Englewood Cliffs, NJ, 2 edition, 1998.
18. P. Lötstedt. Coulomb friction in two-dimensional rigid body systems. *Z. angew. Math. u. Mech.*, 61:605–615, 1981.
19. J.M. MacIejowski. *Multivariable Feedback Design*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989. Electronic Systems Engineering Series.
20. MATLAB. *The Language of Technical Computing*. The MathWorks, March 2006.
21. Pieter J. Mosterman and Gautam Biswas. A hybrid modeling and simulation methodology for dynamic physical systems. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 178(1):5–17, January 2002.
22. Pieter J. Mosterman, Sameer Prabhu, Andrew Dowd, John Glass, Tom Erkkinen, John Kluza, and Rohit Shenoy. Embedded real-time control via matlab, simulink, and xpc target. In Dimitrios Hristu-Varsakelis and William S. Levine, editors, *Handbook on Networked and Embedded Systems*, pages 419–446. Birkhäuser, Boston, MA, 2005.
23. Pieter J. Mosterman, Sameer Prabhu, and Tom Erkkinen. An industrial embedded control system design process. In *Proceedings of The Inaugural CDEN Design Conference*, pages CD–ROM, Montreal, July 2004.
24. Pieter J. Mosterman and Hans Vangheluwe. Computer automated multiparadigm modeling: An introduction. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 80(9):433–450, September 2004.
25. Neural Network Toolbox. *Neural Network Toolbox User's Guide*. The MathWorks, Natick, MA, March 2006.
26. Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall Inc., Englewood Cliffs, NJ, 4 edition, 2001.
27. Real-Time Workshop. *Real-Time Workshop User's Guide*. The MathWorks, Natick, MA, March 2002.
28. Signal Processing Toolbox. *Signal Processing Toolbox User's Guide*. The MathWorks, Natick, MA, March 2006.
29. Ian Postlethwaite Sigurd Skogestad. *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, Inc., New York, 1996.
30. SimDriveline. *SimDriveline User's Guide*. The MathWorks, Natick, MA, March 2006.
31. SimHydraulics. *SimHydraulics User's Guide*. The MathWorks, Natick, MA, March 2006.
32. SimMechanics. *SimMechanics User's Guide*. The MathWorks, Natick, MA, March 2006.
33. Simulink. *Using Simulink*. The MathWorks, Inc., Natick, MA, March 2006.
34. Simulink Control Design. *Simulink Control Design User's Guide*. The MathWorks, Inc., Natick, MA, March 2006.
35. Simulink Response Optimization. *Simulink Response Optimization User's Guide*. The MathWorks, Inc., Natick, MA, March 2006.

DRAFT

36. Simulink Verification and Validation. *Simulink Verification and Validation User's Guide*. The MathWorks, Inc., Natick, MA, March 2006.
37. SolidWorks. *Introducing SolidWorks*. SolidWorks Corporation, Concord, MA, 2002.
38. System Identification Toolbox. *System Identification Toolbox User's Guide*. The MathWorks, Natick, MA, March 2006.
39. Michael M. Tiller. *Introduction to Physical Modeling with Modelica*. Kluwer Academic Publishers, Boston, 2001. ISBN 0-7923-7367-7.
40. Giles D. Wood and Dallas C. Kennedy. Simulating mechanical systems in simulink with simmechanics. Technical Report 91124v00, The MathWorks, Inc., Natick, MA, 2003.
41. xPC Target. *xPC Target User's Guide*. The MathWorks, Inc., Natick, MA, March 2004.
42. Kemin Zhou and John C. Doyle. *Essentials of Robust Control*. Prentice Hall Inc., Englewood Cliffs, NJ, 1997.

DRAFT