Design and Verification of Motion Control Algorithms Using Simulation

Douglas Eastman, Paul Lambrechts, Arkadiy Turevskiy

The MathWorks, Inc. Natick, MA, 01760

Identifying system level problems early and ensuring that all design requirements are met are two of the top challenges faced in developing mechatronic systems (1). This paper will illustrate how simulation allows the verification of the system performance throughout the development process, making it possible to identify system level problems early and optimize system level behavior to meet the design requirements. This methodology, known as Model-Based Design, will be applied to a classic motion control application where a load must be precisely positioned through a flexible shaft.

I. Introduction

With the accelerating pace of technology development, companies are in a race to be on the cutting edge. A January 2008 survey of 160 electro-mechanical equipment manufacturers found that the top two drivers for improving development processes were shorter product development schedules and increased customer demand for better performing products (1). These two goals may seem contradictory; increasing performance would necessitate a longer development cycle. So how can we attempt to achieve both of these goals at the same time?

Challenge	Response
Difficulty finding and hiring experienced system engineers / lack of cross-functional knowledge	50%
Early identification of system level problems	45%
Ensuring all design requirements are met in the final system	40%
Difficulty prediction / modeling system product behavior until physical prototypes exist	32%
Difficulty implementing an integrated product development solution for all disciplines involved in mechatronic product development	28%
Inability to understand the impact a design change will have across disciplines	18%

Table 1: These are the top six challenges of mechatronic product development according to a recent survey (1).

Looking more specifically at some of the challenges that are inhibiting mechatronic product development (products that involve mechanical, electrical, control, and embedded components), we can group them into two general categories. Table 1 presents the results from a recent survey of companies doing mechatronic product development. Half of the challenges, such as “lack of cross-functional knowledge,” deal with issues associated with the multi-domain nature of the complete system. With engineers traditionally working only in their area of expertise, this survey found that there are problems when it comes to integrating the domains together and dealing with the complete system. The other half of the challenges focus on testing, such as finding errors early in the development cycle and testing before hardware is available. The importance of early testing can be further illustrated by a NASA study that analyzed the relative costs of fixing errors based on what

phase of development they were introduced in and what phase they were first detected (2). From Figure 1, it is evident that detecting errors early in the design cycle, or as soon as possible after they are introduced, can have a dramatic impact on the cost of a project.

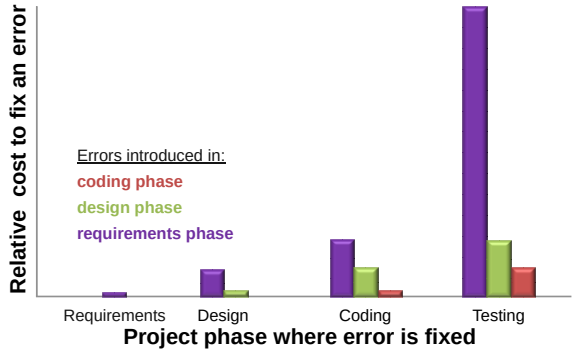

Figure 1: Graph of the relative cost to fix an error based on project phase (2).

To help address the challenges of multi-domain design and early testing for motion control applications, this paper describes how to use Model-Based Design to perform system-level simulation to combine multiple domains such as electrical, mechanical, hydraulic, and control in a software environment where testing can be done throughout the design process.

To illustrate Model-Based Design, we will use an example of a precision motion system that requires a load to be moved from one position to another and back to the original position in a certain amount of time. The system consists of a DC motor driving a load through a flexible shaft. This mimics a drive system as you may typically find in many sorts of mechatronic machinery. The original design moves at a maximum speed of 150 rad/s and a maximum acceleration of 2000 rad/s^2. The goal is to increase the speed to 250 rad/s and the acceleration to 5000 rad/s^2 without losing any position accuracy. This will result in an increased throughput of the larger system. Simply increasing these parameters in the current system leads to unacceptable position accuracy as shown in Figure 2. The settling time for the faster move is almost identical to the slower move (around 1.5 seconds). So even with the higher velocity and acceleration, the move time is approximately the same. To improve the design we will first develop a model of the physical system including the DC motor and transmission shaft and then investigate improving the control algorithm. Finally we will see how we can test and implement the new controller design with the physical system.

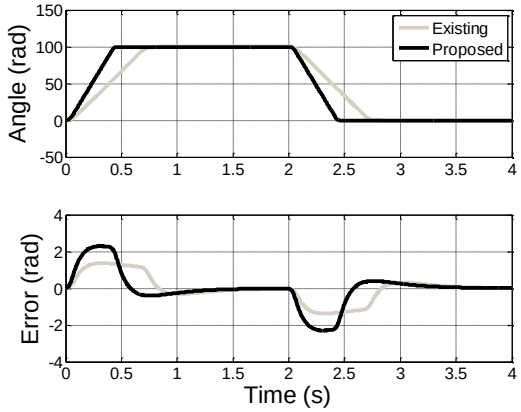

Figure 2: Plot of the position and error in position when making a point to point movement. The original performance is shown in gray; the performance for a more aggressive move is shown in black.

This paper will explore many advantages that a system simulation in a software environment provides including: helping to understand the behavior of the system, optimizing design parameters, developing optimal control algorithms, testing control algorithms, and qualifying the production controller before connecting it to the real plant. Leveraging these advantages of Model-Based Design ultimately results in shorter design cycles while simultaneously improving product performance. For this we will use a software

environment developed by The MathWorks, based on the well-known tools MATLAB® and Simulink®. The results in this paper are based on release 2009a of this environment.

II. Plant Modeling

The plant, or the physical system we are trying to control, is pictured in Figure 3. It consists of a power amplifier driving a DC motor with two rotary optical encoders for measuring the position of the motor shaft and the load. The motor is connected to the load through a small flexible shaft to approximate the compliance found between the actuator and the load in many motion control systems.

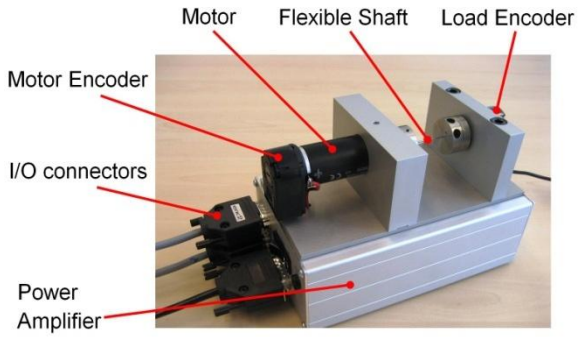

Figure 3: Picture of the plant that is being controlled.

In order to improve the controller performance, which is the ultimate goal, we must first develop a model of the plant to use when designing the controller. There are two primary approaches for creating a model of a system or a component in a system: data-driven and first-principles.

Data-driven modeling involves generating input signals to send into the actual system and then measuring the resulting output. These measurements can be used to derive a dynamic representation of the system, such as a transfer function. Because it requires measured data, one limitation of this approach is that it cannot be used before the actual system exists. But because it uses data and does not require any insight into how the system is constructed, it can be a quick way to develop a model with a clear correspondence to the real system. Because the model is not based on the underlying system components, the parameters in the model have no connection to physical model parameters, such as the stiffness of the transmission shaft. This so-called black-box model can be used to help design and test control algorithms, but cannot be used to investigate making changes to the plant design.

First-principles modeling involves building up a model based on the individual components in the system and their behavior. It generally takes more time than the data-driven approach, but provides insight into the plant and how various parameters can affect the overall system behavior. For a new design with no physical prototype, this is the only available path.

In the motion control example, we are improving the performance of an existing system, so we will start by using the data-driven approach to develop a model. Then we will investigate what a first-principles modeling approach would be for the same system.

Data-Driven Modeling "The Existing System"

There are several approaches for doing data-driven modeling including neural-networks, optimization, and system identification. Working in MATLAB® gives you quick access to these different approaches. For the purposes of this motion control example, we will use linear system identification algorithms that use measured data to identify models of the following form:

$$y = Gu + He$$

G describes the system dynamics from the input *u* to the output *y*. *H* describes the output disturbance and is called a *noise model* (3).

To carry out the system identification and generate the model, we first need to collect data from the plant. If the plant that is being modeled exists on its own, arbitrary signals can be sent into the plant inputs, and the resulting plant outputs can be measured. It is important to construct the input signal such that it captures the plant dynamics of interest.

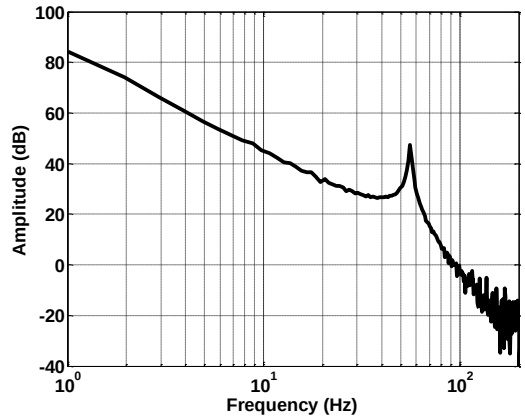

Figure 4: Measured frequency response of the open-loop plant.

In the case of the motion control example, the plant is in use in a larger system so we need to take measurements without disrupting normal operation. We could simply acquire data as its running, but to capture more complete dynamics over a range of frequencies, band-limited white noise with a sampling rate of 500 Hz is added to the voltage signal going into the motor and the total voltage going into the motor is measured. Using existing knowledge of the feedback control structure, these two values can be used to compute a frequency response of just the open loop plant, similar to what you might acquire from a spectrum analyzer as shown in Figure 4. We see one resonance peak associated with the compliance of the shaft that is around 55 Hz.

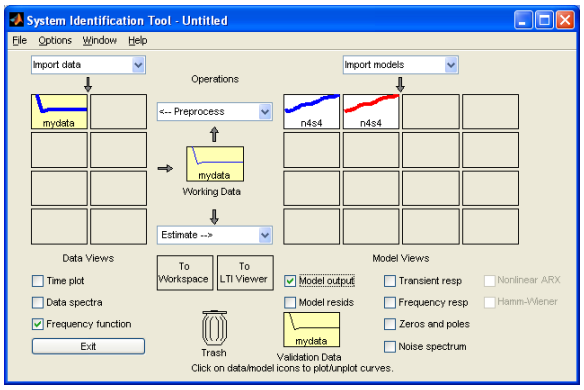

Figure 5: Graphical user interface for the System Identification Toolbox™.

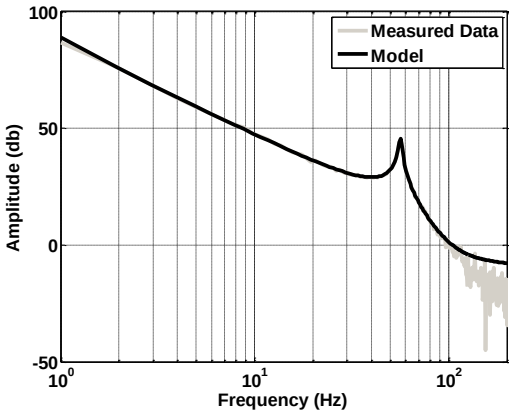

Figure 6: Measured frequency response and the model developed using system identification techniques show good correspondence over the frequency range of interest.

The frequency response data cannot be used directly in a time domain simulation, so next we convert it to a transfer function model using system identification. For this we used the System Identification Toolbox™, which allows you to do this in an interactive environment as shown in Figure 5. The frequency response data is imported into the tool, a model structure is chosen, and then the model estimate can be evaluated. For this example, a fourth-order state-space model was chosen. The model

frequency response and the measured frequency response are both plotted in Figure 6, which shows that the resonance peak in the estimated model closely matches the measured data.

This state-space model represents the dynamics of the plant, *G*, and can now be used directly in Simulink® to design and tune the controller.

First-Principles Modeling "The New Design"

Another approach to model the motion control system is to derive dynamic equations for the system based on the physical components. Assuming there is only one dominant resonance mode, the core dynamics of the transmission system can be simplified to the schematic in Figure 7.

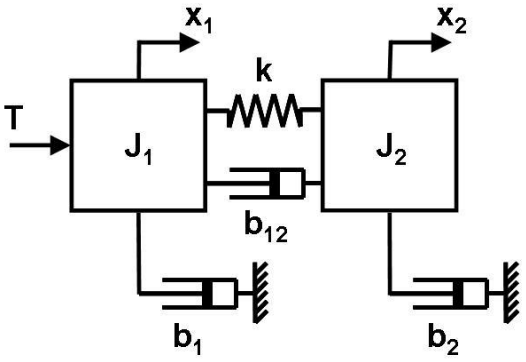

Figure 7: Schematic of the major mechanical components in the motion control plant.

T represents the torque from the motor, x_1, J_1 and x_2, J_2 represent the angular position and inertia of the motor and the load respectively, b_1 and b_2 represent the damping of the bearings, and *k* and b_{12} represent the stiffness and damping of the transmission shaft. For this simple system it is fairly straightforward to use Newton's laws to derive the equation of motion for the two bodies:

$$J_1 x_1'' = -b_1 x_1' - k(x_1 - x_2) - b_{12}(x_1' - x_2') + T$$
$$J_2 x_2'' = -b_2 x_2' + k(x_1 - x_2) + b_{12}(x_1' - x_2')$$

These equations can then be implemented in Simulink® as the block diagram given in Figure 8.

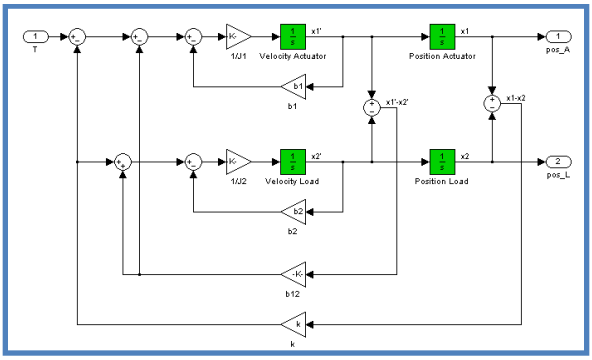

Figure 8: Simulink® model representing the dynamics of the flexible transmission shaft.

This block diagram then can be evaluated with different torque inputs to simulate the behavior of the system.

To have a model of the complete plant as seen from the controller, we also require a model of the electrical part of the system: the power amplifier and the motor coil. The power amplifier contains a high performance current control loop such that the voltage input acts as a reference for the applied motor torque. This allows us to neglect its dynamic behavior in relation to the mechanical dynamics and we can therefore treat it as a simple gain.

For this example, deriving the differential equations to describe the system dynamics was relatively straightforward. In many cases the systems are more complex and deriving those equations can be a challenging and time-consuming task. Another first-principles modeling approach is to leverage advances in modeling tools to build up the plant from basic physical component blocks. Rather than the signal-based blocks that were used to model

the differential equations, physical modeling blocks have energy conserving ports that represent physical connections.

Within Simulink® this technology is provided by Simscape™. It provides blocks for several different physical domains including mechanical, electrical, hydraulic, and thermal. There is also the ability to extend existing blocks or create blocks from scratch using an authoring language that involves describing the basic equations for that particular component. Using physical modeling tools, the model of the motion control system can be created directly from the schematic in Figure 7 without deriving any equations. The block diagram for the physical model of the motion control system is shown in Figure 9. Not shown in the figure are additional sensor and actuator blocks that you can connect to the physical components to transition from the signal domain of Simulink to the physical domain of Simscape and vice-versa. These translation blocks allow you to apply the control system defined in Simulink to the physical model in Simscape in a fully integrated environment.

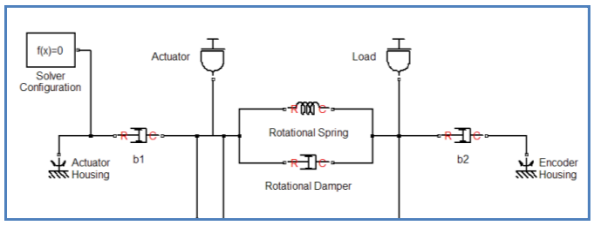

Figure 9: Physical modeling representation of the motion control plant using Simscape blocks.

Another option to obtain an accurate model of the mechanics of the motion system is provided by SimMechanics™. This method is especially useful if a CAD design of the system already exists. Using a CAD translator tool, which is available for a growing number of CAD environments, it is possible to automatically create a model of the rigid bodies in a system and connect them through appropriate joints. This saves even more time in developing the model and can also provide an animated visualization of the plant. Figure 10 shows an example of this for the motion control system considered here. After importing the CAD file, we can add a torque actuator and sensors to connect it to our control system. We also need to add damping to the two bearings (b_1 and b_2) as well as the stiffness and damping of the transmission shaft (b_{12} and k).

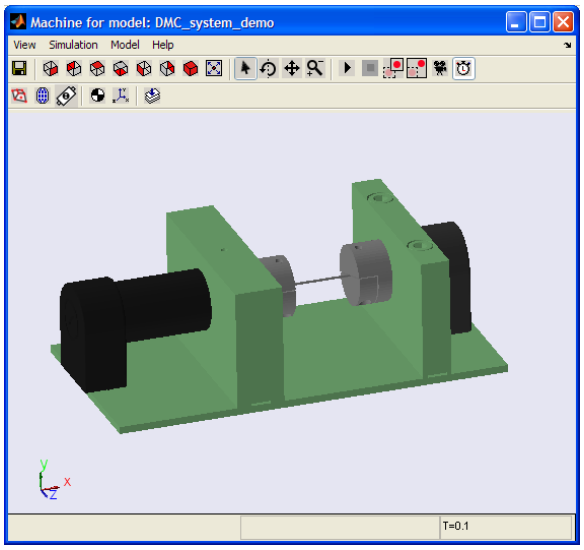

Figure 10: Visualization of the motion control system after importing from a CAD environment.

We have now looked at three different methods for creating a first-principles model: deriving differential equations, modeling using physical components, and importing from an existing CAD design. In all of these methods we are using parameters, like stiffness and damping, which correspond to real physical properties. How do we find the values for these parameters?

In some cases these values could come from a data sheet, they could come from direct measurements that have been made on the

system, or they could simply be guesses. The values provide a starting point for design work, but may not have the accuracy that is necessary for finely tuning a control loop. Once the physical plant is available, it may be helpful to use measurements from the system to validate the model. This ensures that your model provides a reasonable representation of the physical system and in the case of a disparity can be used to tune the parameters to provide a closer match.

This process, known as parameter estimation, involves acquiring data from the actual plant, choosing the parameters to estimate (specifying a bound if desired), and then using optimization algorithms to minimize the error between the simulated and measured responses. Using tools such as Simulink Design Optimization™ this process is largely automated.

In our example, we know all the parameters from our knowledge of the system except the stiffness and damping of the transmission shaft. So we can use measured data (similar to the data we used for data-driven modeling) to estimate those two parameters. We pick an initial guess and then use an optimization algorithm to find values of the two parameters to minimize the error between the simulated and measured data. After this process, we find the following values for the unknown parameters:

$$b_{12} = 2.0 \times 10^{-7}\,\mathrm{N{\cdot}m{\cdot}s/rad}$$
$$k = 1.3 \times 10^{-2}\,\mathrm{N{\cdot}m/rad}$$

Like the data-driven modeling approach, we now have a model of our plant that closely matches its actual behavior. But in addition, we have gained some insight into the physical properties of the plant. We could use this model to investigate how changes to the physical plant would affect our system response. For example, we could try increasing the stiffness of the transmission shaft to improve the positional accuracy of the load rather than changing the control algorithm.

III. Control Development

Once a plant model has been developed, the motion control algorithm can be designed or optimized. We will look at how to develop two different types of control algorithms. First we will discuss classic control algorithms where information from the sensors on the plant is fed back and used to calculate an actuator value based on a differential or difference equation. We will also investigate developing logic, or supervisory, controllers. These controllers usually consist of some heuristic algorithm that makes a decision based on time or measured data. For example, it may be used to schedule different motion profiles over the course of a production run. In our case, we will develop some logic to look for faults in our system and take an appropriate response.

Classic Control

Designing control systems involves first laying out the general control structure. Working in a graphical simulation environment makes it trivial to quickly try different control topologies by simply rerouting lines in the diagram. For example you might have an idea that adding a new sensor would provide valuable feedback to your controller. You can quickly test your theory by making that additional connection and evaluate the result before making the investment in the actual sensor.

In our example, the control structure already exists and we are just interested in improving it, so the first step is to model the existing controller. It consists of both a feed-forward controller and a feedback control. The feed-

forward control calculates a nominal voltage to send to the motor based on the desired trajectory (4). Feedback control on the other hand relies on measurements of the load position to adjust the voltage going to the motor and can compensate for disturbances such as a varying load. For the purposes of this example, we will focus on how to improve the current feedback control system and leave the feed-forward system as is.

The original feedback control consists of a single discrete PD controller running at a sampling rate of 500 Hz (5). The input to the controller is the error in the position and the output is a commanded voltage which is added to the feed-forward voltage. Given this information about the original structure of the feed-forward and feedback controllers, we can construct a complete closed-loop model of our system as shown in Figure 11.

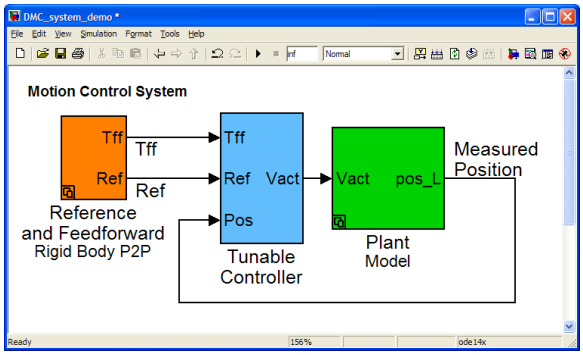

Figure 11: Model of the complete closed loop system.

With this complete system model we can now visualize the change in performance as the controller design is altered. While tuning the closed-loop response, we will look at the response of the load position with a step change in the reference position, disabling the feed-forward control. First we will investigate adding a notch filter to the feedback path to reduce the effect of the resonance peak we discovered while modeling the system, then we will optimize the PD controller gains to further improve the performance of the system.

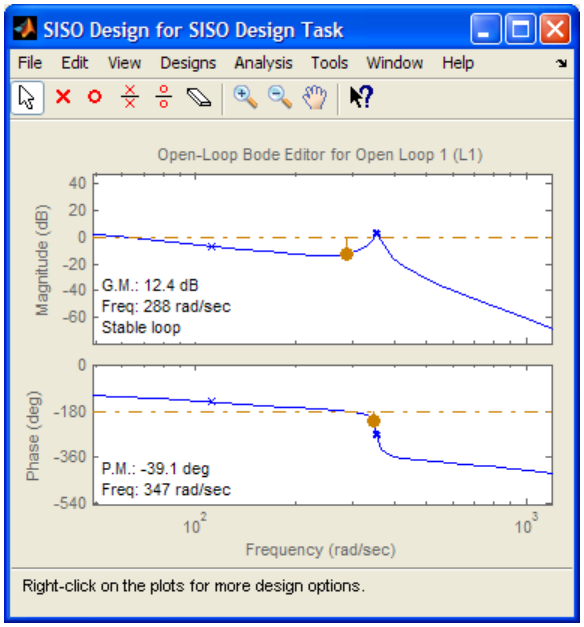

Figure 12: Interactive open-loop bode plot used for designing the controller.

To interactively add complexity or design the structure of a controller from scratch, we can use control design tools that will analyze the model and provide useful information for quickly trying different control strategies. In our case, to add a notch filter, we can first add an arbitrary transfer function that will ultimately represent the filter. Using Simulink Control Design™ we can then display a plot of a linearized version of the open loop system response as shown in Figure 12. We have chosen to use a bode plot, but the root-locus or Nichols plot of the open-loop system could also be displayed.

We can see the resonance peak that is limiting the feedback gain of the system. From within this plot, we can interactively modify our controller by adding poles, zeros, or different filters. As soon as these components are added, the open loop bode response will change to

reflect the new design. At the same time, you can see the resulting change in the closed-loop system response to understand the impact the changes have on the full system. In our case, we place a notch filter centered right on the resonance peak which lowers the magnitude of the peak and reduces the oscillation in the closed-loop step response of the system.

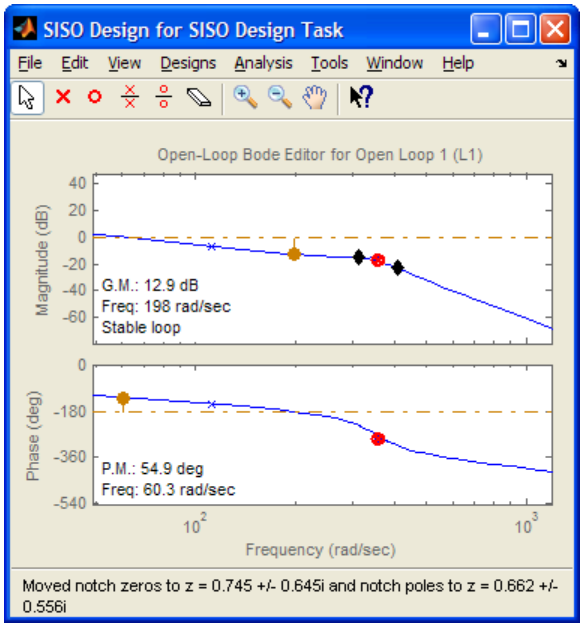

Figure 13: Open-loop bode plot with the notch filter interactively added.

The updated open-loop bode plot is shown in Figure 13. The red circle is the location of the notch filter and the black diamonds represent the width of the filter.

After adding the notch filter, the closed-loop step response has reduced oscillations associated with the resonance mode as shown in Figure 14.

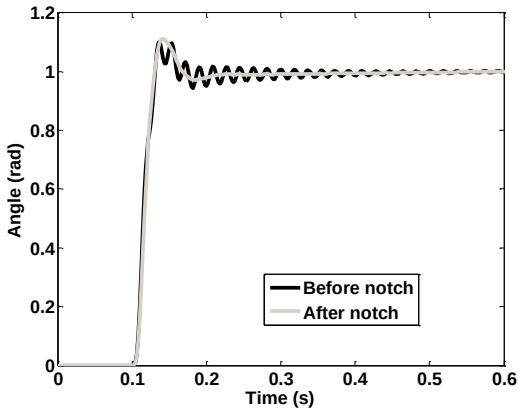

Figure 14: Closed-loop step response before and after adding the notch filter.

Design Optimization

Next we can try to further improve the step response by tuning the PD controller gains. We could do this in a similar manner as the notch filter design, but as an example we will use a different approach based on numerical optimization. Since we are in a simulation environment, we can run many simulations quickly and use optimization algorithms to adjust the parameters to meet a specific goal. We can define the goal as a time-based requirement for one or more signals in the model. In our case we set a specific set of step requirements for the position signal which can be displayed as boundaries on the step response plot as shown in Figure 15.

We can then pick the parameters we want to tune. In our case, we pick the P and D gains of our feedback controller. Keep in mind, however, that these parameters could be any part of the model. So with a first-principles model, you could also tune physical parameters to help design parts of the plant itself in addition to the controller.

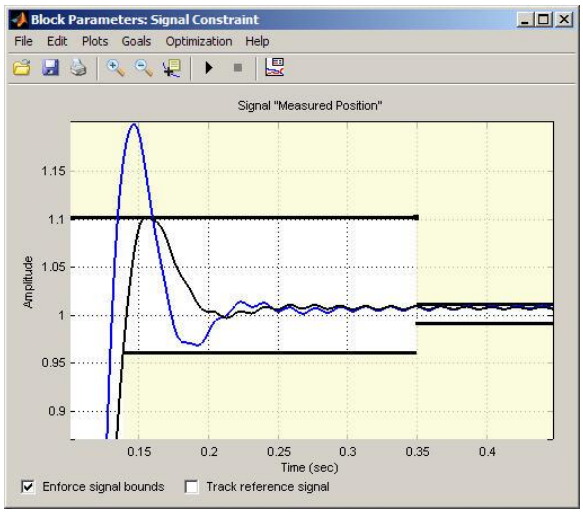

Figure 15: Step response before and after tuning the PID gains with the performance requirements indicated.

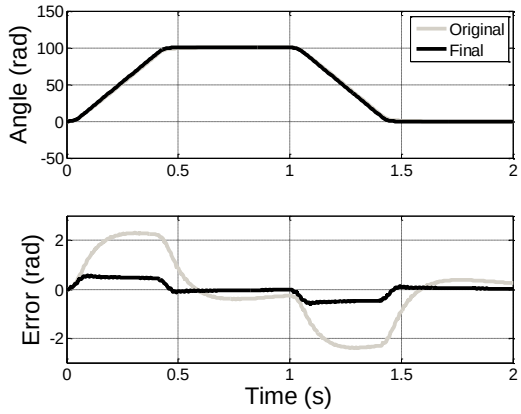

Figure 16: Position and error in position for a move with the original controller and the improved design.

After running the optimization routine you can see in Figure 15 that the step response has been improved and now lies within the design requirements. Now that we have completed the feedback controller design we can simulate the behavior of the complete system for the desired point-to-point trajectory already considered in Figure 2. The result, given in Figure 16, clearly shows the position error is well within the desired range and that the settling time has been reduced to around 0.5 seconds allowing the entire cycle time to be reduced.

Logic Control

In addition to classical control, many motion control applications will also have some logic-based control. Logic control usually changes what the system is doing based on some event. It could be a certain amount of time passing, an alarm going off, or an object reaching a certain position. Once this event happens, the system moves into a different state, perhaps initiating a new motion, going into a shutdown mode, or energizing a different part of the machine.

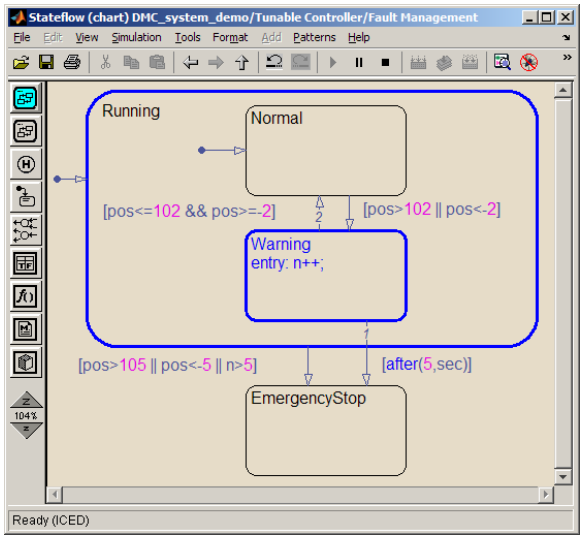

Figure 17: State chart defining the error detection logic in the motion control system.

A natural way to describe this type of logic is to draw the different states the system can exist in and the transitions that cause the system to move from one state to another, known as a state chart. Modern simulation tools allow components in the system to have behavior defined directly by a state chart as shown in Figure 17. This chart, built using Stateflow®, represents the fault-detection logic for our motion control system. It starts in a normal running state, but if the position goes outside of a safe limit, it switches into a warning state. If the position does not return to the safe limit within 5 seconds, we transition into an

emergency stop state. Alternatively, if the position goes too far outside the range it will switch to the emergency stop state immediately.

The system can take different actions depending on what state you are in. In our case, for example, when in the emergency stop state, we will use a backup control system and command the position to a safe location.

Now this is just a simple example, but in more complex cases defining this logic in a state chart makes it easier to understand than pages of written logic statements. Another advantage is that while the simulation is running, the state chart is highlighted to indicate the current state and the transitions that are taken. This allows one to easily visualize the logic and understand why the system is behaving in a certain way. Ultimately it makes it easier to identify and correct problems in the control logic.

IV. Testing

In addition to providing an environment for quickly and interactively designing motion control algorithms, a simulation environment also provides capabilities for thoroughly testing the motion control system at several different levels to help ensure that errors are caught as soon as possible and before fixing them becomes expensive.

Running Simulations

The first stage of testing can be done right in the simulation environment. By simulating the system model, the requirements can be validated very early in the development process. In this case study we had a valid set of performance requirements to meet – but what if the speed and acceleration requirements were 50% faster, how long would that have taken to determine that the motor and controller would not be able to meet the demands being placed on them? Simulation would quickly highlight this and a determination to relax the requirements or change in the motor design would be needed.

Once the control design is complete, a range of different test scenarios can be run by changing the inputs or the parameters of the simulation. This allows testing of different scenarios or variations in the plant parameters. For example, we can run the simulation for a range of stiffness and damping values given a certain distribution to ensure the motion system meets the performance requirements given manufacturing variation in the transmission shaft.

Rapid Prototyping

Once the design has been tested in simulation, you may then want to ensure the control algorithms work with the actual plant, or a physical prototype of the plant. Rapid prototyping is testing the control algorithms in real-time with the physical plant hardware and can be done before deciding on the final controller platform.

To test the design in real-time, we use an extension of Simulink called xPC Target™, which enables the use of common PC-compatible computer hardware for real-time testing. In Simulink, the plant model is exchanged for blocks that represent connections to the actual plant hardware. For example, one can use analog input and output blocks for a particular data acquisition card to represent analog signals going to and coming from the plant. Once that change has been made, the model can be compiled, downloaded and run in real-time on that separate target computer, assuming it has the appropriate data acquisition or communication cards installed. Those cards are

connected to the actual plant so that the model running on the target drives it directly.

While running the controller in real-time on a target computer there can still be a connection to the development computer, or host, so that the signals on the target can be monitored and parameters can be adjusted. With this approach you can use the host machine to run through the same tests that were used in simulation directly on the plant hardware to ensure that any approximations or simplifications made when modeling the plant do not have a significant impact on the system performance. If necessary, controller gains can be further tuned on-line until the performance is acceptable.

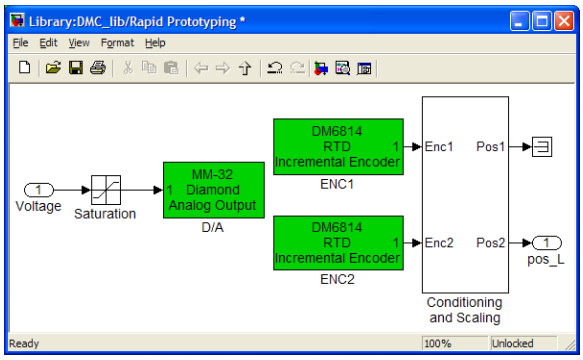

Figure 18: Plant model replaced with a hardware interface for rapid prototyping.

In our motion control example, we use a Diamond MM-32 card to send the voltage command to the motor and a RTD® DM6814 encoder card to read the motor and load positions. In the simulation environment, the voltage signal from the controller is connected to the appropriate analog output block and the encoder input block outputs the measured position signal. We also insert signal conditioning blocks to convert the data types and scale the signals as necessary as shown in Figure 18.

After making changes to the model, we compile and run it on the xPC Target™ platform on a PC-104 form factor PC. The analog output and encoder cards in the target PC are connected to the motion control system shown in Figure 3. From the host PC, we can then send commands to execute the control algorithm and run through a series of tests, logging the data to ensure that it meets our original performance goals. Figure 19 shows the response of the actual system compared to the predicted response from the simulation shown in Figure 16. Note that there are differences, but the simulated response was also a good prediction of the actual system behavior.

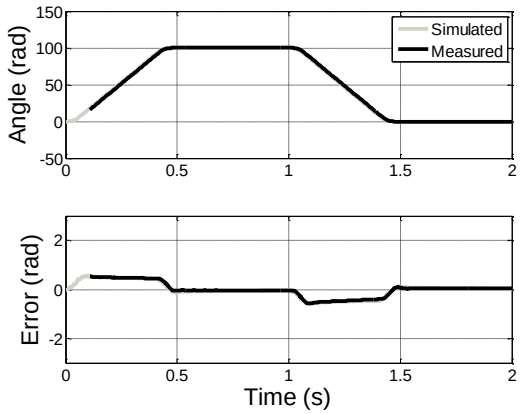

Figure 19: Plot of the position and error in position when making a point to point movement; measured from the actual system using Rapid Prototyping.

V. Next Steps

Implementation

In addition to being used for designing and testing the controller, the system model that has been developed in the simulation environment can serve as an executable specification for software engineers to refer to as they develop the embedded code. Rather than having only a textual description of the algorithm, the model can be run in various different scenarios so the original intent cannot

be misinterpreted. More than that, because the model can represent a complete description of the system, the process of creating the embedded code can be automated. This allows algorithmic changes to be made at the model level and then immediately implemented on the target hardware without a lengthy coding cycle. It also means that the same model can be deployed to different target platforms so that the decision on what platform to adopt can be made later in the design cycle. Additionally this allows for easier upgrades to a different platform in the future without having to re-code the algorithms.

HIL

After implementing the control algorithms onto the final platform, it may be helpful to test the final controller before connecting it to the actual plant if, for example, an error in the controller could cause costly damage to the system or could pose a safety threat. Rather than developing a new tool, the plant model developed earlier can be reused to now test the final control system.

Qualifying the production controller with a simulated version of the plant running in real-time is known as hardware-in-the-loop (HIL) testing. The workflow is similar to rapid prototyping except rather than the controller, you run the plant model on a PC in real-time. By using HIL testing, you can verify the production controller meets the executable specification, and underlying requirements not only across its full operating envelope but even its error handling software. For example, the fault detection logic could be fully tested without risking the actual plant. In the end you gain greater confidence that the controller will work as intended.

VI. Conclusions

It is now common practice for mechanical and mechatronic designers to make use of a computer-aided design (CAD) tool when designing a new mechanical component. It is generally seen as a necessity to prevent small mistakes from becoming very costly when they are not found until the actual assembly of the design. To further reduce the occurrence of (human) errors, there is also a growing use of automatically generated code to machine individual parts of the design on Computer Numerical Controlled (CNC) machines.

Model-Based Design is the extension of those same principles to the functional behavior of a complete mechanical, or more often a mechatronic design. It specifically aims to reduce the risk of not meeting the functional requirements by enabling early and continuous verification throughout the entire design workflow. Furthermore, it provides an environment that is rich with numerical and graphical analysis and design tools that stimulate innovation and cooperation within design teams.

All this is combined with automatic code generation capabilities for real-time testing of the design and a growing acceptance of embedded software designers that code generation technology is not only a viable way to deal with the exponential growth in complexity and size of embedded software, but also becoming sufficiently powerful and efficient for actual production use.

This paper has illustrated these advances using an admittedly simple, but realistic motion control system, showing a range of software technologies in modeling and control design that are now readily available ‘off the shelf.’ The initial application of Model-Based Design on

a real project within a design team will require investment of time and money, as well as willingness from those involved to do things differently. There are, however, many reported cases where this approach has approximately cut development time in half, and where Model-Based Design has been successfully applied company-wide (6).

VII. Acknowledgement

This paper made use of the 2009a release of the Model-Based Design environment of The MathWorks (7). The specific tools needed to reproduce the results in this paper are: MATLAB®, Simulink®, System Identification Toolbox™, Simscape™, SimMechanics™, Control System Toolbox™, Simulink® Control Design™, Simulink® Design Optimization™, Real-Time Workshop®, and xPC Target™. The models and scripts to reproduce most of the results in this paper are publicly available through The MathWorks' user community: www.mathworks.com/matlabcentral (search for motion control demo).

VIII. References

1. **Aberdeen Group.** *System Design: New Product Development for Mechatronics.* Boston : s.n., 2008.

2. **NASA.** *Return on Investment for Independent Verification & Validation.* [Presentation] 2004.

3. **Ljung, L.** *System Identification: Theory for the User.* Upper Saddle River, NJ : PTR Prentice Hall, 1999.

4. **Lambrechts, P., Boerlage, M. and Steinbuch, M.** Trajectory planning and feedforward design for electromechanical motion systems. *Control Engineering Practice.* 2004, Vol. 13, 3.

5. **Franklin, G.F, Powell, J. D. and Workman, M. L.** *Digital Control of Dynamic Systems.* s.l. : Addison-Wesley, 1980.

6. **The MathWorks, Inc.** User Stories - Industrial Automation and machinery . *The MathWorks Web site.* [Online] 2009. http://www.mathworks.com/industries/iam/userstories.html.

7. —. *MATLAB & Simulink Release Notes for R2009a.* Natick, MA : The MathWorks, Inc., 2009.