AC 2008-543: CONTROL SYSTEM PLANT SIMULATOR: A FRAMEWORK FOR HARDWARE-IN-THE-LOOP SIMULATION

David Chandler, Rochester Institute of Technology

Dave Chandler was born in Camarillo, California, on October 27, 1981. He graduated from Rochester Institute of Technology in 2004 with highest honors and a bachelors of science in Computer Engineering. Upon graduation he began his career as a software engineer at Harris RFCD, working on software defined military radio systems. He completed a Masters of Science degree in Computer Engineering from RIT in 2007. He currently lives with his wife Cheri in Rochester NY.

James Vallino, Rochester Institute of Technology

James R. Vallino is an Associate Professor in the Department of Software Engineering at Rochester Institute of Technology. He has been actively involved in the development of this program, the first undergraduate software engineering program in the United States. This involvement included bringing active learning and problem-based learning into the curriculum, developing an inter-disciplinary course sequence in real-time and embedded systems, and guiding the program through its ABET accreditation. Prior to RIT, Dr. Vallino had seventeen years of software development experience in industry, followed by his PhD studies in Computer Science at the University of Rochester. His research interests include pedagogy for software engineering education, software design especially in the real-time and embedded systems area, and model-based development methodologies.

Control System Plant Simulator:
A Framework for Hardware-In-The-Loop Simulation

Abstract

Control systems courses are common in undergraduate engineering programs. These courses focus on the design of the controller's mathematical model but rarely have students explore the practical issues of implementing the controller. Real-time and embedded systems courses focus on these practical issues with students implementing controllers for simplified Hardware-in-the-Loop plants such as a digital servo motor. Designing controllers for complex physical plants is difficult due to prohibitive costs or the risk of accidents caused by faulty controllers. These difficulties can be overcome if a simulator replaces the hardware-in-the-loop physical plant.

We designed and implemented the Control System Plant Simulator (CSPS) as a flexible framework for simulating plant models in control system implementation projects. The framework allows the user to model continuous and discrete plants defined as transfer functions or systems of state-space equations. This paper describes the design of the CSPS framework by highlighting the expansion and modification flexibility it provides with its operating system, non-real-time user interface, and physical device abstraction layers. The CSPS framework has advantages over commercial tools that can provide a hardware-in-the-loop plant simulation. The framework's scope of usage is much narrower than the commercial tools making it easier to learn how to use and modify. Also, we distribute the framework as an open-source project making it readily available for use in any course without licensing, and ensuring that deeper and more complex customizations are possible. The paper concludes with a discussion of our successful experience using the framework in real-time systems course projects, and porting to two operating environments (standard Windows XP and Ardence RTX Real-Time Extensions for Windows), two user interfaces (C-based text, Visual Basic GUI), and two data acquisition devices (USB data acquisition, simulated multi-channel IO device).

Introduction

The popularity and importance of automated controllers has grown rapidly over the past few decades[1]. The subject of Control systems has grown in importance in education as well. There are numerous challenges educators must face when teaching a control systems course. Students learn far more from their studies when they have an actual laboratory experiment to help relate the abstract concepts of engineering to real life design problems[2]. While simplified physical systems such as the inverted pendulum or the digital servo are common in academic environments, design for more practical systems is difficult due to the prohibitive costs or danger associated with the equipment involved[3].

Simulation of the entire system enables the designer to see what is going to happen before spending considerable effort implementing a design or putting expensive equipment – and potentially human life – at risk with an untested controller. However, one cannot ignore the

physical experimentation phase as part of the design and implementation of controllers[4]. Klee and Dumas argue that, "The combination of hands-on experience and computer simulation with the more traditional theoretical lecture material provides a well rounded learning experience that better prepares the students to implement digital control systems in the real world."[4]. They describe a three-step course for undergraduate students that begins with theory and the design of the digital controller mathematically. Students then use simulations to work out any problems with the theoretical design. Finally, the controller is implemented and connected to real physical hardware. This 'start-to-finish' design and implementation is invaluable to students as it provides the complete picture.

The argument that students must implement their controller designs and attempt to actually run their controllers on plants has been made before. Some educators bemoan the recent trend "towards increased use of simulation in engineering education, coupled with a decline of the use of physical experiments."[2]. They admit that the expense of physical equipment is prohibitive, but outline a number of reasons why it is important to implement physical controllers. "The typical student therefore finds it motivating to work with laboratory experiments. A successful laboratory experiment is some proof that the student has been able to perform a task which is of relevance to the real world."[2]. Accepting MATLAB simulations as proof that a controller has been designed properly changes classes in digital control engineering to courses that teach little more than mathematics.

Control system simulators have been developed in the past. The Stanford Universal Plant[5] was one such simulator used in academic course work. This provided analog simulation of plants based on simulation boards plugged into the chassis. The plant models could not be easily changed and did not handle discrete or state-space plant models. The Shadow Plants Dynamic Simulation Testbed[6] by Honeywell is an example of a commercial plant simulator. Typically, this is tailor-made for specific situations, products, or markets. Additionally, it is prohibitively expensive, which reduces their ability to be widely adopted. Another possible solution is to use commercial simulation software such as Simulink® or LabView® driving a data acquisition board. These environments provide the mathematical simulations needed for modeling the plant system and will often support extendable interfaces for both user interfaces and interfaces to data acquisition devices. There is licensing of these commercial packages that may limit wide-spread use of the software. Also, we have found that there is a steep learning curve before one can be productive creating models, custom user interfaces, and interfaces to new data acquisition boards.

The Control System Plant Simulator (CSPS) Framework addresses all of these issues:

- Current plant simulation frameworks are very expensive. The CSPS framework is an open source project, provided at no cost.
- Most simulators are designed to simulate the mathematical model of a plant or controller, and provide information about how these models interact. These simulators cannot test the implementation of these designs. The CSPS framework runs a hardware-in-the-loop simulation that takes the place of a physical plant. It is designed to be connected to an implemented controller, and behave as a real plant would.

- Hardware-in-the-loop test simulations have been made for specific situations, products, or markets. The CSPS framework provides a general framework upon which plants of different natures may be simulated by simply providing a model of the plant.
- The CSPS framework is extendable. User interfaces, plant descriptions, and physical interfaces may be updated or customized with little effort.
- Most hardware-in-the-loop simulators require specialized equipment or test boards to run. The CSPS framework is a Windows XP application that can run in real-time provided the proper Windows extensions have been installed, and can be ported to other operating systems such as Linux.

Control System Plant Simulator (CSPS) Design

The CSPS framework is designed as a Hardware-In-The-Loop simulation framework. Hardware-In-The-Loop simulation replaces physical hardware with a simulated model[7]. In a traditional controller-plant setup, a computer runs an implementation of the digital controller, as shown in Figure 1. The controller sends control signals through a data acquisition device to the inputs of the plant. The plant physically responds to these input signals, as sensors monitor the state of the plant and provide output signals to the controller. A Hardware-In-The-Loop simulated plant behaves the same way as a physical plant does. Viewed from a black box perspective, a plant launched under the CSPS framework appears to be the same as a real plant to the connected controller.

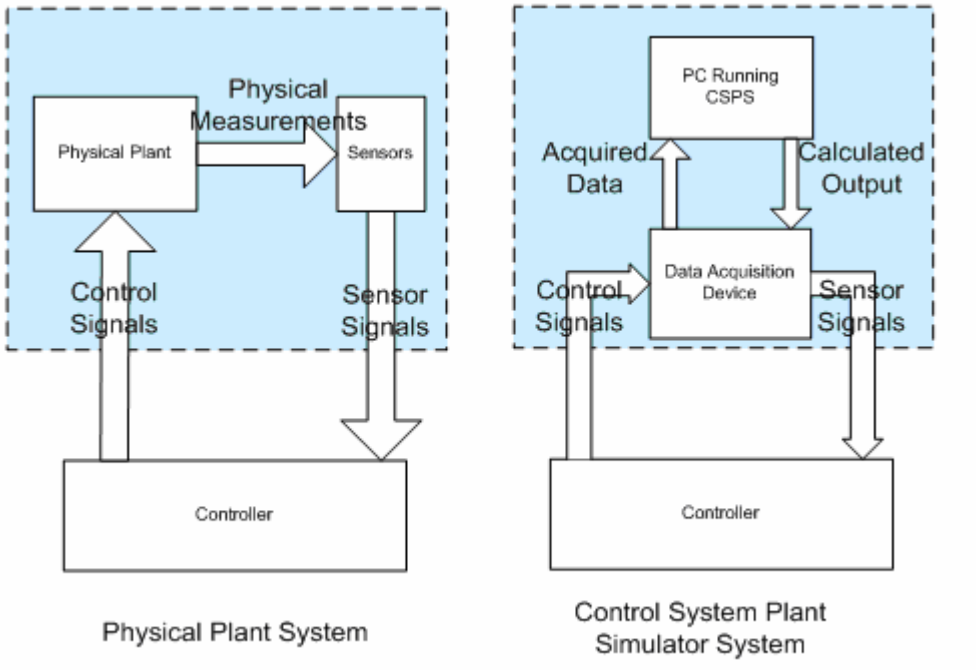

Figure 1: CSPS and Physical Plant Comparison

The CSPS framework is designed with academic environments in mind. It is of the utmost importance that students be able to implement their designs as physical controllers. It is often too expensive or dangerous to test these controllers with a physical plant as hardware-in-the-

loop. The CSPS framework can be used in place of this equipment, and provide results to determine the effectiveness of the controller. The software is an open-source project (http://www.se.rit.edu/~rtembed/csps) which allows use of the CSPS framework on any laboratory computer. This provides each student with a simulated plant when testing an implemented controller without the constraints of limited or unavailable lab equipment. The CSPS framework can simulate many different kinds of plants with different needs for visualizing the plant operation. Using the CSPS's flexible front-end, users may implement a specific text-based or graphical user interface for the simulated plant. The CSPS's operating system abstraction layer provides for easy porting to new operating systems. This abstraction layer provides an interface to all operating system calls thus localizing changes when porting to a new operating system. Similarly, there is an abstraction layer for the data acquisition hardware.

The CSPS Framework simplifies the physical requirements for the experimentation phase of control systems education. Instead of making the decision between asking students to design controllers for low-cost plants that are too simple to be realistic, and never attempting to control anything at all, educators may now elect to simulate the physical hardware using the CSPS framework. Students may perfect their designs at their own workstations without the need of additional equipment.

CSPS Framework Software Architecture

The CSPS Framework is a set of three applications that interact with each other during the simulation. Two main considerations led to this implementation. The first is that Ardence® RTX® Real-Time Extensions for Windows was used to ensure real-time processing from within Windows. RTX® applications are kernel mode applications that must be run in their own process space. Typical RTX® applications consist of a Win32 non-realtime process and an RTX® process that manages what must be run in real-time. The second consideration was that a user may wish to simulate any plant possible, each with different user interface requirements. To achieve this level of flexibility, the CSPS user interface is an interchangeable program that may be written by end users to match the plant they wish to simulate.

The three main process spaces are the Win32 process, the Computation process, and the User Interface process. These processes communicate through pairs of defined unidirectional interfaces as shown in Figure 2.

The Win32 process is the main process in the system. At framework start-up, it launches the other two processes and serves as a bridge between the user interface and the computation core. The Win32 process performs all error checking, all pre-simulation configuration management, and all file system interactions.

The Computation process runs the actual simulation. This process handles all interactions with the connected data acquisition device as well. During a simulation it reads data from the data acquisition device, processes it, calculates a set of output values, and writes those output values to the data acquisition device. Should a user wish to use a different data acquisition device, a small portion of this process must be ported for the new device.

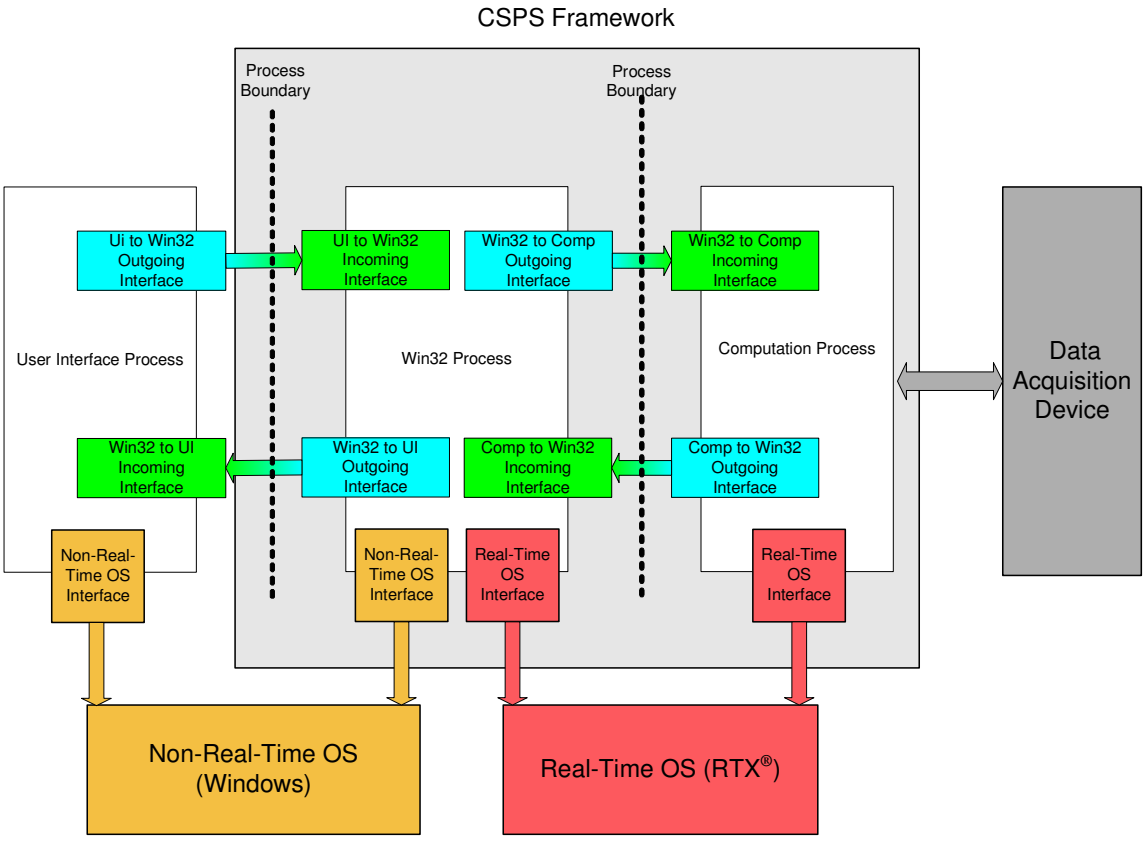

Figure 2: System Design

The User Interface process is not within the bounds of the framework: Framework users can provide a User Interface application specific to the plant being simulated. A user may wish to simulate water entering and leaving a tank, and would like to see water levels rise and fall graphically. Another user may wish to simulate an inverted pendulum and would like to see the pendulum alter position dynamically. Others may elect for simple text based user interfaces that only offer students the ability to start and stop a pre-determined plant. Individual users and implementers are expected to create their own user interface application to go along with the plant they wish to simulate. A dynamic linked library is provided to give implementers an API to use when developing their user interfaces. In addition, two fully functional user interfaces, a text-based console application and a Visual Basic graphical user interface, are provided as examples of how to use the API.

Configuration and Simulation

The CSPS must be properly configured before it can simulate a plant. Configuration of the CSPS consists of defining a plant, and connecting the inputs and outputs of the plant to a data acquisition device.

The CSPS framework API provides a number of different ways to define a plant. The preferred method is for users to enter a plant as a set of state space matrices, but the framework also supports entry as a transfer function, a matrix of transfer functions, or a nonlinear equation.

Transfer functions may be provided either as the coefficients of the terms in the numerator and denominator polynomials, or as a set of poles and zeros. No matter how a transfer function is provided, the CSPS framework will convert it to a state space matrix for simulation. Plants may be either continuous or discrete. The CSPS will discretize continuous plants via one of three available discretization methods: forward rectangular, backward rectangular, and bilinear. Future developers may add more discretization methods as they see fit. The CSPS framework provides the ability to save plants to, or load plants from, a text file. This file is ASCII based and may be written by users outside of the CSPS environment as well.

When a data acquisition device is connected to the CSPS framework, its physical interfaces are abstracted by Physical Ports within the system. The CSPS framework writes to, and reads from these physical ports when it wants to send or retrieve data from the controller. It is expected that at a particular workstation the data acquisition unit will not often change, but the plant to be simulated is likely to be altered quite often. This means that plants with different input/output characteristics will have to interface with the same set of available physical ports. Physical ports are mapped to the inputs and outputs of the user-defined plant via user defined 'Pseudo Ports'. Pseudo ports connect physical ports to the inputs and outputs of a defined plant, and convert the data received from the physical interface to engineering values that are meaningful to the plant. Physical and pseudo ports may be either digital or analog. Digital pseudo ports may divide larger digital physical ports into smaller ports for added flexibility. For example, a 32 bit physical interface may be divided into two 16 bit pseudo ports. Analog pseudo ports scale the data read by the analog physical ports linearly to meaningful engineering values. For example, a data acquisition unit may have an input range between -10 and +10 volts. An engineering value from the controller may be mapped to this interface that represents values between 50 and 100 PSI. The mapped analog pseudo port scales the input from the physical port to the proper engineering values for the plant to use. The CSPS framework can save pseudo port configurations to text based files. It can load pseudo port configurations from these files as well.

To prevent long read or write times from impacting simulation performance, the CSPS framework accesses connected data acquisition devices on a separate thread of execution. This thread periodically reads the input physical ports and stores the results in a cache. It will also write data in the cache to the output physical ports. During a simulation, the input pseudo ports will convert the physical port data to engineering values, and provide them to the simulated plant's inputs. Once the simulation has calculated output values, output pseudo ports convert the engineering values back to physical values and store them in the cache to be written to physical output during the next I/O update cycle. The user may schedule how often each physical port is read or updated individually. This allows some interfaces with fast access times to be read often, while slower interfaces may be left alone for longer periods of time.

The CSPS framework provides the ability to log three different kinds of messages. Informational messages simply provide information about the current status of the system. Critical messages are provided to inform the user of severe system errors, such as when the simulation misses a deadline. I/O messages provide the current value of the inputs and outputs of the plant. These messages are available to the user interface, and may be captured to files. Informational and Critical messages are saved as text files. I/O messages are logged as comma delimited files for easy import to other programs. The user can configure the rate at which the framework

generates I/O messages. All file operations are done in the Win32 process so that there is no impact on the real-time performance of the framework.

Evaluation

There are two versions of the Computation process for the CSPS framework. One version runs as a real-time process under RTX. The second version runs as a standard Windows process making for a simulation running entirely within the Windows environment. The RTX CSPS framework was run using a simulated interface that replicated physical port behavior by writing to or reading from shared memory locations that specifically designed CSPS framework physical ports would access during read or write operations. The Windows CSPS framework was connected to a Data Translations DT-9812 data acquisition device. This data acquisition device has a USB interface which was not supported by RTX.

The CSPS framework was tested using a number of different plants in order to prove how effective it can be. A MATLAB simulation was the baseline used to check the accuracy of the CSPS framework. The unit step responses of the following plants were simulated open loop with no external controller:

A Simple Spring-Mass Plant[8]:

State Matrix: $\begin{bmatrix} 0 & 1 \\ -1.5 & -2.5 \end{bmatrix}$

Input Matrix: $\begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$

Output Matrix: $\begin{bmatrix} 0 & 1 \end{bmatrix}$

Feedthrough Matrix: $\begin{bmatrix} 0 \end{bmatrix}$

An Airplane Pitch Controller[9]:

State Matrix: $\begin{bmatrix} -0.739 & -.921 & 0 \\ 1 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix}$

Input Matrix: $\begin{bmatrix} 0.5 \\ 0 \\ 0 \end{bmatrix}$

Output Matrix: $\begin{bmatrix} 0 & 0.4604 & 0.1419 \end{bmatrix}$

Feedthrough Matrix: $\begin{bmatrix} 0 \end{bmatrix}$

A bus active suspension system[9]:

State Matrix: $\begin{bmatrix} -48.17 & -28.92 & -3.361 & -12.21 \\ 64 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 8 & 0 \end{bmatrix}$

Input Matrix: $\begin{bmatrix} 0.01563 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Output Matrix: $\begin{bmatrix} 0 & 0.003525 & 0.002347 & 0.009766 \end{bmatrix}$

Feedthrough Matrix: $\begin{bmatrix} 0 \end{bmatrix}$

A Car and Wheel Shock Absorber System[8]:

State Matrix: $\begin{bmatrix} 0 & 1 & 0 & 0 \\ -160 & -60 & 80 & 40 \\ 0 & 0 & 0 & 1 \\ 8 & 4 & -8 & -4 \end{bmatrix}$

Input Matrix: $\begin{bmatrix} 20 \\ -1120 \\ 0 \\ 80 \end{bmatrix}$

Output Matrix: $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

Feedthrough Matrix: $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Each of these simulated plants had data collected at various points in time. The measured data is provided in the following table:

Plant	CSPS Version	Expected Value	Simulated value	% difference
Spring-Mass	Win32	0.15 at 1.26 sec	0.14 at 1.26 sec	-6.6%
		0.251 at 2.26 sec	0.25 at 2.26 sec	-0.4%
		0.329 at 5.38 sec	0.33 at 5.38 sec	+0.3%
	RTX®	0.15 at 1.26 sec	0.148 at 1.3 sec	-1.3%
		0.251 at 2.26 sec	0.251 at 2.3 sec	0%
		0.329 at 5.38 sec	0.33 at 5.4 sec	+0.3%
Airplane Pitch	Win32	0.44 at 4.57 sec	0.44 at 4.56 sec	0%
		0.468 at 6.83 sec	0.46 at 6.80 sec	-1.7%
		0.838 at 16.1 sec	0.812 at 16.1 sec	-3.1%
	RTX®	0.44 at 4.57 sec	0.44 at 4.6 sec	0%
		0.468 at 6.83 sec	0.47 at 6.8 sec	+0.4%
		0.838 at 16.1 sec	0.84 at 16.1 sec	+2.4%
Bus Suspension	Win32	2.23e-5 at 0.633 sec	2.04e-5 at 0.64 sec	-8.5%
		3.55e-6 at 1.25 sec	3.30e-5 at 1.28 sec	-7.0%
		1.29e-5 at 28.2 sec	1.02e-5 at 28.2 sec	-20.9%
	RTX®	2.23e-5 at 0.633 sec	2.2e-5 at 0.6 sec	-1.34%
		3.55e-6 at 1.25 sec	4.0e-6 at 1.2 sec	+15%
		1.29e-5 at 28.2 sec	1.3e-5 at 28.2 sec	+0.8%
Car Shock System	Win32 Car Pos	1.42 at 1.34 sec	1.52 at 1.38 sec	+7.0%
		0.988 at 6.14 sec	1.02 at 6.14 sec	+3.2%
	Win32 Wheel Pos	1.26 at 1.24 sec	1.22 at 1.24 sec	-3.2%
		0.955 at 5.76 sec	0.98 at 5.76 sec	+2.6%
	RTX® Car Pos	1.42 at 1.34 sec	1.42 at 1.3 sec	0%
		0.988 at 6.14 sec	0.988 at 6.1 sec	0%
	RTX® Wheel Pos	1.26 at 1.24 sec	1.26 at 1.2 sec	0%
		0.955 at 5.76 sec	1.00 at 5.8 sec	4.7%

In most cases, the results were quite good, especially considering the fact that the physical simulations were of discrete equivalents, sampled at only ten hertz, of the continuous plants simulated in MATLAB. Additionally, these simulations were not produced in real-time. For the Windows simulation, the bus suspension system produced poor results, with one value as bad as 20% off from what was expected. The bus suspension system naturally produces very small output values that range between 0 and 3e-5. This was far too low to send to the analog output lines, so pseudo-port scaling was enacted, magnifying output by over 65000 times. Any errors would be considerably magnified. In addition, this system ran for the longest period of time, compounding the effects of non-real-time operation.

As for RTX simulation, there was only one measurement that experienced a percent difference greater than 5: The bus suspension system simulation experienced a 15 percent difference at 1.25 seconds. However, this is not an accurate comparison as the simulation only produced

values at 1.2 and 1.3 seconds. Comparing this with a value measured at 1.25 seconds is not completely valid. The value measured at 1.3 sec was 3.0e-6, making a value of 3.55e-6 at 1.25 quite likely.

The following images show captures of the oscilloscope plots compared to MATLAB simulations of the plants, demonstrating how closely the CSPS framework matches the expected continuous results:

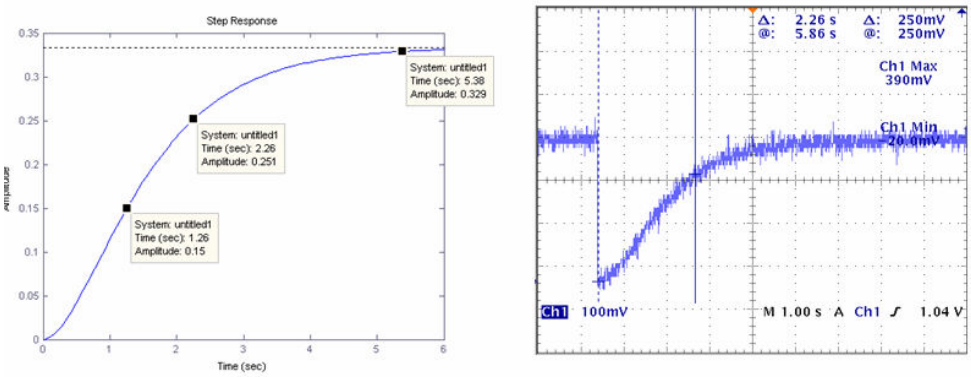

Figure 3: MATLAB and CSPS Simulation of Step Response to Spring Mass System

Figure 3 consists of the MATLAB simulation of the step response to the spring-mass system on the left, and an oscilloscope capture of the output of the CSPS simulation of the step response to the same system on the right. This simulation has a very small steady state value of approximately 300 millivolts. This shows that the CSPS is accurate even under situations where the output changes very slightly.

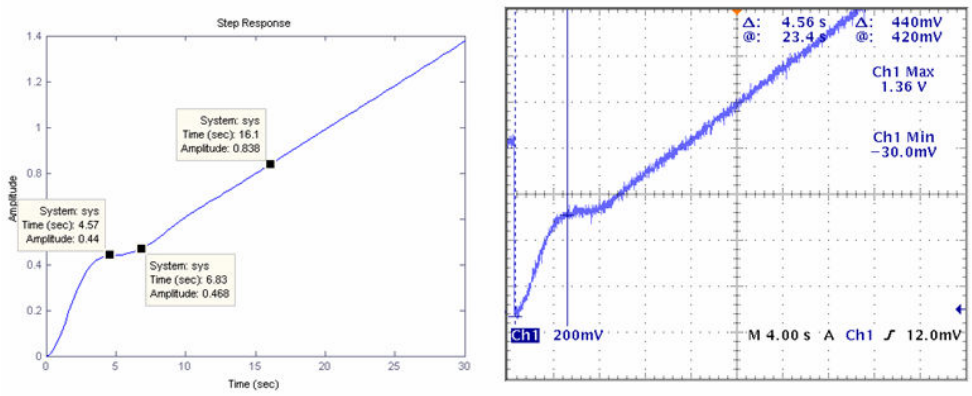

Figure 4: MATLAB and CSPS Simulation of Step Response to Airplane Pitch System

Figure 4 consists of the MATLAB simulation of the step response to the Airplane Pitch plant on the left, and an oscilloscope capture of the output of the CSPS simulation of the step response to the same system on the right. Again the characteristics are very similar.

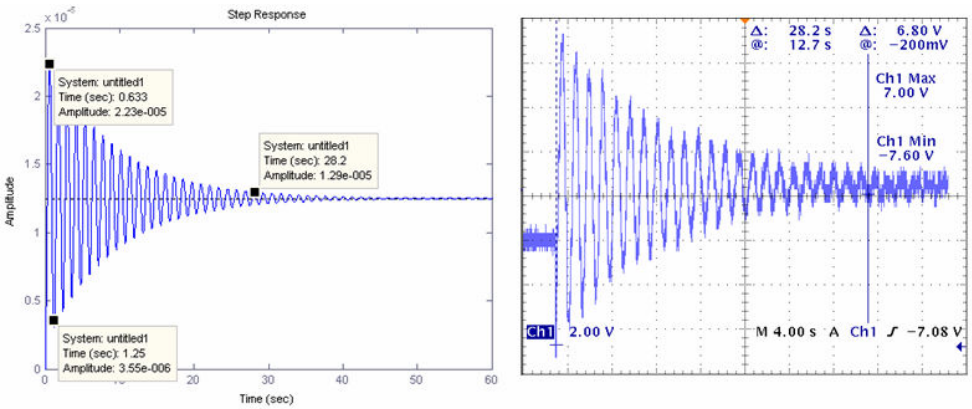

Figure 5: MATLAB and CSPS Simulation of Step Response to Bus Suspension System

Figure 5 consists of the MATLAB simulation of the step response to the bus suspension system plant on the left, and an oscilloscope capture of the output of the CSPS simulation of the step response to the same system on the right. As stated earlier, the maximum and minimum values for this plant are exceptionally small, so this system made use of the CSPS's ability to scale plant engineering values up to larger physical values.

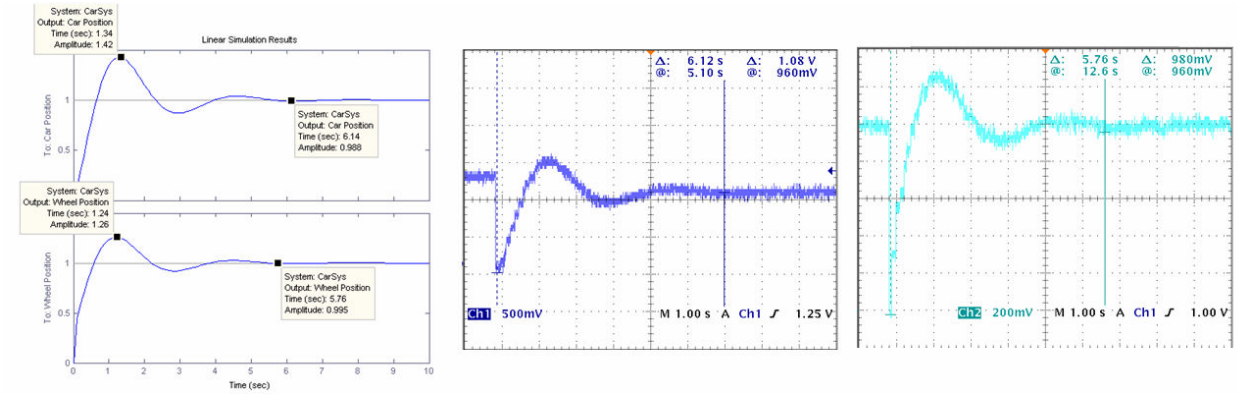

Figure 6: MATLAB and CSPS Simulation of Step Response to Car Shock System

Figure 6 consists of the MATLAB simulation of the car and wheel shock absorption plant on the far left. The middle image is of an oscilloscope capture of the output of the CSPS simulation of the system step response of the car position, and the rightmost image is an oscilloscope capture of the wheel position. Here one can see the ability of the CSPS to provide multiple outputs faithfully.

The CSPS Framework was also provided to a class of graduate and undergraduate Software Engineering and Computer Engineering students. The students used the CSPS framework to develop simple VxWorks controllers. The project required them to manually tune the coefficients of a standard Proportional-Integral-Derivative controller to minimize the overshoot and settling time of the plant. They did not know the plant model and tuned the controller experimentally. The exercise was a success as the students were able to construct reasonable controllers that produced physical signals to control the simulated plant. There were no complaints or problems using the CSPS framework for this project.

Conclusion

Control system education benefits greatly by having students develop real controllers that are connected to real systems to monitor the success or failure of their controller designs and implementations. Students gain the benefit of building and debugging their controller implementation, as well as, a sense of accomplishment that is missing when dealing with control systems in strictly mathematical terms[2]. The CSPS framework, designed for use in control system coursework, provides a suite of applications that make debugging of controllers possible without the use of expensive or dangerous equipment. Comparatively inexpensive data acquisition systems and common Windows workstations can be used instead. The framework is flexible, and provides plenty of hooks upon which end users may attach their own interfaces and data acquisition systems. It is accurate, and has been used in an academic environment as a teaching tool with good success. The CSPS framework is available at no cost for academic, non-commercial use as an open-source project at http://www.se.rit.edu/~rtembed/csps. We hope that users will customize it, update and enhance it, build their own plants and user interfaces for it, and share their extensions thus creating a community library of functional Hardware-In-The-Loop simulations to be used by students everywhere.

Bibliography

1. Ogata, K., Modern Control Engineering. 2002, Englewood Cliffs: Prentice Hall.
2. Foss, B.A., T.I. Eikaas, and M. Hoyd. "Merging Physical Experiments Back Into the Learning Arena". *Proceedings of the 2000 American Control Conference*. 2000. Chicago, IL.
3. Juang, J.-C. "Controller Rapid Prototyping and its Incorporation in Control Education". *Proceedings of the Proceedings of the 4th IFAC Symposium on Advances in Control Conference*. 1997. Istanbul.
4. Klee, H. and J. Dumas, "Theory, Simulation, Experimentation: An Integrated Approach to Teaching Digital Control Systems". *IEEE Transactions on Education*, 1994. **37**(1).
5. Franklin, G.F. and J.D. Powell, "Digital Control Laboratory Courses". *IEEE Control Systems*, 1989. **9**(3).
6. Olukotun, K., M. Heinrich, and D. Ofelt. "Digital System Simulation: Methodologies and Examples". *Proceedings of the Proceedings of the 1998 Design Automation Conference*. 1998.
7. Grega, W. "Hardware-in-the-loop Simulation and its Application in Control Education". *Proceedings of the Frontiers in Education Conference*. 1999.
8. Messner, W.C., et al. "Modeling Tutorials for MATLAB and Simulink". [acessed May 2006]; Available from: http://www.me.cmu.edu/ctms.
9. Messner, W.C. and D. Tilbury, Control Tutorials for MATLAB and Simulink: A Web-Based Approach. 1998, Englewood Cliffs: Prentice Hall.