AC 2011-525: WEB-BASED SCRIPTS FOR ANIMATING SYSTEM SIMULATIONS

Ames Bielenberg

Ames Bielenberg is an engineering student at Swarthmore College.

Erik Cheever, Swarthmore College

Erik Cheever is a Professor of Engineering at Swarthmore College. He teaches in the areas of Circuits, Electronics, Linear Systems, Control Theory and DSP.

Web-based Scripts for Animating System Simulations

Abstract

Visualizing the behavior of physical systems can be an invaluable tool for a student's understanding of linear system dynamics. When presenting web-based educational material involving such systems there have been two standard choices. In the first, the material is presented with a static drawing of the system complemented by graphs of system outputs such as displacements and forces, along with accompanying text to try to explain what is happening, exactly as a textbook would. The second method involves using a web technology (e.g., Java, JavaScript, Flash…) to create an animation of the system moving dynamically. Clearly the second method is preferable, but requires a good understanding on behalf of the author of the technologies used and a considerable investment of time to create each animation.

To lessen the difficulty of producing web-based animations, we have created an easy to use scripting system for defining the system in three steps. First its constitutive mathematical relations are defined, then a drawing is created that depicts the system, and then the drawing is animated. The system is described mathematically by a state-space model (i.e, **A**, **B**, **C** and **D** matrices). The drawing of the system is described in terms of graphics primitives commonly used to depict linear systems. For translating mechanical systems this consists of such commonly used objects as springs, masses, dashpots and sliding friction, along with dynamically resized arrows to show force and displacement. For electrical systems, primitives exist that represent resistors, capacitors, and inductors as well as arrows to show current and dials to show voltage. Other types of systems include rotating, thermal fluid, and electromechanical. Finally the system drawing is animated upon pushing of a "Go" button. There is also the capability to add a dynamically generated plot below the animation that shows important system variables.

The scripting system is written in JavaScript and makes use of the Raphaël vector graphics library. Vector graphics allow the figures to remain sharp when scaled to any size and, ultimately, when printed. The user need not know JavaScript to use the system; the user need only add a script to any HTML document. We have a fully functional system that can describe, draw, and animate a simple mass-spring-dashpot system with about ten lines of script, while also being capable of simulating much more complicated systems.

Background

A course in Linear Systems is common in engineering programs. Understanding the concepts involved can be very important for a student to develop insights into how a wide variety of physical systems behave. However, from a student's perspective there are many hurdles to be overcome in the development of a physical intuition for such systems including the level of mathematics required (typically Laplace Transforms and linear algebra), the wide variety of problem domains (electrical, mechanical, thermal...), and the fact that textbooks necessarily present the material in a static way though the systems are by their very nature dynamic.

Compounding this last difficulty is the fact that many students are "visual" learners according to Felder's index of learning styles [1]. For these students, an animation of a physical system can be an important tool to help them learn [2, 3, 4, 5, 6]. It has been shown that students prefer having access to animations in addition to textbook-only presentations and the animations significantly increased their ability to visualize system behavior, as well as increasing enjoyment and self-confidence [3, 7, 8, 9]. In addition, Kolb's theory of experiential learning posits that experience, which can be provided by animations, and reflection are part of student learning [10].

There are many ways to present animations of simulations of physical systems for students. There are several web pages that have a wide range of animations [11, 12, 13, 14, 15]. However these largely consist of predefined system that can't be changed, or have examples from only a few problem domains. It would be better to be able to animate across a variety of physical domains [16], and to enable an instructor to develop his or her own examples. It is possible to develop animations using MatLab® and/or Simulink® [17, 18], but this presupposes that all students have ubiquitous access to those applications, as well as any necessary toolboxes. Java [14], or other web-based applications can be used to develop animations available to those with only a web browser, but this requires significant knowledge on the part of those developing the applications. These, and other, impediments to the successful use of animations were noted by Naps et al. [2]. The system discussed here avoids these problems by requiring only a little scripting on behalf of the author, and a user with a web-browser with JavaScript.

Introduction

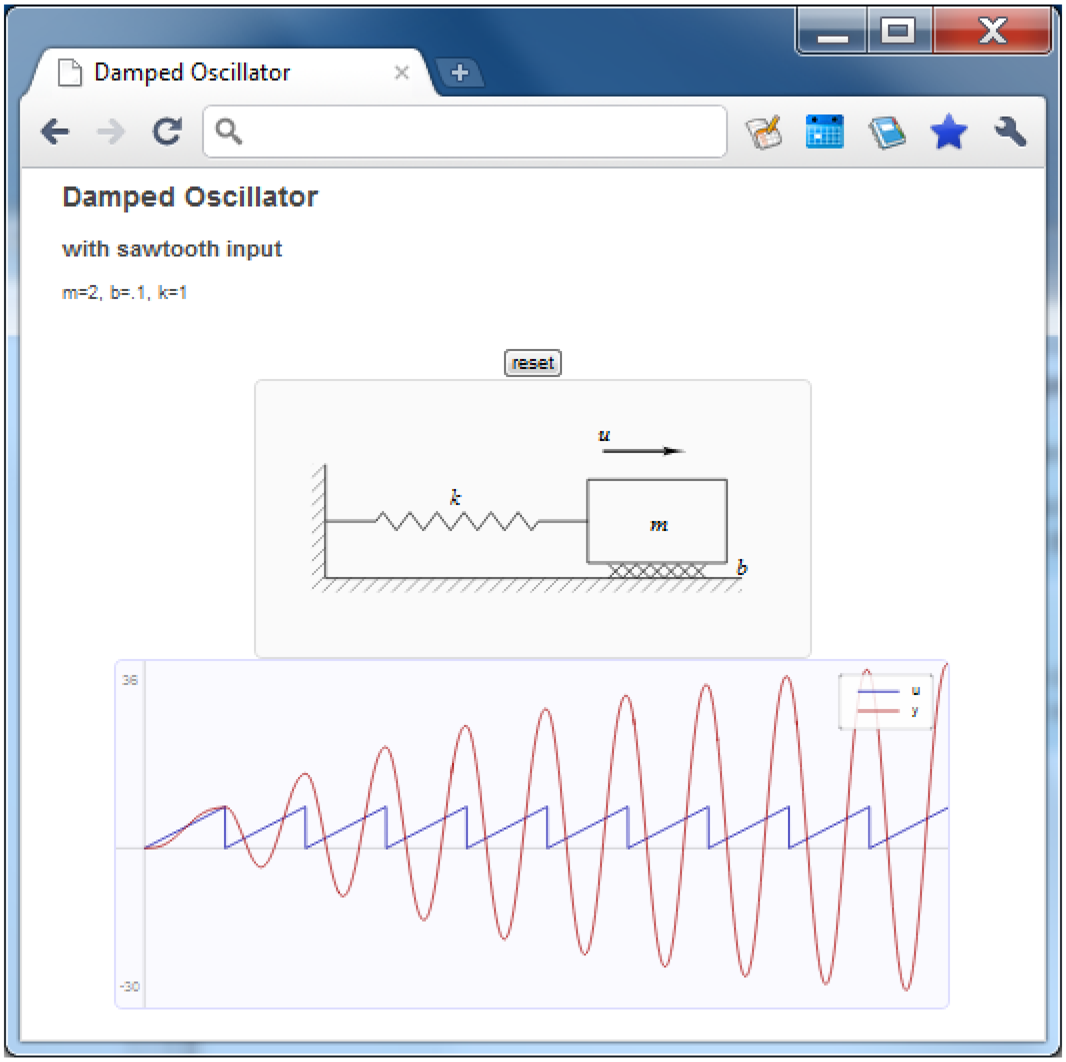
The system described in this paper easily allows a user to create web-based animations of linear systems while requiring no web-programming experience.. S/he simply writes a simple script. The system is described mathematically by using a state-space model (i.e, **A**, **B**, **C** and **D** matrices) with initial conditions and optional input functions. The drawing to be animated is described in terms of graphics primitives such as masses, springs, resistors, and capacitors. System inputs are defined by the user. The system is simulated using a fourth order Runge-Kutta algorithm, and the system outputs define the behavior of the drawing.

For translating mechanical systems the primitives consist of commonly used objects such as springs, masses, dashpots and sliding friction, and dynamically resized arrows to show force and displacement (or, indeed, any system variable). For electrical systems the primitives are resistors, capacitors, inductors, voltage and current sources, and op-amps, as well as arrows and dials to show current and voltage. Other types of systems include rotating mechanical, thermal/fluid, and electromechanical. In addition to the animation it is possible to add a dynamically generated plot below the animation that shows system inputs and outputs.

Implementation

The scripting system is written in JavaScript and uses the Raphaël vector graphics library [19]. Vector graphics allow the figures to remain sharp when scaled to any size and ultimately, when printed. There is no need to know JavaScript to use the system; the user need only add a short script to any HTML document. A detailed description for a simple system follows.

To demonstrate the system in action, consider first the output of an animation, and then the script itself. The output of a simple mass-spring-friction system is shown in figure 1. This image, including the animation, was created with a script of 16 lines (not including comments).



Damped Oscillator

with sawtooth input

m=2, b=.1, k=1

Figure 1: Output of Simple System

The input to the system is a sawtooth, u(t), near the resonant frequency of the system. The output is the position of the mass, y(t). Though obviously not visible in this screen shot, during the course of the animation the mass moves back and forth, the arrow depicting u(t) grows and shrinks, the spring expands and contracts, and the lower plot is created.

To understand the script, it can be examined in pieces. Figure 2 shows the portion of the script that describes the system using a standard state space representation. The notation should be very comfortable to anybody familiar with MatLab or C, with only small variations (for example, a two-dimensional matrix is defined in terms of nested one-dimensional matrices). Note that there are two outputs. The output with index '0' is simply the input (used in the graph), and the output with index '1' is the position of the mass. These two outputs drive the animation and are shown on the plot.

```
//////    State Space representation of system    ///////
m=2, b=.1, k=1;      // Define mass, friction and spring values.

A=[[0,1], [-k/m,-b/m]];           // Define A matrix
B=[[  0], [1/m]];                 // ... B matrix
C=[[0,0], [1,0]];                 // ... C matrix
D=[[  1],   [0]];                 // ... D matrix

sys1=system(A,B,C,D).tMax(90);    // Define system, and end time.

// Define system input (a ramp that repeats every 9 seconds)
sys1.input(0,function(t) {return t%9});
```

Figure 2: State-Space description of the system

Figure 3 shows the description of the drawing. This is a fairly simple drawing, but much more complex drawings are possible. Of critical importance is the line:

```
m1=fig1.mass(1,'m',ax,200,-10,100,60,'b');
```

which defines system output "1" (the first argument) as the variable that determines the position of the mass during the animation. One end of the spring is defined to be attached to the mass (i.e., the last argument)

```
fig1.spring('k',ax,0,m1);
```

so it automatically extends and contracts along with the mass.

```
//////    Description of drawing    ///////
// Define the drawing of sys1 to be 400x200 pixels.
fig1=diagram([400,200],sys1);

// Define the vertical wall:
//   First argument, [40,60], is the upper right corner
//   Second argument, [10,82], is width and height.  The wall is hatched,
//     so the letter 'E' specifies a bold vertical on the 'E'ast side.
fig1.wall([40, 60],[ 10,82],'E');
fig1.wall([50,142],[300,10],'N');  // Define the horizontal wall (the floor)
fig1.wall([40,141],[11,11]);       // Define small square where walls meet

// Define the axis of motion.
//    First argument is origin of the axis (x=50, y=101).
//    Second argument is the direction of motion
//       (in degrees, 0 degrees is to the right).
ax=fig1.axis([50,101],0);

// Define the mass (see figure 4 for details)
//   (note: position is defined by state variable output '1').
m1=fig1.mass(1,'m',ax,200,-10,100,60,'b');

fig1.spring('k',ax,0,m1);   // Define spring (figure 4 for details)

// Define the arrow depicting the input force  (figure 4 for details)
//   (note: length is defined by state variable output '0').
fig1.arrow(0,ax,200,-50,5,'u');
```

Figure 3: Physical description of the system drawing

Figure 4 shows the description of the mass, spring and arrow definitions from the help documentation. For the mass, the fifth argument describes the support under the mass (either wheels, friction, or nothing). A full help system and tutorial is available at http://www.swarthmore.edu/NatSci/echeeve1/Ref/LPSA/Animations/index.html.

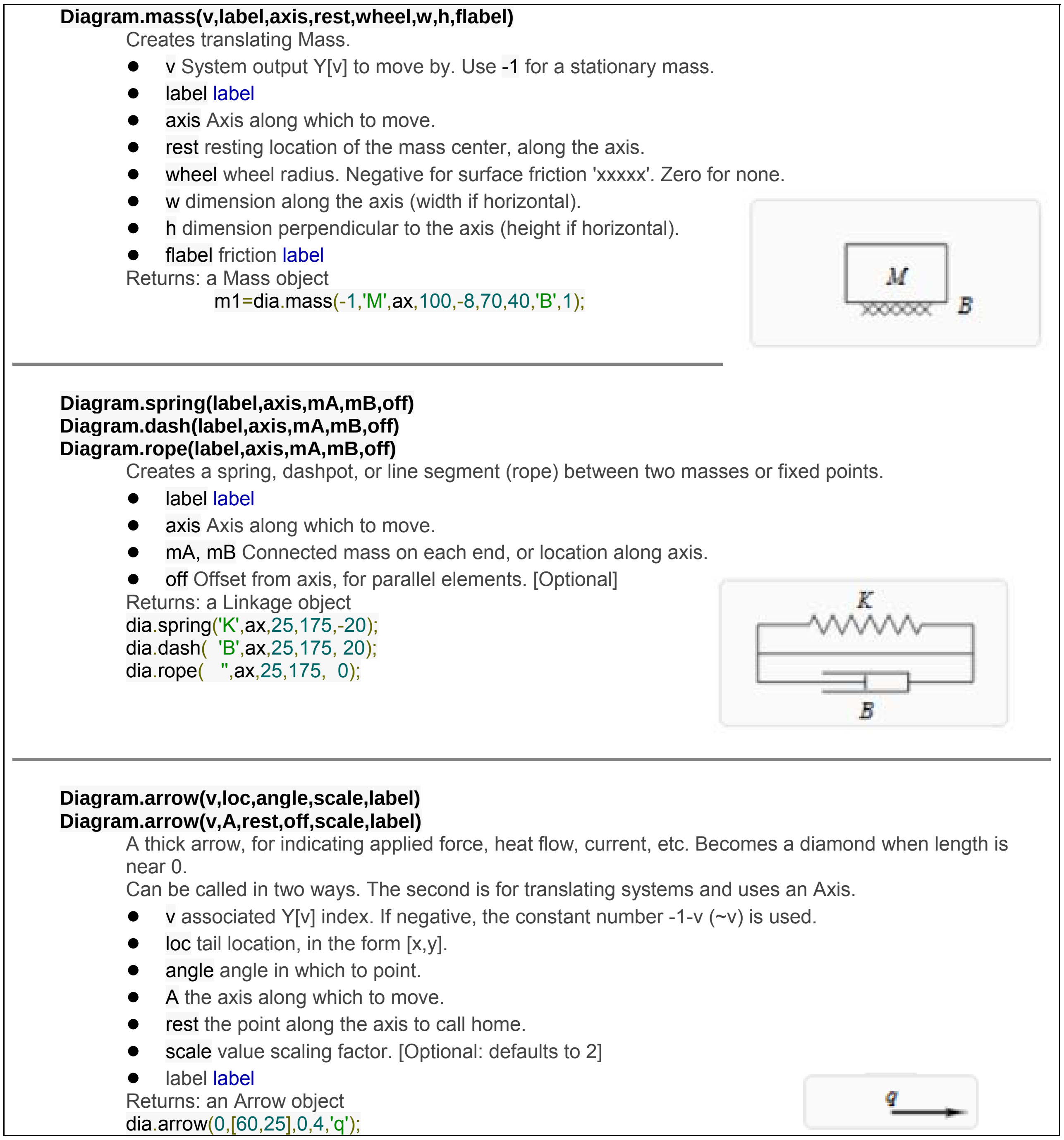


Diagram.mass(v,label,axis,rest,wheel,w,h,flabel)

Creates translating Mass.

- v System output Y[v] to move by. Use -1 for a stationary mass.
- label label
- axis Axis along which to move.
- rest resting location of the mass center, along the axis.
- wheel wheel radius. Negative for surface friction 'xxxxx'. Zero for none.
- w dimension along the axis (width if horizontal).
- h dimension perpendicular to the axis (height if horizontal).
- flabel friction label

Returns: a Mass object

```
m1=dia.mass(-1,'M',ax,100,-8,70,40,'B',1);
```

Diagram.spring(label,axis,mA,mB,off)
Diagram.dash(label,axis,mA,mB,off)
Diagram.rope(label,axis,mA,mB,off)

Creates a spring, dashpot, or line segment (rope) between two masses or fixed points.

- label label
- axis Axis along which to move.
- mA, mB Connected mass on each end, or location along axis.
- off Offset from axis, for parallel elements. [Optional]

Returns: a Linkage object

```
dia.spring('K',ax,25,175,-20);
dia.dash(  'B',ax,25,175, 20);
dia.rope(   '',ax,25,175,  0);
```

Diagram.arrow(v,loc,angle,scale,label)
Diagram.arrow(v,A,rest,off,scale,label)

A thick arrow, for indicating applied force, heat flow, current, etc. Becomes a diamond when length is near 0.

Can be called in two ways. The second is for translating systems and uses an Axis.

- v associated Y[v] index. If negative, the constant number -1-v (~v) is used.
- loc tail location, in the form [x,y].
- angle angle in which to point.
- A the axis along which to move.
- rest the point along the axis to call home.
- scale value scaling factor. [Optional: defaults to 2]
- label label

Returns: an Arrow object

```
dia.arrow(0,[60,25],0,4,'q');
```

Figure 4: Descriptions of mass, spring and arrow in help documentation

Lastly, the output graph is defined, as shown in figure 5. The arguments, in order, are:

- the size of the plot,
- the system being plotted,
- the outputs (of the state variable system to be plotted),
- the axis limits, [final time, min y, max y], and
- the labels for the plot.

```
//////     PLOT      ///////
myPlot=plot([600,250],sys1,[0,1],[90,-30,36],['u','y']);
```

Figure 5: Physical description of the system drawing

There are a few more lines necessary to complete the html file. The entire file (without comments) is in figure 6. This short file is sufficient to create an interactive simulation.

```
<html> <head>
<title>Damped Oscillator</title>
<script src="../raphael.js"></script>
<script src="../linear.js"></script>
<link rel="stylesheet" type="text/css" href="../style.css"/>
</head>
<body>
<h2>Damped Oscillator</h2>
<h3>with sawtooth input</h3>
<p>m=2, b=.1, k=1</p> <br>

<script>
//////    SYSTEM     ///////
m=2, b=.1, k=1;

A=[[   0,   1],   [-k/m,-b/m]];
B=[[  0],  [1/m]];
C=[[0,0],  [1,0]];
D=[[1],    [0]];

sys1=system(A,B,C,D).tMax(90);
sys1.input(0,function(t){return t%9});

//////    DIAGRAM    ///////
fig1=diagram([400,200],sys1);

fig1.wall([40, 60],[ 10,82],'E');
fig1.wall([50,142],[300,10],'N');
fig1.wall([40,141],[11,11]);

ax=fig1.axis([50,101],0);
m1=fig1.mass(1,'m',ax,200,-10,100,60,'b');
fig1.spring('k',ax,0,m1);
fig1.arrow(0,ax,200,-50,5,'u');

//////     PLOT      ///////
myPlot=plot([600,250],sys1,[0,1],[90,-30,36],['u','y']);
</script>
</body> </html>
```

Figure 6: Complete html file for system simulation, animation and plotting

Results

There are many features not described in the example above.

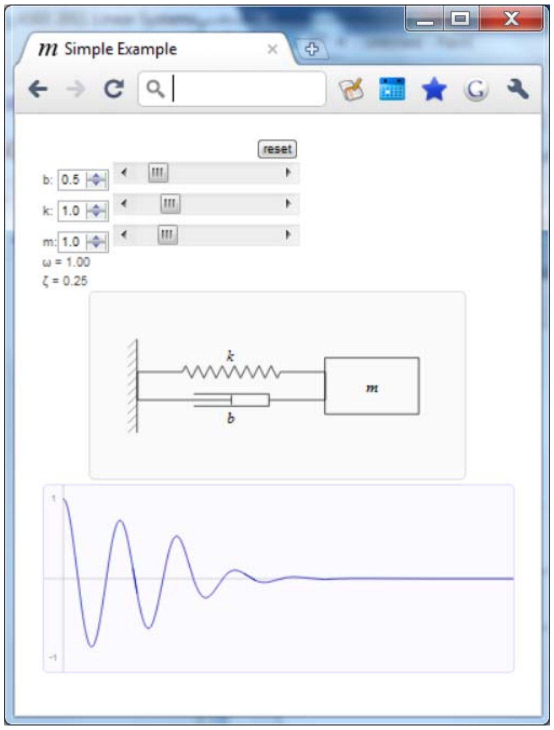
- Figure 7 shows a system that has sliders to define the mass spring and friction. The sliders remain active and can be moved during the simulation. In this simulation the friction was increased during the third oscillation, after which the oscillations quickly dampen.
- Figure 8 shows a system for which the input force was determined interactively by the position of the mouse. The low pass behavior is evident as fast inputs are attenuated.
- Figure 9 shows an electrical system. During the simulation the length of the arrow was determined by the inductor current, while the angle of the dial gauge was determined by the capacitor voltage.
- Figure 10 shows a thermal system with a pulsatile input. During the simulation the length of the arrows between the capacitances and resistances were determined by the heat flow, while the angle of the dial gauge was determined by the temperature. Again the low pass behavior is evident in this second order model.
- Figure 11 shows a motor. During the simulation the rotor spins, and the directions of the current in the wires on the rotor change as they go under the brushes (this is an extremely difficult concept to depict in a static drawing). The curved arrow shows the direction of the torque on the rotor.
- Not shown are translating electromechanical system (e.g.,speaker or microphone), rotating mechanical systems, mechanical systems that both rotate and translate, and mixed thermal/fluid systems Examples of these, and others, are available at http://www.swarthmore.edu/NatSci/echeeve1/Ref/LPSA/Animations/index.html.

Conclusions

A system has been developed to generate web-based simulations of linear physical systems. To develop an animation only a short script is required. The script defines the system (in state-space format), its inputs (as functions or user input) and outputs, as well as the drawing that is to be simulated. Two JavaScript libraries are used: one developed for the simulation and the animation, and the Raphaël Library for vector graphics. A wide variety of systems can be simulated and animated, including rotating and/or translating mechanical systems, electrical systems (including op-amps), electromechanical systems (rotating or translating), thermal systems, and mixed thermal/fluid systems.

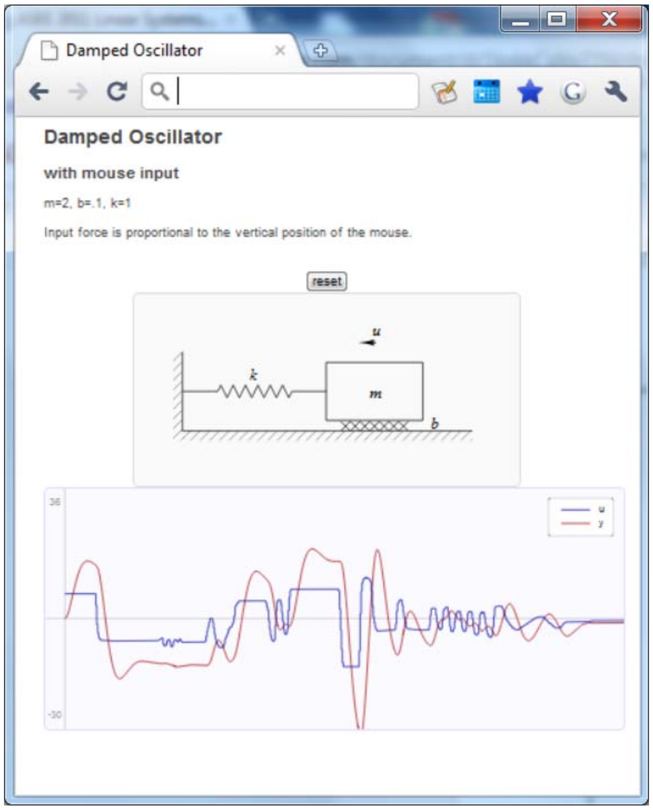
These animations are being incorporated into a web-based resource for a Linear Systems course. After students have had access to the simulations, the effectiveness of the simulations relative to static diagrams will be assessed.

The system discussed in this paper is available at:
http://www.swarthmore.edu/NatSci/echeeve1/Ref/LPSA/Animations/index.html



b: 0.5
k: 1.0
m: 1.0
ω = 1.00
ζ = 0.25

Figure 7: Sliders used to define mass, spring and friction. Friction increased during the third oscillation.



Damped Oscillator

with mouse input

m=2, b=.1, k=1

Input force is proportional to the vertical position of the mouse.

Figure 8: Input determined by mouse

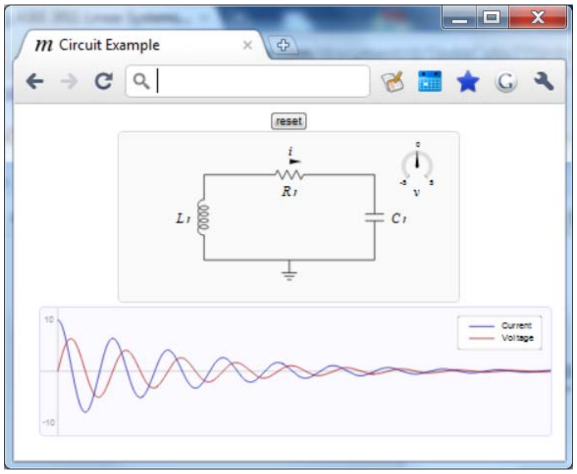
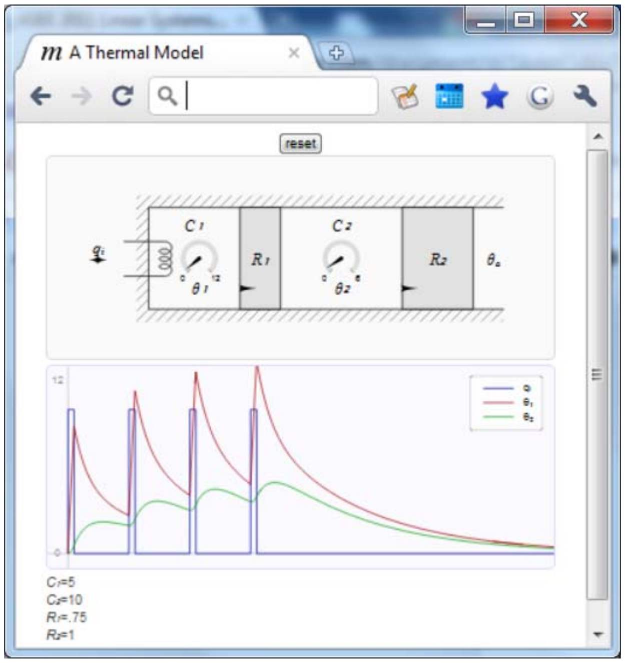


Figure 9: Electrical system



C_1=5
C_2=10
R_1=.75
R_2=1

Figure 10: A thermal system, pulsatile input

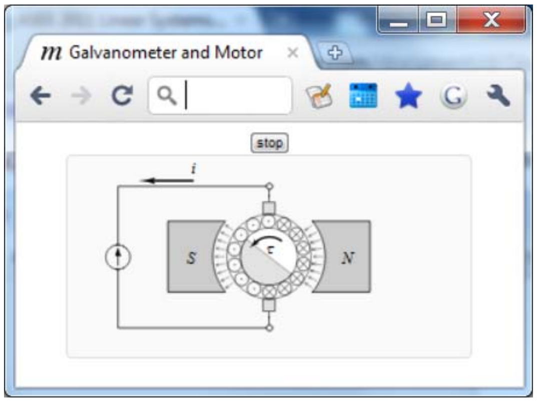


Figure 11: An electromechanical system (motor)

References

[1] “Learning Styles and Teaching Styles in Engineering Education,” Felder & Silverman. Engineering Education, 78:7, 1988, 674-681.

[2] “Evaluating the educational impact of visualization,” Naps, et al., Proceeding ITiCSE-WG, 2003, pp 124-136.

[3] “Visual Learning for Science and Engineering,” McGrath & Brown, IEEE Computer Graphics and Applications, 25:5, 2005, pp 56-63.

[4] “Incorporating *Animation* Concepts and Principles in STEM Education,” Harrison & Hummell, Technology Teacher; v69 n8 p20-25 May-Jun 2010

[5] “Animated Instructional Software for Mechanics of Materials: Implementation and Assessment,” Philpot & Hall, *Computer Applications in Engineering Education,* John Wiley and Sons, 14:1, 2006, pp 31-43.

[6] “Is there a better way to present an example problem?,” Philpot, Hall, Flori, Hubing, Oglesby, & Yellamraju, ASEE Conference Proceedings, 2003.

[7] “Comprehensive Evaluation of Animated Instructional Software For Mechanics Of Materials,” Philpot & Hall, ASEE/IEEE FIE Conference, 2004.

[8] “Interactive Web Based Animation Software: An Efficient Way to Increase the Engineering Student’s Fundamental Understanding of Particle Kinematics and Kinetics”, Stanley, Proceedings of ASEE Zone 1 Conference, 2008

[9] “An Efficient Way to Increase the Engineering Student’s Fundamental Understanding of Particle Kinematics and Kinetics by Utilizing Interactive Web Based Animation Software,” Stanley, ASEE Computers in Education Journal, 18:3, 2008, pp 23-41.

[10] Experiential Learning: Experience as the Source of Learning and Development. Kolb, D. A., Ed. Prentice-Hall Inc, New Jersey, USA, 1984.

[11] “The educational encyclopedia, Engineering Animations and Java applets,” www.educypedia.be/education/physicsoscillations.htm (accessed Jan 3, 2011)

[12] “Math, Physics, and Engineering Applets,” www.falstad.com/mathphysics.html, (accessed Jan 5, 2011).

[13] “Signals, Systems, and Control Demonstrations,” http://www.jhu.edu/signals/, (accesed Jan 3, 2011)

[14] “A Java-based system for building animated presentations over the Web,” Bonifaci, Demetrescu, Finocchi, & Luigi, Science of Computer Programming, 53:1, 2004, pp 37-49

[15] “MyPhysicsLab — Physics Simulation with Java,” www.myphysicslab.com, (accessed Jan 5, 2011).

[16] “Redesigning Undergraduate Control Courseware,” Chang, Jaroonsiriphan & Chang, International Conference on Engineering Education, 2004.

[17] Fundamentals of Signals and Systems Using the Web and MatLab, Kamen & Heck, Prentice Hall, 2002

[18] Mastering Simulink, Dabney & Harman, Prentice Hall, 2003

[19] “Raphaël—JavaScript Library,” www.raphaeljs.com (accessed Jan 3, 2011)