COMPUTER ANIMATION: A VISUALIZATION TOOL FOR DYNAMIC SYSTEM SIMULATIONS

John Watkins, George Piper, Kevin Wedeward, E. Eugene Mitchell
Department of Weapons & Systems Engineering
U.S. Naval Academy
Annapolis, MD 21402-5025

ABSTRACT

This paper describes how animation is being utilized to teach system dynamics and control in the Systems Engineering Department at the United States Naval Academy. Included is a description of how animation has been incorporated into the classroom using the computer software tools VisSim and MATLAB. The animation capabilities of these software tools are demonstrated with several classroom exercises.

INTRODUCTION

Today's students are exposed to multimedia in all aspects of their lives, from MTV to the World Wide Web. They have become accustomed to receiving information in this manner. Multimedia is also finding its way in to computer simulation. Students get excited about commercially available entertainment programs such as Microsoft's Flight Simulator and Maxis' SimCity which contain realistic, highly-detailed simulations. Educators are taking advantage of these animated simulations and incorporating them into to their classrooms. Computer simulation is also being used to enhance students understanding of dynamic systems [1]. Incorporating animation with simulation not only increases the student's understanding, it captures their attention and increases their motivation. Animation bridges the gap between abstract mathematical equations and the physical behavior of the system.

In the past, computer simulation and animation required the use of conventional programming languages such as FORTRAN, PASCAL, or C. While this approach allows for effective dynamic simulation, it is easy for the user to get sidetracked with programming issues and lose sight of the original problem. Animation, in particular, requires a specialized knowledge of the computer platform. With today's windowing environment, there are an increasing number of commercially available programs that allow the user to easily develop simulations of general dynamic systems. These programs include MATLAB, Matrix-X, Control-C, and VisSim. Computer simulation programs encourage students to explore complex and realistic systems. The interactive environment and graphic capability of these programs provides instant feedback to the students. In addition to dynamic simulation capabilities, many of these programs allow the user to incorporate animation into the simulation.

Animation is being incorporated into the systems engineering courses at the United States Naval Academy to help teach system dynamics and control. The Systems Engineering curriculum at the United States Naval Academy focuses mainly on linear systems theory, feedback control, and mechatronics. It is a four year, undergraduate, ABET accredited, engineering program. Throughout the curriculum students learn how to model, simulate, and design various types of control systems. The computer software the students use to analyze, simulate, and implement

their designs are MATLAB, VisSim, and the C programming language. These tools were selected based on their relevance to industrial practices, cost, and availability to the student. Methods of animation in VisSim and MATLAB as well as three examples used in the curriculum are the topics presented below.

ANIMATION WITH VISSIM

This section describes VisSim's animation capabilities and how they are being used at the Naval Academy. VisSim is a comprehensive modeling/simulation tool that allows the user to build, debug, and simulate dynamic system models all within the same environment [2]. VisSim features a graphical user interface in which the user creates block diagram models in a standard Microsoft Windows environment. VisSim provides two easy methods of animating a simulation [3].

The first method utilizes VisSim's *animation* block that lets you animate an image during simulation. An image is a small bitmap displayed on the computer screen. Animation occurs by moving the bitmap image around the screen or by changing its size or appearance. An image can have up to 16 different bitmaps associated with it. Each bitmap is designated as a state from 0 up to 16. To change the bitmap of an image you simply change the state of the image. The *animation* block requires five inputs from the simulation: the state of the image (an integer value corresponding to the desired bitmap), the image's x and y coordinates, and the image width and height. This method is similar to sprite animation found in tradition animation environments [4].

The second method of animation utilizes VisSim's *lineDraw* block to draw a line of specified color, width, and style for each simulation step. The four inputs for this block are the x and y coordinates of both ends of the line. The first method can provide more detailed images than the second method and is therefore more computational intensive. However, animation with the first method is usually slow and choppy. The second method usually provides faster and smoother animation.

Described below are two design projects from an introductory modeling and simulation course that involve animated simulations. In these projects the students were required to develop a model and simulation of a physical system then evaluate its performance. The simulations were developed in VisSim. The intent was not to teach animation, but to use animation as a visualization tool and provide motivation

Hydraulic Network Project

The hydraulic network for this project is shown in Figure 1. It consists of a pump that produces a constant pressure, P_d, and two interconnected tanks with hydraulic capacitance C_1, C_2. The height of the water levels in each tank is indicated by h_1, h_2. Water from a reservoir is pumped into Tank #1. The water in Tank #1 is allowed to flow through valve R_1 into Tank #2. The water in Tank #2 is then allowed to flow through valve R_2 back into the reservoir.

The objective of the project was to find the maximum pump pressure such that neither tank overflows. The students were required to model the system and develop a simulation in VisSim. Inputs to their simulations were pump pressure, hydraulic capacitance, and resistance. Outputs of their simulations were the time histories of each tank's water level. The students were able to change the inputs to the simulation and look at the resulting changes in the simulation output.

To help the students interpret what the time history of the water levels was telling them, the students were required to animate the water levels in each tank. The instructors developed a

compound block that contained animation of each tank's water level. The instructors distributed the water animation block so that the students could import it into their simulation. The water level animation block required only the water levels h_1 and h_2 as input from the student's simulation.

The water level animation is shown in Figure 2. Water level animation was produced using VisSim's *lineDraw* block. Each tank is represented by three stationary lines, two vertical and one horizontal. A single blue horizontal line that can move up or down represents the water level in each tank. The position of the water level lines is updated during the simulation as the values of h_1 and h_2 change.

Vehicle Guidance Project

In this project the students developed an animated simulation of a three wheeled vehicle tracking a stationary or moving target. A diagram depicting the vehicle, target, and coordinate systems is shown in Figure 3. As a minimum, the students simulations were to include the kinematics of a three wheeled vehicle, a target sensor model, a guidance control law, and a target. The instructors gave the students a set of parameters and conditions in which their vehicle simulation worked.

As with the Hydraulic Network Project, the instructors provided a compound block to the students that contained animation of vehicle and target. The vehicle animation block required the Cartesian coordinates of the vehicles front and back axles and the Cartesian coordinates of the target.

The vehicle animation is shown in Figure 4. Vehicle animation was produced using VisSim's *lineDraw* block. The vehicle is represented by a thick line the target is represented by a large dot. The position of the vehicle and target is updated during the simulation as the values of their coordinates change.

Animation was a key motivator in this project. When this project was first introduced, the student's interest toward the project was lukewarm. After seeing a demonstration of a completed simulation, the student's interest grew significantly. The animated motion of the vehicle and target gave the project a "seek and destroy" flare that the students seemed to enjoy. Students were motivated to do more than was required for the project. On their own initiative, students experimented with different types of guidance laws, sensor models, and target strategies.

ANIMATION WITH MATLAB

MATLAB is a another computer-aided-design package which is used extensively in our curriculum for control system analysis and design. Students and professional engineers alike use the package to analyze, simulate, and design control systems for dynamic systems. The end product of this effort is often a graph showing the time response of the closed loop control system. This graph of the time response may provide the professional engineer, who has a strong grasp of the physical system that they are trying to control, adequate information about the performance of the control system. Students, however, often lack the adequate experience and background to translate the time response graph to the behavior of the underlying physical system.

By adding animation, the student is able to gain additional insight into how the physical system will behave when the control system they designed is added to the system. While the development of the animation software requires additional time by the instructor, software

development is relatively easy to do and the additional insight gained by the student justifies this time. Furthermore, once developed, this animation software can be used each time the class is taught.

In this section, we will demonstrate how MATLAB was used to animate an example often considered in control system classes, the ball and beam problem [5]. As shown in Figure 5, the ball and beam problem consists of a beam which is free to rotate about a pivot point. A ball is placed on the beam and the control objective is to balance the ball at a desired location along the beam. This is an interesting problem because the open loop system is unstable and nonlinear. A DC motor is used to control the angle of the beam. It is assumed that sensors are available to measure the angular velocity, $\omega(t)$, the angular position, $\theta(t)$, the velocity of the ball along the beam, v(t), and the position of the ball along the beam, c(t).

Below, we will discuss how this animation software was incorporated into a senior level state-space control systems course. (Note that the animation was also used for demonstration purposes in a junior level classical controls course.) The students were given a block diagram and were asked to develop a state space model. The linearized state space equations are given as

$$\frac{d}{dt}\begin{bmatrix}\omega\\ \theta\\ v\\ c\end{bmatrix}=\begin{bmatrix}-1/\tau_m & 0 & 0 & k_c/\tau_m\\ 1 & 0 & 0 & 0\\ 0 & g & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\omega\\ \theta\\ v\\ c\end{bmatrix}+\begin{bmatrix}k_m/\tau_m\\ 0\\ 0\\ 0\end{bmatrix}e(t)$$

where k_m = 20 rad/sec-volt, τ_m = 5.43 sec, g = 9.81 m/sec^2, k_c=98.1 rad/sec-m, and $e(t)$ is the motor input voltage.

The students were asked to design a compensator of the form $e(t)=N_A(r(t)-K\vec{x}(t))$ where N_A is a scalar gain, $r(t)$ is a reference input, K is the state feedback gain, and $\vec{x}(t)$ is the state vector. The design required specific closed loop poles and a zero steady state error for a step input. After completing the design, the students were asked to verify their design by finding the closed loop poles and the step response.

Finally, the students were asked to simulate the system with all the initial conditions set to zero expect for $\theta(0)$ which was set to 0.1 rad as shown in Figure 5. The desired position, r(t), was specified to be -0.3 m. Because of the nonzero initial conditions, the students were required to simulate the system using the MATLAB command *lsim*, rather than the typical *step* command. While the simulation results which are shown in Figure 6 completely describe the behavior of the ball and beam system, it is difficult for the student to visualize the behavior of the physical system from these results.

Consequently, the students were asked to go one step further. They were given the MATLAB m-file ballbeam.m found in Appendix A. From there simulation results shown in Figure 2, the students were able to extract the angle of the beam and the position of the ball. These were supplied as inputs to the *ballbeam* function which produced an animation of the ball and beam with the initial condition shown in Figure 5. This additional visualization aided the students in their understanding of the problem and provided them additional verification as to whether their

design did or did not work. A discussion of the details of ballbeam.m can be found in Appendix A.

CONCLUSION

This paper described how animation was used to teach system dynamics and control in the Systems Engineering Department at the United States Naval Academy. Animation was incorporated into the classroom using the computer software tools VisSim and MATLAB. The animation capabilities of these software tools were demonstrated with several classroom exercises. The animation software was relatively easy to implement and added significantly to the student's understanding of the problems they were trying to solve.

REFERENCES

1. Glower, J.S., "Design of Computer Simulation Experiments for an Introductory Controls Course", *Proceedings of the American Control Conference*, Seattle, WA, June 1995.
2. Piper, G.E. and Mitchell, E.E., "VISSIM: An Affordable Graphical Simulation Tool for Dynamic Systems", *Computers In Education Journal*, vol. 5, no. 4, pp. 35-39, Oct.-Dec. 1995.
3. VisSim User's Guide, Visual Solutions, Inc.,Westford, MA 01886, 1993.
4. Lampton, C., Flights of Fantasy / Programming 3-D Video Games in C++, Waite Group Press, 1993.
5. Dorf, R. and Bishop, R., Modern Control Systems, Addison-Wesley, 1995, p. 104
6. MATLAB Reference Guide, The MathWorks, Inc., Natick, MA 01760, 1996.

Appendix A

The MATLAB m-file used to animate the ball and beam problem is given below. Note that the line numbers on the left are not part of the original program, but have been added to aid the discussion.

Using MATLAB's handle graphics [6], the *ballbeam* function was reasonably easy to write. Lines 10-26 are used to initialize parameters and calculate the x and y coordinates of the ball and the x and y coordinates of the beam as they change versus time. Two basic MATLAB graphing functions are used to draw the ball and beam system. In line 29, the *plot* function is used to plot the beam. In lines 31 and 33, the *patch* command is used to draw the beam stand and ball, respectively. Let's take a close look at the *plot* command in line 29. The first three function parameters are standard, the x coordinates, the y coordinates, and the color of the line which was specified to be *y* for yellow. The next two parameters set the *EraseMode* to *xor*. We found this setting to produce the best results for animation. The last two parameters provide a wider beam by setting the *Linewidth* to 2.5. The output of the plot command is the "handle" L which allows this plot to be referenced later in the m-file.

In line 35, the program pauses with the initial condition of the ball beam displayed as shown in Figure 5. Once the user presses a key, the program proceeds to display the ball and beam as they change versus time. With MATLAB's handle graphics, this can be done by simply changing the x and y coordinates of the beam, lines 37-38, and the x and y coordinates of the ball, lines 39-40 and issuing a *drawnow* command in line 41 which forces MATLAB to update the screen.

```
1.  function ballbeam(theta,ballx)
2.  %
3.  % BALLBEAM     BALLBEAM(THETA,BALLX) displays the animated response
4.  %               of a ball and beam apparatus where THETA is the angle
5.  %               of the beam in radians and BALLX is the distance of the
6.  %               ball from the center of the beam in meters.  The beam
7.  %               is assumed to be 80 centimeters long.

8.  % Written by K. Wedeward Fall 96
9.  % Modified by J. Watkins Fall 96
10. %
11. ballx=ballx*100;                          % convert from m to cm
12. beam_length = 80;                         % cm
13. bl2 = beam_length/2;

14. radius = 2.0;                             % ball radius
15. arcstep = 36;                             % angle increment (in degrees)
16. % to sweep out arc lengths when
17. % creating sides of ball

18. ltheta = length(theta);                   % find length of theta
19. bally = - ballx .* tan(theta) + radius;   % find y positions of ball

20. j = 0:arcstep:(360-arcstep);              % create ball sides using arclengths
21. arcx = radius * cos((j+arcstep) * pi/180);% x coordinates of ball sides
22. arcy = radius * sin((j+arcstep) * pi/180);% y coordinates of ball sides

23. beamx1 = -bl2 * cos(theta);               % compute coordinates of beam
24. beamx2 =  bl2 * cos(theta);               % ends: left end  => (beamx1,beamy1)
25. beamy1 =  bl2 * sin(theta);               %         right end => (beamx2,beamy2)
26. beamy2 = -bl2 * sin(theta);
27. % construct initial beam using a wide line
28. % LineWidth of 0.5 is normal
29. L = plot([beamx1(1) beamx2(1)], [beamy1(1) beamy2(1)], 'y', 'EraseMode', 'xor','LineWidth',
    [2.5]);

30. axis([-45 45 -45 45]);                    % construct square axes

31. patch([-2 0 2 -2], [-5 0 -5 -5], 'y');    % construct beam stand

32. % construct initial ball
33. H = patch(arcx+ballx(1), arcy+bally(1), 'b', 'EraseMode', 'xor');

34. 'Press a key to continue'
35. pause

36. for i = 2:ltheta,                         % loop over data values
```

```
37. set(L, 'XData', [beamx1(i) beamx2(i)]);     % change x coordinates of beam ends
38. set(L, 'YData', [beamy1(i) beamy2(i)]);     % change y coordinates of beam ends
39. set(H, 'XData', arcx+ballx(i));             % change x coordinate of ball
40. set(H, 'YData', arcy+bally(i));             % change y coordinate of ball
41. drawnow;

42. end;
```

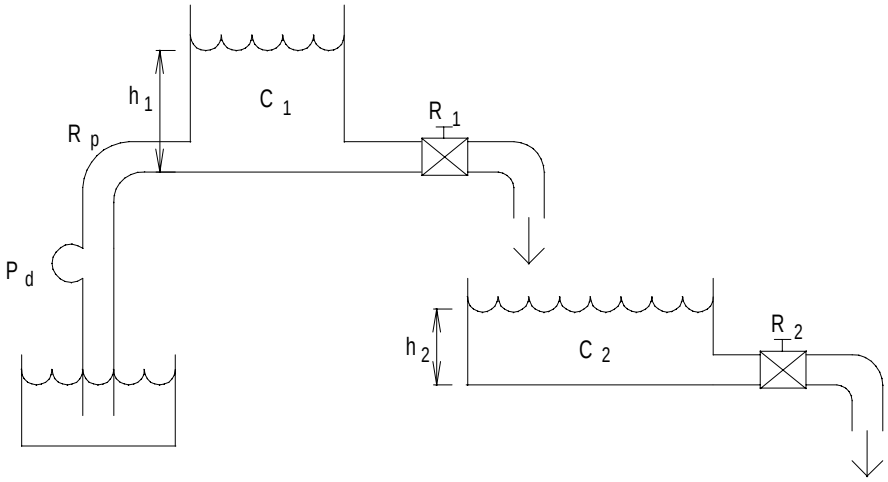

Figure 1. Hydraulic Network Project

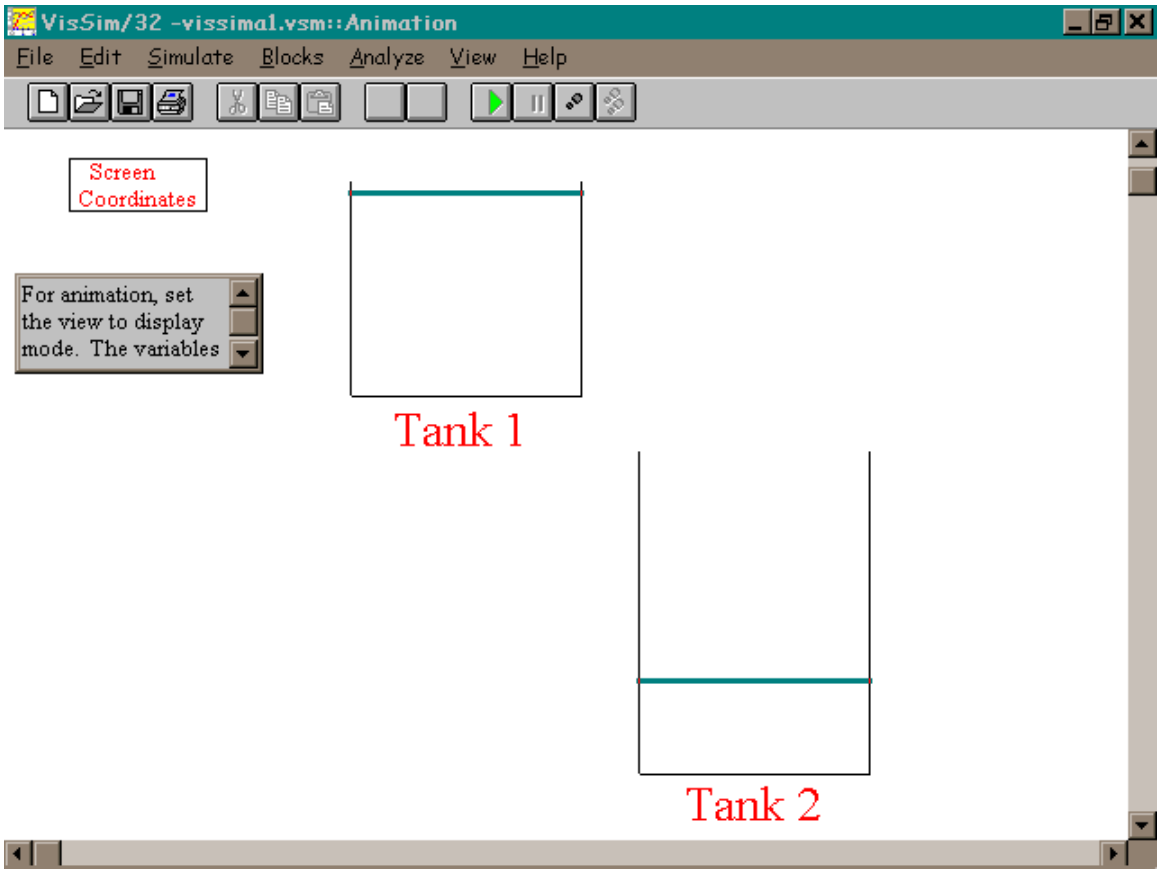

Figure 2. Water Level Animation for the Hydraulic Network Project

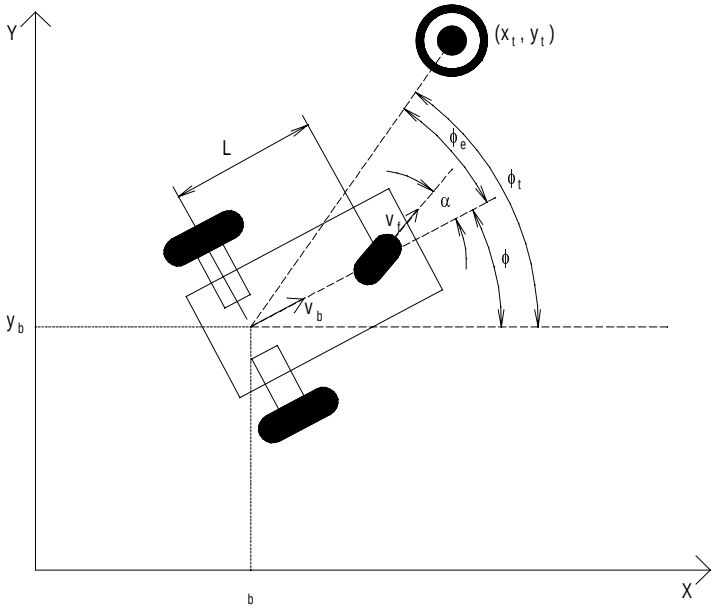

Figure 3. Vehicle Guidance Project

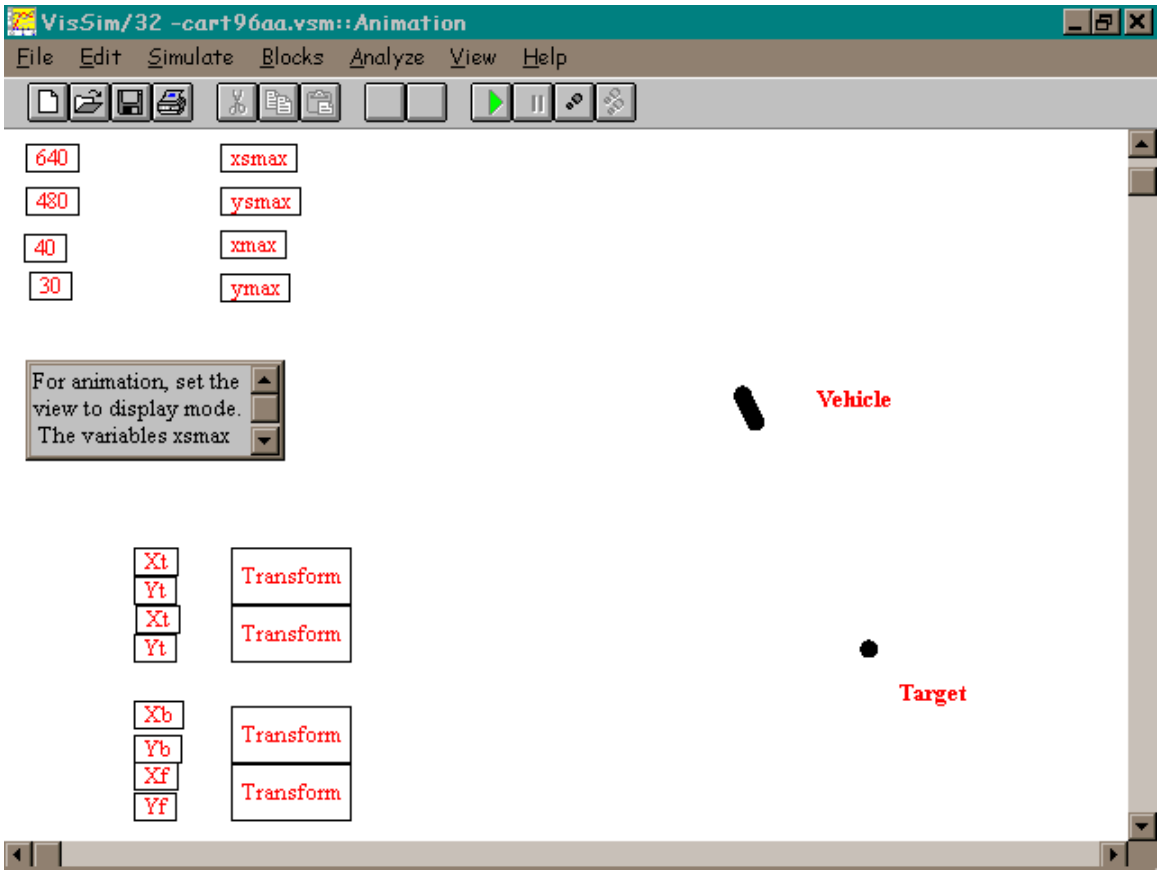

Figure 4. Vehicle Animation for the Vehicle Guidance Project

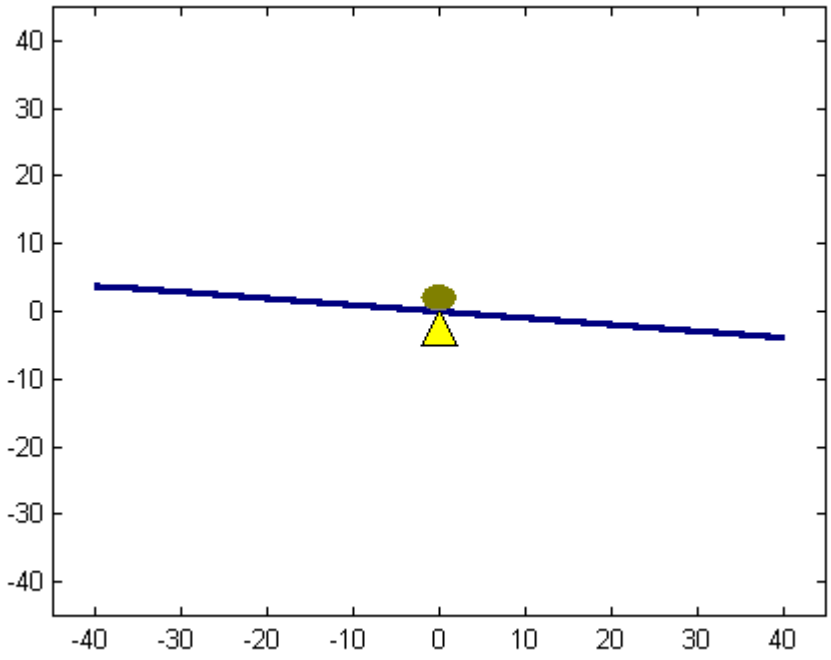

Figure 5. Ball and Beam

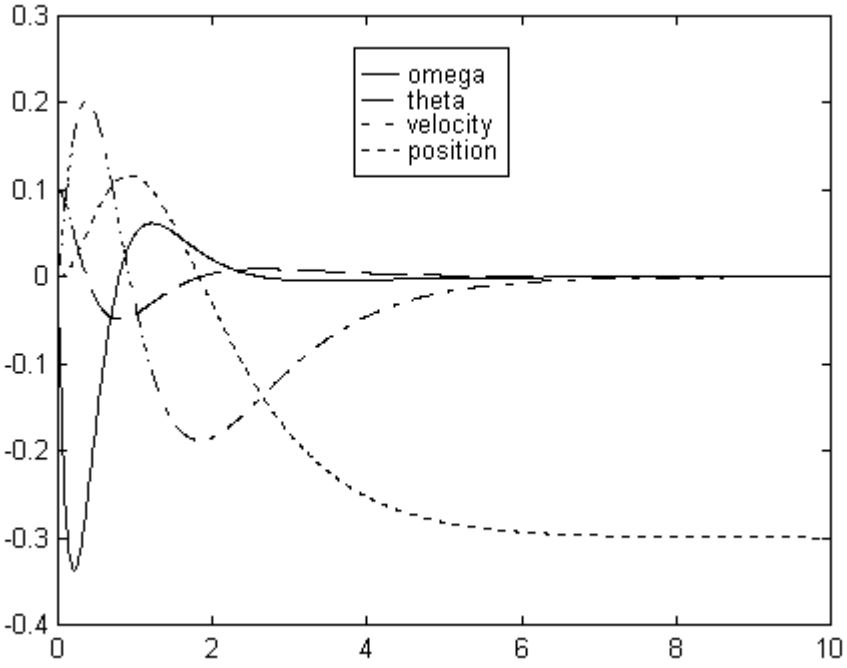

Figure 6. Simulation Results