# Dynamic System Animation Within a Simulink Laboratory Environment

Edwin Zivi, Jenelle Armstrong Piepmeier
U.S. Naval Academy

Abstract

Recent innovations in the animation of student developed, laboratory simulations have been found to significantly improve student understanding and enthusiasm. This animation can be implemented using Matlab s-functions that are typically called from a Simulink simulation. These rudimentary animations have been found to be relatively easy to construct and well received by students. General observations and recommendations are developed from specific examples, hints, and experiences. Runtime animation has been found to be a valuable complement to the existing capability to visually construct the simulation model. S-function animations have been found to be composed of two primary elements: reusable interface/initialization Aboilerplate@ and animation specific graphics. To date, animations have been developed solely as part of faculty course preparation. These animations have been found to efficiently leverage Matlab=s high level programming features to provide an effective teaching tool.

## 1. Introduction

Animation has become a valuable visualization tool for teaching computer simulation of dynamic systems. Engineering education literature is replete with animation examples utilizing various programming environments for a variety of physical systems[1,2,3,4]. Although numerous, most of these papers are very discipline-specific. Equally bewildering is the choice of animation software which ranges from freely distributed packages such as *Ansim*[5] to sophisticated commercial packages such as *Altia Design*[6]. The authors have found that simple animations using Matlab s–functions are a very practical and effective method for assisting and motivating student Simulink laboratory projects. This paper provides a simplified procedure for constructing s–function animations and presents a number of examples.

## 2. Background

The s–function animations, described herein, were supplied to third year System Engineering students at the U.S. Naval Academy. Each animation involves a single Simulink animation block that calls a single animation s–function. The Simulink animation block serves as the interface between the student's Simulink model and the animation. The

animation s–functions are implemented as Matlab m–file scripts that follow examples produced by The Mathworks[7]. S–function documentation[8] and a template file, `sfuntmpl.m`, are included in the Simulink Version 3 software distribution.

## 3. S-function Basics

Matlab s–functions provide a standard way of creating specialized blocks for use in Simulink models. Figure 1 provides a simplified animation s–function template. Line 1 defines the s–function with none of the customary outputs and the following inputs:

| | |
|---|---|
| t | The current simulation time |
| x | The state vector – typically not used |
| u | The input vector – used to define position or state of the animated objects. |
| flag | An integer value that indicates the task to be performed by the s–function |
| str | Unused |
| ts | Unused |

```
1   function generic_anim(t,x,u,flag,ts);
2
3   %   Generic animation s-function template.
4   %   All output variables have been suppressed for brevity
5   %   Jenelle Piepmeier & Ed Zivi, December 2000
6
7   %   Must declare variables that need to be remembered between calls 'persistent'
8   persistent FigAnim
9
10  % Initialize, update, or terminate animation depending on value of 'flag'
11  if flag==0,    % Initialize the figure for use with this simulation
13      animinit('Animation Title');
14      [fflag FigAnim] = figflag('Animation Title');
15      hold on;   % make changes cumulative
16      %   Set position and size of figure and axes.
17      %   If desired, add titles and labels.
18      %   Using 'patch' or 'line' commands, draw each object in initial position.
19      %   Be sure to assign a persistent handle to each object you want to modify in
20      %       subsequent function calls
21
22  elseif flag==2, % Update the properties of the graphics objects
23     if any(get(0,'Children')==FigAnim),   % check that animation window exists
24        if strcmp(get(FigAnim,'Name'),'Animation Title'),
25           set(0,'currentfigure',FigAnim); % make animation window the current window
26           % Define the position data (X & Y) for each object to be moved.
27           % Use the 'set' commands to objects and change graphic attributes
28           drawnow   % updates graphics
29        end
30     end
31
32  elseif flag==9,    % Simulation has terminated
33      % any final annotation can be added here
34  end
```

Figure 1 – Simplified Animation Template

Animations are usually driven by the s–function inputs `t, u,` and `flag`. Simulink automatically provides current simulation time and an initialize, update, or terminate flag.

The simulation is driven by the user input, `u`, defined in the calling Simulink block. Animation "memory" is achieved by defining variables to be *persistent* (see line 8). Initialization of the s–function template is accomplished in lines 11 through 20. The animation figure, with a user specified title, is created with `animinit`. The pointer to the animation figure object is stored in the persistent variable `FigAnim` in line 14. During initialization, polygons and lines can be created to represent the physical system that is being simulated by the student. Every graphic object that will be modified in subsequent s–function calls should have a persistent handle.

Lines 22-30 represent the working part of the s–function. This is the code that animates the figure. The animation is achieved by changing properties such as the `'Xdata'` and `'Ydata'` position of the graphics objects using the `set` command. Similarly, altering the `'String'` property can be used to change animation text. The `'color'` property is often used to display either continuous or discrete changes in a parameter. The command `drawnow` flushes all pending graphics updates.

Lines 32-34 allow for final changes to the animation figure at the end of the simulation.

4. Simple Graphic Objects

This section discusses the specifics of animating graphic objects in a MATLAB figure. The objects must first be created during initialization when `flag==0`. Changing the attributes of these objects during each update (`flag==2`) creates the animation effect. The three most commonly used Matlab commands are `plot, patch`, and `set.`

Figure 2 contains a code fragment to draw a simple pendulum with a lumped mass using a line and a circular patch object. This code would be used to initialize the figure and pendulum objects. Lines 1 through 3 define the initial orientation and the geometry of the pendulum. Note that the variables `R` and `massradius` should be persistent variables since they will be referenced during subsequent s–function calls. Line 4 positions and sizes the figure window while line 5 establishes an engineering coordinate system slightly larger than the pendulum. The `set` command is used to change the attributes of a graphics object pointed to by the first parameter. The standard format is `set(H,'PropertyName',PropertyValue)` where `H` is the appropriate graphics object handle. In lines 4 and 5, the function `gcf` (get current figure) and `gca` (get current axis) return the desired object handles. To obtain a uncluttered figure, line 5 sets the axes visibility off. Line 6 uses the `plot` command to draw a solid blue line from (0,0) to the point (`R*sin(theta),-R*cos(theta)`) and returns the persistent graphics object handle, `hbar`. Lines 9 through 10 create a small polygon with 13 vertices to represent a circular lumped mass at the end of the pendulum. The `patch` command accepts x and y data for the vertices and returns the persistent handle `hmass`.

```
1   theta=0;
2   R=1;
3   massradius=.12;
4   set(gcf,'Position',[120,120,350,350]);
5   set(gca,'visible','off','xlim',[-1.1*R,1.1*R],'ylim',[-1.1*R,1.1*R])
6   hbar=plot([0,R*sin(theta)],[0,-R*cos(theta)],'b-');
7   hold on
8   r=0:pi/6:2*pi;
9   hmass=patch(R*sin(theta)*ones(size(r))+massradius*sin(r),-...
10      R*cos(theta)*ones(size(r))+massradius*cos(r),'r');
```

Figure 2 – Creating a Graphic Object

In this example, animation is achieved by changing the pendulum's 'Xdata' and 'Ydata' properties, avoiding the need to redraw the pendulum arm or mass. Passing the pendulum angle, theta, as the user defined input `u` allows the s–function to compute new `hbar` and `hmass` positions using the persistent geometry variables `R`, `r`, and `massradius`. Figure 3 provides an example code fragment, punctuated by a `drawnow` command to update the display.

```
theta=u;
massX=R*sin(theta)*ones(size(r))+massradius*sin(r);
massY=-R*cos(theta)*ones(size(r))+massradius*cos(r);
barY = [0 -R*cos(theta)];
barX = [0 R*sin(theta)];
set(hbar,'Xdata',barX,'Ydata',barY,'linewidth',2);
set(hmass,'Xdata',massX,'Ydata',massY);
drawnow;
```

Figure 3 – Changing Graphic Object Position

Changing the color of the graphic objects can be used to identify an event or condition. For an animation of a radar antenna servomechanism, the target was defined using a *patch* command. The target color changed from black to red if the antenna was pointing at the target to within $\pm 2^{O}$. This color change indicated when the student's radar illuminated a weaving target. Figure 4 provides a sample code fragment where the pertinent patch color properties are `'edgecolor'` and `'facecolor'`.

```
if(abs(error)<2*pi/180) % error less than 2 degrees
    set(htarget,'edgecolor','k','facecolor','r');  % red
else
    set(htarget,'edgecolor','k','facecolor','k');  % black
end
```

Figure 4 – Discrete Graphic Object Color Change

In a missile animation, color was used to indicate proximity. In Figure 5, the missile object gradually changes from green to red as the missile approaches its target.

```
if range > 500,           % faraway => "cold"  (green)
    colorv = [0 1 0];     % [red green blue]
else
    green = range/500;    % close => "hot" (red)
    red   = 1-green;
    colorv = [red green 0];
end
set(hmissile,'Color',colorv); % change color
```

Figure 5 – Continuous Graphic Object Color Change

Changing its 'string' property provides a convenient method to modify displayed text. In Figure 6, missile flight time and range to target is updated each time the s–function is called.

```
range_str = sprintf('Flight time = %6.2f, range = %6.2f', t, new_range);
set(htext,'string',range_str)
```

Figure 6 – Text Annotation Change

In creating an animation, it is often difficult to determine the specific MATLAB property name or exact parameter value. Often, the Matlab Graphics Properties Editor is very useful for browsing for the correct property name or value.

5. Missile Example

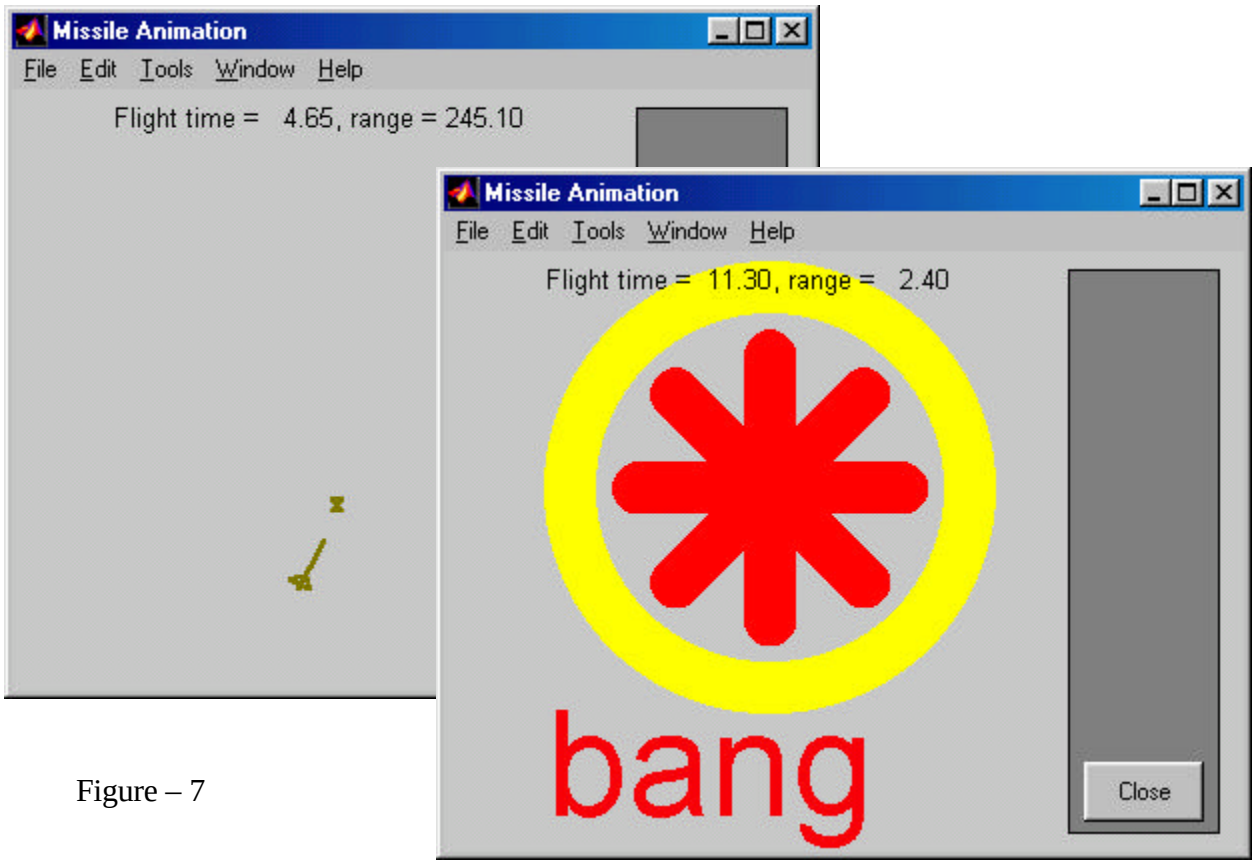


Figure – 7

Missile Intercept Plots

## 6. Conclusion

Of the many different methods to implement simulation animation, basic s−function programming has been found to be simple and effective. Satisfactory animations can be achieved as part of a weekly laboratory assignment preparation. The students have become quite fond of these rudimentary animations. Simulation time visualization improves the student's appreciation of system dynamics and can be a valuable troubleshooting tool.

## Bibliography


1. J. Watkins, G. Piper, K. Wedeward, and E. E. Mitchell, *Computer Animation: A Visualization tool for Dynamic System Simulations*. Proceedings ASEE, June 15-18, 1997.
2. G. P. Adams and I.C. Jong, *Using Matlab to Animate the Generation of a Space Centrode in Kinematics*, Proceedings ASEE, June 15-18, 1997.
3. G. G. Karady D. Tylavsky, *Use of Animation for Improvement of Student Understanding of Energy Conversion*, Proceedings ASEE, June 28-July 1, 1998.
4. B. Jenkins, *Simulation in Optical Fiber Communication*, Proceedings ASEE, June 28-July 1, 1998.
5. L. Dean, Ansim: *The Simulation Animation Block*, The Mathworks Inc, ftp://ftp.mathworks.com/pub/tech-support/solutions/s3399/ansim/v5/, January 2000.
6. *Why Simulation Graphics*, Altia Corp., http://www.altia.com/whitepaper/sae.html
7. N. Gulley, *PNDANIM2 S-function for animating the motion of a double pendulum*, The MathWorks Inc., June 1993.
8. *Writing S-functions*, Supplied with Simulink Version 3 as *sfunctions.pdf*, The Mathworks, 1998.



EDWIN L. ZIVI

Edwin L. Zivi received the B.S. degree in Engineering Science & Mechanics at Virginia Tech. in 1975 and the MS and PhD degrees in Mechanical Engineering at the University of Maryland in 1983 and 1989 respectively. He is an Assistant Professor of Systems Engineering at the U. S. Naval Academy. Research interests include fault tolerant distributed control and communication networks, electromechanical system dynamics, and shipboard applications of integrated power and machinery control systems. Prior to 1998, Ed was a *Senior Research Engineer and Technical Advisor* at the Naval Surface Warfare Center (NSWC), Annapolis, Maryland.

JENELLE ARMSTRONG PIEPMEIER

Jenelle Armstrong Piepmeier received a Bachelor of Science in Engineering from LeTourneau University in 1993, Master of Science in Mechanical Engineering and Doctor of Philosophy in Mechanical Engineering from Georgia Institute of Technology in 1993 and 1999, respectively. Since 1999, she has been on the faculty of the Systems Engineering Department of The United States Naval Academy as an Assistant Professor. Her primary research interest is vision-guided robotics.