Modeling Attitude Dynamics in Simulink: A Study of the Rotational and Translational Motion of a Spacecraft Given Torques and Impulses Generated by RMS Hand Controllers

Rebecca H. Mauldin

Marshall Space Flight Center

December 17, 2010

Reviewed by NASA Mentor
Don Krupp
EV82

Signature Here

Modeling Attitude Dynamics in Simulink: A Study of the Rotational and Translational Motion of a Spacecraft Given Torques and Impulses Generated by RMS Hand Controllers

Rebecca H Mauldin[1]
University of Alabama in Huntsville, Huntsville, AL, Zip 35899

In order to study and control the attitude of a spacecraft, it is necessary to understand the natural motion of a body in orbit. Assuming a spacecraft to be a rigid body, dynamics describes the complete motion of the vehicle by the translational and rotational motion of the body. The Simulink Attitude Analysis Model applies the equations of rigid body motion to the study of a spacecraft's attitude in orbit. Using a TCP/IP connection, Matlab reads the values of the Remote Manipulator System (RMS) hand controllers and passes them to Simulink as specified torque and impulse profiles. Simulink then uses the governing kinematic and dynamic equations of a rigid body in low earth orbit (LE0) to plot the attitude response of a spacecraft for five seconds given known applied torques and impulses, and constant principal moments of inertia.

I. Introduction

Considering a rigid body, defined as a system of particles where the relative distances between particles is fixed, French Mathematician Michel Chasles states that the body's translational motion along a line and its rotational motion about that line completely describes its motion. Therefore, the orientation of a rigid body is described by its position, velocity, angular velocity, and angular position. The angular velocity and position of a body is further described by Leonhard Euler's theory of rotation, which states that the rotation of a rigid body may be described by the three Euler angles. Applying Chasles' Theorem and Euler's Rotation Theorem to the motion of a spacecraft in Low Earth Orbit (LEO), the Simulink Attitude Analysis Model reads the values from the RMS hand controllers using a TCP/IP connection and determines the vehicle's final position and velocity given known applied impulses and models the attitude response of the vehicle given known applied torques.

[1] Fall Intern, EV82 Stage Analysis, Marshall Space Flight Center, University of Alabama in Huntsville

II. Rotational Motion

A. Using Matlab

According to Euler's Rotation Theorem, any rotation can by described by three Euler rotation angles. There are several sequences of Euler angles that depend upon the axes of rotation. The Simulink Attitude Response Model uses the most common Euler angle sequence known as the x-convention sequence or more commonly as the Euler 3-1-3 sequence. As seen in Figure 1, the first rotation is by the spin angle (Ø) about the z-axis, the second rotation is performed by the nutation angle (θ) about the former x-axis, and the third and final rotation is by the precession angle (ψ) about the former z-axis.

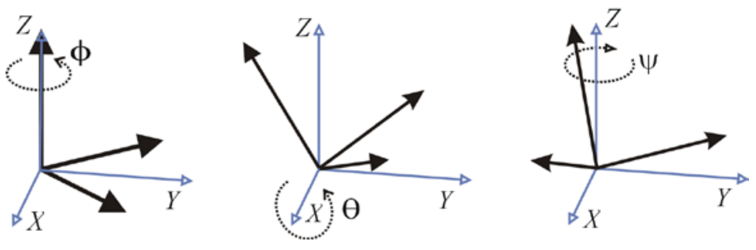
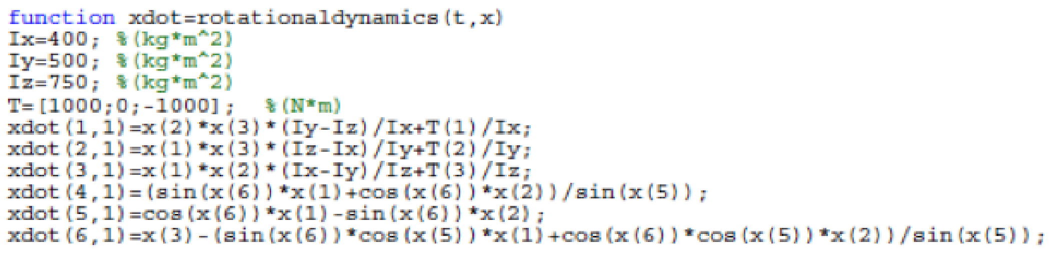


Figure 1: Euler 3-1-3 Sequence

The function developed in Matlab to simulate the attitude response of a spacecraft to known torques is seen to the right in Figure 2. The program rotationaldynamics.m provides the governing differential equations of motion. The first three equations, xdot(1,1), xdot(2,1), and xdot(3,1) calculate the angular velocity of a spacecraft given a known torque profile. The last three equations, xdot(4,1), xdot(5,1), and xdot(6,1) calculate the response of the three Euler angles, Ø, θ, ψ to the same known torque profile. Then using a variable step Runge-Kutta method, Matlab's built in ode45 solver numerically integrates the six governing equations. The results are seen to the right in Figure 3.



```
function xdot=rotationaldynamics(t,x)
Ix=400; %(kg*m^2)
Iy=500; %(kg*m^2)
Iz=750; %(kg*m^2)
T=[1000;0;-1000];   %(N*m)
xdot(1,1)=x(2)*x(3)*(Iy-Iz)/Ix+T(1)/Ix;
xdot(2,1)=x(1)*x(3)*(Iz-Ix)/Iy+T(2)/Iy;
xdot(3,1)=x(1)*x(2)*(Ix-Iy)/Iz+T(3)/Iz;
xdot(4,1)=(sin(x(6))*x(1)+cos(x(6))*x(2))/sin(x(5));
xdot(5,1)=cos(x(6))*x(1)-sin(x(6))*x(2);
xdot(6,1)=x(3)-(sin(x(6))*cos(x(5))*x(1)+cos(x(6))*cos(x(5))*x(2))/sin(x(5));
```

Figure 2: Rotationaldynamics.m

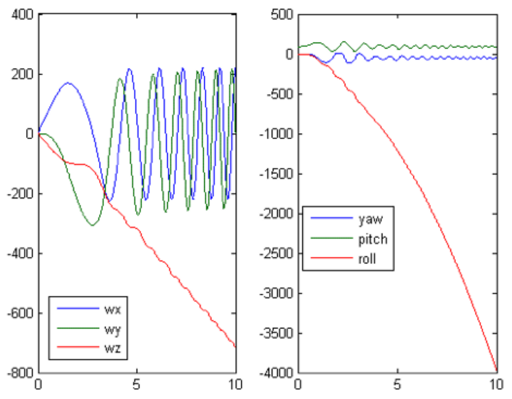
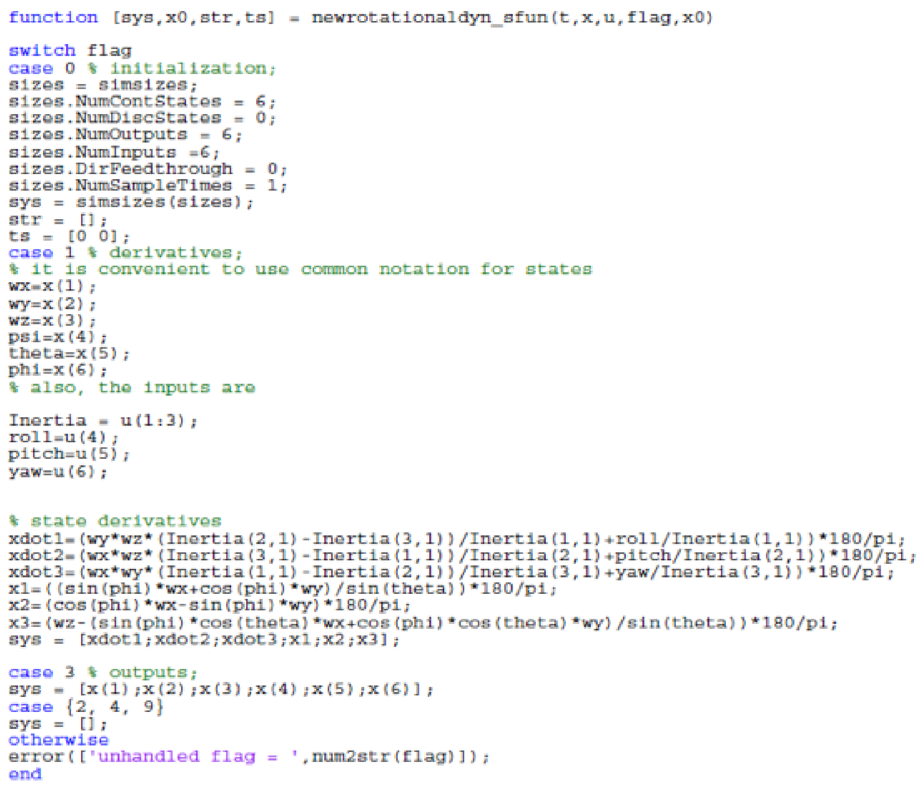


Figure 3: Matlab Results

B. Using Simulink

Developed by Mathworks as an add-on product to Matlab, Simulink is a graphical block-diagramming tool used to simulate dynamical systems. It provides a comprehensive library of block diagrams that the user drags into a workspace to build graphical diagrams as opposed to Matlab's numerical computing and programming environment. In order to integrate the rotational equations developed in Matlab to a graphical model in Simulink, the Attitude Response Model utilizes the S-Function block. S-Functions are useful because of their ability to incorporate existing C code into a Simulink diagram. Similar to rotationaldynamics.m that was built in Matlab, newrotationaldyn_sfun.m, seen in Figure 4, describes the system as set of mathematical equations. Using a set of callback methods, the S-Function block performs tasks at various stages of the simulation. The first step required is the initialization task, which sets the number and dimensions of the inputs and outputs, establishes the sample time, and allocates necessary storage. Then the S-Function calculates the next sample value and the output of each time step. Finally, the S-Function integrates the functions in the same way that Matlab's

ODE solvers do. After writing an S-Function file, the inputs are defined in the Simulink environment and necessary parameters are passed to the file. Using the scope block, the results are displayed as x-y graphs.



```
function [sys,x0,str,ts] = newrotationaldyn_sfun(t,x,u,flag,x0)

switch flag
case 0 % initialization;
sizes = simsizes;
sizes.NumContStates = 6;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 6;
sizes.NumInputs =6;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
str = [];
ts = [0 0];
case 1 % derivatives;
% it is convenient to use common notation for states
wx=x(1);
wy=x(2);
wz=x(3);
psi=x(4);
theta=x(5);
phi=x(6);
% also, the inputs are

Inertia = u(1:3);
roll=u(4);
pitch=u(5);
yaw=u(6);


% state derivatives
xdot1=(wy*wz*(Inertia(2,1)-Inertia(3,1))/Inertia(1,1)+roll/Inertia(1,1))*180/pi;
xdot2=(wx*wz*(Inertia(3,1)-Inertia(1,1))/Inertia(2,1)+pitch/Inertia(2,1))*180/pi;
xdot3=(wx*wy*(Inertia(1,1)-Inertia(2,1))/Inertia(3,1)+yaw/Inertia(3,1))*180/pi;
x1=((sin(phi)*wx+cos(phi)*wy)/sin(theta))*180/pi;
x2=(cos(phi)*wx-sin(phi)*wy)*180/pi;
x3=(wz-(sin(phi)*cos(theta)*wx+cos(phi)*cos(theta)*wy)/sin(theta))*180/pi;
sys = [xdot1;xdot2;xdot3;x1;x2;x3];

case 3 % outputs;
sys = [x(1);x(2);x(3);x(4);x(5);x(6)];
case {2, 4, 9}
sys = [];
otherwise
error(['unhandled flag = ',num2str(flag)]);
end
```

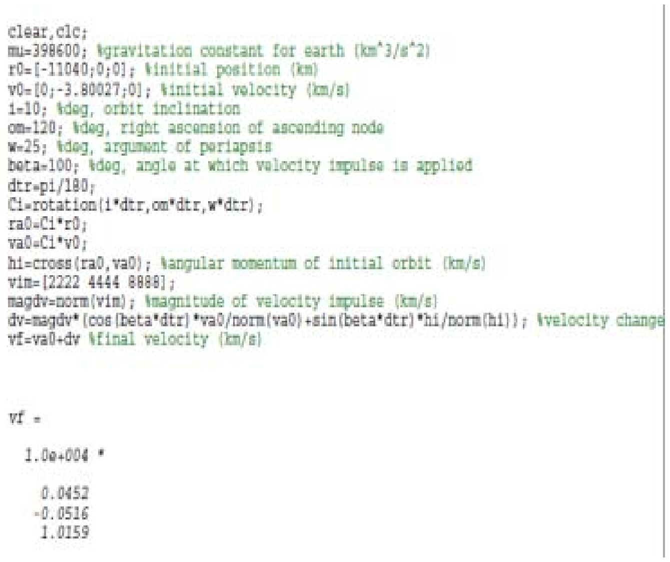
Figure 4: S-Function File

II. Translational Motion

A. Using Matlab

Disregarding the rotational motion of a rigid body and focusing solely on the flight path of the vehicle allows for analysis to be confined to the translational motion of the spacecraft in orbit. When studying translational motion it is important to reduce the vehicle dynamics to the motion of a specific point on the body and study the entire body as a point mass system. The position, velocity, and acceleration of the particle represent the three degrees of freedom of translational motion.

The translational function written in Matlab explores the governing equations that rely on the rotation matrix and four orbital parameters. The first step is to derive the angular momentum vector given the initial position and velocity of the body in motion. After determining the magnitude of the velocity impulse, which depends on the norm of the initial velocity and the orbit inclination, the velocity change can then be determined. The final velocity vector is then calculated by vectorially adding the velocity impulse to the initial velocity. The translation.m file and the rotation.m file are seen below in Figures 5 and 6.



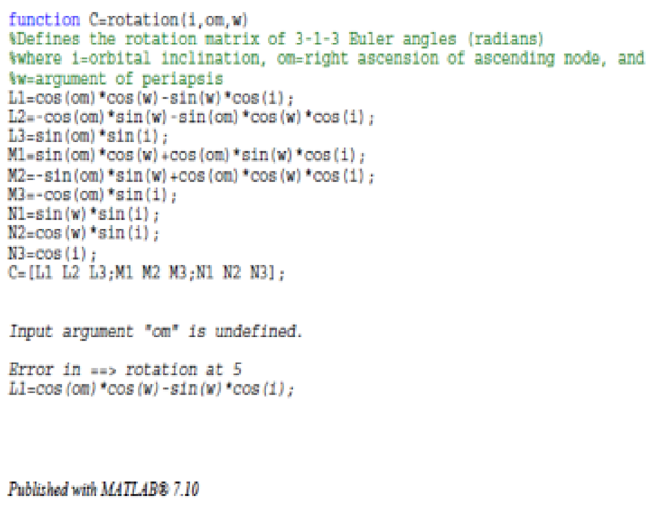
```
clear,clc;
mu=398600; %gravitation constant for earth (km^3/s^2)
r0=[-11040;0;0]; %initial position (km)
v0=[0;-3.80027;0]; %initial velocity (km/s)
i=10; %deg, orbit inclination
om=120; %deg, right ascension of ascending node
w=25; %deg, argument of periapsis
beta=100; %deg, angle at which velocity impulse is applied
dtr=pi/180;
Ci=rotation(i*dtr,om*dtr,w*dtr);
ra0=Ci*r0;
va0=Ci*v0;
hi=cross(ra0,va0); %angular momentum of initial orbit (km/s)
vim=[2222 4444 8888];
magdv=norm(vim); %magnitude of velocity impulse (km/s)
dv=magdv*(cos(beta*dtr)*va0/norm(va0)+sin(beta*dtr)*hi/norm(hi)); %velocity change
vf=va0+dv %final velocity (km/s)


vf =

  1.0e+004 *

    0.0452
   -0.0516
    1.0159
```

Figure 5: Translation.m



```
function C=rotation(i,om,w)
%Defines the rotation matrix of 3-1-3 Euler angles (radians)
%where i=orbital inclination, om=right ascension of ascending node, and
%w=argument of periapsis
L1=cos(om)*cos(w)-sin(w)*cos(i);
L2=-cos(om)*sin(w)-sin(om)*cos(w)*cos(i);
L3=sin(om)*sin(i);
M1=sin(om)*cos(w)+cos(om)*sin(w)*cos(i);
M2=-sin(om)*sin(w)+cos(om)*cos(w)*cos(i);
M3=-cos(om)*sin(i);
N1=sin(w)*sin(i);
N2=cos(w)*sin(i);
N3=cos(i);
C=[L1 L2 L3;M1 M2 M3;N1 N2 N3];


Input argument "om" is undefined.

Error in ==> rotation at 5
L1=cos(om)*cos(w)-sin(w)*cos(i);
```

Published with MATLAB® 7.10

Figure 6: Rotation.m

B. Using Simulink

As seen below in Figure 7, the translational motion modeled in Simulink uses constant blocks to generate inputs and mathematical blocks to modify the inputs. Lines are used to pass the values into the embedded Matlab function block as inputs. The embedded Matlab function allows the user to write a Matlab function that is used in a Simulink model.

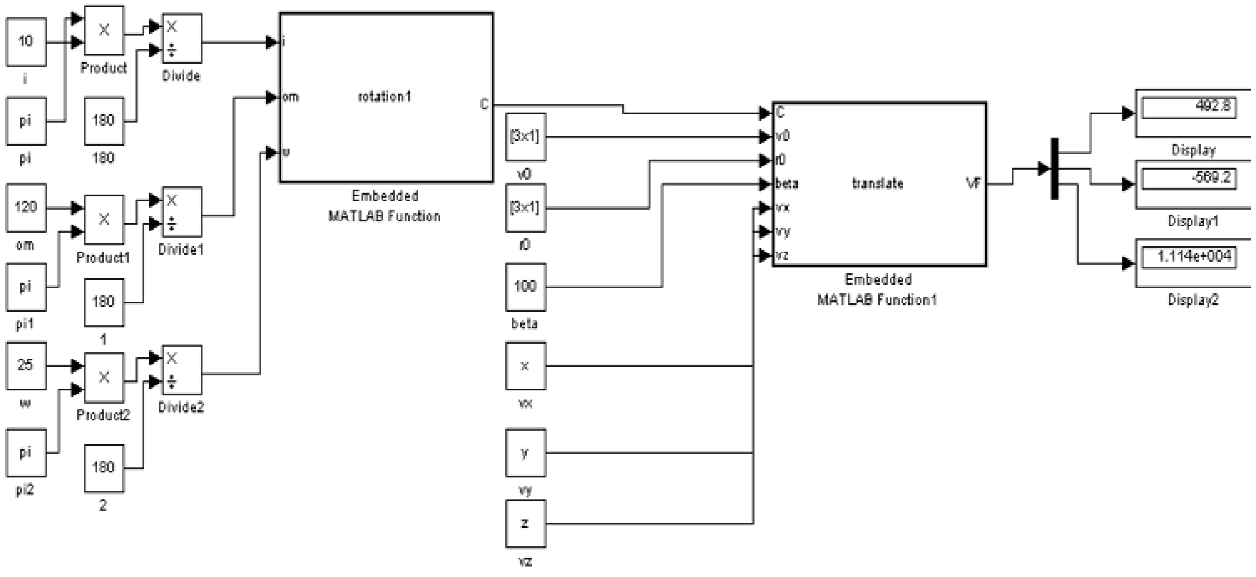
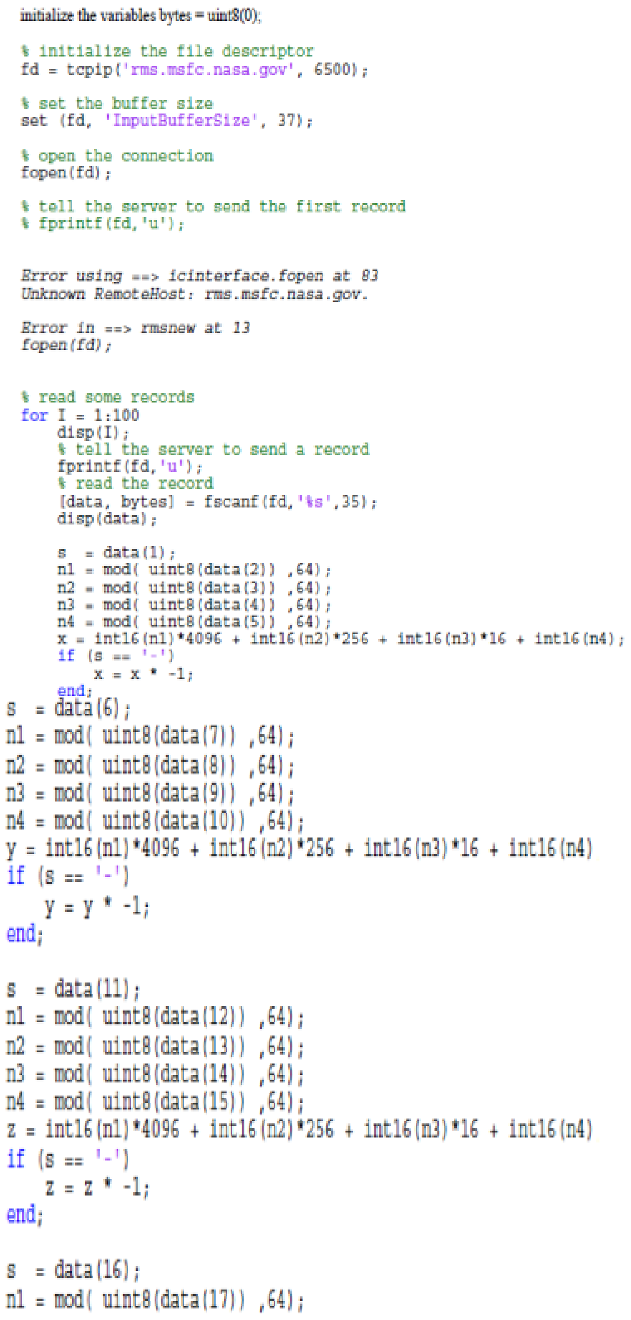


Figure 7: Translational Motion in Simulinnk

III. Connecting Simulink to RMS Hand Controllers

TCP/IP (Transmission Control Protocol/Internet Protocol) is the basic communication language of the Internet. TCP/IP allows for services such as file transfer across a very large system of servers. TCP converts files into data packets that are transmitted over the network connection to the destination computer. The IP simultaneously configures the destination address to verify the data is sent to the right place. In order to open a TCP/IP connection and read the values of the RMS hand controllers, the program seen in Figure 8 was written. It requests a service from a server in the network that reads the request, processes the data, and sends it back to the computer in a format that Matlab can understand. The values are then passed to the Matlab workspace where Simulink reads them as torque and impulse inputs.

.



```
initialize the variables bytes = uint8(0);

% initialize the file descriptor
fd = tcpip('rms.msfc.nasa.gov', 6500);

% set the buffer size
set (fd, 'InputBufferSize', 37);

% open the connection
fopen(fd);

% tell the server to send the first record
% fprintf(fd,'u');


Error using ==> icinterface.fopen at 83
Unknown RemoteHost: rms.msfc.nasa.gov.

Error in ==> rmsnew at 13
fopen(fd);


% read some records
for I = 1:100
    disp(I);
    % tell the server to send a record
    fprintf(fd,'u');
    % read the record
    [data, bytes] = fscanf(fd,'%s',35);
    disp(data);

    s  = data(1);
    n1 = mod( uint8(data(2)) ,64);
    n2 = mod( uint8(data(3)) ,64);
    n3 = mod( uint8(data(4)) ,64);
    n4 = mod( uint8(data(5)) ,64);
    x = int16(n1)*4096 + int16(n2)*256 + int16(n3)*16 + int16(n4);
    if (s == '-')
        x = x * -1;
    end;
s  = data(6);
n1 = mod( uint8(data(7)) ,64);
n2 = mod( uint8(data(8)) ,64);
n3 = mod( uint8(data(9)) ,64);
n4 = mod( uint8(data(10)) ,64);
y = int16(n1)*4096 + int16(n2)*256 + int16(n3)*16 + int16(n4)
if (s == '-')
    y = y * -1;
end;

s  = data(11);
n1 = mod( uint8(data(12)) ,64);
n2 = mod( uint8(data(13)) ,64);
n3 = mod( uint8(data(14)) ,64);
n4 = mod( uint8(data(15)) ,64);
z = int16(n1)*4096 + int16(n2)*256 + int16(n3)*16 + int16(n4)
if (s == '-')
    z = z * -1;
end;

s  = data(16);
n1 = mod( uint8(data(17)) ,64);
```

Figure 8: TCPIP.m

Acknowledgments

Rebecca H. Mauldin thanks Don Krupp, MSFC EV82, for all of his continued support and guidance, Peter Ma, MSFC EV82 for his assistance in all aspects of the project, Fred Flack, MSFC EV82, for export control support, and all of MSFC EV82 for support and assistance throughout entire internship.

References

Tewari, Ashish. *Atmospheric and Space Flight Dynamics*. Boston;Birkhauser. 2007. 369-452.

The MathWorks. The MathWorks, Inc., 1994. Web. June-July 2009. <http://mathworks.com>.