UMN CubeSat Attitude Determination and Control System for EXACT Mission

Attitude Control System

University of Minnesota - Twin Cities
Department of Aerospace Engineering and Mechanics

Contributors:

Evan Majd
Alex Hrovat

1.0 Introduction

A control system shall be developed to achieve 3-axis attitude control of a 3U cube satellite in a sun synchronous and ISS orbit. The control system shall be designed so that the attached grid detector will always point within ±25.8˚ of the sun. In the event of a solar flare, the satellite will capture data and downlink back to a ground station on Earth.

2.0 Requirements and Risks

Specific requirements of the control system include:

1. EXACT shall be 3-axis stabilized
2. ADCS shall keep the sun in the detector's field of view; within ±25.8˚
3. The controller shall reach the desired attitude within 5 orbits
4. The controller shall minimize magnetorquer usage to minimize interference with science sensors.
5. EXACT shall use magnetorquers to actively control its attitude

There are several risks that must be taken into account. These risks include the uncertainties in moment of inertia, sensor noise, and additional forces that are not modeled in simulations, but exist in the real world. The value of the moment of inertia directly affects the plant function. These are values we must know in order to design an optimal controller, but are still subject to change based on the needs of the project as a whole. As such, the controller was designed to be easily modifiable and tunable, but must be updated with each change of mass properties. The sensor noise is taken from specification sheets and may not be exact in accuracy. As a result, the controller must be designed to be robust. Small changes in sensor noise values must not render the controller unstable. Additional forces not modeled include aerodynamic forces, solar radiation pressure, and gravitational effects from celestial bodies. These were considered negligible and were not included in the model. The controller will be robust enough that these small forces will not significantly affect system performance.

3.0 Process to Proposed System

3.1 Magnetorquer Trade Studies

Manufacturer	Model	Magnetic Moment	Power Consumption	Size	Weight	Cost
NewSpace System	CubeSat Magnetorquer Rod	0.2 Am^2	200mW @ 5V	70x9mm	30g	$1600*3
CubeSpace	CubeCoil	0.13 Am^2	300mW @ 5V	90x96x6mm	46g	$1600
CubeSpace	CubeTorquer	0.24 Am^2	200mW @ 2.5V	60x10mm	27.5g	N/A
GOMSpace	Solar Panel	.043 Am^2	73mW @ 3.3V	N/A	N/A	N/A
PCBex	PCB	.38 Am^2	6.8W @ 5V	90x90x5mm	10g	$200

Table 1. Magnetorquer Specifications

The CubeCoil and CubeTorquer combination, provided by CubeSpace, was selected due to the low power consumption it provides, while still generating a reasonable magnetic moment. This combination provides 3-axis control.

3.2 Testing an Approximate Transfer Function

The CubeSat will have active attitude control by activating magnetorquers using a PD controller. The interaction between the magnetorquers and the Earth's magnetic field is what provides the controlling torque and can be represented using the following equation

$$\tau = \mu \times B \tag{1}$$

where τ is the torque applied, μ is the magnetic moment generated by the magnetorquers, and B is the Earth's local magnetic field. The equation that relates torque, τ, moment of inertia, I, and angular acceleration, α, is used as an approximate model for this situation. This yields the equation

$$\tau = I\alpha = I\ddot{\theta}, \tag{2}$$

where θ is the angle of rotation about an arbitrary axis in radians. The input is torque in N*m^2 and the output is angle of rotation in radians. This gives a transfer function of,

$$G(s) = \frac{input}{output} = \frac{1}{s^2}. \tag{3}$$

The Simulink model is located in Appendix A labelled model 1.Testing this in MATLAB at 2.5 Hz gives approximately an 8 second settling time with around a 15% overshoot. The moment of inertia, proportional gain, and damping gain are all assumed to be 1 for simplicity. The reference command was 10 degrees and there is a little steady state error.

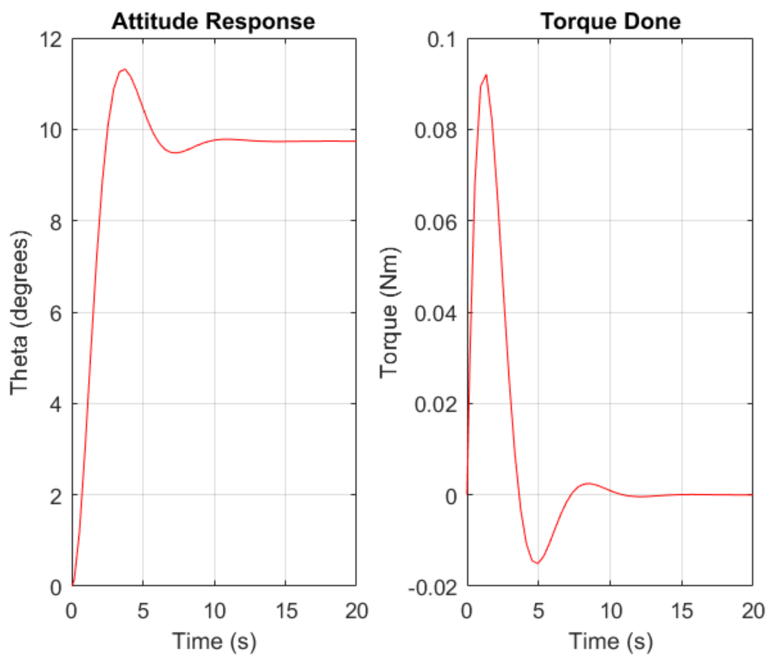


Figure 1. Pictured on the left is the response to a command for the CubeSat to change its attitude by 10 degrees about a single axis. There is a 16% overshoot leading to a max value of 11.35˚. The steady state value is 9.737˚, which is 2.63% off from the desired value. The settling time is 8.038 seconds. On the right is the commanded torque which has a maximum torque of .09202 N*m^2. There is no limit included on torque. In effect, the control system is very fast. This limit will be added in the subsequent algorithms.

3.3 Deciding on an Appropriate Controller and Finding the Correct Plant Function

Due to feasibility, ease of implementation, and current coursework at the University of Minnesota on PID controllers, the proportional and derivative controller was chosen. Integral control was left out since the controller does not need to be exact and will be turned off for extended periods of time. As long as the steady state attitude is close to the desired value, the Sun will be in the detector's field of view. The derivative term, or angular velocity will be fed into the controller via onboard sensors. This will yield good noise rejection, minor steady state error, small overshoot, and faster settling times.

The correct plant function will then be the combination of equation 2 and

$$\tau = input = K_p(\theta_{des} - \theta_{act}) + K_d\left(\dot{\theta}_{des} - \dot{\theta}_{act}\right), \tag{4}$$

which produces the equation,

$$\ddot{\theta} = \frac{1}{I}\left(K_p(\theta_{des} - \theta_{act}) + K_d\left(\dot{\theta}_{des} - \dot{\theta}_{act}\right)\right). \tag{5}$$

Expanding this equation and neglecting the term $\dot{\theta}_{des}$ yields,

$$\ddot{\theta} + \frac{K_d}{I}\dot{\theta}_{act} + \frac{K_p}{I}\theta_{act} = \frac{K_P}{I}\theta_{des}. \tag{6}$$

This equation can be compared to the Second Order Closed Loop Equation for a PD Controller,

$$\ddot{\theta} + (a_1 + b_0K_d)\dot{\theta}_{act} + \left(a_0 + b_0K_p\right)\theta_{act} = \left(b_0K_p\right)\theta_{des}, \tag{7}$$

where $a_1 = a_0 = 0$ and $b_0 = 1/I$.

3.4 One-Dimensional Algorithm

Using the correct transfer function derived in the previous section, a one-dimensional algorithm and model can be created. To find the gains K_p and K_d, a method called PID tuning is employed. First the equation for the plant function needs to be modified to,

$$\ddot{\theta} + (2\zeta\omega_n)\dot{\theta}_{act} + (\omega_n^2)\theta_{act} = (b_0 K_p)\theta_{des}. \tag{8}$$

Using this knowledge, one can solve for the gains of the controller given the damping ratio, ζ, the natural frequency, ω_n, and the moment of inertia, I. This is shown with the proportional gain equation,

$$K_p = \omega_n^2 I, \tag{9}$$

and the damping gain equation,

$$K_d = 2\zeta\omega_n I. \tag{10}$$

A damping ratio of 2, and a natural frequency of .05 seconds is chosen which respectively produces values for K_p and K_d of 1.36e-04 and .0109. The moment of inertia is I=.05440 kg*m^2. The Simulink model is in Appendix A labelled model 2. The input torque is limited since the magnetorquer cannot instantaneously generate large amounts of torque. This is shown by the function $\tau = \mu \times B$, where μ is the maximum dipole magnetic moment of the magnetorquer (.24 Nm/T) and B is the magnetic field of the earth in tesla. Instead of using a constant input, a ramp input is included to smooth the steady state of the response. This is due to a smooth increase in angular velocity.

The Simulink model is run at the sun synchronous orbit and the magnitude of the earth's magnetic field (MATLAB function igrfmagm()) is shown below. It should be noted that the magnetic field will be accurately calculated from attitude determination and fed to the controller. These magnetic field values are just used to test the performance of the controller. The simulation is run at 10 Hz.

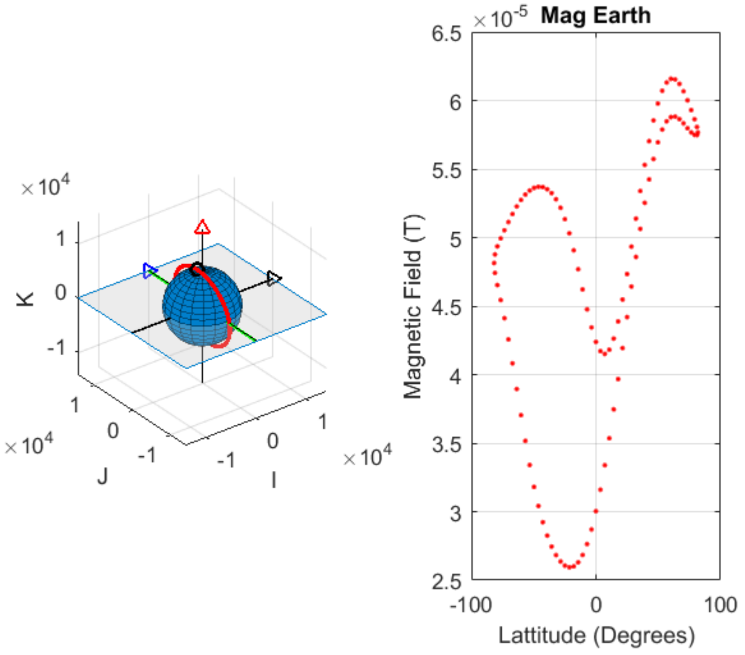


Figure 3. Pictured on the left is Earth with the Sun-synchronous orbit represented by the red line circling the sphere. Pictured on the right is the value of the Earth's magnetic field in the +x direction of Earth Centered Inertial Coordinate frame sampled at various points around the orbit.

The attitude response is shown below to be stable with a rise time of 299.8723 seconds, settling time of 415.0315 seconds, no overshoot, a peak value of 9.7383˚, and a steady state of 9.7383˚. The altitude is 705 km and moment of inertia is .05440 kg*m^2.

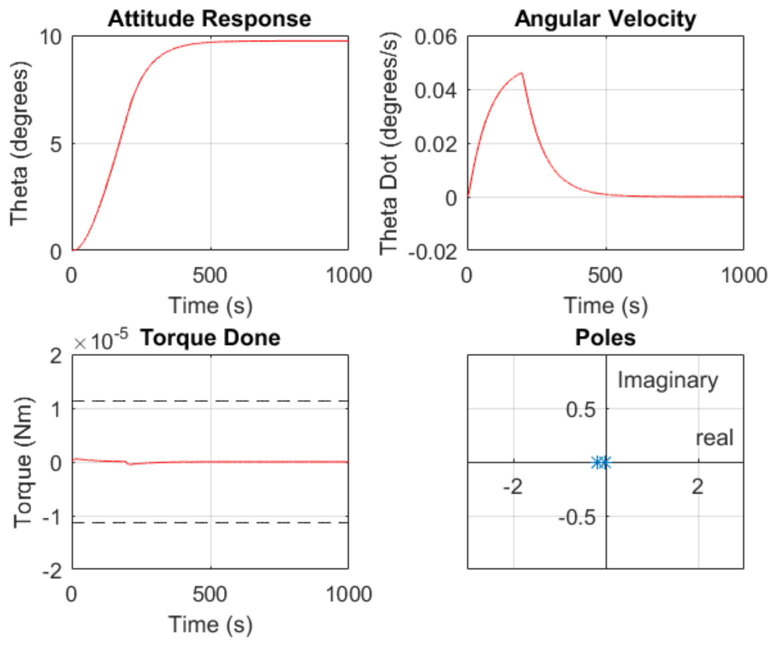


Figure 4. The system output of a 10-degree attitude change command about one axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 299.8723 seconds and a steady state value of 9.7383 ˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .04577 ˚/s. We want to limit angular velocity to .15 ˚/s to prevent the satellite from rotating too fast, in effect taking a long time to slow down. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 1.1301e-05 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.1866 and -.0134. The negative poles ensure stability.

3.5 Three-Dimensional Algorithm

Using the one-dimensional algorithm from the previous section, it can be expanded into three dimensions. The difference is the controllers gains in each direction affect response times. These values will also change due to differing moment of inertias. $\mu_x, \mu_y\ and\ \mu_z$ are respectively .24, .24, and .13 Nm/T.

The torque limit from rotations about the z-axis, or from the x-axis to the y-axis is,

$$\tau_x = \mu_x \times B_y = \mu_x B_y. \tag{11}$$

The torque limit from rotations about the x-axis, or from the y-axis to the z-axis is,

$$\tau_y = \mu_y \times B_z = \mu_x B_y. \tag{12}$$

The torque limit from rotations about the y-axis, or from the z-axis to the x-axis is,

$$\tau_z = \mu_z \times B_x = \mu_x B_y. \tag{13}$$

These are not fixed values since the Earth's magnetic field changes as the satellite goes through orbit.

The damping ratio in the x, y, and z directions are respectively 5, 4, and 2. The natural frequencies in the x, y, and z directions are all .05 seconds. This produces proportional gain values for K_{px}, K_{py}, and K_{pz} of 1.36e-04, 3.6025e-05, and 1.2820e-04. This produces damping gain values for K_{dx}, K_{dy}, and K_{dz} of .0272, .0058, and 1.2820e-04.

The Simulink model is in Appendix A labelled model 3. The simulation is run at 10 Hz. The altitude is 705 km and moment of inertia values are I_{xx}=.05440, I_{yy}=.01441, and I_{zz} kg*m^2.

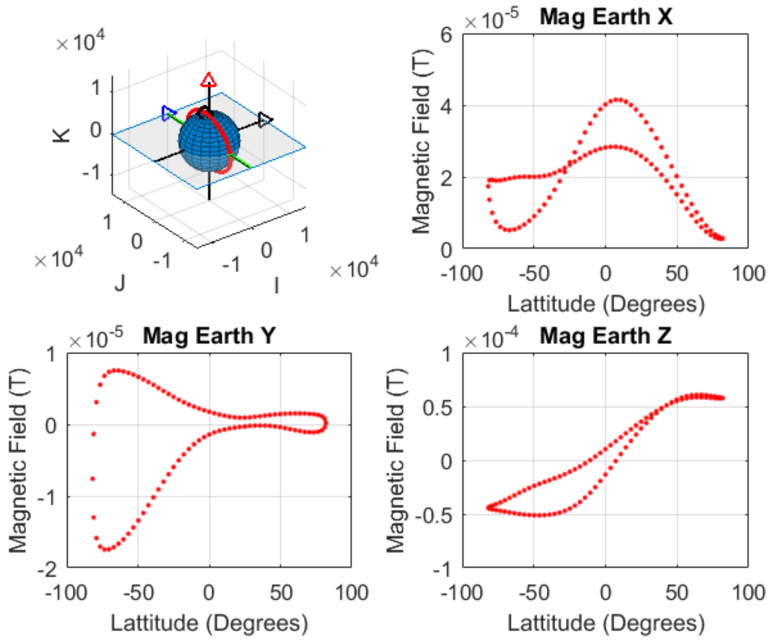


Figure 6. Picture above are the values of the Earth's magnetic field in a sun-synchronous orbit in the earth centered earth fixed frame. Found using the code taken from Classical Orbital Elements GUI and compiled into our own function mag_earth.m[1]. It should be noted that the magnetic field will be accurately calculated from attitude determination and fed to the controller. These magnetic field values are just used to test the performance of the controller. The simulation is run at 10 Hz.

The attitude response shown below about the z-axis is stable with a rise of 632.6488 seconds, settling time of 881.1551 seconds, no overshoot, peak value of 9.7377˚, and a steady state value of 9.7377˚. The altitude is 705 km.

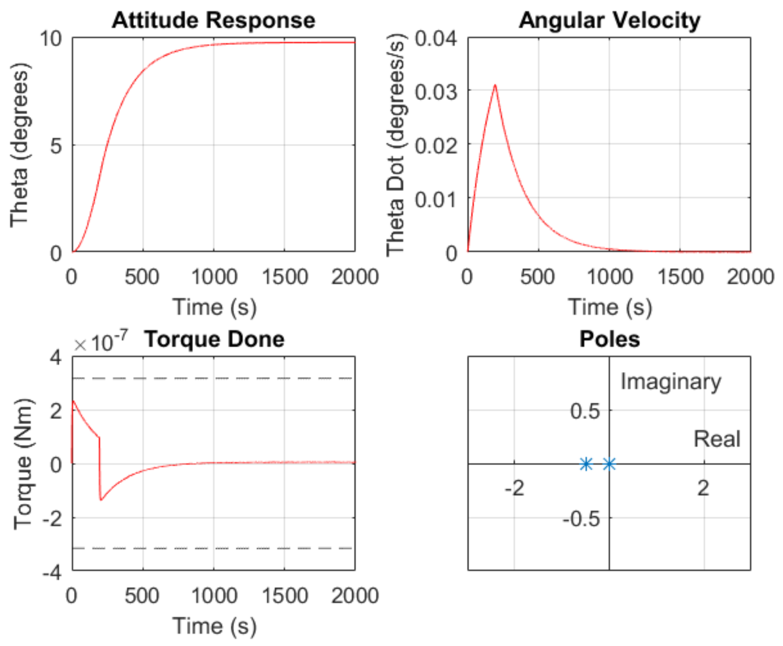


Figure 7. The system output of a 10-degree attitude change command about the z axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 632.6488 seconds and a steady state value of 9.7377˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .03116 ˚/s. We want to limit angular velocity to .15 ˚/s to prevent the satellite from rotating too fast, in effect taking a long time to slow down. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 3.1618e-07 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.4949 and -.0051. The negative poles ensure stability.

The attitude response shown below about the x-axis is stable, with a rise time of 520.1551 seconds, settling time of 725.8628 seconds, no overshoot, peak value of 9.7331˚, and a steady state value of 9.7331˚.

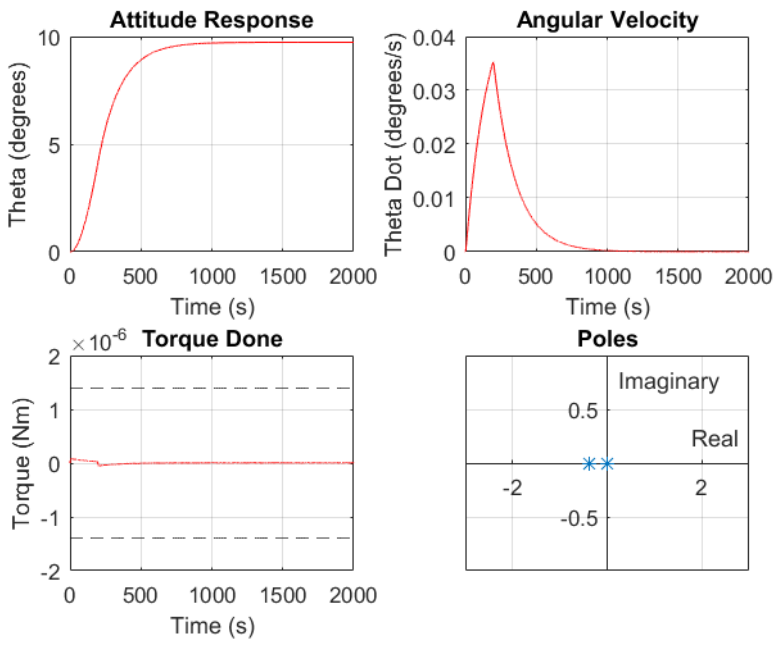


Figure 8. The system output of a 10-degree attitude change command about the x axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 520.1551 seconds and a steady state value of 9.7331˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .03518 ˚/s. We want to limit angular velocity to .15 ˚/s to prevent the satellite from rotating too fast, in effect taking a long time to slow down. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 1.3930e-06 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.3936 and -.0064. The negative poles ensure stability.

The attitude response shown below about the y-axis is stable, with a rise time of 299.8888 seconds, settling time of 415.0782 seconds, no overshoot, peak value of 9.7383˚, and a steady state value of 9.7383˚.

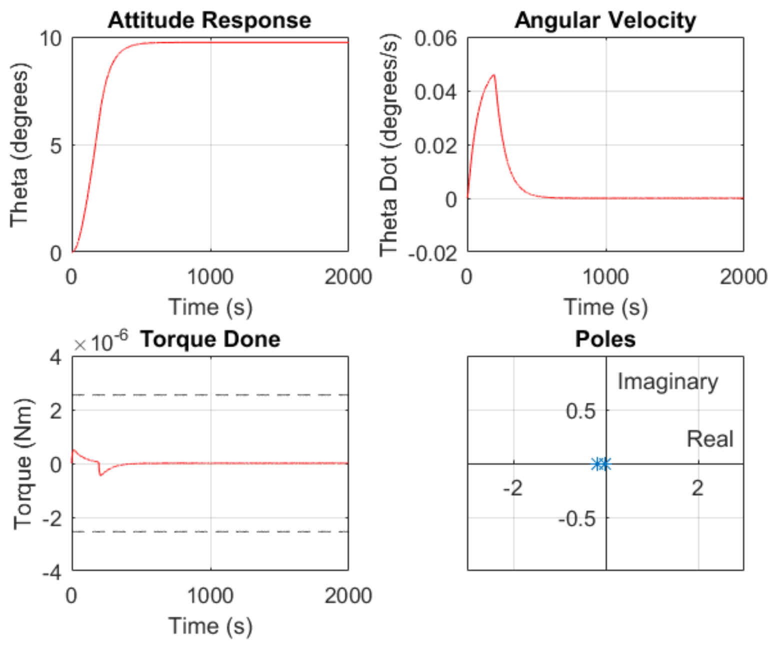


Figure 9. The system output of a 10-degree attitude change command about the x axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 299.8888 seconds and a steady state value of 9.7383˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .04573 ˚/s. We want to limit angular velocity to .15 ˚/s to prevent the satellite from rotating too fast, in effect taking a long time to slow down. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 2.5411e-06 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.1866 and -.0134. The negative poles ensure stability.

Overall, the responses are shown to be fairly quick (couple of minutes), but can be optimized in regards to run time and chosen damping ratios and natural frequency values.

3.5 Detumbling

When the CubeSat is released from the launcher it is often tumbling and needs to be stopped before the main controller can take over. A separate controller must be made to damp the angular velocities to zero. Using the same transfer functions as above, a b-dot controller was developed. The b-dot control law takes advantage of the fact that the derivative of the magnetic field vector $\dot{b}$ is both perpendicular to b and proportional to ω. Using this, the magnetic dipole m can be expressed as:

$$m = -k\dot{b} \tag{14}$$

where k is a positive gain. A bang-bang solution was chosen to give the fastest spin rate decay. The solution was of the form:

$$m = -m_{max} sign(\dot{b}) \tag{15}$$

where m_{max} is the maximum magnetorquer dipole possible. A Simulink model was created and used to run a simulation. The Simulink model is in Appendix A labelled model 4.

4.0 The Proposed System

4.1 Guidance Algorithm

Using Simulink, a guidance algorithm was developed. The algorithm integrates most of the required code and models into one place and ensures that they can run together seamlessly. The model can be found in Appendix A. Upon initialization, algorithm first checks that we can begin active control. A requirement for all CubeSats is that no attitude maneuvers are conducted for 45 minutes after exiting the launch vehicle. If this check is passed, we next see if we are in "low power mode." In low power mode, the magnetorquers must be run as little as possible. This means we do not activate the control system until the sun is within the payload's field of view, rather than activating the magnetorquers until we are at the optimal attitude. Once in the sun's field of view, we turn off the magnetorquers and let the CubeSat drift until an attitude correction is needed again. With the guidance algorithm, we are more likely to spend time with the sun out of the GRID's field of view, but we will be using less power and should be able to recuperate enough battery to continue with normal operations after a short period of time.

The controller next checks that we are not tumbling by reading in the gyroscope data from the attitude determination system. If the rotation rates are above a certain threshold, then the detumbling algorithm, which is detailed in section 4.3, is executed. If the rotation rates are within acceptable limits, the guidance algorithm checks that we are within an acceptable range of the optimal attitude. If we are not, the system engages the active control system, which is the matrix three-dimensional algorithm detailed in the next section. Once steady state is reached, or we are already at an acceptable attitude, the controller turns off and we enter "rest mode," where the magnetic moments of the magnetorquers are commanded to zero, and the guidance algorithm continues running until one of the checks is not passed.

No limits in the guidance algorithm are currently set, as many of them depend upon the final accuracy of the attitude determination system as well as the system-level decisions that have not been made yet.

4.2 Matrix Three-Dimensional Algorithm

Using the three-dimensional algorithm from the previous section, it can be condensed into matrix form. This will reduce run time, and make the code and model much easier to read. The Simulink model is in Appendix A labelled model 5. The disturbance function *d*, and noise function will not be in the final model. Attitude determination will feed the angular velocity, magnetic field of the earth to produce a more accurate control system.

The damping ratio in the x, y, and z directions are all 1. This will produce fastest settling times, which is the current goal. The natural frequencies in the x, y, and z directions are .01, .01, and .07 seconds. This produces proportional gain values for K_{px}, K_{py}, and K_{pz} of 1.0924e-05, 1.0930e-05, and 1.7432e-05. This produces damping gain values for K_{dx}, K_{dy}, and K_{dz} of .0022, .0022, and 4.9805e-04.

The simulation is run at 10 Hz. The altitude is 705 km and moment of inertia is and moment of inertia values are I_{xx}=.1092, I_{yy}=.1093, and I_{zz}=.0036 kg*m^2.

The attitude response is shown below about the x-axis to be stable, with a rise time of 434.7968 seconds, settling time of 594.5861 seconds, no overshoot, peak value of 9.7337˚, and a steady state value of 9.6975˚.

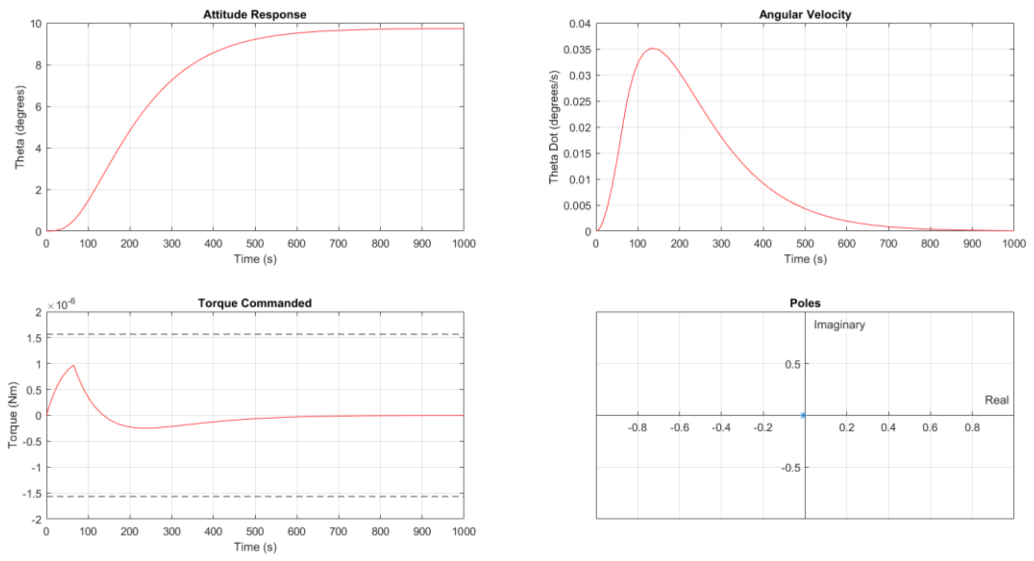


Figure 11. The system output of a 10-degree attitude change command about the x axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 434.7968 seconds and a steady state value of 9.6975˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .03518 ˚/s. We want to limit angular velocity to .15 ˚/s to prevent the satellite from rotating too fast, in effect taking a long time to slow down. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 1.564e-06 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.01 and -.01. The negative poles ensure stability.

The attitude response is shown below about the y-axis to be stable, with a rise time of 434.7968 seconds, settling time of 594.5861 seconds, no overshoot, peak value of 9.7337˚, and steady state of 9.6975˚.

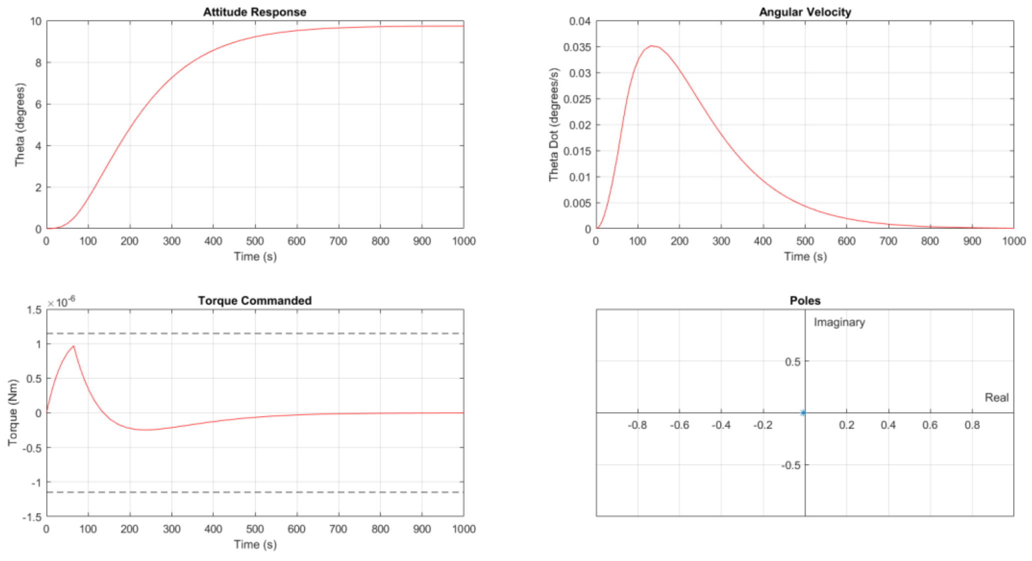


Figure 12. The system output of a 10-degree attitude change command about the y axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 434.7968 seconds and a steady state value of 9.6975˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .03518 ˚/s. We want to limit angular velocity to .15 ˚/s to prevent the satellite from rotating too fast, in effect taking a long time to slow down. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 1.148e-06 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.01 and -.01. The negative poles ensure stability.

The attitude response is shown below about the z-axis to be stable, with a rise time of 90.7348 seconds, settling time of 126.7356 seconds, no overshoot, peak value of 9.7403˚, and steady state value of 9.7403˚.

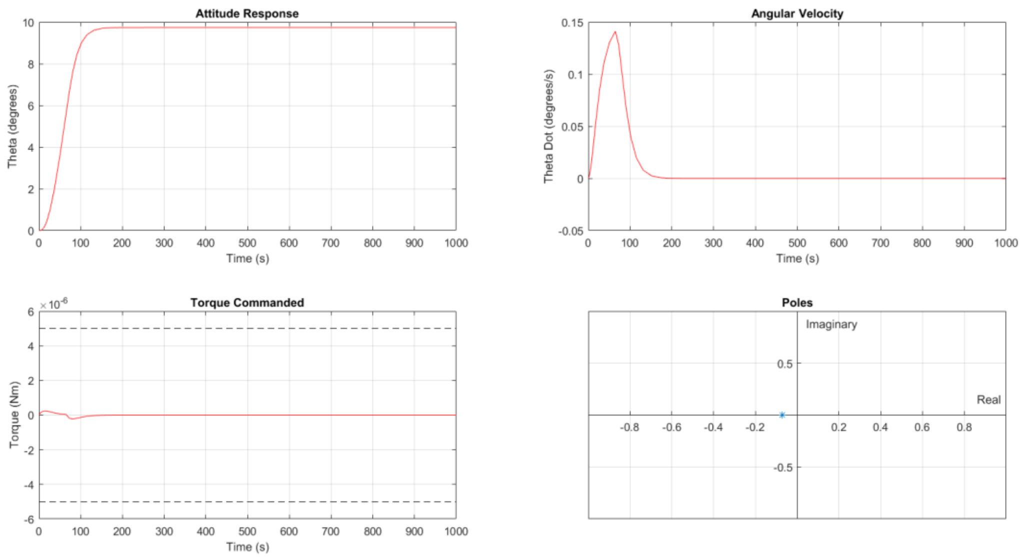


Figure 13. The system output of a 10-degree attitude change command about the z axis is pictured above. The top left graph shows angular position as the simulation runs, with a rise time of 90.7348 seconds and a steady state value of 9.7403˚. The top right graph shows the angular velocity of the CubeSat, with a maximum value of .1412 ˚/s. Angular velocity was limited to .15 ˚/s to prevent the satellite from rotating too fast. Increasing the angular velocity limit would cause the satellite to adjust its attitude quicker, but at the cost of higher power consumption. A lower value was chosen because the mission is not overly time-sensitive, but is fairly power sensitive. The bottom left graph shows the commanded torques. The max torque available is the black line with the magnitude of 5.007e-06 N*m^2. The bottom right graph shows the poles of the controller, which have values of -.07 and -.07. The negative poles ensure stability.

4.3 Detumbling Algorithm

The Simulink model for the detumbling algorithm can be found in Appendix A, and the results of running a simulation using the algorithm can be seen below.

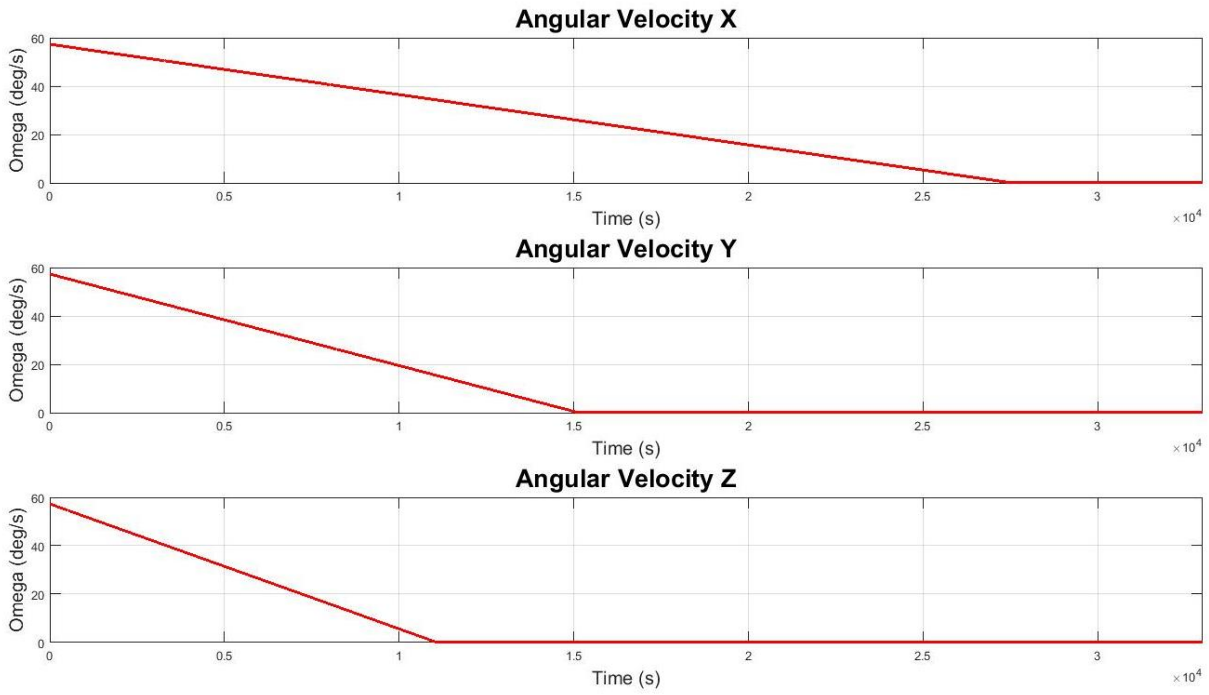


Figure 14. The above plots show the angular velocity about each axis with respect to time. An initial rotational rate of 1 rad/s about each axis was given to the controller. Assuming that the magnetic field was constant, settling times of about 27,500 seconds (5 orbits), 15,100 seconds (2.7 orbits), and 11,100 seconds (2 orbits) for rotation about the x, y, and z axes were returned, respectively.

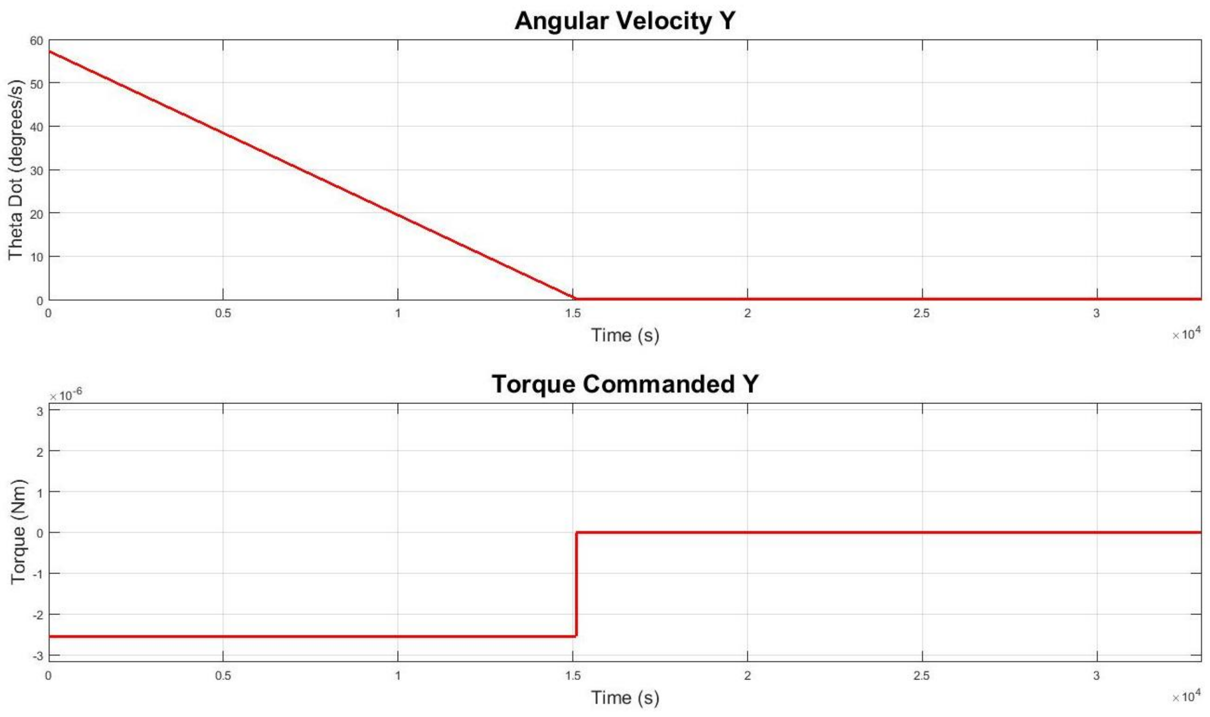


Figure 15. The figure above juxtaposes the graphs of Angular Velocity vs Time and Torque Commanded vs Time for the y-axis. From this, it can be seen that once the satellite's rotation about an axis reaches a certain threshold, currently .15 degrees/second, the controller will deactivate and no torque will be commanded about that axis. The controller does this until all three axes are within the threshold.

5.0 Unfinished Work

There are further revisions that need to be made to complete the proposed system. Currently the guidance algorithm reads in a torque from the controller and calculates the magnetic moment of the magnetorquers and outputs that value. The magnetic moment should be calculated inside of the controllers, and then passed through the guidance algorithm to the magnetorquers.

Another revision is to make the controller compatible with the attitude determination code so the inputs can seamlessly transfer over to the controls code. This has been accommodated through the guidance algorithm, but a full, system-level test with STK was never conducted. This is necessary to ensure that the system as a whole will function as expected, as well as for the final implementation into the CubeSat.

The final revision required is to decide upon and incorporate limits into the guidance algorithm. Many decisions have not been finalized yet, and so any limits currently set would have to be changed in the future.

6.0 System Simulation

To verify the control system will be stable under unknown mass properties and orbital parameters, a Monte Carlo simulation was run. Monte Carlo simulations are essentially the same scenario ran many times with a random draw of initial variables. In the case of our control testing, the system mass, moments of inertia, and orbit altitude are the randomly drawn variables. Using new conditions each time, the control Simulink model was run 10,000 times for each simulation, yielding the probability distribution of system step response in terms of rise time, settling time, overshoot, peak value, and steady-state error. The distribution on inputs was determined by current estimates of mass, moments of inertia, and orbital altitude. The altitude dispersion (Figure 14) is centered around 550 km, with a standard deviation chosen to reflect the insertion accuracy of the Atlas V launch vehicle. The current mass of the satellite was used to generate a clipped Gaussian distribution for mass (Figure 15). The standard deviation on mass was chosen to be large to reflect the uncertainty in this parameter. The current moments of inertia were used to generate a clipped Gaussian distribution, with large standard deviations to reflect the unknown nature of the parameter (Figure 16).

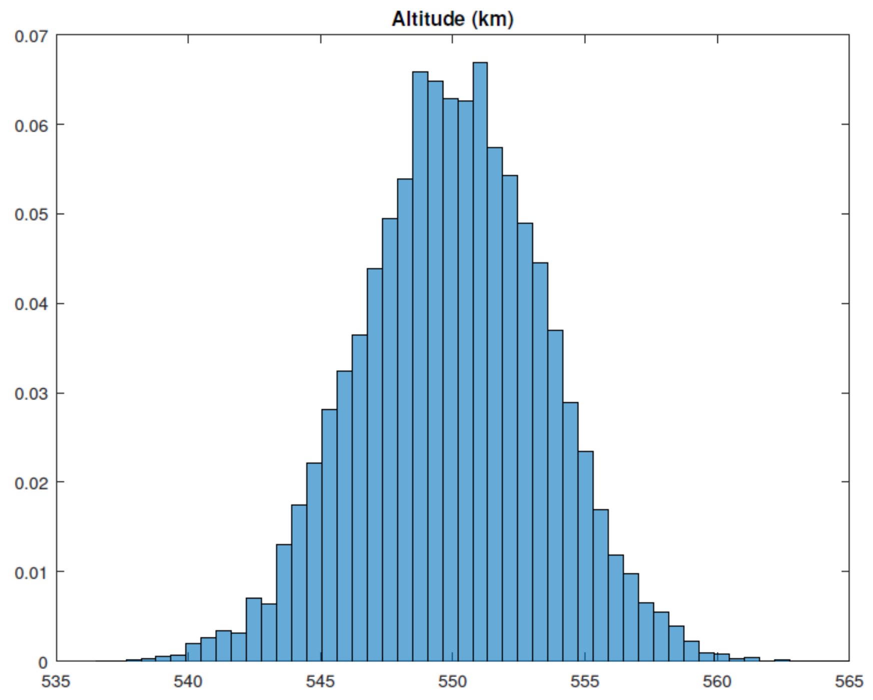


Figure 16. The dispersion on altitude is Gaussian, with a mean value of 550 km. The standard deviation of the altitude injection was taken from the Atlas V launch vehicle's user's guide. The x-axis is in units of kilometers, and the y-axis is normalized by probability.

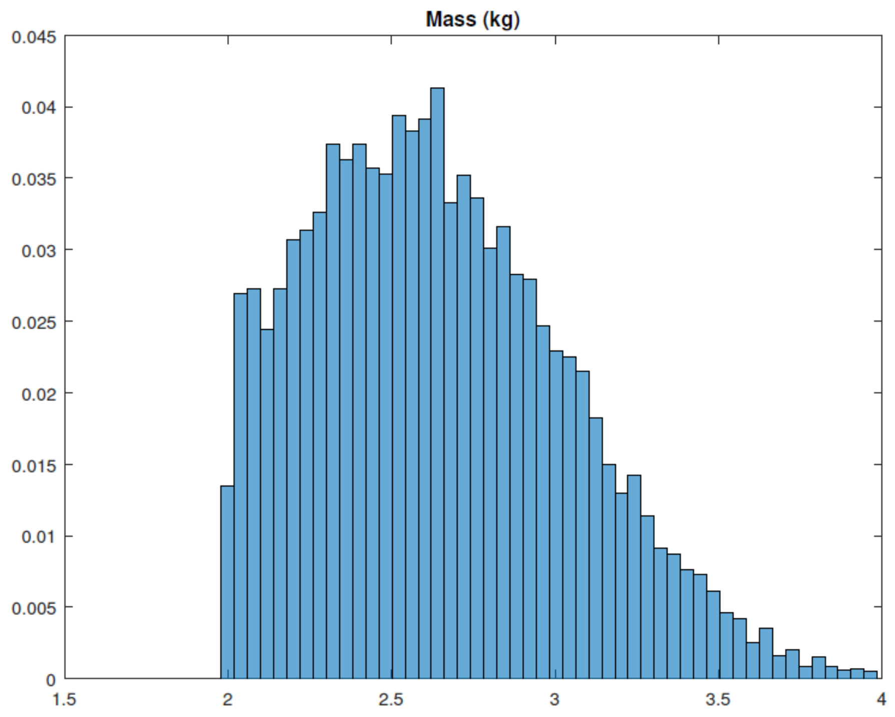


Figure 17. The dispersion on mass is a clipped Gaussian with lower bound set by the current design weight of the spacecraft (2.0 kg) and the upper bound set by the weight limit (4.0 kg). The standard deviation was selected to be large to represent the uncertainty in mass. The x-axis is in units of kilograms, and the y-axis is normalized by probability.

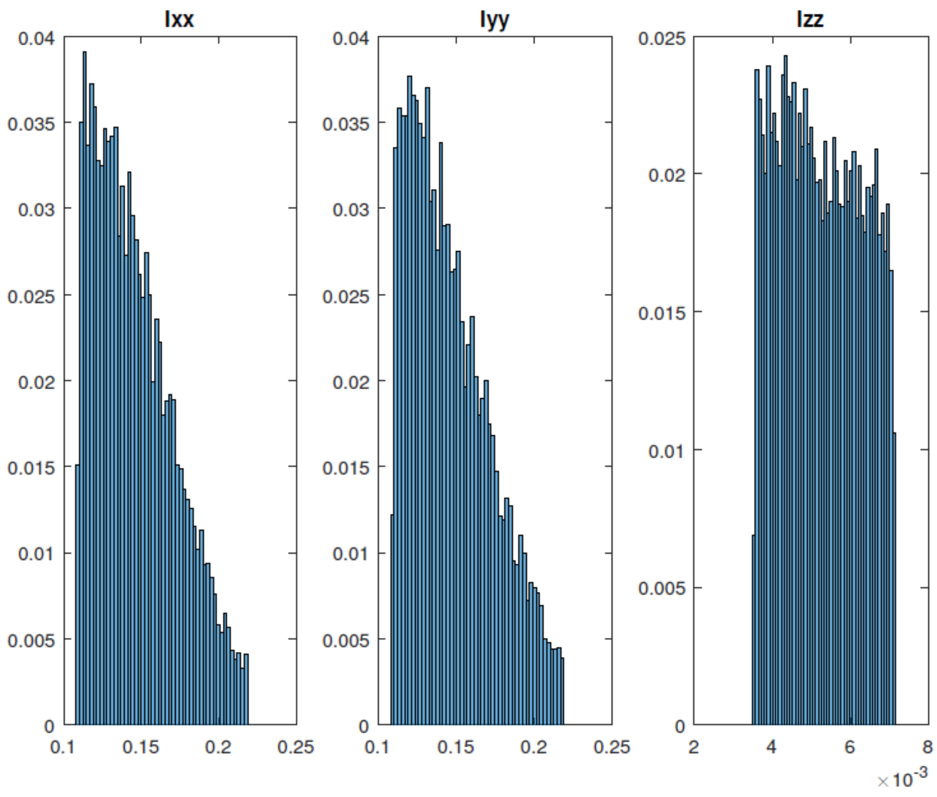


Figure 18. The dispersion on moments of inertia is a clipped Gaussian with a lower bound set by the current moments of the vehicle, and the upper limit set to twice as large to allow for increases in moments. The x-axis is in units of kilogram-meters, and the y-axis is normalized by probability.

From these three inputs (altitude, mass, moment of inertia), the Simulink model was repeatedly run and the results recorded 10,000 times. The output shows that the system is stable under all cases tested, and that the settling time (Figure 17), rise time (Figure 18), overshoot (Figure 19), peak value (Figure 20), and steady state error (Figure 21) are all within acceptable ranges.

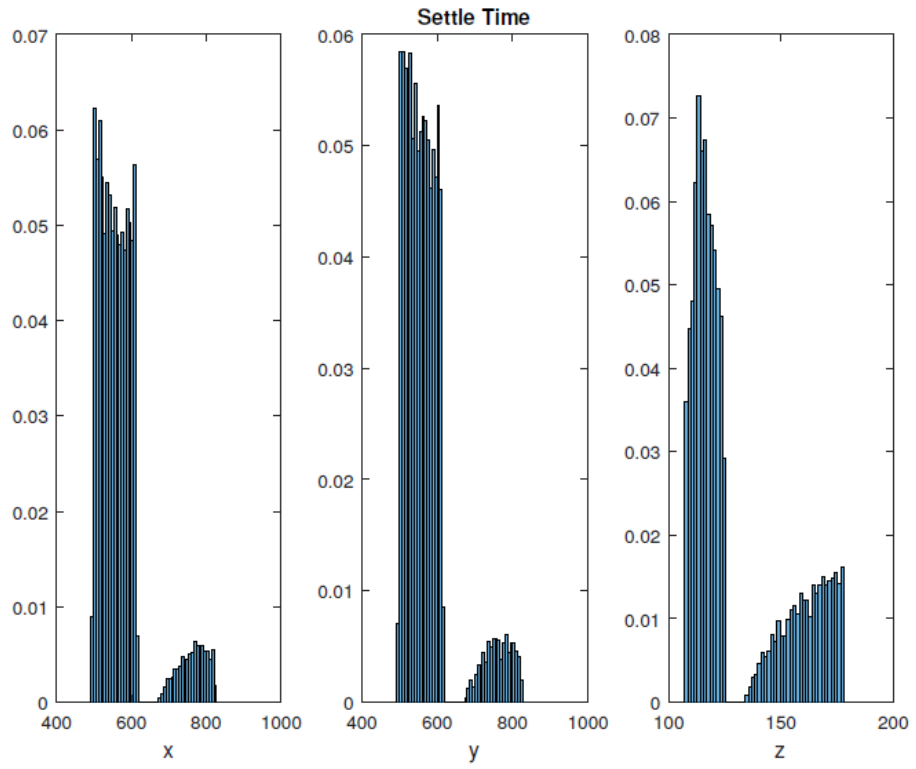


Figure 19. Resulting distribution of step response settling time. The normalized probability is plotted against settling time in seconds. From this distribution, it is shown that under a variety of satellite properties the system is able to settle within 12 minutes, much less than an orbit cycle.

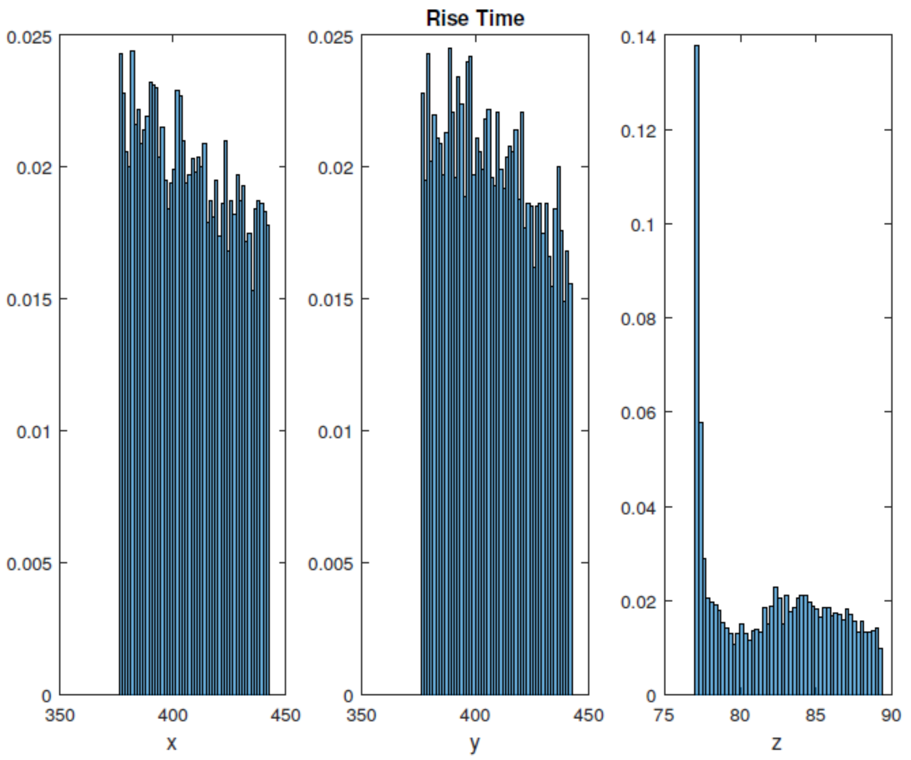


Figure 20. Resulting distribution of step response rise time. The normalized probability is plotted against rise time in seconds. It is shown that under all test cases the rise time is under 6 minutes, with the z-axis taking the longest to rise.

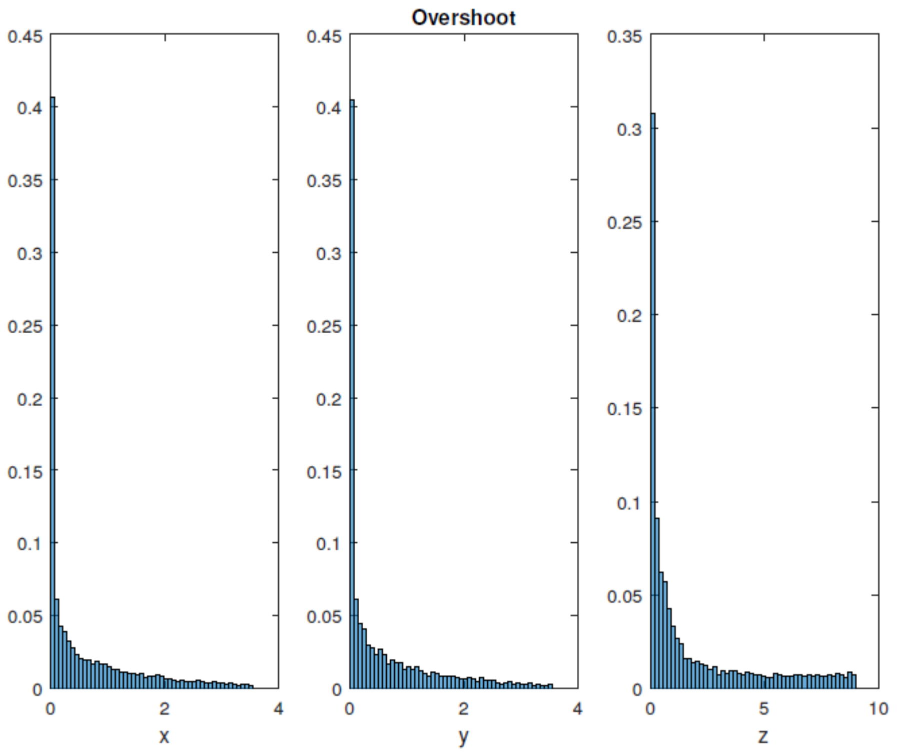


Figure 21. Distribution of step response overshoot. The normalized probability is plotted against overshoot in percent. Both x- and y-axes have large overshoot values, to help reduce the response time of the system.

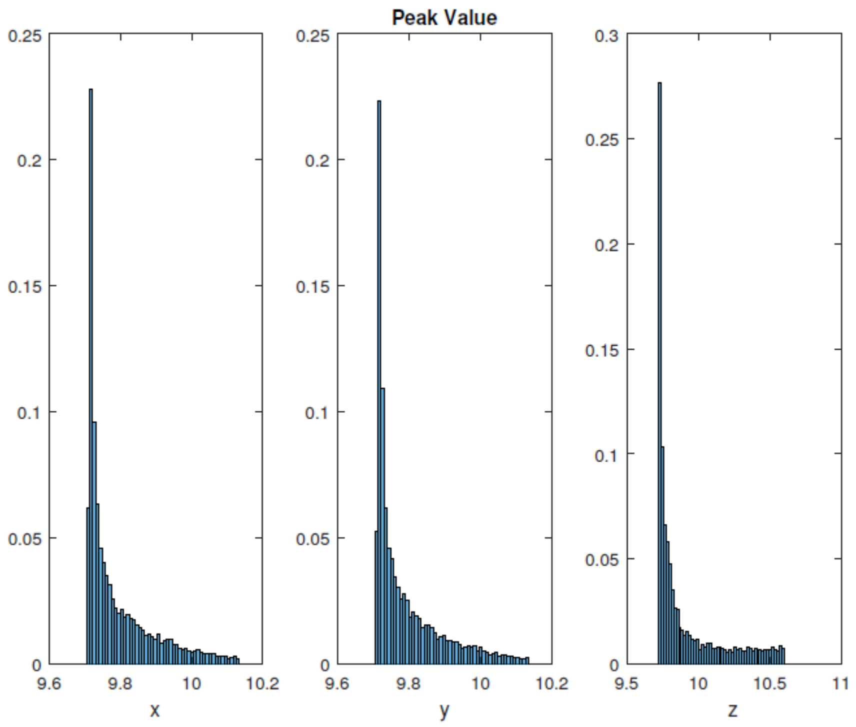


Figure 22. Distribution of peak response value. The normalized probability is plotted against the angle in degrees.

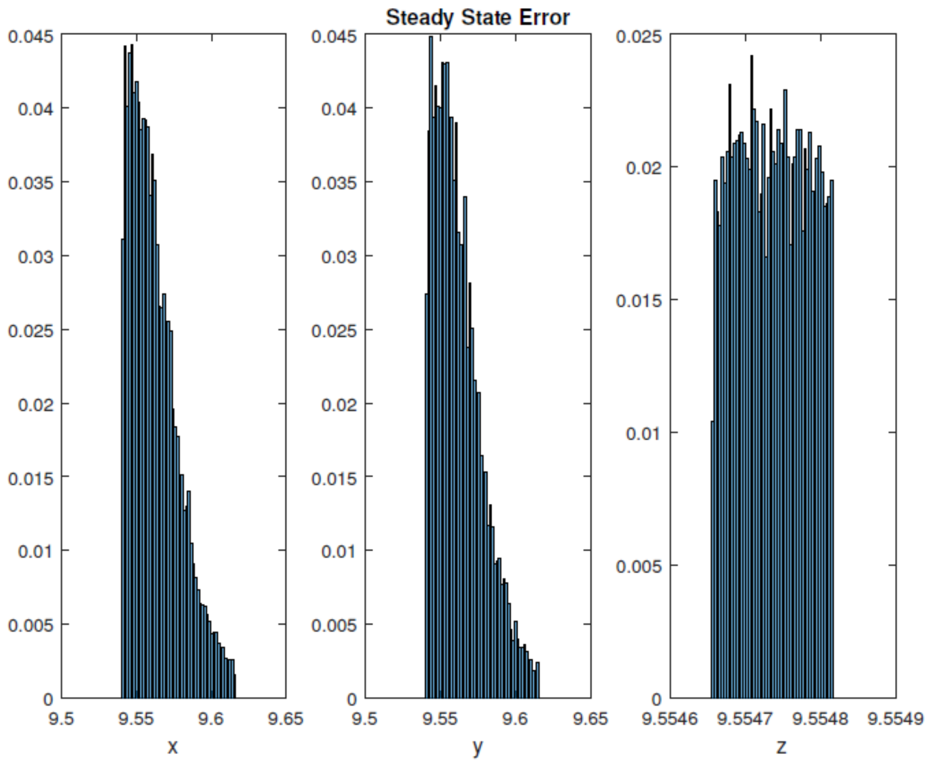


Figure 23. Distribution of steady state value. The normalized probability is plotted against angle in radians. These responses were generated using 9.74-degree step input. Based on that input, the system has extremely small amounts of error, effectively zero in all axes.

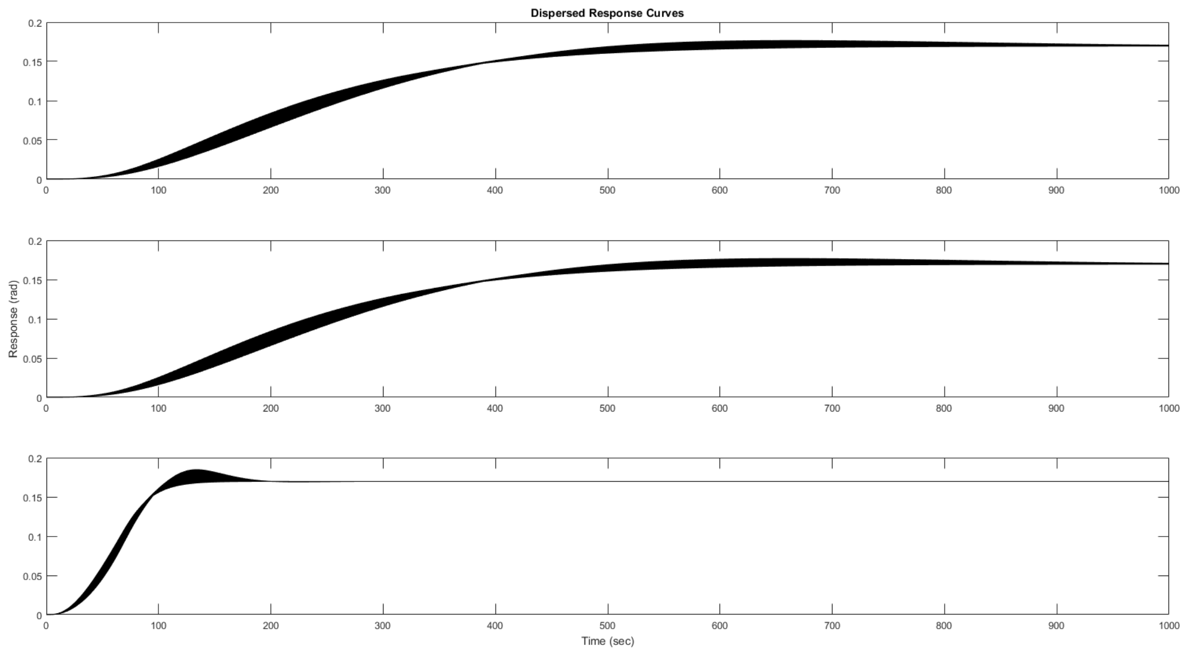
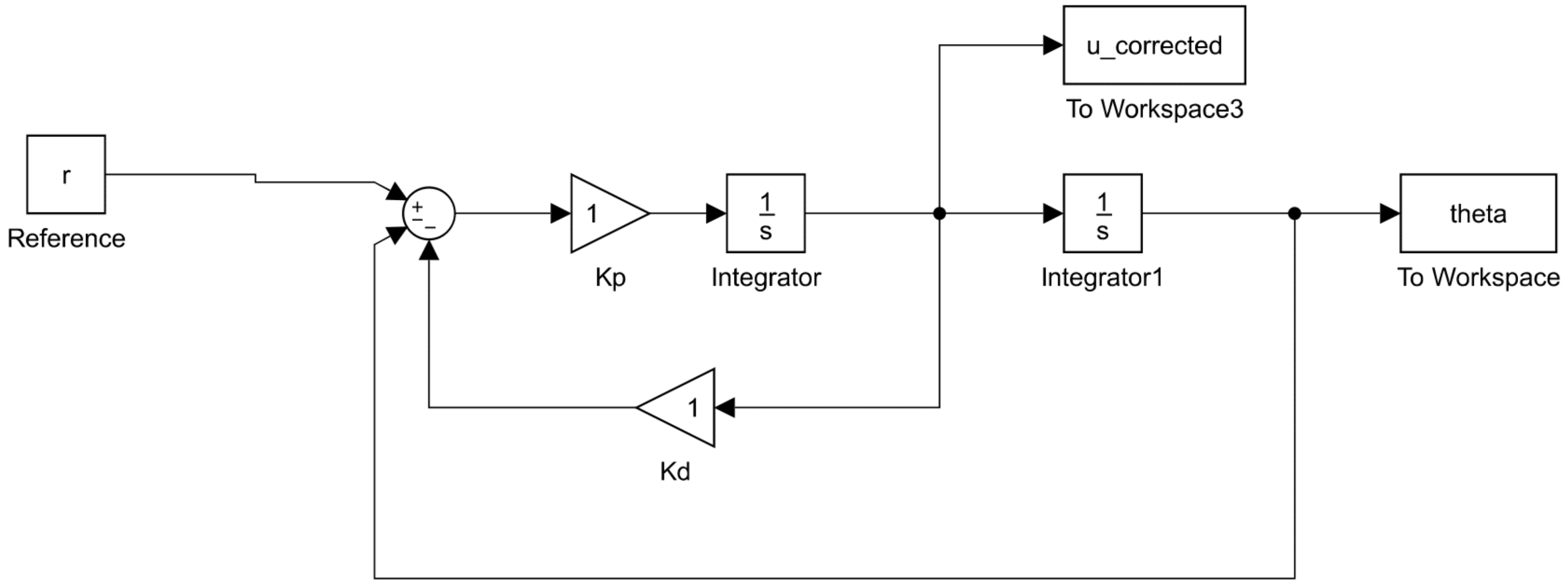


Figure 24. Response curve in x, y, and z axes. These plots show the dispersed response under given inputs, showing that the system is stable for all cases.

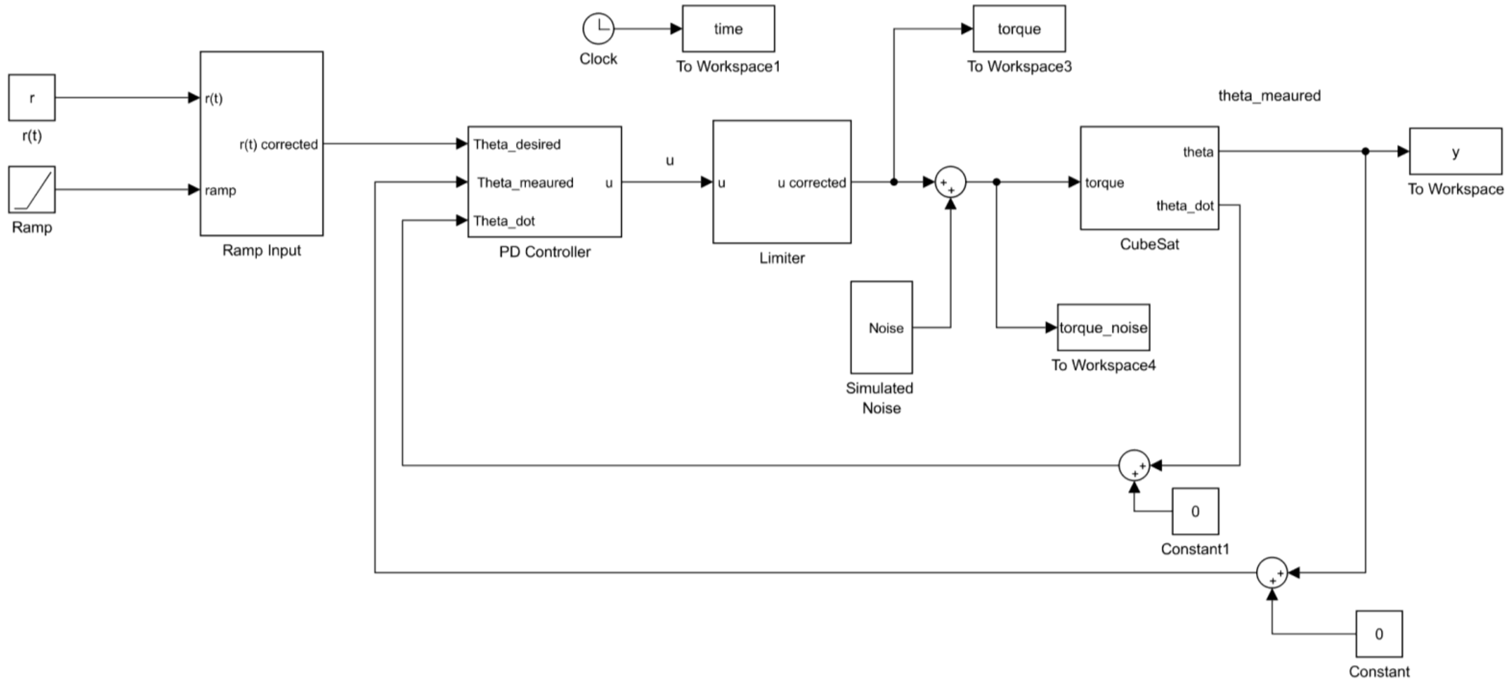
7.0 References

Seiler, Peter. Classical Control. 2017.

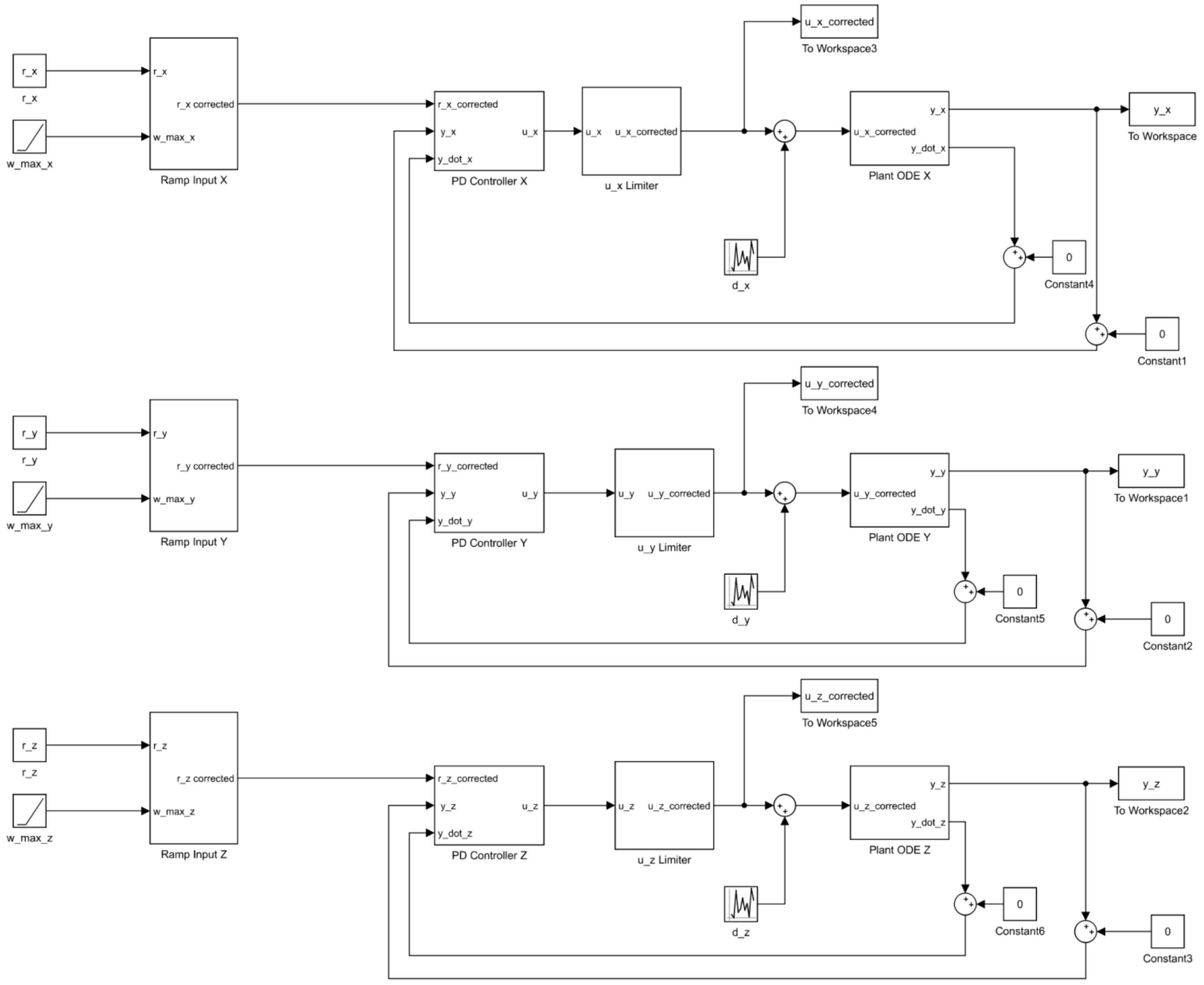
Appendix A – Simulink



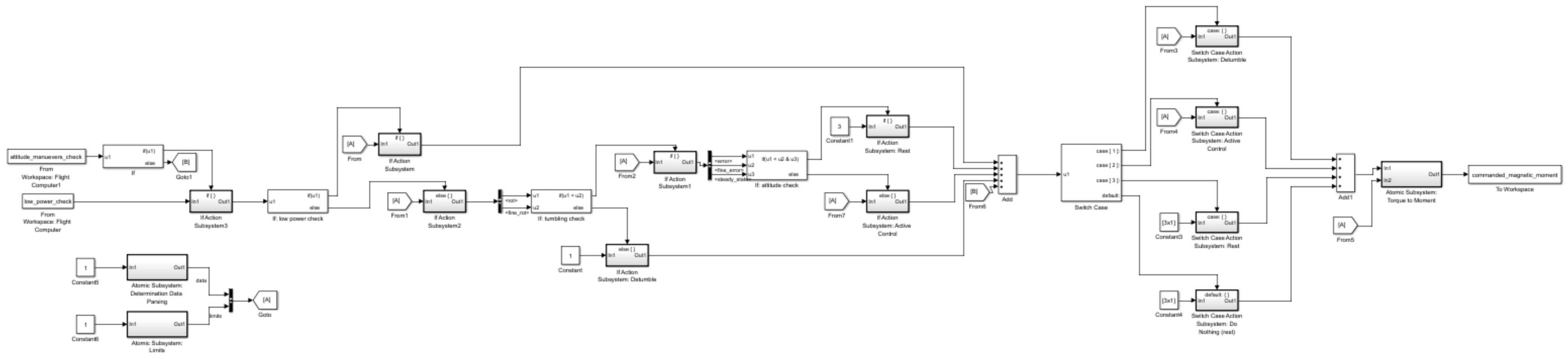
Model 1. An overview of the approximate transfer function controller developed in Simulink. The supporting MATLAB code can be found in Appendix B.



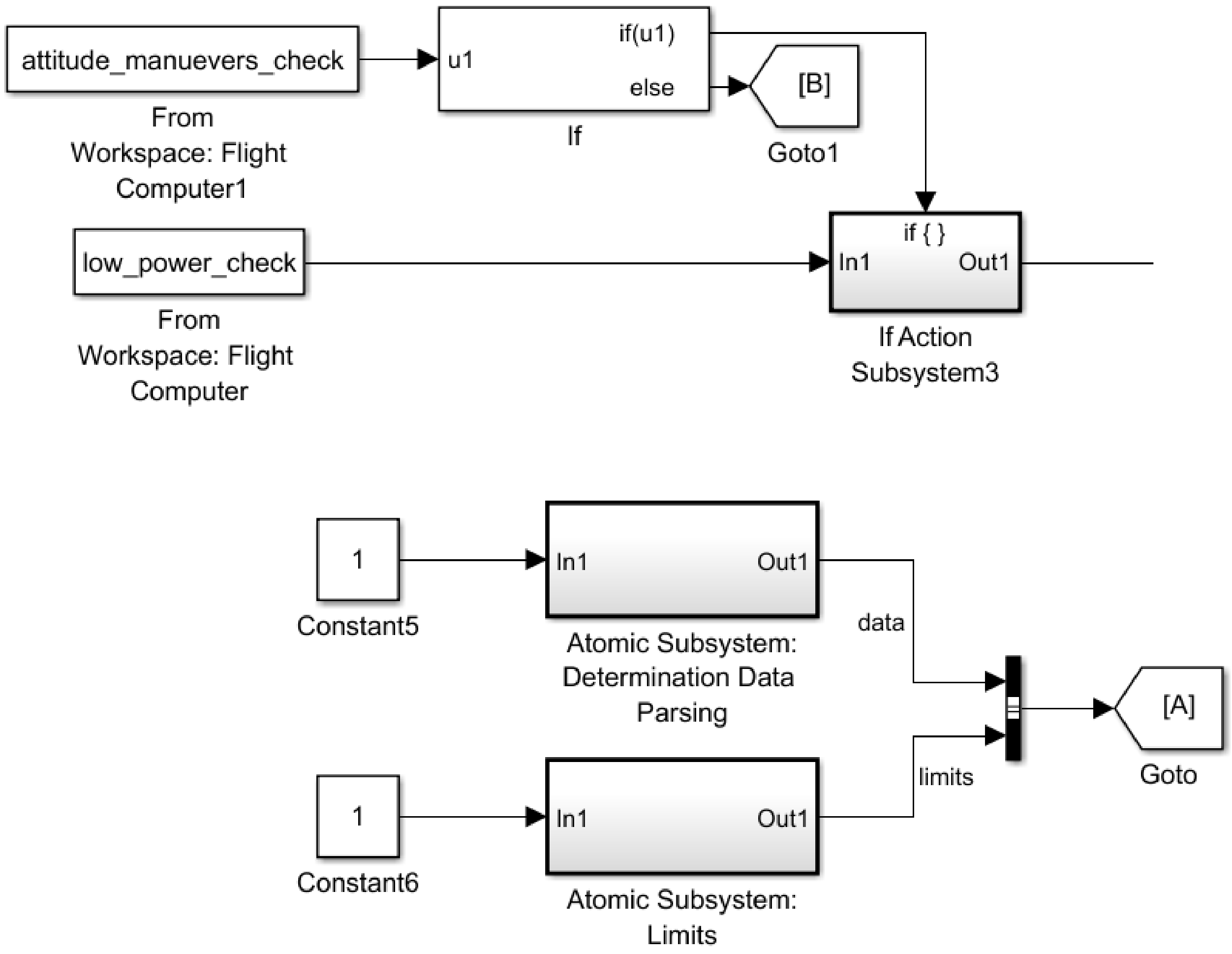
Model 2. An overview of the 1-dimensional controller developed in Simulink. This Simulink model is run at the sun synchronous orbit, plotting the magnetic field in the +x direction in ECI frame throughout the orbit. The supporting MATLAB code can be found in Appendix B.


r_x
r_x
w_max_x
r_x
r_x corrected
w_max_x
Ramp Input X
r_x_corrected
y_x
y_dot_x
u_x
PD Controller X
u_x
u_x_corrected
u_x Limiter
u_x_corrected
To Workspace3
u_x_corrected
y_x
y_dot_x
Plant ODE X
y_x
To Workspace
d_x
0
Constant4
0
Constant1
r_y
r_y
w_max_y
r_y
r_y corrected
w_max_y
Ramp Input Y
r_y_corrected
y_y
y_dot_y
u_y
PD Controller Y
u_y
u_y_corrected
u_y Limiter
u_y_corrected
To Workspace4
u_y_corrected
y_y
y_dot_y
Plant ODE Y
y_y
To Workspace1
d_y
0
Constant5
0
Constant2
r_z
r_z
w_max_z
r_z
r_z corrected
w_max_z
Ramp Input Z
r_z_corrected
y_z
y_dot_z
u_z
PD Controller Z
u_z
u_z_corrected
u_z Limiter
u_z_corrected
To Workspace5
u_z_corrected
y_z
y_dot_z
Plant ODE Z
y_z
To Workspace2
d_z
0
Constant6
0
Constant3

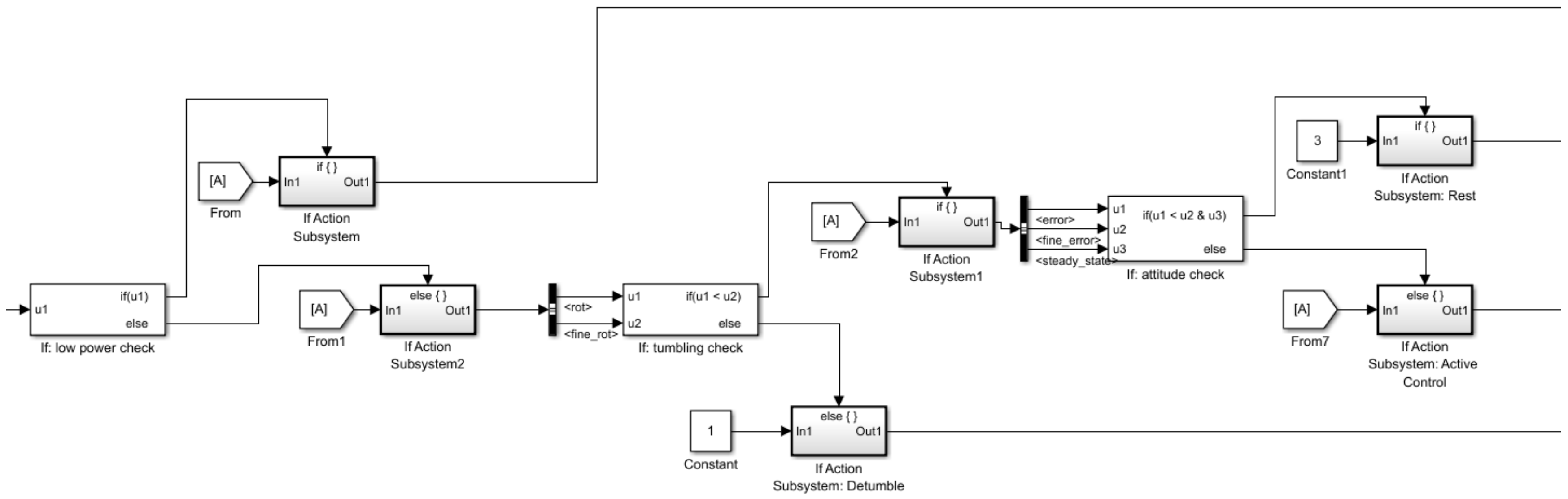
Model 3: An overview of the 3-dimensional controller developed in Simulink. This Simulink model is run at the sun synchronous orbit, plotting the magnetic field in all three directions (+x, +y, +z) in ECI frame throughout the orbit. The supporting MATLAB code can be found in Appendix B.



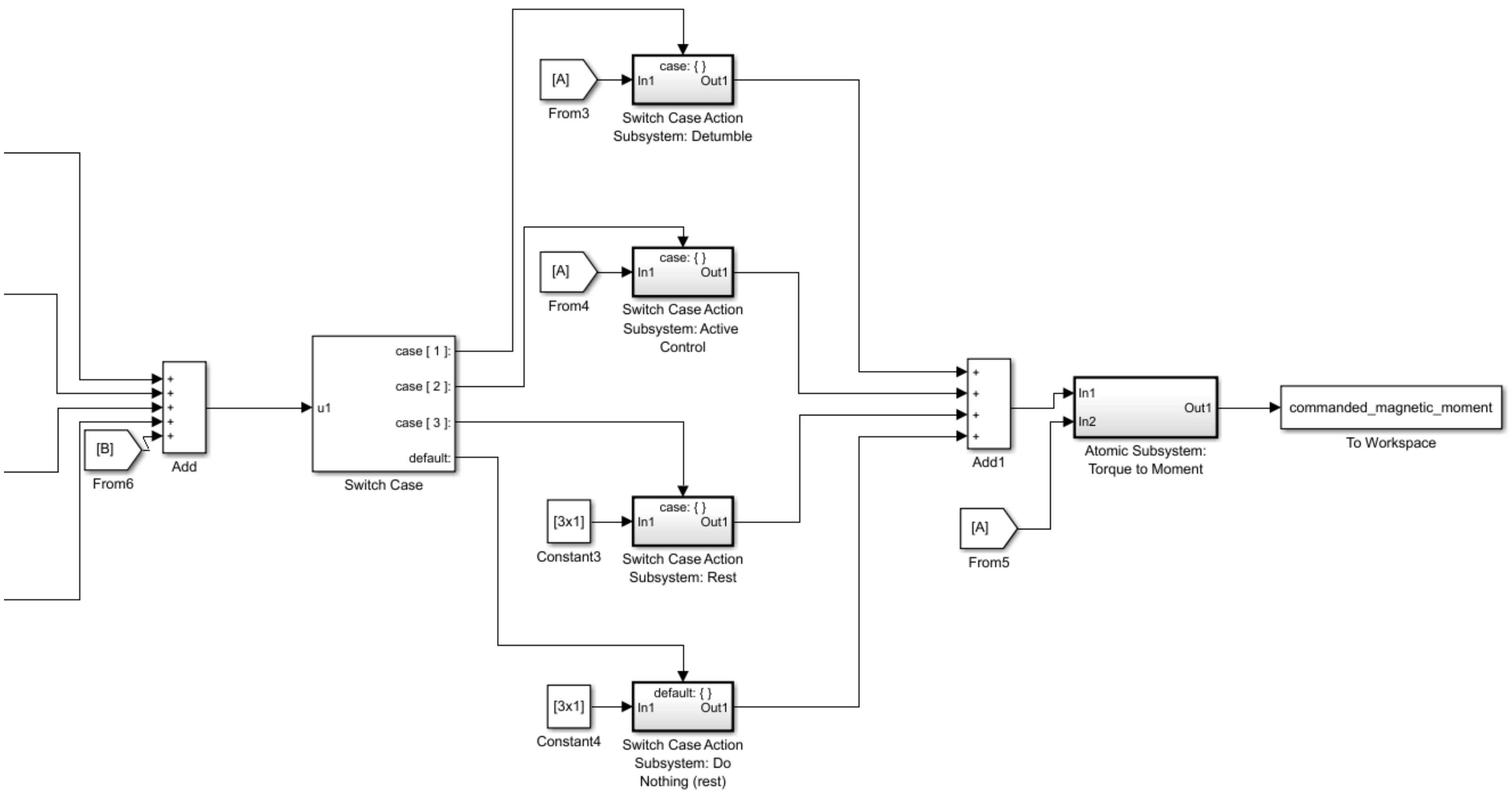
Model 1.0. An overview of the full guidance algorithm. The various parts are broken down and shown below.



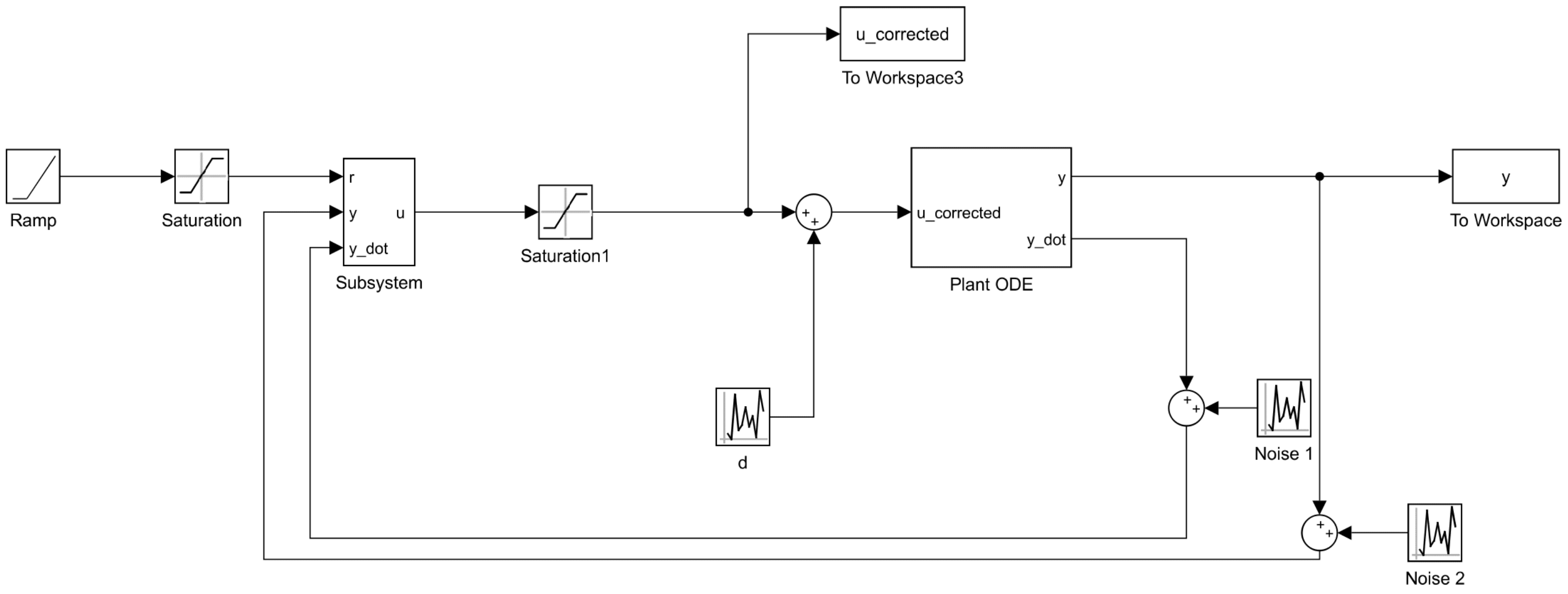
Model 1.1. The first section of the guidance algorithm is pictured above. Data is read in from the flight computer and the attitude determination system and the limits are retrieved. The first check is also conducted, whether or not enough time has passed that we can begin active control. The model continues on the right side, following the line in the top right corner.



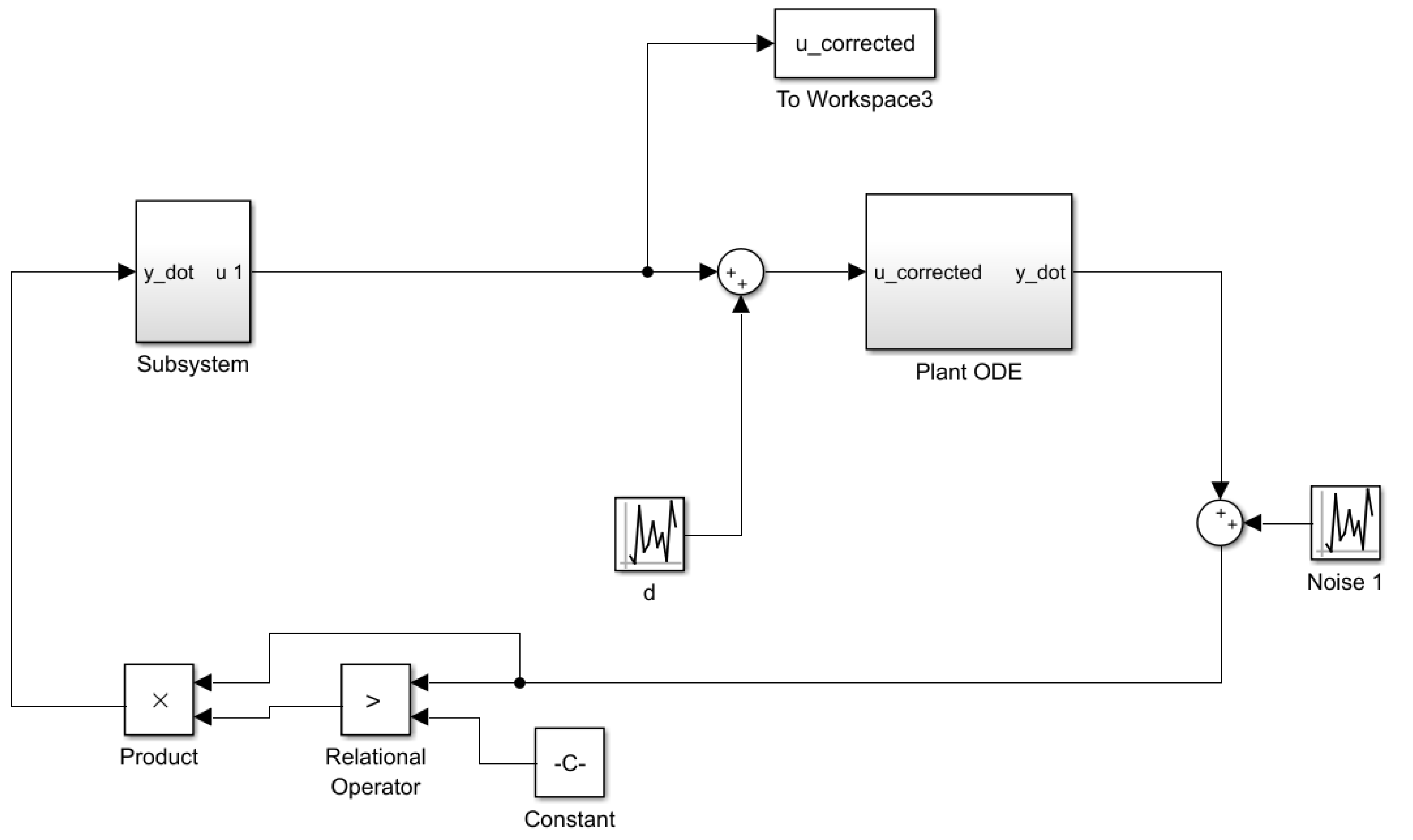
Model 1.2. The second section of the guidance algorithm is pictured above. The model continues from the left side, following the line coming in. The system first checks to see if we are in low power mode, followed by the tumbling check, then finally the attitude check. The state of each (on or off) is fed along the lines that lead to the far right side.


case: { }
[A]
From3
In1
Out1
Switch Case Action Subsystem: Detumble
case: { }
[A]
From4
Switch Case Action Subsystem: Active Control
[B]
From6
Add
u1
case [1]:
case [2]:
case [3]:
default:
Switch Case
[3x1]
Constant3
Switch Case Action Subsystem: Rest
default: { }
[3x1]
Constant4
Switch Case Action Subsystem: Do Nothing (rest)
Add1
In1
In2
Out1
Atomic Subsystem: Torque to Moment
[A]
From5
commanded_magnetic_moment
To Workspace

Model 1.3. The third section of the guidance algorithm is pictured above. The model continues from the left side, following the lines coming in. The system combines the states of each block into one block, and outputs the result in the form of a single state on the leftmost block. The next block to the right turns on the correct subsystem, which are located inside of the four blocks directly in the middle (left to right middle). The next block to the right reads in the commanded torque and outputs that to the next block that determines the commanded magnetorquer magnetic moment and outputs that to the system.



Model 4: An overview of the matrix controller developed in Simulink. This Simulink model is run at the sun synchronous orbit, plotting the magnetic field in all three directions (+x, +y, +z) in ECI frame throughout the orbit. The supporting MATLAB code can be found in Appendix B.



Model 5: An overview of the detumbling controller developed in Simulink. This Simulink model is run at the sun synchronous orbit, plotting the magnetic field in all three directions (+x, +y, +z) in ECI frame throughout the orbit. The supporting MATLAB code can be found in Appendix B.

Appendix B – Code

Code 1: ApproxTfCode.m

```
%% 3.0.1 Testing an Approximate Transfer Function
% By Evan Majd
% 11/27/2016

r = .17; % Angle (Radians)
Tf = 20; % Simulation Time (Seconds)

sim('ApproxTfModel',[0 Tf]);
subplot(1,2,1)
plot(theta.Time(:,1),theta.Data(:,1).*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(1,2,2)
plot(theta.Time(:,1),u_corrected.Data(:,1), 'r');
title('Torque Done')
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;

temp_x = theta.Data(1,1);
for i=2:(length(theta.Data(:,1))-11)
    bool = abs((theta.Data(i,1)-theta.Data(i+10,1)));
    if bool < (1e-2)
        temp_x = theta.Data(i,1);
    end
end

steady_state_x = temp_x*180/pi;

s_x = stepinfo(theta.Data(:,1), theta.Time,temp_x,'RiseTimeLimits',[0.05,0.95]);

 s1_x = strcat('Rise Time X (s):', num2str(s_x.RiseTime));
 s2_x = strcat('Settling Time X (s):',num2str(s_x.SettlingTime));
 s3_x = strcat('Overshoot X (%):',num2str(s_x.Overshoot));
 s4_x = strcat('Peak Value X (deg):',num2str(s_x.Peak*180/pi));
 s5_x = strcat('Peak Time X (s):',num2str(s_x.PeakTime));
 s6_x = strcat('Steady State X (deg):', num2str(steady_state_x));

 disp(steady_state_x)
 disp(s2_x)
disp(s4_x)
```

Code 2: OneDAlgorithm.m

```
%% AEM 4331: CubeSat Control Algorithm
% PD - Controller
% 10/09/2016
% By Evan Majd

%% Sun Synchronous Orbit: Find XYZ positions and Plot
clc
clear vars

T = linspace(0,98.9*60,101); %min orbital period

% Orbital Parameters for Sun Synchronous Orbit (pulled from web)
a = 6378+705; %km semi major axis
inc = 98.2; %degrees inclination
e = 0; %eccentricity
omega = 270; %RAAN degrees
w = 90; %argument of perigee degrees
v = 0; % true anomaly degrees

% Gets the position and velocity vectors
[x,y,z] = orbit(a, e, omega*pi/180, inc*pi/180, w*pi/180);
[xv,yv,zv] = trueanomaly(a,e,omega*pi/180,inc*pi/180,w*pi/180,v*pi/180);
r = 6371;
[xp,yp,zp] = planet(r,20); % earth radius ( km )

if( inc > 90 )
    orbit_color = 'red'; % retrograde
elseif (inc == 90 || inc == 180 )
    orbit_color = 'black'; % polar
else
    orbit_color = 'blue'; % direct
end

figure(1);
subplot(1,2,1)
plot3(x,y,z,orbit_color,'Linewidth',2); % direct
axis equal
hold on

plot3(xv,yv,zv,'blacko','Linewidth',2); % plot true anomaly
surf(xp,yp,zp,'EdgeAlpha',0.4);   % plot the planet
colormap([0  0.5  0.8]);
scale = 2;
axis([-scale*a scale*a -scale*a scale*a -scale*a scale*a])
```

```
hold on


% Get Axis properties
axis_data = get(gca);
xmin = axis_data.XLim(1);
xmax = axis_data.XLim(2);
ymin = axis_data.YLim(1);
ymax = axis_data.YLim(2);
zmin = axis_data.ZLim(1);
zmax = axis_data.ZLim(2);

% I, J ,K vectors
plot3([xmin,xmax],[0 0],[0 0],'black','Linewidth',1); plot3(xmax,0,0,'black>','Linewidth',1.5);
hold on
plot3([0 0],[ymin,ymax],[0 0],'black','Linewidth',1); plot3(0,ymax,0,'blue>','Linewidth',1.5);
hold on
plot3([0 0],[0 0],[zmin,zmax],'black','Linewidth',1); plot3(0,0,zmax,'red^','Linewidth',1.5);
hold on

% right ascending node line plot
xomega_max = xmax*cos(omega*pi/180);
xomega_min = xmin*cos(omega*pi/180);
yomega_max = ymax*sin(omega*pi/180);
yomega_min = ymin*sin(omega*pi/180);

xlabel('I');
ylabel('J');
zlabel('K');

plot3([xomega_min xomega_max], [yomega_min yomega_max], [0 0], 'g','Linewidth',1.5);
hold on

% add equatorial plan
xe = [xmin xmax;xmin xmax]; ye = [ymax ymax;ymin ymin]; ze = [0 0; 0 0];
eq_alpha = 0.3; % transparancy
mesh(xe,ye,ze,'FaceAlpha',eq_alpha,'FaceColor',[0.753,0.753,0.753]);

grid on
hold off

%% Calculate Earth's Mag Field: Convert XYZ to Lat/Lon coordinates
[lat,lon,h]=xyz2ell(x,y,z,a,0);

alt = 705*ones(1,101);
lat = 180*lat/pi;
```

```
lon = 180*lon/pi;

for k =1:length(lat)
    [xyz,h,dec,dip,f] = wrldmagm(alt(k),lat(k),lon(k),2016);
    f_save(k) = f;
end

% F is total magnetic field intensity
f_save = f_save.*(1e-9); % convert nT to T

% Plot of magnetic field (y) vs lattitude location (x)
subplot(1,2,2)
plot(lat,f_save,'r.');
title('Mag Earth')
xlabel('Lattitude (Degrees)');
ylabel('Magnetic Field (T)');
grid on;
earth_mag = sum(f_save)/length(f_save);
T = [T;f_save];
T = T'; % magnetic fields with corresponding time in orbit

%% PD Controller Input and Simulation

% Values you can change
r = .17; % Desired Theta in radian/sec (.17 rad = 10 deg)
damping_ratio = 2; % Damping Ratio
wmax = .05/180*pi;
wn = .05; % Natural Frequency in Hz
mass = 0.54944; % Kg
Ixx = 0.05440; % kg*m^2
Iyy = 0.01441; % kg*m^2
Izz = 0.05128; % kg*m^2

% Set variables
Tf = 1000; % Seconds, time of simulation
I = Ixx; % Rotating About X - Axis

t_limit = .24*(earth_mag); % magnetorquer mag field crossed with earth max mag field
computed in prev. section

Kp = wn^2*I;
Kd = 2*damping_ratio*wn*I;

% Poles calculation
a = 1;
b = Kd/I;
```

```
c = Kp/I;
p = [a b c];
poles = roots(p);

figure(2);
sim('OneDAlgorithmModel',[0 Tf]);
subplot(2,2,1)
plot(y.Time,y.Data.*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(ydot.Time,ydot.Data.*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;

subplot(2,2,3)
plot(ydot.Time,torque.Data, 'r');
title('Torque Done');
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;
hold on
plot(ydot.Time,t_limit.*ones(length(ydot.Time),1),'k--')
hold on
plot(ydot.Time,-t_limit.*ones(length(ydot.Time),1),'k--')

subplot(2,2,4)
plot([poles(1) poles(2)],[0 0], '*');
title('Poles');
xlabel('real');
ylabel('Imaginary');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-3 3 -1 1])

% Find Steady State
temp1 = y.Data(1);
for i=2:(length(y.Data)-11)
    bool = abs((y.Data(i)-y.Data(i+10)));
    if bool < (1e-2)
        temp1 = y.Data(i);
    end
end

steady_state = temp1*180/pi;

% Display
s = stepinfo(y.Data, y.Time,temp1,'RiseTimeLimits',[0.05,0.95]);

s1 = strcat('Rise Time (s):', num2str(s.RiseTime));
s2 = strcat('Settling Time (s):',num2str(s.SettlingTime));
s3 = strcat('Overshoot (%):',num2str(s.Overshoot));
s4 = strcat('Peak Value (deg):',num2str(s.Peak*180/pi));
s5 = strcat('Peak Time (s):',num2str(s.PeakTime));
s6 = strcat('Steady State (deg):', num2str(steady_state));

disp(s1)
disp(s2)
disp(s3)
disp(s4)
disp(s5)
disp(s6)
```

Code 3: ThreeDAlgorithm.m

```
%% PD Controller Input and Simulation
% Constants (Misc)
clc
clear vars

earth_mag
mass = 0.54944; % Mass (kg)
Tf = 2000; % Simulation time (s)
magnetorquer_x = .24; % N*m/T
magnetorquer_y = .24; % N*m/T
magnetorquer_z = .13; % N*m/T

% Reference Commands
r_x = .17; % (theta) radians
r_y = .17; % (theta) radians
r_z = .17; % (theta) radians
w_max_x = .05/180*pi; % (theta dot) rad/s
w_max_y = .05/180*pi; % (theta dot) rad/s
w_max_z = .05/180*pi; % (theta dot) rad/s

% Moments of Inertia
Ixx = 0.05440; % kg*m^2
Ixy = 0.00538; % kg*m^2
Ixz = 0.00161; % kg*m^2
Iyx = 0.00538; % kg*m^2
Iyy = 0.01441; % kg*m^2
Iyz = 0.01271; % kg*m^2
Izx = 0.00161; % kg*m^2
Izy = 0.01271; % kg*m^2
Izz = 0.05128; % kg*m^2

% Controller Values
damping_ratio_x = 5;
damping_ratio_y = 4;
damping_ratio_z = 2;
wn_x = .05; % Natural Frequency (Hz)
wn_y = .05; % Natural Frequency (Hz)
wn_z = .05; % Natural Frequency (Hz)

Kp_x = wn_x^2*Ixx; % Proportional Constant
Kp_y = wn_y^2*Iyy; % Proportional Constant
Kp_z = wn_z^2*Izz; % Proportional Constant
Kd_x = 2 * damping_ratio_x * wn_x * Ixx; % Derivative Constant
Kd_y = 2 * damping_ratio_y * wn_y * Iyy; % Derivative Constant
```

```
Kd_z = 2 * damping_ratio_z * wn_z * Izz; % Derivative Constant

% Max Torque Limitations
t_limit_x = -1*magnetorquer_x * (earth_mag_y); % T = u x B
t_limit_y = magnetorquer_y * (earth_mag_z); % T = u x B
t_limit_z = magnetorquer_z * (earth_mag_x); % T = u x B


% Poles calculation
a_x = 1;
a_y = 1;
a_z = 1;

b_x = Kd_x/Ixx;
b_y = Kd_y/Iyy;
b_z = Kd_z/Izz;

c_x = Kp_x/Ixx;
c_y = Kp_y/Iyy;
c_z = Kp_z/Izz;

p_x = [a_x b_x c_x];
p_y = [a_y b_y c_y];
p_z = [a_z b_z c_z];

poles_x = roots(p_x);
poles_y = roots(p_y);
poles_z = roots(p_z);

%% Plot Graphs X
figure(2);
sim('ThreeDAlgorithmModel',[0 Tf]);
subplot(2,2,1)
plot(y_x.Time,y_x.Data.*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(y_dot_x.Time,y_dot_x.Data.*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;
```

```
subplot(2,2,3)
plot(y_dot_x.Time,u_x_corrected.Data, 'r');
title('Torque Done');
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;
hold on
plot(y_dot_x.Time,t_limit_x.*ones(length(y_dot_x.Time),1),'k--')
hold on
plot(y_dot_x.Time,-t_limit_x.*ones(length(y_dot_x.Time),1),'k--')

subplot(2,2,4)
plot([poles_x(1) poles_x(2)],[0 0], '*');
title('Poles');
xlabel('Real');
ylabel('Imaginary');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-3 3 -1 1])

% Find Steady State
temp_x = y_x.Data(1);
for i=2:(length(y_x.Data)-11)
    bool = abs((y_x.Data(i)-y_x.Data(i+10)));
    if bool < (1e-2)
        temp_x = y_x.Data(i);
    end
end

steady_state_x = temp_x*180/pi;

% Display
s_x = stepinfo(y_x.Data, y_x.Time,temp_x,'RiseTimeLimits',[0.05,0.95]);

s1_x = strcat('Rise Time X (s):', num2str(s_x.RiseTime));
s2_x = strcat('Settling Time X (s):',num2str(s_x.SettlingTime));
s3_x = strcat('Overshoot X (%):',num2str(s_x.Overshoot));
s4_x = strcat('Peak Value X (deg):',num2str(s_x.Peak*180/pi));
s5_x = strcat('Peak Time X (s):',num2str(s_x.PeakTime));
s6_x = strcat('Steady State X (deg):', num2str(steady_state_x));

disp(s1_x)
disp(s2_x)
disp(s3_x)
disp(s4_x)
disp(s5_x)
disp(s6_x)
fprintf('\n')

%% Plot Graphs Y
figure(3);
subplot(2,2,1)
plot(y_y.Time,y_y.Data.*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(y_dot_y.Time,y_dot_y.Data.*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;

subplot(2,2,3)
plot(y_dot_y.Time,u_y_corrected.Data, 'r');
title('Torque Done');
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;
hold on
plot(y_dot_y.Time,t_limit_y.*ones(length(y_dot_y.Time),1),'k--')
hold on
plot(y_dot_y.Time,-t_limit_y.*ones(length(y_dot_y.Time),1),'k--')

subplot(2,2,4)
plot([poles_y(1) poles_y(2)],[0 0], '*');
title('Poles');
xlabel('Real');
ylabel('Imaginary');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-3 3 -1 1])

% Find Steady State
temp_y = y_y.Data(1);
for i=2:(length(y_y.Data)-11)
```

```
    bool = abs((y_y.Data(i)-y_y.Data(i+10)));
    if bool < (1e-2)
        temp_y = y_y.Data(i);
    end
end

steady_state_y = temp_y*180/pi;

% Display
s_y = stepinfo(y_y.Data, y_y.Time,temp_y,'RiseTimeLimits',[0.05,0.95]);

s1_y = strcat('Rise Time Y (s):', num2str(s_y.RiseTime));
s2_y = strcat('Settling Time Y (s):',num2str(s_y.SettlingTime));
s3_y = strcat('Overshoot Y (%):',num2str(s_y.Overshoot));
s4_y = strcat('Peak Value Y (deg):',num2str(s_y.Peak*180/pi));
s5_y = strcat('Peak Time Y (s):',num2str(s_y.PeakTime));
s6_y = strcat('Steady State Y (deg):', num2str(steady_state_y));

disp(s1_y)
disp(s2_y)
disp(s3_y)
disp(s4_y)
disp(s5_y)
disp(s6_y)
fprintf('\n')

%% Plot Graphs Z
figure(4);
subplot(2,2,1)
plot(y_z.Time,y_z.Data.*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(y_dot_z.Time,y_dot_z.Data.*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;

subplot(2,2,3)
plot(y_dot_z.Time,u_z_corrected.Data, 'r');
title('Torque Done');
xlabel('Time (s)');
```

```
ylabel('Torque (Nm)');
grid on;
hold on
plot(y_dot_z.Time,t_limit_z.*ones(length(y_dot_z.Time),1),'k--')
hold on
plot(y_dot_z.Time,-t_limit_z.*ones(length(y_dot_z.Time),1),'k--')

subplot(2,2,4)
plot([poles_z(1) poles_z(2)],[0 0], '*');
title('Poles');
xlabel('Real');
ylabel('Imaginary');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-3 3 -1 1])

% Find Steady State
temp_z = y_z.Data(1);
for i=2:(length(y_z.Data)-11)
    bool = abs((y_z.Data(i)-y_z.Data(i+10)));
    if bool < (1e-2)
        temp_z = y_z.Data(i);
    end
end

steady_state_z = temp_z*180/pi;

% Display
s_z = stepinfo(y_z.Data, y_z.Time,temp_z,'RiseTimeLimits',[0.05,0.95]);

s1_z = strcat('Rise Time Z (s):', num2str(s_z.RiseTime));
s2_z = strcat('Settling Time Z (s):',num2str(s_z.SettlingTime));
s3_z = strcat('Overshoot Z (%):',num2str(s_z.Overshoot));
s4_z = strcat('Peak Value Z (deg):',num2str(s_z.Peak*180/pi));
s5_z = strcat('Peak Time Z (s):',num2str(s_z.PeakTime));
s6_z = strcat('Steady State Z (deg):', num2str(steady_state_z));

disp(s1_z)
disp(s2_z)
disp(s3_z)
disp(s4_z)
disp(s5_z)
disp(s6_z)
```

Code 4: matrixalgorithmcode.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% AEM 4331
% Control Values
%
% Evan Majd
% 12/2/16
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear vars

% Simulation Time
Tf = 1000;

% Earth Magnetic field
earth_mag = [1.9547e-05 -1.3174e-06 5.8042e-06]; % [x y z] T

% Magnetorquer Dipole Magnetic Moment
magnetorquer = [.24 .24 .13]; % [x y z] N*m/T

% Reference Commands
r = [.17 .17 .17]; % (theta) radians
w_max = [.15/180*pi .15/180*pi .15/180*pi]; % (theta dot) rad/s

% Moments (Ixx, Iyy, Izz)
I = [109242619.73177/1000^3 109302085.08329/1000^3 3557528.64834/1000^3]; % kg*m^2

% Controller Values
damping_ratio = [1 1 1];
wn = [.01 .01 .07]; % Natural Frequency (Hz)

Kp = wn.^2.*I; % Proportional Constant
Kd = 2.*damping_ratio.*wn.*I; % Derivative Constant

% Max Torque Limitations
t_limit = abs(cross(magnetorquer,earth_mag)); % T = u x B

% Poles calculation
a = [1 1 1];
b = Kd./I;
c = Kp./I;
```

```
p = [a(1) b(1) c(1); a(2) b(2) c(2); a(3) b(3) c(3)];

poles = [real(roots(p(1,:))),real(roots(p(2,:))),real(roots(p(3,:)))];

%% X Response

figure(2);
sim('matrixalgorithm',[0 Tf]);
subplot(2,2,1)
plot(y.Time(:,1),y.Data(:,1).*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(y_dot.Time(:,1),y_dot.Data(:,1).*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;

subplot(2,2,3)
plot(y_dot.Time(:,1),u_corrected.Data(:,1), 'r');
title('Torque Commanded');
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;
hold on
plot(y_dot.Time(:,1),t_limit(1).*ones(length(y_dot.Time(:,1)),1),'k--')
hold on
plot(y_dot.Time(:,1),-t_limit(1).*ones(length(y_dot.Time(:,1)),1),'k--')

subplot(2,2,4)
plot([poles(1,1) poles(2,1)],[0 0], '*');
title('Poles');
xlabel('Real');
ylabel('Imaginary');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-1 1 -1 1])

% Find Steady State
```

```
temp_x = y.Data(1,1);
for i=2:(length(y.Data(:,1))-11)
    bool = abs((y.Data(i,1)-y.Data(i+10,1)));
    if bool < (1e-2)
        temp_x = y.Data(i,1);
    end
end

steady_state_x = temp_x*180/pi;

% Obtain Data
s_x = stepinfo(y.Data(:,1), y.Time,temp_x,'RiseTimeLimits',[0.05,0.95]);

%% Plot Graphs Y
figure(3);
subplot(2,2,1)
plot(y.Time,y.Data(:,2).*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(y_dot.Time,y_dot.Data(:,2).*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;

subplot(2,2,3)
plot(y_dot.Time,u_corrected.Data(:,2), 'r');
title('Torque Commanded');
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;
hold on
plot(y_dot.Time(:,1),t_limit(2).*ones(length(y_dot.Time(:,1)),1),'k--')
hold on
plot(y_dot.Time(:,1),-t_limit(2).*ones(length(y_dot.Time(:,1)),1),'k--')

subplot(2,2,4)
plot([poles(1,2) poles(2,2)],[0 0], '*');
title('Poles');
xlabel('Real');
ylabel('Imaginary');
grid on;
```

```
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-1 1 -1 1])

% Find Steady State
temp_y = y.Data(1,2);
for i=2:(length(y.Data(:,2))-11)
    bool = abs((y.Data(i,2)-y.Data(i+10,2)));
    if bool < (1e-2)
        temp_y = y.Data(i,2);
    end
end

steady_state_y = temp_y*180/pi;

% Obtain Data
s_y = stepinfo(y.Data(:,2), y.Time,temp_y,'RiseTimeLimits',[0.05,0.95]);

%% Plot Graphs Z
figure(4);
subplot(2,2,1)
plot(y.Time,y.Data(:,3).*(180/pi), 'r');
title('Attitude Response');
xlabel('Time (s)');
ylabel('Theta (degrees)');
grid on;

subplot(2,2,2)
plot(y_dot.Time,y_dot.Data(:,3).*(180/pi), 'r');
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Theta Dot (degrees/s)');
grid on;

subplot(2,2,3)
plot(y_dot.Time,u_corrected.Data(:,3), 'r');
title('Torque Commanded');
xlabel('Time (s)');
ylabel('Torque (Nm)');
grid on;
hold on
plot(y_dot.Time(:,1),t_limit(3).*ones(length(y_dot.Time(:,1)),1),'k--')
hold on
plot(y_dot.Time(:,1),-t_limit(3).*ones(length(y_dot.Time(:,1)),1),'k--')
```

```
subplot(2,2,4)
plot([poles(1,3) poles(2,3)],[0 0], '*');
title('Poles');
xlabel('Real');
ylabel('Imaginary');
grid on;
ax = gca;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
axis([-1 1 -1 1])

% Find Steady State
temp_z = y.Data(1,3);
for i=2:(length(y.Data(:,3))-11)
    bool = abs((y.Data(i,3)-y.Data(i+10,3)));
    if bool < (1e-2)
        temp_z = y.Data(i,3);
    end
end

steady_state_z = temp_z*180/pi;

% Obtain Data
s_z = stepinfo(y.Data(:,3), y.Time,temp_z,'RiseTimeLimits',[0.05,0.95]);
```

Code 5: earth_mag.m

```
%% Sun Synchronous Orbit: Find XYZ positions and Plot

T = linspace(0,98.9*60,101); %min orbital period

% Orbital Parameters for Sun Synchronous Orbit (pulled from web)
a = 6378+705; %km semi major axis
inc = 98.2; %degrees inclination
e = 0; %eccentricity
omega = 270; %RAAN degrees
w = 90; %argument of perigee degrees
v = 0; % true anomaly degrees

% Gets the position and velocity vectors
[x,y,z] = orbit(a, e, omega*pi/180, inc*pi/180, w*pi/180);
[xv,yv,zv] = trueanomaly(a,e,omega*pi/180,inc*pi/180,w*pi/180,v*pi/180);
r = 6371;
[xp,yp,zp] = planet(r,20); % earth radius ( km )

if( inc > 90 )
    orbit_color = 'red'; % retrograde
elseif (inc == 90 || inc == 180 )
    orbit_color = 'black'; % polar
else
    orbit_color = 'blue'; % direct
end

figure(1);
subplot(2,2,1)
plot3(x,y,z,orbit_color,'Linewidth',2); % direct
axis equal
hold on

plot3(xv,yv,zv,'blacko','Linewidth',2); % plot true anomaly
surf(xp,yp,zp,'EdgeAlpha',0.4);   % plot the planet
colormap([0  0.5  0.8]);
scale = 2;
axis([-scale*a scale*a -scale*a scale*a -scale*a scale*a])
hold on


% Get Axis properties
axis_data = get(gca);
xmin = axis_data.XLim(1);
xmax = axis_data.XLim(2);
```

```
ymin = axis_data.YLim(1);
ymax = axis_data.YLim(2);
zmin = axis_data.ZLim(1);
zmax = axis_data.ZLim(2);

% I, J ,K vectors
plot3([xmin,xmax],[0 0],[0 0],'black','Linewidth',1); plot3(xmax,0,0,'black>','Linewidth',1.5);
hold on
plot3([0 0],[ymin,ymax],[0 0],'black','Linewidth',1); plot3(0,ymax,0,'blue>','Linewidth',1.5);
hold on
plot3([0 0],[0 0],[zmin,zmax],'black','Linewidth',1); plot3(0,0,zmax,'red^','Linewidth',1.5);
hold on

% right ascending node line plot
xomega_max = xmax*cos(omega*pi/180);
xomega_min = xmin*cos(omega*pi/180);
yomega_max = ymax*sin(omega*pi/180);
yomega_min = ymin*sin(omega*pi/180);

xlabel('I');
ylabel('J');
zlabel('K');

plot3([xomega_min xomega_max], [yomega_min yomega_max], [0 0], 'g','Linewidth',1.5);
hold on

% add equatorial plan
xe = [xmin xmax;xmin xmax]; ye = [ymax ymax;ymin ymin]; ze = [0 0; 0 0];
eq_alpha = 0.3; % transparancy
mesh(xe,ye,ze,'FaceAlpha',eq_alpha,'FaceColor',[0.753,0.753,0.753]);

grid on
hold off

%% Calculate Earth's Mag Field: Convert XYZ to Lat/Lon coordinates
[lat,lon,h]=xyz2ell(x,y,z,a,0);

alt = 705*ones(1,101);
lat = 180*lat/pi;
lon = 180*lon/pi;

for k =1:length(lat)
   [xyz,h,dec,dip,f] = wrldmagm(alt(k),lat(k),lon(k),2016);
   f_save(k) = f;
   xmag(k) = xyz(1);
   ymag(k) = xyz(2);
```

```
    zmag(k) = xyz(3);


end

% F is total magnetic field intensity
f_save = f_save.*(1e-9); % convert nT to T
xmag = xmag.*(1e-9); % convert nT to T
ymag = ymag.*(1e-9); % convert nT to T
zmag = zmag.*(1e-9); % convert nT to T

% Plot of magnetic field (y) vs lattitude location (x)
subplot(2,2,2)
plot(lat,xmag,'r.');
title('Mag Earth X')
xlabel('Lattitude (Degrees)');
ylabel('Magnetic Field (T)');
grid on;

subplot(2,2,3)
plot(lat,ymag,'r.');
title('Mag Earth Y')
xlabel('Lattitude (Degrees)');
ylabel('Magnetic Field (T)');
grid on;

subplot(2,2,4)
plot(lat,zmag,'r.');
title('Mag Earth Z')
xlabel('Lattitude (Degrees)');
ylabel('Magnetic Field (T)');
grid on;


earth_mag_x = sum(xmag)/length(xmag);
earth_mag_y = sum(ymag)/length(ymag);
earth_mag_z = sum(zmag)/length(zmag);

Tx = [T;xmag];
Ty = [T;ymag];
Tz = [T;zmag];
Tx = Tx'; % magnetic fields with corresponding time in orbit
Ty = Ty'; % magnetic fields with corresponding time in orbit
Tz = Tz'; % magnetic fields with corresponding time in orbit
```

Code 6: orbit.m

```
function [x,y,z] = orbit(a,e,omega,inc,w,n)
%
% orbit(a,e,i)
%
% a: semimajor axis
% e: excentricity
% i: inclination
% w: argument of perigee
% n: number of points [optional]

if (nargin == 5 )
    n = 100; % Default
end

if (nargin == 0 || nargin > 6)
    fprintf(1,'error -> wrong number of arguments\n');
    return;
end

% v = true anomaly
v = 0:2*pi/n:2*pi;

% 2-body problem orbit determination
r = a*(1-e^2)./(1+e*cos(v));

x = r.*cos(v);
y = r.*sin(v);
z = zeros(1,length(x));

[x,y,z] = pointrot_oiw(x,y,z,omega,inc,w);
end
```

Code 7: planet.m

```
function [x,y,z] = planet(r,n)
%
% planet_sphere(r,[n])
%
% r = radius
% n = number of points ( optional )
%
% author: Anders Edfors Vannevik
%

if (nargin == 1 )
    n = 20; % Default
end

if (nargin == 0 || nargin > 2)
    fprintf(1,'error -> wrong number of arguments\n');
    return;
end

[x1,y1,z1] = sphere(n);

x = r.*x1;
y = r.*y1;
z = r.*z1;

end
```

Code 8: pointrot_oiw.m

```
function [x2,y2,z2] = pointrot_oiw(x1,y1,z1,omega,inc,w)
%
% pointrot_oiw(omega,inc,w)
%
% omega: right ascension angle
% inc : inclination angle
% w : argument of perigee angle
%
% point rotation in the plane ( frame fixed )
%
%  Rz(omega)*Rx(inc)*Rz(w)
%

Mrot = [cos(omega) * cos(w) - sin(omega) * cos(inc) * sin(w) -cos(omega) * sin(w) -
sin(omega) * cos(inc) * cos(w) sin(omega) * sin(inc);
        sin(omega) * cos(w) + cos(omega) * cos(inc) * sin(w) -sin(omega) * sin(w) + cos(omega) *
cos(inc) * cos(w) -cos(omega) * sin(inc);
        sin(inc) * sin(w) sin(inc) * cos(w) cos(inc);];


RotResult = Mrot*[x1;y1;z1];


x2 = RotResult(1,:);
y2 = RotResult(2,:);
z2 = RotResult(3,:);

end
```

Code 9: trueanomaly.m

```
function [x,y,z] = trueanomaly(a,e,omega,inc,w,v)
%
% a: semimajor axis
% e: excentricity
% i: inclination
% w: argument of perigee
% v: true anomaly
%
% 2-body problem orbit determination

r = a*(1-e^2)./(1+e*cos(v));

x = r.*cos(v);
y = r.*sin(v);
z = zeros(1,length(x));

[x,y,z] = pointrot_oiw(x,y,z,omega,inc,w);
end
```

Code 10: xyz2ell.m

```
function [lat,lon,h]=xyz2ell(X,Y,Z,a,e2)
% XYZ2ELL  Converts cartesian coordinates to ellipsoidal.
%   Uses iterative alogithm.  Vectorized.  See also XYZ2ELL2,
%   XYZ2ELL3.
% Version: 2012-02-24
% Usage:   [lat,lon,h]=xyz2ell(X,Y,Z,a,e2)
%          [lat,lon,h]=xyz2ell(X,Y,Z)
% Input:   X \
%          Y  > vectors of cartesian coordinates in CT system (m)
%          Z /
%          a   - ref. ellipsoid major semi-axis (m); default GRS80
%          e2  - ref. ellipsoid eccentricity squared; default GRS80
% Output:  lat - vector of ellipsoidal latitudes (radians)
%          lon - vector of ellipsoidal longitudes (radians)
%          h   - vector of ellipsoidal heights (m)

% Revised 2012-02-24 Corrected starting latitude for iteration.

% Copyright (c) 2012, Michael R. Craymer
% All rights reserved.
% Email: mike@craymer.com

if nargin ~= 3 & nargin ~= 5
  warning('Incorrect number of input arguments');
  return
end
if nargin == 3
  [a,b,e2]=refell('grs80');
end

% Latitude and height convergence criteria
elat=1.e-12;
eht=1.e-5;

% Initial values for iteration
p=sqrt(X.*X+Y.*Y);
lat=atan2(Z,p.*(1-e2));
h=0;
dh=1;
dlat=1;

% Iterate until lat & h converge to elat & eht
while sum(dlat>elat) | sum(dh>eht)
  lat0=lat;
  h0=h;
  v=a./sqrt(1-e2.*sin(lat).*sin(lat));
  h=p./cos(lat)-v;
  lat=atan2(Z, p.*(1-e2.*v./(v+h)));
  dlat=abs(lat-lat0);
  dh=abs(h-h0);
end
lon=atan2(Y,X);
```