Francesca Afruni
AEE 4807 – Space Vehicle Control
Dr. C. Riano-Rios

SVC Project

Introduction

The objective of this project was to design the Attitude Determination and the Control System of a space vehicle simulating the Early Orbit Phase of a Medium Earth Orbit. The space vehicle had to detumble within its first ten orbits, and its angular velocity (ω) had to reduce to less than 0.2 RPMs. Once de-tumbled, the space vehicle had to rotate so that its $-\hat{b}_3$ vector pointed to the center of the Earth for better communications. Finally, the space vehicle also had to align its $\hat{b}_2$ vector with the velocity vector.

The only components available for the project were up to three Magnetorquers (MTQ) with a maximum of 500 mA each, up to five Reaction Wheels (RW), and three sensors: a magnetometer, a sun sensor, and a gyro.

Space Vehicle Design Overview (Initial Conditions and Properties)

To develop a simulator for both attitude and orbital dynamics, MATLAB and Simulink were used. Figure 1 shows the complete Simulink model designed for the space vehicle. The model features many different MATLAB functions that represent the models for the actuators, pointing, attitude determination, and more.

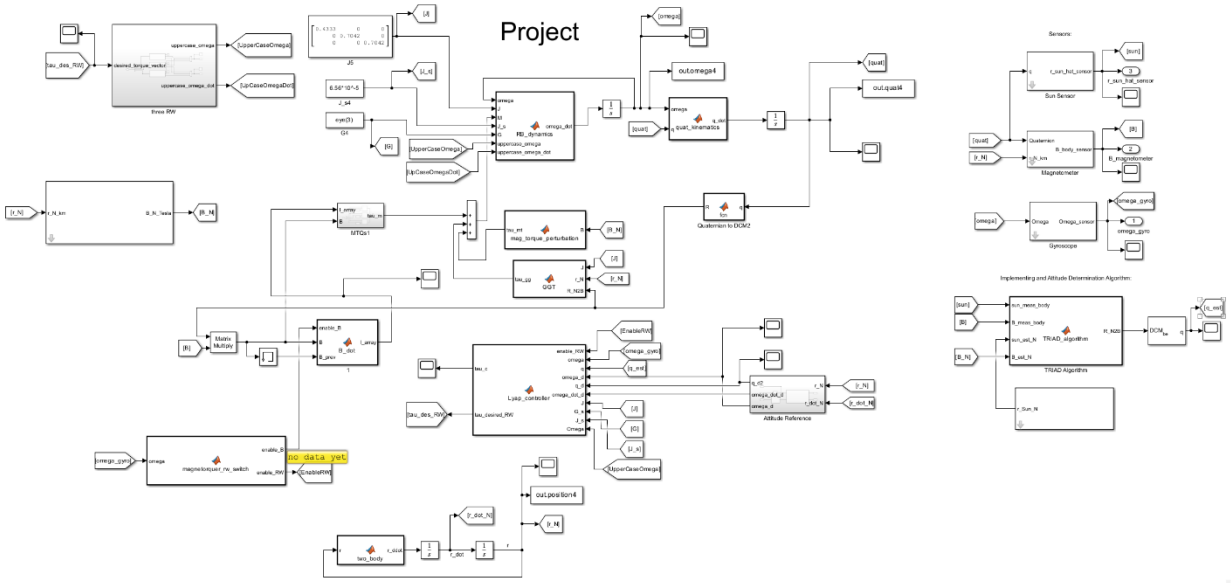

Figure 1: Complete Simulink Model of the Space Vehicle

The space vehicle had to conform to the initial conditions and different system properties given. Firstly, the space vehicle used two-body dynamics to simulate its orbit, and the values given for the orbit state at injection were the following (position, velocity, omega, inertia sensor, respectively):

$$^{\mathcal{N}}\boldsymbol{r}_0 = [750.6,\ 6874.3,\ -1925.1]^T\ \ km$$
$$^{\mathcal{N}}\dot{\boldsymbol{r}}_0 = [-4.9379, -0.9985, -5.4909]^T\ \ km/s$$
$$\boldsymbol{\omega_0} = [\mathbf{0.093}, \mathbf{-0.034},\ \mathbf{-0.044}]\ \ \boldsymbol{rad/s}$$

$$J = \begin{bmatrix} 0.4333 & 0 & 0 \\ 0 & 0.7042 & 0 \\ 0 & 0 & 0.7042 \end{bmatrix}\ \ kg/m^2$$

These values were incorporated into the function blocks. The position and velocity initial conditions were added to the two-body dynamics MATLAB function (Figure 2) in the integrator blocks (Figures 3 and 4). The scope for the position vector can be found at the end of the document (Figure 53).

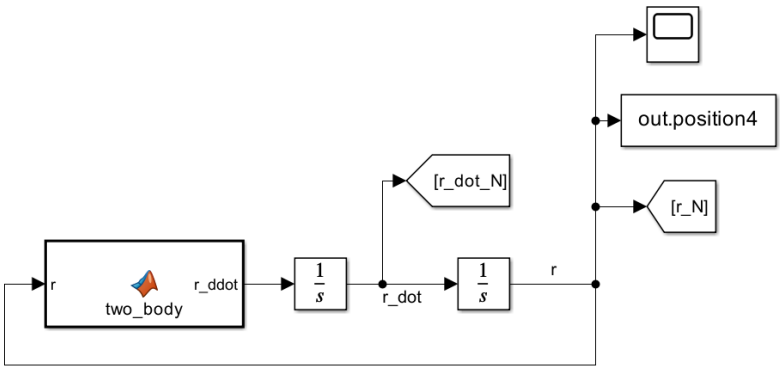

Figure 2: Two-body MATLAB function

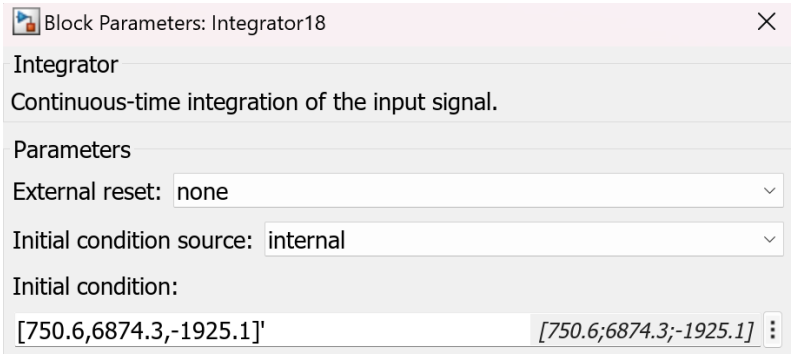

Figure 3: Position Initial Conditions

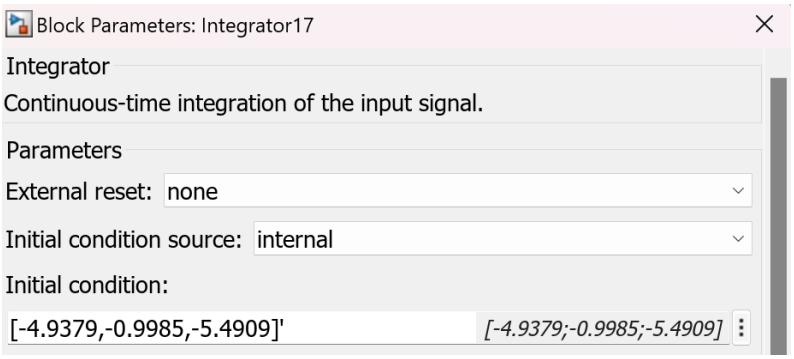

Figure 4: Velocity Initial Conditions

The code for the two-body dynamics block can be seen in the following figure.

```
function r_ddot = two_body(r)
    GMe = 398600.4405;
    r_ddot = -GMe/norm(r)^3*r;
end
```

Figure 5: Two-body function code

The initial condition for the space vehicle's Omega (Figure 8) was implemented in the integrator that connects to the Rigid Body MATLAB function (RB_dynamics) (Figure 6), which also shows the input inertia tensor J, the moment of inertia J_s, the RW configuration matrix G, the torques M, and the angular velocity and acceleration of the RW (Ω). The reaction wheels block will be further explained in the next sections.

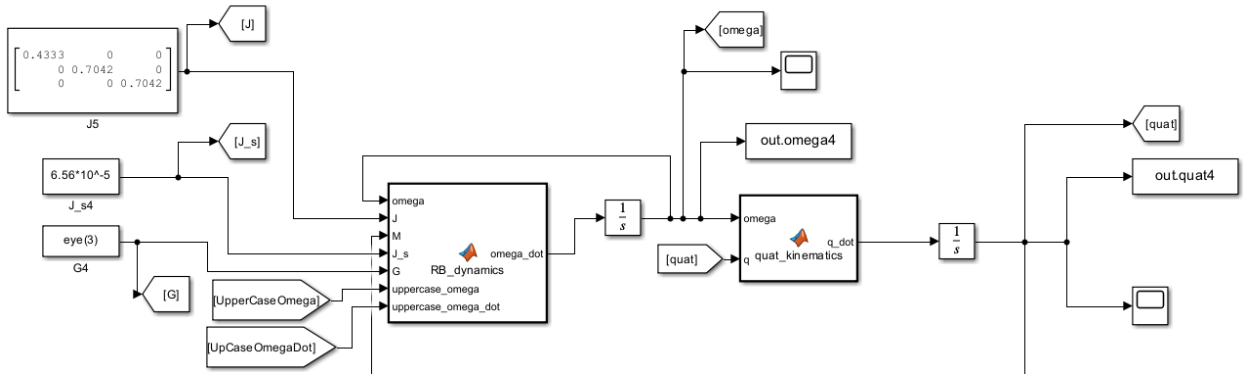

Figure 6: Close-up of the rigid body dynamics function and quaternion kinematics function

```
function omega_dot = RB_dynamics(omega, J, M, J_s, G, ...
    uppercase_omega, uppercase_omega_dot)

omega_dot = inv(J)*(M - (J_s*G*uppercase_omega_dot) - ...
    cross(omega, (J*omega + J_s*G*uppercase_omega)));

end
```

Figure 7: RB Dynamics function code

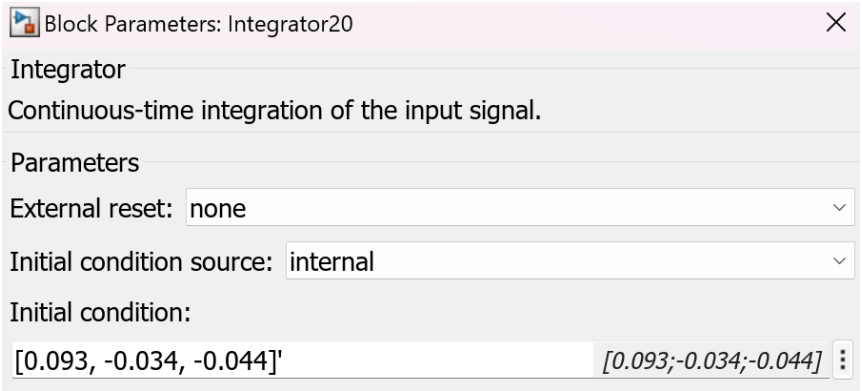

Figure 8: Omega Initial Conditions

The rigid body dynamics block is connected to the quaternion kinematics block (quat_kinematics), which code can be seen in the following figure.

```
function q_dot = quat_kinematics(omega, q)

q0 = q(1);
qv = q(2:4);

qv_skew = [0  -qv(3)   qv(2);
           qv(3)    0    -qv(1);
            -qv(2)   qv(1)    0];
q0dot = -0.5*qv'*omega;

qvdot = 0.5*(qv_skew + q0*eye(3))*omega;

q_dot = [q0dot; qvdot];
```

Figure 9: Quaternion kinematics function code

The RB dynamics and quaternion kinematics functions output ideal omegas and quaternions, which space vehicles don't really have in real-life scenarios. However, in this case, they are used and implemented into the sensors that will make them "more realistic" and then feed them back into attitude determination algorithms, controllers, etc. (which will be further explained in future sections).

The last initial condition the space vehicle had to adhere to was that the space vehicle's residual magnetic dipole had to be 0.03 Am^2. This condition was implemented into the detumbling (B_dot) MATLAB function (Figure 10).

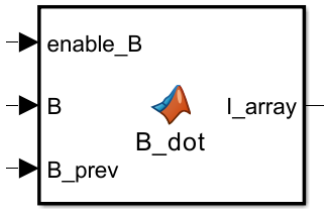

Figure 10: Detumbling function block

```
function I_array = B_dot(enable_B, B, B_prev)
N = 500; %"randomly" chosen
A = 1.2*10^-4; %calculated
sim_step_size = 0.2; %given
D_max = 0.03; %given

% D = NIA if N = 500, I = 500e-3, A = 1.2e-4 m2

if enable_B == 1
    B_hat = B/norm(B);
    B_hat_prev = B_prev/norm(B_prev);
    B_hat_dot = (B_hat - B_hat_prev)/sim_step_size;
    D_desired = -D_max * sign(B_hat_dot);
    I_array = D_desired/(N*A);
else
    I_array = [0;0;0];
end

end
```

Figure 11: B_dot function code

The equation to calculate the magnetic dipole is the following:

$$D = NIA$$

Where N is the number of loops of the coil, I is the current required (Figure 52), and A is the area of the coil. The magnetic dipole and the amount of current available were given for this project (0.03 Am^2 and $500e^{-3}$ A, respectively); therefore, the above equation was used to calculate N and A. The number of loops was "randomly" chosen as 500, and the equation was adjusted to solve for the area, which was calculated to be $1.2e^{-4}$ m^2.

Implementing a Detumbling Algorithm

The B_dot algorithm is essential to ensure the detumbling of the space vehicle. As mentioned in the introduction, one of the space vehicle's mission objectives was to detumble within ten orbits and with an angular velocity of less than 0.2 RPM. Because of the limits imposed on the number of system components available, the detumbling of this space vehicle was achieved using both magnetorquers and reaction wheels. This meant that the system had to be able to switch between the two to reach detumbling. This explains the reason why the code in Figure 11 shows an "enable/disable" feature. Figure 12 shows how the detumbling function block is connected to the magnetorquers and to the reaction wheels through another function block called "magnetorquer_rw_switch," which, as can be seen in Figure 13, holds the code that using logic, allows the switch between MTQ and RW once a certain Omega limit (0.2 RPM) has been reached.

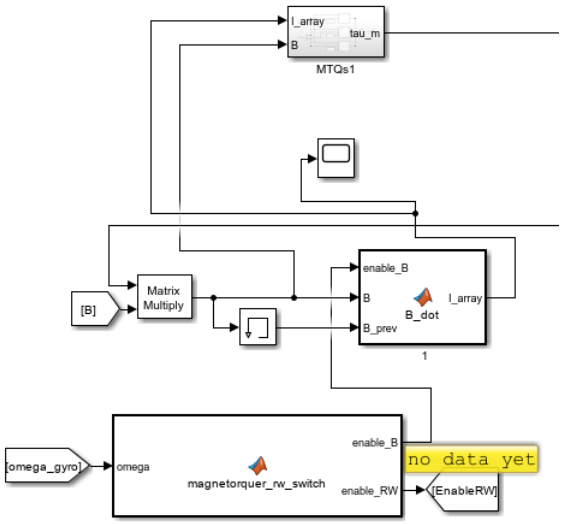

Figure 12: Connections between detumbling functions, magnetorquers, and reaction wheels

```
function [enable_B, enable_RW] = magnetorquer_rw_switch(omega)
    % Persistent variable to track the state
    persistent reaction_wheels_enabled;

    % Initialize the persistent variable on the first run
    if isempty(reaction_wheels_enabled)
        reaction_wheels_enabled = false;
    end

    % Convert angular velocity to RPM
    omega_RAD = norm(omega) * 9.55; % convert to RPM
    omega_limit = 0.2; % RPM threshold

    % State logic: Once reaction wheels are enabled, magnetorquers remain disabled
    if reaction_wheels_enabled
        enable_B = 0; % Magnetorquers are permanently disabled
        enable_RW = 1; % Reaction wheels remain enabled
    else
        if omega_RAD < omega_limit
            % Enable reaction wheels and disable magnetorquers
            enable_B = 0;
            enable_RW = 1;
            reaction_wheels_enabled = true; % Lock the state
        else
            % Enable magnetorquers and disable reaction wheels
            enable_B = 1;
            enable_RW = 0;
        end
    end
end
```

Figure 13: "magnetorquer_rw_switch" code

Another thing to be noted is that the Omega that comes into the MTQ/RW switch function comes from the gyro sensor, which will be explained further in the sensors section. Also, the “enable RW” Goto block connects the switch function to the RW block function, which will also be explained in its own section. To see the full connection, refer to Figure 1.

Including Attitude Perturbations in the simulation

Another report requirement was to implement the two most relevant attitude perturbations in order to make the simulation as realistic as possible. The decision was mainly made referring to Figure 14, which shows the main perturbations that affect space vehicles. As can be observed in the figure, in MEO, the two most influential perturbations are the Gravity Gradient and the Magnetic Torque.

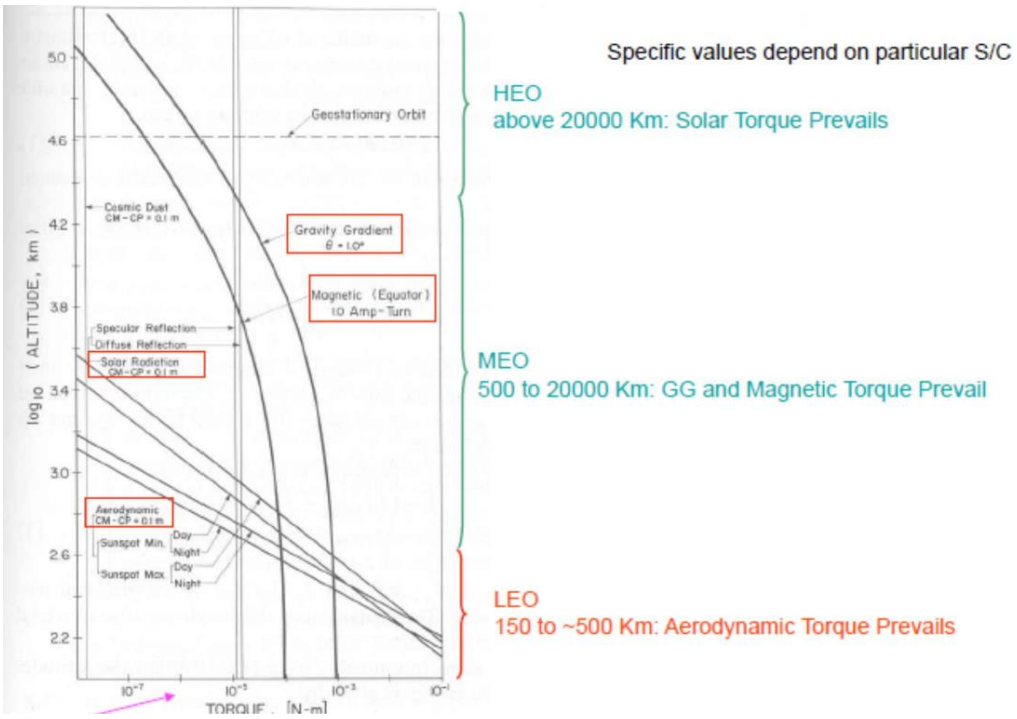

Figure 14: Perturbations Graph (Altitude vs Torque) [1]

The Gravity Gradient Torque refers to the gradient torque that is formed due to different space vehicle parts experiencing different gravitational forces based on their position relative to Earth. The mathematical expression for the gravity gradient torque is as follows:

$$\tau_{gg} = \frac{3GM_E}{\|r\|^5} r \times Jr$$

Where G is the universal gravitational constant, r is the space vehicle's position vector, and J is the inertia tensor. This equation was then translated into the GGT MATLAB function block (Figure 15), as can be seen in Figure 16 below.

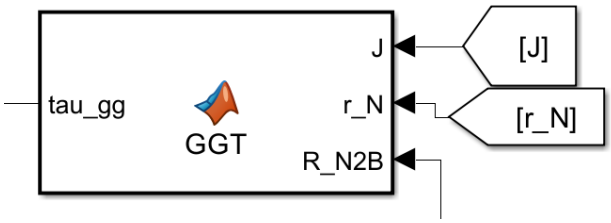

Figure 15: GGT Block

```
function tau_gg = GGT(J, r_N, R_N2B)
    GMe = 398600.4405 * 1000^3;
    r_B = R_N2B * r_N * 1000;
    tau_gg = 3*GMe/norm(r_B)^5 * cross(r_B,J*r_B);
end
```

Figure 16: GGT function code

The Magnetic Torque is due to the residual magnetic dipole and can be expressed as follows:

$$\tau_m = D \times B$$

Where D is the magnetic dipole vector in Am and B is the Earth's magnetic field vector in Tesla. Figure 17 shows the magnetic torque perturbation block, which gets the magnetic field value from the Earth's Magnetic Field block shown in Figure 19, provided by Dr. Riano-Rios, which gets the position vector from the two-body function previously shown. Figure 18 shows how the equation has been implemented as code.

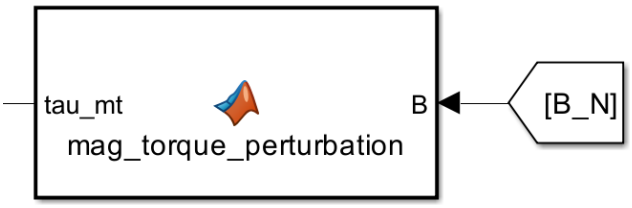

Figure 17: Magnetic torque perturbation function block

```
function tau_mt = mag_torque_perturbation(B)

D = [0.03; 0; 0]; %Am2

tau_mt = cross(D,B);
```

Figure 18: Magnetic torque function code

Figure 19: Earth's Magnetic Field Block

Figure 20 shows how the two attitude perturbation blocks connect with the magnetorquers block and then feed back into the rigid body dynamics function block as the external torques. Refer to Figure 1 to better understand how the perturbations connect to the RB_dynamics block.

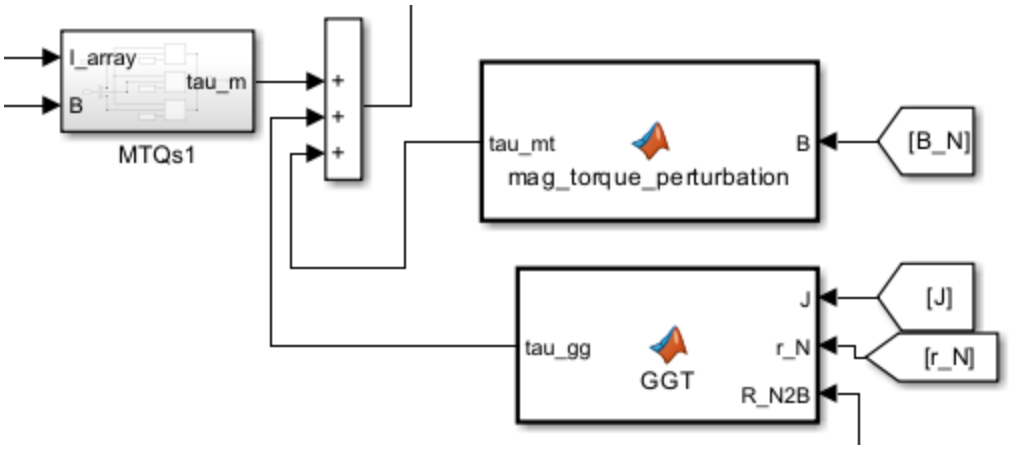

Figure 20: Connection between perturbations

Actuator Model 1: Magnetorquers

Magnetorquers are devices that, by trying to align their field with the Earth's magnetic field, create a torque on the spacecraft. They were fundamental in this project to achieve the detumbling of the space vehicle. As mentioned in the "Implementing a Detumbling Algorithm" section, the reaction wheels and the magnetorquers used a switch function that enabled one of the two when a certain omega was reached. Figure 20 shows the magnetorquers block on the top left, and Figure 12 shows how the B_dot function feeds the magnetic field and the current values into it. Figure 21 shows what is inside the MTQ subsystem, and it can be noted how there are three different magnetorquers that output three different torques, one for each axis. Figure 22 shows just how each of the torques was calculated, which was by using the two following equations:

$$D = NIA$$

$$\tau_m = D \times B$$

As mentioned before, N (number of loops) = 500, I (current) = $500e^{-3}$ A, A (area) = $1.2e^{-4}$ m^2, and D = 0.03 Am^2.

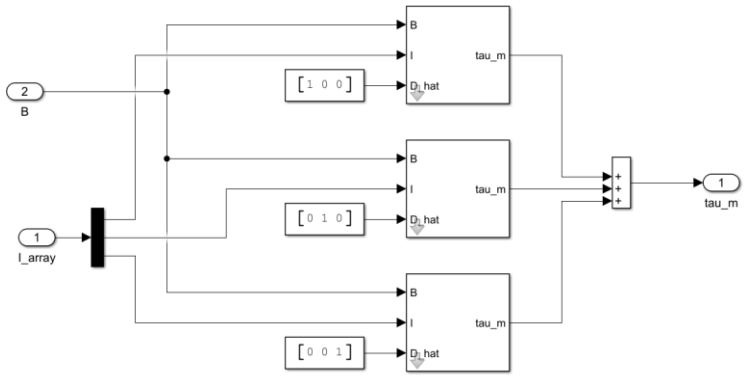

Figure 21: Inside the MTQ subsystem

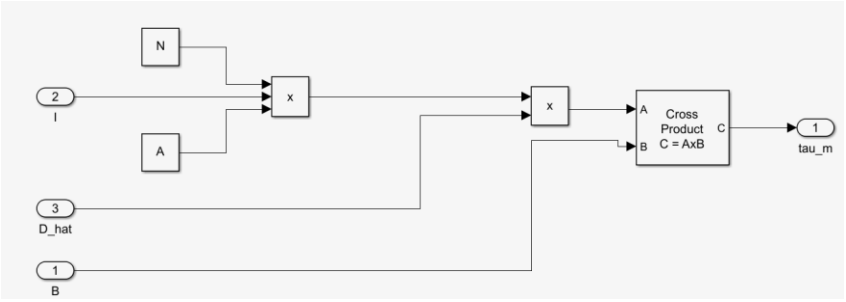

Figure 22: Inside each magnetorquer block

Actuator Model 2: Reaction Wheels

Reaction Wheels are actuators that are used to align the body of the space vehicle to some desired attitude and to reject external disturbances. This project implemented three reaction wheels, one for each axis. Figure 23 shows the RW subsystem, with as input the desired torque, which comes from the Lyap controller (that will be explained further in the next sections), and the RW velocity and acceleration as outputs, which are then implemented into the rigid body dynamics block (see Figure 1).

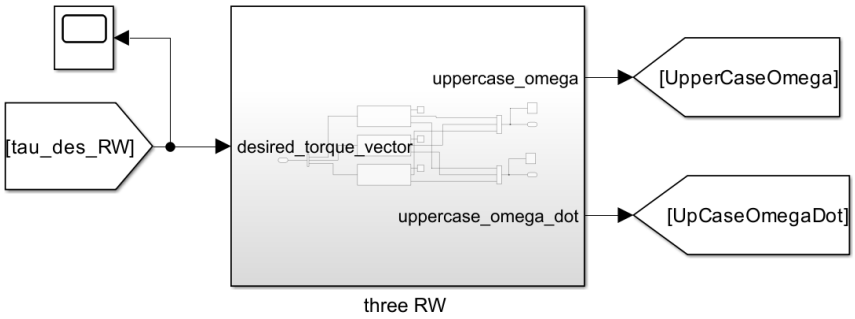

Figure 23: Reaction Wheels subsystem

The general form for the equations of motions for a space vehicle with three RW is as follows:

$$M = J\dot{\omega} + J_{RW}G\dot{\Omega} + \omega \times (J\omega + J_{RW}G\Omega)$$

Where, J_{RW} is the moment of inertia of the RW, and G is the configuration matrix.

G for a configuration of three reaction wheels is as follows:

$$G_{3\,RW} = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

The moment of inertia of the RW was calculated, using the assumption that the RW was a perfect disk, using the following equation:

$$I = \frac{1}{2}mr^2$$

Where m is the mass of the RW, and r is the radius of the disk. Research on reaction wheels sold on the market was conducted to obtain these values. AAC Clyde Space's RW400 [2] produces high-performance reaction wheels for 12U CubeSats, and their product was chosen. Tables 1 and 2 show the specifications of the aforementioned reaction wheels.

Table 1: AAC's RW400 Specs 1 [2]

Performance		
Total momentum storage	+/- 15; +/- 30; +/- 50	mN.m.s
Maximum torque	+/- 8	mN.m
Maximum rotation speed	5000	rpm
Control accuracy	+/- 1	% of target rpm

Table 2: AAC's RW400 Specs 2 [2]

Dimensions		
Outer dimensions	50 x 50 x 27	mm
Mass	197 / 210 / 375	g

The previous moment of inertia equation was applied to obtain the moment of inertia of the reaction wheels, which was also used as the moment of inertia for the rigid body dynamics block (J_s).

$$\boldsymbol{I_{RW}} = \frac{1}{2} * (0.210\ kg) * (0.025\ m)^2 = \boldsymbol{6.56e^{-5}\ kg * m^2}$$

Figure 24 shows the inside of the RW subsystem, which consists of three RWs where velocity and acceleration are output.

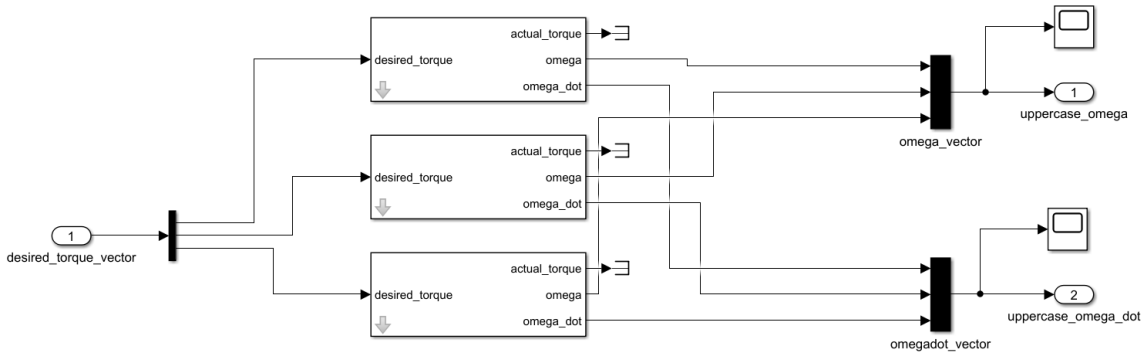

Figure 24: Inside the RW subsystem

Figure 25 shows the inside of one of the RW blocks, which is simply a block form of the general equation of motion equation for torque mentioned at the beginning of this section. Figure 25 shows the parameters of the mask made for the RW subsystem, which were all filled in using Tables 1 and 2.

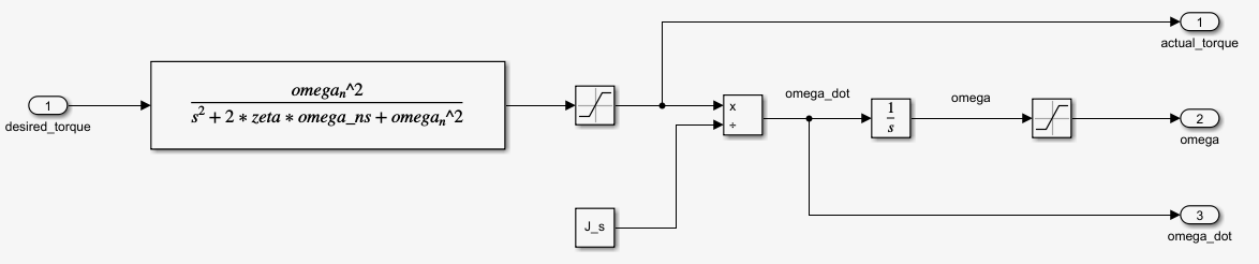

Figure 25: Inside each RW block

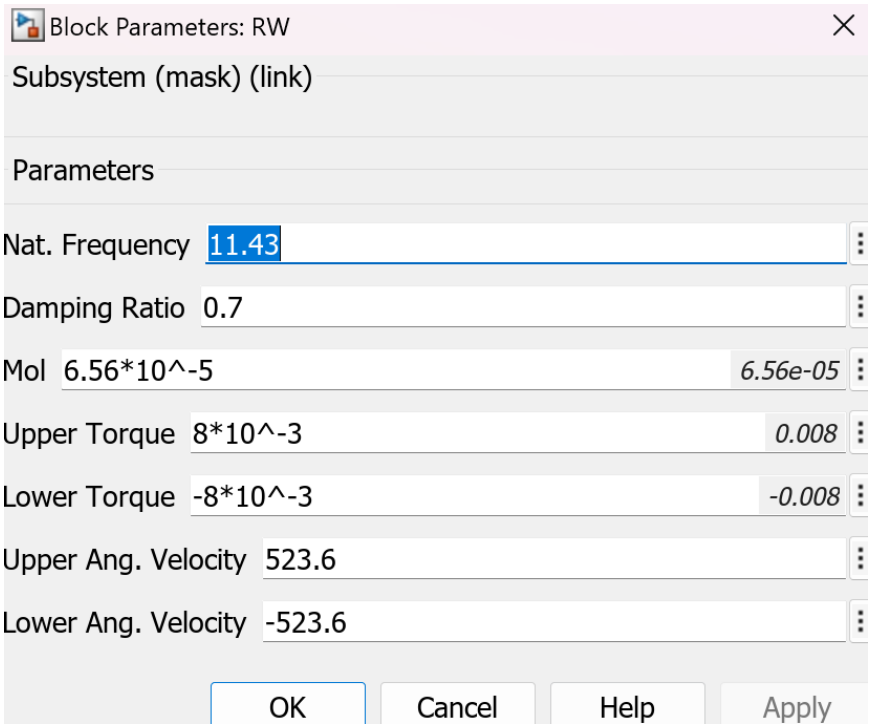

Figure 26: RW parameters

Implementing the controller and the attitude reference

A Lyapunov-based controller is very useful in space vehicles because it ensures global stability, robustness to uncertainties, and precise trajectory or attitude control in nonlinear dynamic environments. Unlike a proportional-derivative (PD) controller, which is typically designed for linear systems and may fail to guarantee stability in the presence of disturbances or highly nonlinear dynamics, a Lyapunov controller is derived using Lyapunov stability theory, providing a mathematical guarantee of convergence to the desired state. This makes it more reliable and efficient for handling complex space operations, such as orbital maneuvers and attitude adjustments, especially under unpredictable conditions.

Figure 27 shows the function block created for the controller and how it connects to the attitude reference subsystem, which will be explained further in the next section. The inputs of the controller are J, Js, G, Ω, but also the Omega coming from the gyro and the quaternion estimated from the TRIAD algorithm, which will be explained in future sections. This block is also very important because it outputs the torque desired (Figure 54) for the reaction wheels, which connects directly to the RW subsystem shown previously.

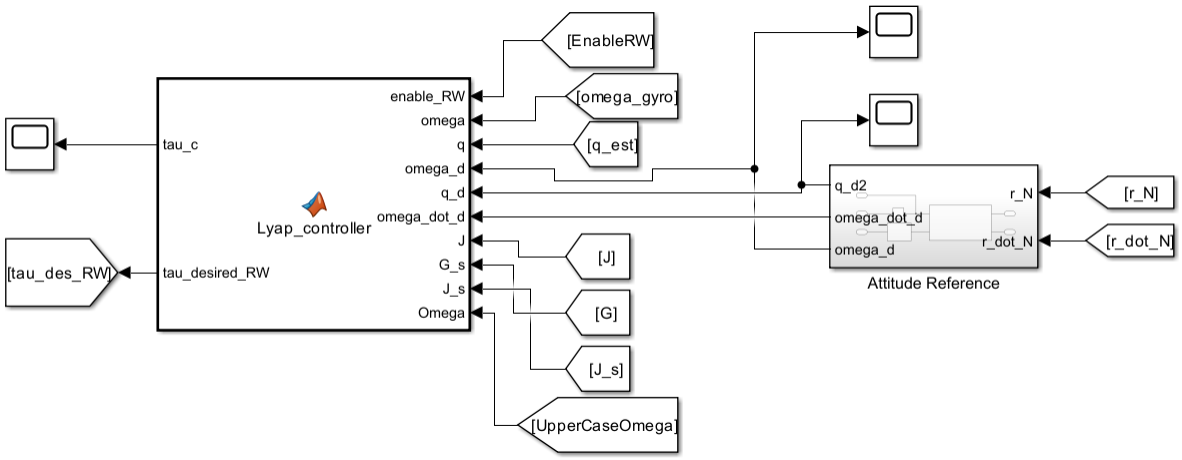

Figure 27: Attitude reference and Lyapunov controller

Figure 28 shows the code utilized inside the Lyapunov controller. It can be noticed that this code presents the enable function as well. This function is fundamental because it sends the desired torque to the reaction wheels when the magnetorquers are turned off and the RWs are turned on by the switch function (Figure 12).

The Lyapunov controller also needs the desired quaternion, angular velocity, and acceleration. These values are produced by the attitude reference subsystem or, more precisely, by the Nadir pointing function inside (Figure 29). This function is essential to achieve the $-\hat{b}_3$ vector pointing to the center of the earth and the aligning of the $\hat{b}_2$ vector with the velocity vector. Figure 30 shows how the code was modified to achieve these results graphically shown by the animation in Figures 31 and 32.

```
function [tau_c, tau_desired_RW] = Lyap_controller(enable_RW, omega, q, omega_d, q_d,...
                                                   omega_dot_d, J, G_s, J_s, Omega)
if enable_RW == 1
    q0 = q(1);
    qv = q(2:4);
    q0d = q_d(1);
    qvd = q_d(2:4);
    qv_skew = [0    -qv(3)  qv(2);...
               qv(3)   0   -qv(1);...
               -qv(2)  qv(1)   0];

    e0 = q0 * q0d + qv' * qvd;
    ev = q0d * qv - q0 * qvd + qv_skew * qvd;

    ev_skew = [0    -ev(3)  ev(2);...
               ev(3)   0   -ev(1);...
               -ev(2)  ev(1)   0];

    R_tilde = (e0^2 - ev'*ev)*eye(3) + 2*(ev*ev') - 2*e0 * ev_skew;

    omega_tilde = omega - R_tilde * omega_d;

    K = 1e-2 * eye(3);
    alpha = 1e-2 * eye(3);
    beta = 1e-3;

    r = omega_tilde + alpha*ev;

    omega_skew = [0     -omega(3)   omega(2);...
                  omega(3)    0   -omega(1);...
                  -omega(2)   omega(1)    0];

    tau_c = cross(omega, (J*omega + J_s*G_s*Omega)) + ...
        J*(-omega_skew*R_tilde)*omega_d + ...
        J*R_tilde*omega_dot_d - ...
        J*alpha*(1/2*(ev_skew+e0*eye(3))*omega_tilde) - K*r - beta*ev;
else
    tau_c = [0;0;0];
end
    tau_desired_RW = -pinv(G_s)*tau_c;
end
```

Figure 28: Lyap controller code

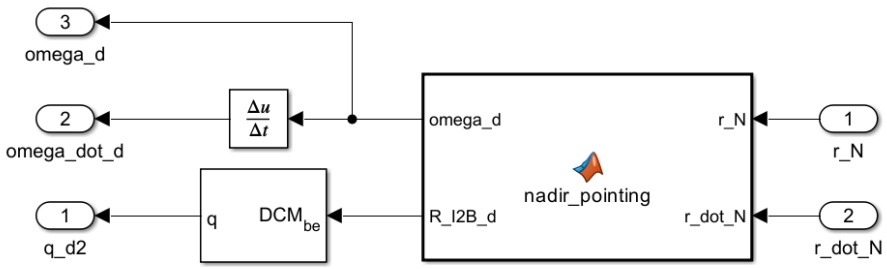

Figure 29: Attitude Reference

```
function [omega_d, R_I2B_d] = nadir_pointing(r_N, r_dot_N)
    GMe = 398600.4405;

    n = sqrt(GMe/norm(r_N)^3);

    o3_hat = r_N / norm(r_N);
    o2_hat = r_dot_N/norm(r_dot_N);
    o1_hat = cross(o2_hat, o3_hat);

    R_I2B_d = [o1_hat, o2_hat, o3_hat]';

    omega_d = [-n;0;0];
end
```

Figure 30: Nadir pointing function code

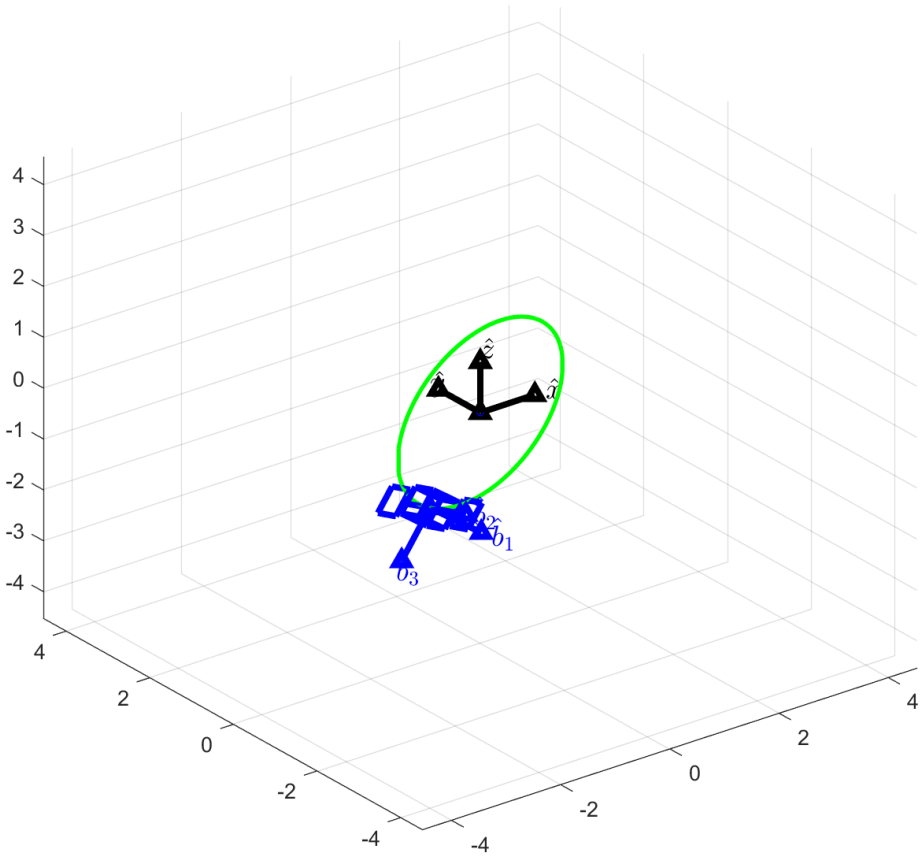

Figure 31: SV animation 1

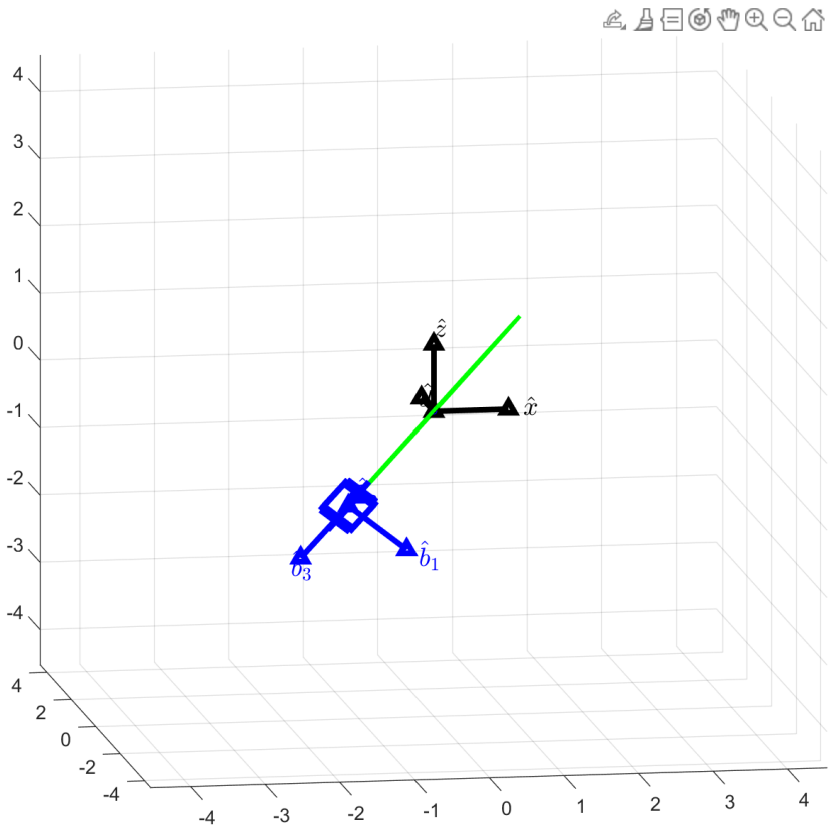

Figure 32: SV animation 2

Sensors and Attitude Determination Algorithm

It was also allowed to utilize and implement sensors in the space vehicle's Simulink to achieve the objectives of the project. Figure 33 shows the three sensors implemented to aid in attitude determination and control. The Sun Sensor (Figures 38 and 39) measures the unit vector direction of the Sun relative to the spacecraft's body frame, providing critical orientation data. The Magnetometer (Figures 36 and 37) captures the magnetic field vector in the spacecraft's body frame, utilizing quaternion transformations and position vectors to align it with Earth's magnetic field. It also provides the value of the Earth's magnetic field utilized in the detumbling function block. Lastly, the Gyroscope (Figures 40 and 41) measures the angular velocity of the spacecraft, offering precise data on rotational motion and providing the very important "new" omega, which is then used in the controller and for the actuators. The outputs from these sensors are also implemented into the TRIAD algorithm (Figure 34), which is designed to estimate a spacecraft's orientation in space based on observations of two non-collinear vectors, typically derived from onboard sensors. Figure 35 shows the algorithm that makes this estimation possible. The quaternion estimated output by the TRIAD is then used in the Lyap controller.

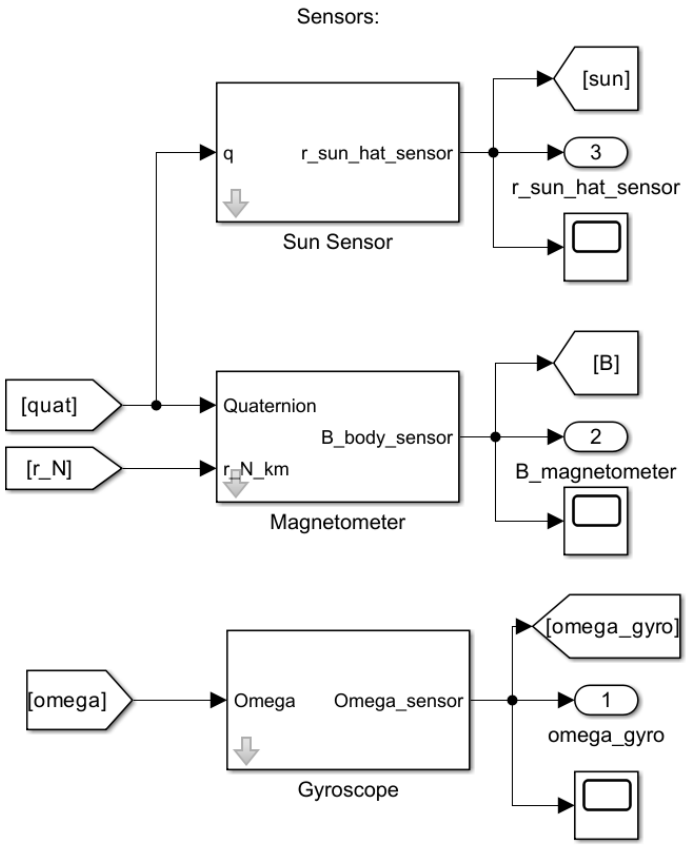

Figure 33: Sensors

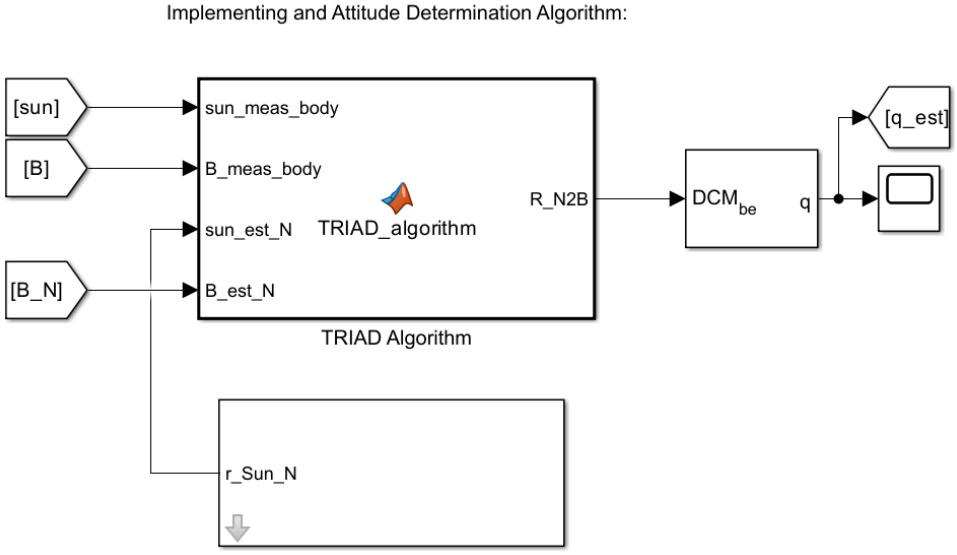

Figure 34: TRIAD Algorithm

```
function R_N2B = TRIAD_algorithm(sun_meas_body, B_meas_body, sun_est_N, B_est_N)
    t1_body = sun_meas_body / norm(sun_meas_body);
    t2_body = cross(t1_body, B_meas_body);
    t2_body = t2_body / norm(t2_body);
    t3_body = cross(t1_body, t2_body);

    t1_inertial = sun_est_N / norm(sun_est_N);
    t2_inertial = cross(t1_inertial, B_est_N);
    t2_inertial = t2_inertial / norm(t2_inertial);
    t3_inertial = cross(t1_inertial, t2_inertial);

    DCM_body = [t1_body, t2_body, t3_body];
    DCM_inertial = [t1_inertial, t2_inertial, t3_inertial];

    R_N2B = DCM_body * DCM_inertial';
end
```

Figure 35: TRIAD Algorithm code

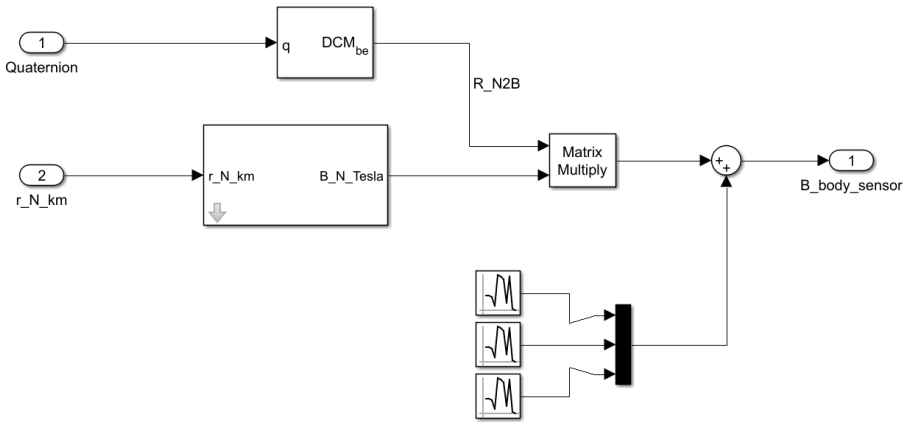

Figure 36: Inside the Magnetometer sensor

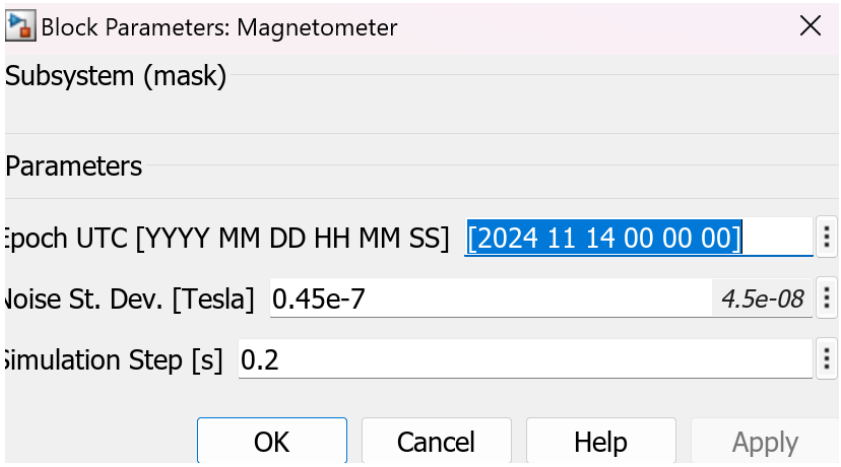

Figure 37: Magnetometer mask values

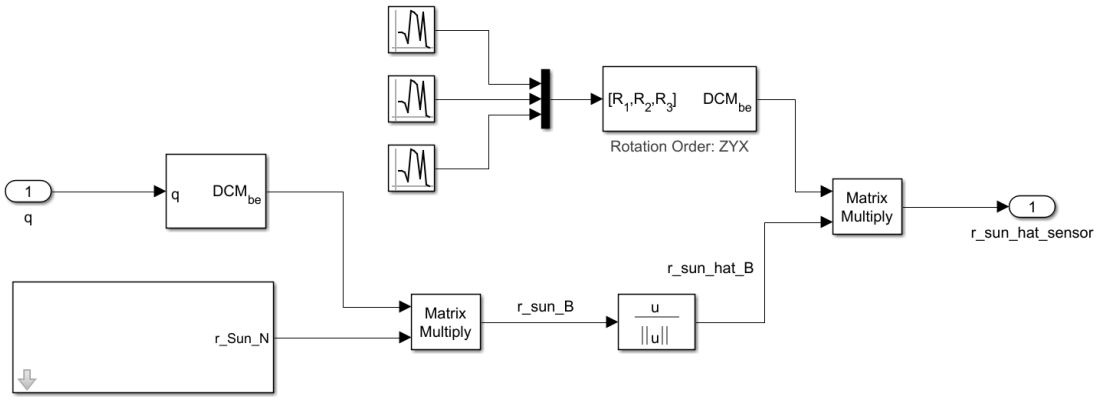

Figure 38: Inside Sun sensor

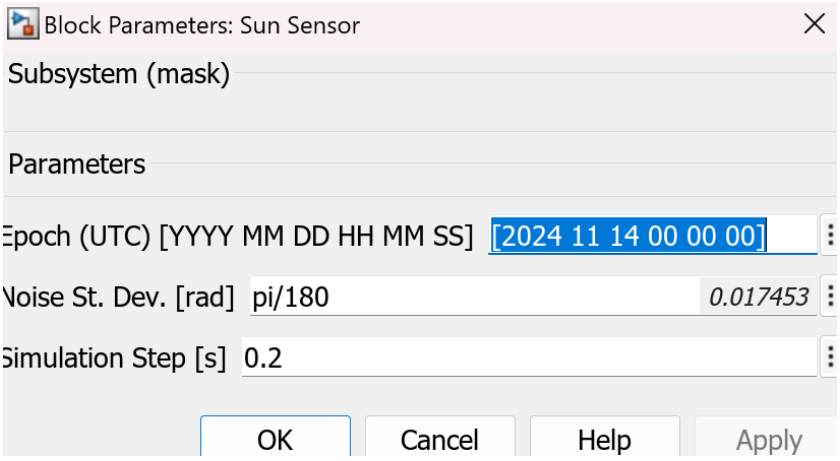

Figure 39: Sun sensor mask values

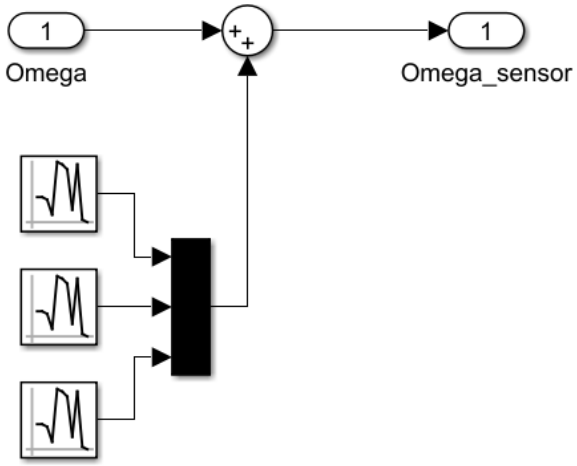

Figure 40: Inside Gyroscope sensor

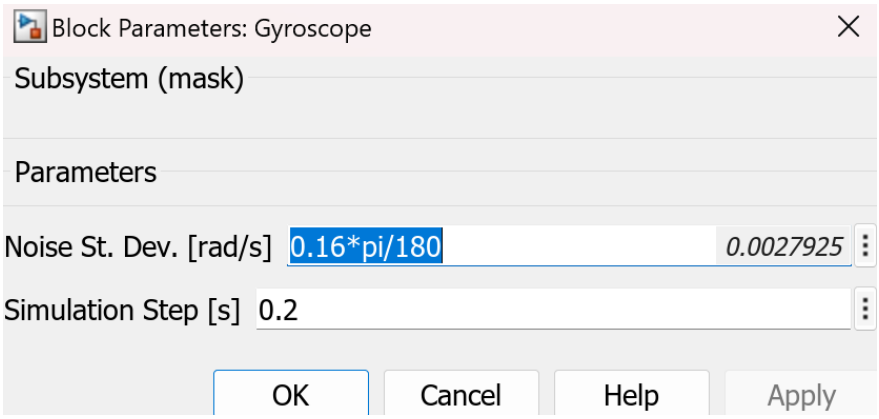

Figure 41: Gyroscope mask values

Figures 42 to 44 show the scopes obtained from the sensors. It can be noted how they are affected by a lot of noise and how, in the end, they all settle.

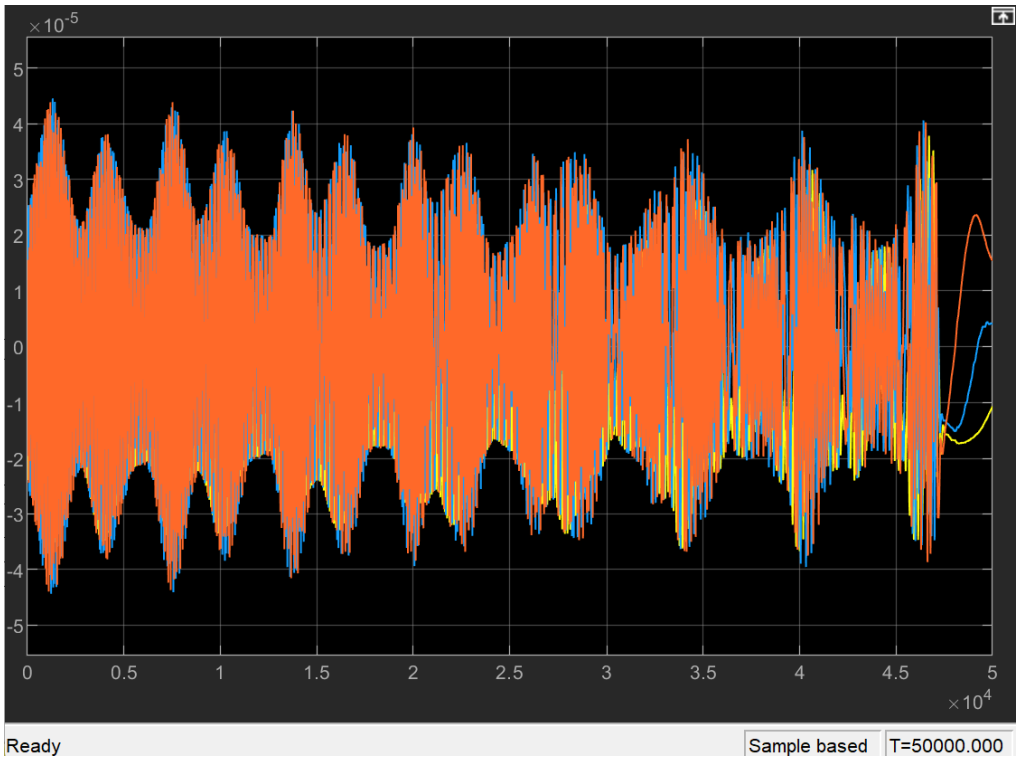

Figure 42: Scope of the Magnetometer

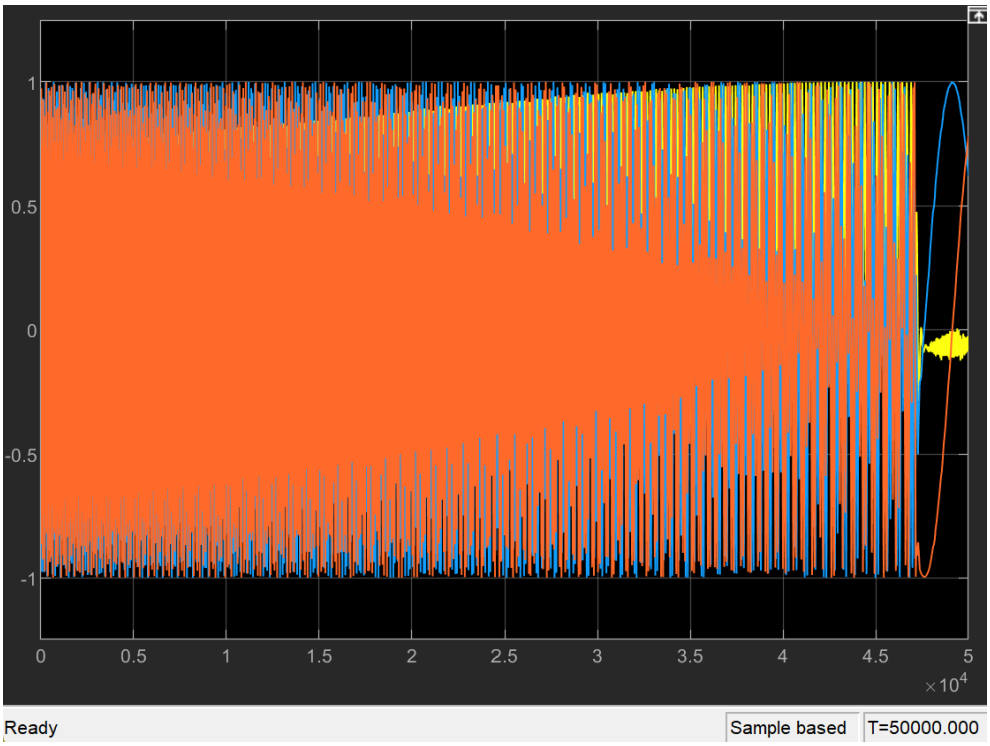

Figure 43: Scope of the sun sensor

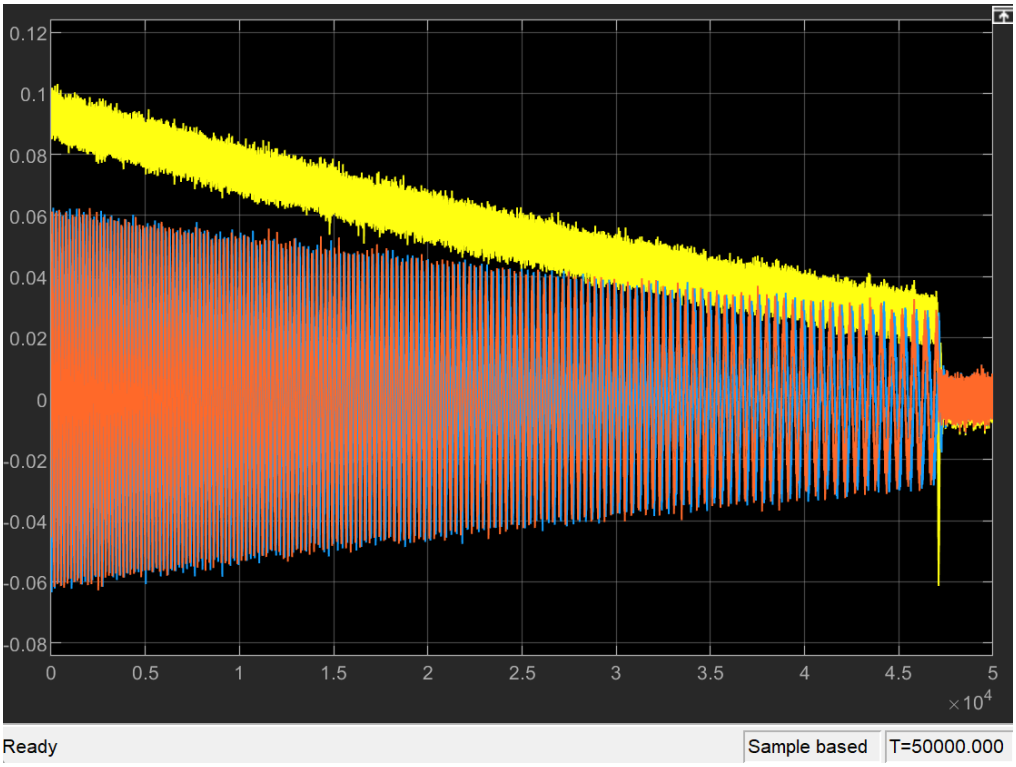

Figure 44: Scope of the gyroscope (omega gyro)

Results & Conclusions

Figures 45 and 46 show the scopes from omega of the rigid body dynamics and omega desired from the attitude reference subsystem, respectively. By comparing Figures 45 and 44, it's obvious the two plots portray the same overall behavior; however, as expected, the omega produced by the gyroscope presents much more noise than the perfect and ideal omega from the rigid body dynamics. The two graphs do converge to zero, confirming that the detumbling was successful.

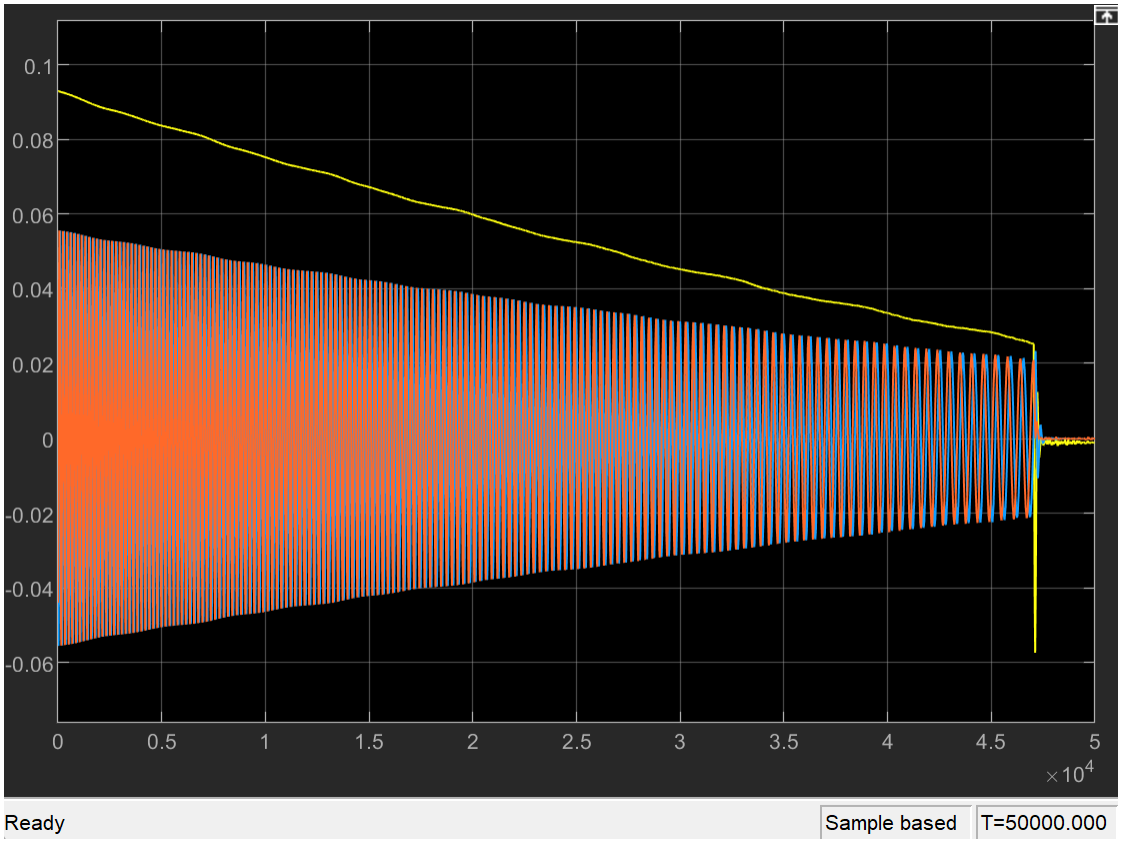

Figure 45: Omega RB dynamics

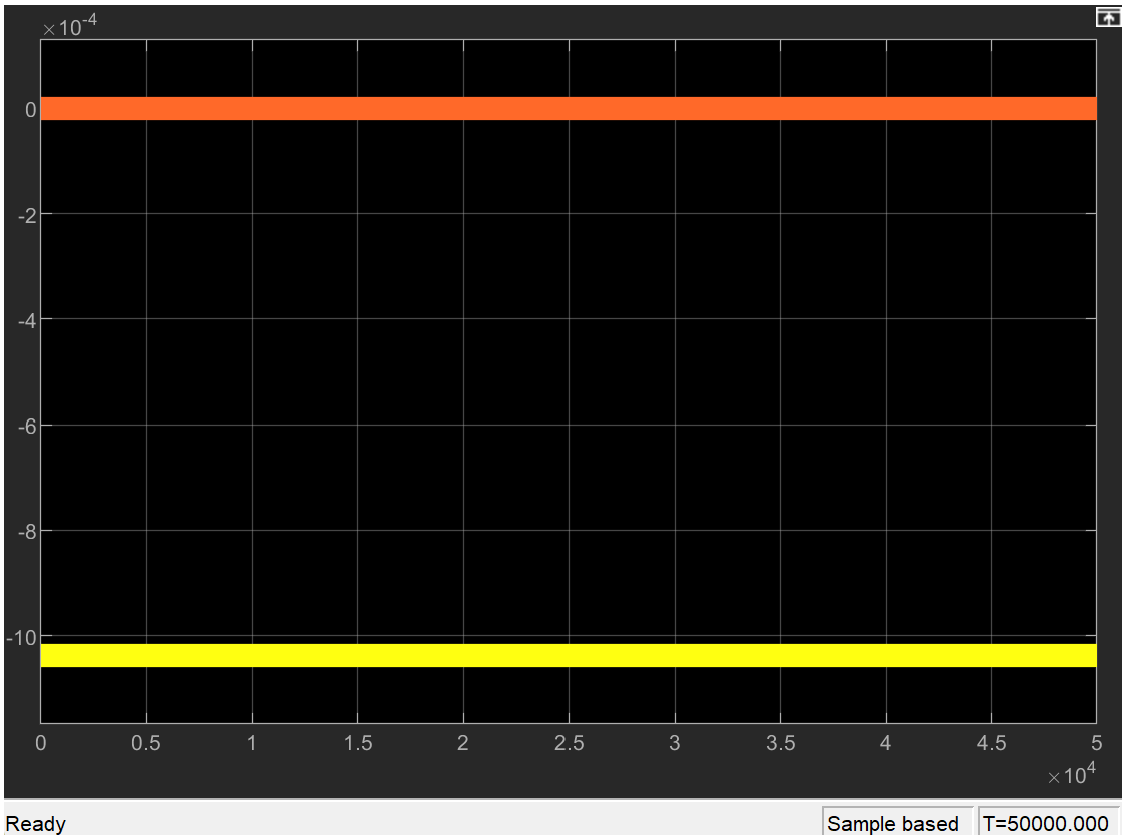

Figure 46: Omega desired

Figures 47 to 51 show the rigid body dynamics quaternion, the desired quaternion, and the quaternion obtained from the TRIAD algorithm. It can be noticed how Figures 48 and 51 show the same overall behavior, confirming the correctness of the model. Like the previous case for omega, the quaternion from the TRIAD (Figure 51) shows more noise than the ideal rigid body dynamics one (Figure 48). However, there is a major difference between the two, and that is dictated by the sign switch of the quaternion. In real life, this big jump represents a problem and would need to be fixed with logic; however, in this project, it hasn't been addressed.

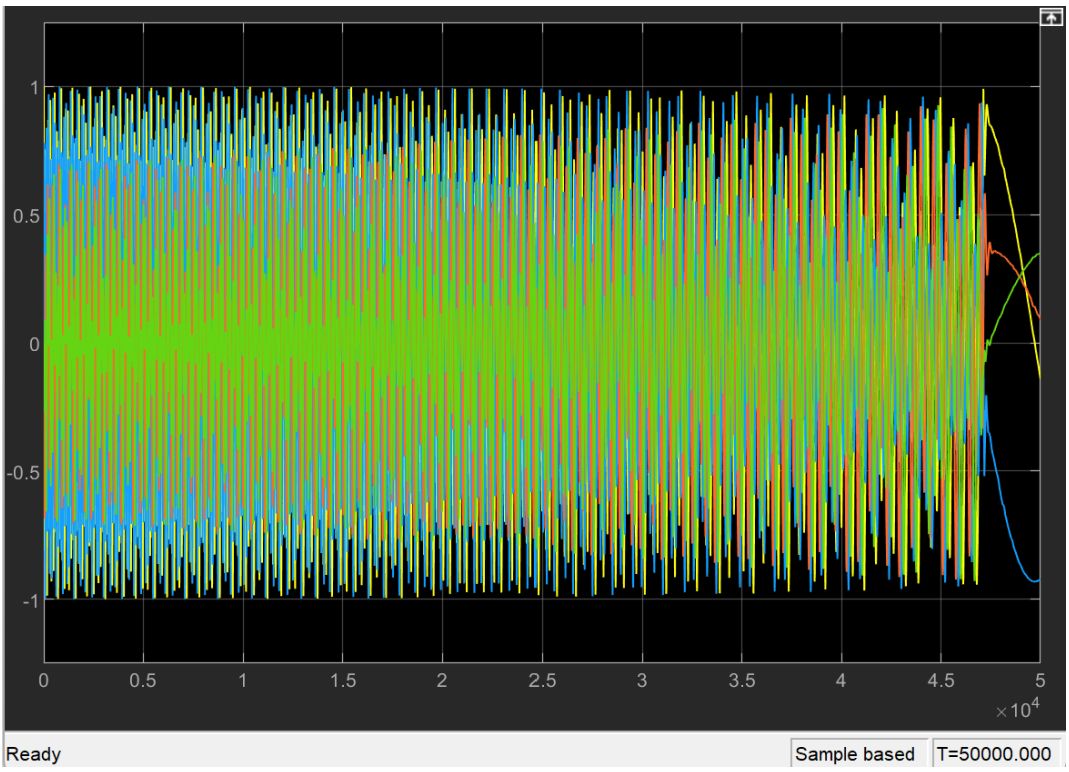

Figure 47: Quaternion kinematics

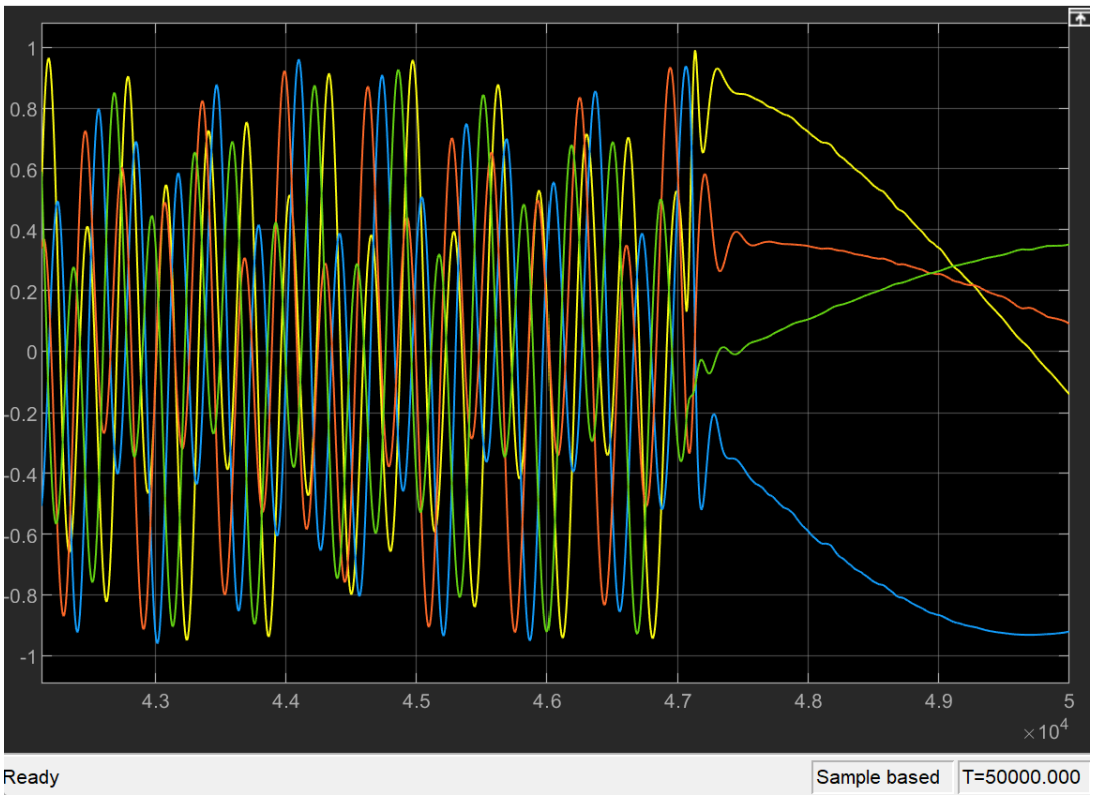

Figure 48: Quaternion kinematics close up

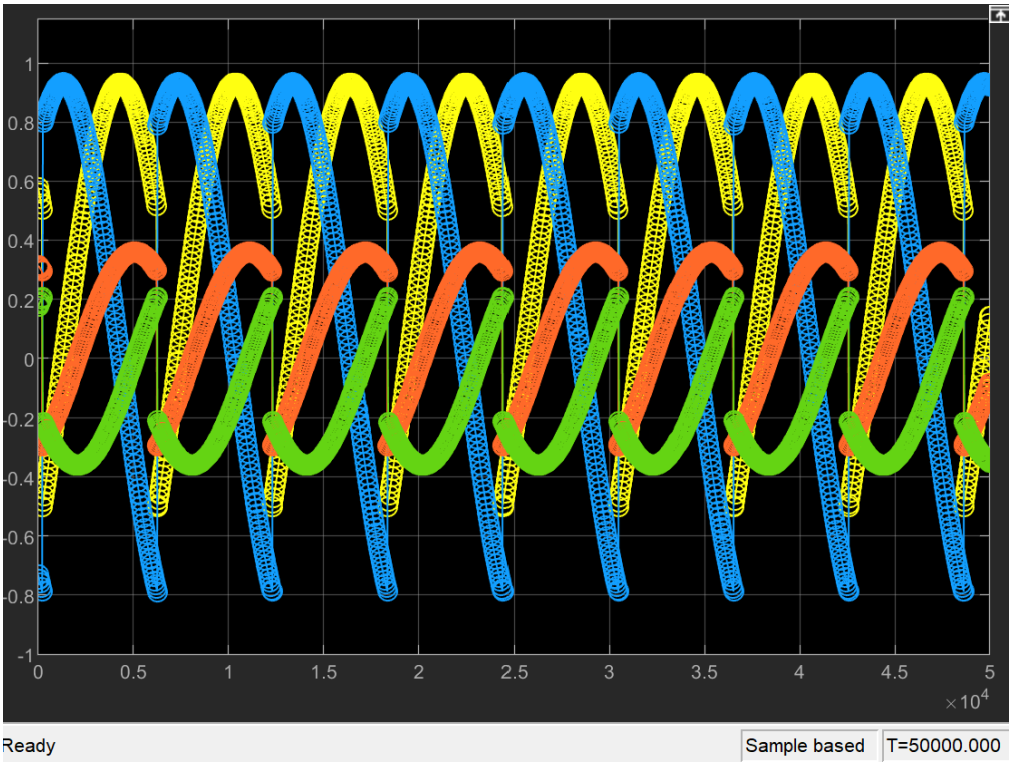

Figure 49: Quaternion desired

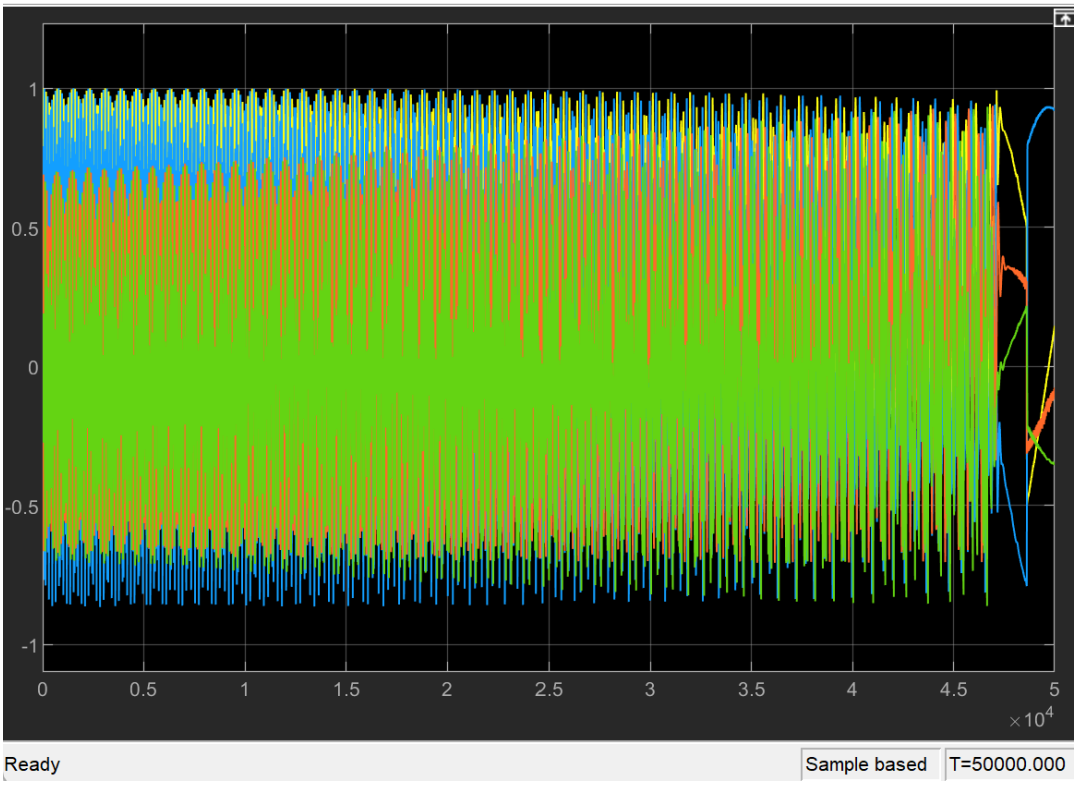

Figure 50: TRIAD quaternion

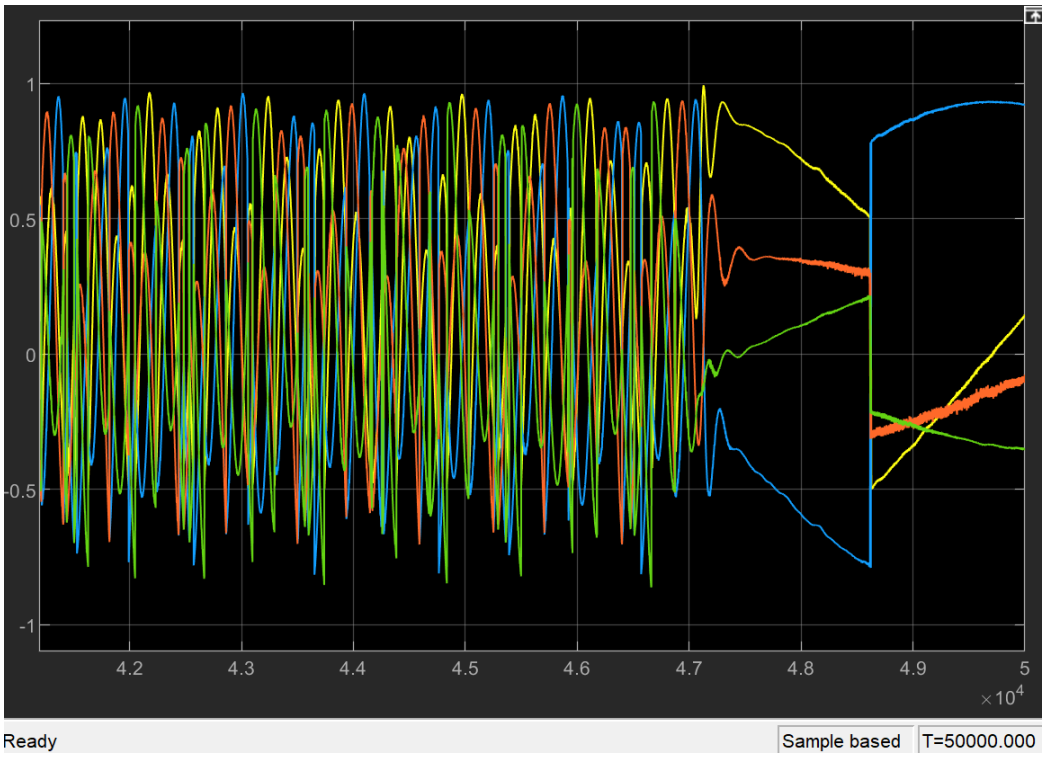

Figure 51: TRIAD quaternion close up

Other useful plots:

Figure 52 shows the current that was calculated in the B_dot block and used in the magnetorquers. Figure 53 shows the scope of the position vector from the two-body function block. Figure 54 shows the torque desired for the reaction wheels that were calculated in the Lyap controller, and Figure 55 shows the scope for the torque from the control input.

In conclusion, all the objectives of the project have been completed. The space vehicle detumbles in less than ten orbits, with an angular velocity of less than 0.2 RPM. The space vehicle b2 and b3 vectors are pointing toward the velocity vector and the center of the Earth, respectively. The many scopes of the attitude states also confirm that the model is correct, as the ideal and estimated functions behave very similarly.

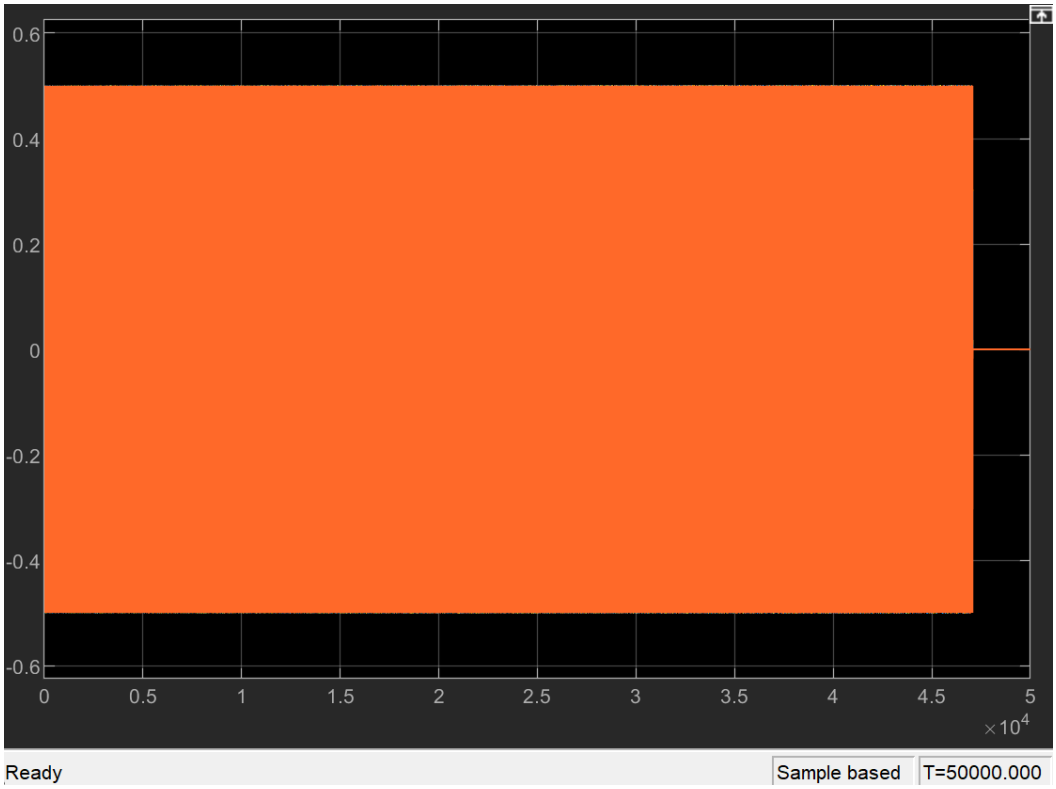

Figure 52: Scope of the current (I_array)

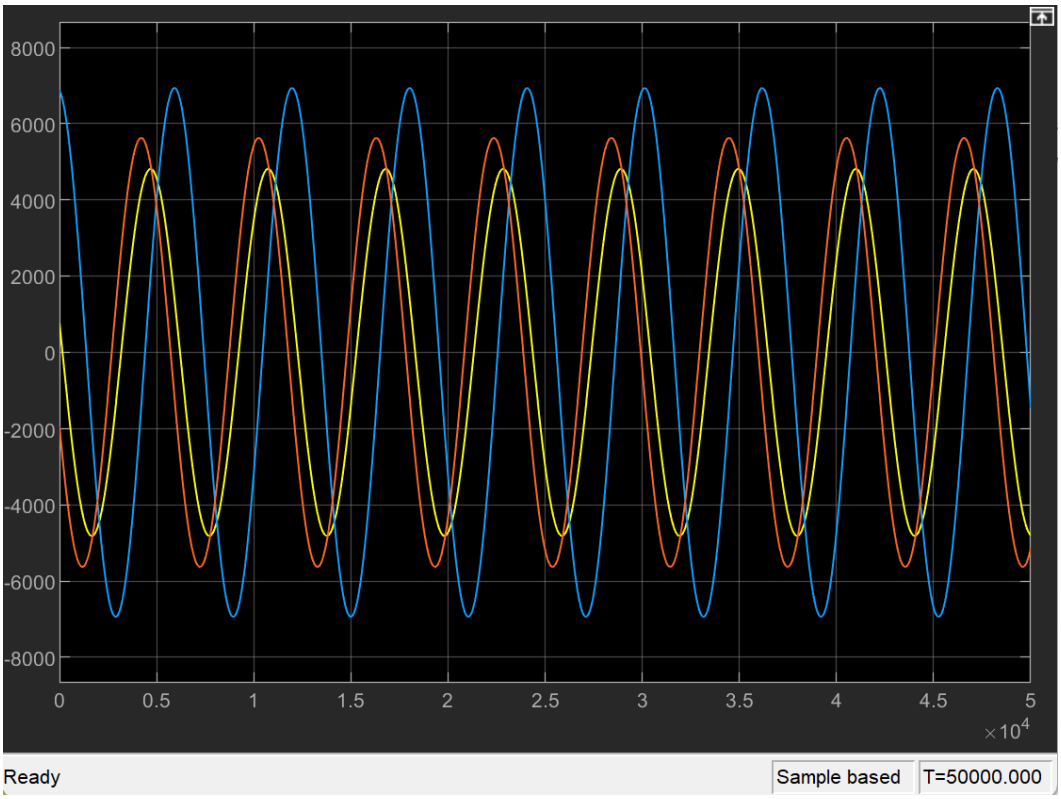

Figure 53: Scope of the position vector

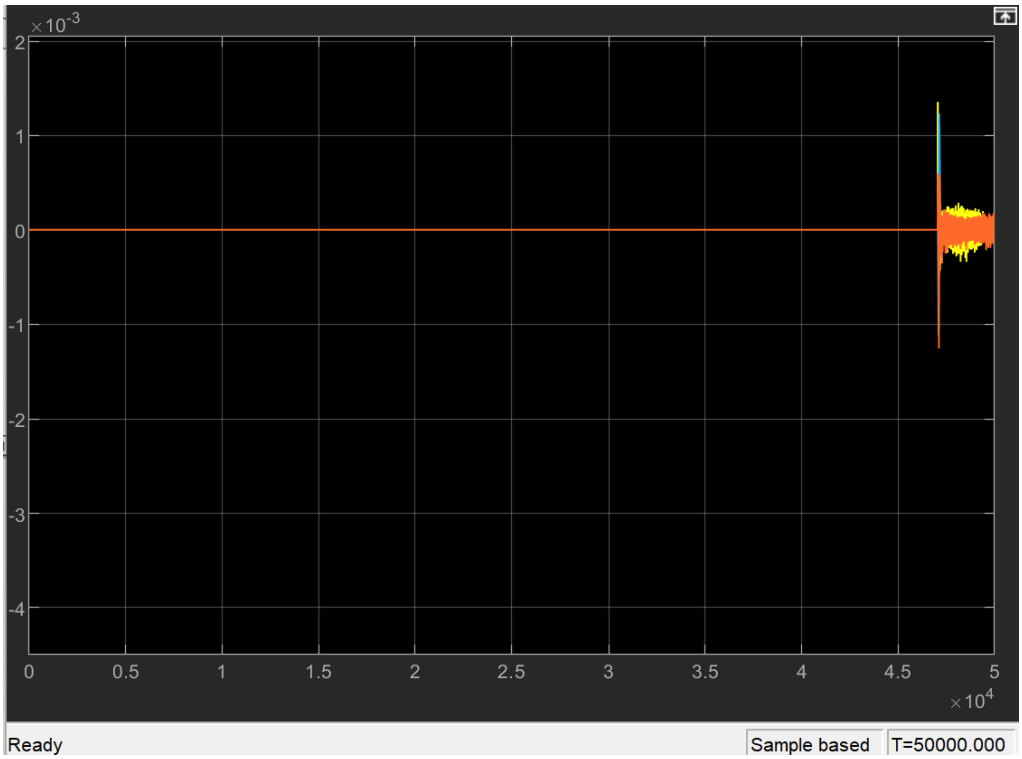

Figure 54: Scope of RW torque desired

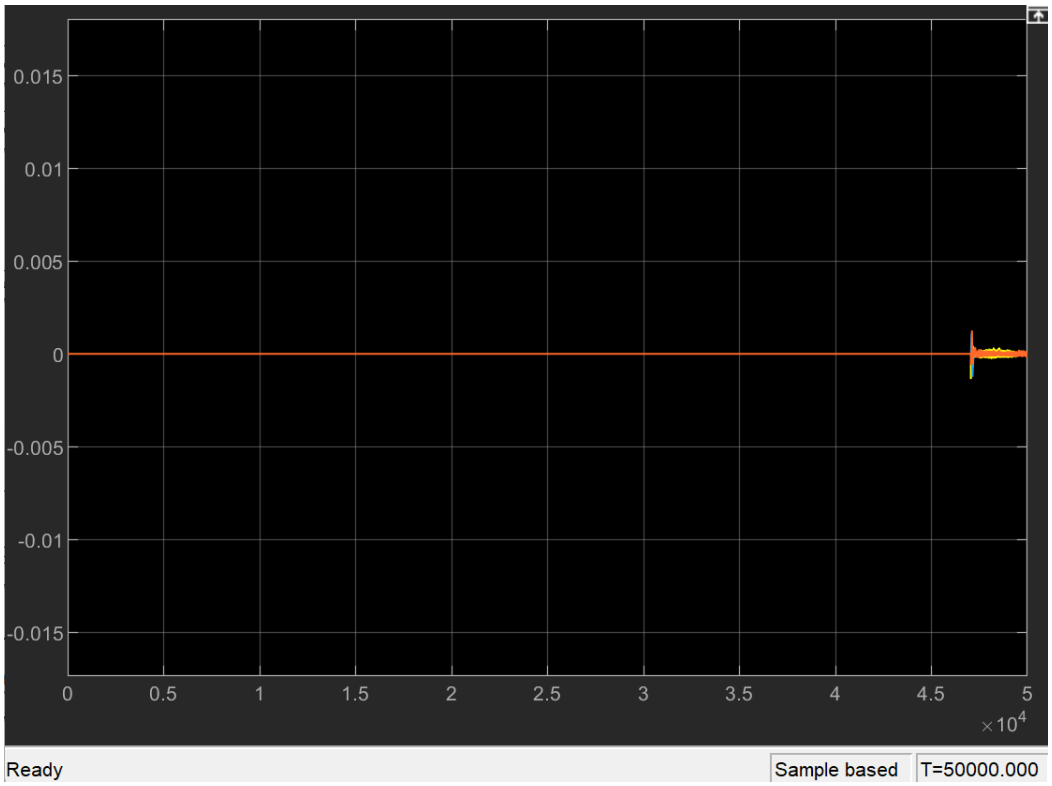

Figure 55: Scope of control input torque

References

[1] “Other Perturbations ,” Dr. C. Riano-Rios. Available: https://fit.instructure.com/courses/648094/files/50145703?module_item_id=10891035.

[2] “RW400 - high performance CubeSat Reaction Wheels ,” AAC Clyde Space Available: https://www.aac-clyde.space/what-we-do/space-products-components/adcs/rw400#:~:text=Our%20high%2Dperformance%20RW400%20reaction,15%2C%2030%20or%2050%20mN.

Appendix

MATLAB Code

```
%Project

%variables
sigma = 4.5e-8;
```

```
scale_factor = 4000;

animate_attitude(out.quat4.time, out.quat4.data, 'qua', out.position4.data/scale_factor)
```