PROBA-2 Attitude System Control Design

Spacecraft Attitude Dynamics and Control Project

Ahmed A. Moussa [101142994]

December 6, 2023

1 Dynamical Equations

Since the torque applied to the reaction wheels is an internal torque, their angular momentum contributes to the spacecraft's total angular momentum. This is due to the principle of angular momentum conservation. The following expression yields the components of the total angular momentum of PROBA-2 in the body-fixed reference frame.

$$\mathbf{h_{tot_B}} = \mathbf{J}\omega_\mathbf{B} + \sum_{i=1}^{3} h_{a_i}\mathbf{a_{i_B}} = \mathbf{J}\omega_\mathbf{B} + \mathbf{h_{rw_B}} \tag{1}$$

The total angular momentum of the spacecraft is used in Euler's equation of motion to determine its instantaneous rate of change.

$$\tau_\mathbf{B} = \dot{\mathbf{h}}_{\mathbf{tot_B}} + \omega_\mathbf{B}^\mathbf{x}\mathbf{h_{tot_B}}$$

$$\dot{\mathbf{h}}_{\mathbf{tot_B}} = \tau_\mathbf{B} - \omega_\mathbf{B}^\mathbf{x}\mathbf{h_{tot_B}} \tag{2}$$

2 Dynamical Model for Angular Velocity

Differentiating Eq. (1) with respect to time yields another equation for $\dot{\mathbf{h}}_{\mathbf{tot_B}}$.

$$\dot{\mathbf{h}}_{\mathbf{tot_B}} = \mathbf{J}\dot{\omega}_\mathbf{B} + \dot{\mathbf{h}}_{\mathbf{rw_B}} \tag{3}$$

Equating both equations (2) & (3) and re-arranging for $\mathbf{J}\dot{\omega}$ results in:

$$\mathbf{J}\dot{\omega}_\mathbf{B} = \tau_\mathbf{B} - \omega_\mathbf{B}^\mathbf{x}\mathbf{h_{tot_B}} - \dot{\mathbf{h}}_{\mathbf{rw_B}}$$

Integrating both sides of this equation, substituting Eq. (1) for $\mathbf{h_{tot_B}}$, and solving for $\omega_\mathbf{B}$ results in the following final expression.

$$\omega_\mathbf{B} = \mathbf{J}^{-1}\left[\int \tau_\mathbf{B} - \int \omega_\mathbf{B}^\mathbf{x}[\mathbf{J}\omega_\mathbf{B} + \mathbf{h_{rw_B}}] - \mathbf{h_{rw_B}}\right] \tag{4}$$

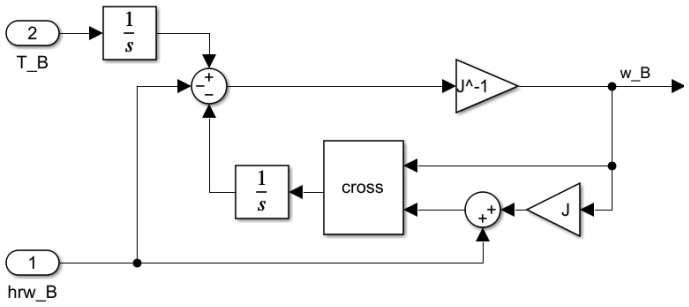

Figure 1: Block scheme diagram for the dynamical model of

3 Implementation of Attitude Dynamics & Kinematics

See Fig. 2 for the reaction wheel torque inputs over time using simple periodic control torque functions.
See Fig. 3 for the spacecraft quaternions and angular rates over time.

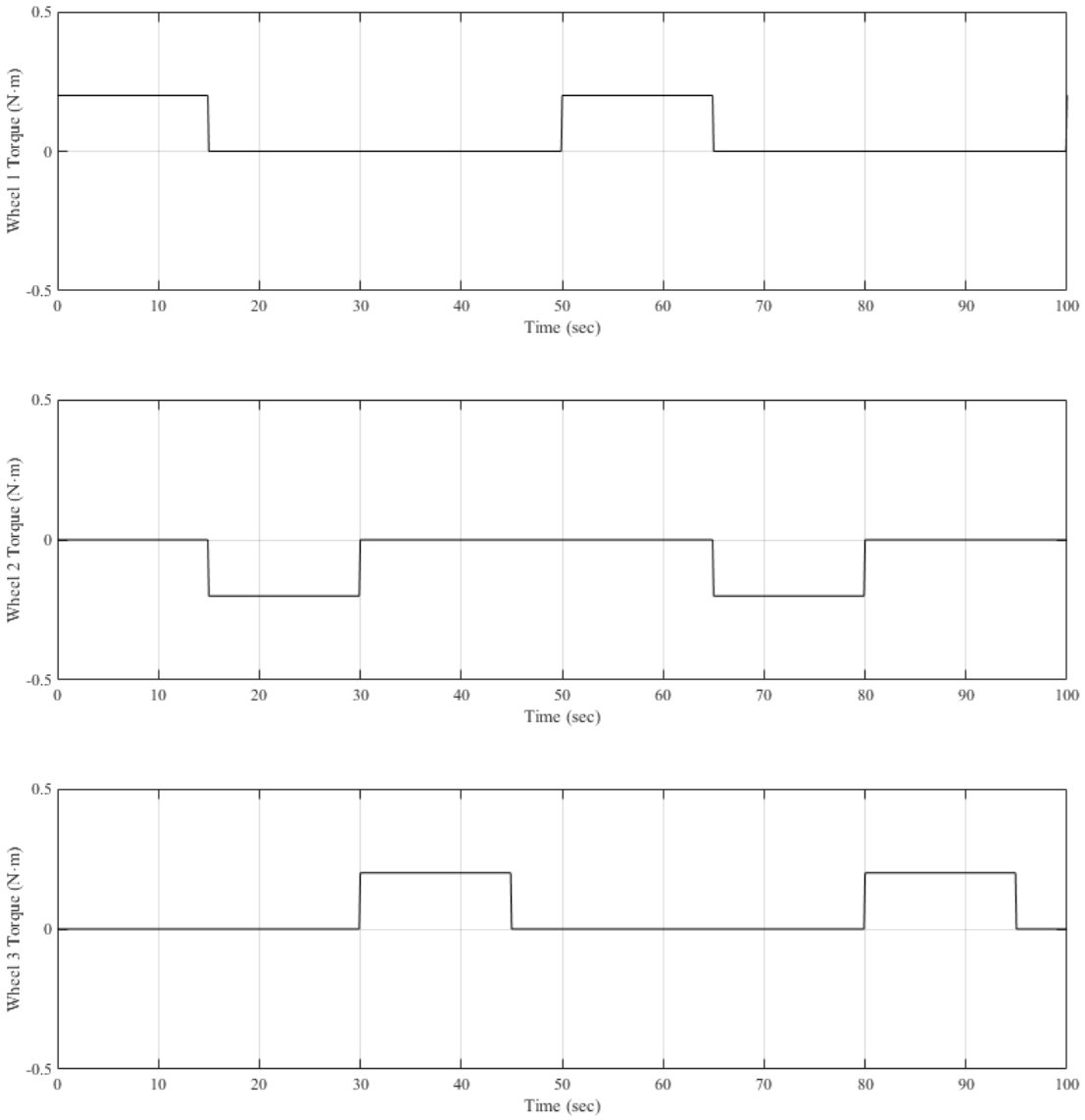

Figure 2: Reaction wheel torque inputs over time.

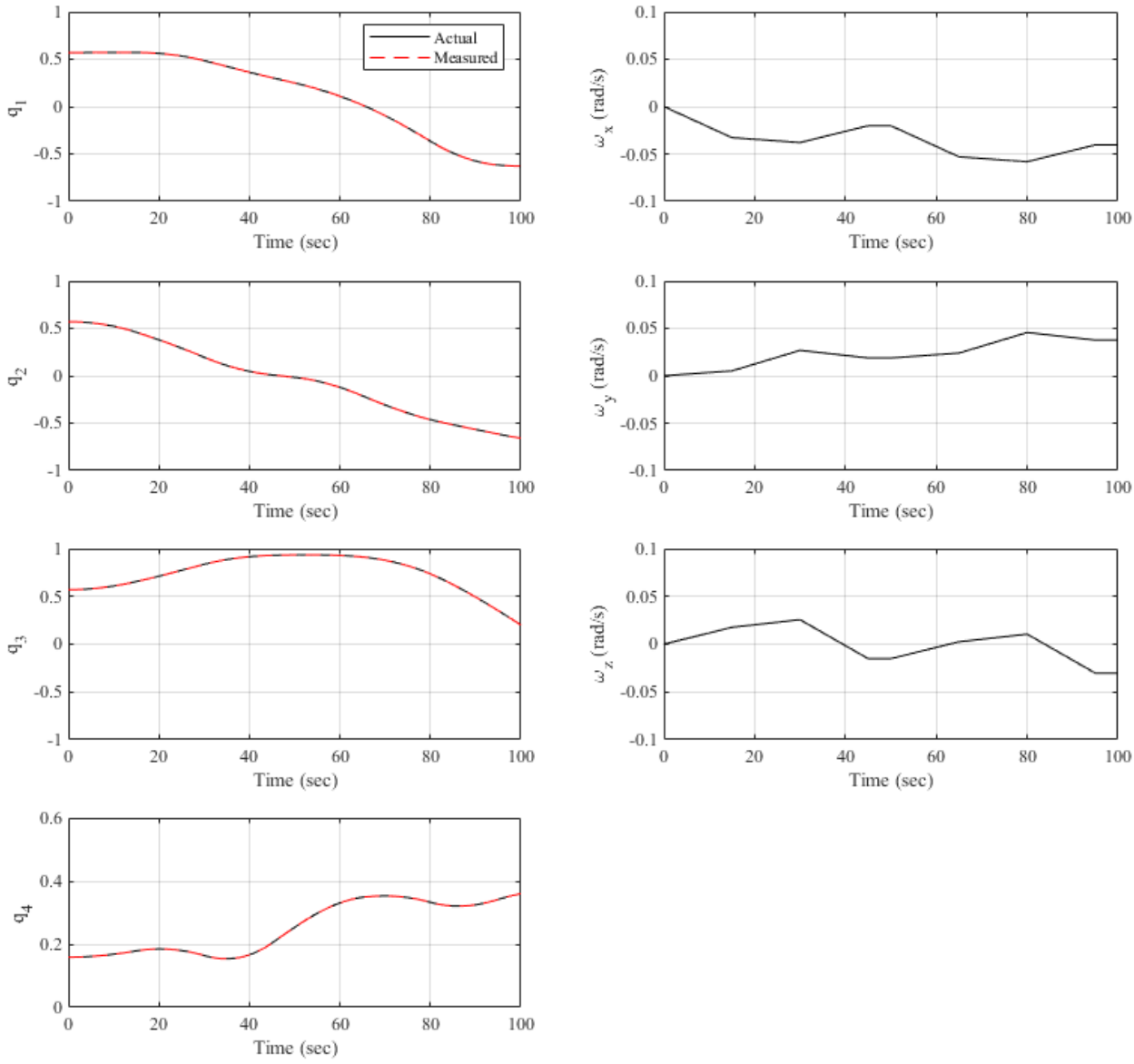

Figure 3: Spacecraft quaternions and angular rates over time.

4 Implementation of Modified PD Controller

4.1 Gain Calculations

The inverse inertia design method for proportional gain determination uses equation (4.1).

$$K_{p_i} = \frac{k}{J_i^*} \tag{5}$$

Selecting $k = 300$ results in:

$$\mathbf{K_p} = \begin{bmatrix} 300/120 & 0 & 0 \\ 0 & 300/150 & 0 \\ 0 & 0 & 300/100 \end{bmatrix} = \begin{bmatrix} 2.5 & 0 & 0 \\ 0 & 2.0 & 0 \\ 0 & 0 & 3.0 \end{bmatrix}$$

Normalizing the values for $K_{22} = 11$ results in the following proportional gain matrix

$$\mathbf{K_{p_n}} = \begin{bmatrix} 13.75 & 0 & 0 \\ 0 & 11 & 0 \\ 0 & 0 & 16.5 \end{bmatrix}$$

The scaled identity design method for proportional gain determination uses equation (6).

$$\mathbf{K_p} = k\mathbf{I_3} \tag{6}$$

Selecting $k = 20$ results in:

$$\mathbf{K_p} = 20 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{bmatrix}$$

Normalizing the values for $K_{22} = 11$ results in the following proportional gain matrix

$$\mathbf{K_{p_n}} = \begin{bmatrix} 11 & 0 & 0 \\ 0 & 11 & 0 \\ 0 & 0 & 11 \end{bmatrix}$$

The alpha-beta eigenaxis design method for proportional gain determination uses equation (7) where alpha and beta are defined by equations (8) and (9).

$$\mathbf{K_p} = (\alpha \mathbf{J}^* + \beta \mathbf{I_3})^{-1} \tag{7}$$

$$\alpha = \frac{\left[9 - \left(\sum_{i=1}^{3} 1/J_i\right)\left(\sum_{i=1}^{3} J_i\right)\right]}{\left[3\left(\sum_{i=1}^{3} J_i^2\right) - \left(\sum_{i=1}^{3} J_i\right)^2\right]} \tag{8}$$

$$\beta = \frac{\left[\left(\sum_{i=1}^{3} 1/J_i\right)\left(\sum_{i=1}^{3} J_i^2\right) - 3\left(\sum_{i=1}^{3} J_i\right)\right]}{\left[3\left(\sum_{i=1}^{3} J_i^2\right) - \left(\sum_{i=1}^{3} J_i\right)^2\right]} \tag{9}$$

Solving with J as an array of the diagonal elements of the spacecraft's inertia matrix yields $\alpha = -6.5789 \times 10^{-5}$ and $\beta = 0.0164$. Substituting these values into equation (7) yields:

$$\mathbf{K_p} = \left((-6.5789 \times 10^{-5}) \begin{bmatrix} 120 & 0 & 0 \\ 0 & 150 & 0 \\ 0 & 0 & 100 \end{bmatrix} + 0.0164 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)^{-1}$$

$$= \begin{bmatrix} 0.0086 & 0 & 0 \\ 0 & 0.0066 & 0 \\ 0 & 0 & 0.0099 \end{bmatrix}^{-1} = \begin{bmatrix} 116.9231 & 0 & 0 \\ 0 & 152 & 0 \\ 0 & 0 & 101.3333 \end{bmatrix}$$

Normalizing the values for $K_{22} = 11$ results in the following proportional gain matrix

$$\mathbf{K_{p_n}} = \begin{bmatrix} 8.4615 & 0 & 0 \\ 0 & 11 & 0 \\ 0 & 0 & 7.3333 \end{bmatrix}$$

The specification-based eigenaxis design method for proportional gain determination is characterized by equation (10). For reference, this attitude control system's natural frequency is desired to be 0.156 rad/s.

$$\mathbf{K_p} = 2\omega_n^2 \mathbf{J}^* \tag{10}$$

Solving with a natural frequency of 0.156 yields:

$$\mathbf{K_p} = 2\left(0.156^2\right) \begin{bmatrix} 120 & 0 & 0 \\ 0 & 150 & 0 \\ 0 & 0 & 100 \end{bmatrix} = \begin{bmatrix} 5.8406 & 0 & 0 \\ 0 & 7.3008 & 0 \\ 0 & 0 & 4.8672 \end{bmatrix}$$

Normalizing the values for $K_{22} = 11$ results in the following proportional gain matrix

$$\mathbf{K_{p_n}} = \begin{bmatrix} 8.8 & 0 & 0 \\ 0 & 11 & 0 \\ 0 & 0 & 7.3333 \end{bmatrix}$$

The specification-based eigenaxis design method for feedback gain determination is characterized by equation (11).

$$\mathbf{K_d} = 2\zeta\omega_n \mathbf{J}^* \tag{11}$$

Using a modified settling time relation $t_s = 8/\zeta\omega_n$, equation (11) can be rewritten as

$$\mathbf{K_d} = \frac{16}{t_s}\mathbf{J}^* \tag{12}$$

With the desired settling time of 50 seconds, the feedback gain matrix is solved via equation (12) as follows:

$$\mathbf{K_d} = \frac{16}{50} \begin{bmatrix} 120 & 0 & 0 \\ 0 & 150 & 0 \\ 0 & 0 & 100 \end{bmatrix} = \begin{bmatrix} 38.4 & 0 & 0 \\ 0 & 48 & 0 \\ 0 & 0 & 32 \end{bmatrix}$$

4.2 Quaternion Plots

See Fig. 4 for quaternion and angular velocity plots in a torque-free environment.
See Fig. 5 for q_i vs. q_j plots in a torque-free environment.

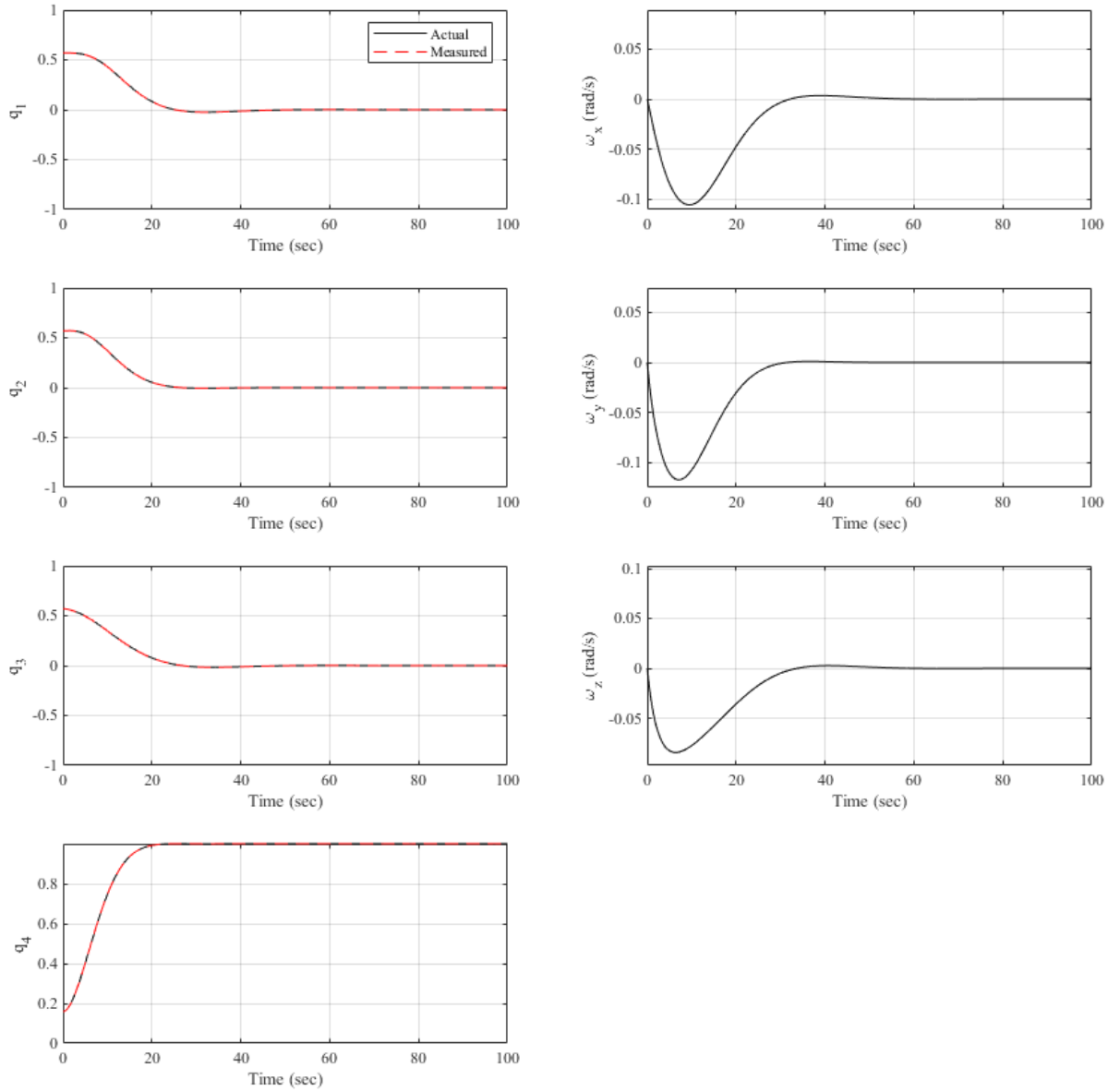

Figure 4: Spacecraft quaternion and angular velocity component plots without the effect of perturbations.

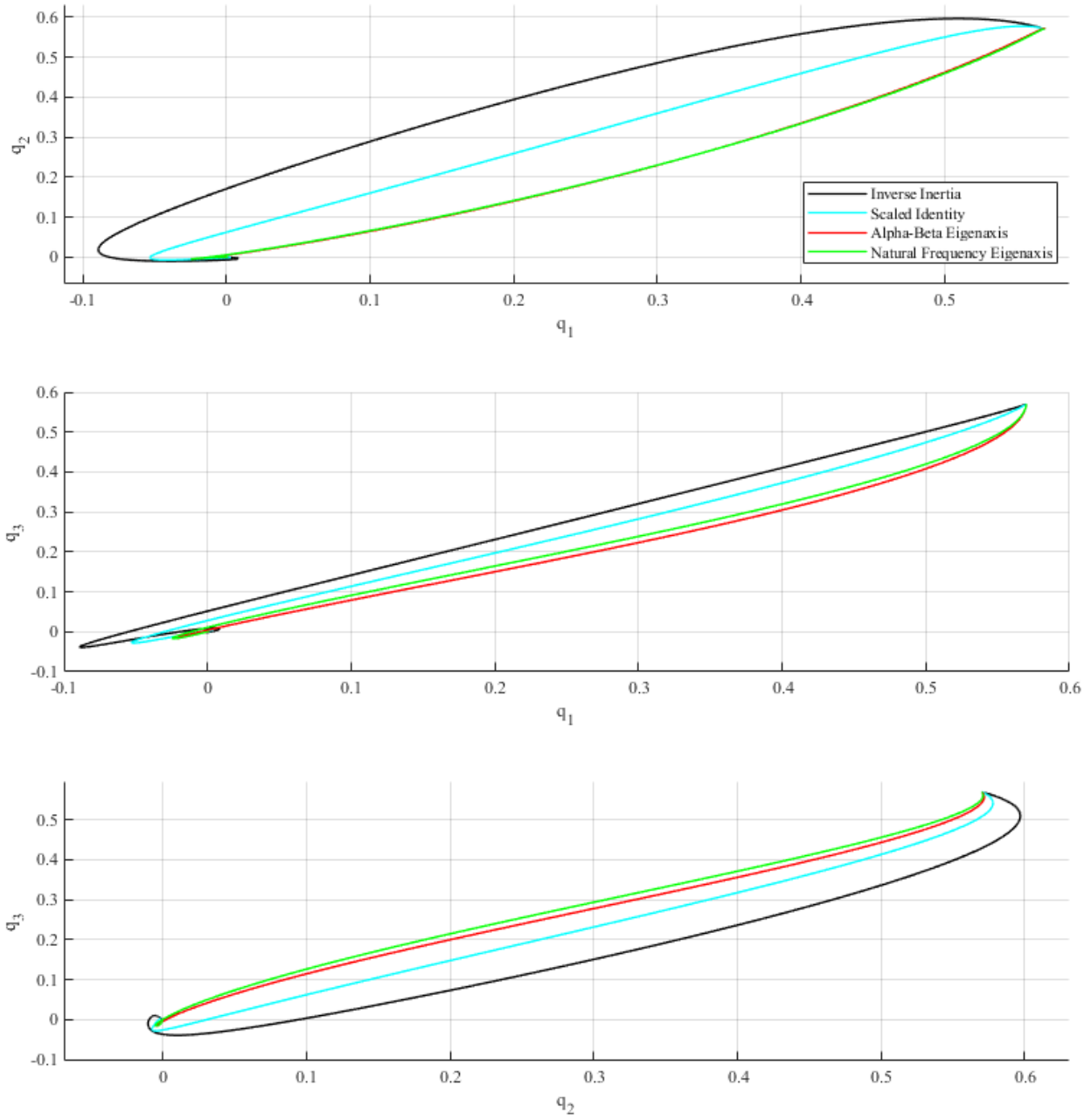

Figure 5: Spacecraft q_i vs. q_j plots without the effect of perturbations.

5 Implementation of External Torques

5.1 Orbital Distance & Speed Plots

See Fig. 6 for plots of orbital distance and speed as function of time.

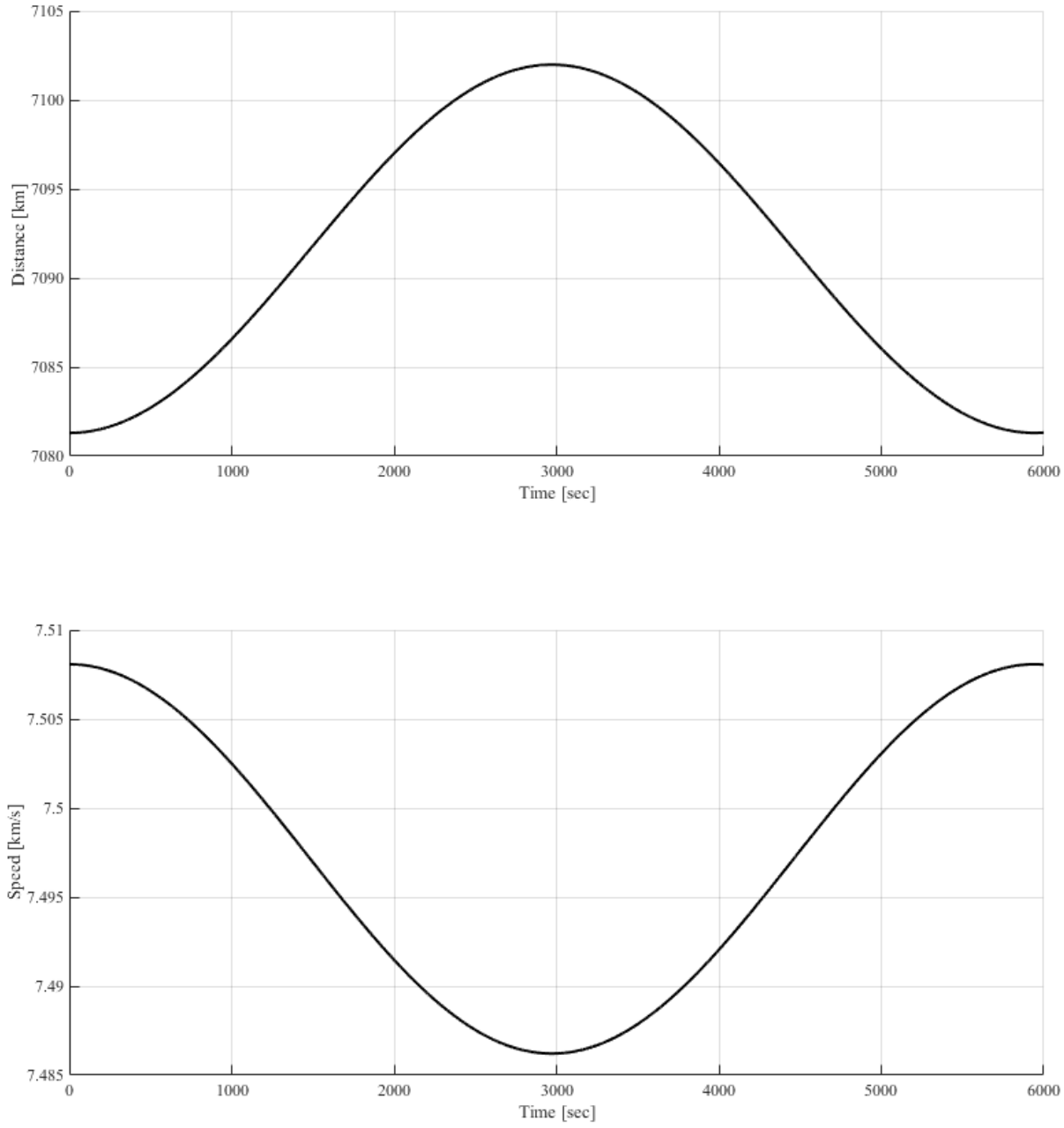

Figure 6: Spacecraft orbital distance and speed plots for one full orbit while disregarding the effects of perturbations.

5.2 Net Perturbing Torque Plot

See Fig. 7 for a plot of the net perturbing torque as function of time.

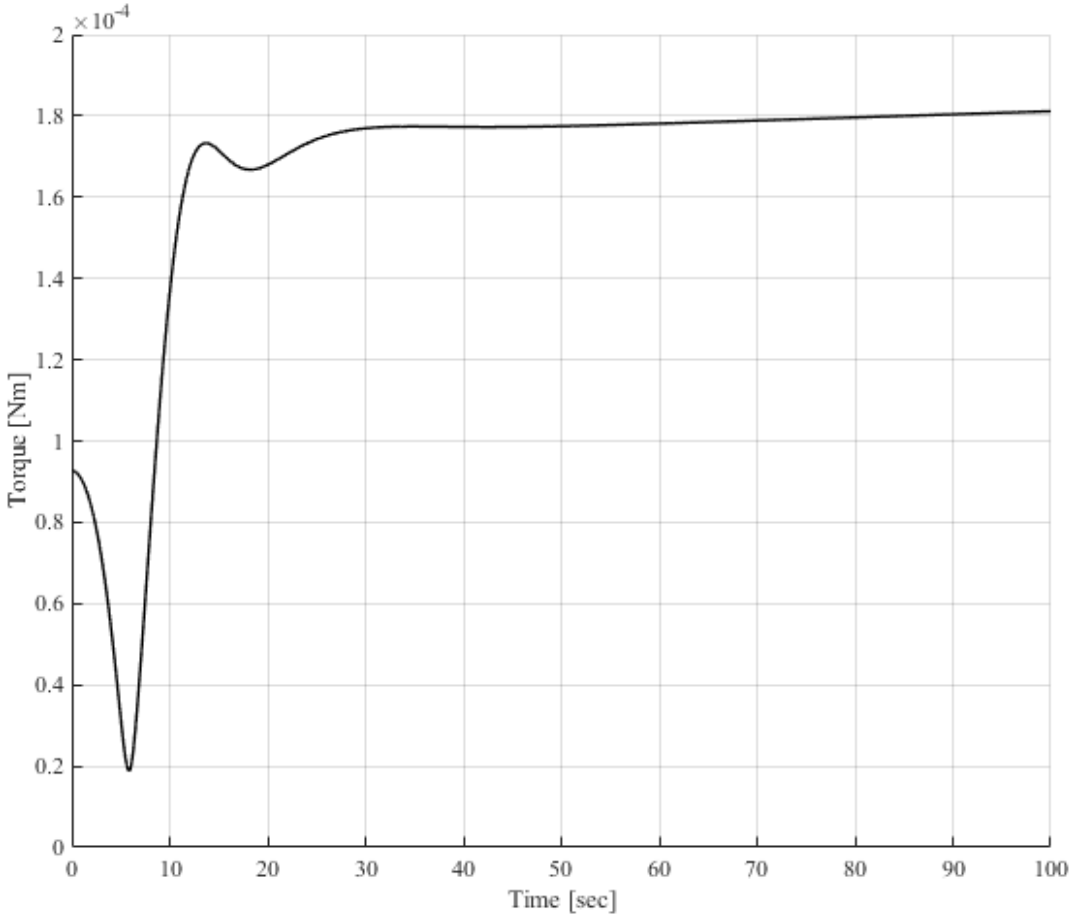

Figure 7: Magnitude of external torques acting on the spacecraft.

5.3 Quaternion Plots

See Fig. 8 for quaternion and angular velocity plots using control gains from the alpha-beta tuning methodology. See Fig. 9 for q_i vs. q_j plots.

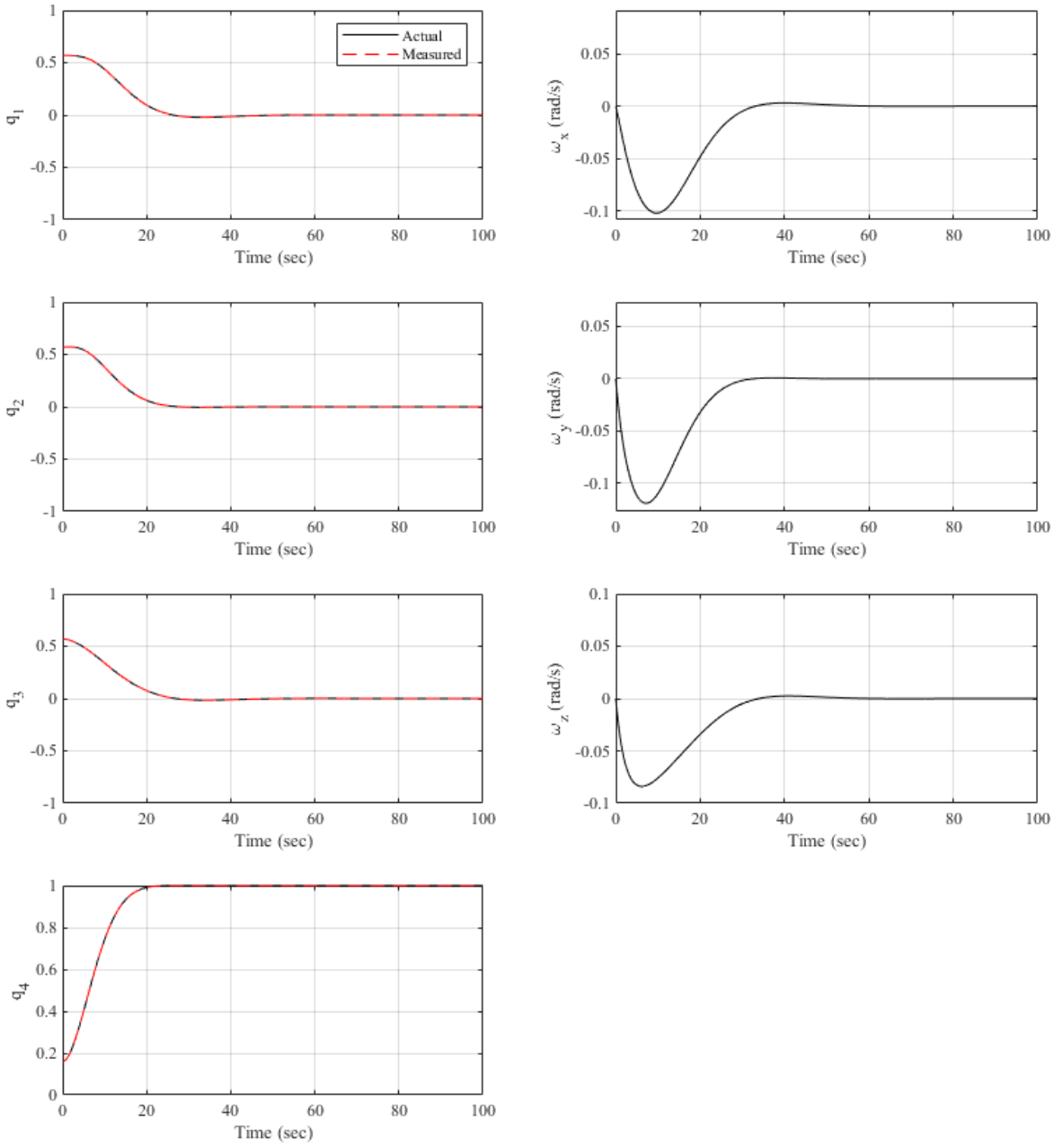

Figure 8: Spacecraft quaternion and angular velocity component plots.

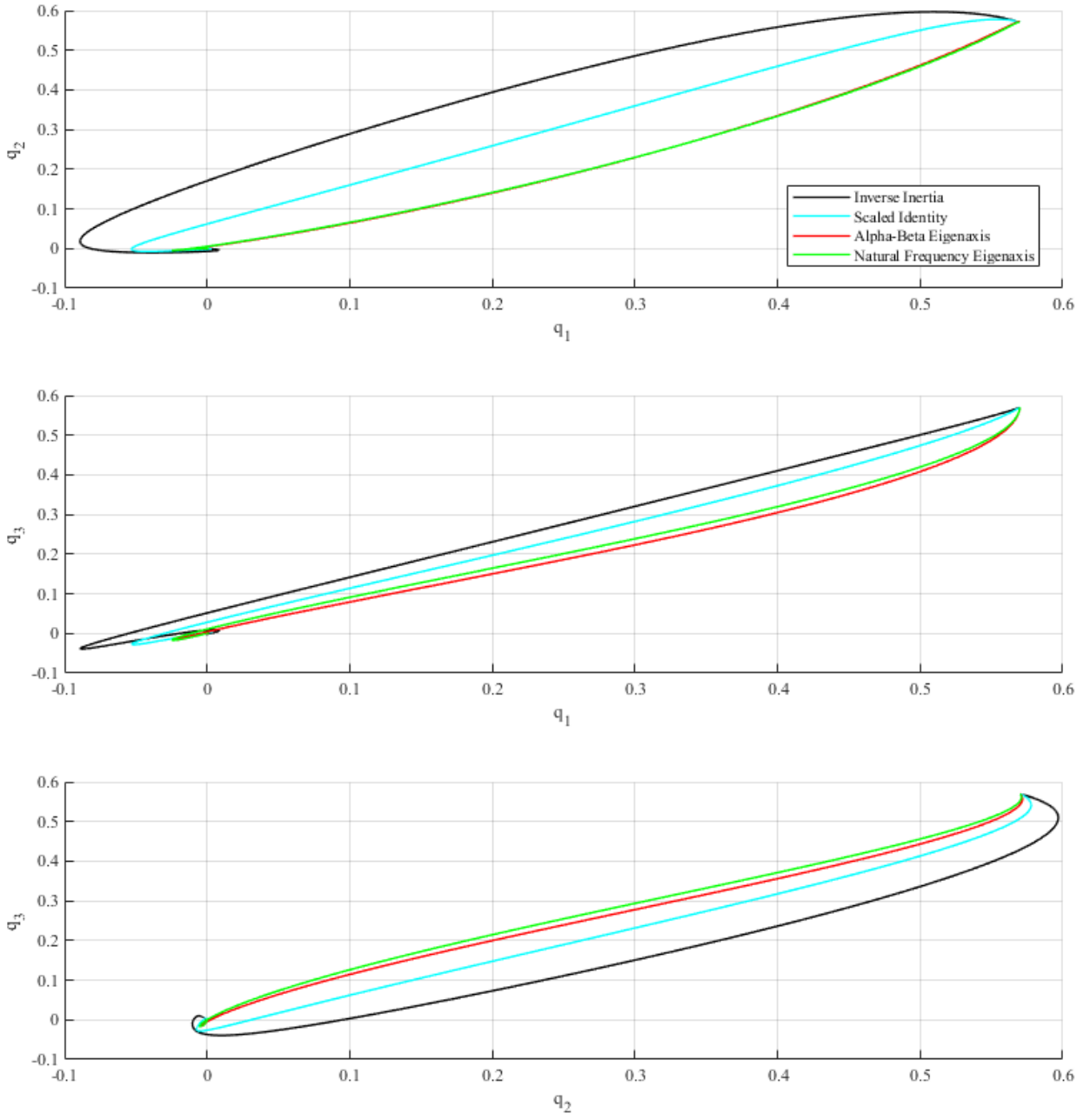

Figure 9: Spacecraft q_i vs. q_j plots.

Appendix A - Simulink Blocks

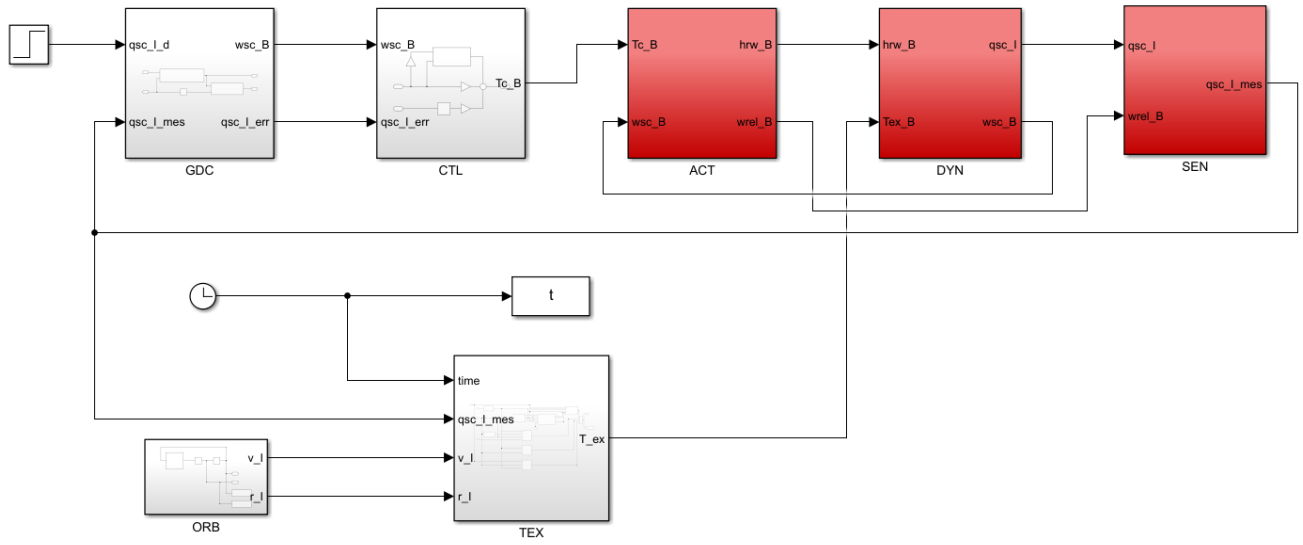

Figure 10: Simulink Top-Level Block Model

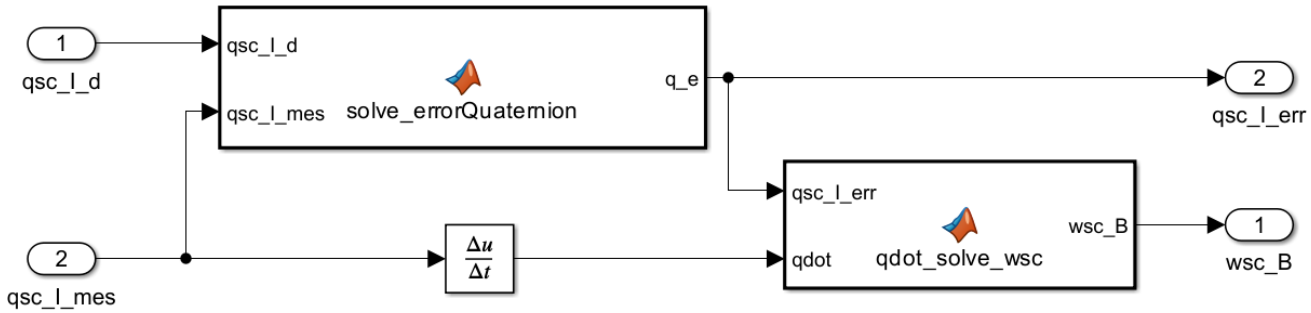

Figure 11: GDC Block

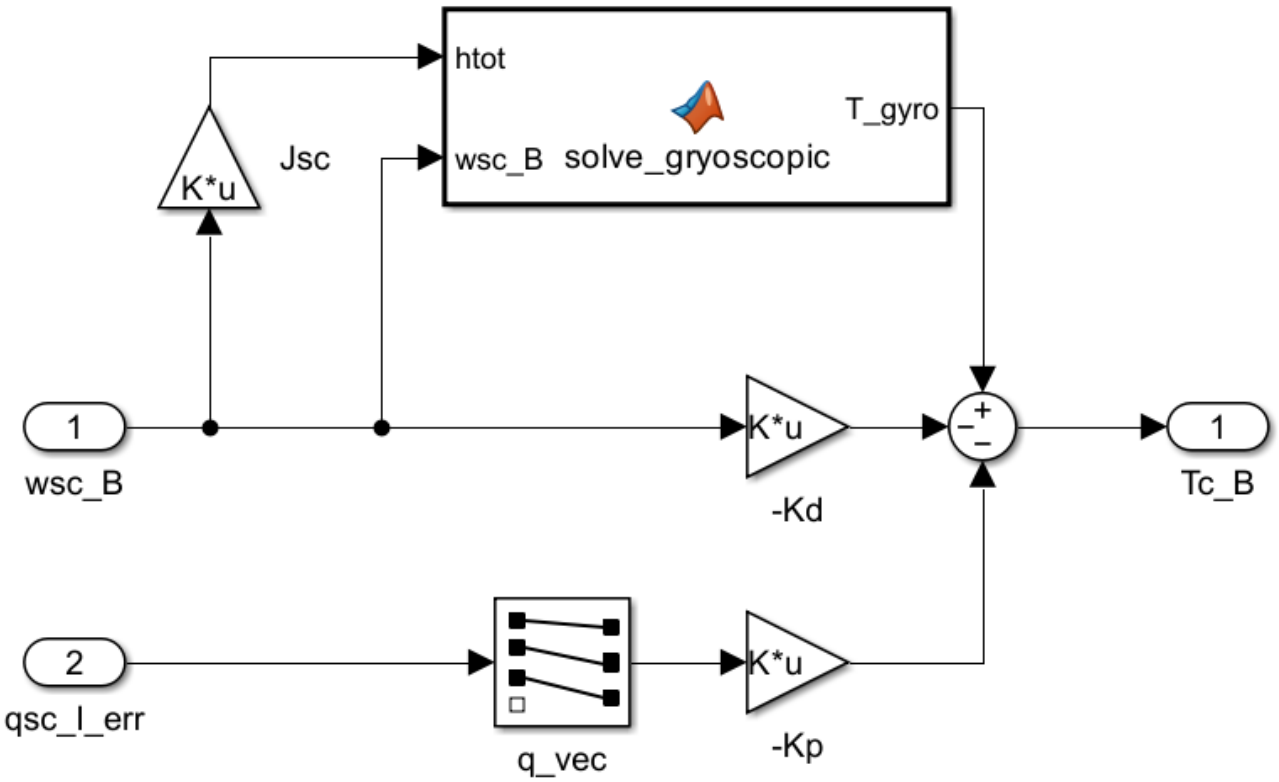

Figure 12: CTL Block

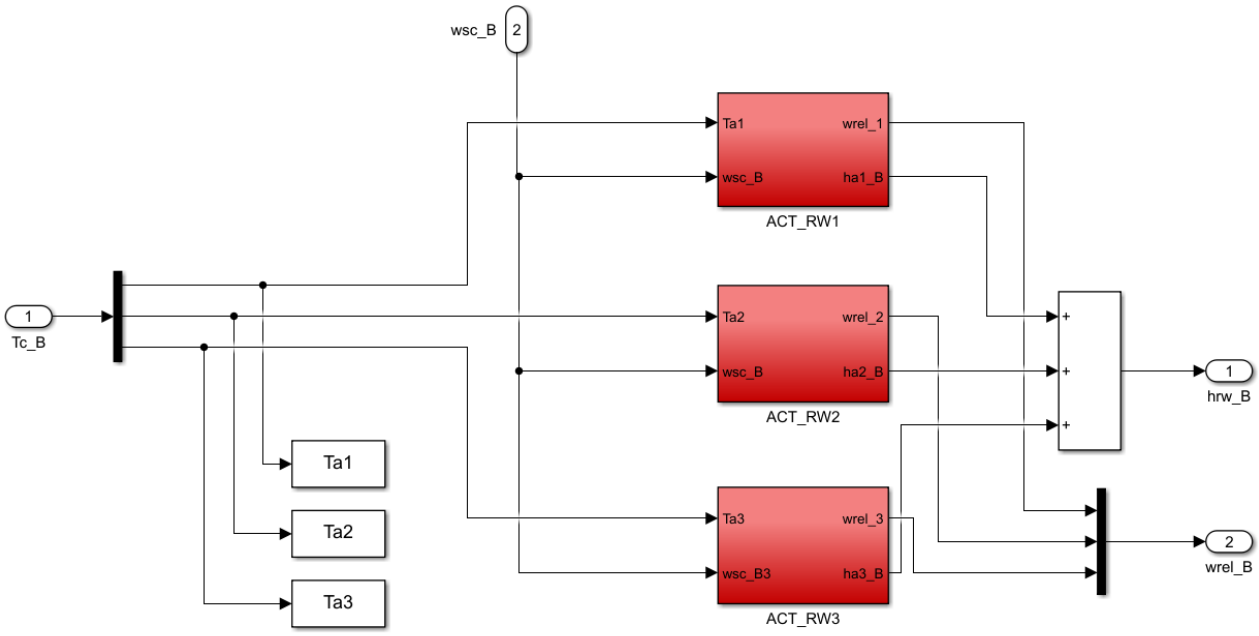

Figure 13: ACT Block

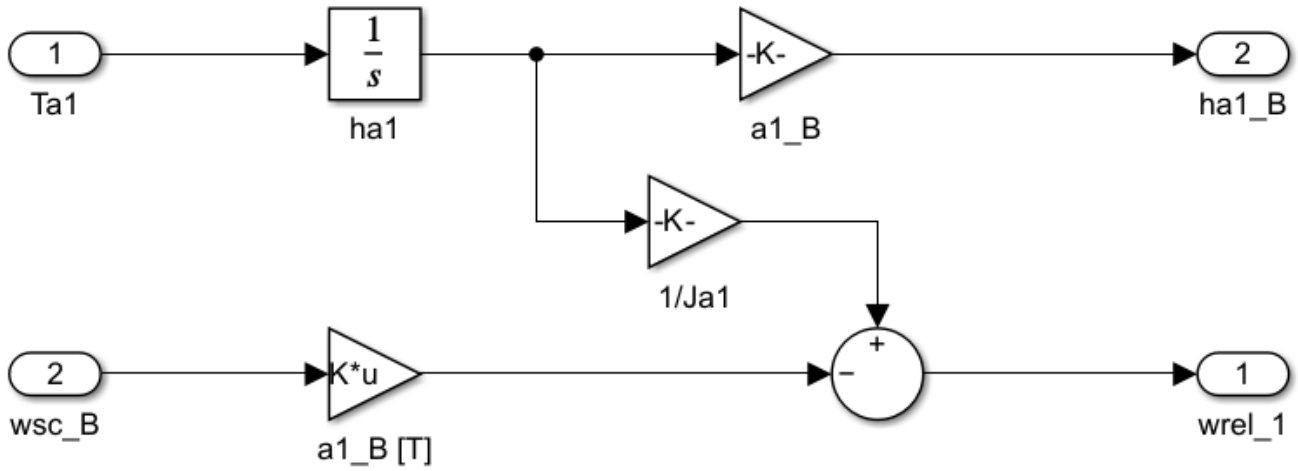

Figure 14: ACT_RW Block

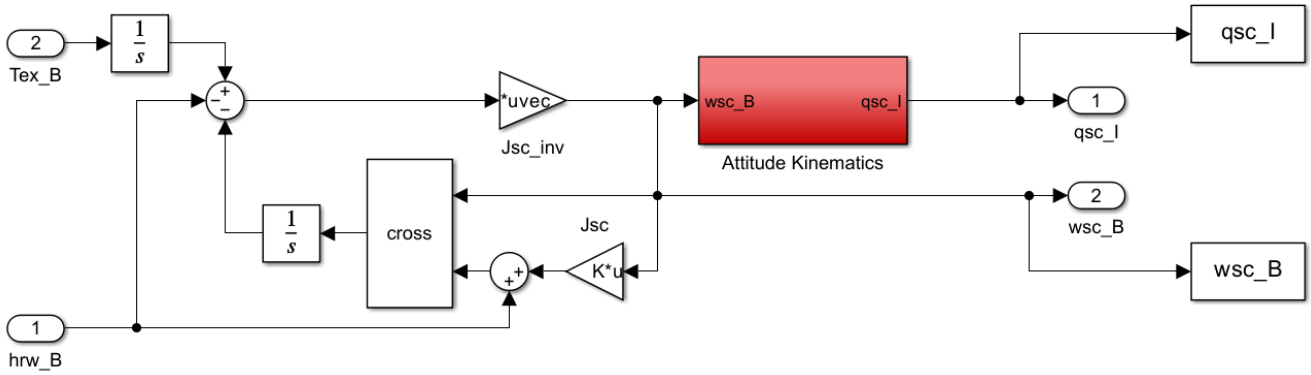

Figure 15: DYN Block

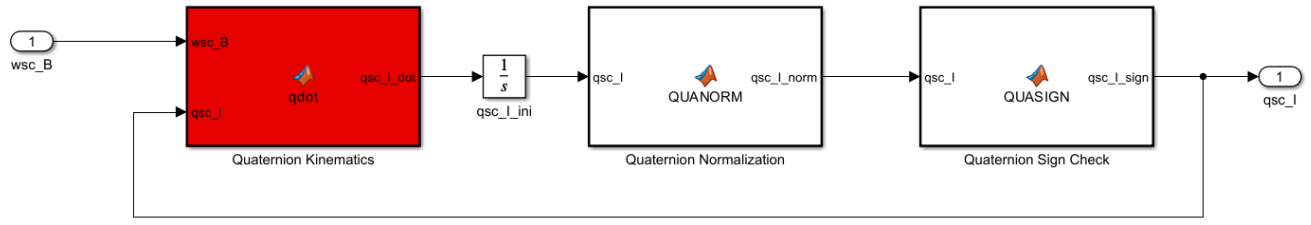

Figure 16: Attitude Kinematics Block

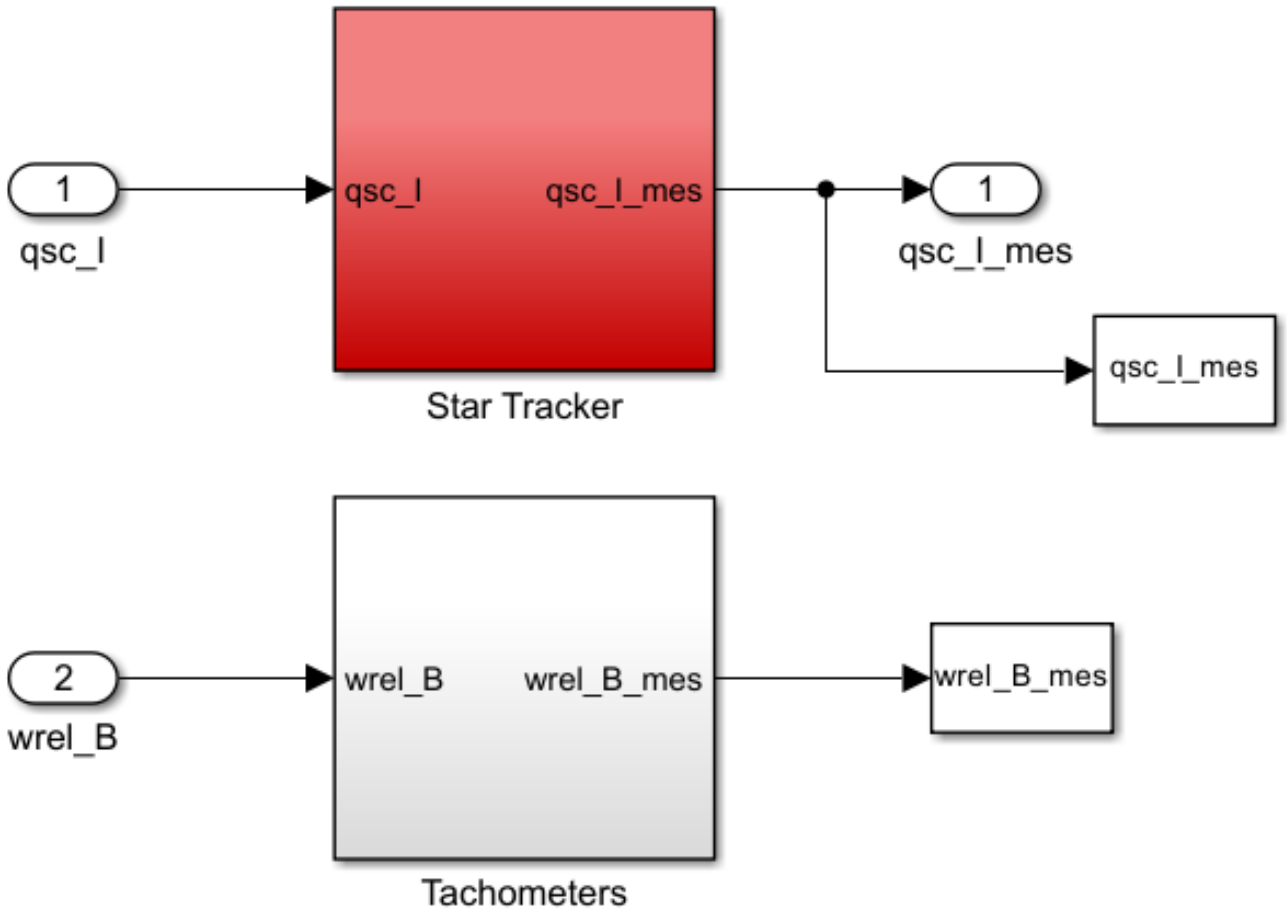

Figure 17: SEN Block

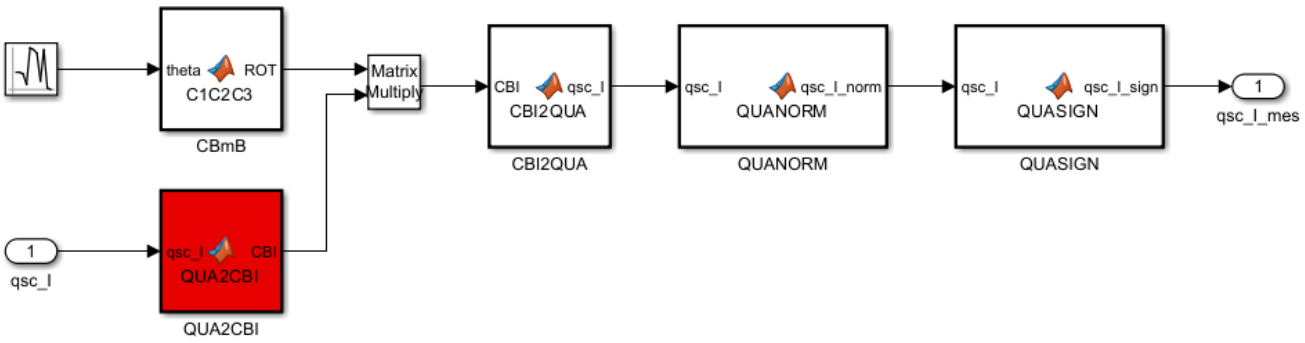

Figure 18: Star Tracker Block

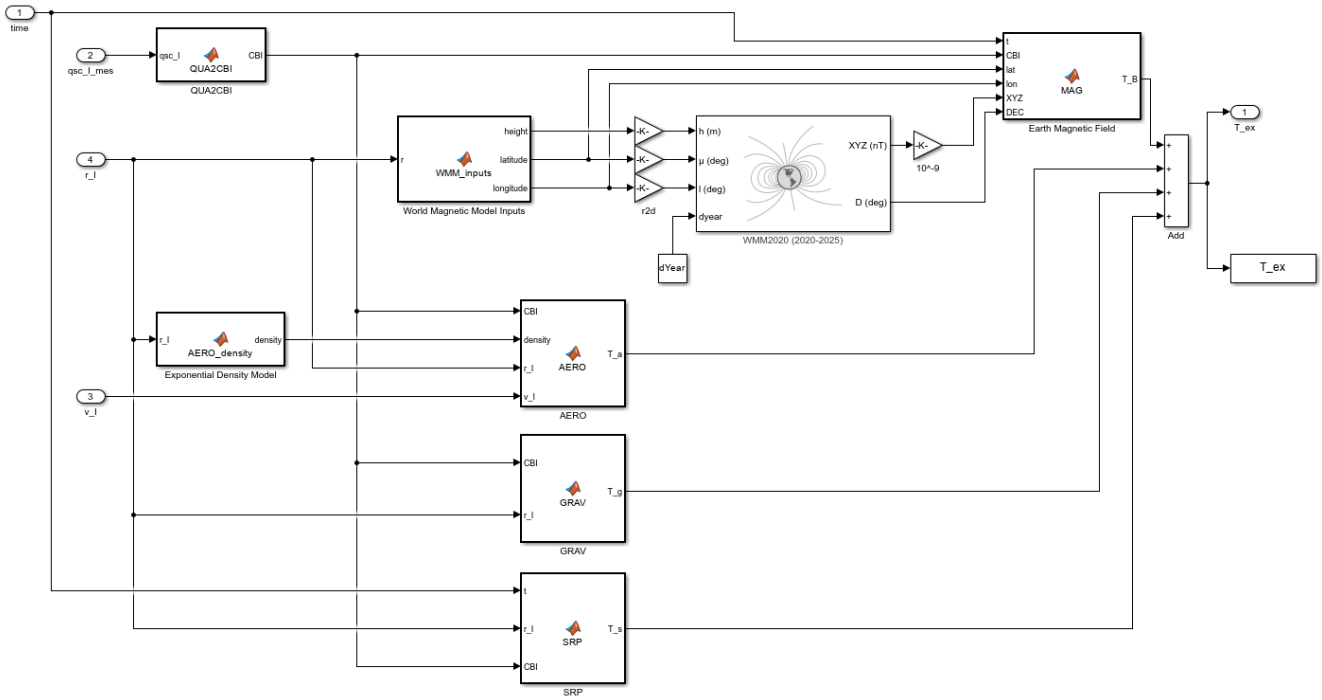

Figure 19: TEX Block

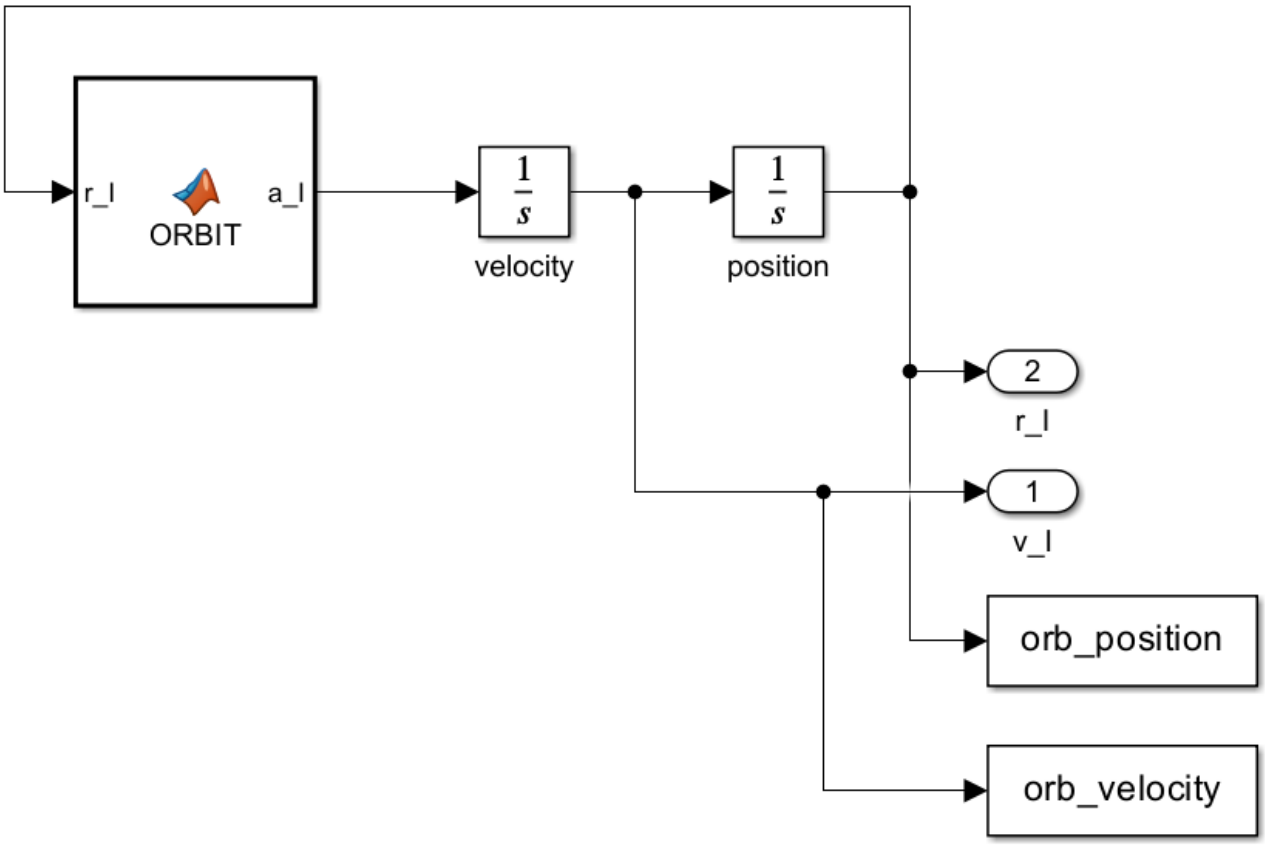

Figure 20: ORB Block

Appendix B - Simulink Functions

[NOTE] All "skew()" calls use a function I wrote in a file named "skew.m" seen in Appendix C.

```
function q_e = solve_errorQuaternion(qsc_I_d, qsc_I_mes)
% ---------------------------------------------------------------
% solve_errorQuaternion
% ---------------------------------------------------------------
% Description:
% Calculate the quaternion error between
% ---------------------------------------------------------------
% Inputs:
% qsc_I_d => desired quaternion (4 x 1 matrix)
% qsc_I_mes => measured quaternion (4 x 1 matrix)
% ---------------------------------------------------------------
% Outputs:
% q_e => quaternion error (4 x 1 matrix)
% ---------------------------------------------------------------
% Parameters:
% None
% ---------------------------------------------------------------

q_BI = [qsc_I_mes(4)*eye(3)-skew(qsc_I_mes(1:3)) qsc_I_mes(1:3); -qsc_I_mes(1:3).'
    qsc_I_mes(4)];
q_IBd = [-qsc_I_d(1:3); qsc_I_d(4)];

q_e = q_BI*q_IBd;
end
```

1: solve_errorQuaternion

```
function wsc_B = qdot_solve_wsc(qsc_I_err, qdot)
% ---------------------------------------------------------------
% qdot_solve_wsc
% ---------------------------------------------------------------
% Description:
% Solve for control angular rates from quaternion rate and error.
% ---------------------------------------------------------------
% Inputs:
% qsc_I_err => error quaternion (4 x 1 matrix)
% qdot => measured quaternion rate (4 x 1 matrix)
% ---------------------------------------------------------------
% Outputs:
% wsc_B => angular rate (3 x 1 matrix)
% ---------------------------------------------------------------
% Parameters:
% None
% ---------------------------------------------------------------

vec = qsc_I_err(1:3);
eta = qsc_I_err(4);

wsc_B = 2*((eta^2*eye(3) - eta*skew(vec) + vec*vec.')/eta)*qdot(1:3);
end
```

2: qdot_solve_wsc

```
function T_gyro = solve_gryoscopic(htot,wsc_B)
% ---------------------------------------------------------------
% solve_gyroscopic
% ---------------------------------------------------------------
```

```
% Description:
% Solve for the gyroscopic torque of the spacecraft's body.
% -------------------------------------------------------------
% Inputs:
% htot => angular momentum of spacecraft body (3 x 1 matrix)
% wsc_B => angular rate (3 x 1 matrix)
% -------------------------------------------------------------
% Outputs:
% T_gyro => gyroscopic torque (3 x 1 matrix)
% -------------------------------------------------------------
% Parameters:
% None
% -------------------------------------------------------------

T_gyro = skew(wsc_B)*htot;
end
```

3: solve_gryoscopic

```
function qsc_I_dot = qdot(wsc_B,qsc_I)
% -------------------------------------------------------------
% qdot
% -------------------------------------------------------------
% Description:
% Calculate the quaternion differential kinematics
% -------------------------------------------------------------
% Inputs:
% qsc_I => quaternion (4 x 1 matrix)
% wsc_B => angular rates (3 x 1 matrix)
% -------------------------------------------------------------
% Outputs:
% qsc_I_dot => quaternion derivatives (4 x 1 matrix)
% -------------------------------------------------------------
% Parameters:
% None
% -------------------------------------------------------------
% Copyright:
% Steve Ulrich, 2023
% -------------------------------------------------------------

qsc_I_dot = [0.5*(qsc_I(4)*eye(3)+skew(qsc_I(1:3))); -0.5*qsc_I(1:3).']*wsc_B;
end
```

4: qdot

```
function CBI = QUA2CBI(qsc_I)
% -------------------------------------------------------------
% QUA2CBI
% -------------------------------------------------------------
% Description:
% Convert a quaternion into the attitude matrix
% -------------------------------------------------------------
% Inputs:
% qsc_I => quaternion (4 x 1 matrix)
% -------------------------------------------------------------
% Outputs:
% CBI => attitude matrix (3 x 3 matrix)
% -------------------------------------------------------------
% Parameters:
% None
% -------------------------------------------------------------
```

```
% Copyright:
% Steve Ulrich, 2023
% -------------------------------------------------------------------
q_scalar = qsc_I(4);
q_vector = qsc_I(1:3);

CBI = (q_scalar^2-q_vector.'*q_vector)*eye(3)+2*(q_vector*q_vector.')-2*q_scalar*skew(
    q_vector);
end
```

5: QUA2CBI

```
function a_I  =  ORBIT(r_I, mu)
% -------------------------------------------------------------------
% ORBIT
% -------------------------------------------------------------------
% Description:
% Compute the spacecraft acceleration in ECI, using the two-body
% equation of motion.
% -------------------------------------------------------------------
% Inputs:
% r_I => components of r in ECI (3 x 1 matrix)
% -------------------------------------------------------------------
% Outputs:
% a_I => components of the acceleration in ECI (3 x 1 matrix)
% -------------------------------------------------------------------
% Parameters:
% mu => gravitational constant of the Earth
% -------------------------------------------------------------------

% Compute orbital radius
rmod = sqrt( r_I(1)*r_I(1) + r_I(2)*r_I(2) + r_I(3)*r_I(3) );

% Spacecraft acceleration
a_I = (-mu/rmod^3)*r_I;
end
```

6: ORBIT

```
function [height, latitude, longitude] = WMM_inputs(r, rE)
% -------------------------------------------------------------------
% WMM_inputs
% -------------------------------------------------------------------
% Description:
% Compute height, latitude, and longitude inputs for the WMM.
% -------------------------------------------------------------------
% Inputs:
% r => components of orbital position in ECI (3 x 1 matrix)
% -------------------------------------------------------------------
% Outputs:
% height => altitude of spacecraft [km]
% latitude => latitude of spacecraft [rad]
% longitude => longitude of spacecraft [rad]
% -------------------------------------------------------------------
% Parameters:
% rE => average radius of Earth
% -------------------------------------------------------------------

height = norm(r) - rE;
latitude = atan2(r(3),sqrt(r(1)^2+r(2)^2));
longitude = atan2(r(2),r(1));
```

```
end
```

7: WMM_inputs

```
function T_B = MAG(t, CBI, lat, lon, XYZ, DEC, wE, RMM_skew)
% -------------------------------------------------------------
% MAG
% -------------------------------------------------------------
% Description:
% Compute magnetic perturbation torque.
% -------------------------------------------------------------
% Inputs:
% t => time [sec]
% CBI => rotation matrix BI (3 x 3 matrix)
% lat => latitude [deg]
% lon => longitude [deg]
% XYZ => NED field components (3 x 1 matrix)
% DEC => declinasion
% -------------------------------------------------------------
% Outputs:
% T_B => magnetic perturbation torque (3 x 1 matrix)
% -------------------------------------------------------------
% Parameters:
% wE => spin rate of Earth
% RMM_skew => skew of residual magnetic moment
% -------------------------------------------------------------

e = lat - DEC*pi/180;
Bt = [
    -XYZ(1)*sin(e)-XYZ(3)*cos(e);
    -XYZ(1)*cos(e)+XYZ(3)*sin(e);
    XYZ(2)
];

GMT = wE*t;

B_I = [
    (Bt(1)*cos(lat)+Bt(2)*sin(lat))*cos(lon+GMT)-Bt(3)*sin(lon+GMT);
    (Bt(1)*cos(lat)+Bt(2)*sin(lat))*sin(lon+GMT)-Bt(3)*cos(lon+GMT);
    (Bt(1)*cos(lat)+Bt(2)*cos(lat))
];

T_B = RMM_skew * CBI * B_I;
end
```

8: MAG

```
function density = AERO_density(r_I, rE)
% -------------------------------------------------------------
% AERO_density
% -------------------------------------------------------------
% Description:
% Compute atmospheric density.
% -------------------------------------------------------------
% Inputs:
% r_I => orbital position in ECI (3 x 1 matrix)
% -------------------------------------------------------------
% Outputs:
% density => atmospheric density
% -------------------------------------------------------------
% Parameters:
```

```
% rE => average radius of Earth
% ------------------------------------------------------------

h_ref = 700;                % Reference Altitude
p_ref = 3.614*10^-14;       % Reference Density
h_scale = 88.667;           % Scale Height
altitude = norm(r_I) - rE;  % Altitude
density = p_ref*exp((h_ref-altitude)/h_scale);
end
```

9: AERO_density

```
function T_a = AERO(CBI, density, r_I, v_I, s_area, wE, CD, s_norm, s_ctrp)
% ------------------------------------------------------------
% AERO
% ------------------------------------------------------------
% Description:
% Compute aerodynamic perturbation torque.
% ------------------------------------------------------------
% Inputs:
% CBI => rotation matrix BI (3 x 3 matrix)
% density => atmospheric density
% r_I => orbital position in ECI (3 x 1 matrix)
% v_I => orbital velocity in ECI (3 x 1 matrix)
% ------------------------------------------------------------
% Outputs:
% T_a => aerodyanmic perturbation torque (3 x 1 matrix)
% ------------------------------------------------------------
% Parameters:
% wE => spin rate of Earth
% s_area => surface area of spacecraft panels
% s_norm => surface normals of spacecraft panels
% s_ctrp => center of pressure of spacecraft panels
% CD => drag coefficient
% ------------------------------------------------------------

T_a = [0;0;0];
v_rI = v_I - wE*r_I;

for s=1:length(s_area)
    cAlpha = transpose(s_norm(:,s))*CBI*(v_rI./norm(v_rI));
    if cAlpha <= 0
        continue
    end
    F_I = -0.5*density*CD*s_area(s)*v_rI*cAlpha;
    T_ak = skew(s_ctrp(:,s))*CBI*F_I;
    T_a = T_a + T_ak;
end
end
```

10: AERO

```
function T_g = GRAV(CBI, r_I, Jsc, mu)
% ------------------------------------------------------------
% GRAV
% ------------------------------------------------------------
% Description:
% Compute gravity gradient perturbation torque.
% ------------------------------------------------------------
% Inputs:
% CBI => rotation matrix BI (3 x 3 matrix)
```

```
% r_I => orbital position in ECI (3 x 1 matrix)
% -------------------------------------------------------------
% Outputs:
% T_g => gravity gradient perturbation torque
% -------------------------------------------------------------
% Parameters:
% Jsc => spacecraft inertia matrix
% mu => gravitational constant of the Earth
% -------------------------------------------------------------

r_B = CBI*r_I;
T_g = (3*mu/norm(r_I)^5)*skew(r_B)*Jsc*r_B;
end
```

11: GRAV

```
function T_s = SRP(t, r_I, JD0, s_norm, s_ctrp, CBI, s_area)
% -------------------------------------------------------------
% SRP
% -------------------------------------------------------------
% Description:
% Compute solar radiation pressure's perturbation torque.
% -------------------------------------------------------------
% Inputs:
% t => time [sec]
% CBI => rotation matrix BI (3 x 3 matrix)
% r_I => orbital position in ECI (3 x 1 matrix)
% -------------------------------------------------------------
% Outputs:
% T_s => solar radiation pressure's perturbation torque
% -------------------------------------------------------------
% Parameters:
% JD0 => julian date
% s_area => surface area of spacecraft panels
% s_norm => surface normals of spacecraft panels
% s_ctrp => center of pressure of spacecraft panels
% -------------------------------------------------------------

T_s = [0;0;0];

p = 4.51*10^-6;
JD = JD0 + 1.157*10^-5*t;
T_ut = (JD - 2451545)/36525;



lambda_ms = 280 + 36000.77*T_ut;
M = 357.5277233 + 35999.0534*T_ut;
e = 23.439291 - 0.0130042*T_ut;

lambda = lambda_ms + 1.914666471*sind(M) + 0.019994643*sind(2*M);
r = 1.000140612 - 0.016708617*cosd(M) + 0.019994643*sind(2*M);

r_s = r*[cosd(lambda); cosd(e)*sind(lambda); sind(e)*sind(lambda)];
s_I = (r_s - r_I)/norm(r_s - r_I);


for s=1:length(s_norm)
    cAlpha = transpose(s_norm(:,s))*CBI*s_I;
    if cAlpha <= 0
        continue
    end
```

```
    F_I = -p*s_area(s)*0.3*s_I*cAlpha;
    T_sk = skew(s_ctrp(:,s))*CBI*F_I;
    T_s = T_s + T_sk;
end
end
```

12: SRP

Appendix C - MATLAB Scripts

13: Skew Symmetric Function

```
function skewMatrix = skew(vec)

    skewMatrix = [0 -vec(3) vec(2); vec(3) 0 -vec(1); -vec(2) vec(1) 0];
end
```

14: Variable Initialization Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SPACECRAFT ATTITUDE DYNAMICS AND CONTROL - INITIALIZATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Steve Ulrich & Ahmed Moussa
% 6 December, 2023
% AERO 4540
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
close all
clear all

disp('Initalizing␣Parameters...')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Date and Astrodynamics Constants
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

YYYY    = 2023;
MM      = 10;
DD      = 9;
dYear   = decyear(YYYY,MM,DD);
C       = floor((MM-14)/12);
JD0     = DD - 32075 + floor(1461*(YYYY+4800+C)/4) + floor(367*(MM - 2 - 12*C)/12) -
    floor(0.75*floor((YYYY+4900+C)/100));

rE      = 6378.14;                  % Earth radius, km
wE      = 15.04*(pi/180)*(1/3600);  % Earth rotation speed, rad/s
mu      = 398600.4418;              % Gravitational parameter, km^3/s^2
p       = 0.00000451;               % Solar pressure constant, N/m^2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Orbital Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Gravitational parameter
mu = 398600.4418; % km^3/s^2

% J2 Constant
J2=0.00108264;

```

```
% Conversion constants
d2r = pi/180; % rad/deg
r2d = 1/d2r; % deg/rad

% Orbital elements
a       = rE + 713.5; % km
e       = 0.0014581;
i       = 98.2281*d2r; % rad
w       = 72.9288*d2r; % rad
RAAN    = 105.3095*d2r; % rad
tp      = 0; % sec

p = a*(1-e^2);

% Eccentric anomaly at t = 0 sec
eano   = acos((e+cos(0))/(1+e*cos(0))); % rad

% True anomaly at t = 0 sec
tano = 0; % rad

% Magnitude of position vector at t = 0 sec
r = p/(1+e*cos(0)); % km

RAAN_sec = -(3*pi*J2*rE^2*cos(i))/p^2;



% Components of position and velocity vectors in perifocal at t = 0 sec
r_P_ini = [r; 0; 0];

v_P_ini = [-sqrt(mu/p)*sin(0); sqrt(mu/p)*(e+cos(0)); 0];

% Rotation matrix from perifocal to ECI, i.e., C_IP

C_IP = [
    cos(RAAN)*cos(w)-sin(RAAN)*cos(i)*sin(w), -cos(RAAN)*sin(w)-sin(RAAN)*cos(i)*cos(w),
        sin(RAAN)*sin(i);
    sin(RAAN)*cos(w)+cos(RAAN)*cos(i)*sin(w), -sin(RAAN)*sin(w)+cos(RAAN)*cos(i)*cos(w),
        -cos(RAAN)*sin(i);
    sin(i)*sin(w), sin(i)*cos(w), cos(i);
];

% Components of position and velocity vectors in ECI at t = 0 sec
r_I_ini = C_IP*r_P_ini;
v_I_ini = C_IP*v_P_ini;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spacecraft Parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

RMM          = 0.5*[1;1;1]./sqrt(3);      % Residual Magnetic Moment, A m2;
RMM_skew     = skew(RMM);
CD           = 2.2;                       % Drag Coefficient
eta          = 0.3;                       % Absorption Coefficient

s_area = [0.595, 0.510, 0.420, 0.595, 0.510, 0.420];
s_norm = [[1;0;0], [0;1;0], [0;0;1], [-1;0;0], [0;-1;0], [0;0;-1]];
s_ctrp = [[0.3;0;-0.2], [0;0.35;-0.2], [0;0;0.425], [-0.3;0;-0.2], [0;-0.35;-0.2],
    [0;0;-0.425]];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spacecraft Platform Dynamics (without the wheels)
```

```
100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
101
102 Jsc          = [120 10 50; 10 150 -25; 50 -25 100];      % Spacecraft inertia (kg m^2)
103 Jsc_inv      = inv(Jsc);                                 % inverse of spacecraft inertia
    matrix (kg^-1 m^-2)
104 wsc_ini_B    = [0;0;0];                                  % components of initial
    spacecraft angular velocity vector wrt ECI in BOF (rad/s)
105 hsc_ini_B    = Jsc*wsc_ini_B;                            % components of initial
    spacecraft angular momentum vector in BOF (Nms)
106 qsc_I_ini    = [0.57;0.57;0.57;0.159];                   % initial quaternion
107
108 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
109 % Reaction Wheel 1 about Spin Axis a1
110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
111
112 Ja1         = 0.00041;                % wheel 1 moment of inertia about its spin axis, kg
     m^2
113 Ja1_inv     = 1/Ja1;                  % inverse of wheel 1 inertia, kg^-1 m^-2
114 wa1_ini     = 0;                      % wheel 1 initial angular rate about its spin axis
    wrt ECI, rad/s
115 ha1_ini     = 0;                      % wheel 1 initial angular momentum about its spin
    axis, Nms
116 a1_B        = [1;0;0];                % components of wheel 1 spin axis in BOF
117 ha1_B_ini   = Ja1*(wa1_ini+a1_B.'*wsc_ini_B)*a1_B;    % components of initial angular
    momentum vector of wheel 1 in BOF, Nms
118 % NOTE: this calculation exists in case different initial angular velocities besides 0
    are defined.
119
120 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121 % Reaction Wheel 2 about Spin Axis a2
122 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
123
124 Ja2         = 0.000411;               % wheel 2 moment of inertia about its spin axis, kg
     m^2
125 Ja2_inv     = 1/Ja2;                  % inverse of wheel 2 inertia, kg^-1 m^-2
126 wa2_ini     = 0;                      % wheel 2 initial angular rate about its spin axis
    wrt ECI, rad/s
127 ha2_ini     = 0;                      % wheel 2 initial angular momentum about its spin
    axis, Nms
128 a2_B        = [0;1;0];                % components of wheel 2 spin axis in BOF
129 ha2_B_ini   = Ja1*(wa2_ini+a2_B.'*wsc_ini_B)*a2_B;    % components of initial angular
    momentum vector of wheel 2 in BOF, Nms
130 % NOTE: this calculation exists in case different initial angular velocities besides 0
    are defined.
131
132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
133 % Reaction Wheel 3 about Spin Axis a3
134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
135
136 Ja3         = 0.000412;               % wheel 3 moment of inertia about its spin axis, kg
     m^2
137 Ja3_inv     = 1/Ja3;                  % inverse of wheel 3 inertia, kg^-1 m^-2
138 wa3_ini     = 0;                      % wheel 3 initial angular rate about its spin axis
    wrt ECI, rad/s
139 ha3_ini     = 0;                      % wheel 3 initial angular momentum about its spin
    axis, Nms
140 a3_B        = [0;0;1];                % components of wheel 3 spin axis in BOF
141 ha3_B_ini   = Ja3*(wa3_ini+a3_B.'*wsc_ini_B)*a3_B;    % components of initial angular
    momentum vector of wheel 3 in BOF, Nms
142 % NOTE: this calculation exists in case different initial angular velocities besides 0
    are defined.
143
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Total Angular Momentum (spacecraft platform + wheels)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

htot_B_ini  = hsc_ini_B + ha1_B_ini + ha2_B_ini + ha3_B_ini;    % components of initial
    total angular momentum vector in BOF, Nms

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Attitude Sensors
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

wrel_noise = 2*pi/60;               % tachometer noise in rad/s (1 RPM)
wrel_var = wrel_noise^2;            % tachometer variance

qsc_noise = 1/(60*60) * pi/180;     % star tracker noise in rad (1 arcsec)
qsc_var = qsc_noise^2;              % star tracker variance

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller Gain Selection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

qsc_d   = [0;0;0;1];                % desired quaternion
J = diag(Jsc);                      % diagonal of spacecraft inertia matrix

w_n = 0.156;                        % desired natural frequency
t_s = 50;                           % desired settling time

Kd = (J*16/t_s).*eye(3);            % feedback gain matrix

% [NOTE] For simplicity, 3x1 column matricies are used in computation of
% Kp instead of 3x3. Resulting Kp array will still be a 3x3 matrix. I would've
% rather just use a Kp column matrix and element-wise multiplication throughout
% the whole simulation for efficiency, but I don't know if I'd lose marks for that.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inverse Inertia Design

k = 300;
Kp_1 = (k ./ J).*eye(3);
Kp_1 = Kp_1*11/Kp_1(2,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scaled Identity Design

k = 20;
Kp_2 = k * eye(3);
% Normalize
Kp_2 = Kp_2*11/Kp_2(2,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Alpha-Beta Eigenaxis Design

alpha = (9 - sum(1./J) * sum(J)) / (3 * sum(J.^2) - sum(J)^2);
beta = (sum(1./J) * sum(J.^2) - 3 * sum(J)) / (3 * sum(J.^2) - sum(J)^2);
Kp_3 = inv((alpha*J + beta).*eye(3));
% Normalize
Kp_3 = Kp_3*11/Kp_3(2,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Time Specification Eigenaxis Design

Kp_4 = J*2*w_n^2.*eye(3);
```

```
% Normalize
Kp_4 = Kp_4*11/Kp_4(2,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run Simulation and Plot Results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
run_SPACECRAFT_ATTITUDE
```

15: Simulation Execution and Results Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SPACECRAFT ATTITUDE DYNAMICS AND CONTROL - RUN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Steve Ulrich & Ahmed Moussa
% 6 December, 2023
% AERO 4540
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run Simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

T = 100;                                                % Simulation stop time (sec)
% [NOTE] Simulation time of 6000 for full orbit.

set_param('SPACECRAFT_ATTITUDE', 'StopTime', 'T')       % Sets the simulation stop time
disp('Running␣Simulations...')                          % Display message

Kp = Kp_1;
sim('SPACECRAFT_ATTITUDE')                              % Run the Simulink model
qsc_I_1 = qsc_I_mes;

Kp = Kp_2;
sim('SPACECRAFT_ATTITUDE')                              % Run the Simulink model
qsc_I_2 = qsc_I_mes;

Kp = Kp_4;
sim('SPACECRAFT_ATTITUDE')                              % Run the Simulink model
qsc_I_4 = qsc_I_mes;

Kp = Kp_3;
sim('SPACECRAFT_ATTITUDE')                              % Run the Simulink model
qsc_I_3 = qsc_I_mes;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot Results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
disp('Plotting␣Results...')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Open-Loop Wheels' Torques

figure
set(gcf, 'Position', [0 0 900 675])
subplot(3,1,1)
plot(t,Ta1,'k')
axis([0 100 -0.5 0.5])
```

```
xlabel('Time (sec)')
ylabel('Wheel 1 Torque (N\cdotm)')
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(3,1,2)
plot(t,Ta2,'k')
axis([0 100 -0.5 0.5])
xlabel('Time (sec)')
ylabel('Wheel 2 Torque (N\cdotm)')
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(3,1,3)
plot(t,Ta3,'k')
axis([0 100 -0.5 0.5])
xlabel('Time (sec)')
ylabel('Wheel 3 Torque (N\cdotm)')
set(gca,'FontSize',9,'FontName', 'Times')
grid on



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Spacecraft Quaternions and Angular Rates

figure
set(gcf, 'Position', [0 0 900 900])
subplot(4,2,1)
plot(t,qsc_I(:,1),'k');
xlabel('Time (sec)')
ylabel('q_1')
axis([0 100 -1 1])
hold on
plot(t,qsc_I_mes(:,1),'r--');
legend('Actual','Measured')
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(4,2,3)
plot(t,qsc_I(:,2),'k');
xlabel('Time (sec)')
ylabel('q_2')
axis([0 100 -1 1])
hold on
plot(t,qsc_I_mes(:,2),'r--');
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(4,2,5)
plot(t,qsc_I(:,3),'k');
xlabel('Time (sec)')
ylabel('q_3')
axis([0 100 -1 1])
hold on
plot(t,qsc_I_mes(:,3),'r--');
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(4,2,7)
plot(t,qsc_I(:,4),'k');
xlabel('Time (sec)')
ylabel('q_4')
axis([0 100 0 0.6])
hold on
plot(t,qsc_I_mes(:,4),'r--');
set(gca,'FontSize',9,'FontName', 'Times')
```

```
grid on
subplot(4,2,2)
plot(t,wsc_B(:,1),'k');
xlabel('Time (sec)')
ylabel('\omega_x (rad/s)')
axis([0 100 -0.1 0.1])
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(4,2,4)
plot(t,wsc_B(:,2),'k');
xlabel('Time (sec)')
ylabel('\omega_y (rad/s)')
axis([0 100 -0.1 0.1])
set(gca,'FontSize',9,'FontName', 'Times')
grid on
subplot(4,2,6)
plot(t,wsc_B(:,3),'k');
xlabel('Time (sec)')
ylabel('\omega_z (rad/s)')
axis([0 100 -0.1 0.1])
set(gca,'FontSize',9,'FontName', 'Times')
grid on



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Orbit Plots

orb_distance = squeeze(vecnorm(orb_position));
orb_speed = squeeze(vecnorm(orb_velocity));

figure
set(gcf, 'Position', [0 0 900 900])
subplot(2,1,1)
xlabel('Time [sec]')
ylabel('Distance [km]')
hold on
plot(t,orb_distance,'k','LineWidth',1.5);
set(gca,'FontSize',9,'FontName', 'Times')
grid on

subplot(2,1,2)
xlabel('Time [sec]')
ylabel('Speed [km/s]')
hold on
plot(t,orb_speed,'k','LineWidth',1.5);
set(gca,'FontSize',9,'FontName', 'Times')
grid on



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% External Torque Plot

T_ex_magnitude = squeeze(vecnorm(T_ex));

figure
set(gcf, 'Position', [0 0 900 900])
xlabel('Time [sec]')
ylabel('Torque [Nm]')
hold on
plot(t,T_ex_magnitude,'k','LineWidth',1.0);
```

```
set(gca,'FontSize',9,'FontName', 'Times')
grid on



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% qi vs. qj Plots

figure
set(gcf, 'Position', [0 0 900 900])
subplot(3,1,1)
xlabel('q_1')
ylabel('q_2')
hold on
plot(qsc_I_1(:,1),qsc_I_1(:,2),'k','LineWidth',1.0);
plot(qsc_I_2(:,1),qsc_I_2(:,2),'c','LineWidth',1.0);
plot(qsc_I_3(:,1),qsc_I_3(:,2),'r','LineWidth',1.0);
plot(qsc_I_4(:,1),qsc_I_4(:,2),'g','LineWidth',1.0);
legend('Inverse␣Inertia','Scaled␣Identity', 'Alpha-Beta␣Eigenaxis', 'Natural␣Frequency␣
    Eigenaxis')
set(gca,'FontSize',9,'FontName', 'Times')
grid on

subplot(3,1,2)
xlabel('q_1')
ylabel('q_3')
hold on
plot(qsc_I_1(:,1),qsc_I_1(:,3),'k','LineWidth',1.0);
plot(qsc_I_2(:,1),qsc_I_2(:,3),'c','LineWidth',1.0);
plot(qsc_I_3(:,1),qsc_I_3(:,3),'r','LineWidth',1.0);
plot(qsc_I_4(:,1),qsc_I_4(:,3),'g','LineWidth',1.0);
set(gca,'FontSize',9,'FontName', 'Times')
grid on

subplot(3,1,3)
xlabel('q_2')
ylabel('q_3')
hold on
plot(qsc_I_1(:,2),qsc_I_1(:,3),'k','LineWidth',1.0);
plot(qsc_I_2(:,2),qsc_I_2(:,3),'c','LineWidth',1.0);
plot(qsc_I_3(:,2),qsc_I_3(:,3),'r','LineWidth',1.0);
plot(qsc_I_4(:,2),qsc_I_4(:,3),'g','LineWidth',1.0);
set(gca,'FontSize',9,'FontName', 'Times')
grid on
```