Final Project - ASE 389P.8 Satellite Control Systems

Corey Marcus

April 24, 2023

1 Introduction

An eccentric billionaire has built his own space station. To save on design costs, his station is an exact replica of the ISS. He does not intend to share it with anyone, so it was scaled down in size such that its inertia matrix is smaller than that of the ISS by a factor of 1E-4. However, he demanded the station be as fast and nimble as his companies (he is a disruptor). The station's CMGs have the performance as the full size ISS. Super cooled bearings made of unobtainium mean there is no limit on how fast the gimbals can spin.

2 Attitude Dynamics and Kinematics

The station lives in an equatorial 400km circular orbit. Its attitude is perturbed by gravity gradient. The angular inertia matrix is given by

$$\boldsymbol{J} = 10^{-4}\begin{bmatrix} 24181836 & 3783405 & 3898808 \\ 3783405 & 37621803 & -1171849 \\ 3898808 & -1171849 & 51576634 \end{bmatrix} \tag{1}$$

3 Reference Attitude

The station is generally prescribed to remain fixed with respect to the LVLH frame. It begins with a nominal quaternion $\begin{bmatrix} 0.028 & -0.0788 & 0.1141 & 0.9899 \end{bmatrix}$ from LVLH to body and angular rate with respect to the LVLH frame of zero. Through the course of a 100 run Monte Carlo (MC), the nominal quaternion and angular rate are dispersed using zero-mean gaussian noise with covariance $10^{-2}\boldsymbol{I}_{3\times3}$ and $10^{-3}\boldsymbol{I}_{3\times3}$.

The station is commanded to rotate to quaternion $\begin{bmatrix} 0.0481 & -0.0500 & 0.1420 & 0.9874 \end{bmatrix}$ with respect to the LVLH frame and then remain fixed. This corresponds to a 3 degree rotation about each body axis. The reference rotation is designed as a constant angular rate euler-axis rotation.

Figures 1 and 2 show the estimated command errors, or difference between the estimated pose and desired pose.

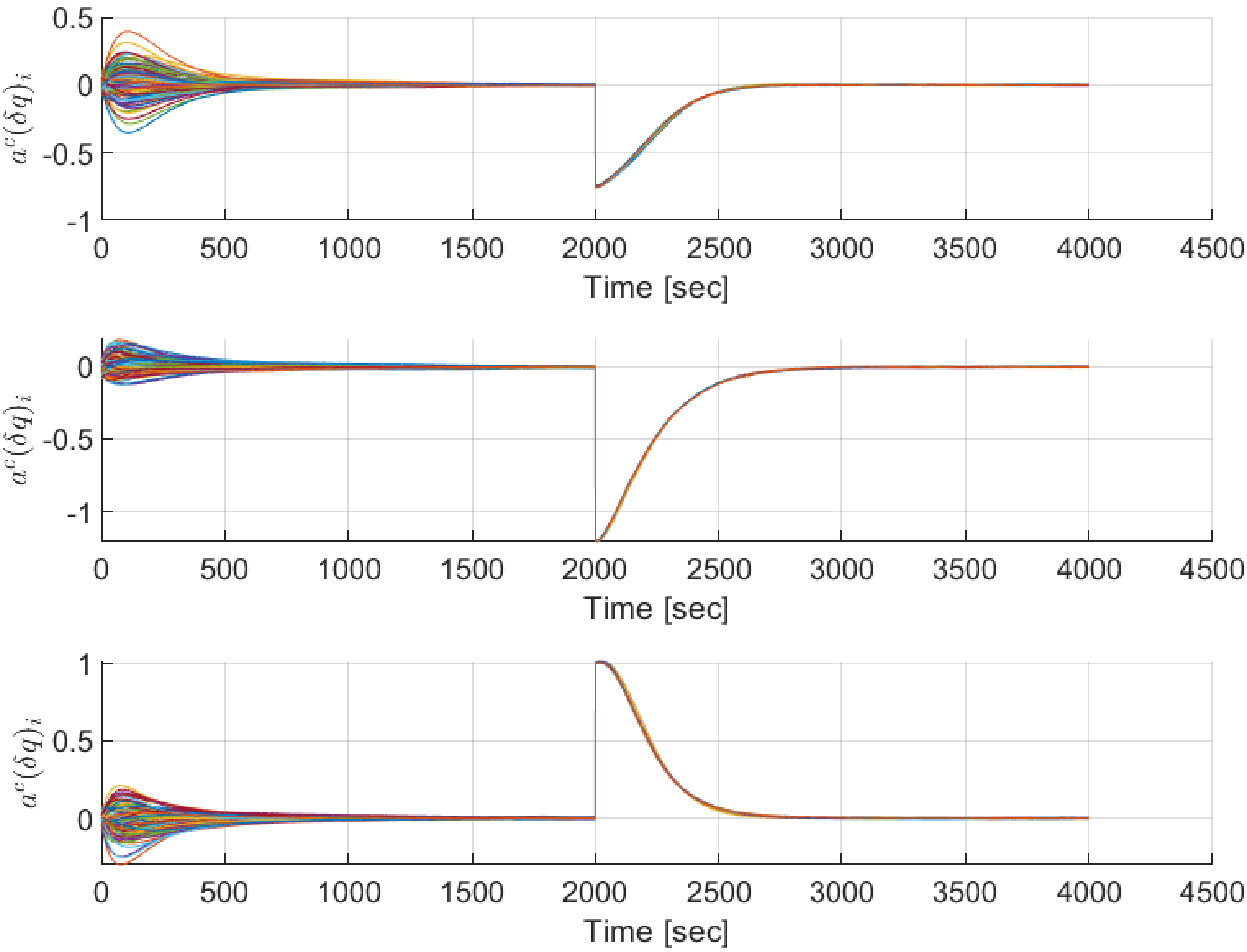


Figure 1: The error in commanded inertial to body attitude for each MC run, expressed as a parameterization of the error quaternion. We begin with a constant rate maneuver, followed by a LVLH hold.

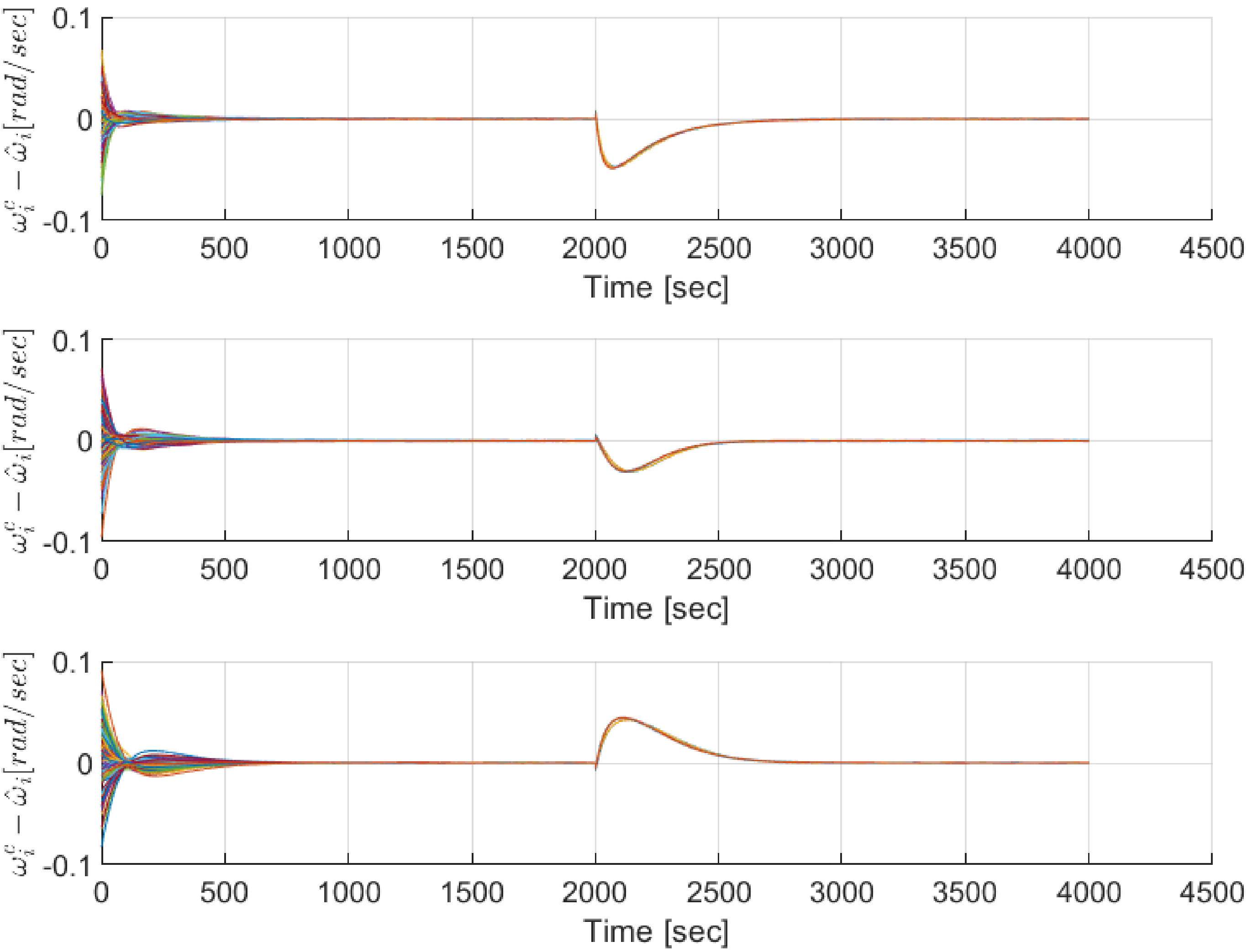


Figure 2: The error in commanded body rate with respect to the inertial frame for each MC run. We begin with a constant rate maneuver, followed by a LVLH hold.

4 Actuators

The station's actuators are CMGs which are exact replicas of those on the ISS. They have been upgraded to support infinite gimbal rotation rates. Our billionaire decided that electro-magnetic shielding on many critical components was unnecessary. This has a negative effect of introducing noise on the commanded torque for the CMGs. This is modeled as zero-mean Gaussian noise with covariance $10^{-6}\boldsymbol{I}_{3\times3}$.

The orientations of the CMGs are identical to those described in the first class midterm.

Figure 3 shows CMG gimbal rates for one sample MC run.

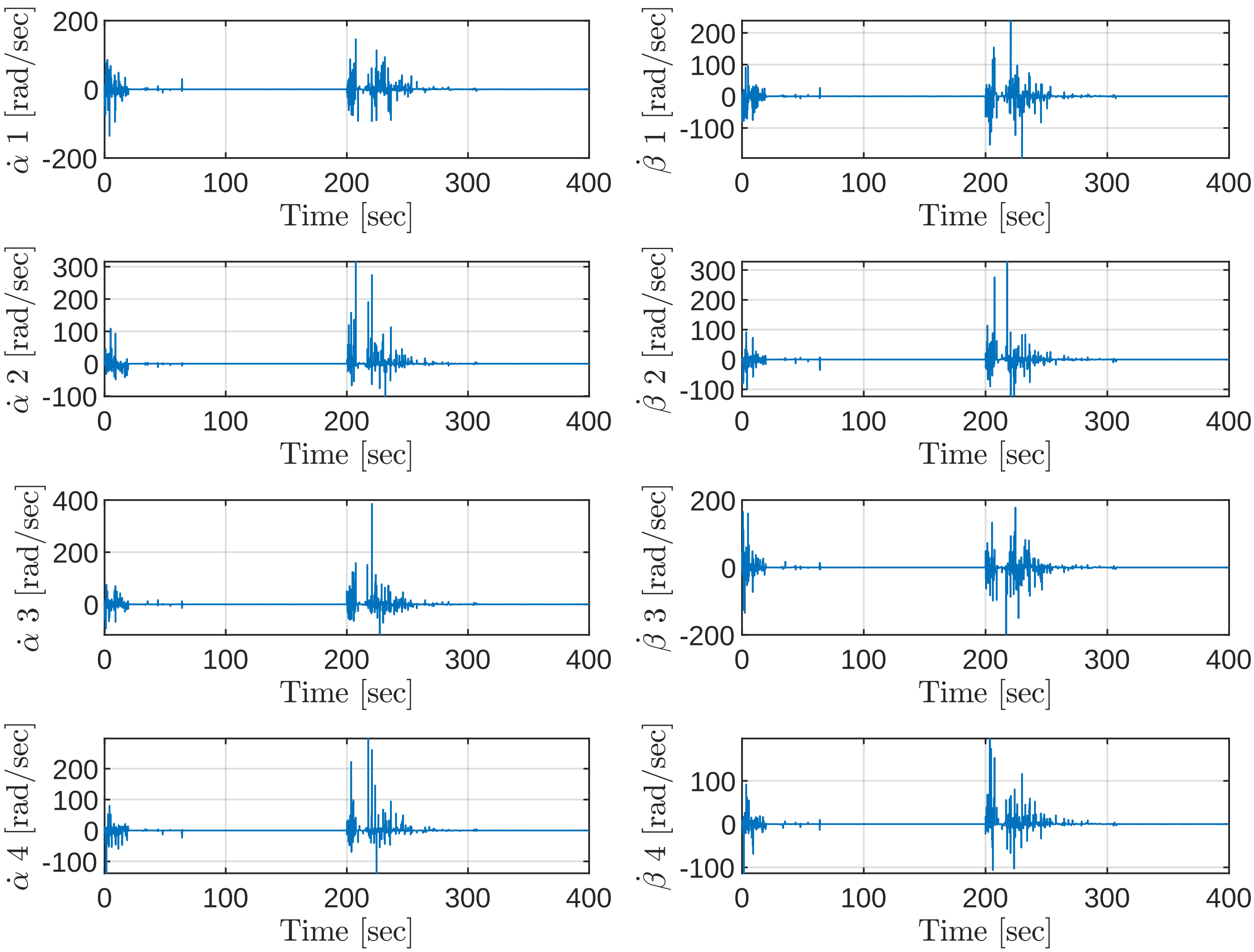


Figure 3: The CMG gimbal rates for the first MC run.

5 Sensors

The station's sensors include a gyroscope and star tracker. Neither sensor contains a bias and their true alignment is known. Both measurements occur at 10 Hz and are corrupted by zero-mean Gaussian noise. The noise covariance is $10^{-3}\boldsymbol{I}_{3\times3}$ for both sensors.

6 MEKF

The vehicle's true position and velocity are known. But it must use an MEKF to estimate its inertial to body attitude and angular rates. Both gyroscope and star tracker measurements are fused into the state during the update portion. The vehicle state is propagated using the torque command. We model process noise with covariance blkdiag$\{10^{-8}\boldsymbol{I}_{3\times3},\, 3.5\times10^{-11}\boldsymbol{I}_{3\times3}\}$. We implement underweighting by finding our Kalman Gain as $\boldsymbol{K} = \boldsymbol{P}\boldsymbol{H}^T\left((1+\beta)\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^T + \boldsymbol{R}\right)^{-1}$ where $\beta = 0.5$.

The initial state error is sampled from a zero-mean gaussian distribution with covariance blkdiag$\{10^{-3}\boldsymbol{I}_{3\times3},\, 10^{-4}\boldsymbol{I}_{3\times3}\}$.

Figures 4 and 5 show our MEKF performance. Our attitude representation is double the vector portion of the error quaternion. We see that our MEKF is unbiased, and slightly smug.

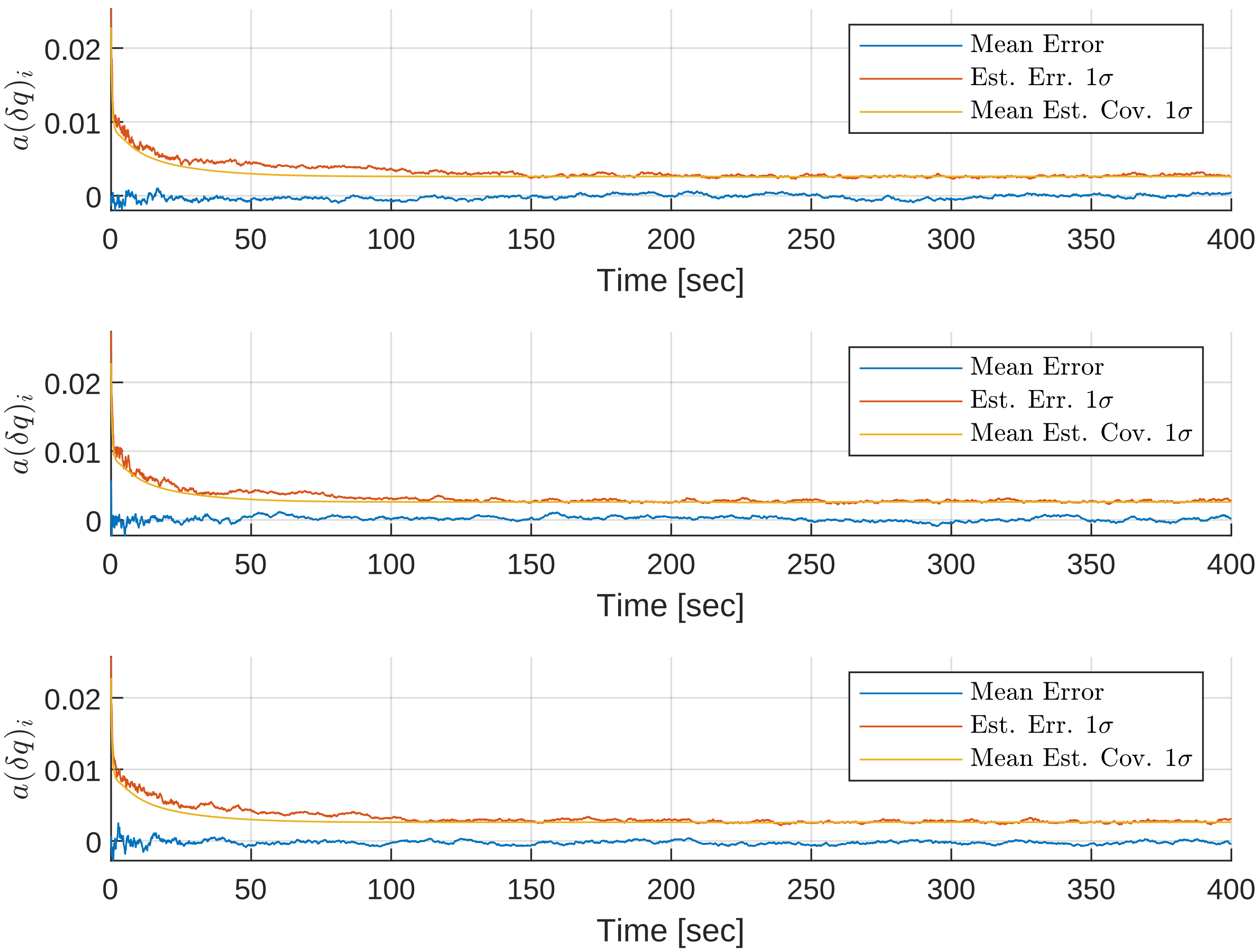


Figure 4: The attitude estimation performance as shown by the average estimate error, sample error covariance, and average estimate covariance. We use an error parameterization which is double the vector component of the error quaternion.

7 Code

```
%% Setup
clear
close all
clc

addpath("..\util\","..\..\matlabScripts")

modelname = 'project_sim';

%% Symbolic math for some functions
perform_symbolic = true;
if(perform_symbolic)
    w_hat = sym('w_hat',[3 1],'real');
    syms dt tau 'real'

    wx = CrossProductMat(w_hat);
    A = ExpmSkewSym(-wx*dt);
    B = int(ExpmSkewSym(-wx*(dt - tau)),tau,0,dt);

    matlabFunction(A,'File','PaPaFunc.m');
    matlabFunction(B,'File','PaPdomegaFunc.m');
end

%% Problem Initalization

% Number of monte carlos
N_MC = 100;

% Noise parameters
Sigma_a = 1E-3*eye(3); % Measurement noise for q_inertial2body
Sigma_w = 1E-3*eye(3); % Measurement noise for body angular rate
underweight = 0.5;

% Initial state dispersion parameters
Disp_a = 1E-2*eye(3);
Disp_w = 1E-3*eye(3);

% Vehicle inertia matrix
J = 1E-4*[24181836 3783405 3898808
    3783405 37621803 -1171849
    3898808 -1171849 51576634];

% Actuator noise
Sigma_act = 1E-6*eye(3); % Noise on torque commands to acctuator
w_noise_scale = max(J\diag(Sigma_act));

% Process noise for filter
Q_filter = blkdiag(1E-8*eye(3),0.1*w_noise_scale*eye(3));

% Initial uncertainty for filter
Phat0 = blkdiag(1E-3*eye(3),1E-4*eye(3));
```

```
% Flight software frequency
FSW_freq = 1/10;

% Gravity
mu = 398600; %km3/s2

% Orbital elements
a = 6371 + 400;
e = 0;
i = 0;
Ohm = 0;
w = 0;
theta = 0;
[r_inertial_0, v_inertial_0] = OE2State(a, e, i, Ohm, w, theta);

% Find orbit rate
n = sqrt(mu/a^3);

% Find LVLH rotation rate
w_LVLH_wrt_inertial_in_LVLH = [0, -n, 0]';

% Initial rotation between LVLH and inertial
x_LVLH_inertial_0 = v_inertial_0/norm(v_inertial_0);
z_LVLH_inertial_0 = -r_inertial_0/norm(r_inertial_0);
y_LVLH_inertial_0 = cross(z_LVLH_inertial_0,x_LVLH_inertial_0);
T_inertial2LVLH_0 = [x_LVLH_inertial_0'; y_LVLH_inertial_0'; z_LVLH_inertial_0'];
q_inertial2LVLH_0 = DCM2Quat(T_inertial2LVLH_0);

% Rotation formulations
w_b_LVLH_0 = [0 0 0]'; % Initial LVLH rotation rate, rad/sec
q_LVLH2body_0 = [0.028, -0.0788, 0.1141, 0.9899]'; % Initial attitude quaternion
%q_LVLH2body_f = q_LVLH2body_0;
R_change = angle2dcm(3*pi/180,3*pi/180,3*pi/180);
q_change = DCM2Quat(R_change);
% q_LVLH2body_f = [-0.0607, -0.0343, -0.7045, 0.7062]'; % Attitude quaternion at end of↙
the manuever
q_LVLH2body_f = QuatProduct(q_change,q_LVLH2body_0);

% Initial pose and rate in inertial
q_inertial2body_0 = QuatProduct(q_LVLH2body_0,q_inertial2LVLH_0);
w_body_wrt_inertial_0 = QuatTransform(q_LVLH2body_0,w_LVLH_wrt_inertial_in_LVLH) +↙
w_b_LVLH_0;

% Final simulation time
Tf = 400;
% Tf = 100;
% Tf = 10000;

% Final manuever time
Tf_man = 200;
```

```
% CMG momentum
h0 = 4881;

% Maximum CMG rates
rate_max = Inf*(pi/180); % Rad/sec


%% Nonlinear controller design

kp_nonlin = 100;
kd_nonlin = 1000;

%% Storage
out_data = cell(N_MC,1);

%% Run MC

for ii = 97:N_MC

    fprintf("MC Iteration: %d / %d \n",ii,N_MC)
    rng(ii+18)

    % Sample randomness
    tsample = 0:FSW_freq:Tf;
    Nsample = length(tsample);
    q_meas_noise = mvnrnd(zeros(3,1),Sigma_a,Nsample);
    w_meas_noise = mvnrnd(zeros(3,1),Sigma_w,Nsample);
    act_noise = mvnrnd(zeros(3,1),Sigma_act,Nsample);
    simin = [];
    simin.act_noise = timeseries(act_noise,tsample);
    simin.q_meas_noise = timeseries(q_meas_noise,tsample);
    simin.w_meas_noise = timeseries(w_meas_noise,tsample);
    init_err = mvnrnd(zeros(6,1),Phat0)';
    dq0 = [0.5*init_err(1:3); 1];
    dq0 = dq0/norm(dq0);

    init_disp = mvnrnd(zeros(6,1),blkdiag(Disp_a,Disp_w))';
    dq0_disp = [0.5*init_disp(1:3); 1];
    dq0_disp = dq0_disp/norm(dq0_disp);

    % Disperse true initial state
    q_inertial2body_0_est = QuatProduct(dq0_disp,q_inertial2body_0);
    w_body_wrt_inertial_0_est = w_body_wrt_inertial_0 + init_disp(4:6);
    q_LVLH2body_0_est = QuatProduct(q_inertial2body_0_est,QuatInv(q_inertial2LVLH_0));

    % Corrupt initial state certainty
    q_inertial2body_0_true = QuatProduct(dq0,q_inertial2body_0_est);
    w_body_wrt_inertial_0_true = w_body_wrt_inertial_0_est + init_err(4:6);

    % Design the maneuver in the LVLH frame
```

```
    % Change in quaternion
    dq_LVLH = QuatProduct(q_LVLH2body_f,QuatInv(q_inertial2body_0_est));

    % Euler axis and angle change
    [dtheta_LVLH, dn_LVLH] = Quat2AxisAngle(dq_LVLH);

    % Find angular rate in rad/sec
    w_b_LVLH_man = dtheta_LVLH/Tf_man*dn_LVLH;

    % Run sim
    out_data{ii} = sim(modelname);
    save("data2.mat","out_data","-v7.3")

end
%% Extract Information From First Run

w_body_inertial = out_data{1}.w;
w_body_inertial_est = out_data{1}.w_body_est;
w_body_inertial_meas = out_data{1}.w_body_meas;
q_inertial2LVLH = out_data{1}.q_inertial2LVLH;
q_inertial2body = out_data{1}.quat;
q_inertial2body_est = out_data{1}.q_inertial2body_est;
q_inertial2body_meas = out_data{1}.q_inertial2body_meas;
CMG_rates = out_data{1}.CMG_rates;
% err_quat = squeeze(out_data{1}.error_quat)';
w_body_inertial_ref = out_data{1}.ref_rate';
CMG_h = squeeze(out_data{1}.CMG_h)';

% Squeeze for some reason
q_inertial2body_est.Data = squeeze(q_inertial2body_est.Data)';
w_body_inertial_est.Data = squeeze(w_body_inertial_est.Data)';
q_inertial2body_meas.Data = squeeze(q_inertial2body_meas.Data)';
% w_body_inertial_meas.Data = squeeze(w_body_inertial_meas.Data)';

% % Find quaternion from body to LVLH
% q_LVLH2body = zeros(size(q_inertial2LVLH));
% for ii = 1:length(tout)
%     q_LVLH2body(:,ii) = QuatProduct(q_inertial2body(:,ii),QuatInv(q_inertial2LVLH(:, ↙
ii)));
% end

%% Extract MC Statistics

% Initialize data storage
Ntime = length(q_inertial2body_est.Time);
est_err_quats = zeros(3,Ntime,N_MC);
command_err_quats = zeros(3,Ntime,N_MC);
est_err_w = zeros(3,Ntime,N_MC);
command_err_w = zeros(3,Ntime,N_MC);
mean_est_err_quat = zeros(3,Ntime);
mean_command_err_quat = zeros(3,Ntime);
```

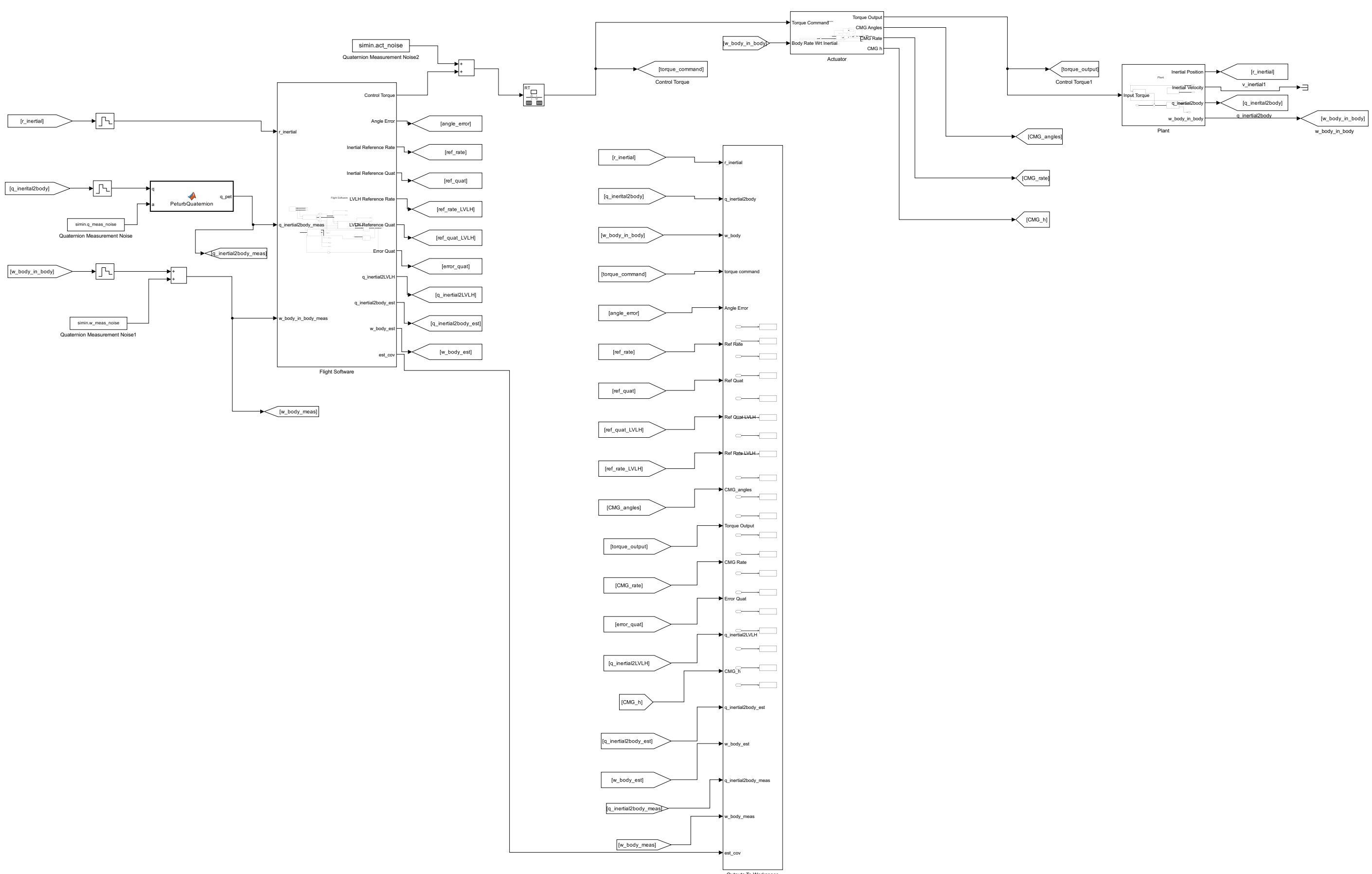
```
var_est_err_quat = zeros(3,Ntime);
disp_command_err_quat = zeros(3,Ntime);
mean_est_var_quat = zeros(3,Ntime);
mean_est_err_w = zeros(3,Ntime);
mean_command_err_w = zeros(3,Ntime);
var_est_err_w = zeros(3,Ntime);
disp_command_err_w = zeros(3,Ntime);
mean_est_var_w = zeros(3,Ntime);
est_cov = zeros(6,6,Ntime,N_MC);
for jj = 1:Ntime
    for ii = 1:N_MC
        q_est = squeeze(out_data{ii}.q_inertial2body_est.Data(:,:,jj));
        q_true = out_data{ii}.quat.Data(jj,:)';
        q_ref = out_data{ii}.ref_quat.Data(jj,:)';
        q_est_err = QuatProduct(QuatInv(q_est),q_true);
        q_com_err = QuatProduct(QuatInv(q_true),q_ref);
        est_err_quats(:,jj,ii) = 2*q_est_err(1:3);
        command_err_quats(:,jj,ii) = 2*q_com_err(1:3);

        w_est = squeeze(out_data{ii}.w_body_est.Data(:,:,jj));
        w_true = out_data{ii}.w.Data(jj,:)';
        w_ref = out_data{ii}.ref_rate.Data(jj,:)';
        est_err_w(:,jj,ii) = w_est - w_true;
        command_err_w(:,jj,ii) = w_true - w_ref;
        est_cov(:,:,jj,ii) = out_data{ii}.est_cov.Data(:,:,jj);
    end

    mean_est_err_quat(:,jj) = mean(squeeze(est_err_quats(:,jj,:)),2);
    mean_est_err_w(:,jj) = mean(squeeze(est_err_w(:,jj,:)),2);
    mean_command_err_quat(:,jj) = mean(squeeze(command_err_quats(:,jj,:)),2);
    mean_command_err_w(:,jj) = mean(squeeze(command_err_w(:,jj,:)),2);

    var_est_err_quat(:,jj) = var(squeeze(est_err_quats(:,jj,:)),0,2)';
    var_est_err_w(:,jj) = var(squeeze(est_err_w(:,jj,:)),0,2)';
    disp_command_err_quat(:,jj) = var(squeeze(command_err_quats(:,jj,:)),0,2)';
    disp_command_err_w(:,jj) = var(squeeze(command_err_w(:,jj,:)),0,2)';

    mean_est_var_quat(:,jj) = diag(mean(est_cov(1:3,1:3,jj,:),4));
    mean_est_var_w(:,jj) = diag(mean(est_cov(4:6,4:6,jj,:),4));


end

save("data2.mat","-v7.3")


%% Plotting Part 3

figure
for ii = 1:3
    subplot(3,1,ii)
```

```
    plot(out_data{1}.error_quat.Time,out_data{1}.error_quat.Data(:,ii),"LineWidth",2)
    xlabel('Time [s]',"Interpreter","latex")
    ylabel("$\delta q$","Interpreter","latex")
    grid on
    title("Estimated Pointing Error")
end

figure
for ii = 1:4
    subplot(4,1,ii)
    hold on
    plot(q_inertial2body.Time, q_inertial2body.Data(:,ii),"LineWidth",2)
    plot(q_inertial2body_est.Time, q_inertial2body_est.Data(:,ii),"LineWidth",2)
%     plot(q_inertial2body_meas.Time, q_inertial2body_meas.Data(:,ii),"LineWidth",2)
    xlabel('Time [s]',"Interpreter","latex")
    ylabel('Quat Element',"Interpreter","latex")
%     legend("Actual","Estimate","Meas")
legend("Actual",'Estimate')
    grid on
end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(w_body_inertial.Time, w_body_inertial.Data(:,ii),"LineWidth",2)
    plot(w_body_inertial_est.Time, w_body_inertial_est.Data(:,ii),"LineWidth",2)
%     plot(w_body_inertial_meas.Time, w_body_inertial_meas.Data(:,ii),"LineWidth",2)
    xlabel('Time [s]',"Interpreter","latex")
    ylabel('Angular Rate',"Interpreter","latex")
%     legend("Actual","Estimate","Meas")
    legend("Actual","Estimate")
    grid on
end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(w_body_inertial.Time, w_body_inertial_ref.Data(:,ii) - w_body_inertial.Data(:,↙
ii),"LineWidth",2)
    xlabel('Time [s]',"Interpreter","latex")
    ylabel('$\delta \omega$ [rad/sec]',"Interpreter","latex")
%     legend("Actual","Estimate")
    grid on
end

figure
for ii = 1:4
    subplot(4,2,2*ii-1)
    plot(CMG_rates.Time,CMG_rates.Data(:,ii))
```

```
    xlabel('Time [sec]',"Interpreter","latex")
    ylabel(strcat("$\dot{\alpha}$ ",num2str(ii)," [rad/sec]"),"Interpreter",'latex')
    grid on

    subplot(4,2,2*ii)
    plot(CMG_rates.Time,CMG_rates.Data(:,ii+4))
    xlabel('Time [sec]',"Interpreter","latex")
    ylabel(strcat("$\dot{\beta}$ ",num2str(ii)," [rad/sec]"),"Interpreter",'latex')
    grid on

end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(CMG_h.Time, abs(CMG_h.Data(:,ii)),"LineWidth",2)
    xlabel('Time [s]',"Interpreter","latex")
    ylabel('$|h_{CMG}|$ [kg-m\textsuperscript{2}/sec]',"Interpreter","latex")
    grid on
end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(CMG_rates.Time,mean_est_err_quat(ii,:))
    plot(CMG_rates.Time,sqrt(var_est_err_quat(ii,:)))
    plot(CMG_rates.Time,sqrt(mean_est_var_quat(ii,:)))
    ylabel('$a(\delta q)_i$','Interpreter','latex')
    xlabel('Time [sec]')
    grid on
    %title("MC Quat Estimation Performance")
    legend("Mean Error","Est. Err. $1\sigma$","Mean Est. Cov.↙
$1\sigma$",'Interpreter','latex')

end
saveas(gcf,"MC_quat_est_perf.pdf")

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(CMG_rates.Time,mean_est_err_w(ii,:))
    plot(CMG_rates.Time,sqrt(var_est_err_w(ii,:)))
    plot(CMG_rates.Time,sqrt(mean_est_var_w(ii,:)))
    grid on
%     title("MC Body Rate Estimation Performance")
    ylabel('$\omega_i - \hat{\omega}_i$ [rad/sec]','Interpreter','latex')
    xlabel('Time [sec]')
    legend("Mean Error","Est. Err. $1\sigma$","Mean Est. Cov.↙
```

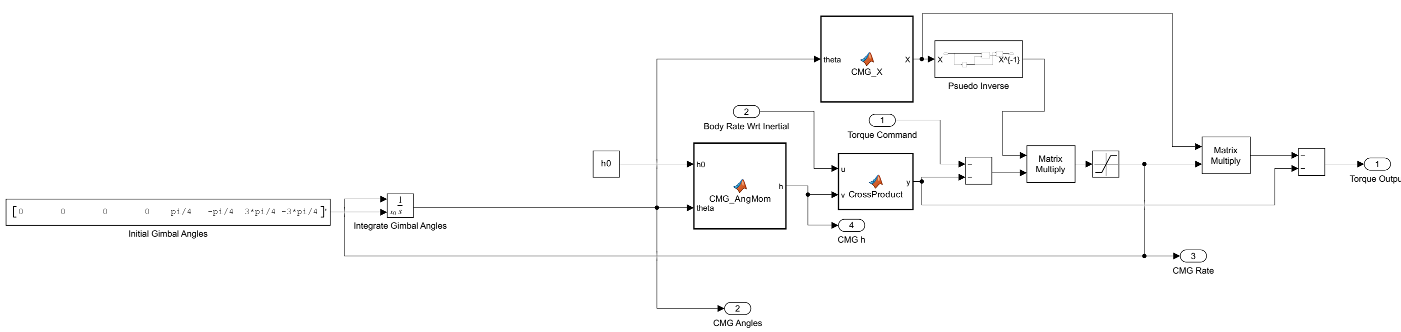
```
$1\sigma$",'Interpreter','latex')
end
saveas(gcf,"MC_rate_est_perf.pdf")

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(CMG_rates.Time,mean_command_err_quat(ii,:))
    plot(CMG_rates.Time,disp_command_err_quat(ii,:))
    title("MC Quat Control Performance")
    legend("Mean Error","Err Disp")


end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    plot(CMG_rates.Time,mean_command_err_w(ii,:))
    plot(CMG_rates.Time,disp_command_err_w(ii,:))
    title("MC Ang Rate Control Performance")
    legend("Mean Error","Err Disp")
end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    for jj = 1:N_MC
        plot(est_err_quats(ii,:,jj))
    end
    title("Quat Est Err Traces")
end

figure
for ii = 1:3
    subplot(3,1,ii)
    hold on
    for jj = 1:N_MC
        plot(command_err_quats(ii,:,jj))

    end
    ylabel('$a^c(\delta q)_i$','Interpreter','latex')
    xlabel('Time [sec]')
    grid on
end
saveas(gcf,"MC_quat_command_perf.pdf")

figure
```

```
for ii = 1:3
    subplot(3,1,ii)
    hold on
    for jj = 1:N_MC
        plot(command_err_w(ii,:,jj))

    end
    ylabel('$\omega^c_i - \hat{\omega}_i [rad/sec]$','Interpreter','latex')
    xlabel('Time [sec]')
    grid on
end
saveas(gcf,"MC_rate_command_perf.pdf")
```



simin.act_noise
Quaternion Measurement Noise2
[r_inertial]
[q_inertial2body]
PeturbQuaternion
simin.q_meas_noise
Quaternion Measurement Noise
[w_body_in_body]
simin.w_meas_noise
Quaternion Measurement Noise1
[q_inertial2body_meas]
[w_body_meas]
Flight Software
Control Torque
Angle Error
Inertial Reference Rate
Inertial Reference Quat
LVLH Reference Rate
LVLH Reference Quat
Error Quat
q_inertial2LVLH
q_inertial2body_est
w_body_est
est_cov
[angle_error]
[ref_rate]
[ref_quat]
[ref_rate_LVLH]
[ref_quat_LVLH]
[error_quat]
[q_inertial2LVLH]
[q_inertial2body_est]
[w_body_est]
[torque_command]
Control Torque
Actuator
Torque Command
Body Rate Wrt Inertial
Torque Output
CMG Angles
CMG Rate
CMG h
[torque_output]
Control Torque1
[CMG_angles]
[CMG_rate]
[CMG_h]
Plant
Input Torque
Inertial Position
Inertial Velocity
q_inertial2body
w_body_in_body
v_inertial1
[r_inertial]
[q_inertial2body]
[w_body_in_body]
Outputs To Workspace
r_inertial
q_inertial2body
w_body
torque command
Angle Error
Ref Rate
Ref Quat
Ref Quat LVLH
Ref Rate LVLH
CMG_angles
Torque Output
CMG Rate
Error Quat
q_inertial2LVLH
CMG_h
q_inertial2body_est
w_body_est
q_inertial2body_meas
w_body_meas
est_cov
[torque_command]
[angle_error]
[ref_rate]
[ref_quat]
[ref_quat_LVLH]
[ref_rate_LVLH]
[CMG_angles]
[torque_output]
[CMG_rate]
[error_quat]
[q_inertial2LVLH]
[CMG_h]
[q_inertial2body_est]
[w_body_est]
[q_inertial2body_meas]
[w_body_meas]

Actuator

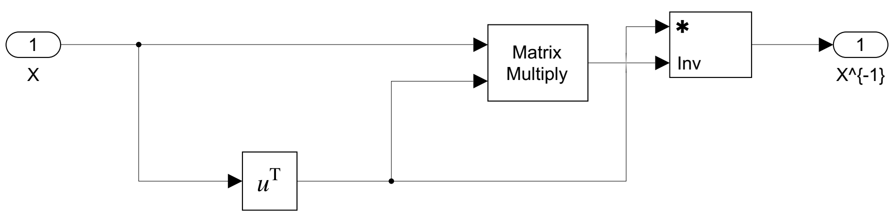


[0 0 0 0 pi/4 -pi/4 3*pi/4 -3*pi/4]'
Initial Gimbal Angles
1/s
Integrate Gimbal Angles
h0
h0
theta
CMG_AngMom
h
theta
CMG_X
X
X
X^{-1}
Psuedo Inverse
2
Body Rate Wrt Inertial
1
Torque Command
u
v
CrossProduct
y
Matrix Multiply
Matrix Multiply
1
Torque Output
4
CMG h
3
CMG Rate
2
CMG Angles

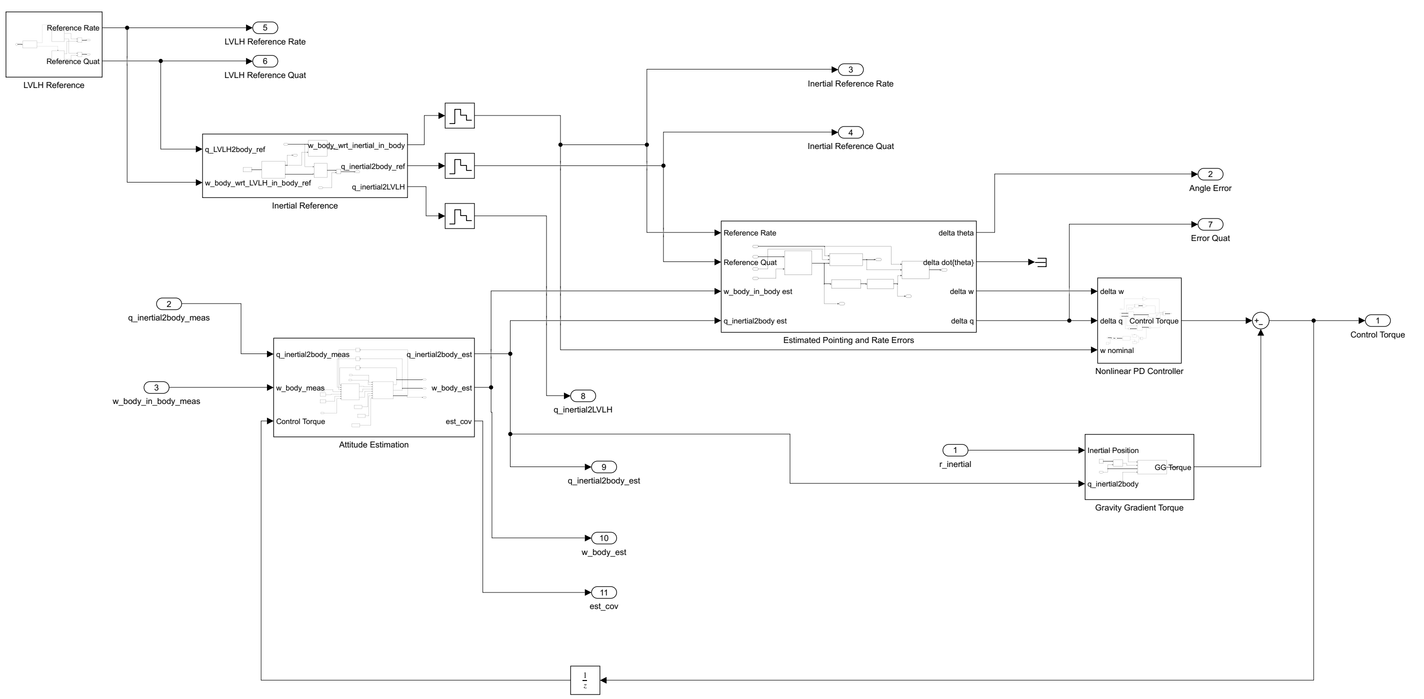
```
function h = CMG_AngMom(h0, theta)
% Produces the angular momentum vector for the CMGs

% Extract alpha and betas
alphas = theta(1:4);
betas = theta(5:8);

h = zeros(3,1);
for ii = 1:4
    h = h + h0*[sin(alphas(ii)); cos(alphas(ii))*cos(betas(ii)); cos(alphas(ii))*sin(betas(ii))];
end
```

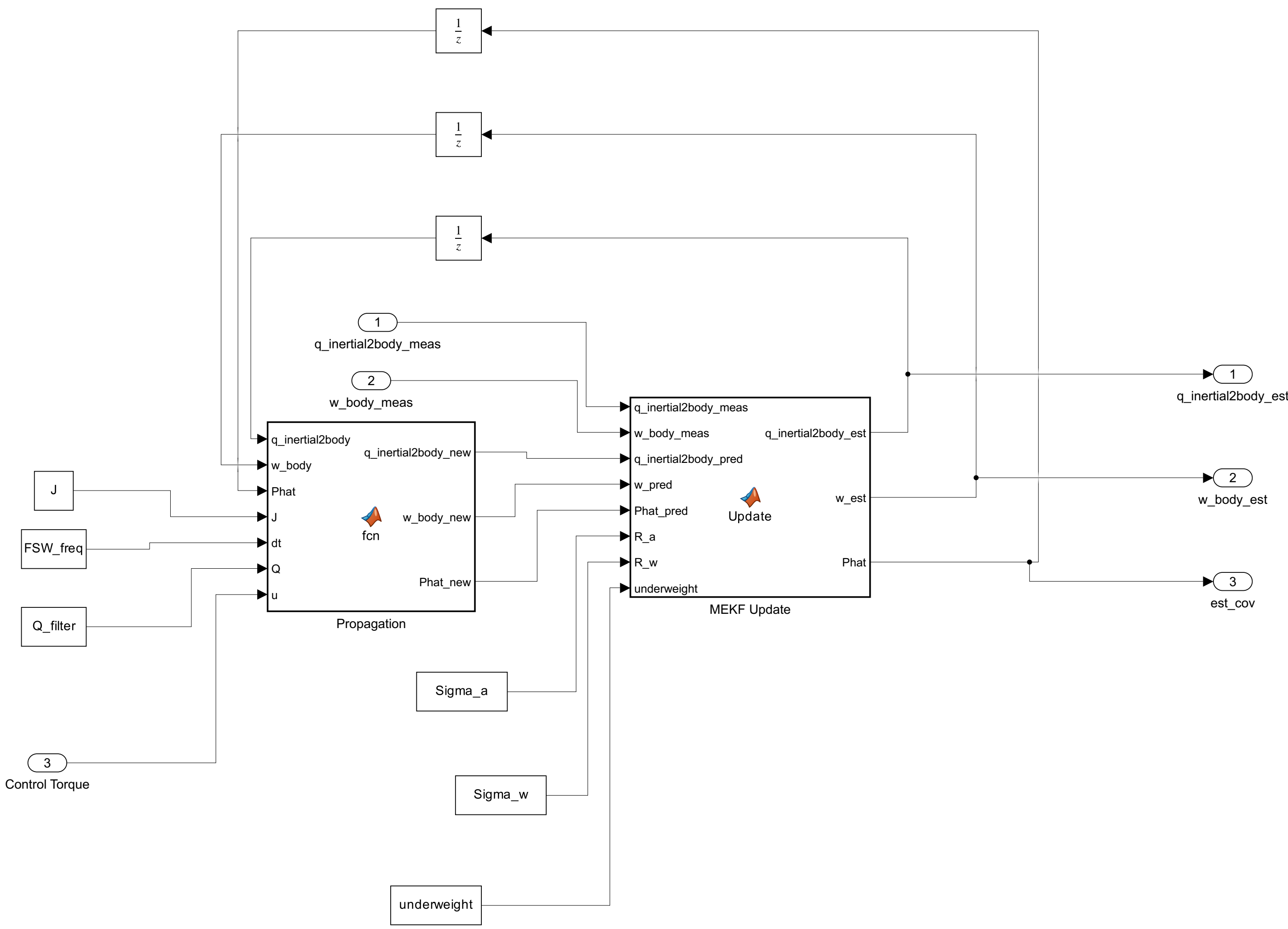
```
function X = CMG_X(theta)
% Produces the matrix of CMG spin axes

% Extract alpha and betas
alphas = theta(1:4);
betas = theta(5:8);

X = zeros(3,8);
for ii = 1:4
    X(:,2*ii-1:2*ii) = [cos(alphas(ii)), 0;
        -sin(alphas(ii))*cos(betas(ii)), -cos(alphas(ii))*sin(betas(ii));
        -sin(alphas(ii))*sin(betas(ii)), cos(alphas(ii))*cos(betas(ii))];
end
```

```
function y = CrossProduct(u,v)
y = CrossProductMat(u)*v;
```


1
X
Matrix
Multiply
*
Inv
1
X^{-1}
u^{T}

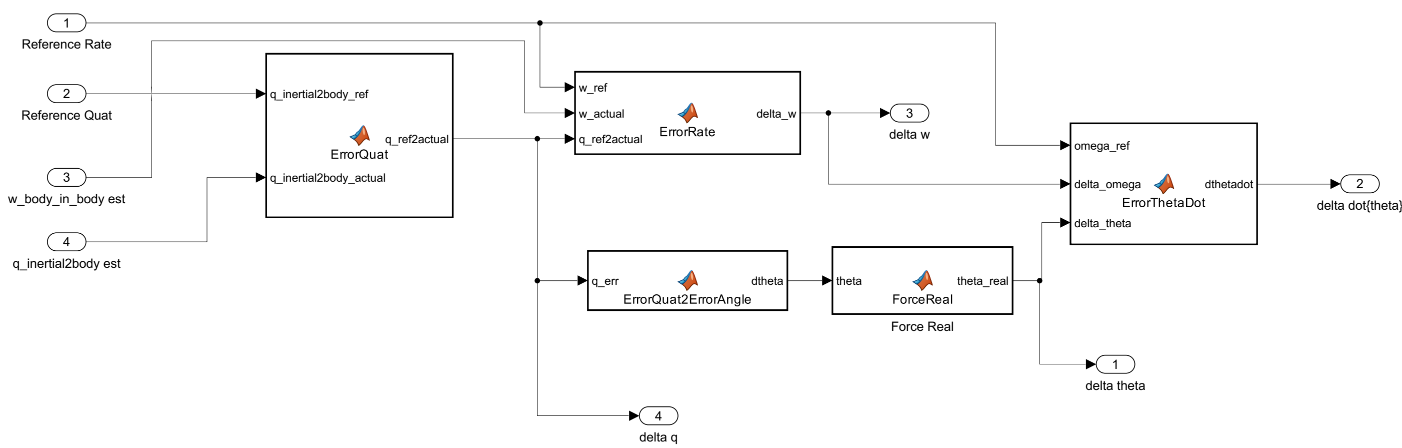
Flight Software




1/z
1/z
1/z
1
q_inertial2body_meas
2
w_body_meas
q_inertial2body
w_body
Phat
J
dt
Q
u
q_inertial2body_new
w_body_new
Phat_new
fcn
Propagation
q_inertial2body_meas
w_body_meas
q_inertial2body_pred
w_pred
Phat_pred
R_a
R_w
underweight
q_inertial2body_est
w_est
Phat
Update
MEKF Update
J
FSW_freq
Q_filter
3
Control Torque
Sigma_a
Sigma_w
underweight
1
q_inertial2body_est
2
w_body_est
3
est_cov

```
function [q_inertial2body_est, w_est, Phat] = Update(q_inertial2body_meas, w_body_meas, q_inertial2body_pred, w_pred, Phat_pred, R

% Find the error quaternion
dq = QuatProduct(q_inertial2body_meas,QuatInv(q_inertial2body_pred));
a_meas = 2*dq(1:3);

% Jacobian for direct measurement of a and w
H = [eye(3), zeros(3);
    zeros(3), eye(3)];

% Kalman gain
R = blkdiag(R_a,R_w);
K = Phat_pred*H'/((1+underweight)*H*Phat_pred*H' + R);

% update
xbar = [zeros(3,1); w_pred];
xhat = xbar + K*([a_meas; w_body_meas] - H*xbar);
w_est = xhat(4:6);

% Apply update to quaternion
dqhat = [xhat(1:3); 1];
dqhat = dqhat/norm(dqhat);
q_inertial2body_est = QuatProduct(dqhat,q_inertial2body_pred);

% Update covariance
Phat = (eye(6) - K*H)*Phat_pred*(eye(6) - K*H)' + K*R*K';

end
```

```
function [q_inertial2body_new, w_body_new, Phat_new] = fcn(q_inertial2body, w_body, Phat, J, dt, Q, u)

% Propagate Attitude
[q_inertial2body_new, w_body_new] = AttitudePropagate(q_inertial2body, w_body, J, dt, "RK4",u);

% Propagate Covariance
Aeval = PaPaFunc(dt,...
    w_body(1),...
    w_body(2),...
    w_body(3)+1E-10); % Add very small peturbation to prevent divide by zero
Beval = PaPdomegaFunc(dt,...
    w_body(1),...
    w_body(2),...
    w_body(3)+1E-10); % Add very small peturbation to prevent divide by zero
F = [ Aeval, Beval;
    zeros(3), eye(3)];
Phat_new = F*Phat*F' + Q;

end
```


1
Reference Rate
2
Reference Quat
3
w_body_in_body est
4
q_inertial2body est
q_inertial2body_ref
q_inertial2body_actual
q_ref2actual
ErrorQuat
w_ref
w_actual
q_ref2actual
delta_w
ErrorRate
3
delta w
omega_ref
delta_omega
delta_theta
dthetadot
ErrorThetaDot
2
delta dot{theta}
q_err
dtheta
ErrorQuat2ErrorAngle
theta
theta_real
ForceReal
Force Real
1
delta theta
4
delta q

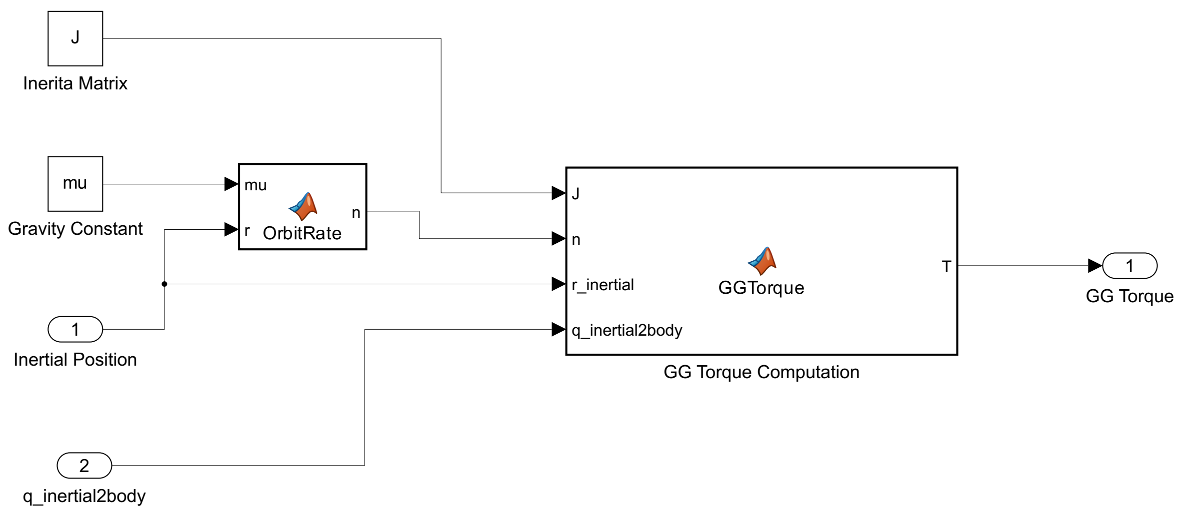
```
function theta_real = ForceReal(theta)
theta_real = real(theta);
```

```
function q_ref2actual = ErrorQuat(q_inertial2body_ref, q_inertial2body_actual)

q_ref2actual = QuatProduct(q_inertial2body_actual,QuatInv(q_inertial2body_ref));

% Force normalization
q_ref2actual = q_ref2actual/norm(q_ref2actual);
```

```
function delta_w = ErrorRate(w_ref, w_actual, q_ref2actual)
delta_w = w_actual - QuatTransform(q_ref2actual,w_ref);
```

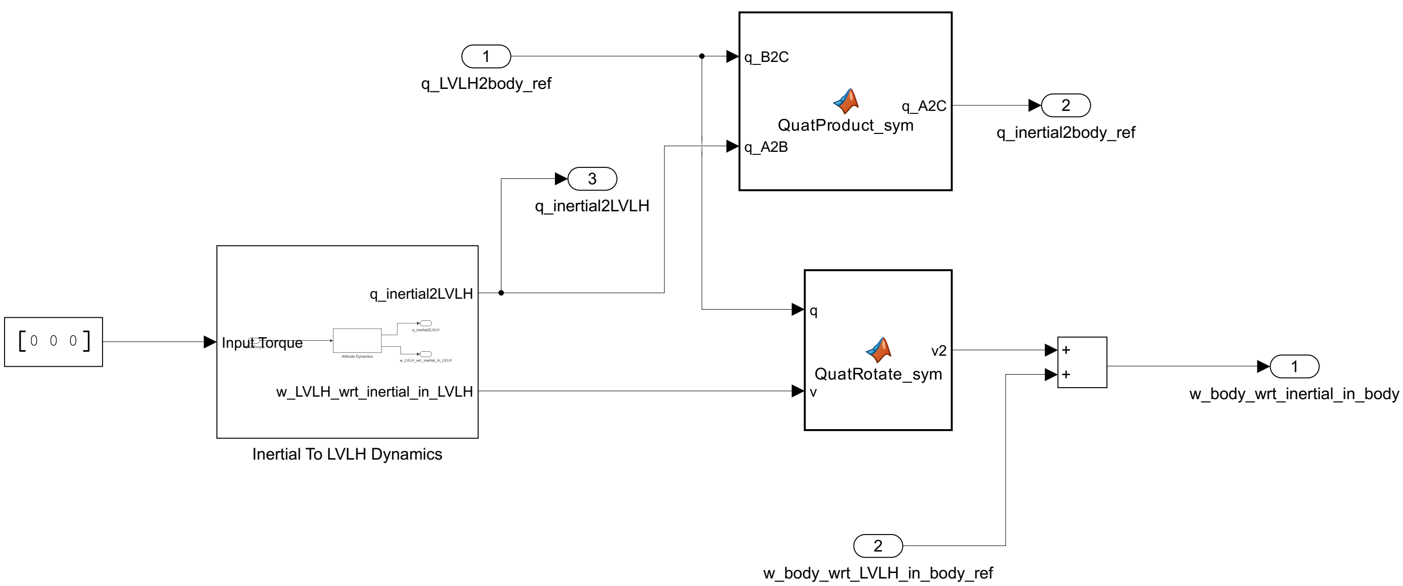
```
function dtheta = ErrorQuat2ErrorAngle(q_err)

% Quaternion components
q_v = real(q_err(1:3));
q_s = real(q_err(4));

dtheta = [atan2(q_v(1),q_s);
    atan2(q_v(2),q_s);
    atan2(q_v(3),q_s)];
```

```
function dthetadot = ErrorThetaDot(omega_ref, delta_omega, delta_theta)
dthetadot = delta_omega - cross(omega_ref,delta_theta);
```

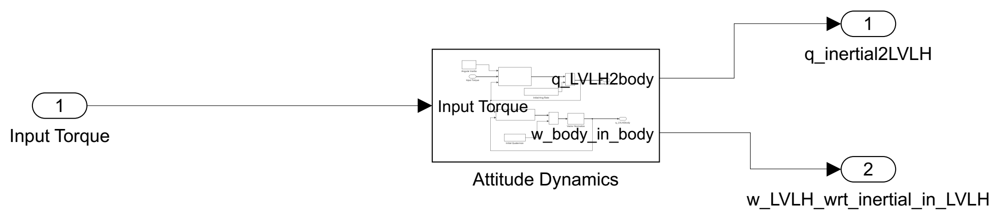

J
Inerita Matrix
mu
Gravity Constant
mu
r
OrbitRate
n
1
Inertial Position
2
q_inertial2body
J
n
r_inertial
q_inertial2body
GGTorque
T
GG Torque Computation
1
GG Torque

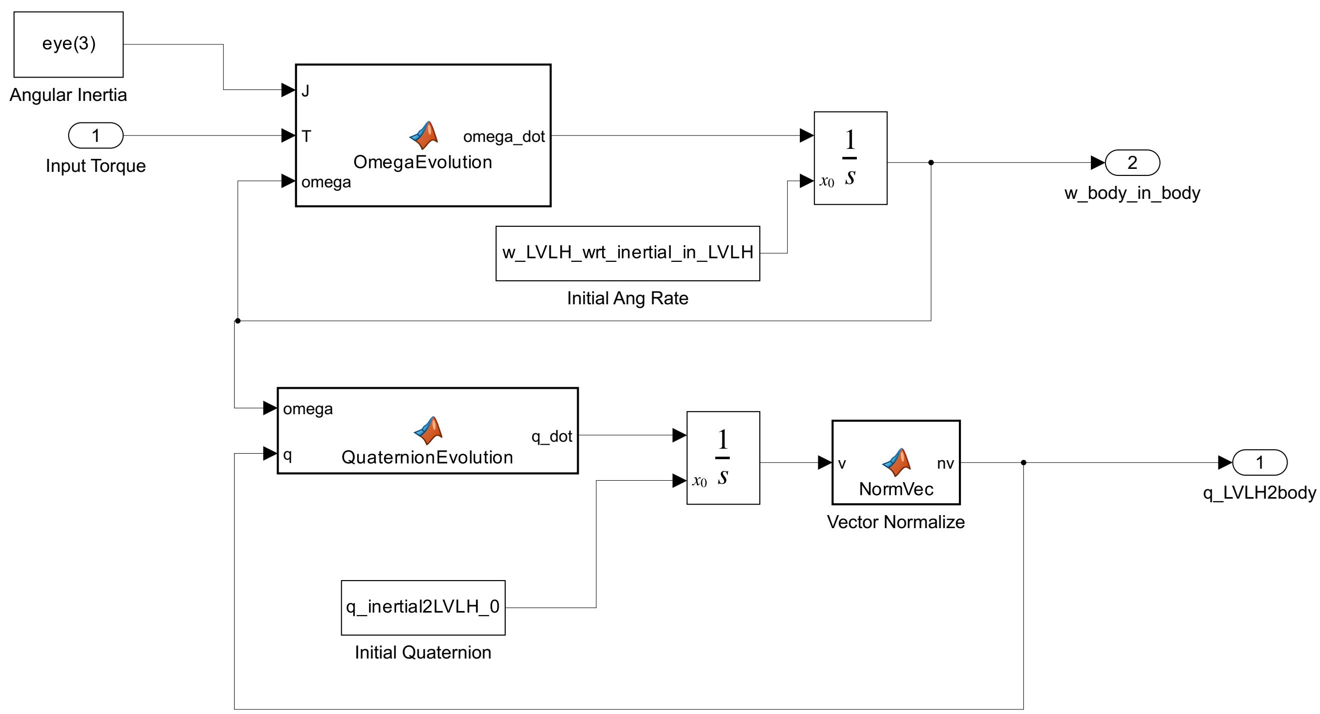
```
function T = GGTorque(J, n, r_inertial, q_inertial2body)

% Down in body frame
down_inertial = -r_inertial/norm(r_inertial);
down_body = QuatTransform(q_inertial2body,down_inertial);

% Gravity gradient torque
T = 3*n^2*cross(down_body,J*down_body);
```

```
function n = OrbitRate(mu,r)
n = sqrt(mu/norm(r)^3);
```



1
q_LVLH2body_ref
q_B2C
q_A2C
QuatProduct_sym
q_A2B
2
q_inertial2body_ref
3
q_inertial2LVLH
q_inertial2LVLH
Input Torque
w_LVLH_wrt_inertial_in_LVLH
Inertial To LVLH Dynamics
q
v2
QuatRotate_sym
v
+
+
1
w_body_wrt_inertial_in_body
2
w_body_wrt_LVLH_in_body_ref


1
Input Torque
q_LVLH2body
Input Torque
w_body_in_body
Attitude Dynamics
1
q_inertial2LVLH
2
w_LVLH_wrt_inertial_in_LVLH


eye(3)
Angular Inertia
1
Input Torque
J
T
omega
omega_dot
OmegaEvolution
1
s
x0
w_LVLH_wrt_inertial_in_LVLH
Initial Ang Rate
2
w_body_in_body
omega
q
q_dot
QuaternionEvolution
1
s
x0
v
nv
NormVec
Vector Normalize
1
q_LVLH2body
q_inertial2LVLH_0
Initial Quaternion

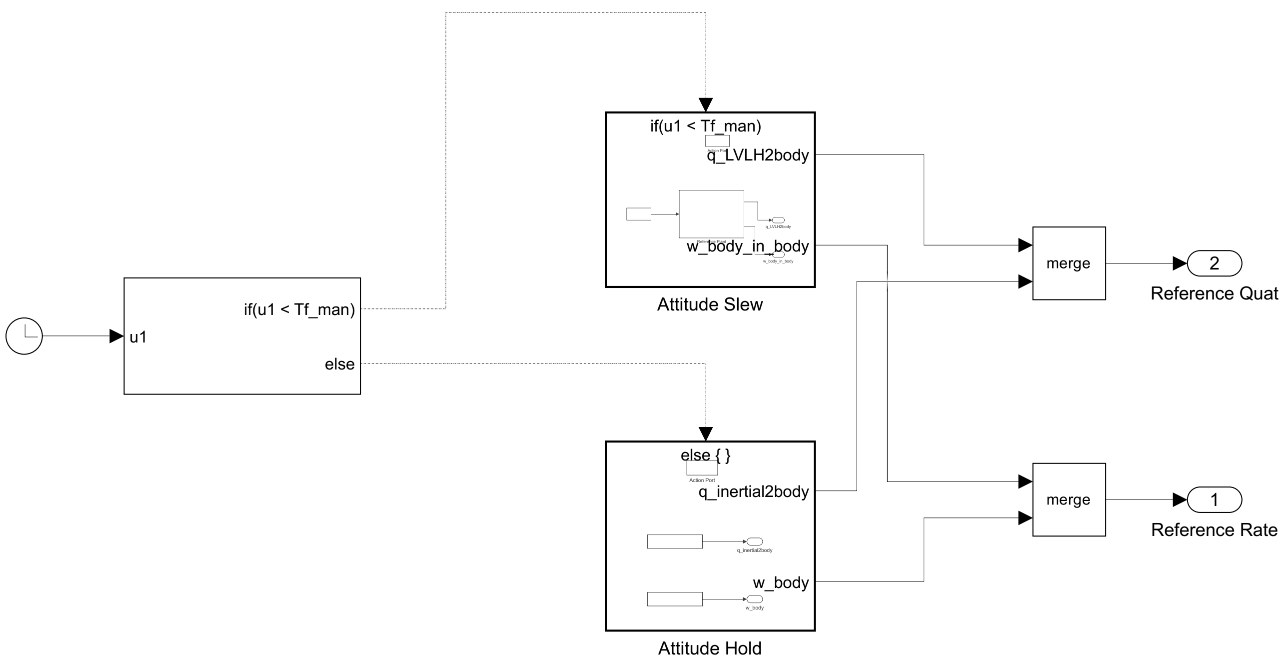
```
function q_dot = QuaternionEvolution(omega, q)
intermed = [omega; 0];
q_dot = 0.5*QuatProduct(intermed,q);
```

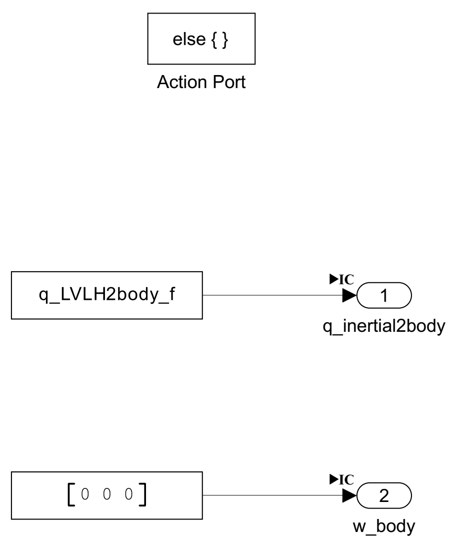
```
function omega_dot = OmegaEvolution(J, T, omega)
omega_dot = J\(T - cross(omega, J*omega));
```

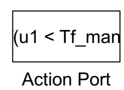
```
function nv = NormVec(v)
nv = v/norm(v);
```

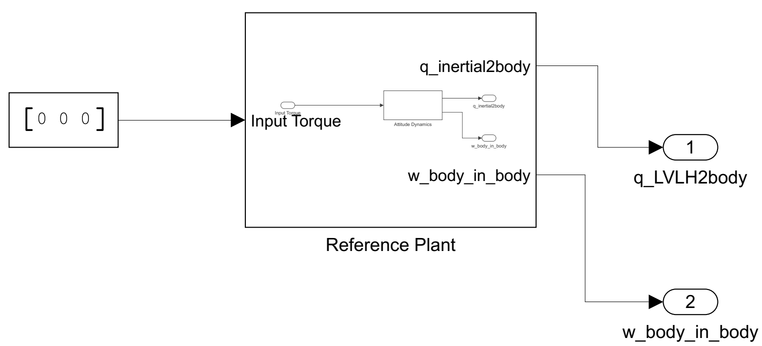
```
function q_A2C = QuatProduct_sym(q_B2C,q_A2B)
q_A2C = QuatProduct(q_B2C,q_A2B);
```

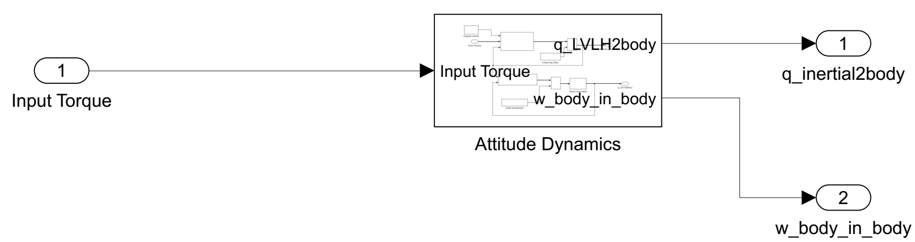
```
function v2 = QuatRotate_sym(q,v)
v2 = QuatTransform(q,v);
```

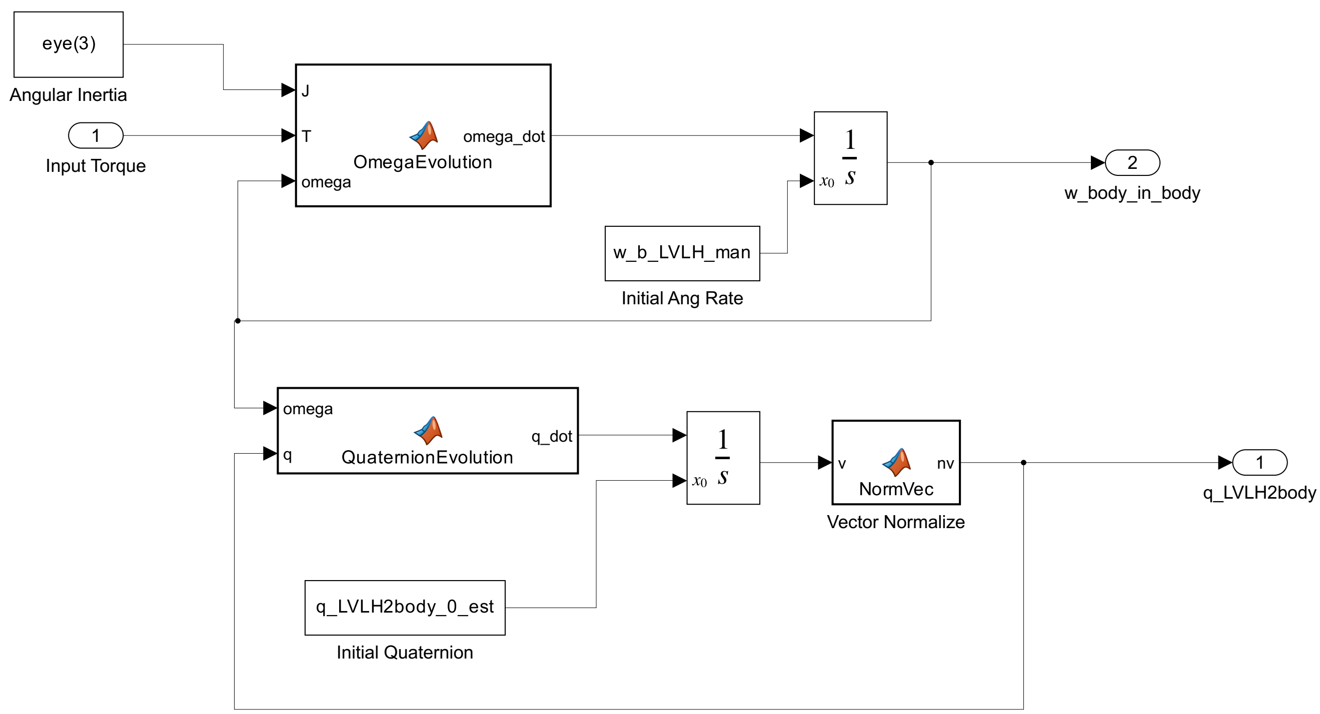

if(u1 < Tf_man)
u1
else
if(u1 < Tf_man)
q_LVLH2body
w_body_in_body
Attitude Slew
else { }
q_inertial2body
w_body
Attitude Hold
merge
merge
2
Reference Quat
1
Reference Rate


else { }
Action Port
q_LVLH2body_f
IC
1
q_inertial2body
[0 0 0]
IC
2
w_body


(u1 < Tf_man
Action Port


[0 0 0]
Input Torque
q_inertial2body
w_body_in_body
Reference Plant
1
q_LVLH2body
2
w_body_in_body

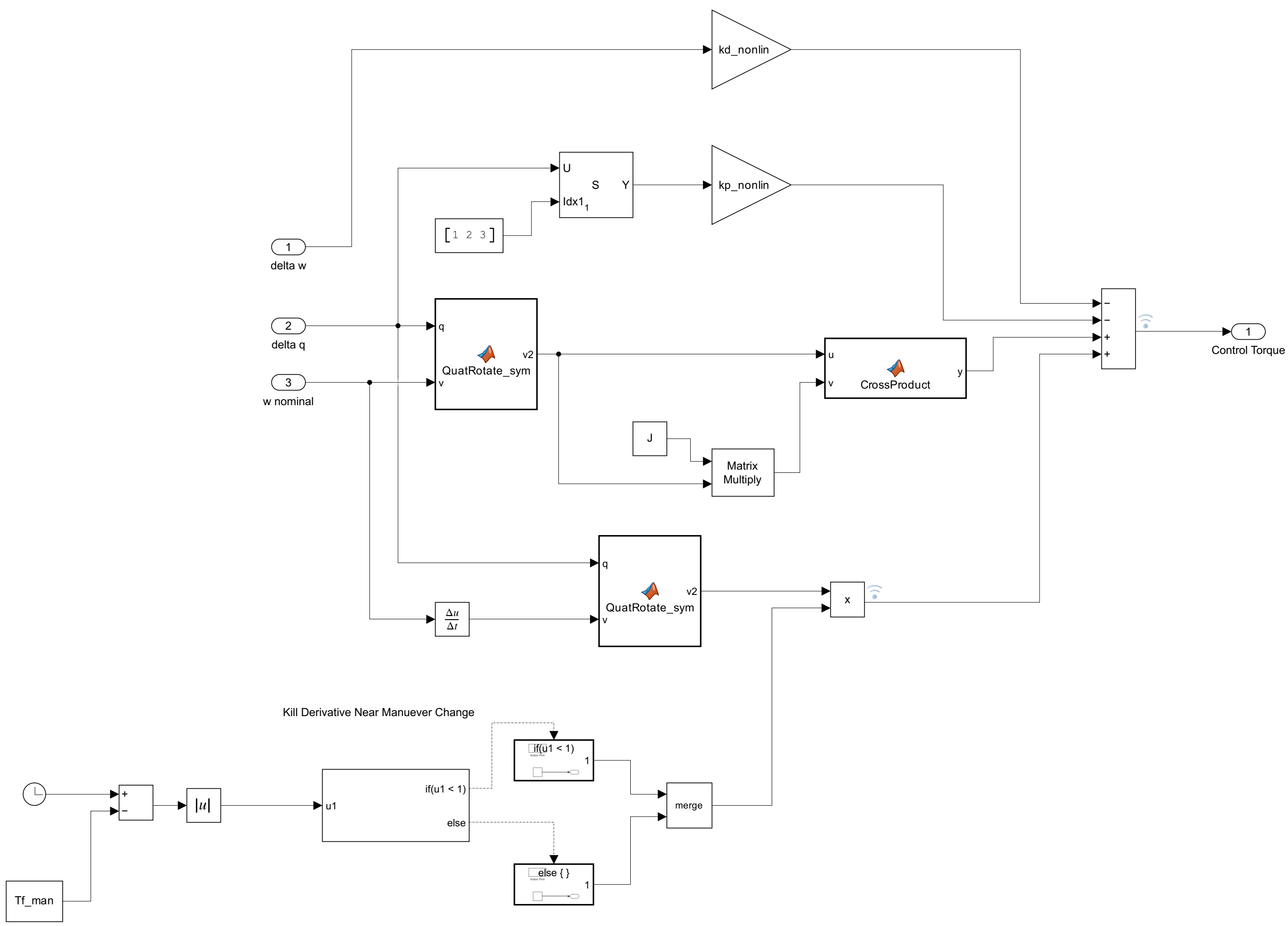

1
Input Torque
q_LVLH2body
Input Torque
w_body_in_body
Attitude Dynamics
1
q_inertial2body
2
w_body_in_body

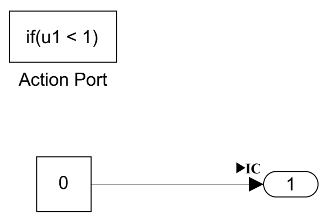

eye(3)
Angular Inertia
1
Input Torque
J
T
omega
OmegaEvolution
omega_dot
1/s
x0
w_b_LVLH_man
Initial Ang Rate
2
w_body_in_body
omega
q
QuaternionEvolution
q_dot
1/s
x0
v
NormVec
nv
Vector Normalize
q_LVLH2body_0_est
Initial Quaternion
1
q_LVLH2body

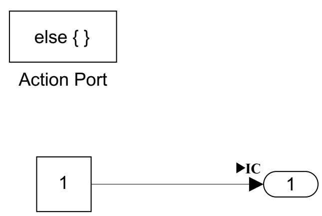
```
function q_dot = QuaternionEvolution(omega, q)
intermed = [omega; 0];
q_dot = 0.5*QuatProduct(intermed,q);
```

```
function omega_dot = OmegaEvolution(J, T, omega)
omega_dot = J\(T - cross(omega, J*omega));
```

```
function nv = NormVec(v)
nv = v/norm(v);
```


kd_nonlin
U
S
Y
Idx1
kp_nonlin
[1 2 3]
1
delta w
2
delta q
3
w nominal
q
v
v2
QuatRotate_sym
u
v
y
CrossProduct
J
Matrix Multiply
1
Control Torque
q
v2
QuatRotate_sym
v
x
Kill Derivative Near Manuever Change
if(u1 < 1)
1
merge
u1
if(u1 < 1)
else
else { }
1
|u|
Tf_man


if(u1 < 1)
Action Port
0
IC
1



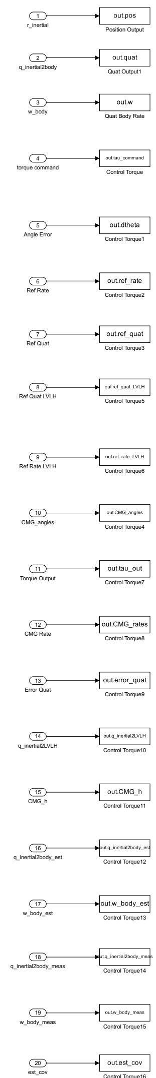
else { }
Action Port
1
IC
1

```
function y = CrossProduct(u,v)
y = CrossProductMat(u)*v;
```

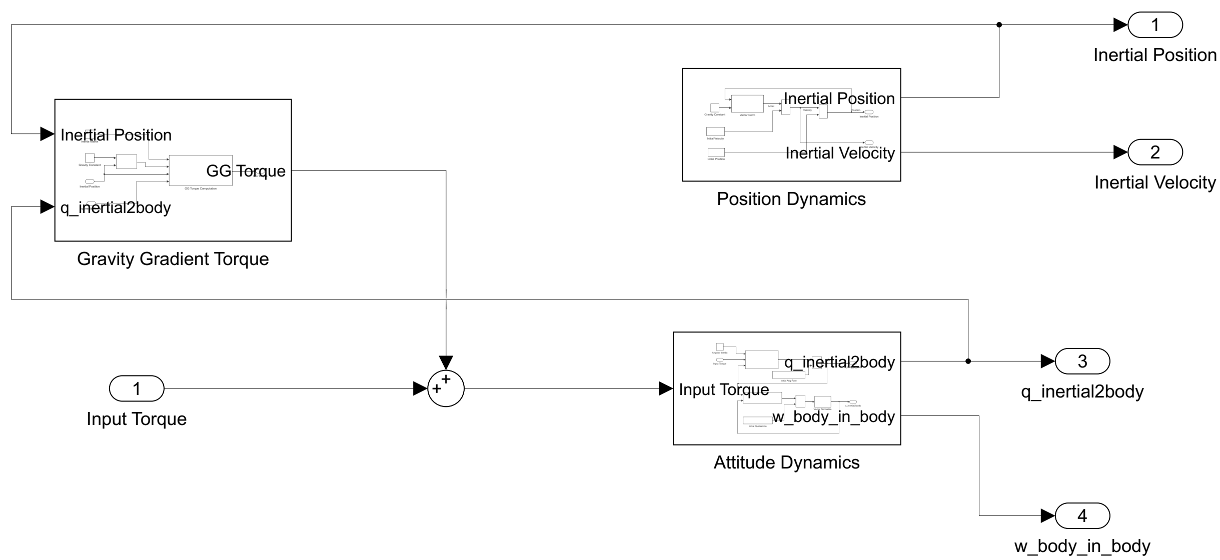
```
function v2 = QuatRotate_sym(q,v)
v2 = QuatTransform(q,v);
```

```
function v2 = QuatRotate_sym(q,v)
v2 = QuatTransform(q,v);
```

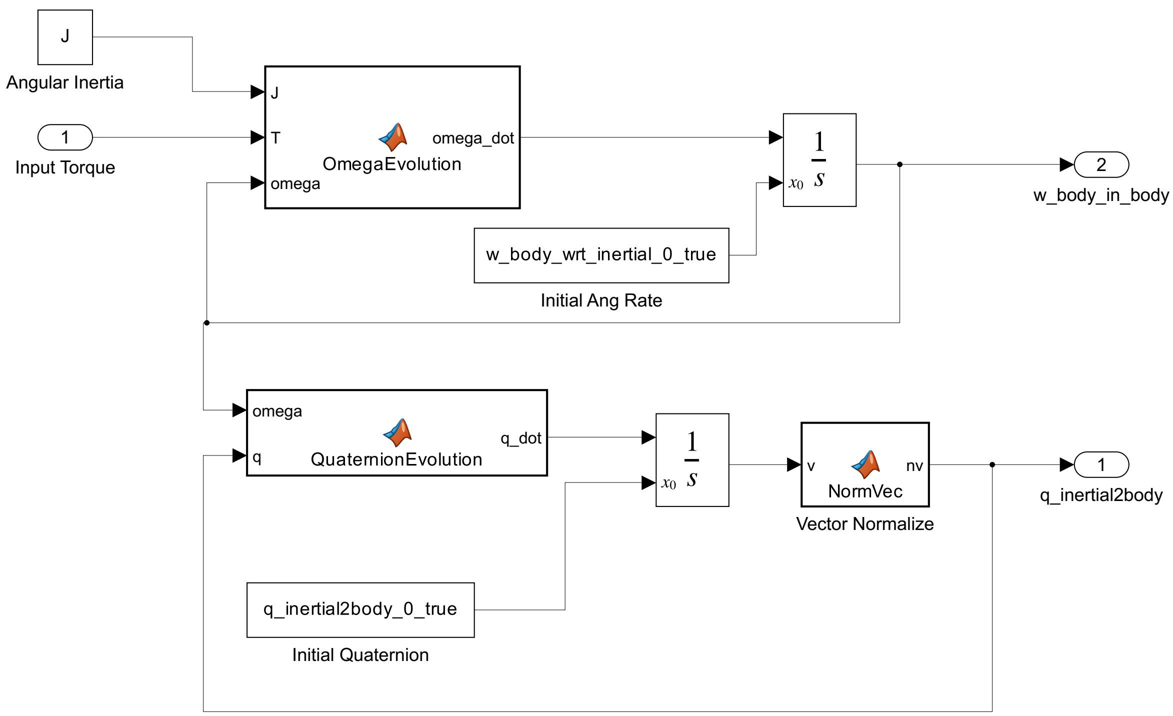
```
function q_pet = PeturbQuaternion(q, a)

% Convert a to dq
dq = [0.5*a; 1];
dq = dq/norm(dq);

% Peturb
q_pet = QuatProduct(dq,q);
```


1
r_inertial
out.pos
Position Output
2
q_inertial2body
out.quat
Quat Output1
3
w_body
out.w
Quat Body Rate
4
torque command
out.tau_command
Control Torque
5
Angle Error
out.dtheta
Control Torque1
6
Ref Rate
out.ref_rate
Control Torque2
7
Ref Quat
out.ref_quat
Control Torque3
8
Ref Quat LVLH
out.ref_quat_LVLH
Control Torque5
9
Ref Rate LVLH
out.ref_rate_LVLH
Control Torque6
10
CMG_angles
out.CMG_angles
Control Torque4
11
Torque Output
out.tau_out
Control Torque7
12
CMG Rate
out.CMG_rates
Control Torque8
13
Error Quat
out.error_quat
Control Torque9
14
q_inertial2LVLH
out.q_inertial2LVLH
Control Torque10
15
CMG_h
out.CMG_h
Control Torque11
16
q_inertial2body_est
out.q_inertial2body_est
Control Torque12
17
w_body_est
out.w_body_est
Control Torque13
18
q_inertial2body_meas
out.q_inertial2body_meas
Control Torque14
19
w_body_meas
out.w_body_meas
Control Torque15
20
est_cov
out.est_cov
Control Torque16

Plant



1
Inertial Position
Inertial Position
Inertial Velocity
Position Dynamics
2
Inertial Velocity
Inertial Position
GG Torque
q_inertial2body
Gravity Gradient Torque
1
Input Torque
+
+
Input Torque
q_inertial2body
w_body_in_body
Attitude Dynamics
3
q_inertial2body
4
w_body_in_body

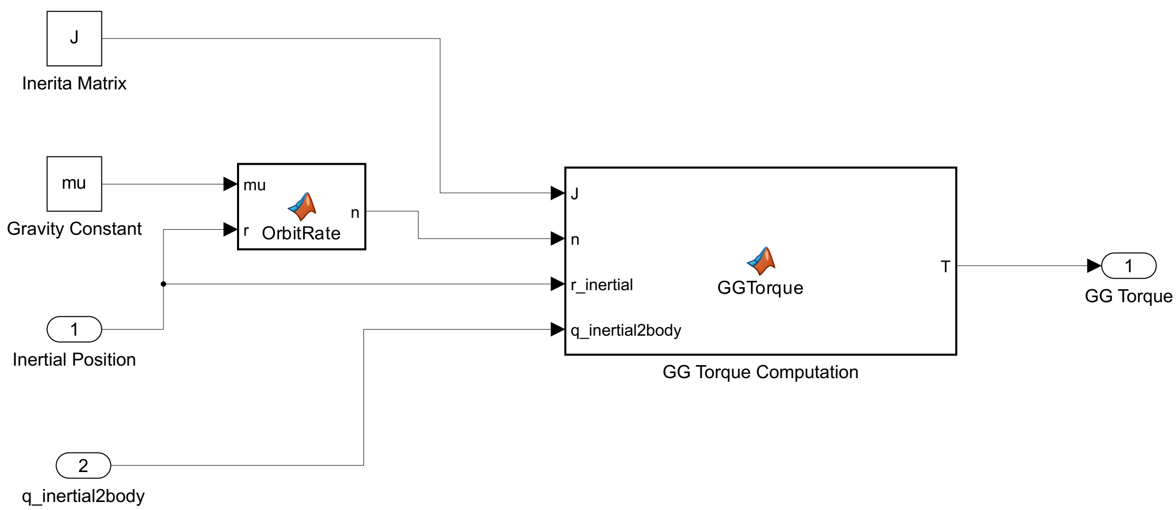


J
Angular Inertia
1
Input Torque
J
T
omega
OmegaEvolution
omega_dot
1
s
x0
2
w_body_in_body
w_body_wrt_inertial_0_true
Initial Ang Rate
omega
q
QuaternionEvolution
q_dot
1
s
x0
v
NormVec
nv
Vector Normalize
1
q_inertial2body
q_inertial2body_0_true
Initial Quaternion

```
function q_dot = QuaternionEvolution(omega, q)
intermed = [omega; 0];
q_dot = 0.5*QuatProduct(intermed,q);
```

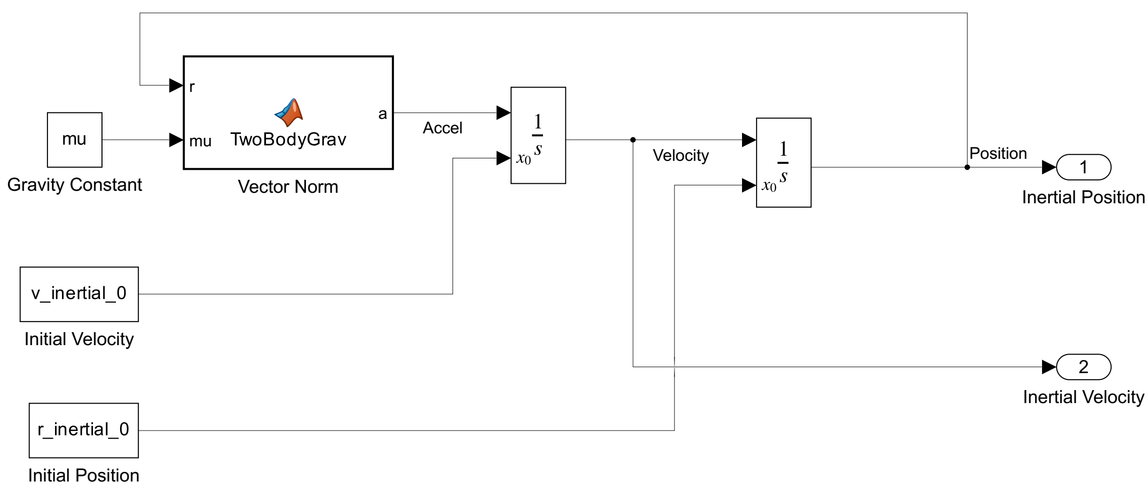
```
function omega_dot = OmegaEvolution(J, T, omega)
omega_dot = J\(T - cross(omega, J*omega));
```

```
function nv = NormVec(v)
nv = v/norm(v);
```


J
Inerita Matrix
mu
Gravity Constant
mu
r
OrbitRate
n
1
Inertial Position
2
q_inertial2body
J
n
r_inertial
q_inertial2body
GGTorque
T
GG Torque Computation
1
GG Torque

```
function T = GGTorque(J, n, r_inertial, q_inertial2body)

% Down in body frame
down_inertial = -r_inertial/norm(r_inertial);
down_body = QuatTransform(q_inertial2body,down_inertial);

% Gravity gradient torque
T = 3*n^2*cross(down_body,J*down_body);
```

```
function n = OrbitRate(mu,r)
n = sqrt(mu/norm(r)^3);
```


r
mu
TwoBodyGrav
a
mu
Gravity Constant
Vector Norm
Accel
1
s
x0
Velocity
Position
1
Inertial Position
v_inertial_0
Initial Velocity
2
Inertial Velocity
r_inertial_0
Initial Position

```
function a = TwoBodyGrav(r,mu)

nr = norm(r);
a = -mu*r/(nr^3);
```

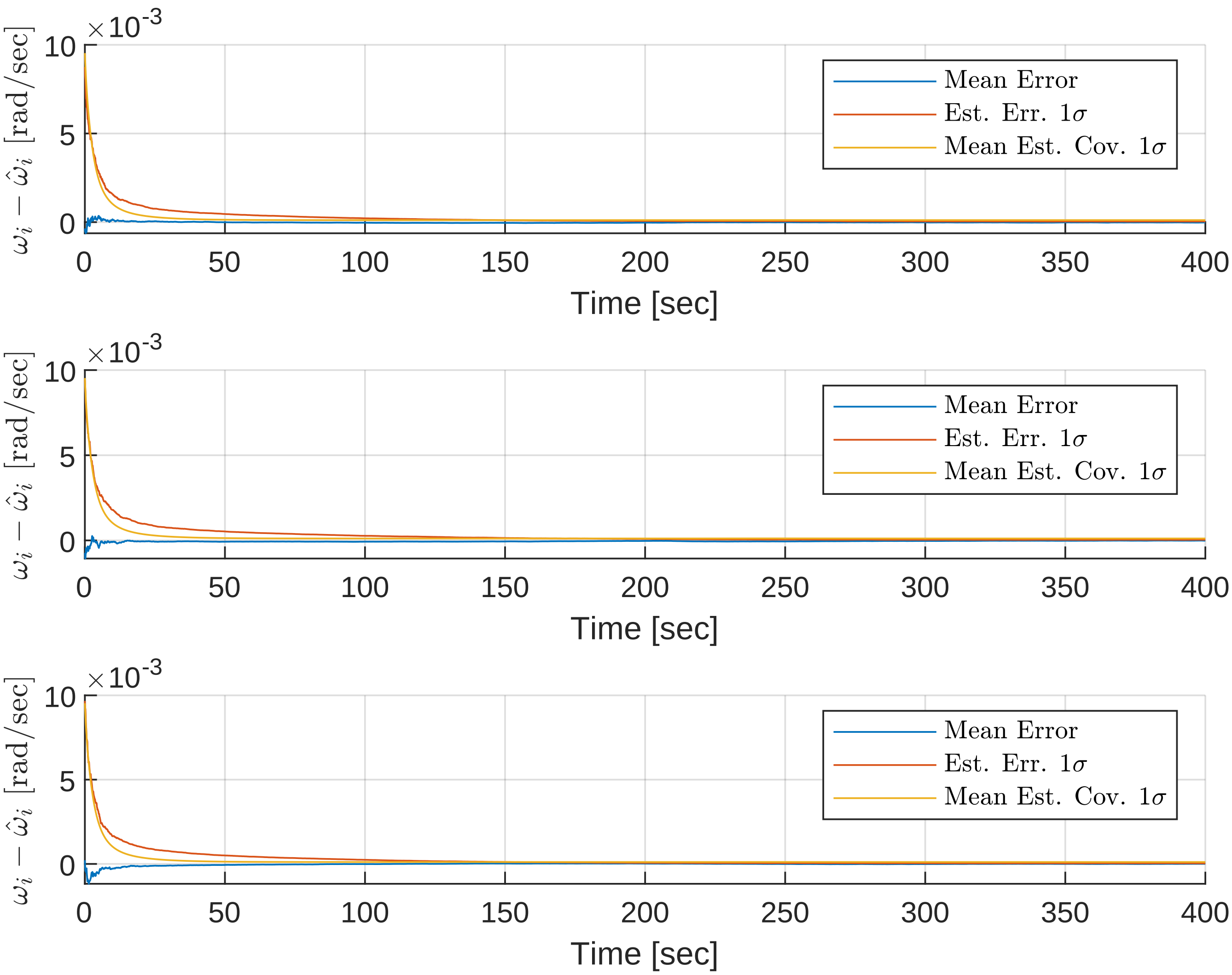


Figure 5: The angular rate estimation performance as shown by the average estimate error, sample error covariance, and average estimate covariance.