Reaction Wheel Control Assignment

Table of Contents

Name: Trisha Babu

This report presents the redesign of a satellite attitude control system using reaction wheels, explicitly accounting for hardware saturation limits. The redesigned system adheres to stability criteria (Gain Margin ≥ 6 dB, Phase Margin ≥ 60°) while modeling:

- Time delay of 0.01 s
- 2nd order reaction wheel dynamics
- Reaction wheel saturation limits for torque and angular momentum

Preliminaries

This clears all variables and sets the format to display more digits.

```
clearvars
close all
clc
format long
```

Control Preferences

ctrlpref

Addpath to Required Folders

```
addpath('../Attitude representations')
addpath('../Attitude Kinematics')
```

```
addpath('../Attitude Dynamics')

load qBus.mat;
```

Load mass properties

```
mass_properties;
```

Reaction Wheel properties

```
wn = 2*pi*10; % Reaction wheel natural frequency
zeta = sqrt(2)/2; % Reaction wheel damping ratio
hwmax = 0.015; % Nms
hwdotmax = 0.004; % Nm
safety = 0.5;
```

Initial reaction wheel angular momentum

```
hw0_B = [0;0;0];
```

Parameters for Design and Simulation

Control System Time Delay

```
dt_delay = 0.01; % seconds
```

Initial Satellite Attitude

Body axes aligned with inertial axes

```
q0_BI.s = 1;
q0_BI.v = [0;0;0];
```

Desired satellite attitude

Create a desired attitude quaternion by roattating through 90 degrees by x axis

```
e = [1;2;3];
e = e/norm(e);
qstar_BI = e2q(e,180*pi/180);
```

The intial attitude in the simulation can be specified by converting into inertial DCM

```
A0_BI = q2A(q0_BI);
A0_IB = A0_BI';
```

Initial satellite angular velocity

```
wbi0_B = [0;0;0];%rad/s
```

Inner loop Controller design

Design proportional control gains such that all three axes have identical responses. Use the following parameters and satisfy the following requirements: Following Control Design Requirements Gain Margin >= 10 dB Phase Margin >= 60 degrees Rise Time <= 0.025 seconds Overshoot <= 1%

Transfer Function Variable

```
s = tf('s');
```

Plant Model for Each Axis

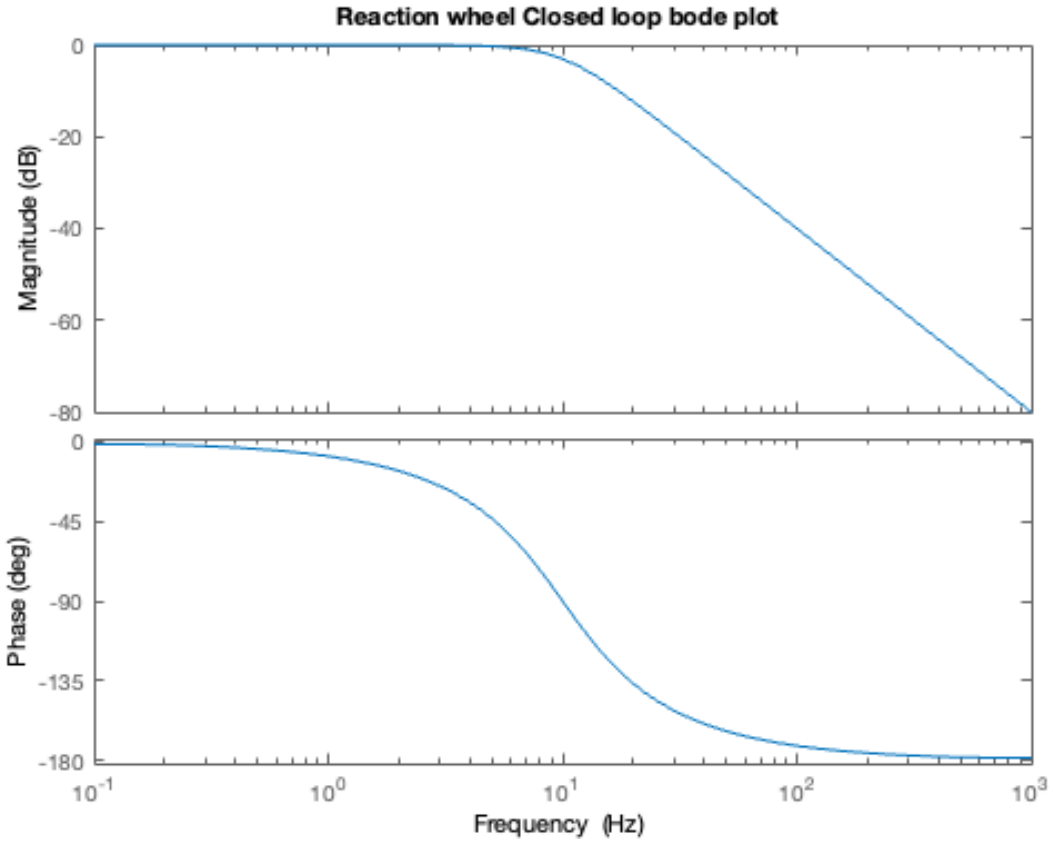
```
G1 = 1/(J_C_P(1,1) * s);
display(G1);
G2 = 1/(J_C_P(2,2) * s);
display(G2);
G3 = 1/(J_C_P(3,3) * s);
display(G3);
```

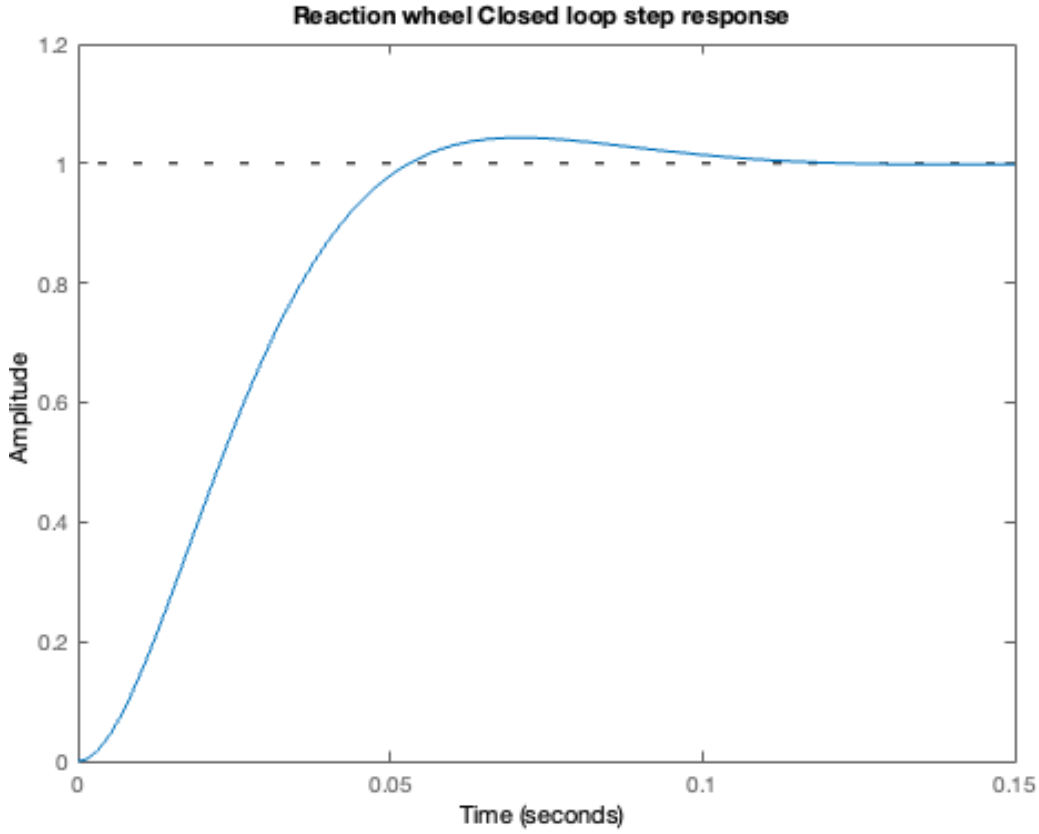
```
G1 =

      1
  --------------
  0.007215 s

Continuous-time transfer function.


G2 =

      1
  -------------
  0.03841 s

Continuous-time transfer function.


G3 =

      1
  -------------
  0.03993 s

Continuous-time transfer function.
```

Define reaction wheel transfer function

```
Gw = wn^2/(s^2 + 2*zeta*wn*s + wn^2);
```

```
figure
bode(Gw);
title('Reaction wheel Closed loop bode plot');

figure
step(Gw);
title('Reaction wheel Closed loop step response');

% Configure control system designer
% [num, den] = pade(dt_delay,8);
% C_pade8 = tf(num,den);
%
% config = sisoinit(1);
% config.G.value = G1;
% config.C.value = C_pade8*Gw;
% % config.OL1.View = {'bode'};
% % config.CL2.View = {'bode'};
% controlSystemDesigner(config);
```





Select the gain crossover frequency to use for all three plants

```
w_crossover = 2*pi*2.5; %rad/s
display(w_crossover);
```

```
w_crossover =

  15.707963267948966
```

Design a proportional gain for each of the three principle axes that meet the design requirements above. Choose the gains such that each of the three axes have identical responses.

```
[num,den] = pade(dt_delay,8);
C_pade8 = tf(num,den);

Kd1 = 1 / bode(G1*Gw, w_crossover);
```

```
display(Kd1);
Kd2 = 1 / bode(G2*Gw, w_crossover);
display(Kd2);
Kd3 = 1 / bode(G3*Gw, w_crossover);
display(Kd3);

Kd = diag([Kd1; Kd2; Kd3]);
```

```
Kd1 =

   0.113558849793423


Kd2 =

   0.604499920413859


Kd3 =

   0.628418389070616
```

Controller with delay by cascading the proportional gain

```
C1 = Kd1 * C_pade8;
display(C1);
C2 = Kd2 * C_pade8;
display(C2);
C3 = Kd3 * C_pade8;
display(C3);
```

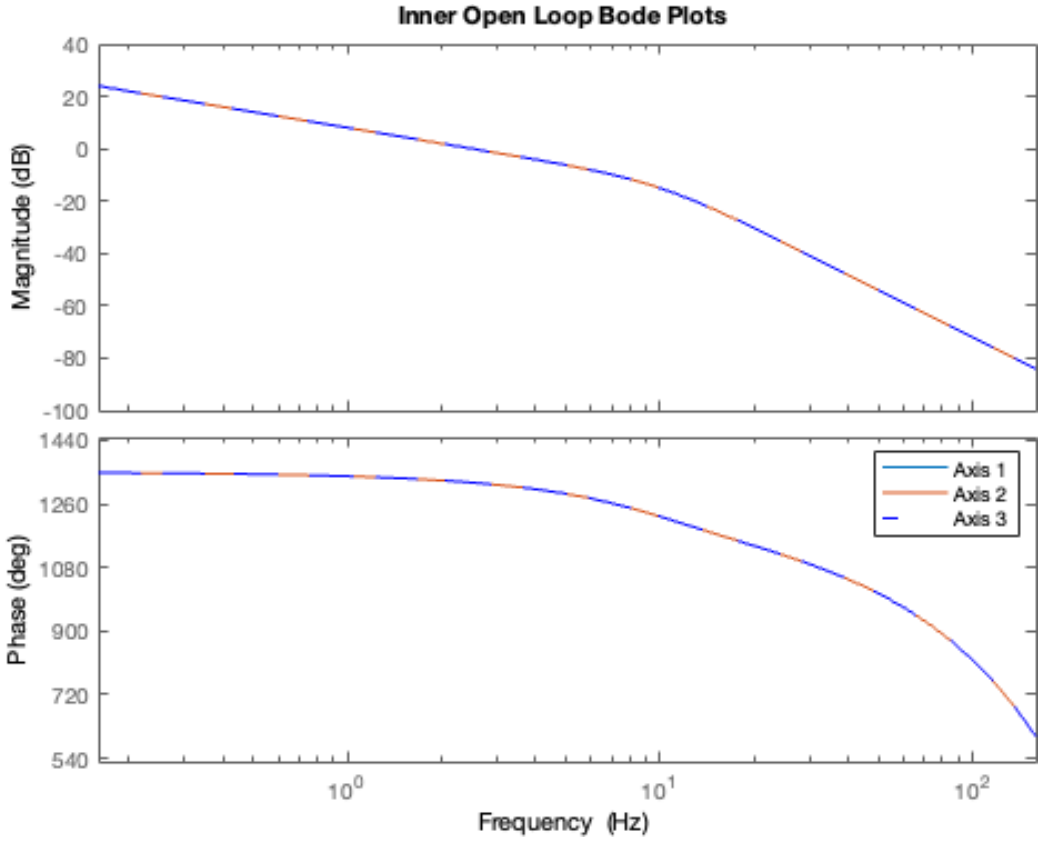
```
C1 =


   0.1136 s^8 - 817.6 s^7 + 2.862e06 s^6 - 6.296e09 s^5 + 9.444e12 s^4
                        - 9.821e15 s^3 + 6.875e18 s^2 - 2.946e21 s + 5.893e23
  ----------------------------------------------------------------------------------------------------
  s^8 + 7200 s^7 + 2.52e07 s^6 + 5.544e10 s^5 + 8.316e13 s^4 + 8.649e16 s^3
                                      + 6.054e19 s^2 + 2.595e22 s + 5.189e24

Continuous-time transfer function.
```

```
C2 =

   0.6045 s^8 - 4352 s^7 + 1.523e07 s^6 - 3.351e10 s^5 + 5.027e13 s^4
                        - 5.228e16 s^3 + 3.66e19 s^2 - 1.568e22 s + 3.137e24
  ----------------------------------------------------------------------------------------------------
  s^8 + 7200 s^7 + 2.52e07 s^6 + 5.544e10 s^5 + 8.316e13 s^4 + 8.649e16 s^3
                                      + 6.054e19 s^2 + 2.595e22 s + 5.189e24

Continuous-time transfer function.

C3 =

   0.6284 s^8 - 4525 s^7 + 1.584e07 s^6 - 3.484e10 s^5 + 5.226e13 s^4
                        - 5.435e16 s^3 + 3.804e19 s^2 - 1.63e22 s + 3.261e24
  ----------------------------------------------------------------------------------------------------
  s^8 + 7200 s^7 + 2.52e07 s^6 + 5.544e10 s^5 + 8.316e13 s^4 + 8.649e16 s^3
                                      + 6.054e19 s^2 + 2.595e22 s + 5.189e24

Continuous-time transfer function.
```

Open Loop Bode Plot of controlled plant

```
figure
bode(C1*G1*Gw, C2*G2*Gw, C3*G3*Gw, {1, 1000}, '--');
title('Inner Open Loop Bode Plots');
legend('Axis 1', 'Axis 2', 'Axis 3');

[Gm1,Pm1] = margin(C1*G1*Gw);
[Gm2,Pm2] = margin(C2*G2*Gw);
[Gm3,Pm3] = margin(C3*G3*Gw);

display(mag2db(Gm1), 'Inner Gain margin 1 (dB)');
display( Pm1, 'Inner Phase margin 1 (degrees)');

display(mag2db(Gm2), 'Inner Gain margin 2 (dB)');
display( Pm2, 'Inner Phase margin 2 (degrees)');

display(mag2db(Gm3), 'Inner Gain margin 3 (dB)');
display( Pm3, 'Inner Phase margin 3 (degrees)');
```

```
Inner Gain margin 1 (dB) =

  10.126232675303228

Inner Phase margin 1 (degrees) =

  60.337410047789028

Inner Gain margin 2 (dB) =

  10.126232675303179

Inner Phase margin 2 (degrees) =

  60.337410047789028

Inner Gain margin 3 (dB) =

  10.126232675303209

Inner Phase margin 3 (degrees) =

  60.337410047789028
```

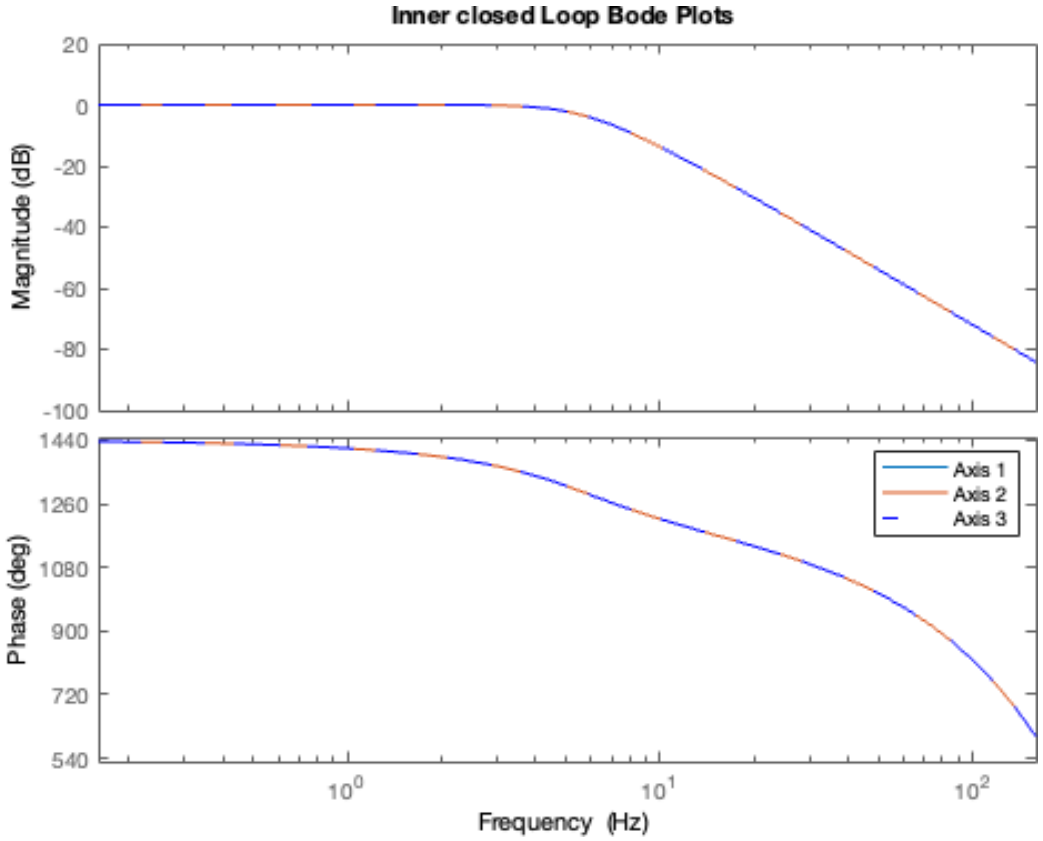


Closed Loop Transfer Functions

```
CLTF1 = feedback(C1 * G1 * Gw, 1);
CLTF2 = feedback(C2 * G2 * Gw, 1);
CLTF3 = feedback(C3 * G3 * Gw, 1);
```

Closed Loop Bode Plots

```
figure
bode(CLTF1, CLTF2, CLTF3, {1, 1000}, '--');
title('Inner closed Loop Bode Plots');
legend('Axis 1', 'Axis 2', 'Axis 3');
```



Closed Loop 3dB Bandwidth

```
BW1 = bandwidth(CLTF1)/(2*pi);
BW2 = bandwidth(CLTF2)/(2*pi);
BW3 = bandwidth(CLTF3)/(2*pi);

display(BW1, 'Inner Closed Loop 3 dB Bandwidth Axis 1 (Hz)');
display(BW2, 'Inner Closed Loop 3 dB Bandwidth Axis 2 (Hz)');
display(BW3, 'Inner Closed Loop 3 dB Bandwidth Axis 3 (Hz)');
```

```
Inner Closed Loop 3 dB Bandwidth Axis 1 (Hz) =

   5.483040380603219


Inner Closed Loop 3 dB  Bandwidth Axis 2 (Hz) =

   5.483040380603284


Inner Closed Loop 3 dB Bandwidth Axis 3 (Hz) =
```

```
   5.483040380603242
```

Step Response Information

```
stepinfo1 = stepinfo(CLTF1);
stepinfo2 = stepinfo(CLTF2);
stepinfo3 = stepinfo(CLTF3);
display(stepinfo1);
display(stepinfo2);
display(stepinfo3);
```

```
stepinfo1 =

  struct with fields:

         RiseTime: 0.064496515808807
    TransientTime: 0.200022669066274
     SettlingTime: 0.200022701606210
      SettlingMin: 0.914019720509337
      SettlingMax: 1.068681211130506
        Overshoot: 6.868121113050596
       Undershoot: 1.655587440289646e-04
             Peak: 1.068681211130506
         PeakTime: 0.148850543451916


stepinfo2 =

  struct with fields:

         RiseTime: 0.064496515808807
    TransientTime: 0.200022669066274
     SettlingTime: 0.200022701606210
      SettlingMin: 0.914019720509333
      SettlingMax: 1.068681211130505
        Overshoot: 6.868121113050529
       Undershoot: 1.655587440289565e-04
             Peak: 1.068681211130505
         PeakTime: 0.148850543451915


stepinfo3 =

  struct with fields:

         RiseTime: 0.064496515808807
    TransientTime: 0.200022669066274
     SettlingTime: 0.200022701606210
      SettlingMin: 0.914019720509332
      SettlingMax: 1.068681211130505
        Overshoot: 6.868121113050529
```

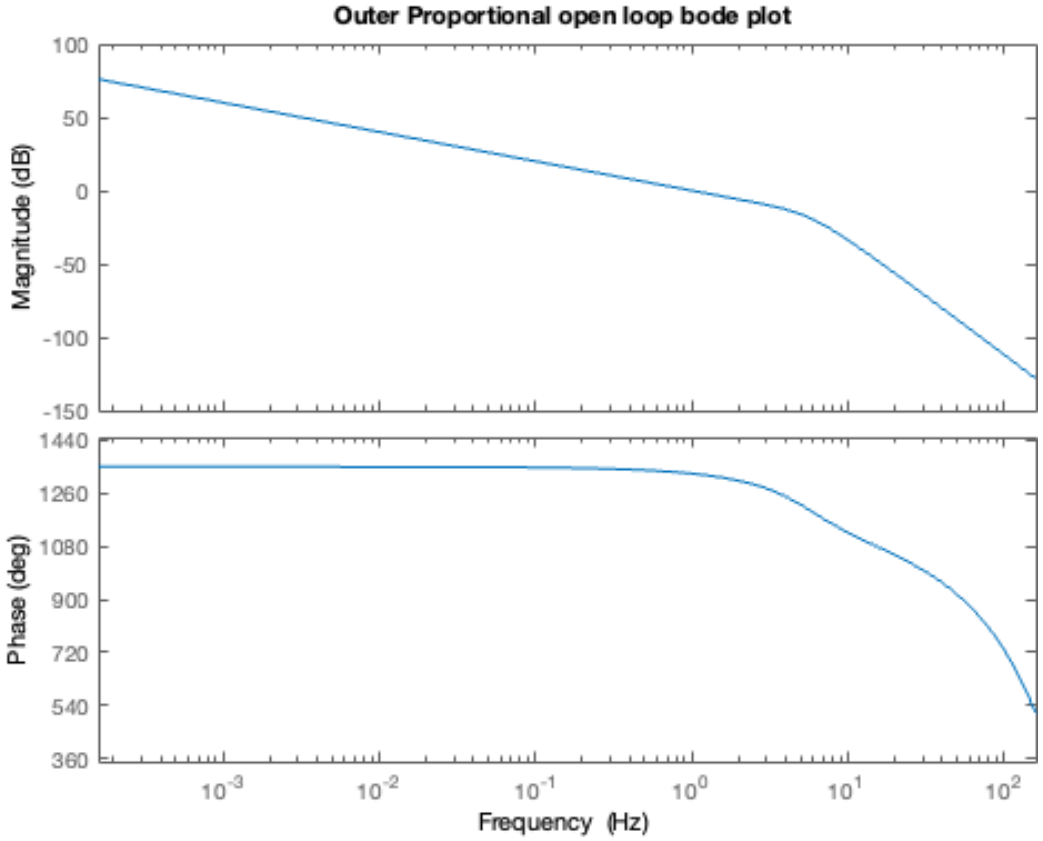
```
    Undershoot: 1.655587440289186e-04
          Peak: 1.068681211130505
      PeakTime: 0.148850543451915
```

Outer loop proportional design

Configure control system designer to start in correct loop configuration

```
% config = sisoinit(6);
% config.G1.value = G1;
% config.C1.value = 1;
% config.C2.value = Kd1*C_pade8*Gw;
% config.G2.value = 1/s;
% config.OL1.View = {'bode'};
% config.OL2.View = {};
% controlSystemDesigner(config);

Go = 1/s;

Kp = 1/bode(CLTF1*Go,2*pi*1);
OLTF = Kp*CLTF1*Go;

figure;
bode(OLTF,{0.001,1000});
title('Outer Proportional open loop bode plot');
[Gm, Pm]= margin(OLTF);

display(mag2db(Gm),'Outer proportional gain margin');
display(Pm,'Outer proportional gain margin(deg)');

% %% Step info of the proportional closed loop transfer function
% stepinfo_outer = stepinfo(CLTF);
% display(stepinfo_outer);
```

```
Outer proportional gain margin =

  11.641780619277423


Outer proportional gain margin(deg) =

  66.953447279826861
```

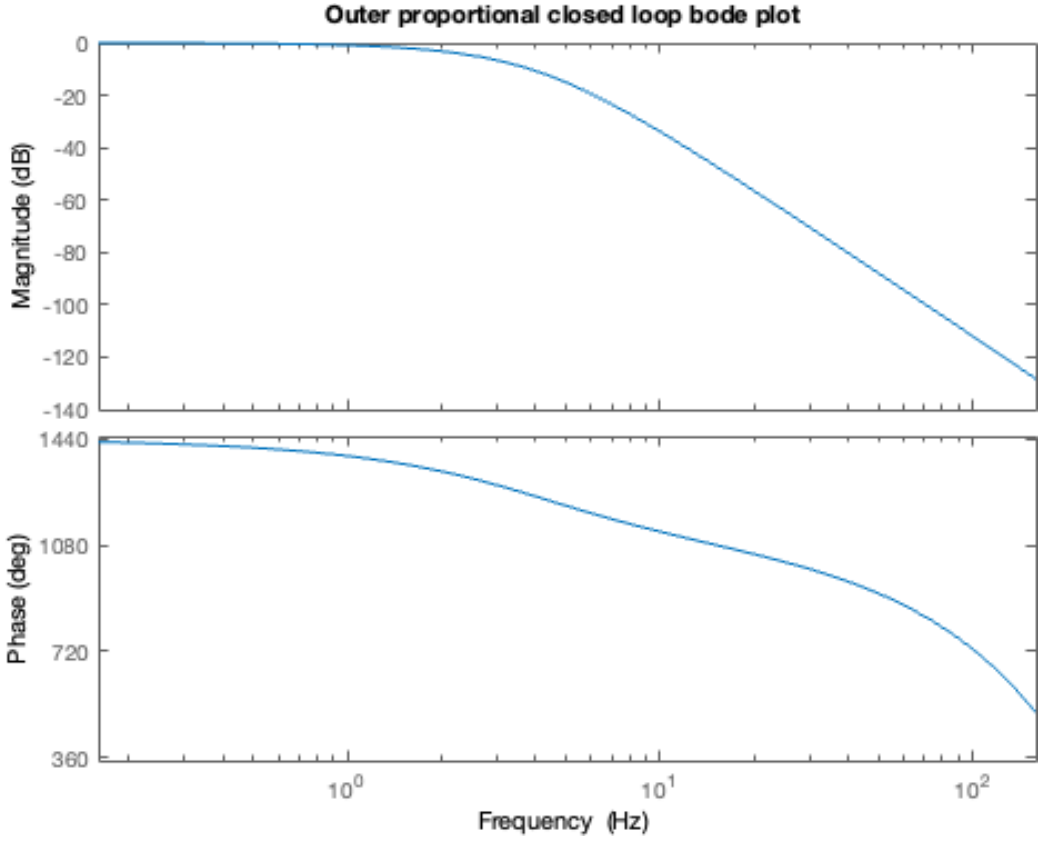


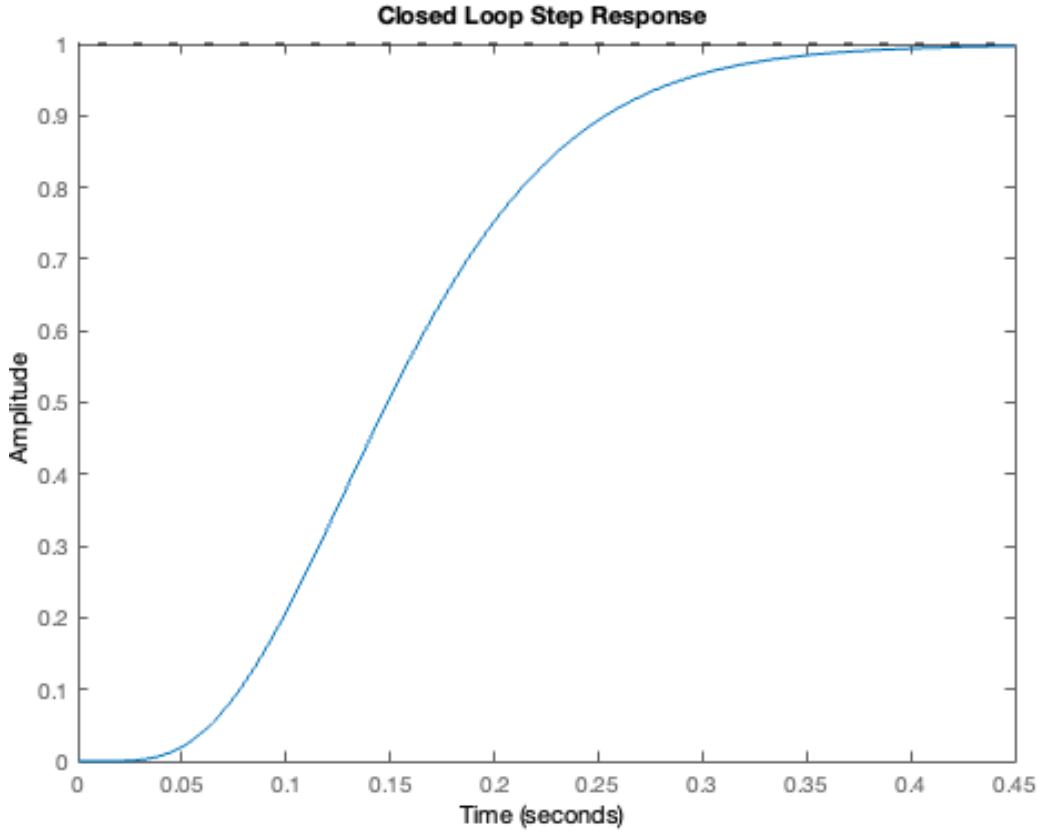
Closed Loop Bode plot of controlled plant

```
CLTF = feedback(OLTF,1);
figure;
bode(CLTF,{1,1000});
title('Outer proportional closed loop bode plot');
display(bandwidth(CLTF)/2*pi),'Outer Closed loop 3dB bandwidth(Hz)';

stepinfo_outer = stepinfo(CLTF);
display(stepinfo_outer);

figure
step(CLTF)
title('Closed Loop Step Response');
```

```
  19.176608675763841


stepinfo_outer =

  struct with fields:

        RiseTime: 0.175542688185157
```

```
TransientTime: 0.337668522074204
 SettlingTime: 0.337668522445256
  SettlingMin: 0.903759903910375
  SettlingMax: 0.998914646022949
    Overshoot: 0
   Undershoot: 7.237248397607562e-07
         Peak: 0.998914646022949
     PeakTime: 0.513529547870276
```



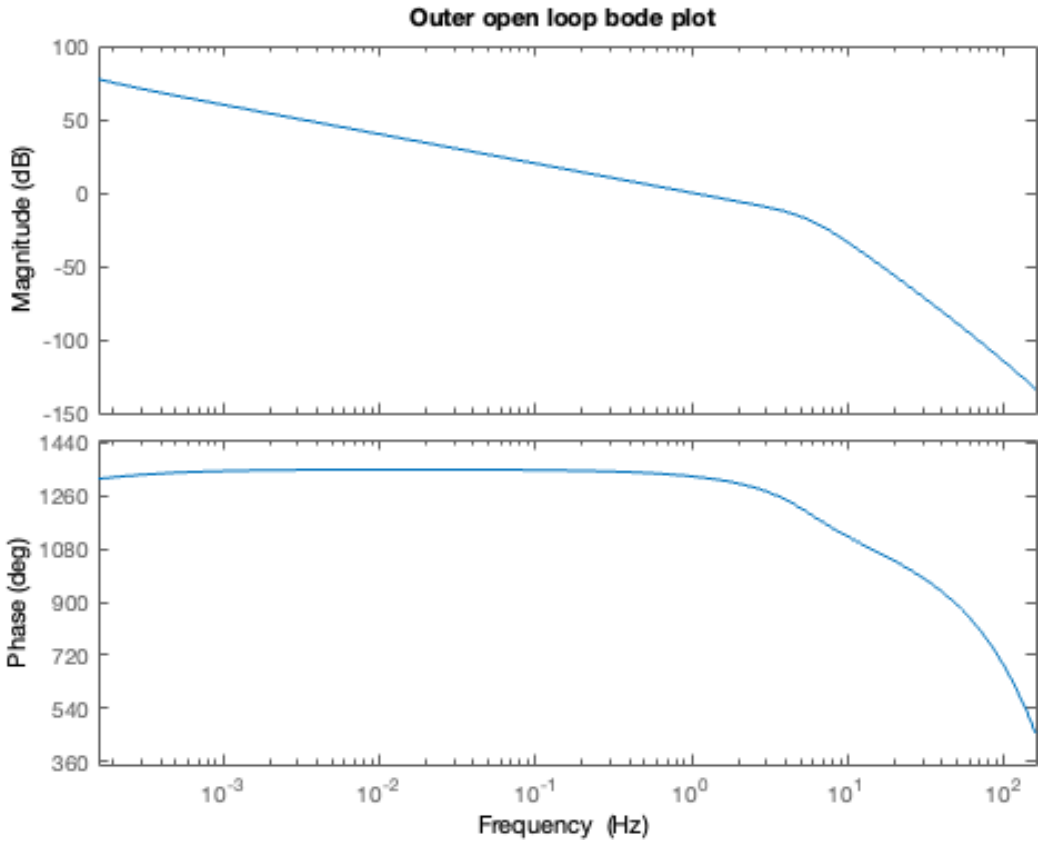


Outer loop design with integrator and lead

```
z = 2*pi*0.0001;
p = 2*pi*100;
Co = (s + z)/(s*(s+p));
K = 1/bode(Co*CLTF1*Go,2*pi*1);
Co = K*Co;

OLTF = Co*CLTF1*Go;

figure;
bode(OLTF,{0.001,1000});
title('Outer open loop bode plot');
[Gm,Pm] = margin(OLTF);
display(mag2db(Gm), 'Outer gain margin(dB)');
display(Pm, 'Outer phase margin(degrees)');
```

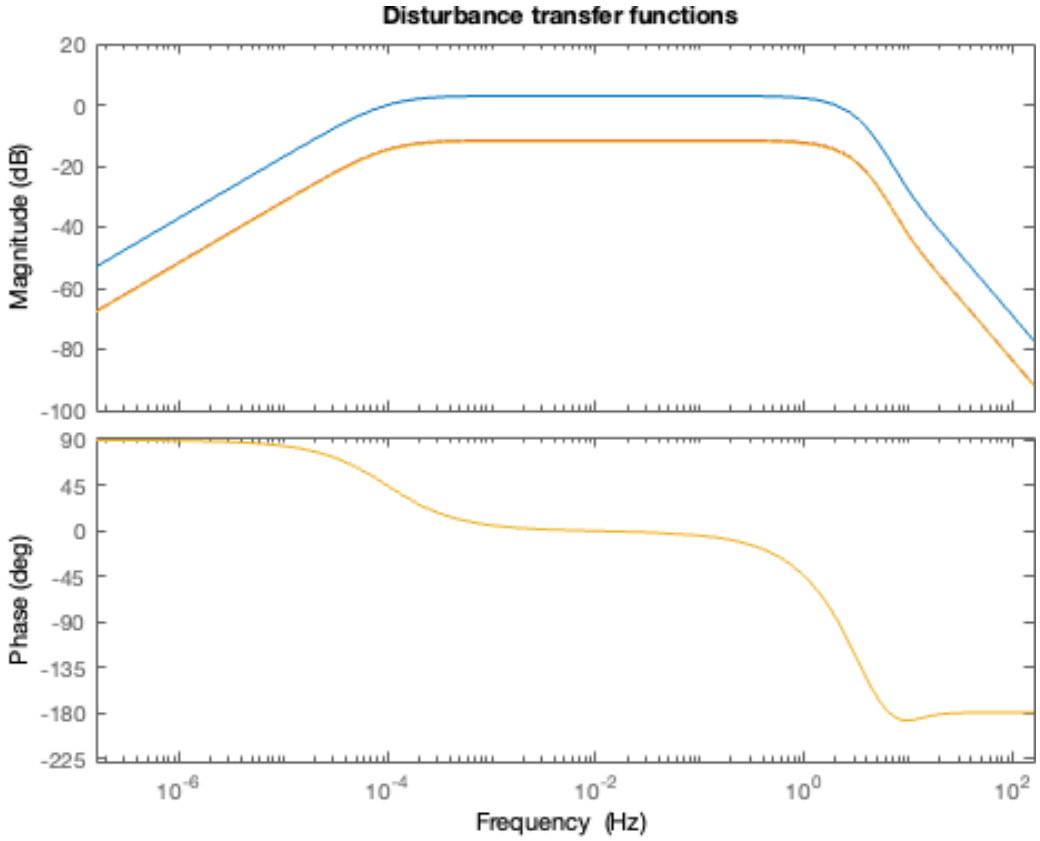
```
Outer gain margin(dB) =

   11.423927687143065
```

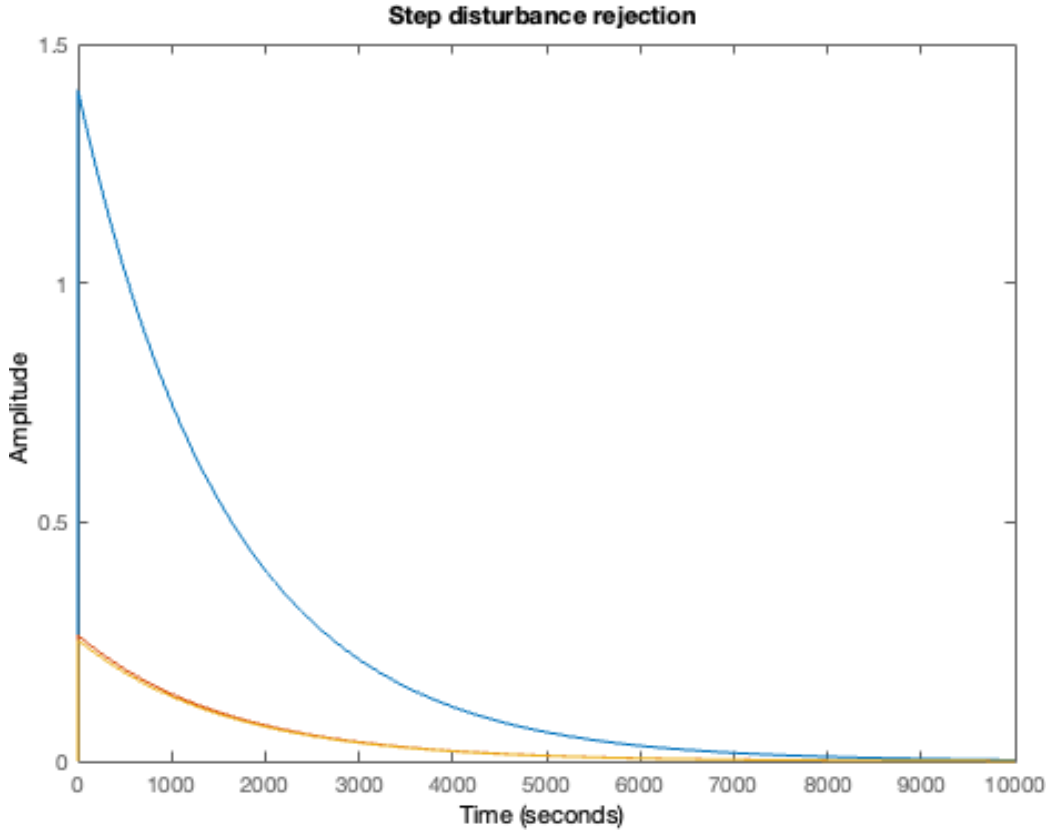
```
Outer phase margin(degrees) =

   66.374799496378543
```



Disturbance transfer functions

```
C1 = C_pade8*Kd1*Gw;
C2 = C_pade8*Kd2*Gw;
C3 = C_pade8*Kd3*Gw;

%The disturbance transfer function is given by:
disturbance1 = G1*Go/(1+C1*G1+Co*C1*G1*Go);
disturbance2 = G2*Go/(1+C2*G2+Co*C2*G2*Go);
disturbance3 = G3*Go/(1+C3*G3+Co*C3*G3*Go);

figure
bode(disturbance1,disturbance2,disturbance3,{1e-6,1000});
title('Disturbance transfer functions');

figure
step(disturbance1,disturbance2,disturbance3);
title('Step disturbance rejection');
```


Disturbance transfer functions
Magnitude (dB)
20
0
-20
-40
-60
-80
-100
Phase (deg)
90
45
0
-45
-90
-135
-180
-225
10^{-6}
10^{-4}
10^{-2}
10^{0}
10^{2}
Frequency (Hz)



Simulink

```
sim('reaction_wheel_1.slx');
```

Published with MATLAB® R2021b

Four control scenarios are evaluated to assess the impact of saturation-aware control strategies and feedforward command design.

System Parameters

2.1 Control Design Requirements:
Gain Margin (GM): ≥ 6 dB
Phase Margin (PM): ≥ 60 degrees
Time Delay: 0.01 seconds

2.2 Reaction Wheel Specifications
Based on the RWP015 datasheet from Blue Canyon Technologies:
Max Stored Angular Momentum: 15 Nms
Max Torque: 4 Nm

Reaction Wheel Dynamics:
Natural Frequency: = 2*pi*10
Damping Ratio: zeta = sqrt(2)/2

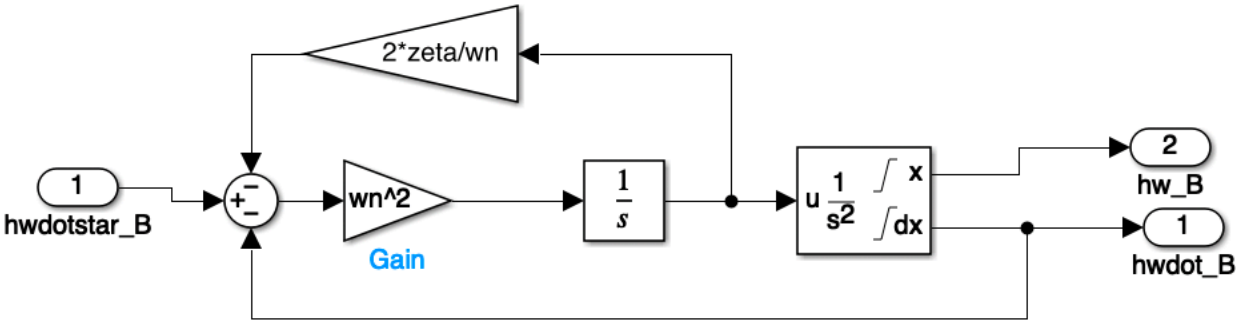
2.3 Initial Conditions:
wbi0_B = [0;0;0]; % Initial Satellite Angular Velocity rad/s
hw0_B = [0;0;0]; % Initial Reaction Wheel Angular Momentum Nms
q0_BI = [1;0;0;0]; % Initial orientation of the B frame relative to the I frame.

2.4 Desired Orientation:
e = [1;2;3]; e = e/norm(e);
qstar_BI = e2q(e,180*pi/180);

3. Control System Design

Scenario 1: Standard Outer Loop Controller

In this scenario, we implemented a step command input for the desired quaternion qstar_BI using the standard outer loop controller composed of a lead compensator with an integrator. This configuration does not account for actuator saturation or integrator windup. The reaction wheel subsystem was made to undergo the following changes to account for saturations:

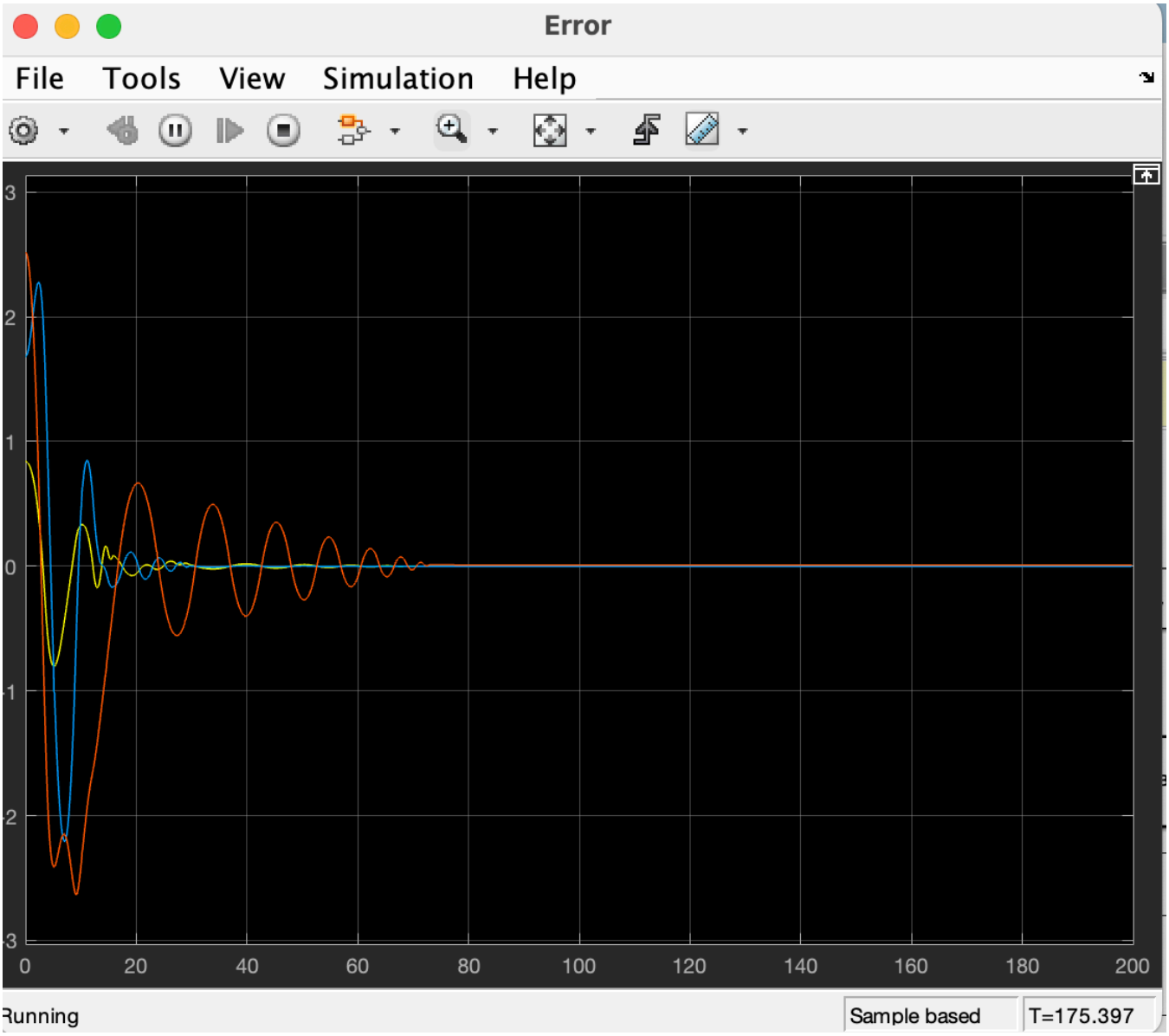


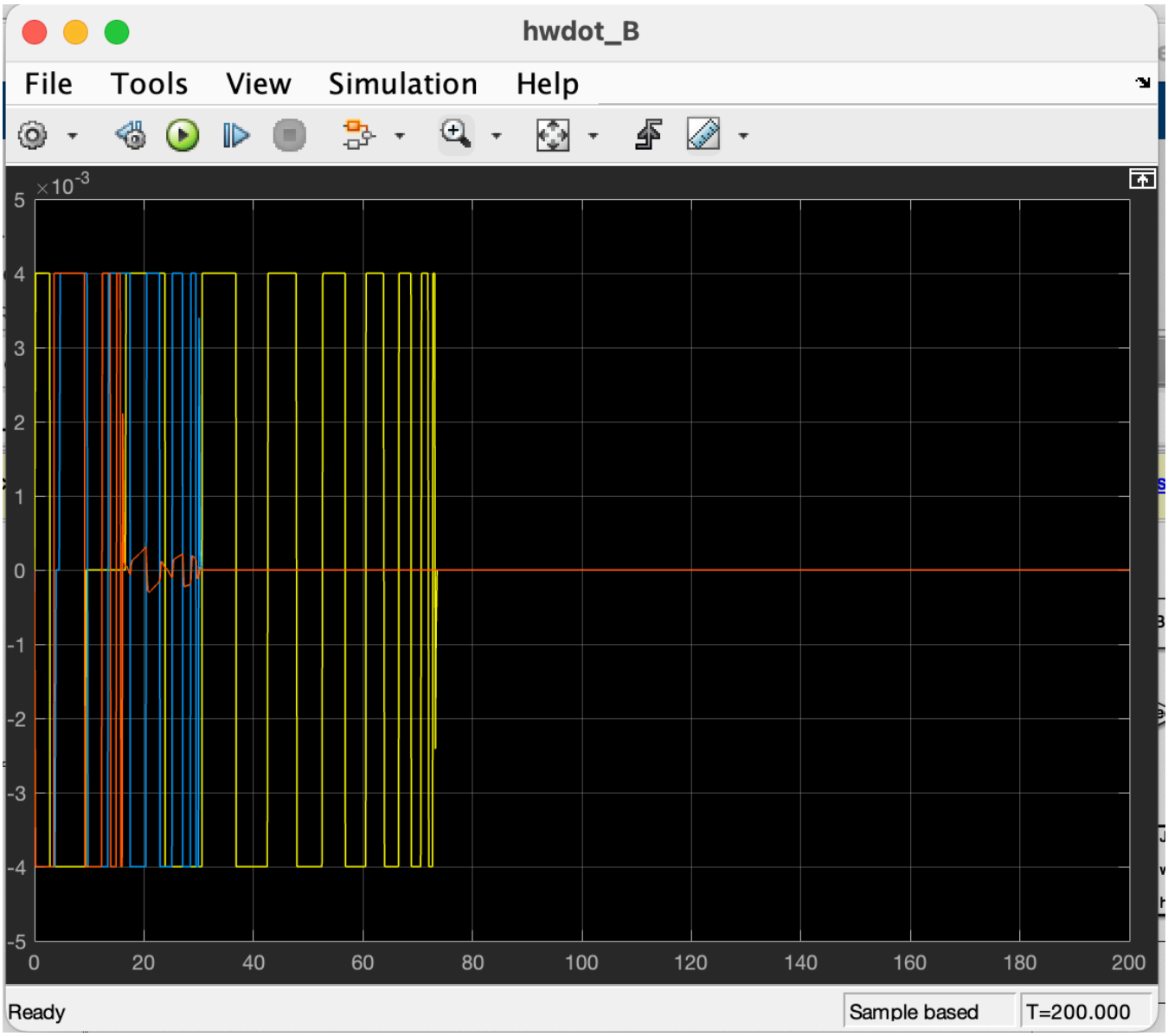
Upon running the simulation, the attitude error, as visualized through the error rotation vector, initially begins to converge toward zero as expected. However, we observe significant oscillatory behavior during the slew maneuver. This oscillation is especially apparent when the control torque demand exceeds the saturation limits of the reaction wheels.

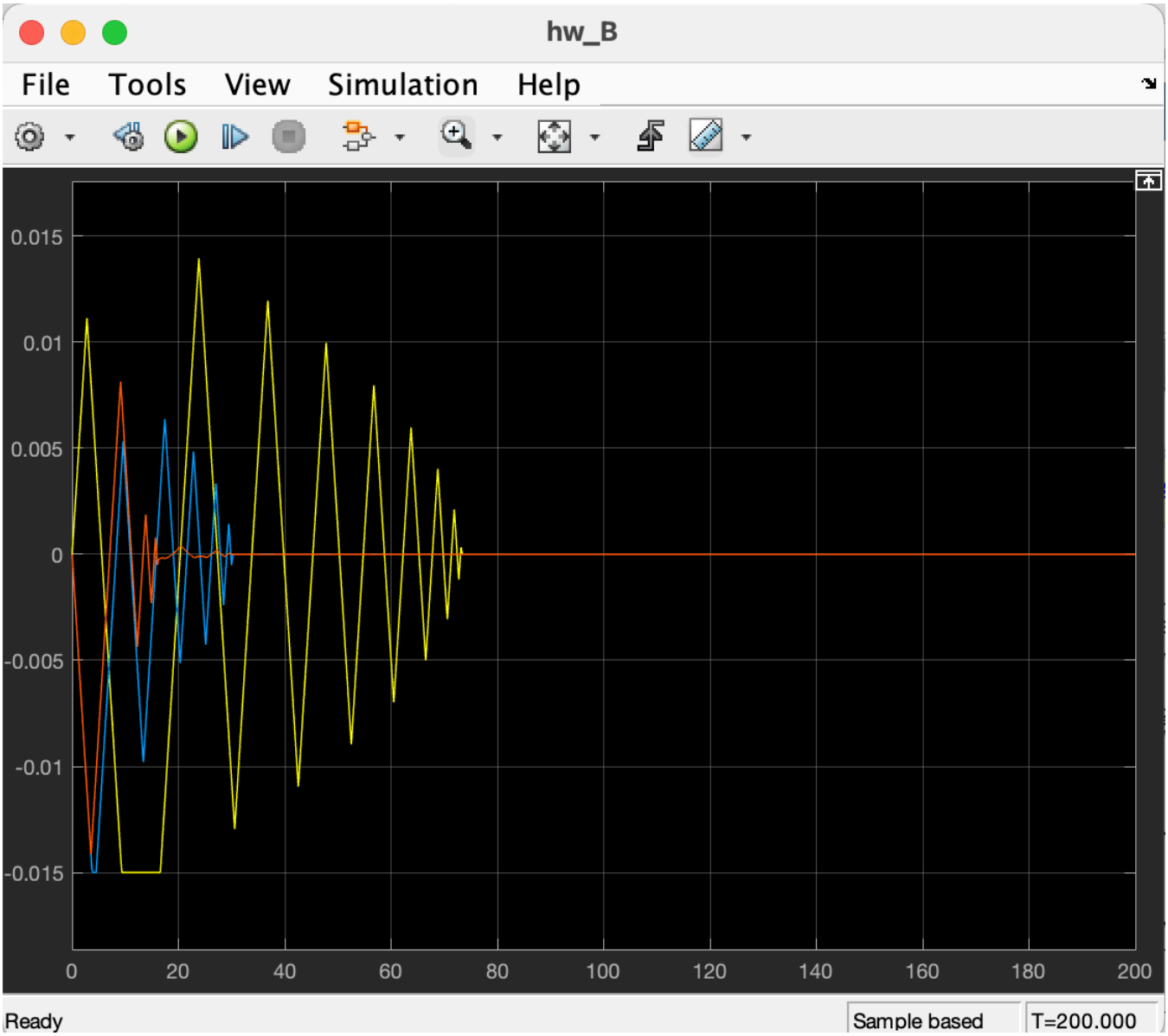
By examining the plots of the reaction wheel torque command hwdot_B and stored angular momentum hw_B, we see clear instances of actuator saturation. These saturations manifest as flat regions in the torque plots where the wheels are constrained at their maximum torque output. As a result, the control authority is lost temporarily, preventing the system from smoothly continuing along the desired trajectory.

Importantly, due to the lack of integrator anti-windup, the integrator continues to accumulate error even when the actuator is saturated. This results in a loss of stability and delayed convergence once the actuator regains authority. The system never truly stabilizes within the desired attitude error threshold.

This behavior highlights the nonlinear effects introduced by actuator saturation, which were not accounted for in the original controller design. Since the standard controller assumes linear operation, the saturation introduces dynamics that are external to the design model, making the closed-loop system performance unpredictable and unstable.




hwdot_B
File
Tools
View
Simulation
Help
×10^-3
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
20
40
60
80
100
120
140
160
180
200
Ready
Sample based
T=200.000



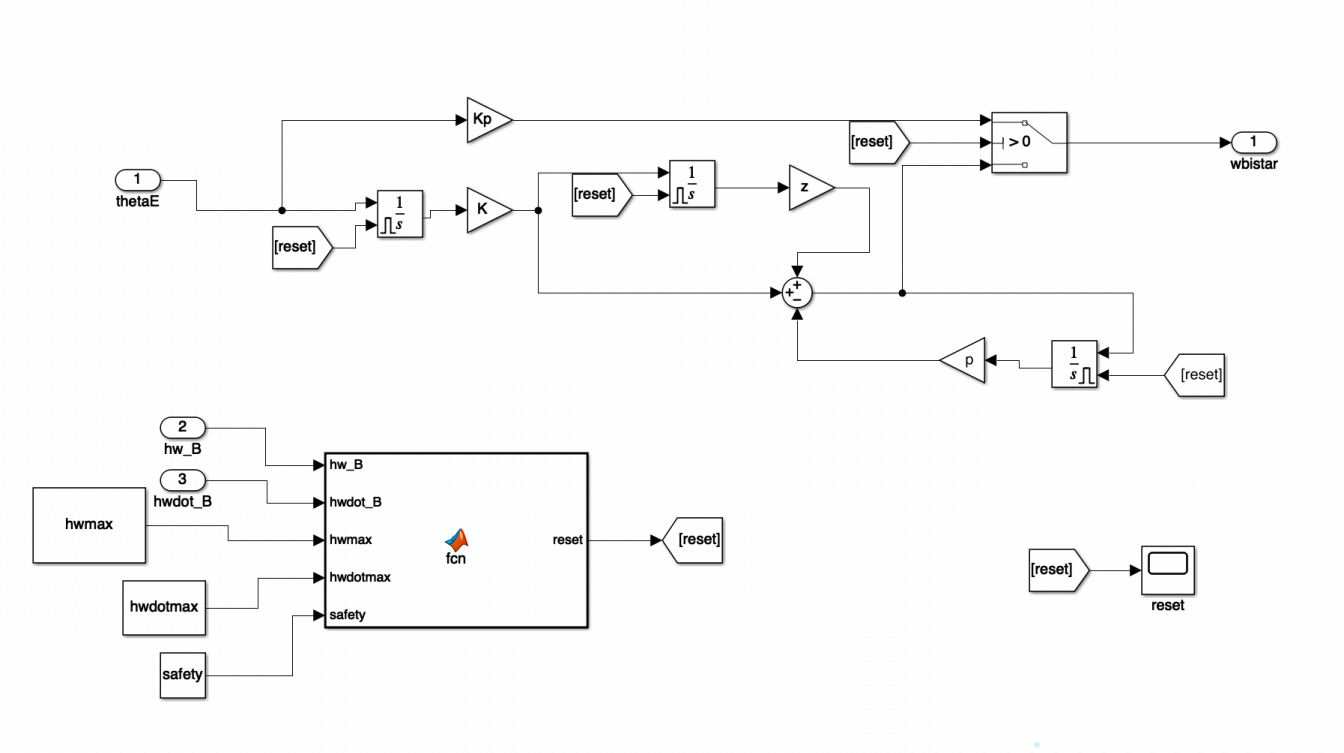
Scenario 2: Outer Loop Controller with Integrator Windup

In this scenario, the system was modified to address the issues caused by actuator saturation in Scenario 1. Specifically, an anti-windup strategy was implemented to mitigate the adverse effects of integrator windup. The motivation for this change stems from the observation that, under saturation conditions, the rotation error thetaE cannot be driven to zero. As a result, the integrator continues to accumulate error, leading to an unbounded increase in the commanded angular velocity.

In many systems, this can result in sustained oscillations known as a limit cycle, where the system continuously overshoots due to an over-aggressive integrator. Although in this case a limit cycle was not observed—likely due to the specific placement of zeros and poles in the system—the potential for instability remains if left unaddressed.

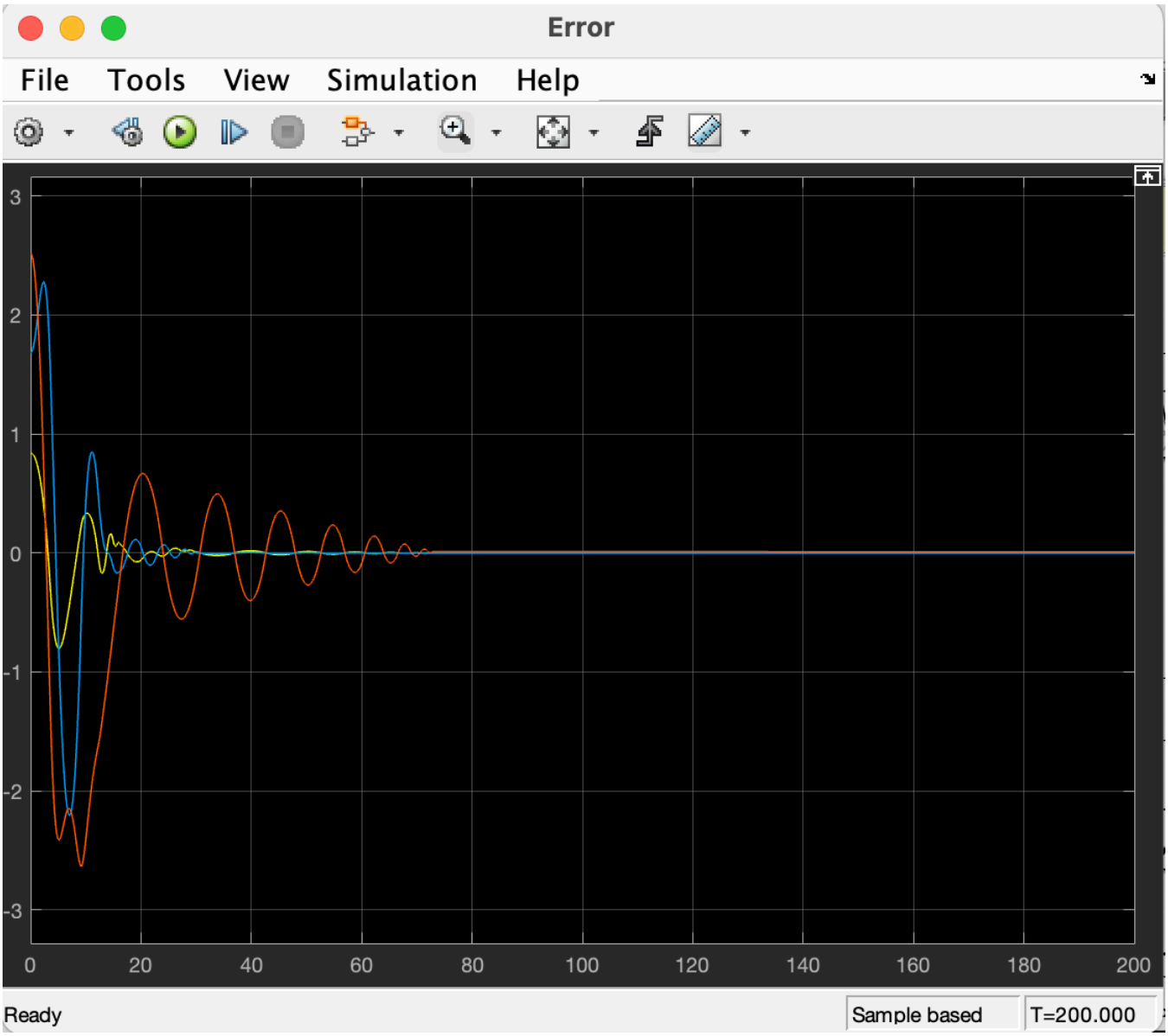
To counter this, a proportional-only mode was introduced when the system approaches saturation. During these periods, the integrators are reset, effectively pausing their contribution to the control action. This is reflected in the updated block diagram, where the three linear time-invariant (LTI) components used previously (i.e., the lead compensator with an integrator) were replaced with a

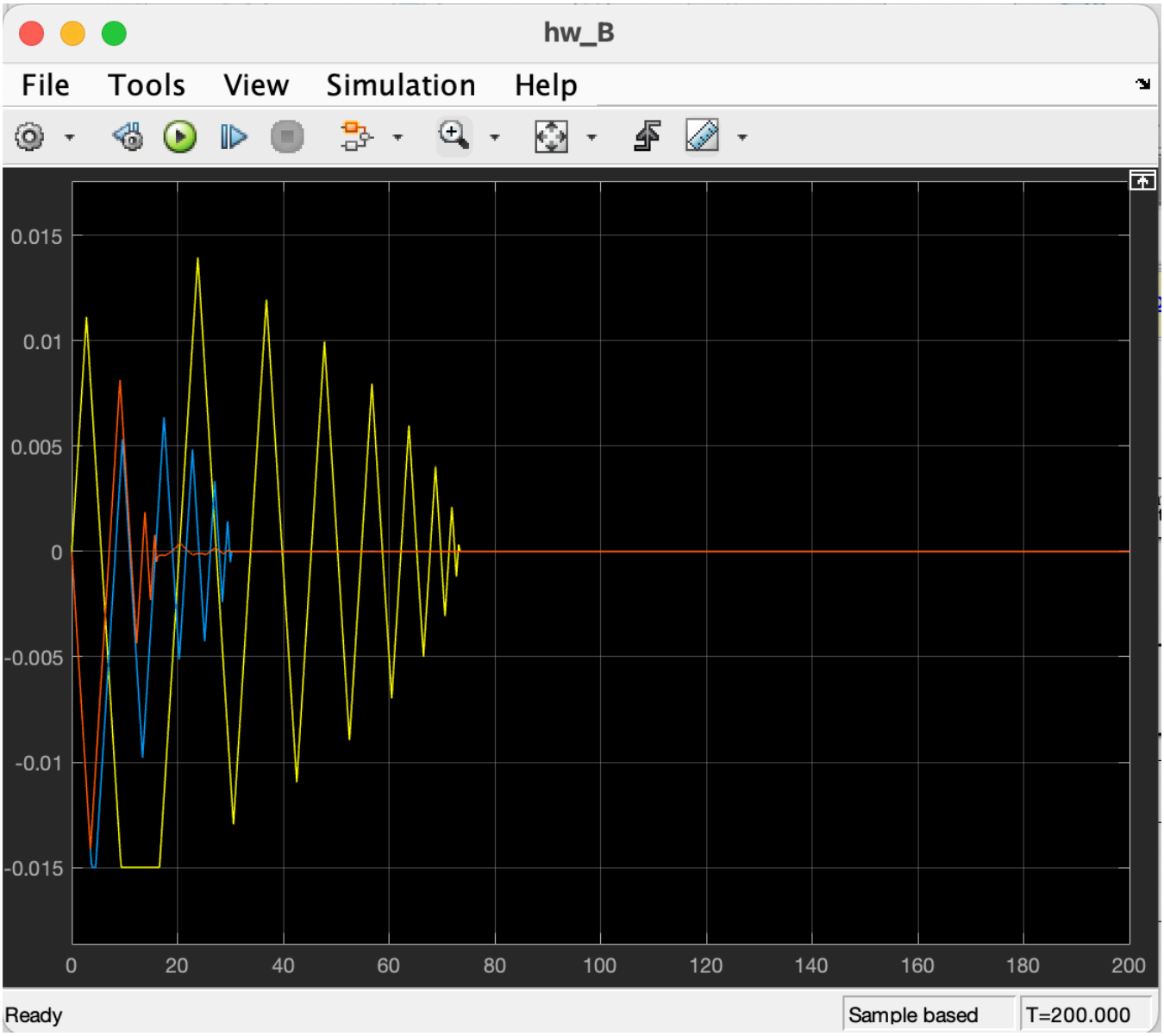
single subsystem implementing the same transfer function but with integrator reset logic embedded.

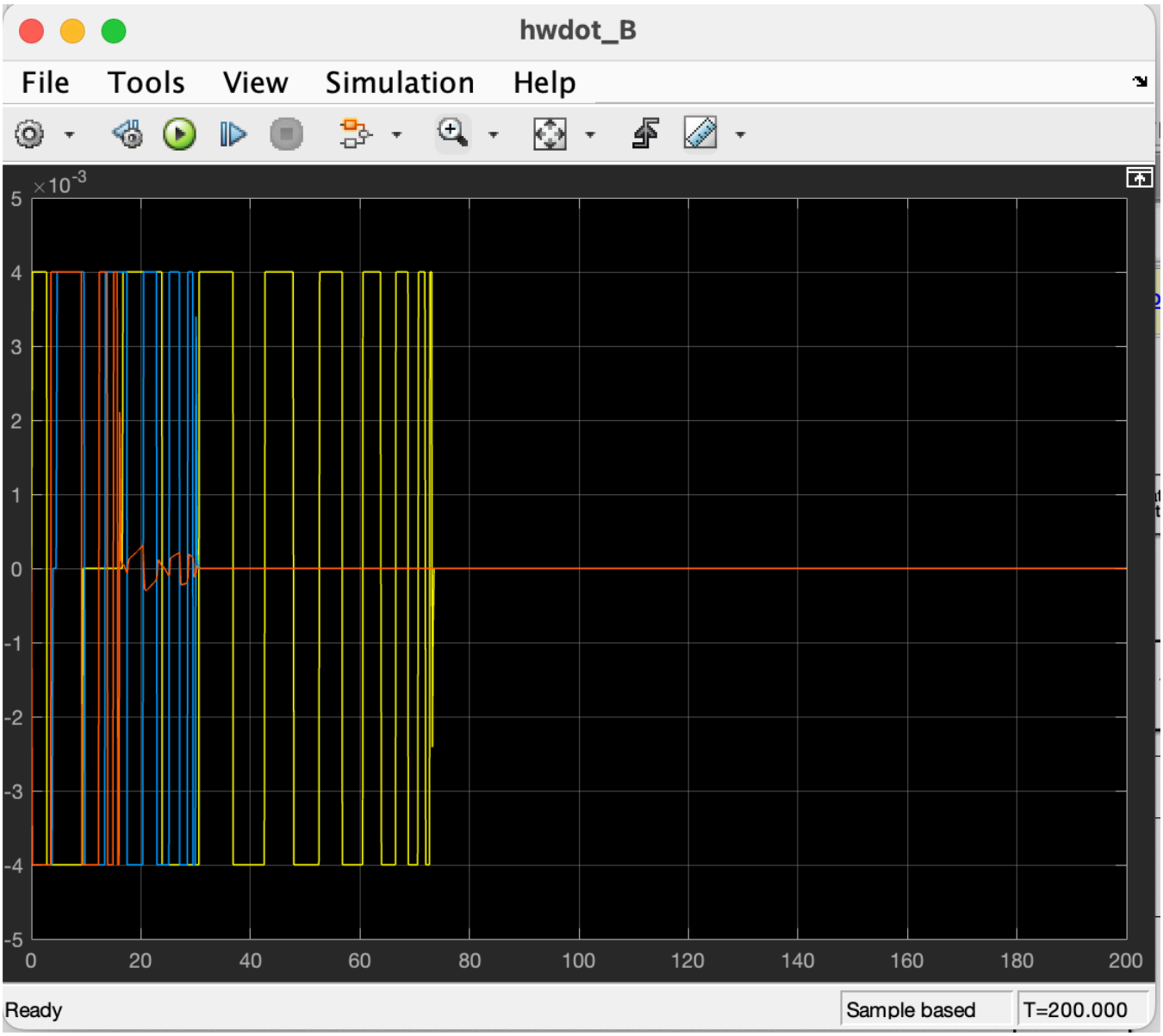


As a result, the controller still enforces the same dynamic relationship between the error signal and the commanded angular velocity but now includes logic to prevent excessive integrator buildup during saturation.

From the output plots, we observe that this modification successfully prevents the system from entering a limit cycle. However, significant overshoot is still evident in the error signal, likely due to the sudden re-engagement of the integrators after reset, or the transition between proportional and full compensator modes. While stability has improved, there is a trade-off with performance in terms of transient error.


Error
File
Tools
View
Simulation
Help
3
2
1
0
-1
-2
-3
0
20
40
60
80
100
120
140
160
180
200
Ready
Sample based
T=200.000


hw_B
File
Tools
View
Simulation
Help
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
20
40
60
80
100
120
140
160
180
200
Ready
Sample based
T=200.000



Scenario 3: Trajectory-Based Command Tracking

In this scenario, the control architecture was extended to include a trajectory generator that produces a smooth, time-parameterized quaternion command qstar_PI . The objective was to ensure that the angular velocity and angular acceleration commands are constrained to lie within allowable limits defined by the maximum wheel momentum and maximum wheel torque respectively.

A MATLAB function `trajectory_scalar` was developed to generate a scalar profile for angular acceleration , angular velocity , and angular displacement . This function computes either a triangular or trapezoidal angular profile depending on the total desired rotation angle. The trajectory is designed such that:
- If the required angle can be achieved before the maximum angular velocity is reached, a triangular profile is used.
- Otherwise, the profile transitions into a trapezoidal profile with an intermediate coasting phase at maximum angular velocity.

```
function[wbidotstar, wbistar, angle] =
trajectory_scalar(thetastar,J_C_P,hwmax,hwdotmax,safety,t)

%Calculate maximum allowable satellite angular acceleration
wbidotmax = safety*hwdotmax/max(max(J_C_P));

%Calculate maximum allowable satellite angular velocity
wbimax = safety*hwmax/max(max(J_C_P));

%Calculate time to torque at maximum until maximum angular velocity is
%reached
tmax = wbimax/wbidotmax;

%Calculate angle that can be traveled during maximum torque
anglemax = 1/2*wbidotmax*tmax^2;

% If the angle traveled during maximum torque is greater than half the
% total angle then the trajectry will be triangle not a trapezoid
if anglemax > thetastar/2

    %the time at the top of the triangle
    t1 = sqrt(thetastar/wbidotmax);

    %The angle traveled at t1
    angle1 = thetastar/2;

    %the angular velocity at t1
    w1 = wbidotmax*t1;

    %Because we are a triangle there is not going to be any coasting period
    t2 = t1;
    angle2 = angle1;

    % The final time will be twice the time it took to get to the top
    tf = t2 + t1;

else

    %Calculate the angle traveled during the coast period
    anglecoast = thetastar - 2*anglemax;

    %Calculate the time to coast
    tcoast = anglecoast/wbimax;

    %Torque for the maximum allowable amount of time
    t1 = tmax;

    %Calculate the angle traveled at that point
    angle1 = anglemax;

    %Calculate the angular velocity at that point
    w1 = wbimax;

    %Calculate the time at the end of the coast
    t2 = t1 + tcoast;

    %Calculate the angle traveled at that point
    angle2 = angle1 + w1*tcoast;
```

```
    %Calculate the final time
    tf = t2 + t1;

end

%After the final team hold the position
if t >= tf

    wbidotstar = 0;
    wbistar = 0;
    angle = thetastar;

    %Deceleration time period
elseif t>= t2

    %Torque at negative maximum
    wbidotstar = -wbidotmax;

    %Agular velocity decrease linearly
    wbistar = w1 - wbidotmax*(t - t2);

    %Angle decreases parabolically
    angle = angle2 + w1*(t - t2) - 1/2*wbidotmax*(t - t2)^2;

    %Coasting time period
elseif t >= t1

    %No acceleration
    wbidotstar = 0;

    %Constant angular velocity
    wbistar = w1;

    %Angle increases linearly
    angle = angle1 + w1*(t - t1);

else

    %Accelerating time period
    %Torque at maximum allowable
    wbidotstar = wbidotmax;

    %Angular velocity increases linearly
    wbistar = wbidotmax*t;

    %Angle increases parabolically
    angle = 1/2*wbidotstar*t^2;

end

end
```

These profiles ensure compliance with the actuator constraints, applying safety margins through a scaling factor to reduce risk of saturation. The scalar profile is then extended to three dimensions using the function `trajectory`, which:

1. Projects the initial and final quaternions to the principal axis frame (P-frame).

2. Computes the quaternion error and extracts the corresponding axis-angle representation.
3. Applies the scalar trajectory to generate time-varying angular velocity and acceleration vectors aligned with the maneuver axis.
4. Reconstructs a smooth quaternion command using the integrated angle and axis.

```
function [wbidotstar_P, wbistar_P, qstar_PI] = trajectory(qstar_BI, q0_BI, q_PB,
J_C_P, hwmax, hwdotmax,safety,t)

% wbidotstar_P = zeros(3,1);
% wbistar_P = zeros(3,1);
% qstar_PI = qstar_BI;

%Project intial and desired quaternion to principal frame
qstar_PI = qX(q_PB,qstar_BI);
qstar_PI = qUnit(qstar_PI);

q0_PI = qX(q_PB,q0_BI);
q0_PI = qUnit(q0_PI);

%Calculate the error quaternion between desired and initial
qe = qX(qstar_PI,qT(q0_PI));
qe = qUnit(qe);

%Convert the quaternion to an axis and angle
[e, thetastar] = q2e(qe);

[wbidotstar, wbistar, angle] =
trajectory_scalar(thetastar,J_C_P,hwmax,hwdotmax,safety,t);

%Project acceleration and velocity to the P frame
wbidotstar_P = wbidotstar*e;
wbistar_P = wbistar*e;

%Create the desired quaternion
qrot = e2q(e,angle);
qstar_PI = qX(qrot,q0_PI);
qstar_PI = qUnit(qstar_PI);

end
```

This function is called in real time from the Simulink trajectory generator block, which serves as the source of the quaternion command.

Observations from Simulation:

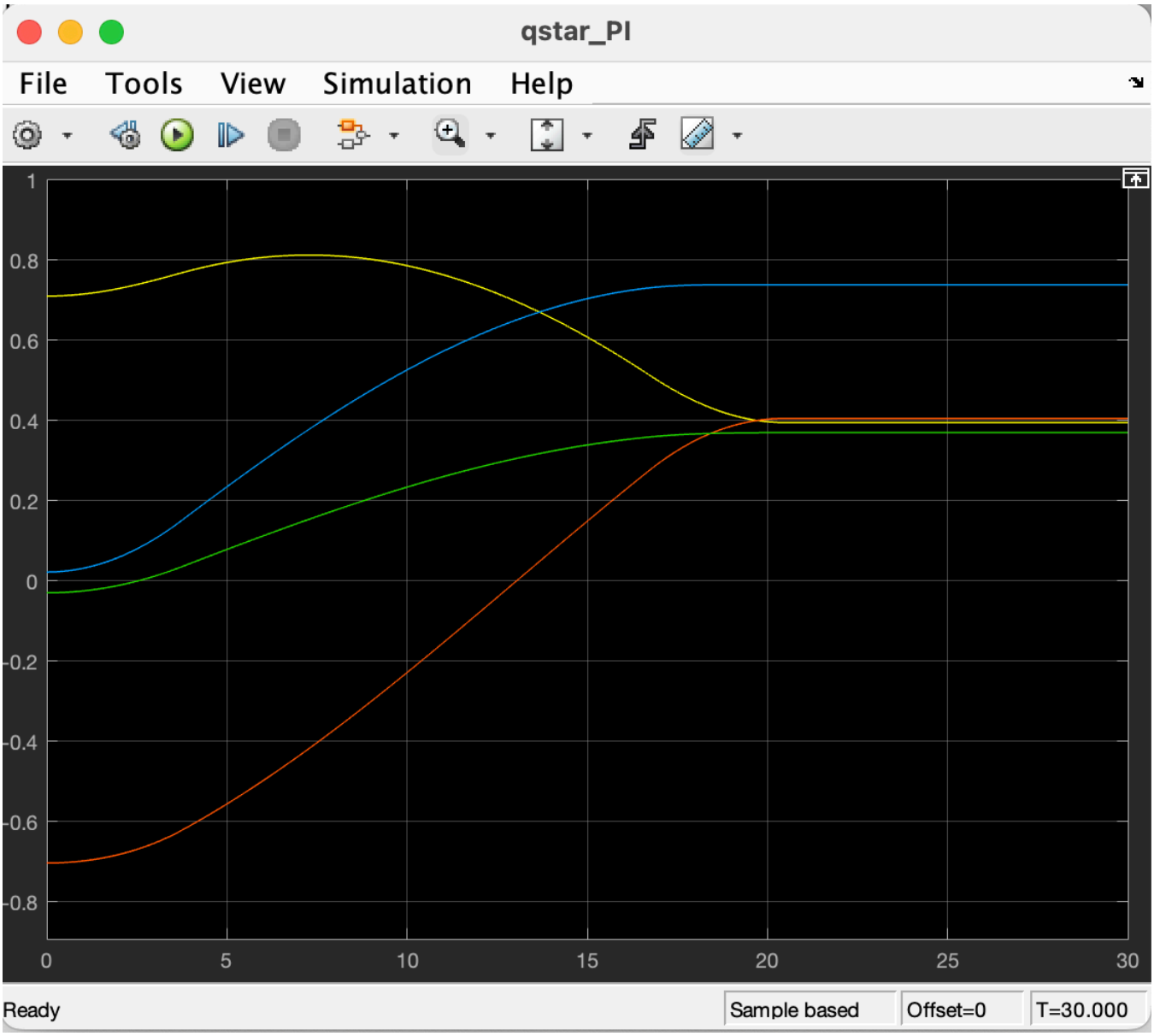
- Quaternion Tracking: The command quaternion transitions smoothly from the initial to the final attitude. The actual quaternion closely tracks this command, indicating excellent closed-loop performance.

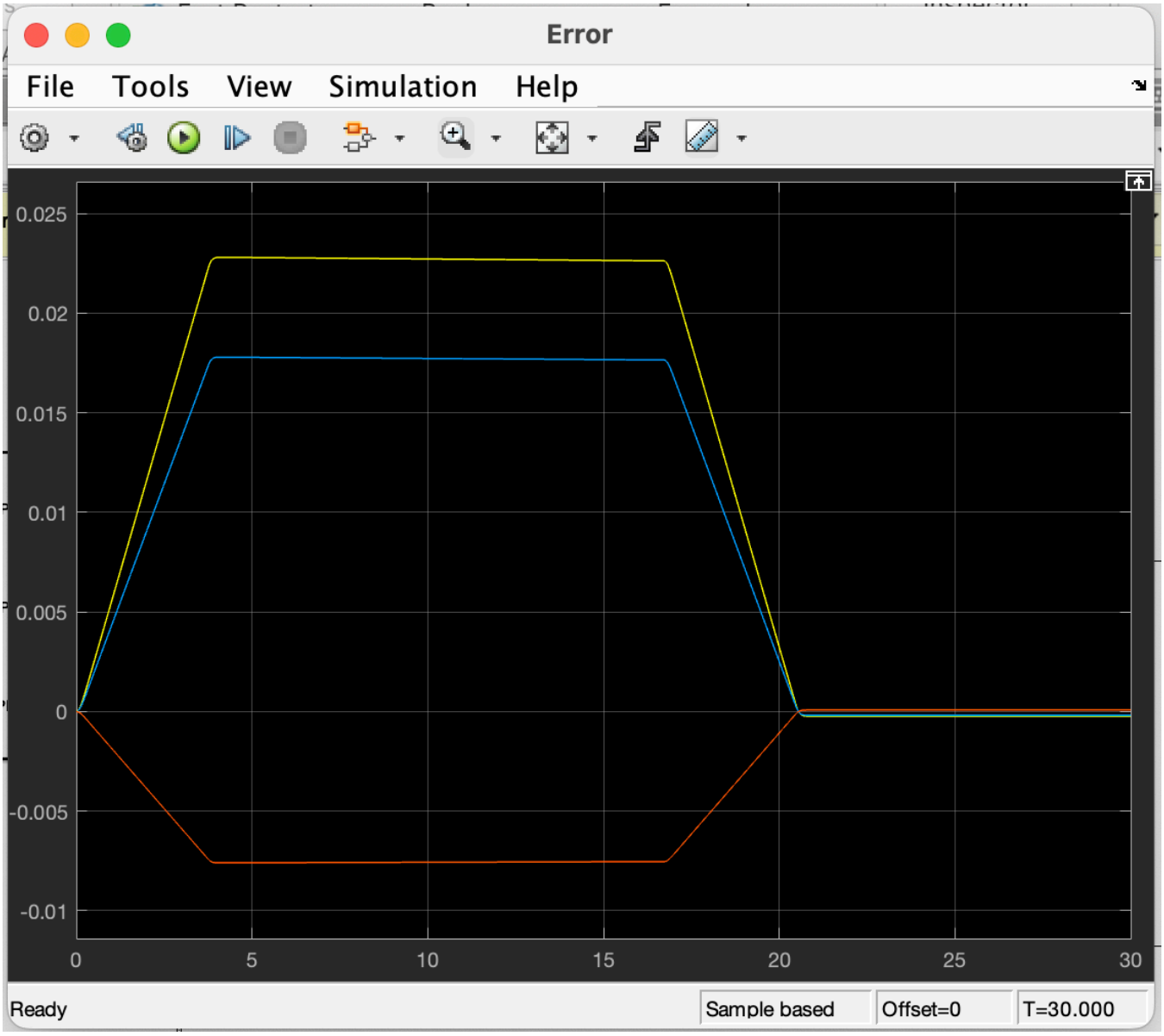
- Angular Momentum Profile: The reaction wheels follow a trapezoidal angular momentum profile, demonstrating that the controller operates within saturation limits. There are no sharp discontinuities, confirming that the profiled trajectory is within wheel capacity.

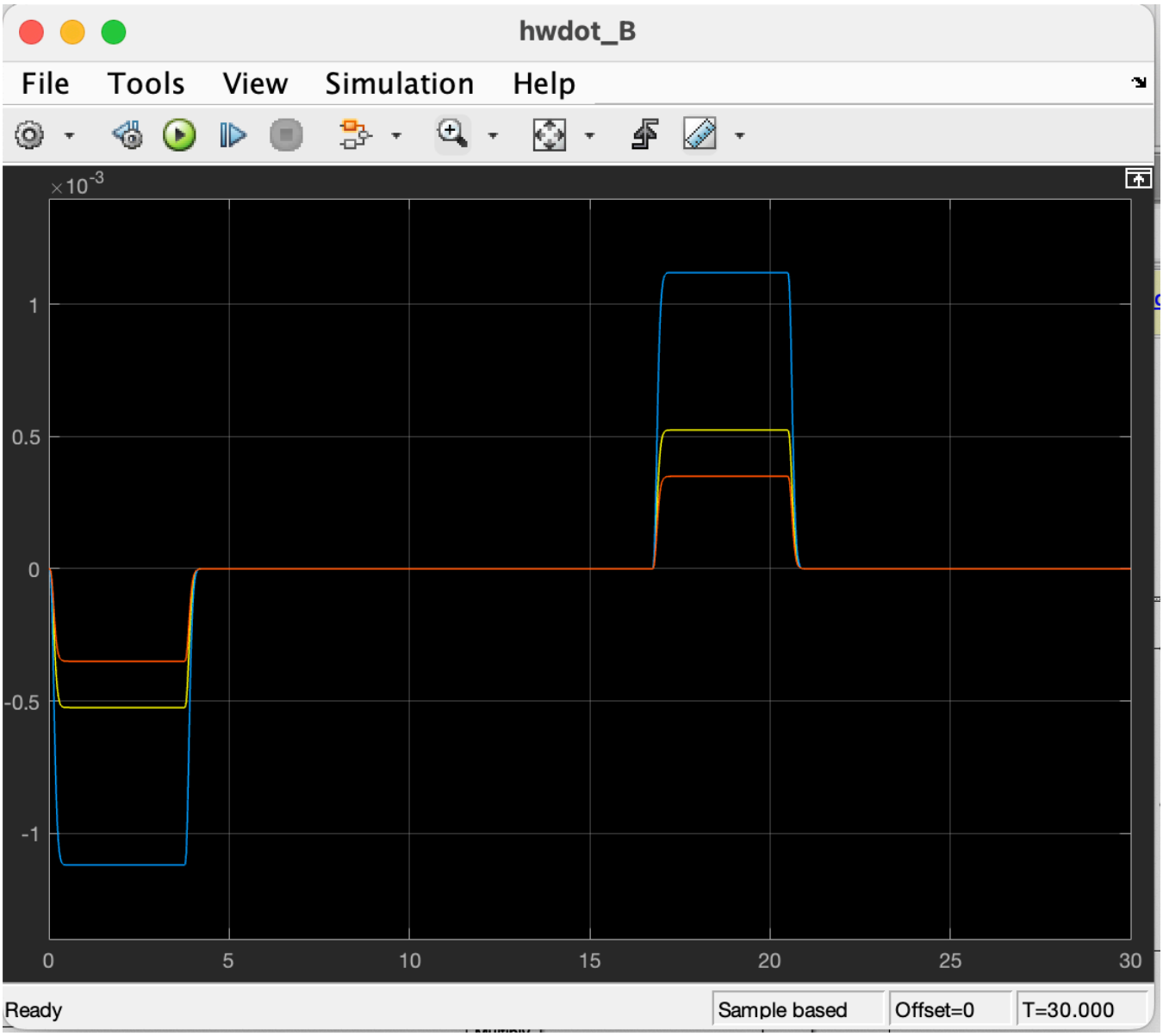
- Torque Profile: The torque also remains within bounds, though its shape reflects smoother curves

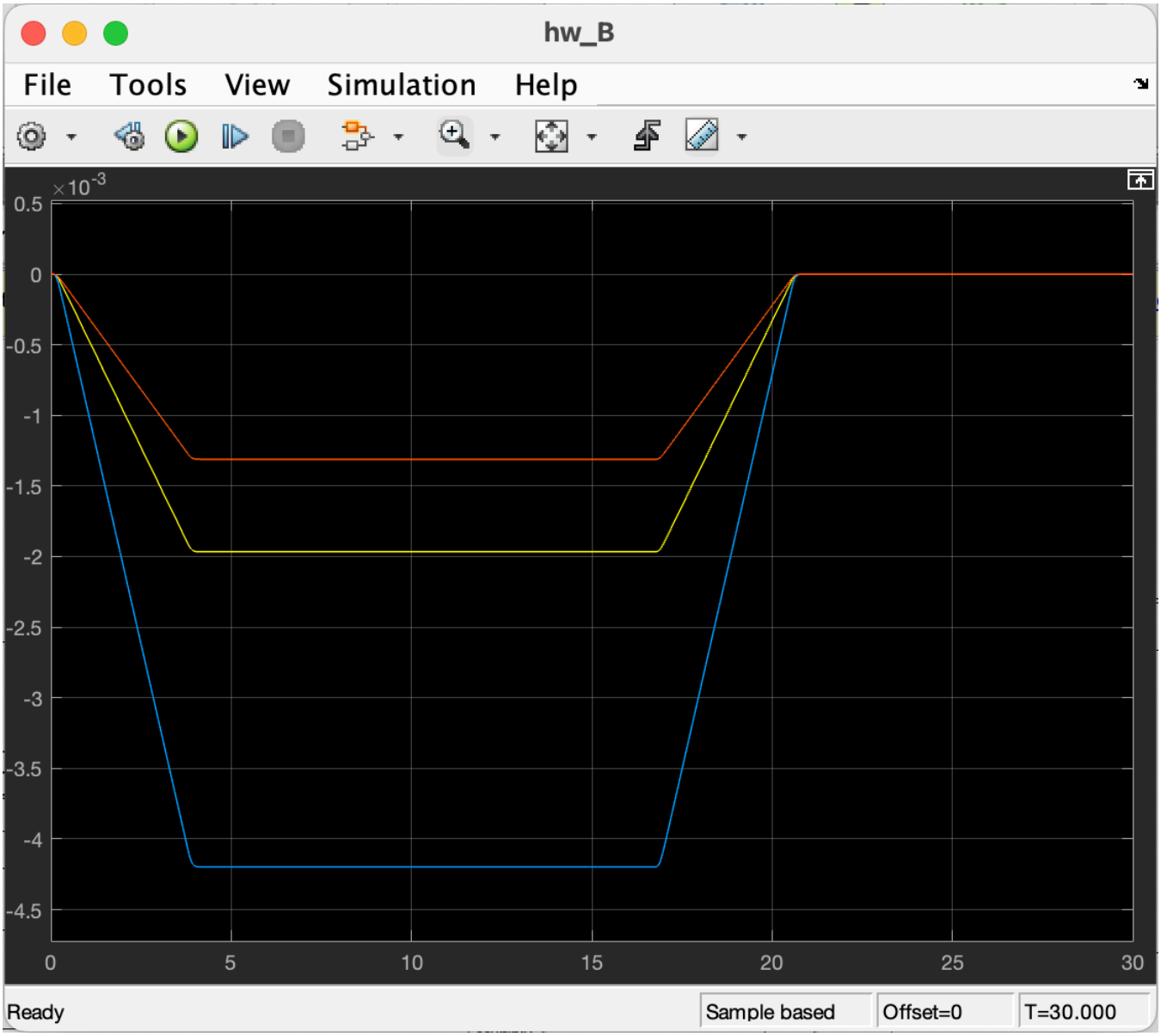
at the transitions, which is expected since the wheels cannot respond instantaneously to high-frequency inputs. This results in rounded corners rather than sharp torque changes.

- Error Behavior: The quaternion error initially reflects the full rotational discrepancy, then gradually reduces as the system tracks the profiled angle. Because the desired quaternion varies linearly in angle, the controller takes time to follow, so the error only gradually trends to zero. During deceleration, the error continues to shrink, ultimately reaching zero at the final attitude.




Error
File
Tools
View
Simulation
Help
0.025
0.02
0.015
0.01
0.005
0
-0.005
-0.01
0
5
10
15
20
25
30
Ready
Sample based
Offset=0
T=30.000


hwdot_B
File
Tools
View
Simulation
Help
×10^-3
1
0.5
0
-0.5
-1
0
5
10
15
20
25
30
Ready
Sample based
Offset=0
T=30.000



Scenario 4: Trajectory-Based Control with Command Feedforward

The goal is to track a continuous trapezoidal trajectory from an initial attitude to a desired final attitude using a reaction wheel controller that incorporates command feedforward. Unlike step-based control, this approach ensures that the controller follows the entire path, not just the final attitude. This method significantly improves transient performance and reduces attitude tracking error.

When using traditional feedback-only control, the system cannot respond instantaneously due to time delay and actuator dynamics. Although the controller eventually compensates for these lags, the result is a delay in achieving the desired attitude and increased attitude error during maneuver transitions. By contrast, command feedforward uses the known mass properties and the desired trajectory (angular velocity and acceleration) to generate a nominal torque profile required to perfectly track the trajectory, assuming an ideal system.

In Simulink, we pre-multiply the desired angular acceleration ω_BI and velocity ω_BI by the inertia

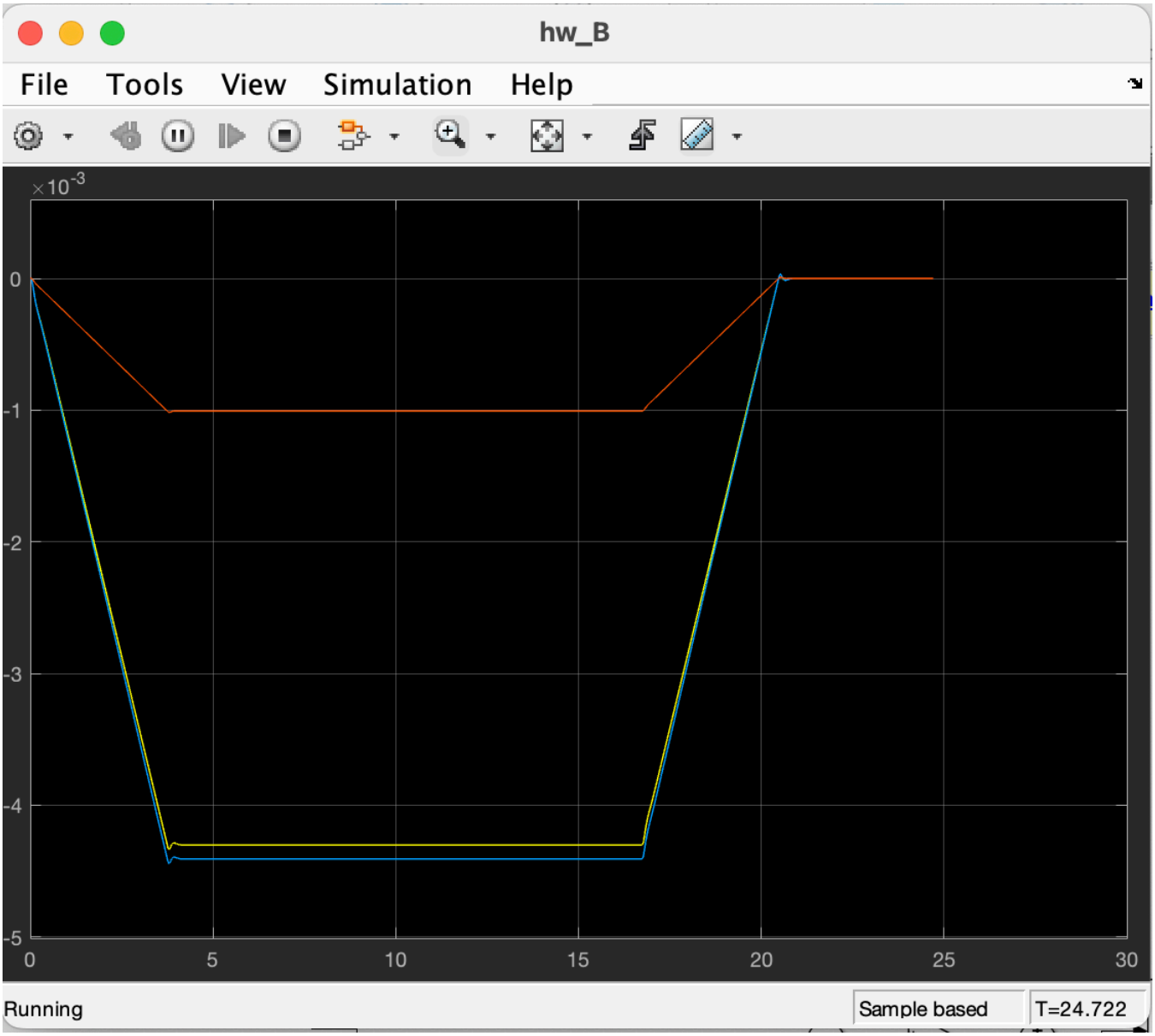
matrix J_C_P the inner loop gain Kd, respectively. These form the feedforward torque, which is added to the feedback torque before being applied to the plant.

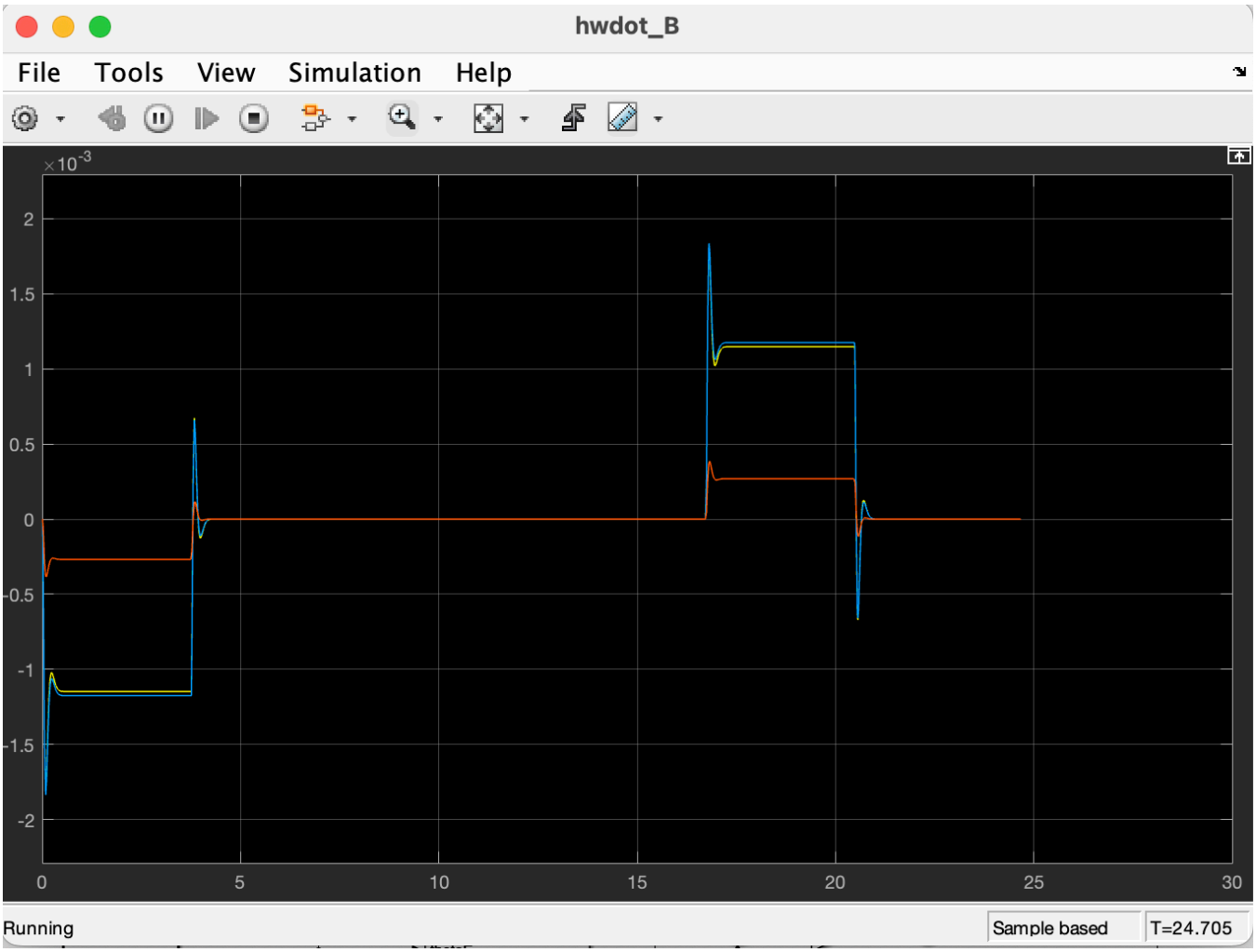
To remain within actuator limits, a 50% safety factor is applied — meaning the trajectory is designed to not exceed 50% of the maximum torque and angular momentum capacity of the reaction wheels. This provides headroom for the controller to generate additional torque to compensate for model uncertainties and unmodeled dynamics.

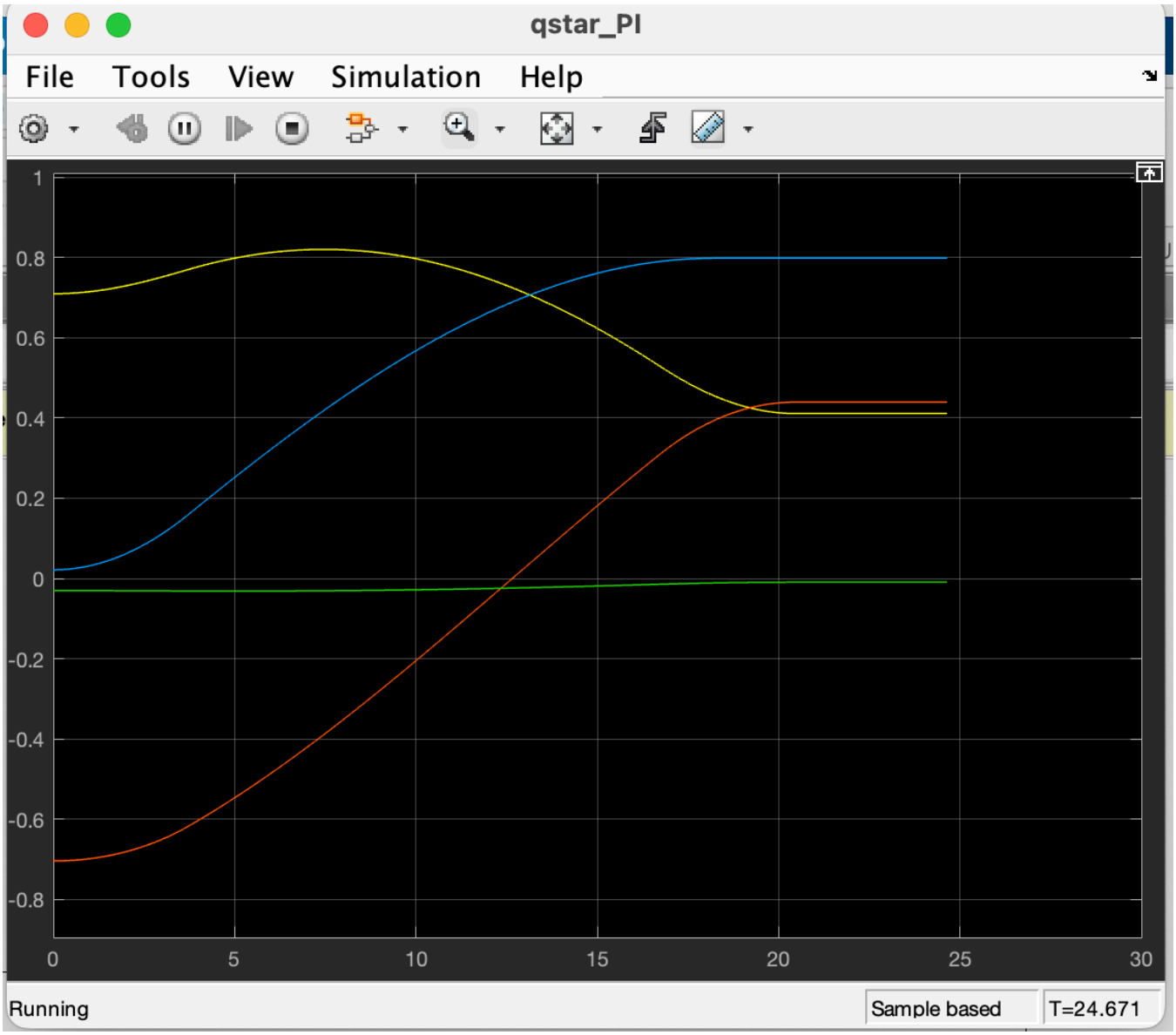
The performance of feed forward command was significantly better than previous scenarios:

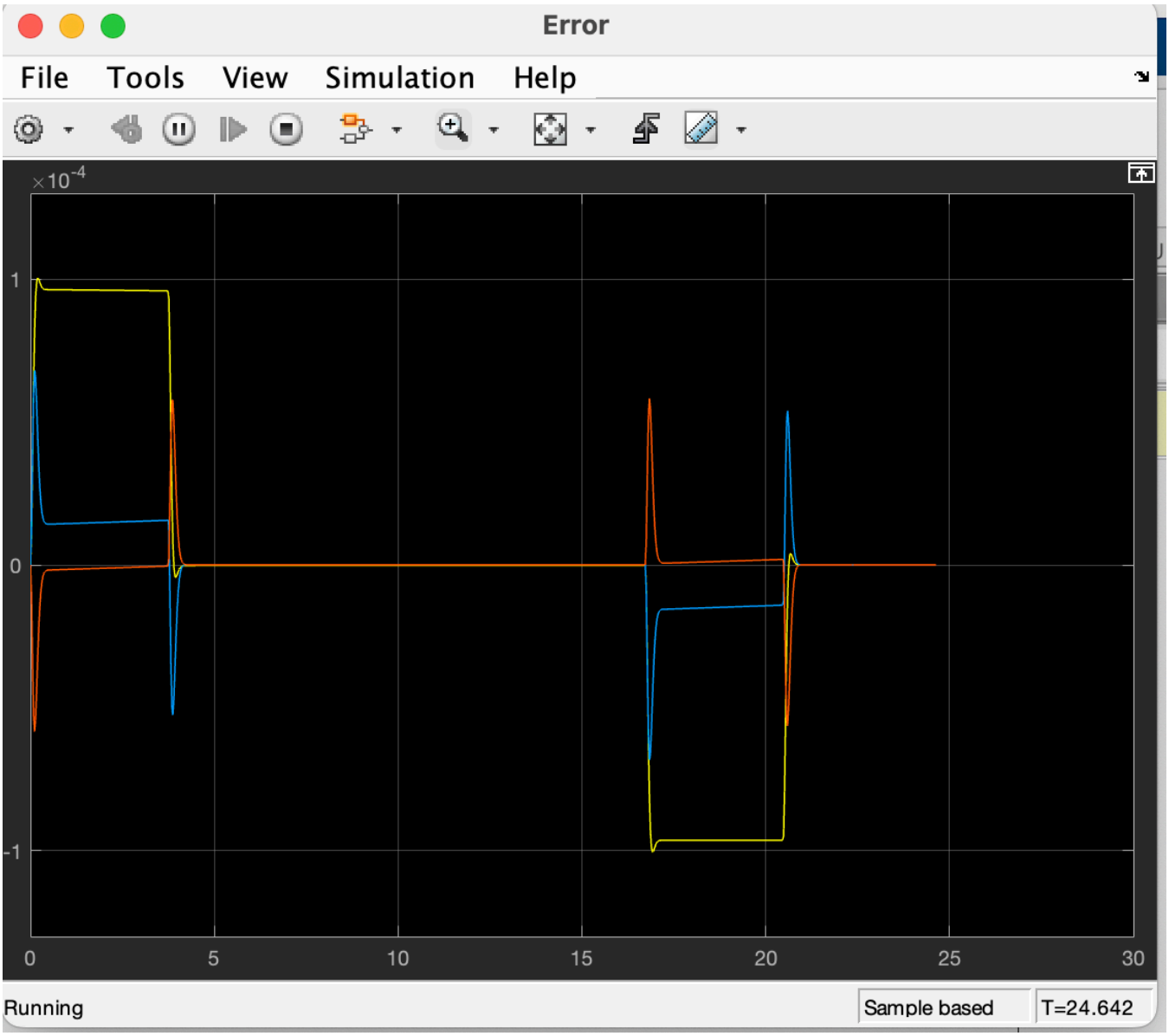
- Tracking Error: The attitude error reduced to the order of 10^−4, and during coasting phases, even to 10^−6, demonstrating precise path tracking.
- Reaction Wheel Torque: The hwdot_B plot shows a distinct peak at the beginning and end of the maneuver, corresponding to the controller compensating for initial time delay and actuator dynamics. This feedforward torque helped align the system to the trajectory early, minimizing deviation.
- Margin of Safety Justification: Despite generating a trajectory that uses only 50% of actuator capacity, the controller occasionally required more torque to stay on track further reinforcing the importance of the safety margin to absorb unmodeled effects.

This experiment demonstrates that command feedforward enables near-perfect tracking in the presence of time delay and actuator saturation, provided a proper trajectory and control architecture are used. The trajectory profile is not only smoother, but also respects actuator limits, while the controller dynamically compensates for imperfections and disturbances.


hw_B
File
Tools
View
Simulation
Help
×10^-3
0
-1
-2
-3
-4
-5
0
5
10
15
20
25
30
Running
Sample based
T=24.722


hwdot_B
File
Tools
View
Simulation
Help
×10^-3
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
Running
Sample based
T=24.705


qstar_PI
File
Tools
View
Simulation
Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
5
10
15
20
25
30
Running
Sample based
T=24.671



Derivation of the Command Feedforward:

We define the tracking error in attitude:
$\theta_E = \theta^* - \theta$

The desired angular velocity is generated using the outer-loop controller:
$\omega_{BI}^* = C_O * \theta_E$

The error in angular velocity is:
$\omega_E = \omega_{BI}^* - \omega_{BI}$

The inner loop uses this to compute torque:
$\omega_{BI} = C_1 \omega_E + H\theta^* G_1$

Finally, the attitude dynamics are governed by:
$\theta = \omega_{BI} G_O$

```
Assuming the transfer function from θ* to θ should be unity (i.e.,
perfect tracking), we solve:
θ / θ* = (C_I C_O G_I G_O + H G_I G_O) / (1 + C_I C_O G_I G_O + C_I G_I)
= 1

Solving for H:
H = 1 / (G_1 G_O) + C_1 / G_O

Substituting:
- C_1 = k_d,
- G_O = 1 / s,
- G_1 = 1 / (Js),

We get:
H = J s^2 + K_d s

The command feedforward torque is then:
T_FF = J θ¨* + K_d θ˙*
```

The redesigned control system successfully manages reaction wheel limitations and maintains system stability and performance across all tested scenarios. The feedforward-enhanced, saturation-aware control architecture provides the best performance with minimum error and torque usage.

4. Results and Discussion:

Scenario 1: Standard Outer Loop Controller

The initial configuration using a lead compensator with integrator demonstrated severe limitations when exposed to actuator saturation. Although the system showed initial convergence toward zero attitude error, large oscillations quickly emerged during high-torque phases of the maneuver. These oscillations were directly linked to the reaction wheels reaching their saturation limits, as evidenced by plateaued regions in the torque plots.

Critically, the integrator continued accumulating error during these saturated periods, leading to instability and delayed convergence. The absence of anti-windup mechanisms caused the controller to behave as though it retained authority, resulting in a persistent mismatch between the commanded and actual system dynamics. Overall, the system exhibited poor robustness and performance under realistic actuator constraints.

Scenario 2: Outer Loop Controller with Integrator Windup

To mitigate the issues observed in Scenario 1, integrator windup logic was implemented. This approach involved disabling the integrator when saturation was detected, transitioning the controller to a proportional-only mode and resetting accumulated error.

Simulation results showed that this modification significantly improved the system's stability by preventing uncontrolled error growth. The system no longer entered into unstable oscillations or limit cycles, even when actuator limits were reached. However, overshoot and transient error remained, especially during transitions between saturation and normal operation. The system was more stable but still lacked the precision required for high-performance attitude maneuvers.

Scenario 3: Trajectory-Based Command Tracking

In this scenario, a trajectory generator produced time-parameterized quaternion commands based on the actuator capabilities. The command profile constrained angular velocity and acceleration to prevent saturation, using triangular or trapezoidal motion profiles depending on the maneuver magnitude.

Results indicated marked improvement. The controller was able to track the command quaternion smoothly, and both the torque and momentum remained within bounds throughout the maneuver. Unlike the previous scenarios, there was no evidence of saturation-induced instability or significant overshoot. However, due to the time-dependent nature of the command, the attitude error decreased gradually over the trajectory duration and only approached zero at the final time. While transient accuracy was improved, some delay in error reduction was still present.

Scenario 4: Trajectory-Based Control with Command Feedforward

This final configuration incorporated command feedforward in addition to the trajectory generator. By precomputing the required torque from the desired angular acceleration and angular velocity (using known mass properties and controller gains), the system was able to anticipate dynamic requirements rather than react to them.

The outcome was exceptional. The attitude error decreased to the order of 10^-4 and reached 10^-6 during coasting phases. The torque and angular momentum stayed within limits, confirming adherence to the designed safety margin. Occasional spikes in torque demand were successfully absorbed by the 50% safety buffer, which proved essential to maintain performance in the presence of unmodeled dynamics and delays. Overall, this approach delivered near-perfect tracking with minimal error and smooth control effort.

5. Conclusion

The progression through the four scenarios illustrates the importance of integrating actuator constraints, trajectory planning, and predictive control in the design of spacecraft attitude controllers.

- The standard outer loop controller was unable to handle actuator saturation, resulting in instability and poor performance.
- Anti-windup logic improved stability but still suffered from transition-related overshoot.
- Trajectory generation helped avoid saturation entirely, improving overall system smoothness and maintaining actuator feasibility.
- The final scenario, combining feedforward control with a saturation-aware trajectory, achieved superior performance with negligible tracking error, even in the presence of dynamic uncertainties.

These results underscore that predictive, constraint-aware control architectures significantly outperform traditional reactive methods. By anticipating actuator limits and incorporating feedforward strategies, it is possible to design high-fidelity attitude control systems that remain robust, efficient, and accurate under realistic operational conditions.