# STATE-SPACE MODELING AND POLE PLACEMNT CONTROL OF A REACTION-WHEEL SATELLITE ATTITUDE SYSTEM

**Dante Benedetti, Nathan Heil**
Department of Electrical Engineering, Cal Poly, San Luis Obispo, CA 93407

**This project develops and evaluates a state-space control solution for a single-axis satellite attitude system actuated by a reaction wheel and an external thruster torque. Starting from conservation of angular momentum and rotational dynamics, a third-order linear time-invariant model is derived with states $\theta_s$, $\dot{\theta}_s$, and $\omega_w$. The open-loop plant exhibits integrator behavior and is unstable due to attitude drift, motivating feedback control. Controllability and observability are verified using the rank conditions on the controllability and observability matrices, enabling pole-placement design and full-state estimation. A multi-input state-feedback controller is designed via pole placement and augmented with a reference prefilter to achieve unity steady-state tracking for step attitude commands. A full-order Luenberger observer is implemented to estimate the unmeasured satellite rate from measured attitude and wheel speed. Simulation results demonstrate stable, well-damped tracking for a $10^{\circ}$step and for a sequence of step commands, with bounded actuator torques and angular rates. Additional parametric studies quantify how required torque and wheel speed scale with satellite inertia and how closed-loop pole location trades response time against actuator effort, providing practical guidance for controller tuning under realistic actuator constraints.**

## PROBLEM STATEMENT

Satellites must maintain precise attitude control to ensure proper orientation for communication, navigation, imaging, and scientific missions. Even small angular deviations can significantly degrade system performance. Reaction wheels are commonly used to generate control torque internally through conservation of angular momentum, enabling orientation control without continuous propellant expenditure [1]. In many satellite, thrusters are also available and can provide an additional external torque input for attitude maneuvers or supplemental control authority.

In open loop, a reaction-wheel satellite exhibits integrator-driven drift and effectively behaves as a double integrator from applied torque to attitude angle, meaning the system does not naturally regulate itself back to a desired pointing orientation. As a result, active feedback control is required to stabilize the attitude response and achieve reliable reference tracking.

The problem addressed in this project is the modeling and stabilization of a single-axis satellite using modern state-space control methods with two control inputs: reaction wheel torque and thruster torque. A physically derived state-space model is developed, and a pole-placement state-feedback controller is designed to achieve stable, well-damped closed-loop performance for step attitude commands. Additionally, a full-state observer is implemented to estimate internal states that may not be directly measured.

Solving this problem is important because it demonstrates how modern control theory can be applied to an aerospace system with non-self-correcting open-loop dynamics, while illustrating practical controller/observer synthesis for a multi-input satellite attitude system.

## DYNAMIC EQUATIONS

The reaction wheel–satellite attitude system is governed by **c**onservation of angular momentum, with an important distinction between internal actuator torques (reaction wheel motor torque) and external torques (thruster torque). Reaction wheels generate attitude control torque by exchanging angular

momentum with the satellite body, while thrusters directly apply an external torque to the combined satellite –wheel system.

Using Newton's rotational torque law for rigid body rotational dynamics,

$$\sum \tau = J\ddot{\theta}$$

where $\tau$ is torque, $J$ is the moment of inertia, and $\ddot{\theta}$ is angular acceleration, we write the equations of motion for both the satellite body and the reaction wheel about a single axis.

The satellite experiences a reaction torque from the wheel motor equal in magnitude and opposite in direction to the wheel torque (internal torque exchange). Additionally, a thruster torque $\tau_t$ acts as an external torque input:

$$J_s\ddot{\theta}_s = -\tau_w + \tau_t$$

where $J_s$ is the satellite inertia, $\theta_s$ is the satellite attitude angle, $\tau_w$ is the motor torque applied to the reaction wheel (which produces an equal and opposite torque on the satellite body), and $\tau_t$ is the external thruster torque about the same axis.

The wheel is directly accelerated by the applied motor torque:

$$J_w\dot{\omega}_w = \tau_w$$

where $J_w$ is the wheel inertia and $\omega_w$ is the wheel angular velocity.

To verify that the above sign conventions and coupling are consistent, consider the total angular momentum of the satellite –wheel system about the controlled axis:

$$H = J_s\dot{\theta}_s + J_w\omega_w$$

Taking the time derivative,

$$\frac{dH}{dt} = J_s\ddot{\theta}_s + J_w\dot{\omega}_w$$

Because $\tau_w$is an internal actuator torque, it does not change the total angular momentum of the combined system; only external torques contribute to $\frac{dH}{dt}$. Therefore, the momentum rate must equal the external thruster torque (and any modeled disturbance torque if included)[2]:

$$\frac{dH}{dt} = \tau_t$$

Substituting the wheel dynamics $J_w\dot{\omega}_w = \tau_w$into the momentum-rate equation yields:

$$J_s\ddot{\theta}_s + \tau_w = \tau_t \quad \Rightarrow \quad J_s\ddot{\theta}_s = -\tau_w + \tau_t$$

which matches the satellite body equation derived directly from Newton's torque law.

Using this relationship, we obtain the final coupled system of equations used for modeling and control design:

$$J_s\ddot{\theta}_s = -\tau_w + \tau_t$$

$$J_w\dot{\omega}_w = \tau_w$$

These equations explicitly capture that the reaction wheel torque $\tau_w$ redistributes angular momentum internally between the wheel and satellite body, while the thruster torque $\tau_t$ provides an external control input capable of changing the total angular momentum of the system.

**STATE SPACE MODEL**

Analyzing the coupled equations of motion, the satellite attitude dynamics contain one second-order differential equation in $\theta_s$ and one first-order differential equation in $\omega_w$. This results in a third-order system overall, implying that a minimum of three state variables are required to represent the dynamics in first-order form.

To form a state-space model suitable for modern control design, the state vector is selected to represent the satellite attitude, satellite angular rate, and reaction wheel speed:

$x_1 = \theta_s$: satellite angular position
$x_2 = \dot{\theta}_s$: satellite angular velocity
$x_3 = \omega_w$: reaction wheel angular velocity

The control input vector includes both the internal wheel torque and the external thruster torque:

$$u = \begin{bmatrix} \tau_w \\ \tau_t \end{bmatrix}$$

and the measured output vector is chosen as satellite attitude and wheel speed:

$$y = \begin{bmatrix} \theta_s \\ \omega_w \end{bmatrix}$$

Using the governing dynamics defined, the first-order state equations follow directly from the state definitions:

- From $x_1 = \theta_s$, we have $\dot{x}_1 = x_2$.
- From the satellite equation, $\dot{x}_2 = \ddot{\theta}_s = -\frac{1}{J_s}\tau_w + \frac{1}{J_s}\tau_t$.
- From the wheel equation, $\dot{x}_3 = \dot{\omega}_w = \frac{1}{J_w}\tau_w$.

These relations yield the standard linear time-invariant (LTI) state-space form commonly used in modern control analysis

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

with

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ -\frac{1}{J_s} & \frac{1}{J_s} \\ \frac{1}{J_w} & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

This representation highlights several important physical and control-relevant properties of the system. First, the matrix $A$ contains a single integrator chain from $x_2$ to $x_1$, while the remaining open-loop dynamics contain no natural damping terms. As a result, the plant is unstable, consistent with the fact that a satellite will not inherently return to a desired attitude after a disturbance without feedback control.

Second, the input matrix $B$ clearly separates the roles of the actuators. The reaction wheel torque $\tau_w$ appears with opposite signs in the satellite and wheel dynamics: it accelerates the wheel directly ($+\frac{1}{J_w}\tau_w$) while producing an equal-and-opposite effect on the satellite body acceleration ($-\frac{1}{J_s}\tau_w$). In contrast, the thruster torque $\tau_t$ acts only on the satellite body, contributing directly to $\dot{x}_2$through $+\frac{1}{J_s}\tau_t$, consistent with its role as an external torque input.

Finally, the output matrix $C$ reflects a practical sensing assumption: satellite attitude $\theta_s$is typically available from attitude sensors or estimation filters, and reaction wheel speed $\omega_w$is directly measurable via motor/encoder telemetry. This output selection supports subsequent observer design when full-state feedback is desired but not all states are directly measured.

Overall, this state-space model provides a physically interpretable, minimal-order LTI description of the satellite –reaction wheel–thruster system and forms the foundation for controllability analysis, pole-placement state-feedback design, and full-state observer implementation in the following sections.

## SYSTEM TRANSFER FUNCTION

Using the state space model, the plant can be interpreted in the Laplace domain as a multi-input, multi-output (MIMO) system relating actuator torques to the measurable quantities of interest. Taking Laplace transforms of the dynamic equations with zero initial conditions,

$$J_s s^2 \Theta(s) = -T_w(s) + T_t(s)$$

$$J_w s\,\Omega_w(s) = T_w(s)$$

Solving for the outputs in terms of the inputs yields the transfer relations

$$\Theta(s) = -\frac{1}{J_s s^2} T_w(s) + \frac{1}{J_s s^2} T_t(s)$$

$$\Omega_w(s) = \frac{1}{J_w s} T_w(s) + 0 \cdot T_t(s)$$

Therefore, the MIMO transfer function matrix $G(s)$(from $u$ to $y$) is

$$G(s) = \frac{Y(s)}{U(s)} = \begin{bmatrix} \frac{\Theta(s)}{T_w(s)} & \frac{\Theta(s)}{T_t(s)} \\ \frac{\Omega_w(s)}{T_w(s)} & \frac{\Omega_w(s)}{T_t(s)} \end{bmatrix} = \begin{bmatrix} -\frac{1}{J_s s^2} & \frac{1}{J_s s^2} \\ \frac{1}{J_w s} & 0 \end{bmatrix}.$$

This form makes the physical behavior of the system explicit. The satellite attitude output $\Theta(s)$ has a $1/s^2$ dependence on torque, meaning the satellite angle behaves as a double integrator with respect to net applied body torque; as a result, constant torque produces linearly increasing angular rate and quadratically growing angle. In contrast, the reaction wheel speed output $\Omega_w(s)$ has a $1/s$ dependence on the wheel torque, meaning wheel speed behaves as a single integrator driven by the motor torque. The transfer element $\Omega_w(s)/T_t(s)$ is equal to zero because the thruster torque $\tau_t$ acts on the satellite body as an external torque but does not directly apply torque to the wheel; in the assumed model there is no mechanical coupling term from $\tau_t$ into the wheel's rotational equation, so thrusters cannot directly change wheel speed except indirectly through control decisions that command $\tau_w$.

**STABILITY**

The open-loop stability of the reaction-wheel satellite model can be assessed directly from the state matrix

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

which is naturally in Jordan form and therefore has eigenvalues given by its diagonal entries:

$$\lambda_1 = \lambda_2 = \lambda_3 = 0.$$

Although the eigenvalues lie at the origin, the stability outcome depends on the Jordan structure. The nonzero term $A_{12} = 1$ produces a $2 \times 2$ Jordan block at $\lambda = 0$, which implies the presence of a double-integrator mode. This can be seen directly from the open-loop state equations $\dot{x} = Ax$, which give $\dot{x}_1 = x_2$, $\dot{x}_2 = 0$, and $\dot{x}_3 = 0$. As a result, $x_2(t) = x_2(0)$ and $x_3(t) = x_3(0)$, while the attitude evolves as:

$$x_1(t) = x_1(0) + x_2(0)t$$

Therefore, for any nonzero initial angular rate $x_2(0) \neq 0$, the attitude angle grows without bound (linearly in time), meaning the satellite will drift indefinitely and cannot naturally return to an equilibrium orientation. Because the system includes a repeated eigenvalue on the imaginary axis with a Jordan block of size greater than one, it is open-loop unstable, and disturbances or small rate biases will accumulate into large pointing errors. This instability motivates the use of active feedback control; in the following sections a state-feedback pole-placement controller is designed to shift the closed-loop poles into the left-half plane to eliminate drift, ensure exponential convergence to the commanded attitude, and enable theoretical disturbance rejection.

**CONTROLABILITY**

To determine whether the reaction-wheel satellite model can be driven to an arbitrary state using the available control inputs, the system is tested for controllability. For the continuous-time LTI system defined in our state space model the controllability matrix is defined as

$$P = [B \quad AB \quad A^2B \quad \cdots \quad A^{n-1}B],$$

where $n$ is the number of states. The state space model has 3 state variables ($n = 3$), so the required controllability matrix is

$$P = [B \quad AB \quad A^2B].$$

Using the A and B matrices we compute

$$AB = \begin{bmatrix} -\frac{1}{J_s} & \frac{1}{J_s} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad A^2B = 0.$$

Defining $a = \frac{1}{J_s}$ and $b = \frac{1}{J_w}$ for compactness, the controllability matrix becomes

$$P = \begin{bmatrix} 0 & 0 & -a & a & 0 & 0 \\ -a & a & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The system is fully controllable if and only if $\text{rank}(P) = n$ . [3] Here,

$$\text{rank}(P) = 3,$$

which equals the number of state variables in the model ($n = 3$). This result implies that the two inputs $\tau_w$ and $\tau_t$ provide sufficient authority, through the system dynamics, to independently influence all three states $\theta_s$, $\dot{\theta}_s$, and $\omega_w$. Consequently, the system is fully controllable and it is theoretically possible to place the closed-loop poles arbitrarily using state-feedback control (subject to practical actuator limits).

**OBSERVABILITY**

For implementation of state-feedback control, it is often unrealistic to assume that all internal states are directly measured. Therefore, the system is tested for observability, which determines whether the full state vector $x(t)$ can be uniquely reconstructed from the measured outputs $y(t)$ over time. The LTI system is observable if the observability matrix

$$Q = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

has full rank $n$, where $n$ is the number of states. Since this satellite model is third order ($n = 3$), the required observability matrix is

$$Q = \begin{bmatrix} C \\ CA \\ CA^2 \end{bmatrix}.$$

In this project the measured outputs are chosen as satellite attitude and reaction wheel speed, with the state matrix A. Computing the required terms,

$$CA = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$CA^2 = (CA)A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Thus, the observability matrix becomes

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The three nonzero rows [1 0 0], [0 0 1], and [0 1 0]are linearly independent, so

$$\text{rank}(Q) = 3$$

which equals the number of states $n = 3$. Therefore, the system is fully observable with the selected measurements $y = [\theta_s \quad \omega_w]^T$. Physically, measuring $\theta_s$ and $\omega_w$ is sufficient because the satellite rate $\dot{\theta}_s$ influences the measured attitude through the kinematic relation $\dot{\theta}_s = \dot{x}_1$, allowing the angular rate state to be inferred from the time evolution of $\theta_s$. This full observability result justifies the use of a full-order state observer to estimate the unmeasured states and implement observer-based state-feedback control in the subsequent controller design.

**STATE FEEDBACK CONTROLLER**

Because the open-loop satellite –reaction wheel dynamics contain integrator behavior and exhibit drift in attitude, a stabilizing feedback controller is required to achieve bounded tracking and disturbance rejection. With the system verified to be controllable, full-state feedback was selected to place the closed-loop poles in the left-half plane and enforce stable, well-damped motion. The control law is implemented in the standard reference-tracking form

$$u = -Kx + Fr$$
$$r = \theta_{\text{ref}}$$

where $u = [\tau_w \quad \tau_t]^T$ is the actuator torque vector, $K \in \mathbb{R}^{2\times 3}$ is the state-feedback gain matrix, and $F \in \mathbb{R}^{2\times 1}$ is a reference prefilter chosen to ensure unity steady-state gain from the commanded attitude $r$ to the output $\theta_s$.

The feedback gain $K$ was obtained using pole-placement (MATLAB place(A,B,p)), where the desired closed-loop poles were chosen as

$$p = \{-0.4, -0.6, -1.5\}.$$

These poles were selected to be all real to produce an overdamped response with minimal overshoot, which is desirable for high-precision pointing applications (e.g., antennas and imaging payloads). Importantly, the pole locations were not selected to directly satisfy a prescribed rise time or settling time. Instead, they were chosen to enforce stable dynamics while maintaining realistic actuator torque demands consistent with physically reasonable satellite and wheel inertias. In practice, moving poles further left increases response speed but can also require larger control torques; therefore, the selected pole set represents a deliberate trade between performance and actuator realism.

For the representative inertias used in simulation ($J_s = 20$ kg\m$^2$, $J_w = 0.1$ kg\m$^2$), the resulting state-feedback gain was computed as

$$K = \begin{bmatrix} 0 & 0 & 0.04 \\ 18 & 42 & 0.04 \end{bmatrix},$$

yielding the closed-loop state matrix $A_{cl} = A - BK$ with eigenvalues equal to the chosen pole set. The controller therefore stabilizes the integrator chain and eliminates the open-loop drift behavior.

To enable accurate steady-state tracking of a constant attitude command, a reference gain $F$ was computed such that the DC gain from $r$ to $\theta_s$ is unity. Using the selection matrix $C_\theta = [1\ 0\ 0]$, $F$ was chosen to satisfy

$$-C_\theta (A - BK)^{-1} BF = 1,$$

and MATLAB's pseudoinverse was used to obtain a minimum-norm solution. For the same operating point, this produced

$$F = \begin{bmatrix} -9 \\ 9 \end{bmatrix}.$$

This prefilter distributes the required steady-state torque command across the wheel and thruster channels in a way that achieves the correct equilibrium output without requiring integral action.

**STATE FEEBACK OBSERVER**

Although the state-feedback controller $u = -Kx + Fr$assumes full state availability, in practice not all states are directly measurable. For the reaction-wheel satellite system, the measured outputs were selected as satellite attitude and reaction wheel speed, meaning $\theta_s$ and $\omega_w$ are available from attitude sensors and wheel telemetry, while the satellite angular rate $\dot{\theta}_s$is treated as an unmeasured state. Because matrices A and C were shown to be observable, a full-order Luenberger observer can be constructed to estimate the complete state vector $\hat{x}$ from $y$ and the applied control input $u$. The observer is defined as

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x}),$$

where $L \in \mathbb{R}^{3\times 2}$is the observer gain matrix and $y - C\hat{x}$ is the innovation term (measurement residual). Defining the estimation error $e = x - \hat{x}$, the error dynamics follow directly [4]:

$$\dot{e} = (A - LC)e$$

Therefore, the observer is asymptotically convergent if the eigenvalues of $(A-LC)$are placed in the left-half plane. In this project, the observer poles were chosen as

$$p_{\text{obsv}} = \{-1,\ -1.5,\ -3\},$$

which are intentionally faster than the controller poles to ensure the state estimate converges quickly relative to the closed-loop plant response. These poles were selected using the MATLAB command

$$L = \text{place}(A^T, C^T, p_{\text{obsv}})^T,$$

consistent with the duality between controllability and observability.

Using the same parameter set and MATLAB implementation shown in the provided code, the resulting observer gain was computed as

$$L = \begin{bmatrix} 4.5 & 0 \\ 4.5 & 0 \\ 0 & 1 \end{bmatrix},$$

and the resulting eigenvalues of $(A-LC)$ match the desired observer pole locations. The structure of $L$is physically intuitive: the first output $\theta_s$ corrects the attitude and rate estimates (first two rows), while the second output $\omega_w$ directly corrects the wheel-speed estimate (third row).

In simulation, the observer performance is evaluated by intentionally introducing an initial-condition mismatch between the true plant state and the estimator state (i.e., $x(0) \neq \hat{x}(0)$). This provides a direct test of the convergence property predicted by the error dynamics $\dot{e} = (A - LC)e$. With the chosen observer poles in the left-half plane, the estimation error $e(t)$decays exponentially, so $\hat{x}(t)$rapidly approaches $x(t)$even when the observer is initialized with an incorrect attitude, rate, or wheel speed. As a result, the estimated outputs (e.g., $\hat{\theta}_s$, $\hat{\dot{\theta}}_s$, and $\hat{\omega}_w$) quickly align with the true trajectories, and the controller—implemented using $\hat{x}$rather than $x$—maintains stable tracking while the observer transient dies out.

## SIMULATION

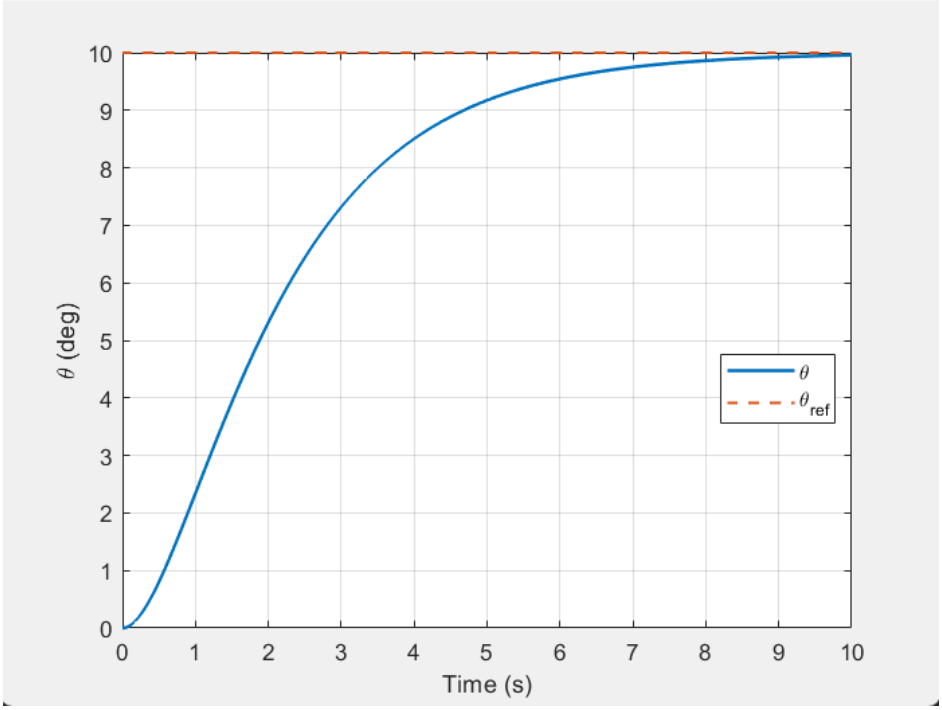


*Figure 1. Closed-loop satellite attitude step response for a* $10°$*command using observer-based state feedback.* $J_s = 20\ kgm^2$, $J_w = 0.1\ kgm^2$

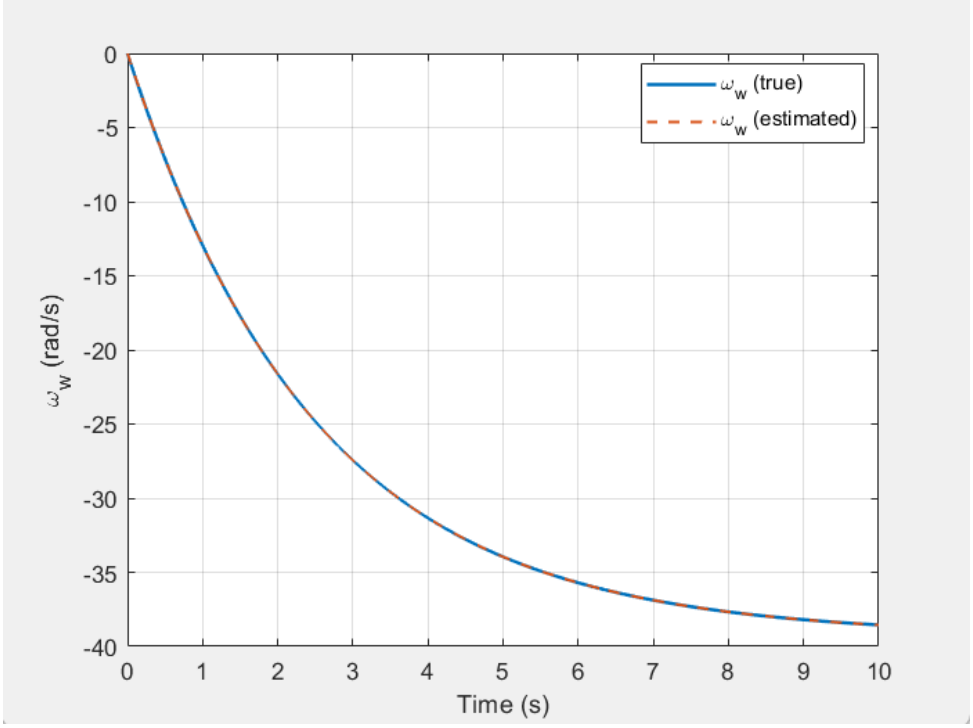


*Figure 2. Reaction wheel angular velocity for a* $10°$*step command.* $J_s = 20\ kgm^2$, $J_w = 0.1\ kgm^2$

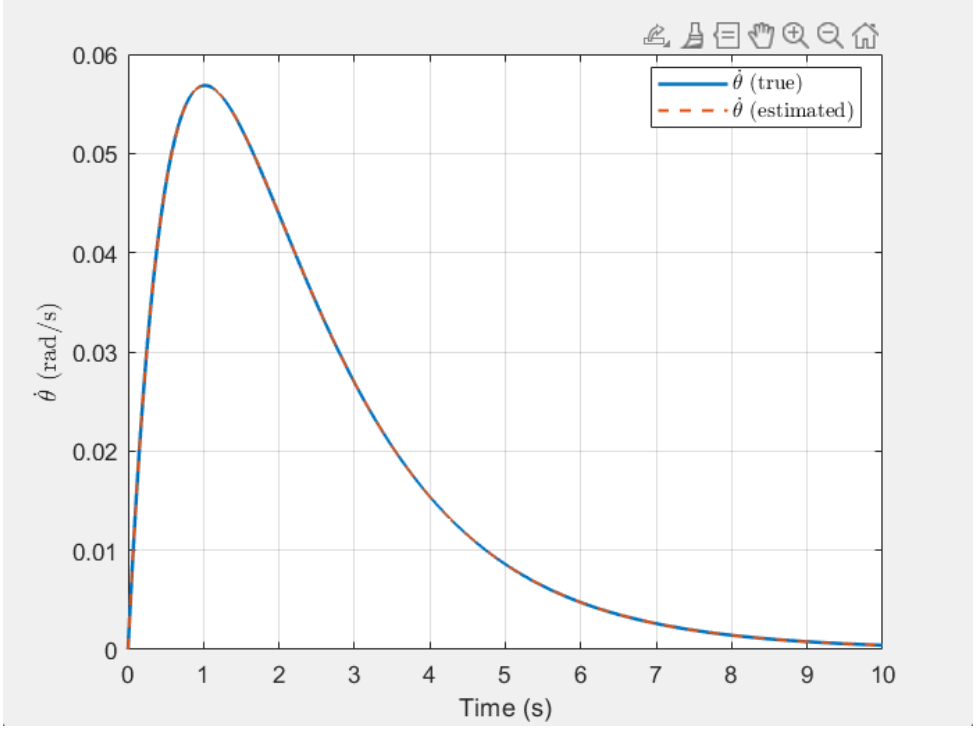


*Figure 3. Satellite angular velocity for a* $10°$ *step command.*
$J_s = 20\ kgm^2$, $J_w = 0.1\ kgm^2$

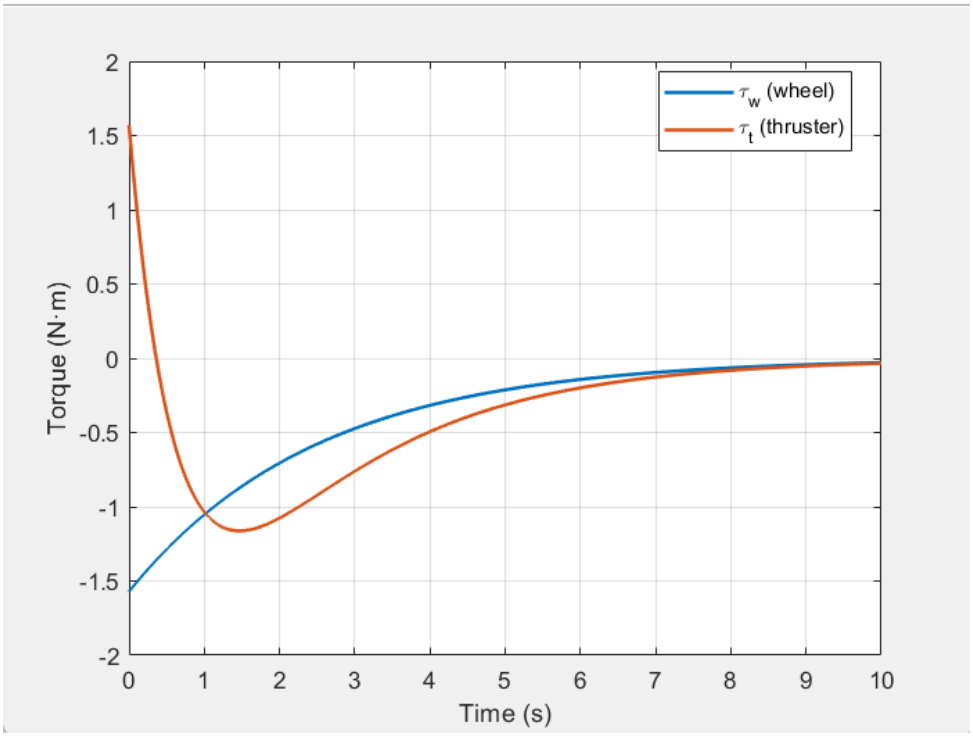


*Figure 4. Torque inputs for a* $10°$ *step command.*
$J_s = 20\ kgm^2$, $J_w = 0.1\ kgm^2$

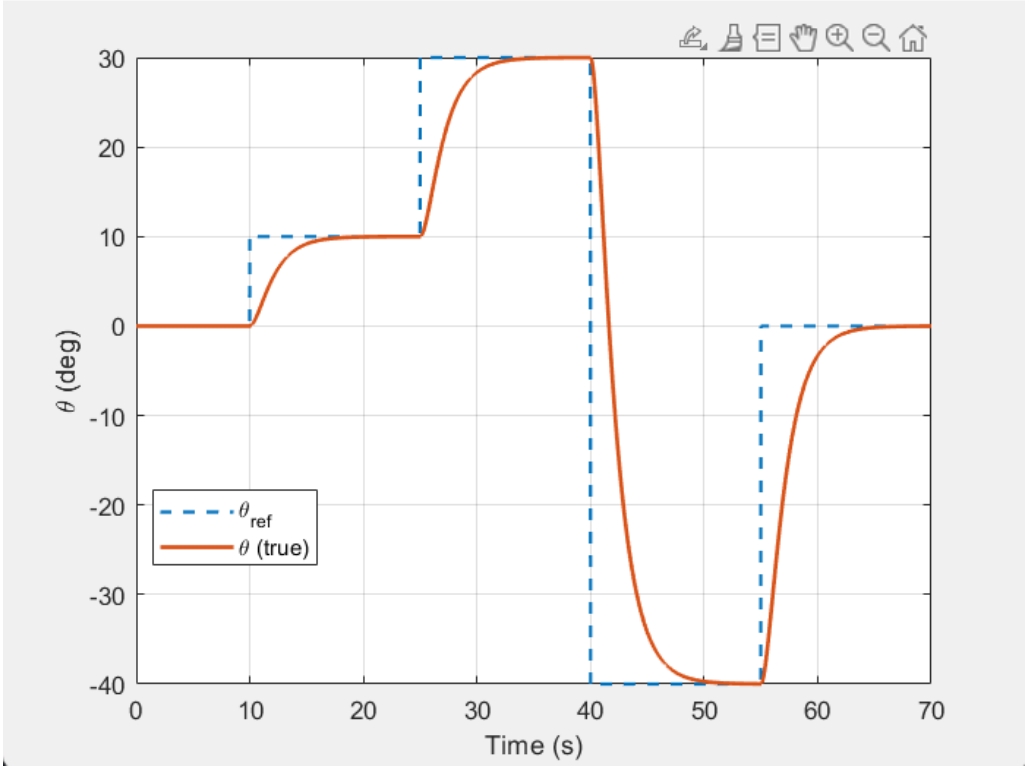


*Figure 5. Attitude of satellite with series of $\theta_{ref}$ commands as step inputs: 0, +10, +30, -40, 0.*

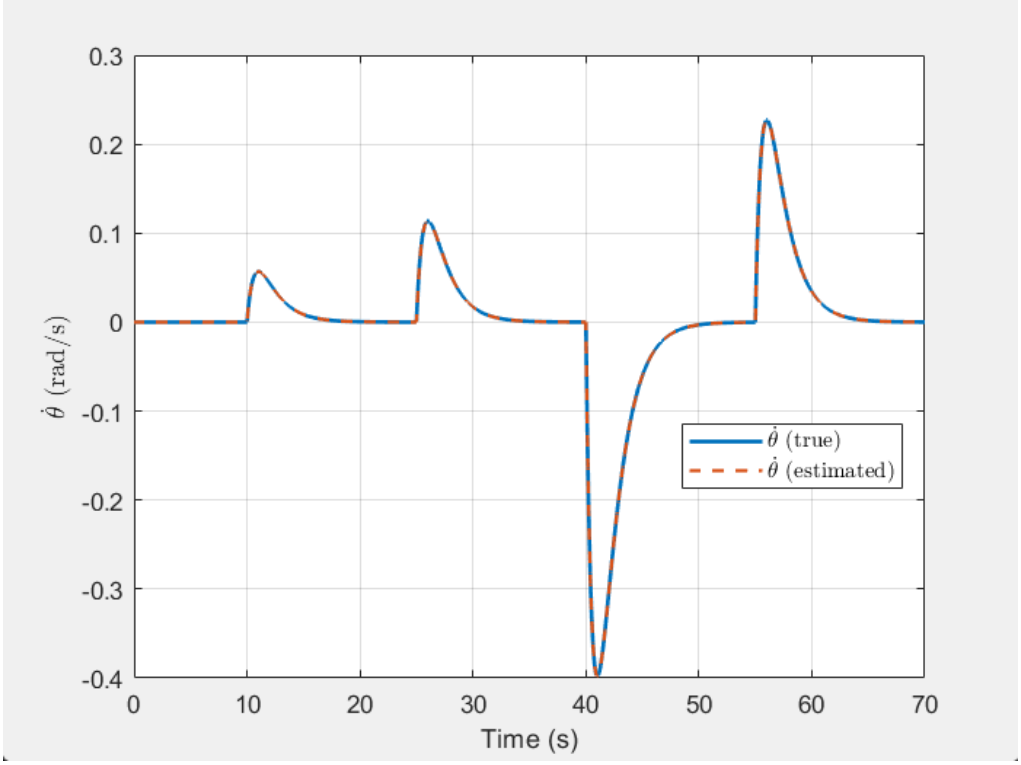


*Figure 6. Angular velocity of satellite for series of inputs.*

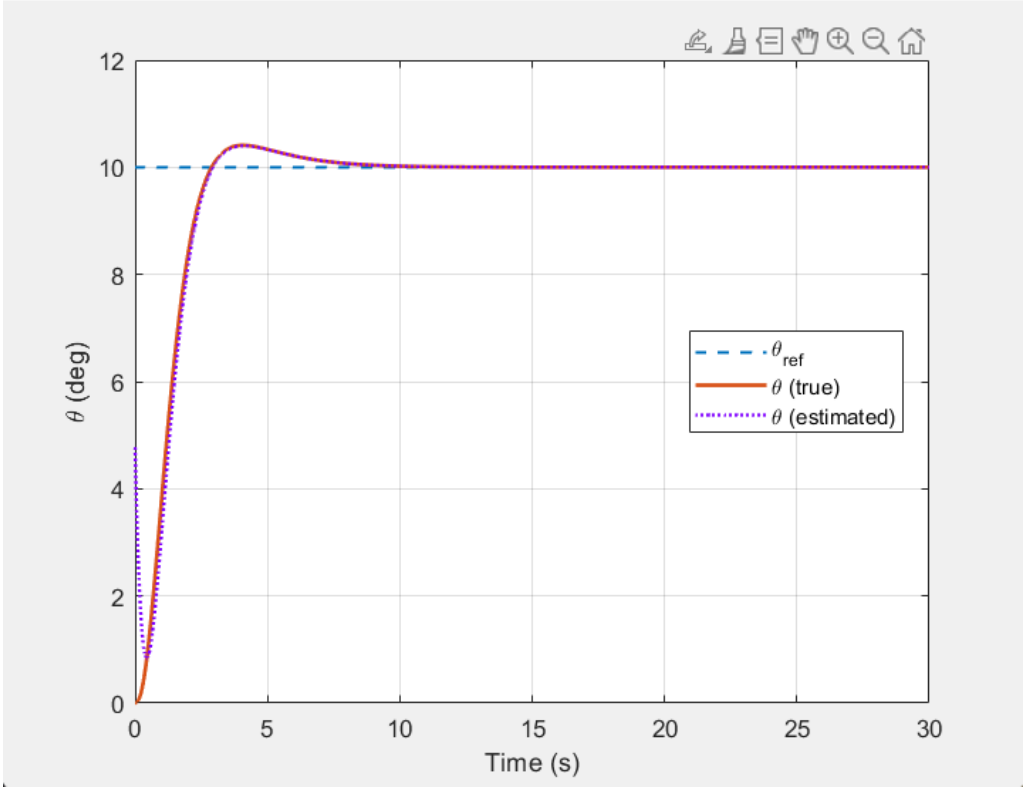


*Figure 7. Observer convergence for a* $10°$ *step with mismatched attitude initial conditions* ($\theta$, $\hat{\theta}$).

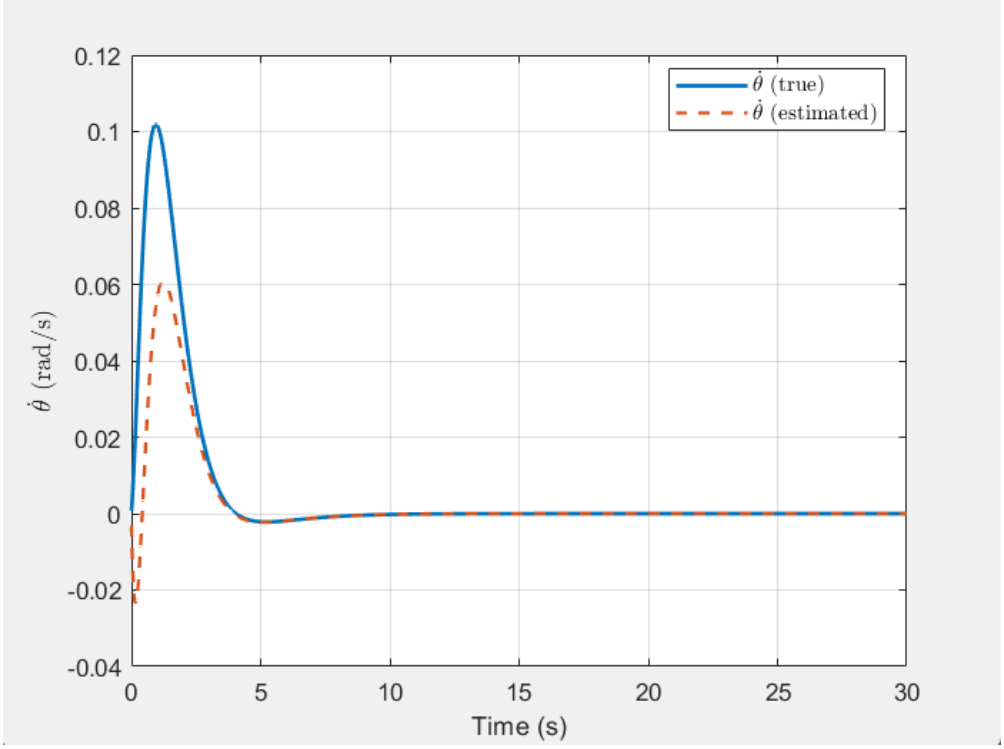


*Figure 8. True vs. estimated satellite angular rate* $\dot{\theta}$ *during observer convergence with mismatched initial conditions.*

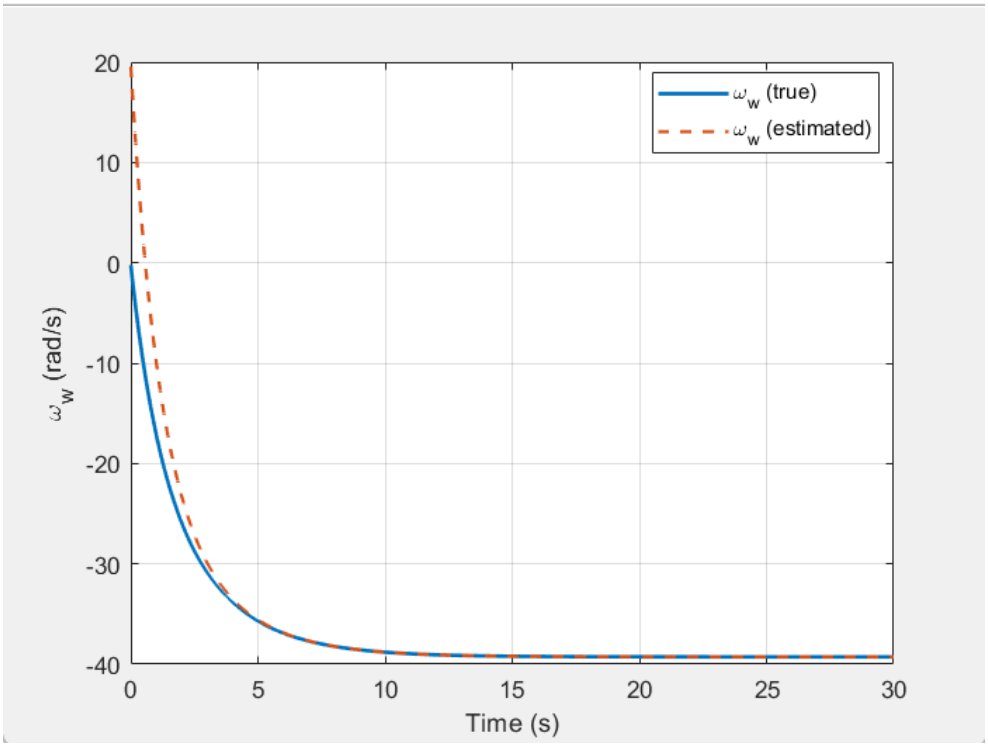


*Figure 9. True vs. estimated reaction wheel angular rate $\omega_w$ during observer convergence with mismatched initial conditions.*

*Table 1. Results for a $10°$ step with fixed controller poles $p = \{-0.4, -0.6, -1.5\}$ and constant wheel inertia $J_w = 0.1\ kg/m^2$: swept satellite inertia $J_s$ versus peak reaction wheel torque, peak thruster toque, and peak reaction wheel angular velocity.*

| $J_s$ (kg·m^2) | $\lvert\tau_{w,\max}\rvert$ (Nm) | $\lvert\tau_{t,\max}\rvert$ (Nm) | $\lvert\omega_{w,\max}\rvert$ (rad/s) |
|---|---|---|---|
| 20 | 1.5708 | 1.5708 | 39.257 |
| 40 | 3.1416 | 3.1416 | 78.513 |
| 60 | 4.7124 | 4.7124 | 117.77 |
| 80 | 6.2832 | 6.2832 | 157.03 |
| 100 | 7.8540 | 7.8540 | 196.28 |
| 120 | 9.4248 | 9.4248 | 235.54 |
| 140 | 10.996 | 10.996 | 274.80 |
| 160 | 12.566 | 12.566 | 314.05 |
| 180 | 14.137 | 14.137 | 353.31 |
| 200 | 15.708 | 15.708 | 392.57 |

*Table 2. Pole-scaling for a* $10°$*attitude step with fixed inertias* $J_s = 20\ kg \cdot m^2$*and* $J_w = 0.1kg \cdot m^2$*. With resulting timing characterisitics.*

| Controller Poles $(p_1, p_2, p_3)$ | Rise time (s) | Settling time (s) | $\lvert\tau_{w,\max}\rvert$ (Nm) | $\lvert\tau_{t,\max}\rvert$ (Nm) |
|---|---|---|---|---|
| (-0.029,-0.043,-0.109) | 56.700 | 101.700 | 0.008 | 0.008 |
| (-0.045,-0.068,-0.169) | 36.500 | 65.450 | 0.020 | 0.020 |
| (-0.070,-0.105,-0.262) | 23.450 | 42.100 | 0.048 | 0.048 |
| (-0.097,-0.146,-0.365) | 16.900 | 30.250 | 0.093 | 0.093 |
| (-0.136,-0.203,-0.508) | 12.150 | 21.700 | 0.180 | 0.180 |
| (-0.211,-0.316,-0.790) | 7.800 | 13.950 | 0.436 | 0.436 |
| (-0.293,-0.440,-1.100) | 5.600 | 10.050 | 0.844 | 0.844 |
| (-0.408,-0.612,-1.531) | 4.000 | 7.200 | 1.635 | 1.635 |

**ANALYSIS**

Figures 1–4 evaluate the nominal closed-loop response to a $10°$step command using the observer-based state-feedback controller. Figure 1 shows that the satellite attitude $\theta$tracks the reference smoothly with a well-damped, non-oscillatory transient and negligible overshoot, which is desirable for missions requiring stable pointing such as high-resolution imaging and narrow-beam RF communications. Figures 2 and 3 further support this behavior by showing that both the reaction wheel speed and satellite angular rate remain bounded during the maneuver and decay back toward steady values as the attitude converges. In particular, the satellite angular rate returns to approximately zero once the commanded angle is reached (Figure 3), which matches the expected physical behavior: at the desired orientation, the satellite should no longer be rotating. Figure 4 shows the commanded wheel and thruster torques required to achieve this response; both inputs are transient and decay toward zero as the system settles, indicating that the controller does not demand sustained actuator effort once steady pointing is achieved.

Figures 5 and 6 extend the evaluation to a sequence of step commands $(0, +10, +30, -40, 0)$ applied in series. Figure 5 demonstrates that the controller can successfully regulate the satellite attitude through multiple commanded setpoint changes, provided sufficient time is allowed for each maneuver to complete before the next command. Consistent with the single-step case, Figure 6 shows that the satellite angular rate rises during each maneuver and then returns to near zero after the attitude settles at each intermediate reference. This again reflects the expected requirement for precision pointing: the satellite should come to rest at each commanded attitude rather than maintaining residual rotation.

Figures 7–9 demonstrate the effectiveness of the full-order observer under mismatched initial conditions between the true plant state and the estimated state. Figure 7 shows that even when $\hat{\theta}(0)$differs from $\theta(0)$, the estimated attitude rapidly converges to the true attitude trajectory while maintaining accurate tracking of the $10°$reference. Figures 8 and 9 show the same convergence behavior for the unmeasured satellite angular rate $\dot{\theta}$and the reaction wheel speed $\omega_w$, respectively: the estimation errors decay quickly, confirming that the chosen observer poles provide fast correction without destabilizing the closed-loop response. Together, these results validate the observer-based implementation as practical for cases where not all states are directly measurable.

Table 1 quantifies how actuator demands scale with satellite inertia when the controller poles are held fixed at $p = \{-0.4, -0.6, -1.5\}$and $J_w$is held constant. As $J_s$increases, both the peak commanded wheel torque and thruster torque increase substantially, and the peak reaction wheel speed also increases. This trend is physically intuitive: a larger satellite inertia requires more torque (and correspondingly more wheel momentum exchange) to achieve the same attitude maneuver with the same closed-loop dynamics. The results highlight an important practical implication—controllers tuned for a small satellite will generally require significantly greater actuator capability when applied to a larger satellite, even for the same commanded angle.

Table 2 shows the complementary trade study: with inertias fixed ($J_s = 20\ \mathrm{kg \cdot m^2}, J_w = 0.1\ \mathrm{kg \cdot m^2}$), varying the controller pole locations directly changes both timing and torque requirements. Pole sets closer to the origin (smaller-magnitude poles) produce slower rise and settling times but require much less control torque, while poles placed farther into the left-half plane yield faster response at the cost of higher peak wheel and thruster torques. This confirms the expected control-design tradeoff: faster closed-loop dynamics demand increased actuator effort, whereas slower poles reduce actuator demand but increase the time required to converge to the commanded attitude.

## CONCLUSION

This project derived a third-order state-space model for a single-axis reaction-wheel satellite with an external thruster torque and showed the open-loop system exhibits drift, requiring feedback control. A pole-placement state-feedback controller with a reference prefilter and a full-order observer was implemented after confirming full controllability and observability. Simulations demonstrated stable, well-damped 10°step tracking suitable for precision pointing (imaging and RF), successful regulation of multiple step commands when adequate settling time is provided, and fast observer convergence despite mismatched initial conditions. Parametric studies showed that increasing satellite inertia significantly raises required peak torques and wheel speed for a fixed pole set, while varying pole locations at fixed inertia reveals the expected tradeoff: slower poles reduce torque demand but increase rise/settling time.

Future work should incorporate actuator limits and saturation (wheel torque/speed, thruster duty cycle), realistic disturbances and sensor noise/bias, and robustness to inertia uncertainty. Additional improvements include adding integral/disturbance rejection for constant torques, implementing momentum dumping to prevent wheel saturation, and extending the approach to a full 3-axis coupled satellite model.

## REFERENCES

[1] Z. Ismail and R. Varatharajoo, “A study of reaction wheel configurations for a 3-axis satellite attitude control,” Advances in Space Research, vol. 45, no. 6, pp. 750–759, 2010.

[2] B. Wie, *Space Vehicle Dynamics and Control*, 2nd ed. Reston, VA, USA: AIAA, 2008.

[3] C.-T. Chen, *Linear System Theory and Design*, 4th ed. New York, NY, USA: Oxford University Press, 2012.

[4] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[5] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 7th ed. Boston,

## APPENDIX

---

### State Space Model MATLAB Code

---

```
%% sat_attitude_angle_control.m
% Satellite attitude (angle) control using state feedback + full-order observer
% Model: from EE 513 Final Project Presentation (u = [tau_w; tau_t], y = [theta; omega_w])
%
% States:
%   x1 = theta      (rad)   spacecraft angle
%   x2 = theta_dot  (rad/s) spacecraft angular rate
%   x3 = omega_w    (rad/s) reaction wheel speed
%
% Inputs:
%   u1 = tau_w (N*m) wheel torque
%   u2 = tau_t (N*m) thruster torque
%
% Outputs:
%   y1 = theta
%   y2 = omega_w
%
% Control law (observer-based):
%   u = -K*x_hat + F*r,   r = theta_ref

clear; clc; close all;
format short g;

Js = 40;     % spacecraft inertia [kg*m^2]
Jw = 0.10;   % reaction wheel inertia [kg*m^2]
%% ---------------------- State-space model -----------------------------
A = [0 1 0;
     0 0 0;
     0 0 0];

B = [ 0      0;
     -1/Js  1/Js;
      1/Jw   0  ];

C = [1 0 0;
     0 0 1];

D = zeros(2,2);

%% ---------------------- Parameters (edit these) ----------------------

% Desired closed-loop poles for (A - B*K)
P_ctrb = [-0.4 -0.6 -1.5];      % controller poles (distinct)

% Desired observer poles for (A - L*C)
P_obsv = [-1 -1.5 -3];      % observer poles (faster than controller)

% Simulation setup
t  = 0:0.01:10;            % time vector [s]
theta_ref_deg = 10;        % step command [deg]
r  = (theta_ref_deg*pi/180) * ones(size(t));  % reference input [rad]
```

```
% Initial conditions (true plant and estimator can be different)
x0    = [0; 0; 0];         % [theta; theta_dot; omega_w]
y0 = C*x0;             % y0 = [theta0; omega_w0]
xhat0 = [y0(1); 0; y0(2)];       % estimator IC (mismatch to show convergence)
x0_total = [x0; xhat0];

%% ---------------------- Display A,B,C,D ------------------------------
disp('=================== OPEN-LOOP MATRICES ===================');
disp('A ='); disp(A);
disp('B ='); disp(B);
disp('C ='); disp(C);
disp('D ='); disp(D);

disp('Open-loop eigenvalues eig(A) ='); disp(eig(A).');

%% ---------------------- Controllability / observability --------------
Co = ctrb(A,B);
Ob = obsv(A,C);
disp('rank(ctrb(A,B)) ='); disp(rank(Co));
disp('rank(obsv(A,C)) ='); disp(rank(Ob));

%% ---------------------- State feedback (pole placement) ----------------
% Multi-input pole placement: u = -K*x
K = place(A, B, P_ctrb);     % K is (2x3)
Acl = A - B*K;

disp('=================== CONTROLLER (STATE FEEDBACK) ===================');
disp('Desired controller poles ='); disp(P_ctrb);
disp('K ='); disp(K);
disp('eig(A - B*K) ='); disp(eig(Acl).');

%% ---------------------- Reference gain F (unity DC gain for theta) ----
% We want steady-state theta -> r for a constant reference r (step),
% using u = -K*x_hat + F*r, and tracking theta = x1.
C_theta = [1 0 0];     % select theta output only

% M = -C_theta * inv(Acl) * B is 1x2
M = -C_theta * (Acl \ B);

% Choose the minimum-norm solution F (2x1) such that M*F = 1
F = pinv(M) * 1;

disp('=================== REFERENCE GAIN ===================');
disp('Tracking output: theta = [1 0 0] * x');
disp('M = -C_theta*(Acl\\B) ='); disp(M);
disp('F (2x1) chosen so that M*F = 1:');
disp('F ='); disp(F);
disp('Check (should be ~1): M*F ='); disp(M*F);

%% ---------------------- Full-order observer gain L ---------------------
% Observer uses measured y = C*x (theta and omega_w are measured)
L = place(A', C', P_obsv)';   % L is (3x2)

% Optional: exact gain shown in the presentation for poles [-8 -9 -10]
% L = [17 0; 72 0; 0 10];

Aobs = A - L*C;

disp('=================== OBSERVER ===================');
disp('Desired observer poles ='); disp(P_obsv);
disp('L ='); disp(L);
disp('eig(A - L*C) ='); disp(eig(Aobs).');

%% ---------------------- Augmented (plant + observer) system ------------
% Total state: x_total = [x; x_hat]
% xdot      = A*x      + B*(-K*x_hat + F*r)
% xhat_dot  = A*x_hat  + B*(-K*x_hat + F*r) + L*(y - C*x_hat)
%           = L*C*x + (A - B*K - L*C)*x_hat + B*F*r
```

```
A_total = [A,        -B*K;
           L*C, (A - B*K - L*C)];

B_total = [B*F;
           B*F];

C_total = [C, zeros(size(C))];   % output y based on true plant states only
D_total = zeros(size(C,1), 1);   % r -> y direct term (D is zero)

disp('==================== AUGMENTED (TOTAL) SYSTEM MATRICES ====================');
disp('A_total ='); disp(A_total);
disp('B_total ='); disp(B_total);
disp('C_total ='); disp(C_total);
disp('D_total ='); disp(D_total);

disp('eig(A_total) ='); disp(eig(A_total).');

sys_total = ss(A_total, B_total, C_total, D_total);
```

---

## Figures 1-9 Simulation Code

---

```
%% ----------------------- Simulation (lsim) -----------------------------
[y, t_out, x_total] = lsim(sys_total, r, t, x0_total);

x    = x_total(:,1:3);
xhat = x_total(:,4:6);

% Reconstruct control inputs u(t) = -K*x_hat + F*r
u = zeros(length(t_out), 2);
for k = 1:length(t_out)
    u(k,:) = (-K*xhat(k,:).' + F*r(k)).';
end
tau_w = u(:,1);
tau_t = u(:,2);

%% ----------------------- Plots ----------------------------------------
% Output y = [theta; omega_w]
theta = y(:,1);
omega_w = y(:,2);

%% ---- Figure 1: Reference vs True Sat Theta ----
figure('Name','Angle Tracking');
plot(t_out, theta*180/pi, 'LineWidth', 1.5); hold on;
plot(t_out, theta_ref_deg*ones(size(t_out)), '--', 'LineWidth', 1.2);
grid on;
xlabel('Time (s)'); ylabel('\theta (deg)');
legend('\theta','\theta_{ref}','Location','best');

%% ---- Figure 2: Wheel Speed vs Estimated Wheel speed ----
figure('Name','Wheel Speed');
plot(t_out, omega_w, 'LineWidth', 1.5); hold on;
plot(t_out, xhat(:,3), '--', 'LineWidth', 1.2);
grid on;
xlabel('Time (s)'); ylabel('\omega_w (rad/s)');
legend('\omega_w (true)','\omega_w (estimated)','Location','best');

%% ---- Figure 3: Theta dot vs Estimated Theta dot ----
figure('Name','Unmeasured State (theta-dot) Estimation');
plot(t_out, x(:,2), 'LineWidth', 1.5); hold on;
plot(t_out, xhat(:,2), '--', 'LineWidth', 1.2);
grid on;
xlabel('Time (s)'); ylabel('$\dot{\theta}$ (rad/s)','Interpreter','latex');
legend({'$\dot{\theta}$ (true)', '$\dot{\theta}$ (estimated)'}, ...
       'Location','best','Interpreter','latex');
```

```
%% ---- Figure 4: Wheel and thruster torque ----
figure('Name','Control Inputs');
plot(t_out, tau_w, 'LineWidth', 1.5); hold on;
plot(t_out, tau_t, 'LineWidth', 1.5);
grid on;
xlabel('Time (s)'); ylabel('Torque (N·m)');
legend('\tau_w (wheel)','\tau_t (thruster)','Location','best');

%% ==================== MULTI-STEP COMMAND ====================
% Keep your existing code untouched above. This is an additional test.

% ---- Multi-step simulation time ----
dt2 = 0.01;
t_end2 = 70;
t2 = 0:dt2:t_end2;

% ---- Multi-step reference in degrees ----
% 0–10s:   10 deg
% 10–25s:  30 deg
% 25–40s: -40 deg
% 40–60s:  0 deg
r2_deg = zeros(size(t2));
r2_deg(t2 >= 0  & t2 < 10) = 0;
r2_deg(t2 >= 10 & t2 < 25) = 10;
r2_deg(t2 >= 25 & t2 < 40) = 30;
r2_deg(t2 >= 40 & t2 < 55) = -40;
r2_deg(t2 >= 55 & t2 < 70)  = 0;

% Convert to radians for lsim
r2 = (pi/180) * r2_deg;

% ---- Simulate with SAME closed-loop system and SAME ICs ----
[y2, t2_out, x2_total] = lsim(sys_total, r2, t2, x0_total);

x2    = x2_total(:,1:3);
x2hat = x2_total(:,4:6);

% True outputs
theta2_deg = y2(:,1)*180/pi;

% Estimated theta from observer state xhat(1)
theta2hat_deg = x2hat(:,1)*180/pi;

% True and estimated spacecraft angular rate theta_dot
thetaDot2_true = x2(:,2);
thetaDot2_hat  = x2hat(:,2);

%% ---- Figure 5: Reference vs True Theta vs Estimated Theta ----
figure('Name','Multi-step: Ref vs True vs Estimated Theta');
plot(t2_out, r2_deg(1:length(t2_out)), '--', 'LineWidth', 1.3); hold on;
plot(t2_out, theta2_deg, 'LineWidth', 1.6);
%plot(t2_out, theta2hat_deg, ':', 'LineWidth', 1.6);
grid on;
xlabel('Time (s)'); ylabel('\theta (deg)');
legend('\theta_{ref}','\theta (true)','\theta (estimated)','Location','best');

%% ---- Figure 6: True vs Estimated Spacecraft Angular Rate ----
figure('Name','Multi-step: True vs Estimated \theta-dot');
plot(t2_out, thetaDot2_true, 'LineWidth', 1.6); hold on;
plot(t2_out, thetaDot2_hat, '--', 'LineWidth', 1.4);
grid on;
xlabel('Time (s)'); ylabel('$\dot{\theta}$ (rad/s)','Interpreter','latex');
legend({'$\dot{\theta}$ (true)', '$\dot{\theta}$ (estimated)'}, ...
       'Location','best','Interpreter','latex');

%% ==================== OBSERVER CONVERGENCE ====================
% Plant + observer simulated explicitly (so you can show convergence from IC mismatch)
% Uses the same K, F, L, A, B, C computed above.
```

```
dtN   = 0.01;
tEndN = 30;
tN    = 0:dtN:tEndN;

% Reference: step to 10 deg
rN_deg = 10*ones(size(tN));
rN     = (pi/180)*rN_deg;

% Initial conditions: mismatch to show convergence
xN    = [0; 0; 0];                  % true plant state
xhatN = [5*pi/180; 0; 20];          % estimator starts wrong on purpose

% Allocate logs
x_log    = zeros(3,length(tN));
xhat_log = zeros(3,length(tN));

for k = 1:length(tN)
    % Perfect measurement (NO noise)
    y_meas = C*xN;

    % Control law uses estimated state
    u = -K*xhatN + F*rN(k);    % u is 2x1  (tau_w, tau_t)

    % Plant dynamics
    xdot = A*xN + B*u;

    % Observer dynamics
    xhatdot = A*xhatN + B*u + L*(y_meas - C*xhatN);

    % Integrate (Euler)
    xN    = xN    + dtN*xdot;
    xhatN = xhatN + dtN*xhatdot;

    % Log
    x_log(:,k)    = xN;
    xhat_log(:,k) = xhatN;
end

% Convert for plotting
theta_true_deg = x_log(1,:)*180/pi;
theta_hat_deg  = xhat_log(1,:)*180/pi;

thetaDot_true  = x_log(2,:);
thetaDot_hat   = xhat_log(2,:);

%% Figure 7: theta (ref, true, estimated)
figure('Name','Observer Convergence (No Noise): Theta');
plot(tN, rN_deg, '--', 'LineWidth', 1.2); hold on;
plot(tN, theta_true_deg, 'LineWidth', 1.6);
plot(tN, theta_hat_deg, ':', 'LineWidth', 1.4, 'Color', [0.5 0.0 1.0]);
grid on;
xlabel('Time (s)'); ylabel('\theta (deg)');
legend('\theta_{ref}','\theta (true)','\theta (estimated)','Location','best');

%% Figure 8: theta-dot (true vs estimated)
figure('Name','Observer Convergence (No Noise): theta-dot');
plot(tN, thetaDot_true, 'LineWidth', 1.6); hold on;
plot(tN, thetaDot_hat,  '--', 'LineWidth', 1.4);
grid on;
xlabel('Time (s)'); ylabel('$\dot{\theta}$ (rad/s)','Interpreter','latex');
legend({'$\dot{\theta}$ (true)', '$\dot{\theta}$ (estimated)'}, ...
       'Location','best','Interpreter','latex');

%% Figure 9: omega_w (true vs estimated)  <-- ADD THIS FIGURE
omegaW_true = x_log(3,:);      % true wheel speed (rad/s)
omegaW_hat  = xhat_log(3,:);   % estimated wheel speed (rad/s)

figure('Name','Observer Convergence (No Noise): omega_w');
plot(tN, omegaW_true, 'LineWidth', 1.6); hold on;
```

```
plot(tN, omegaW_hat,  '--', 'LineWidth', 1.4);
grid on;
xlabel('Time (s)'); ylabel('\omega_w (rad/s)');
legend({'\omega_w (true)', '\omega_w (estimated)'}, 'Location','best');
```

---

## Tables 1-2 Simulation Code

---

```
%% ==================== PARAM SWEEP: MASS 100–1000 kg (Step 100 kg) ====================
% This section sweeps "satellite mass" and maps it to inertia values (Js, Jw),
% then simulates a 10-deg step and records performance/actuator metrics.
%
% IMPORTANT: In this script, the dynamics depend on inertias (Js, Jw), not mass directly.
% A simple, explicit mapping is used here:
%   Js(m) = Js_base * (m/m_base)      (linear scaling)
%   Jw(m) = Jw_base                   (kept constant here; update if you want wheel to scale)
%
% If you prefer a geometry-based inertia model (e.g., cylinder/box), replace the mapping below.

disp('==================== MASS SWEEP STUDY ====================');

% ---- Sweep definition ----
m_vec = (100:100:1000)';     % kg

% ---- Inertia mapping (EDIT IF DESIRED) ----
m_base  = 100;               % kg (reference mass)
Js_base = 20;                % kg*m^2 (your nominal spacecraft inertia at m_base)
Jw_base = 0.10;              % kg*m^2 (nominal wheel inertia)

% Option A (default): wheel inertia constant
Jw_map = @(m) Jw_base;

% Option B (uncomment to scale wheel inertia linearly with mass too)
% Jw_map = @(m) Jw_base * (m/m_base);

% ---- Reference step (10 deg) and simulation time ----
theta_ref_deg_sweep = 10;
t_sweep = 0:0.01:20;   % longer to accommodate slower large-inertia cases
r_sweep = (theta_ref_deg_sweep*pi/180) * ones(size(t_sweep));

% ---- Storage for results ----
N = length(m_vec);
Js_list = zeros(N,1);
Jw_list = zeros(N,1);
RiseTime_s = zeros(N,1);
SettlingTime_s = zeros(N,1);
MaxTauW_Nm = zeros(N,1);
MaxTauT_Nm = zeros(N,1);
MaxOmegaW_rads = zeros(N,1);
MaxThetaDot_rads = zeros(N,1);

% ---- Helper for rise/settling times ----
rise_low  = 0.10 * theta_ref_deg_sweep;
rise_high = 0.90 * theta_ref_deg_sweep;
settle_band = 0.02 * theta_ref_deg_sweep;  % 2% band in deg

for i = 1:N
    m = m_vec(i);

    % Map mass -> inertia
    Js_i = Js_base * (m/m_base);
    Jw_i = Jw_map(m);
```

```
Js_list(i) = Js_i;
Jw_list(i) = Jw_i;

% Rebuild A,B,C for this inertia set
A_i = [0 1 0;
       0 0 0;
       0 0 0];

B_i = [ 0      0;
       -1/Js_i  1/Js_i;
        1/Jw_i  0   ];

C_i = [1 0 0;
       0 0 1];
D_i = zeros(2,2);

% Recompute K, F, L for this plant (B changes with Js,Jw)
K_i = place(A_i, B_i, P_ctrb);
Acl_i = A_i - B_i*K_i;

% Reference gain for theta tracking
C_theta_i = [1 0 0];
M_i = -C_theta_i * (Acl_i \ B_i);
F_i = pinv(M_i) * 1;

% Observer gain (same desired poles, recomputed for this plant)
L_i = place(A_i', C_i', P_obsv)';

% Build augmented system (plant + observer), same structure as above
A_total_i = [A_i,            -B_i*K_i;
             L_i*C_i, (A_i - B_i*K_i - L_i*C_i)];

B_total_i = [B_i*F_i;
             B_i*F_i];

C_total_i = [C_i, zeros(size(C_i))];
D_total_i = zeros(size(C_i,1),1);

sys_total_i = ss(A_total_i, B_total_i, C_total_i, D_total_i);

% Initial conditions (same pattern: x0 = 0, xhat0 uses measured theta/omega)
x0_i = [0;0;0];
y0_i = C_i*x0_i;
xhat0_i = [y0_i(1); 0; y0_i(2)];
x0_total_i = [x0_i; xhat0_i];

% Simulate
[y_i, t_i, x_total_i] = lsim(sys_total_i, r_sweep, t_sweep, x0_total_i);

% Extract true states and estimated states
x_i    = x_total_i(:,1:3);
xhat_i = x_total_i(:,4:6);

theta_deg = y_i(:,1) * 180/pi;      % theta output in deg
omega_w   = y_i(:,2);               % omega_w output (rad/s)
theta_dot = x_i(:,2);               % spacecraft angular rate (rad/s)

% Reconstruct control inputs u(t) = -K*x_hat + F*r
u_i = zeros(length(t_i),2);
for k = 1:length(t_i)
    u_i(k,:) = (-K_i*xhat_i(k,:).' + F_i*r_sweep(k)).';
end
tau_w_i = u_i(:,1);
tau_t_i = u_i(:,2);

% ---- Metrics ----
% Rise time (10% -> 90%) using theta_deg
idx10 = find(theta_deg >= rise_low, 1, 'first');
idx90 = find(theta_deg >= rise_high, 1, 'first');
```

```
    if ~isempty(idx10) && ~isempty(idx90) && idx90 >= idx10
        RiseTime_s(i) = t_i(idx90) - t_i(idx10);
    else
        RiseTime_s(i) = NaN;
    end

    % Settling time (2% band around final value = 10 deg)
    upper = theta_ref_deg_sweep + settle_band;
    lower = theta_ref_deg_sweep - settle_band;
    outside = find(theta_deg > upper | theta_deg < lower);
    if isempty(outside)
        SettlingTime_s(i) = 0;
    else
        last_out = outside(end);
        if last_out < length(t_i)
            SettlingTime_s(i) = t_i(last_out);
        else
            SettlingTime_s(i) = NaN; % did not settle within sim window
        end
    end

    % Max actuator torques
    MaxTauW_Nm(i) = max(abs(tau_w_i));
    MaxTauT_Nm(i) = max(abs(tau_t_i));

    % Max wheel speed and max spacecraft rate
    MaxOmegaW_rads(i)   = max(abs(omega_w));
    MaxThetaDot_rads(i) = max(abs(theta_dot));
end

% ---- Compile results table ----
resultsTbl = table( ...
    m_vec, Js_list, Jw_list, ...
    RiseTime_s, SettlingTime_s, ...
    MaxTauW_Nm, MaxTauT_Nm, ...
    MaxOmegaW_rads, MaxThetaDot_rads, ...
    'VariableNames', { ...
      'Mass_kg','Js_kgm2','Jw_kgm2', ...
      'RiseTime_s','SettlingTime_s', ...
      'MaxTauWheel_Nm','MaxTauThruster_Nm', ...
      'MaxOmegaWheel_rads','MaxThetaDot_rads'});

disp('--- Sweep Results Table ---');
disp(resultsTbl);

% Optional: write to CSV for report
% writetable(resultsTbl, 'mass_sweep_results.csv');

%% ==================== FAST TEST: POLE SCALING vs TORQUE ====================
% Updates made to avoid NaN rise/settling times:
%  1) Longer simulation horizon (300 s) so slow designs reach 10%/90% and settle.
%  2) Coarser timestep (0.05 s) to keep runtime reasonable.
%  3) Settling time computed about the *actual final value* to avoid false NaNs.

disp('==================== FAST POLE-SCALING STUDY (UPDATED) ====================');

% ---- Fixed inertias ----
Js_fix = 20;
Jw_fix = 0.10;

A_fix = [0 1 0;
         0 0 0;
         0 0 0];

B_fix = [ 0         0;
         -1/Js_fix  1/Js_fix;
          1/Jw_fix  0     ];

C_fix = [1 0 0;
```

```
         0 0 1];

% ---- Step and sim time (UPDATED) ----
theta_ref_deg_pole = 10;
dt_pole = 0.05;                 % UPDATED: coarser time step (faster)
t_end_pole = 300;               % UPDATED: longer horizon for slow cases
t_pole = 0:dt_pole:t_end_pole;
r_pole = (theta_ref_deg_pole*pi/180) * ones(size(t_pole));

% ---- Rise/settle defs ----
rise_low  = 0.10 * theta_ref_deg_pole;
rise_high = 0.90 * theta_ref_deg_pole;
settle_band = 0.02 * theta_ref_deg_pole;   % 2% band in deg

% ---- Torque targets ----
tauTargets = [0.01 0.02 0.05 0.1 0.2 0.5 1 2]';

% ---- Base pole shape (your nominal) ----
p_base = [-0.4 -0.6 -1.5];

% ---- Alpha search range ----
alpha_grid = logspace(log10(0.03), log10(20), 60);   % allow slightly slower than before

% ---- Observer poles (fixed) ----
P_obsv_fix = P_obsv;

% ---- Initial conditions ----
x0 = [0;0;0];
y0 = C_fix*x0;
xhat0 = [y0(1); 0; y0(2)];
x0_total = [x0; xhat0];

% ---- Storage ----
M = length(tauTargets);
AlphaBest = NaN(M,1);
Pole1 = NaN(M,1); Pole2 = NaN(M,1); Pole3 = NaN(M,1);
RiseTime_s = NaN(M,1);
SettlingTime_s = NaN(M,1);
MaxTauW_Nm = NaN(M,1);
MaxTauT_Nm = NaN(M,1);

for it = 1:M
    tau_lim = tauTargets(it);

    bestAlpha = NaN;
    bestRise = NaN;
    bestSettle = NaN;
    bestTauW = NaN;
    bestTauT = NaN;

    % Scan alpha from large -> small to get fastest feasible in this family
    for ia = length(alpha_grid):-1:1
        alpha = alpha_grid(ia);
        P_try = alpha * p_base;

        % Gains
        try
            K = place(A_fix, B_fix, P_try);
        catch
            continue;
        end
        Acl = A_fix - B_fix*K;

        % Reference gain (unity DC gain for theta)
        C_theta = [1 0 0];
        Mmap = -C_theta * (Acl \ B_fix);
        F = pinv(Mmap) * 1;

        % Observer
```

```
    L = place(A_fix', C_fix', P_obsv_fix)';

    % Augmented system
    A_total = [A_fix,            -B_fix*K;
               L*C_fix, (A_fix - B_fix*K - L*C_fix)];
    B_total = [B_fix*F;
               B_fix*F];
    C_total = [C_fix, zeros(size(C_fix))];
    D_total = zeros(size(C_fix,1),1);

    sys_total = ss(A_total, B_total, C_total, D_total);

    % Simulate
    [y, t_out, x_total] = lsim(sys_total, r_pole, t_pole, x0_total);

    xhat = x_total(:,4:6);
    theta_deg = y(:,1) * 180/pi;

    % Control history (vectorized)
    U = (-xhat * K.') + (r_pole(:) * F.');   % Nx2
    tau_w = U(:,1);
    tau_t = U(:,2);

    maxTauW = max(abs(tau_w));
    maxTauT = max(abs(tau_t));

    % Feasible wrt wheel torque limit
    if maxTauW > tau_lim
        continue;
    end

    % ---- Rise time (10% -> 90%) ----
    idx10 = find(theta_deg >= rise_low, 1, 'first');
    idx90 = find(theta_deg >= rise_high, 1, 'first');

    % If still not reaching 90% by t_end, treat as not feasible timing-wise
    if isempty(idx10) || isempty(idx90) || idx90 < idx10
        continue;
    end
    riseT = t_out(idx90) - t_out(idx10);

    % ---- Settling time (2% band around final value) UPDATED ----
    theta_final = theta_deg(end);         % UPDATED: final simulated value
    upper = theta_final + settle_band;
    lower = theta_final - settle_band;

    outside = find(theta_deg > upper | theta_deg < lower);
    if isempty(outside)
        settleT = 0;
    else
        last_out = outside(end);
        if last_out < length(t_out)
            settleT = t_out(last_out);
        else
            % did not settle within sim window -> treat as not feasible
            continue;
        end
    end

    % First feasible from largest alpha is fastest in this family
    bestAlpha = alpha;
    bestRise = riseT;
    bestSettle = settleT;
    bestTauW = maxTauW;
    bestTauT = maxTauT;
    break;
end

if ~isnan(bestAlpha)
```

```
        AlphaBest(it) = bestAlpha;
        Pole1(it) = bestAlpha*p_base(1);
        Pole2(it) = bestAlpha*p_base(2);
        Pole3(it) = bestAlpha*p_base(3);
        RiseTime_s(it) = bestRise;
        SettlingTime_s(it) = bestSettle;
        MaxTauW_Nm(it) = bestTauW;
        MaxTauT_Nm(it) = bestTauT;
    end
end

poleScalingTbl = table( ...
    tauTargets, AlphaBest, Pole1, Pole2, Pole3, ...
    RiseTime_s, SettlingTime_s, MaxTauW_Nm, MaxTauT_Nm, ...
    'VariableNames', { ...
      'WheelTorqueTarget_Nm','Alpha','Pole1','Pole2','Pole3', ...
      'RiseTime_s','SettlingTime_s','MaxTauWheel_Nm','MaxTauThruster_Nm'});

disp('--- Pole Scaling Results (UPDATED) ---');
disp(poleScalingTbl);

% Optional:
% writetable(poleScalingTbl,'pole_scaling_vs_wheeltorque_UPDATED.csv');
```