Design, Modeling, and Testing of an Active Roll Control System for a Model Rocket Using Fin-Mounted Control Surfaces

Pierre Arbaji and Majd Boulos

McGill University

September 1, 2025

Author Note

Acknowledgments. We thank Andy Cai, Matteo Hapiot, and Prof. Nadarajah for their guidance and support throughout this project.

Correspondence. Direct correspondence to Pierre Arbaji (pierre.arbaji@mail.mcgill.ca) or Majd Boulos (majd.boulos@mail.mcgill.ca).

Design, Modeling, and Testing of an Active Roll Control System for a Model Rocket Using Fin-Mounted Control Surfaces

Webster's dictionary defines control as "to exercise restraining or directing influence over: regulate - for example, Control one's anger." In the same way that one's energy might be misdirected in emotion, the rocket's energy might be misdirected through uncontrolled motion, wasting efficiency and compromising stability. For that reason, we developed an active flight control system, to ensure that the rocket uses its energy efficiently. As a first step towards developing a full three axis flight control system, we focused on developing a roll control mechanism. Roll control is essential in a rocket to maintain stability and precise orientation. After research, design, and iteration, we implemented servo-controlled surfaces located on the fins' edge to control the rocket's roll rate. These are controlled by a PID controller, modeled and tuned in MATLAB. Simulations show that the controller effectively reduced the roll rate as well as increased stability during flight, which validates the feasibility of our design and sets the foundation for future development of pitch and yaw control systems.

1. Introduction

The McGill Rocket Team (MRT), founded in 2015, is a student-led design team that builds rockets and advances student involvement in aerospace. The team brings together fun, safety and innovation in order to build efficient, safe, and unique rockets. Throughout the years, the team has had several notable milestones, including winning first place in the Spaceport America Cup 2018 and flying the first MRT hybrid rocket in 2021. The team is now ready for a new long term plan that will heavily impact the performance of our rocket. This is where Active Flight Controls come into play.

Active Flight Controls (AFC) is a system that can control the roll, pitch, and yaw of a rocket using automatically actuated control surfaces. Due to the scope of AFC, this project will only focus on the control of roll, or more specifically roll rate. Current iterations of MRT rockets are able to achieve high apogees, however their trajectories impact their performance. This project aims to change that.

It is important to recognize that this project needs to meet certain criteria, both made by the team but also by Launch Canada (LC), the competition that we have been participating in for the past few years. LC's Design, Testing and Evaluation Guide (DTEG) section 7.0 discusses the main constraints of using AFC. Section 7.1 restricts the control to roll, pitch and yaw, while completely banning the use of guided controls. Section 7.2 discussed the need for the rocket to remain stable without the use of controls. Sections 7.3 and 7.4 respectively discuss several necessary failsafes and a system that is completely dormant during the thrust phase. Sections 7.5 and 7.6 discuss the importance of applying electronic and energetic safety mentioned in other sections of DTEG.

2. Background

In rocketry, roll is the rotation about the rocket's longitudinal axis. While it does not directly affect the trajectory like pitch and yaw, uncontrolled roll compromises stability, interferes with sensor data and strains structural components. For active control, a closed loop roll control algorithm is crucial to maintain a predictable and efficient flight profile.

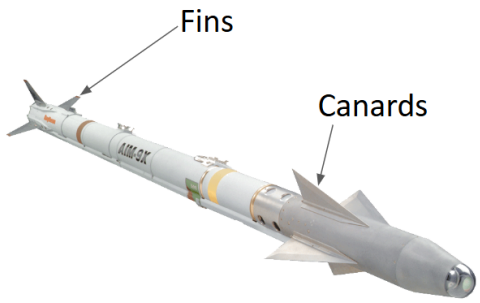

Figure 1

Fins v/s Canards

The two main approaches to roll control are control surfaces and canard fins. Control surfaces on fins are usually mounted on the rear of the rocket, and generate counteracting roll moments by deflecting airflow. This system has been demonstrated by BPS Space, which used a PID controller to control roll. On the other hand canard surfaces are located in the front of the rocket, and control roll by producing asymmetrical lift forces. This method has been used by a

research group at the University of Leeds, and is commonly used in amateur and professional rocketry, as well as military reasons due to its responsiveness and control authority.

Table 1

Comparison of roll control approaches

Category	**Fin Control Surfaces**	**Canard Control**
Placement	At the rocket's end	Near the front-middle
Stability	Increases passive stability; does not require constant actuation	Reduces stability; requires continuous correction
Responsiveness	Slower, but smoother and inherently stable	Quick response but less naturally stable
Mechanical Complexity	Easier to design and assemble; system already complete	More complex geometry and structural requirements
Electrical Complexity	Needs room in fincan, but longer wiring to avionics	Short wiring paths but less available space
Stress	Moderate torque; standard servos sufficient	Higher torque; requires stronger, often heavier servos
Drag	Minimal additional drag	Higher drag due to extra surfaces up front
Precision	Less precise but adequate for roll-only	High precision, better suited for full 3-axis control
Speed of Correction	Slower response suitable for roll	Fast response ideal for pitch/yaw

Given that this is a first iteration of active flight control focused on roll, we opted for a mechanically and electrically simpler model, using fin-mounted control surfaces. This approach enables faster prototyping and testing, and lays the foundation for a more complex control mechanism in the future iterations.

3. System Design

3.1 Rocket Overview

This single-stage, solid-motor testbed follows a conventional layout and is intentionally not optimized for mass or aerodynamics, prioritizing rapid development and evaluation of the fin-mounted, servo-actuated roll-control system.

Figure 2

Test Rocket Overview

Table 2

Rocket overview and key specifications

Category	Details
Dimensions	Total length: 36.2 in; diameter: 3.5 in; nose cone length: 8 in; fincan length: 8.2 in; control-surface span: 1.8 in^2.
Mass & weight distribution	Dry mass: 1.2 kg; estimated loaded mass: 1.69 kg; CG: 23.403 in; CP: 27.109 in; stability margin: 1.06 cal.
Materials	Airframe: PVC; fins: PETG; internal mounts: PETG.
Motor selection	Motor: Aerotech I300T; total impulse: 437 N; expected apogee: 3306 ft.

3.2 Fincan Design

The fincan consists of the section of the tube where the fins and control mechanism are located. Our design uses three identical clipped delta fins.While placing control surfaces at the rear maximizes the moment arm about the roll axis, it's also important that the rocket remains stable even when the control system is inactive. By placing bigger control surfaces closer to the tube, we are able to have a more stable rocket that can still be controlled using its control surfaces.

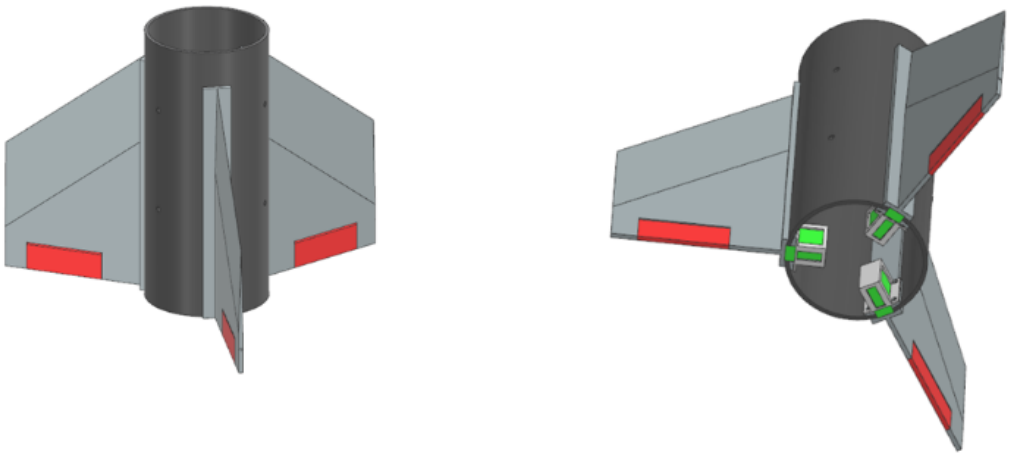

Figure 3

Fincan Assembly

A major design challenge was having a compact detachable mechanism that would fit in the rear of the rocket with the motor. To address this, the fins are detachable and screwable onto the surface of the tube, the servo is attached in its attributed section as seen in the picture. The local assembly (servo, control surface and fin) is done before each fin is mounted on the tube, at that point the only remaining step is to connect the wires to the avionics bay.

3.3 Servos and Electronics

Servos:

Servo size played an important role in their selection. Several mini-servos that fit our design were compared by the amount of stall torque at both 4.8V and 6V. In parallel, the amount of torque required by the servos in order to remain robust could be calculated using the

Reynold Transport Theorem. By using a constant uniform velocity assumption, this can be simplified to,

$$T \;=\; L \cdot \frac{\rho\, V^2\, A\, C_L}{2} \tag{1}$$

Where T is torque, L is the lever arm, p is density of air (assumed to be constant and at sea level), V is vertical velocity, A is the control surface's surface area and CL is the pressure coefficient, where sin(20) can be used as the control surface will never deviate more than that.

$$\begin{aligned} T &= (0.018\,\mathrm{m}) \cdot \frac{(1.225\,\mathrm{kg\,m^{-3}})\,(226\,\mathrm{m\,s^{-1}})^2\,(4.78\times10^{-4}\,\mathrm{m^2})\,\sin(10^\circ)}{2} \\ &\approx 0.092\,\mathrm{N\,m} \;=\; 0.938\,\mathrm{kg\,cm}. \end{aligned} \tag{2}$$

Applying a factor of safety of 2, we would need a servo with stall torque of 1.876 kgcm. Our servo that we chose is the EMAX ES08MA II [cite servo website], which has 1.6kgf.cm at 4.8V and 2.0kgf.cm at 6.0V. At 6.0V, it would be enough to deflect the servo by 20 degrees, while at 4.8V, servos would need to deflect less in order to maintain a similar factor of safety.

Flight Computer:

The base flight computer that will be used in all primary AFC testing will be the Arduino MEGA. It is a lightweight, inexpensive, and simple to use microcontroller that can allow use to test all mechanical aspects of the AFC. Its width is about 3.3 inches, meaning that it can fit nicely into our mini-rocket 3.3 inner diameter using a support structure. Its operating voltage is 5V while it could run at recommended voltages up to 12V, which is suitable for our servos.

The inertial measurement unit (IMU) used is a GFTGIW IMU 10DOF. Capable of measuring all the needed variables, including roll and multi-axes speeds.The IMU code C++ was created and calibrated in order to obtain realistic values.

Both the Arduino and the IMU will be substituted with a more complex FC, similar to ones already created by the students on the team.

Wiring and power:

The following describes the simple electrical hardware setup that was done. Note that while simple wires are used to connect the breadboard, the arduino, and the other electrical components, a perfboard will be used for final assemblies.

The Arduino, powered by the 9V rechargeable battery, supplies direct power to the IMU through the 3.3V port. The IMU can equally run on 5V; however, it seems unnecessary as it does not require much power to run. The servos are wired to the 5V arduino port during the initial code testing phase, but will be wired to a 9V rechargeable battery, with voltage controlled using external resistors. It is crucial to limit the voltage going across each servo to 6V, as higher voltages are not in the operational range and could possibly damage or break the servos, which accidentally occured during one of the wiring trials in the beginning of this project.

The future flight computer will require a more complex power system due to the several electronics that will be used. Although further studies and tests are needed, two rechargeable redundant 9V batteries could be used to power the FC. The servos should have their own power supply system, a 9V battery, while also being able to switch the source to the FC batteries as a redundancy.

The choice of rechargeable batteries was mainly due to the nature of this project. Reducing the amount of fuel used should also be accompanied by a greener choice of electrical power. They are also the cheaper alternative in the long run. might require more power)

4. Modeling and Simulation

4.1 Open Rocket model

OpenRocket is an open-source, high-fidelity 6-DOF rocket simulation software that enables detailed modeling of both the vehicle and its flight environment. By incorporating real GPS-derived atmospheric data from GFS forecasts, it can generate accurate wind, temperature, and air density profiles specific to our launch site and date.

We used OpenRocket to validate that the test rocket design is both realistic and compliant

with our stability requirements before physical construction. An exact digital replica of the rocket was created, including precise geometry, materials, component masses, and the selected motor. This ensured that predicted aerodynamic and dynamic behavior closely reflects the real vehicle.

Beyond basic trajectory predictions, OpenRocket provides critical parameters such as moments of inertia, roll damping coefficients, and time-resolved velocity and altitude profiles. These outputs form the foundation for the next stage of the project, simulation and control system development, by supplying accurate, flight-representative input values for our MATLAB model.

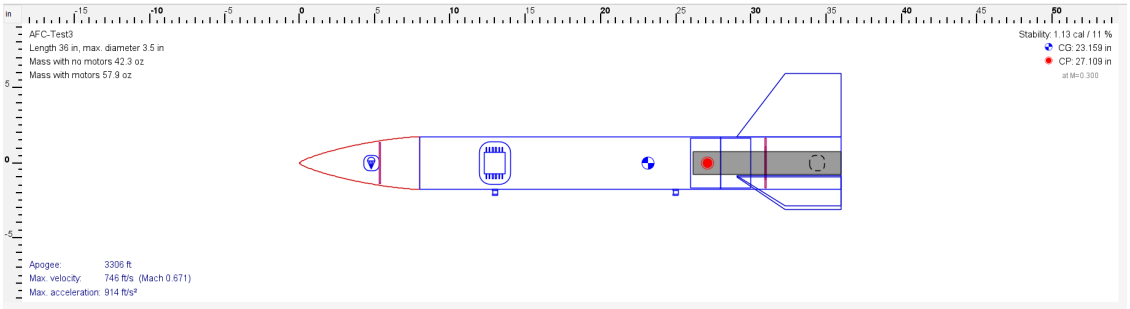

Figure 4

OpenRocket model of the test rocket, with all major components accurately represented

4.2 Matlab Model

The MATLAB model developed simulates the rocket's roll-rate control system, incorporating dynamic pressure effects, fin cant torques, aerodynamic damping, and actuator lag. This section details the model's structure, control strategy, equations of motion, torque components, tuning process, design rationale, and safety mechanisms, with updates reflecting controller enhancements.

4.21 Equations of Motion. The rocket's roll dynamics are modeled using the rotational equation of motion about the roll axis:

$$I_{xx} \cdot \dot{\omega} = T_{\text{act}} + T_{\text{cant}} - T_{\text{damp}} \tag{3}$$

where:

- $I_{xx} = 1.375 \times 10^{-5}\,\mathrm{kg \cdot m^2}$ is the moment of inertia about the roll axis, calculated from the rocket's mass distribution and geometry.
- $\dot{\omega}$ is the angular acceleration (rad/s^2).
- T_{act} is the actual control torque from fin deflection.
- T_{cant} is the torque induced by the fin cant angle.
- $T_{\mathrm{damp}} = D_{\mathrm{roll}} \cdot \omega$ is the aerodynamic damping torque, with $D_{\mathrm{roll}} = 1.0 \times 10^{-5}\,\mathrm{N \cdot m \cdot s/rad}$ estimated from wind tunnel data and scaled for the rocket's size.

This equation is numerically integrated using the Euler method with a timestep $dt = 0.01\,\mathrm{s}$, updating the roll rate as:

$$\omega(k) = \omega(k-1) + \dot{\omega}(k) \cdot dt \tag{4}$$

The roll angle integrates the rate: $\dot{\phi} = \omega$.

4.22 Aerodynamic Torques and Controller Updates.

Aerodynamic Torques. A small, fixed cant on each main fin generates a constant roll torque that scales with dynamic pressure:

$$T_{\mathrm{cant}} = K_{\mathrm{cant}} \cdot \delta_{\mathrm{cant}}, \qquad K_{\mathrm{cant}} = n \cdot q \cdot S_{\mathrm{fin}} \cdot c \cdot a_\ell \cdot d_{\mathrm{fin}}. \tag{5}$$

Here $q = \frac{1}{2}\rho v^2$ is the dynamic pressure, $n = 3$ is the number of fins, $S_{\mathrm{fin}} = 0.01441\,\mathrm{m^2}$ is the main fin planform area, $c = 0.1358\,\mathrm{m}$ the representative chord, $a_\ell = 0.0573\,\mathrm{rad^{-1}}$ the lift slope (per-radian), and $d_{\mathrm{fin}} = 0.04445\,\mathrm{m}$ the roll moment arm of the fin force.

Each fin also carries a trailing control surface (servo-driven). Their net roll torque is

$$T_{\mathrm{act}} = K_{\mathrm{ctrl}} \cdot \delta_{\mathrm{servo}}, \qquad K_{\mathrm{ctrl}} = n \cdot q \cdot S_{\mathrm{ctrl}} \cdot c \cdot a_\ell \cdot d_{\mathrm{ctrl}}. \tag{6}$$

Geometry used in the model:

$$S_{\mathrm{fin}} = 22.35\,\mathrm{in^2} = 0.01441\,\mathrm{m^2}, \qquad d_{\mathrm{fin}} = 0.04445\,\mathrm{m}, \tag{7}$$

$$S_{\mathrm{ctrl}} = 1.8\,\mathrm{in^2} = 0.001161\,\mathrm{m^2}, \qquad d_{\mathrm{ctrl}} = 4.15\,\mathrm{in} = 0.10541\,\mathrm{m}. \tag{8}$$

The maximum commanded surface angle is now $\delta_{\max} = \pm 45^\circ$.

4.23 PID Controller. The model employs a Proportional-Integral-Derivative (PID) controller with filtered derivative and feed-forward to stabilize the rocket's roll rate at 0 rad/s. The control law is implemented as operating in the torque domain and tracking a rate reference ω_{ref}:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e \, \mathrm{d}t + K_d \cdot \dot{e}_f(t) + u_{\text{ff}}(t), \tag{9}$$

where:

- u is the commanded torque ($\mathrm{N} \cdot \mathrm{m}$),
- $e = \omega_{\text{ref}} - \omega$,
- $K_p = 0.0042$, $K_i = 0.0070$, $K_d = 2.5 \times 10^{-5}$.

The derivative is filtered as $\dot{e}_f = \frac{\tau_d s}{1+\tau_d s} \cdot \dot{e}$. The feed-forward term is:

$$u_{\text{ff}}(t) = -T_{\text{cant}}(t) + D_{\text{roll}} \cdot \omega_{\text{ref}}(t). \tag{10}$$

Actuator Model, Deadband, and Saturation. Commanded surface angle passes through a hinge deadband of $\pm 0.5^\circ$ and a first-order servo lag with $\tau_{\text{servo}} = 0.02\,\mathrm{s}$. Torque is saturated by current authority: $u \in [-T_{\max}, T_{\max}]$, with $T_{\max} = K_{\text{ctrl}} \cdot \delta_{\max}$.

Anti-Windup and Low-q Guard. Back-calculation anti-windup with $k_{\text{aw}} = 0.4$ and integrator clamp to $\pm 1.1 \cdot T_{\max}$. When $K_{\text{ctrl}} < 10^{-8}$, integrator is bled and surfaces are held at zero.

4.24 Justification for Design Decisions. The PID with feed-forward control was chosen for its ability to handle time-varying dynamics and track nonzero references during rocket ascent, where dynamic pressure and aerodynamic forces change with altitude and velocity. The inclusion of dynamic pressure q ensures the model reflects real-world conditions, using the ISA troposphere model for air density ($\rho = \rho_0 \cdot (T/T_0)^{g_0/(R \cdot L)-1}$) up to 11 km, with an exponential fallback above. The 1.5° fin cant introduces a persistent torque T_{cant}, necessitating active control, while D_{roll} accounts for natural damping. The updated $\delta_{\max} = \pm 45^\circ$ and servo lag (τ_{servo}) mimic hardware constraints, ensuring practicality. The model incorporates two distinct fin areas: the actual fin area ($S_{\text{fin}} = 0.01441\,\mathrm{m}^2$) and the control surface area

($S_{\text{ctrl}} = 0.001161\,\text{m}^2$) for active roll control, with the arm length ($d_{\text{ctrl}} = 0.10541\,\text{m}$) enhancing torque authority.

4.25 Gain Tuning. The PID gains were iteratively tuned based on simulation results:

- $K_p = 0.0042$
- $K_i = 0.0070$
- $K_d = 2.5 \times 10^{-5}$
- $\tau_{\text{servo}} = 0.02\,\text{s}$

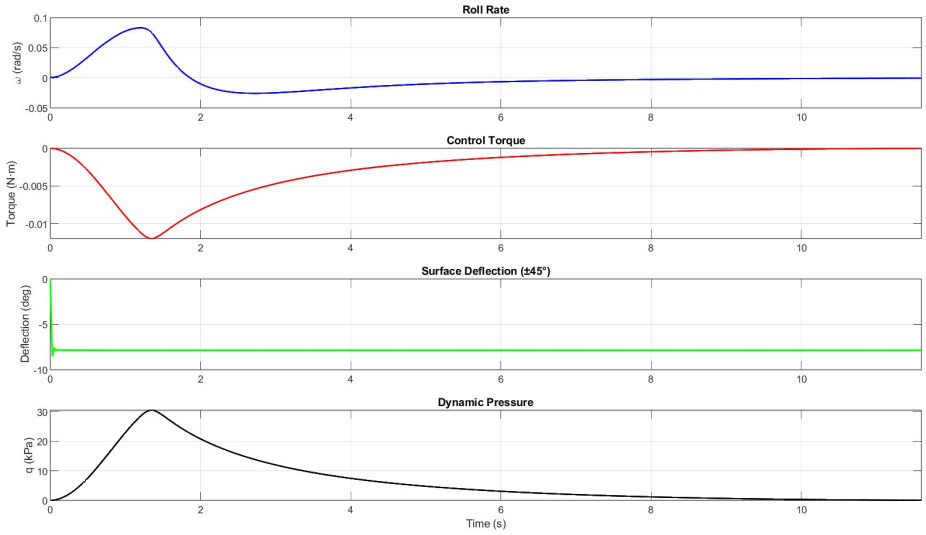

Figure 5

PID Control Response

4.26 Safety Logic. Safety is ensured through:

- **Torque Saturation**: The commanded torque u is capped at $\pm T_{\max} = K_{\text{ctrl}} \cdot \delta_{\max}$, preventing excessive fin deflection.
- **Anti-Windup**: Back-calculation with $k_{\text{aw}} = 0.4$ and integrator clamping to $\pm 1.1 \cdot T_{\max}$ avoid windup.
- **Low-q Guard**: When $K_{\text{ctrl}} < 10^{-8}$, the integrator is bled, and surfaces are zeroed to prevent instability.

4.27 Performance Evaluation. Simulation results, plotted over the flight duration, show the roll rate stabilizing near 0 rad/s or tracking commanded rates, with actual torque and fin deflection responding to T_{cant} and dynamic pressure variations. The final roll rate, indicates the controller's effectiveness. The model assumes accurate flight data for velocity and altitute, that come from Open Rocket simulations.

4.28 Reference-Tracking Test: $0 \to 2 \to 0$ rad/s (11 s). To validate tracking under time-varying q, we command

$$\omega_{\text{ref}}(t) = \begin{cases} 0, & 0 \le t < 5.0\,\text{s}, \\ 2\,\text{rad/s}, & 5.0 \le t < 8.5\,\text{s}, \\ 0, & 8.5 \le t \le 11.0\,\text{s}, \end{cases} \tag{11}$$

with feed-forward term $+D_{\text{roll}} \cdot \omega_{\text{ref}}$ to hold the plateau. The observed response shows a brief saturation plateau followed by an exponential taper as angle error shrinks, with minimal chatter and no windup.

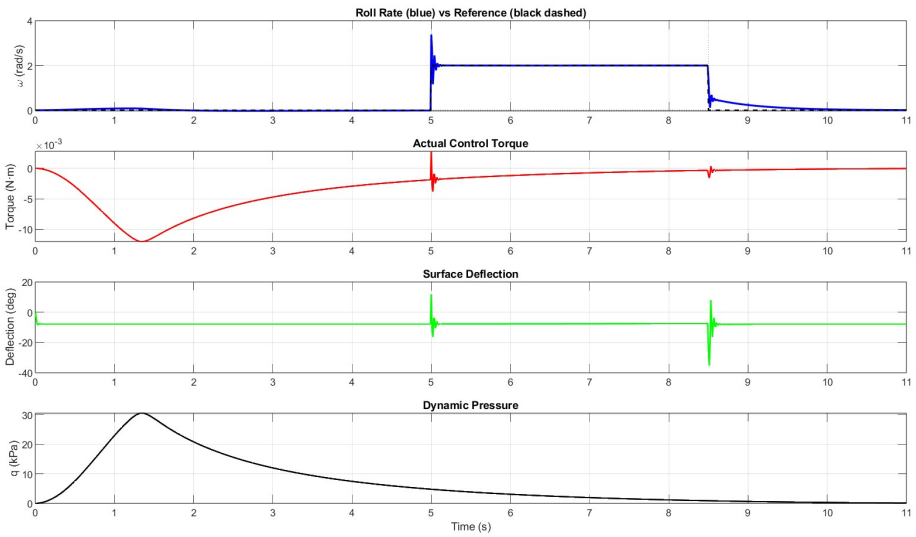

Figure 6

Reference Tracking Response

4.29 Angle Steps: Two $+90°$ Rotations (11 s). A cascaded outer angle loop produces a

rate command from angle error:

$$\omega_{\text{ref}}(t) = \text{sat}\Big(K_\phi[\phi_{\text{ref}}(t) - \phi(t)], \pm\omega_{\max}\Big), \qquad K_\phi = 1.5\,\text{s}^{-1},\ \omega_{\max} = 2\,\text{rad/s}. \tag{12}$$

The reference angle steps are $\phi_{\text{ref}} = 0$ for $t < 5.0\,\text{s}$, $+90°$ for $5.0 \le t < 8.5\,\text{s}$, and $+180°$ for $8.5 \le t \le 11.0\,\text{s}$. This yields two clean $+90°$ rotations within the window while respecting $\pm 45°$ deflection and servo dynamics.

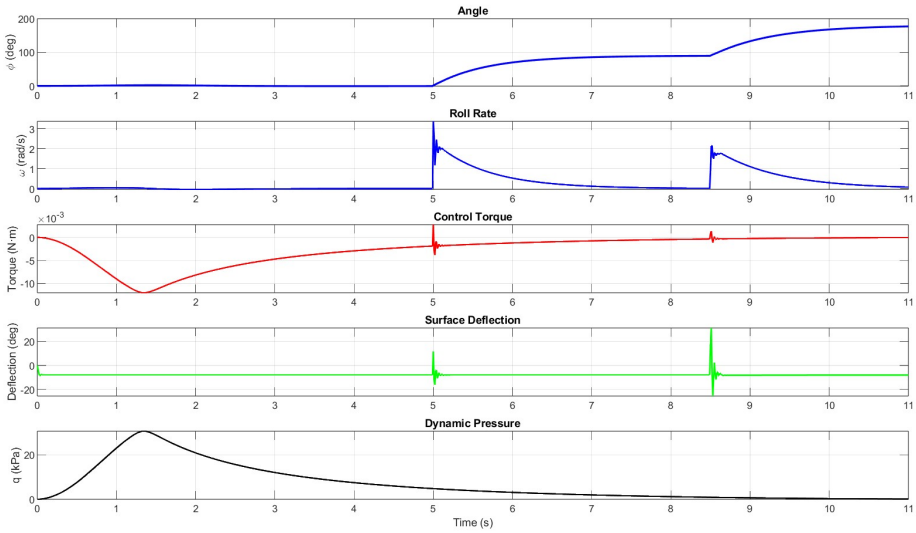

Figure 7

Angle Steps Response

5. Conclusion and Future Work

This project successfully demonstrated the design, modeling, and simulation of an active roll control system for a model rocket using fin-mounted control surfaces. Through iterative development, we created a mechanically and electrically feasible system and validated its performance using OpenRocket and MATLAB-based simulations. The results confirm that a PID controller integrated with aerodynamic models can effectively reduce roll rate and improve overall flight stability.

Future work will focus on quantifying and reducing overshoot in the roll response by re-tuning the controller gains, as well as accounting for the influence of structural vibrations and

aeroelastic effects. Incorporating these refinements will improve both precision and robustness of the control system.

The immediate next step is to conduct a full-scale flight test of the prototype rocket. Flight data will provide critical insight into real-world actuator response, aerodynamic effects, and system latency, allowing us to refine and re-tune the controller. Following these tests, further iterations will focus on validating repeatability and robustness of the roll control system under varying flight conditions.

Ultimately, this work serves as the foundation for a more comprehensive active flight control system. Once roll control is fully developed and validated, the project will progress toward implementing pitch and yaw control, enabling full three-axis active stabilization in future rockets.

References

[1] Bertin, J. J., & Cummings, R. M. (2014). *Aerodynamics for engineers* (6th ed.). Pearson Education. https://books.google.com/books/about/Aerodynamics_for_Engineers.html?id=CcgvAAAAQBAJ

[2] Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2014). *Feedback control of dynamic systems* (7th ed.). Pearson Education. https://books.google.com/books/about/Feedback_Control_of_Dynamic_Systems.html?id=yO2hoAEACAAJ

[3] Meriam, J. L., & Kraige, L. G. (2019). *Engineering mechanics: Dynamics* (9th ed.). Wiley. https://www.wiley.com/en-us/Engineering%2BMechanics%3A%2BDynamics%2C%2B9th%2BEdition-p-x001053661

[4] Menon, P. K., & Briggs, M. E. (1991). Design of a roll autopilot for a guided missile. *Journal of Guidance, Control, and Dynamics, 14*(1), 183–189. https://doi.org/10.2514/3.20615

[5] NASA Glenn Research Center. (n.d.). Beginner's guide to aeronautics: Flight control surfaces. https://www.grc.nasa.gov/www/k-12/airplane/contro.html

[6] Nelson, R. C. (1998). *Flight stability and automatic control* (2nd ed.). McGraw-Hill Education. https://books.google.com/books/about/Flight_Stability_and_Automatic_Control.html?id=Z4lTAAAAMAAJ

[7] Ogata, K. (2009). *Modern control engineering* (5th ed.). Pearson Education. https://www.pearson.com/en-us/subject-catalog/p/modern-control-engineering/P200000003521/9780137551064

[8] NASA Glenn Research Center. (n.d.). Lift equation. Retrieved September 21, 2025, from https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/lift-equation/

[9] Kinnas, S. A. (n.d.). Reynolds Transport Theorem (RTT) notes. Department of Civil, Architectural and Environmental Engineering, University of Texas at Austin. https://www.caee.utexas.edu/prof/kinnas/319LAB/fmnotes/RTT.pdf

[10] EMAX. (n.d.). EMAX ES08MA II 12g mini metal gear analog servo for RC model robot PWM servo. https://emaxmodel.com/products/emax-es08ma-ii-12g-mini-metal-gear-analog-servo-for-rc-model-robot-pwm-servo

[11] Arduino. (n.d.). Arduino Mega 2560 Rev3. https://store.arduino.cc/collections/giga/products/arduino-mega-2560-rev3

[12] McGill Rocket Team. (n.d.). About us. https://www.mcgillrocketteam.com/

[13] OpenRocket. (2021). RSE file. Retrieved January 16, 2022, from http://wiki.openrocket.info/RSE_File

Appendix

Appendix A: Source Code

A.1 Roll Controller with Dynamic Pressure (Final)

```
%% roll_controller_dynamic_pressure_final.m
% 1.8 1 -in control surface on 4.15-in arm,  45  travel
% Feed-forward cancels fin-cant torque each step.
% PI-D gains tuned for <~3 s settle with minimal chatter.
% Includes a low-q guard to avoid integrator windup near liftoff.

clear; clc; close all;

%% 1. Geometry & constants
S_fin   = 22.35 * 0.0254^2;                 % 0.01441 m   (full fin area)
S_ctrl  = 1.8   * 0.0254^2;                 % 0.001161 m  (control
    surface area)
n       = 3;
c       = 0.1358;                           % m (    ctrl-surface chord)
d_fin   = 0.04445;                          % m (cant arm for fixed fin)
d_ctrl  = ((3.5 + 0.8)/2 + 2) * 0.0254;     % 0.10541 m (4.15 in arm)
liftSlope = 0.0573;                         % 1/rad (CL_alpha approx)
deltaCant = deg2rad(1.5);
deltaMax  = deg2rad(45);                    %   45   servo limit (matches
    header)

Ixx     = 1.375e-5;                         % kg m  (roll inertia)
D_roll  = 1e-5;                              % N m s/rad (small aero
    roll damping)
dt      = 0.01;                              % s (controller step, 100 Hz)
servoTau= 0.02;                              % s ( 20  ms servo 1st-order
     lag)

%% 2. PI-D gains
```

```
Kp     = 0.0042;                                % N m per (rad/s)
Ki     = 0.0070;                                % N m per rad
Kd     = 0.000025;                              % N m per (rad/s^2)
tau_d = 0.04;                                   % s (40 ms derivative low-
    pass)
deadBand = deg2rad(0.5);                        %  0 .5   hinge friction
    equiv
lowKctrlThresh = 1e-8;                          % torque authority guard

%% 3. Load flight profile (ft, ft/s     m, m/s)
altData = readmatrix('Altitude vs time.csv');
velData = readmatrix('Velocity vs time.csv');
t_alt = altData(:,1);   alt_m  = altData(:,2) * 0.3048;
t_vel = velData(:,1);   vel_ms = velData(:,2) * 0.3048;

T_total = ceil(max([t_alt; t_vel]) / dt) * dt;
time    = 0:dt:T_total;  N = numel(time);
alt_now = interp1(t_alt, alt_m,  time, 'linear', 'extrap');
vel_now = interp1(t_vel, vel_ms, time, 'linear', 'extrap');

%% 4. ISA constants (troposphere + simple exponential fallback)
T0=288.15; rho0=1.225; L=0.0065; g0=9.80665; R=287.058;

%% 5. Pre-allocate state vectors
omega = zeros(1,N);
deltaCmd = zeros(1,N);
torqueCtrl = zeros(1,N);
q_hist = zeros(1,N);
integrator = 0; deltaServo = 0; prevErr = 0; prevDf = 0;

%% 6. Main simulation loop
for k = 2:N
    %        Atmosphere & dynamic pressure
```

```
    T   = T0 - L*alt_now(k);
    rho = (T>0)*rho0*(T/T0)^(g0/(R*L)-1) + (T<=0)*rho0*exp(-alt_now(k)
        /7000);
    v   = max(vel_now(k), 0);               % no negative speeds
    q   = 0.5*rho*v^2;   q_hist(k) = q;

    %       Aerodynamic torque gain (cant vs control surfaces)
    K_cant = n*q*S_fin  * c * liftSlope * d_fin;   % N m per rad (cant)
    K_ctrl = n*q*S_ctrl * c * liftSlope * d_ctrl;  % N m per rad (servo
        surfaces)
    T_cant = K_cant * deltaCant;                    % fixed bias torque
    Tmax   = K_ctrl * deltaMax;                     % current authority

    %       Low-q authority guard: freeze integrator when K_ctrl ~ 0
    if K_ctrl < lowKctrlThresh
        integrator = 0.98*integrator;             % gentle bleed
        deltaServo = 0;
        deltaCmd(k) = 0;
        torqueCtrl(k) = 0;
        % Roll dynamics with cant only
        omega_dot = (T_cant - D_roll*omega(k-1)) / Ixx;
        omega(k)  = omega(k-1) + omega_dot*dt;
        prevErr = -omega(k-1);                    % keep derivative stable
        continue
    end

    %       Control error & filtered derivative
    err  = -omega(k-1);                           % target    = 0
    d_raw= (err - prevErr)/dt;
    d_f  = prevDf + (dt/(tau_d+dt))*(d_raw - prevDf);
    prevErr = err;  prevDf = d_f;

    %       PI-D (torque-domain) + feedforward
```

```
    integrator = integrator + Ki*err*dt;
    % Clamp integrator to  1 .1*Tmax so it cannot demand more than
       authority
    integLim = 1.1*Tmax;
    integrator = min(max(integrator, -integLim), integLim);

    u_pid = Kp*err + integrator + Kd*d_f;          % N m
    u_ff  = -T_cant;                               % cancel cant each step
    u_tot = u_ff + u_pid;

    %        Saturation & back-calculation anti-windup
    u_sat = max(min(u_tot, Tmax), -Tmax);
    integrator = integrator + (u_sat - u_tot)*0.4; % back-calc term

    %        Desired surface deflection (with dead-band and limits)
    delta_des = u_sat / K_ctrl;
    if abs(delta_des) < deadBand
        delta_des = 0;
    else
        delta_des = delta_des - sign(delta_des)*deadBand;
    end
    delta_des = max(min(delta_des, deltaMax), -deltaMax);

    % Servo 1st-order lag (acts like a simple rate/response limit)
    deltaServo = deltaServo + (delta_des - deltaServo)*(dt/servoTau);
    deltaServo = max(min(deltaServo, deltaMax), -deltaMax);
    deltaCmd(k) = deltaServo;

    torqueCtrl(k) = K_ctrl * deltaServo;

    %        Roll dynamics
    omega_dot = (torqueCtrl(k) + T_cant - D_roll*omega(k-1)) / Ixx;
    omega(k)  = omega(k-1) + omega_dot*dt;
```

```
120 end
121
122 %% 7. Zero-cross time (if any)
123 idxZero = find(omega(2:end).*omega(1:end-1) <= 0, 1);
124 if ~isempty(idxZero)
125     idxZero = idxZero + 1;
126     tZero = time(idxZero);
127 else
128     tZero = NaN;
129 end
130
131 %% 8. Plots
132 figure('Name','Roll Control     final,   45   , retuned');
133 subplot(4,1,1)
134 plot(time, omega, 'b', 'LineWidth', 1.4); grid on
135 ylabel('\omega (rad/s)'); title('Roll Rate'); xlim([0 T_total])
136
137 subplot(4,1,2)
138 plot(time, torqueCtrl, 'r', 'LineWidth', 1.4); grid on
139 ylabel('Torque (N m)'); title('Control Torque'); xlim([0 T_total])
140
141 subplot(4,1,3)
142 plot(time, rad2deg(deltaCmd), 'g', 'LineWidth', 1.4); grid on
143 ylabel('Deflection (deg)'); title('Surface Deflection (  45  )'); xlim([0
    T_total])
144
145 subplot(4,1,4)
146 plot(time, q_hist/1000, 'k', 'LineWidth', 1.4); grid on
147 xlabel('Time (s)'); ylabel('q (kPa)'); title('Dynamic Pressure'); xlim([0
    T_total])
148
149 if isnan(tZero)
```

```
    fprintf('Peak |  | = %.1f rad/s, no zero-cross detected, final    =
        %.3f rad/s\n', ...
            max(abs(omega)), omega(end));
else
    fprintf('Peak |  | = %.1f rad/s, crosses zero at %.2f s, final    =
        %.3f rad/s\n', ...
            max(abs(omega)), tZero, omega(end));
end
```

Listing 1: Final roll controller with dynamic pressure scaling.

A.2 Reference Tracking Roll Controller

```
%% roll_controller_dynamic_pressure_final_ref.m
% Reference tracking: 0     2 rad/s at 5.0 s     0 at 8.5 s

clear; clc; close all;

%% 1. Geometry & constants
S_fin   = 22.35 * 0.0254^2;                 % 0.01441 m    (full fin area)
S_ctrl  = 1.8   * 0.0254^2;                 % 0.001161 m   (control
    surface area)
n       = 3;
c       = 0.1358;                           % m (    ctrl-surface chord)
d_fin   = 0.04445;                          % m (cant arm for fixed fin)
d_ctrl  = ((3.5 + 0.8)/2 + 2) * 0.0254;     % 0.10541 m (4.15 in arm)
liftSlope = 0.0573;                         % 1/rad (CL_alpha approx)
deltaCant = deg2rad(1.5);
deltaMax  = deg2rad(45);                    %   45   servo limit

Ixx     = 1.375e-5;                         % kg m   (roll inertia)
D_roll  = 1e-5;                              % N m s/rad (small aero
    roll damping)
dt      = 0.01;                              % s (controller step, 100 Hz)
```

```
servoTau= 0.02;                                % s (  20   ms servo lag)

%% 2. PI-D gains
Kp     = 0.0042;                               % N m per (rad/s)
Ki     = 0.0070;                               % N m per rad
Kd     = 0.000025;                             % N m per (rad/s^2)
tau_d = 0.04;                                  % s (40 ms derivative low-
    pass)
deadBand = deg2rad(0.5);                       %  0 .5   hinge friction
    equiv
lowKctrlThresh = 1e-8;                         % torque authority guard

%% 3. Reference timing (0     2     0)
t_step_up    = 5.0;       % s  (0       +2)
t_step_down = 8.5;       % s  (+2       0)
rate_high    = 2.0;       % rad/s (target plateau)

%% 4. Load flight profile ( f t m , ft/ s m /s)
altData = readmatrix('Altitude vs time.csv');
velData = readmatrix('Velocity vs time.csv');
t_alt = altData(:,1);    alt_m  = altData(:,2) * 0.3048;
t_vel = velData(:,1);    vel_ms = velData(:,2) * 0.3048;

% Use the whole CSV horizon; plots will zoom to 0  11   s
T_total = ceil(max([t_alt; t_vel]) / dt) * dt;
time    = 0:dt:T_total;  N = numel(time);
alt_now = interp1(t_alt, alt_m,  time, 'linear', 'extrap');
vel_now = interp1(t_vel, vel_ms, time, 'linear', 'extrap');

% Build reference vector
omega_ref = zeros(1,N);
omega_ref(time >= t_step_up & time < t_step_down) = rate_high;

```

```
%% 5. ISA constants
T0=288.15; rho0=1.225; L=0.0065; g0=9.80665; R=287.058;

%% 6. Pre-allocate state vectors
omega = zeros(1,N);
deltaCmd = zeros(1,N);
torqueCtrl = zeros(1,N);
q_hist = zeros(1,N);
integrator = 0; deltaServo = 0; prevErr = 0; prevDf = 0;

%% 7. Main simulation loop
for k = 2:N
    %        Atmosphere & dynamic pressure
    T   = T0 - L*alt_now(k);
    rho = (T>0)*rho0*(T/T0)^(g0/(R*L)-1) + (T<=0)*rho0*exp(-alt_now(k)
       /7000);
    v   = max(vel_now(k), 0);
    q   = 0.5*rho*v^2;   q_hist(k) = q;

    %        Aerodynamic torque gains
    K_cant = n*q*S_fin  * c * liftSlope * d_fin;   % N m per rad (cant)
    K_ctrl = n*q*S_ctrl * c * liftSlope * d_ctrl;  % N m per rad (servo
       surfaces)
    T_cant = K_cant * deltaCant;                   % fixed bias torque
    Tmax   = K_ctrl * deltaMax;                    % current authority

    %        Low-q authority guard
    if K_ctrl < lowKctrlThresh
        integrator = 0.98 * integrator;            % gentle bleed
        deltaServo = 0;
        deltaCmd(k) = 0;
        torqueCtrl(k) = 0;
        % Roll dynamics with cant only
```

```
        omega_dot = (T_cant - D_roll*omega(k-1)) / Ixx;
        omega(k)  = omega(k-1) + omega_dot*dt;
        prevErr = omega_ref(k) - omega(k-1);       % keep derivative
            stable
        continue
    end

    %        Reference tracking error & filtered derivative
    err  = omega_ref(k) - omega(k-1);              % track  _ref
    d_raw= (err - prevErr)/dt;
    d_f  = prevDf + (dt/(tau_d+dt))*(d_raw - prevDf);
    prevErr = err;  prevDf = d_f;

    %        PI-D (torque-domain) + feed-forward
    %     Feed-forward cancels cant AND adds D_roll*omega_ref to hold the
        plateau.
    integrator = integrator + Ki*err*dt;
    integLim = 1.1*Tmax;                           % anti-windup clamp
    integrator = min(max(integrator, -integLim), integLim);

    u_pid = Kp*err + integrator + Kd*d_f;          % N m
    u_ff  = -T_cant + D_roll*omega_ref(k);         % N m (cant cancel +
        steady-rate hold)
    u_tot = u_ff + u_pid;

    %        Saturation & back-calculation anti-windup
    u_sat = max(min(u_tot, Tmax), -Tmax);
    integrator = integrator + (u_sat - u_tot)*0.4;

    %        Desired surface deflection (deadband + limits)
    delta_des = u_sat / K_ctrl;
    if abs(delta_des) < deadBand
        delta_des = 0;
```

```
    else
        delta_des = delta_des - sign(delta_des)*deadBand;
    end
    delta_des = max(min(delta_des, deltaMax), -deltaMax);

    % Servo lag
    deltaServo = deltaServo + (delta_des - deltaServo)*(dt/servoTau);
    deltaServo = max(min(deltaServo, deltaMax), -deltaMax);
    deltaCmd(k) = deltaServo;

    torqueCtrl(k) = K_ctrl * deltaServo;

    %        Roll dynamics
    omega_dot = (torqueCtrl(k) + T_cant - D_roll*omega(k-1)) / Ixx;
    omega(k)  = omega(k-1) + omega_dot*dt;
end

%% 8. Plots
t_view = [0, 11];

figure('Name','Roll-Rate Tracking: 0     2     0');
subplot(4,1,1)
plot(time, omega, 'b', 'LineWidth', 1.6); hold on
plot(time, omega_ref, 'k--', 'LineWidth', 1.2);
xline(t_step_up,'k:');   xline(t_step_down,'k:'); yline(0,'k:');
hold off; grid on; ylabel('\omega (rad/s)')
title('Roll Rate (blue) vs Reference (black dashed)'); xlim(t_view)

subplot(4,1,2)
plot(time, torqueCtrl, 'r', 'LineWidth', 1.3); grid on
ylabel('Torque (N m)'); title('Actual Control Torque'); xlim(t_view)

subplot(4,1,3)
```

```
plot(time, rad2deg(deltaCmd), 'g', 'LineWidth', 1.3); grid on
ylabel('Deflection (deg)'); title('Surface Deflection'); xlim(t_view)

subplot(4,1,4)
plot(time, q_hist/1000, 'k', 'LineWidth', 1.3); grid on
xlabel('Time (s)'); ylabel('q (kPa)'); title('Dynamic Pressure'); xlim(
    t_view)

fprintf('Final roll rate at %.1f s = %.4f rad/s\n', t_view(2), ...
        omega(find(time>=t_view(2),1,'first')));
```

Listing 2: Reference-tracking roll controller ($0 \to 2 \to 0$ rad/s).

A.3 Angle Step Controller (Two +90° Rotations in 11 s)

```
%% roll_controller_angle_two_90s_11s.m
% Command timing (11 s total): 0       +90   at t=5.0 s, then +180   at t
    =8.5 s

clear; clc; close all;

%% 1) Geometry & constants
S_fin   = 22.35 * 0.0254^2;                 % 0.01441 m   (full fin area)
S_ctrl  = 1.8   * 0.0254^2;                 % 0.001161 m   (control
    surface area)
n       = 3;
c       = 0.1358;                           % m (    ctrl-surface chord)
d_fin   = 0.04445;                          % m (cant arm for fixed fin)
d_ctrl  = ((3.5 + 0.8)/2 + 2) * 0.0254;     % 0.10541 m (4.15 in arm)
liftSlope = 0.0573;                         % 1/rad
deltaCant = deg2rad(1.5);
deltaMax  = deg2rad(45);                    %   45

Ixx     = 1.375e-5;                         % kg m
```

```
D_roll   = 1e-5;                                % N  m  s/rad
dt       = 0.01;                                % s (100 Hz)
servoTau= 0.02;                                 % s (servo lag)
deadBand = deg2rad(0.5);                        %  0 .5   hinge friction
lowKctrlThresh = 1e-8;                          % low-q guard

%% 2) Inner PI-D gains
Kp     = 0.0042;                                % N m per (rad/s)
Ki     = 0.0070;                                % N m per rad
Kd     = 0.000025;                              % N m per (rad/s^2)
tau_d = 0.04;                                   % s (derivative LPF)

%% 3) Outer angle loop     rate command
% omega_ref = sat( K_angle * (phi_ref - phi),   rate_limit   )
K_angle    = 1.5;                               % 1/s (outer-loop gain)
rate_limit = 2.0;                               % rad/s cap

%% 4) Two 90   steps in angle reference (0 -> +90   , then +180   ) within
   11 s
t_step1 = 5.0;                                  % first 90   at 5.0 s
t_step2 = 8.5;                                  % second 90   at 8.5 s
T_total = 11.0;                                 % total simulation duration
   (s)

%% 5) Load flight profile ( f t m , ft/ s m /s) and build 0  11   s timeline
altData = readmatrix('Altitude vs time.csv');
velData = readmatrix('Velocity vs time.csv');
t_alt = altData(:,1);   alt_m  = altData(:,2) * 0.3048;
t_vel = velData(:,1);   vel_ms = velData(:,2) * 0.3048;

time    = 0:dt:T_total;  N = numel(time);
alt_now = interp1(t_alt, alt_m,  time, 'linear', 'extrap');
vel_now = interp1(t_vel, vel_ms, time, 'linear', 'extrap');
```

```
49
50 % Angle reference vector (radians)     used internally only (not plotted)
51 phi_ref = zeros(1,N);
52 phi_ref(time >= t_step1 & time < t_step2) =  pi/2;   % +90
53 phi_ref(time >= t_step2)                  =  pi;     % +180
54
55 %% 6) ISA constants
56 T0=288.15; rho0=1.225; L=0.0065; g0=9.80665; R=287.058;
57
58 %% 7) Pre-allocate state & logs
59 phi = zeros(1,N);                              % roll angle (rad)
60 omega = zeros(1,N);                            % roll rate (rad/s)
61 omega_ref = zeros(1,N);                        % commanded roll rate (not
       plotted)
62 deltaCmd = zeros(1,N);                         % servo deflection (rad)
63 torqueCtrl = zeros(1,N);                       % control torque (N m)
64 q_hist = zeros(1,N);                           % dynamic pressure
65 integrator = 0; deltaServo = 0; prevErr = 0; prevDf = 0;
66
67 %% 8) Main loop
68 for k = 2:N
69     %       Atmosphere & dynamic pressure
70     T   = T0 - L*alt_now(k);
71     rho = (T>0)*rho0*(T/T0)^(g0/(R*L)-1) + (T<=0)*rho0*exp(-alt_now(k)
          /7000);
72     v   = max(vel_now(k), 0);
73     q   = 0.5*rho*v^2;   q_hist(k) = q;
74
75     %       Aerodynamic torque gains
76     K_cant = n*q*S_fin  * c * liftSlope * d_fin;   % N m per rad (cant)
77     K_ctrl = n*q*S_ctrl * c * liftSlope * d_ctrl;  % N m per rad (servo
          surfaces)
78     T_cant = K_cant * deltaCant;                   % fixed bias torque
```

```
    Tmax   = K_ctrl * deltaMax;                    % current authority

    %        Outer angle loop     rate command (with saturation)
    e_angle   = phi_ref(k) - phi(k-1);
    omega_ref(k) = max(min(K_angle * e_angle, rate_limit), -rate_limit);

    %        Low-q guard
    if K_ctrl < lowKctrlThresh
        integrator = 0.98 * integrator;            % bleed integrator
        deltaServo = 0; deltaCmd(k) = 0; torqueCtrl(k) = 0;
        % Free response with cant only
        omega_dot = (T_cant - D_roll*omega(k-1)) / Ixx;
        omega(k)  = omega(k-1) + omega_dot*dt;
        phi(k)    = phi(k-1) + omega(k)*dt;        % integrate angle
        prevErr   = omega_ref(k) - omega(k-1);     % keep derivative
           stable
        continue
    end

    %        Inner PI-D error & filtered derivative (rate loop)
    err   = omega_ref(k) - omega(k-1);
    d_raw = (err - prevErr)/dt;
    d_f   = prevDf + (dt/(tau_d+dt))*(d_raw - prevDf);
    prevErr = err;  prevDf = d_f;

    %        PI-D (torque-domain) + feed-forward
    %     Feed-forward cancels cant AND adds D_roll*omega_ref for steady
       rate.
    integrator = integrator + Ki*err*dt;
    integLim   = 1.1*Tmax;                         % clamp to authority
    integrator = min(max(integrator, -integLim), integLim);

    u_pid = Kp*err + integrator + Kd*d_f;          % N  m
```

```
    u_ff  = -T_cant + D_roll*omega_ref(k);          % N m
    u_tot = u_ff + u_pid;

    %       Saturation & back-calculation anti-windup
    u_sat = max(min(u_tot, Tmax), -Tmax);
    integrator = integrator + (u_sat - u_tot)*0.4;

    %       Desired surface deflection (deadband + limits)
    delta_des = u_sat / K_ctrl;
    if abs(delta_des) < deadBand
        delta_des = 0;
    else
        delta_des = delta_des - sign(delta_des)*deadBand;
    end
    delta_des = max(min(delta_des, deltaMax), -deltaMax);

    %       Servo lag
    deltaServo = deltaServo + (delta_des - deltaServo)*(dt/servoTau);
    deltaServo = max(min(deltaServo, deltaMax), -deltaMax);
    deltaCmd(k) = deltaServo;

    %       Apply torque & integrate dynamics
    torqueCtrl(k) = K_ctrl * deltaServo;
    omega_dot = (torqueCtrl(k) + T_cant - D_roll*omega(k-1)) / Ixx;
    omega(k)  = omega(k-1) + omega_dot*dt;
    phi(k)    = phi(k-1) + omega(k)*dt;            % integrate angle
end

%% 9) Plots (0  11  s window)     no reference curves drawn
t_view = [0, 11];

figure('Name','Angle Tracking: two +90  rotations (11 s)');
subplot(5,1,1)
```

```
plot(time, rad2deg(phi), 'b', 'LineWidth', 1.6);
grid on; ylabel('\phi (deg)')
title('Angle'); xlim(t_view)

subplot(5,1,2)
plot(time, omega, 'b', 'LineWidth', 1.4);
grid on; ylabel('\omega (rad/s)'); title('Roll Rate'); xlim(t_view)

subplot(5,1,3)
plot(time, torqueCtrl, 'r', 'LineWidth', 1.3);
grid on; ylabel('Torque (N m)'); title('Control Torque'); xlim(t_view)

subplot(5,1,4)
plot(time, rad2deg(deltaCmd), 'g', 'LineWidth', 1.3);
grid on; ylabel('Deflection (deg)'); title('Surface Deflection'); xlim(
    t_view)

subplot(5,1,5)
plot(time, q_hist/1000, 'k', 'LineWidth', 1.3);
grid on; xlabel('Time (s)'); ylabel('q (kPa)'); title('Dynamic Pressure');
     xlim(t_view)

idxEnd = find(time>=T_total,1,'first');
fprintf('Final angle = %.1f deg; final roll rate = %.3f rad/s\n', ...
        rad2deg(phi(idxEnd)), omega(idxEnd));
```

Listing 3: Angle-tracking roll controller (two sequential 90° rotations)