Department of Mechanical Engineering
Henry Benedictus
ME 486C

To: Professor David Willy & TA Gray Becker

From: Henry Benedictus

Date: Saturday, January 31st, 2026

Re: ME 486C Individual Analytical Analysis

Introduction:

To induce a spin for the Active Rocket Control (ARC), the rocket will have 4 fins placed in the aft section of the rocket that will adjust its angle of attack (AoA). The force applied to the rocket fins will create a moment that will cause the rocket to rotate about its z-axis. The rocket will have to continuously adjust the fins to maintain a roll of 100 RPMs, with the change in altitude and pressure. The individual analysis is to create a MATLAB code that will help simulate the change in AoA during the change of variables during the flight. Being able to analyze the rotation of the rocket during flight can help improve any last-minute design decisions, like the area of the fin.

Assumptions:

The assumptions made for the assignment are based on the current measurements of the ARC design. Table 1 has the measurements of the fins based on the values from Figure 4. Table 2 holds the measured values of the rocket body and nose cone used to find the total moment of inertia (I_{Total}).

Distance Between Fin Root and Fin Tip (x_t or x_r) [1]	Fin Root Cord (c_r) [1]	Fin Tip Cord (c_t) [1]	Exposed Fin Semispan Exposed at Root (s) [1]
0.05162m	0.1524m	0.0762m	0.0613m

Table 1: Fin Measurements

$Mass\ of\ Nose\ Cone\ (m_{cone}) = 0.1333\ kg$	$Mass\ of\ Rocket\ (m_{body}) = 0.78kg$
$Radius\ of\ Nose\ Cone = 0.039624m$	$Radius\ of\ Rocket = 0.039624m$

Table 2: Mass and Radius of Rocket

$Time\ to\ Start\ Roll\ =\ 6\ Sec$	$RPM\ Target\ =\ 100RPM$	$RPM\ Tolerance\ =\ \pm\ 0.25\ RPM$
$Hold\ Duration\ =\ 2\ Sec$	$Starting\ AoA\ =\ 0\ Degrees$	$AoA\ Tolerance\ =\ \pm\ 15$

Table 3: Control Settings

Equations:

Normal Force

The Normal Force (F_N) is the force that acts on the center of pressure of an object due to the external fluid acting upon the object [2]. It is dependent on the density and its relative velocity at each point during the launch. Using RockSim, and a rocket file similar to the one that is going to be used (similar weights and rocket motor), a excel file was imported with the corresponding velocities, and densities at each time increment during the launch.

$$N_F = F_\alpha = qA_{Fin}\alpha(C_{N\alpha}) \quad [2] \qquad (1)$$

$$q = \frac{1}{2}\rho V^2 \quad [2] \qquad (2)$$

$$A = \frac{1}{2}(a + b)h \quad [3] \qquad (3)$$

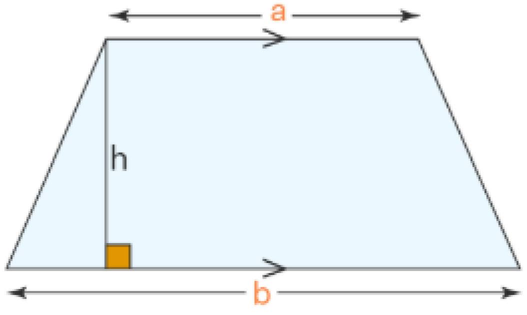


Figure 1: Area of a Trapezoid [3]

Using the Normal Force, the moment of the rocket (M_α) can be found by using equation 4 based on Figure 2. RockSim accounts for the drag of the fins during launch, and it also assumes the rocket is a smooth body and its fins are attached perfectly straight to the body. These two assumptions account for the rocket having no spin during the launch, which can be seen within the Excel file.

$$M_\alpha = 4 * F_N * \bar{y} \quad [2] \qquad (4)$$

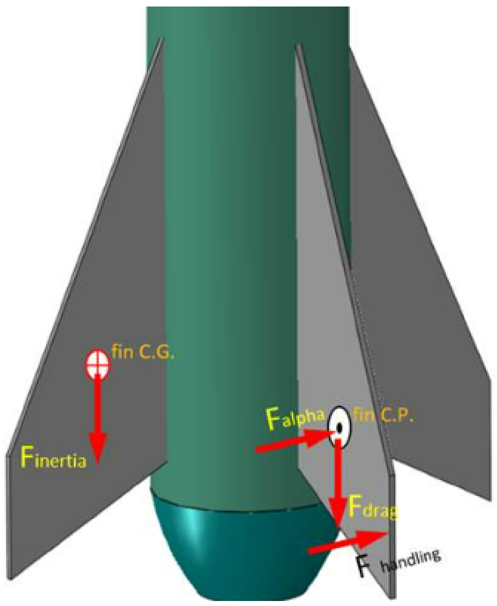


Figure 2: Forces Experienced by Fins [2]

Normal Force Coefficient

The Normal Force Coefficient ($C_{N\alpha}$) is the coefficient of force perpendicular to the object, dependent on the air flow along the object [1]. $C_{N\alpha}$ is notated by equation 5 [2]. The locations of each variable can be found within Table 1.

$$(C_{N\alpha}) = (1 + \frac{fR}{s+R})[\frac{4N(\frac{s}{d})^2}{1+\sqrt{1+(\frac{2l}{c_r+c_t})^2}}] \quad [2] \quad (5)$$

$$l = \frac{s}{\cos\theta} \quad [2] \quad (6)$$

$$\theta = arctan[\frac{1}{s}(x_r + \frac{1}{2}(c_t - c_r))] \quad [2] \quad (7)$$

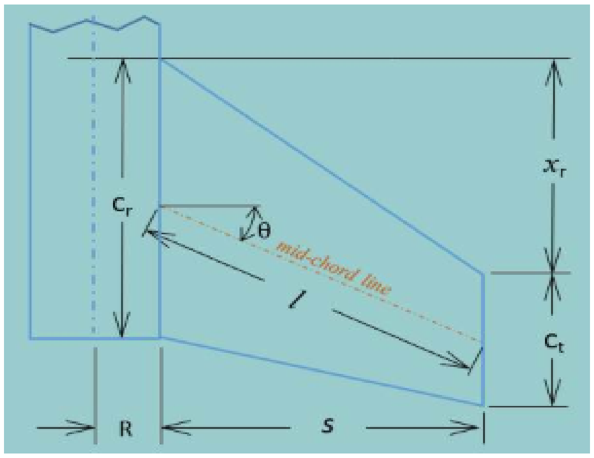


Figure 3: Normal Force Coefficient Locations [2]

Using the Disp function within MATLAB, the length (l) is $0.17m$Theta (θ) 1.21 degrees, and Normal Force Coefficient $(C_{N\alpha})$ 4.69.

Center of Pressure of Fins
A trapezoid shape was determined to be the best shape to be implemented since it is structurally robust and flutter resistant [1]. The group then designed the fin to be used based on the trapezoid fin used on the Hi-Tech rocket that was used for prototypes 1 and 2.

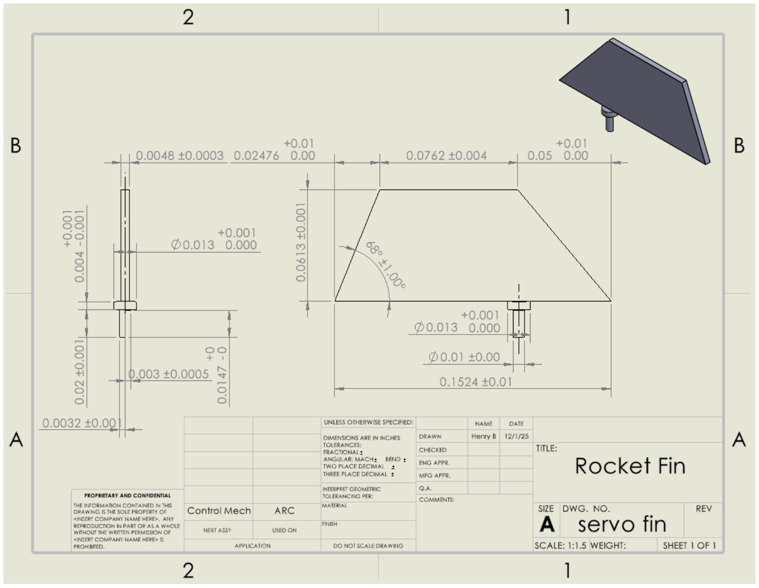


Figure 4: Rocket Fin Drawing (m)

The locations of all the forces applied to the fins act on the center of pressure. Equations 8 and 9 are used to find the C.P. along the x and y coordinates of the fins. The location of the fins is dependent on the distance between the Fin Root and Fin Tip $(x_t \; or \; x_r)$, Fin Root Cord (c_t), Fin Tip Cord (c_t), and the Exposed Fin Semispan (s). Figure 4 is used to find the corresponding dimensions used to find the center of pressure.

$$\bar{x} = \frac{x_t}{3}\left(\frac{c_r+2c_t}{c_r+c_t}\right) + \frac{1}{6}\left[c_r + c_t - \left(\frac{c_r c_t}{c_r+c_t}\right)\right] \qquad [2] \qquad (8)$$

$$\bar{y} = \frac{s}{3}\left(\frac{c_r+2c_t}{c_r+c_t}\right) \qquad [2] \qquad (9)$$

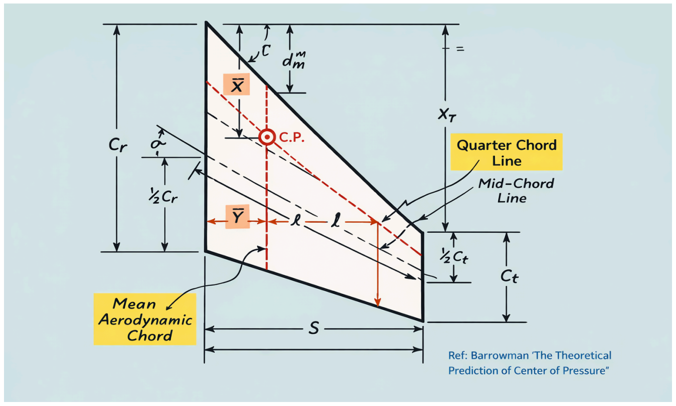


Figure 5: Location of Fin Center of Pressure [2]

Using the "Disp" function within MATLAB, the center of pressure locations are shown to be $\bar{x} = 0.052m$ and $\bar{y} = 0.027m$.

Roll Rate:
The roll rate is dependent on the Total Moment of Inertia (I_{Total}) of the rocket, which is comprised of the Moment of Inertia of the Cone (I_{Cone}) and of the body itself (I_{Body}). I_{Body} uses only one radius because the thickness of the rocket is small enough that it doesn't impact the results. It is assumed that the maximum radius of the cone is the same as the outside radius of the body. The mass of the rocket is also assumed to be without the weight of the propellant since it has already been burnt before the flight started.

$$I_{Total} = I_{Cone} + I_{Body} \qquad (10)$$

$$I_{Body} = m_{body} r^2 \qquad [4] \qquad (11)$$

$$I_{Nose} = \frac{3}{10} m_{nose} r^2 \qquad [5] \qquad (12)$$

Using the "Disp" function within MATLAB, $I_{Body} = 0.0012\, kg * m^2$, $I_{Nose} = 1.87 * 10^{-5} kg * m^2$, and $I_{Total} = 0.00124\, kg * m^2$.

Using the moment of the rocket, and the moment of inertia, the angular acceleration and RPMs can be found using equations 13 and 14.

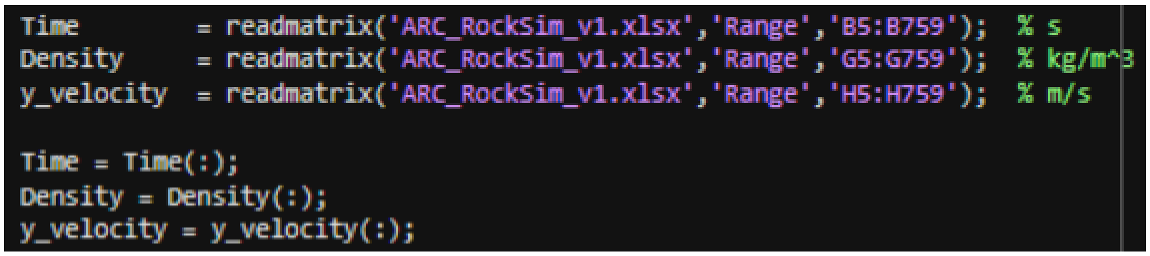
$$\omega = \frac{M_\alpha}{I_{Total}} \qquad [6] \qquad (13)$$

$$n = \frac{\omega}{2\pi} \qquad [6] \qquad (14)$$

MATLAB Integration:

Imported Data:

To import the data from the Excel sheet that was obtained through RockSim, the “Readmatrix” function is used, which calls the file that needs to be opened and the range of the cells that contain the data. The data used were the density and the y-velocity of the rocket with its associated time. Each piece of data is then forced into a column vector using the command (:).

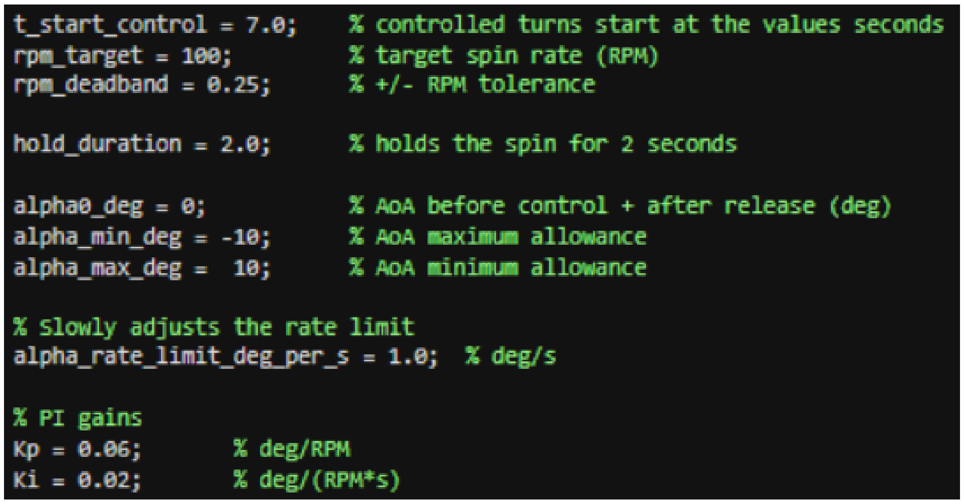


```
Time       = readmatrix('ARC_RockSim_v1.xlsx','Range','B5:B759');  % s
Density    = readmatrix('ARC_RockSim_v1.xlsx','Range','G5:G759');  % kg/m^3
y_velocity = readmatrix('ARC_RockSim_v1.xlsx','Range','H5:H759');  % m/s

Time = Time(:);
Density = Density(:);
y_velocity = y_velocity(:);
```

Figure 6: Imported Data Commands

Control Settings:

To easily adjust the settings of the future control functions, the control settings are placed towards the top of the script. The first setting is when the roll will start, which, for the test conducted for the assignment, is 7 seconds. It has a threshold of 100 RPM, a tolerance of $\pm$ 0.25 RPMs, and a duration to hold the 100 RPM spin for 2 seconds, which are customer requirements. An established initial AoA is 0 degrees, and the maximum/minimum is $\pm$ 10 degrees. The change in degrees is limited to 1 degree per second. A Kp and Ki setting is also added to help the simulation to see how hard it should change the AoA when wrong, and then slowly adjust to get to the exact point when needed, respectively [7].

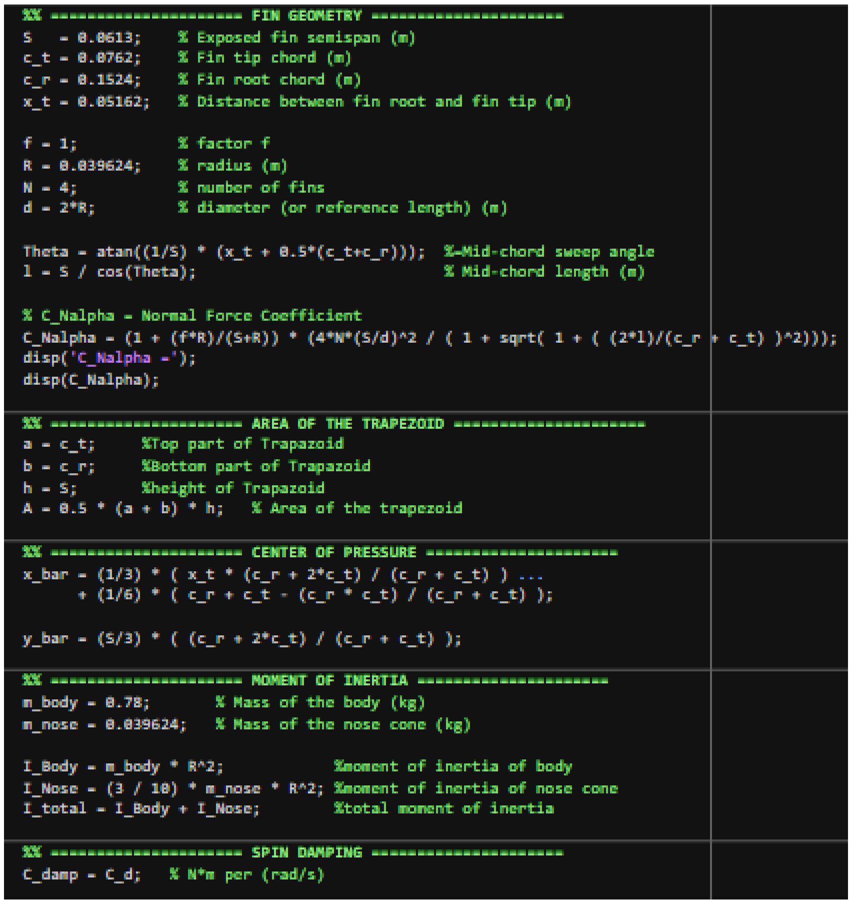


```
t_start_control = 7.0;    % controlled turns start at the values seconds
rpm_target = 100;         % target spin rate (RPM)
rpm_deadband = 0.25;      % +/- RPM tolerance

hold_duration = 2.0;      % holds the spin for 2 seconds

alpha0_deg = 0;           % AoA before control + after release (deg)
alpha_min_deg = -10;      % AoA maximum allowance
alpha_max_deg =  10;      % AoA minimum allowance

% Slowly adjusts the rate limit
alpha_rate_limit_deg_per_s = 1.0;  % deg/s

% PI gains
Kp = 0.06;        % deg/RPM
Ki = 0.02;        % deg/(RPM*s)
```

Figure 7: Control Settings

Constant Equation Implementation:

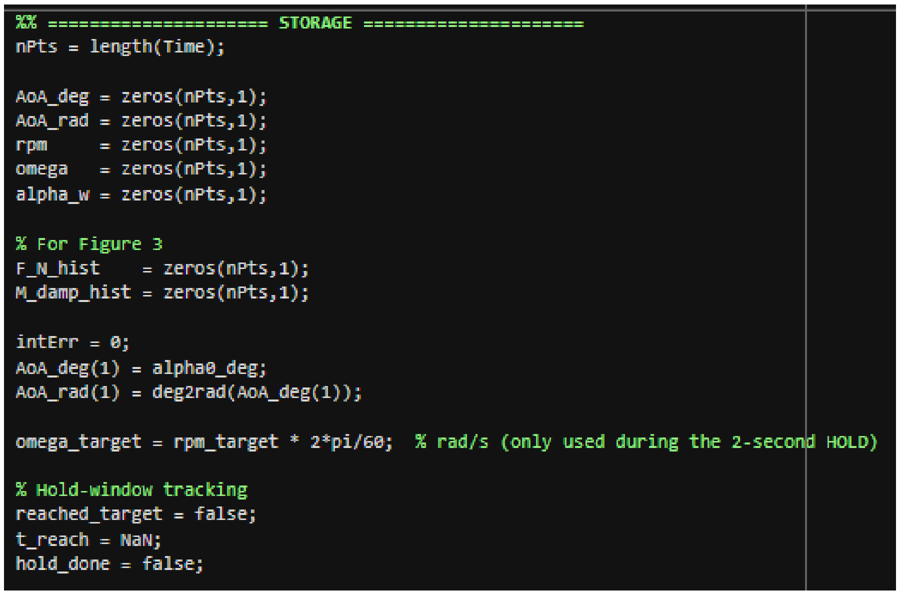
Equations 1 - 12 are entered into the script, using corresponding measured values from Tables 1 and 2 A resistive rotation coefficient (C_d) is used to help simulate a resistive torque applied to the script, which will later be used in a resistant moment. It helps to simulate aerodynamic skin friction, fin profile drag, energy lost to turbulence, etc.



```
%% ==================== FIN GEOMETRY ====================
S   = 0.0613;    % Exposed fin semispan (m)
c_t = 0.0762;    % Fin tip chord (m)
c_r = 0.1524;    % Fin root chord (m)
x_t = 0.05162;   % Distance between fin root and fin tip (m)

f = 1;           % factor f
R = 0.039624;    % radius (m)
N = 4;           % number of fins
d = 2*R;         % diameter (or reference length) (m)

Theta = atan((1/S) * (x_t + 0.5*(c_t+c_r)));  %-Mid-chord sweep angle
l = S / cos(Theta);                           % Mid-chord length (m)

% C_Nalpha = Normal Force Coefficient
C_Nalpha = (1 + (f*R)/(S+R)) * (4*N*(S/d)^2 / ( 1 + sqrt( 1 + ( (2*l)/(c_r + c_t) )^2)));
disp('C_Nalpha =');
disp(C_Nalpha);

%% ==================== AREA OF THE TRAPEZOID ====================
a = c_t;     %Top part of Trapazoid
b = c_r;     %Bottom part of Trapazoid
h = S;       %height of Trapazoid
A = 0.5 * (a + b) * h;   % Area of the trapezoid

%% ==================== CENTER OF PRESSURE ====================
x_bar = (1/3) * ( x_t * (c_r + 2*c_t) / (c_r + c_t) ) ...
      + (1/6) * ( c_r + c_t - (c_r * c_t) / (c_r + c_t) );

y_bar = (S/3) * ( (c_r + 2*c_t) / (c_r + c_t) );

%% ==================== MOMENT OF INERTIA ====================
m_body = 0.78;       % Mass of the body (kg)
m_nose = 0.039624;   % Mass of the nose cone (kg)

I_Body = m_body * R^2;            %moment of inertia of body
I_Nose = (3 / 10) * m_nose * R^2; %moment of inertia of nose cone
I_total = I_Body + I_Nose;        %total moment of inertia

%% ==================== SPIN DAMPING ====================
C_damp = C_d;   % N*m per (rad/s)
```

Figure 8: Constant Equation Implementations

Applied Storage:
The length of time from the Excel file is identified using the length function. The degrees, AoA, and RPMs, angular velocity (ω), and angular acceleration (α_ω), normal aerodynamic force, and damping torque histories are zeroed out with the associated length of time. Any integral error state for the PI controller is equal to zero as well. The AoA is converted from radians to degrees. The reached target is set as false until the RPM enters the required threshold, which becomes true once the threshold has been reached, which implements the hold.



```
%% ==================== STORAGE ====================
nPts = length(Time);

AoA_deg = zeros(nPts,1);
AoA_rad = zeros(nPts,1);
rpm     = zeros(nPts,1);
omega   = zeros(nPts,1);
alpha_w = zeros(nPts,1);

% For Figure 3
F_N_hist    = zeros(nPts,1);
M_damp_hist = zeros(nPts,1);

intErr = 0;
AoA_deg(1) = alpha0_deg;
AoA_rad(1) = deg2rad(AoA_deg(1));

omega_target = rpm_target * 2*pi/60;  % rad/s (only used during the 2-second HOLD)

% Hold-window tracking
reached_target = false;
t_reach = NaN;
hold_done = false;
```

Figure 9: Applied Storage

Main Loop:
A time-lapse calculation is designated as dt = Time(i) - time(i-1), and if dt is greater than 0, dt = eps, which prevents any positive number from being divided by zero. This helps to protect against duplicate timestamps and Excel rounding errors. The angular velocity (ω) is converted to RPM. RPM now is what is now what is being used throughout the script.

The dynamic pressure (q), which is shown within equation 2, is written so that the associated density and y-velocity are written with their corresponding time. Using equation 13, the damping torque is calculated with its corresponding time.

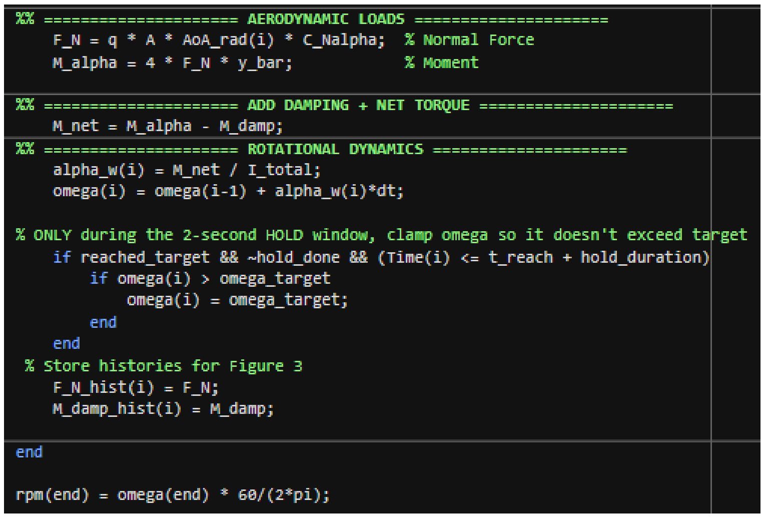
$$M_{damp} = C_{damp} * \omega \qquad [8] \qquad (15)$$

Once the parameters are set, the script launch simulation can begin, by using an "if" statement to start the launch until it's done, by utilizing the time information from the Excel sheet. Another "if" statement is used to set the RPM when the set goal is reached, by setting the reach target and the absolute value $RPM_{now} - RPM_{Target}$. Using the combination of equations 4 and 15, the moment is found. An "if" loop is used to keep the RPM constant once it is set.

A track mode for the PI controller is used to reduce any calculation error, and uses the Kp and Ki values (established at the beginning of the script) to ensure that the target is reached. If $RPM_{Now} > RPM_{Target}$ it prevents any adding torque by setting the AoA degree equal to AoA 0 [7]. The logic is then released once the desired RPM is reached, and the hold RPM command begins. AoA is designated with the maximum and minimum set at the start of the script to prevent fin stall, fin shear, and unrealistic commands.

The purpose of the main storage loop is to figure out how fast ARC is spinning, decide to induce a spin, hold or let go of the spin, compute the value of the AoA, apply physical limits, and store the AoA for other updates.

Applied Loads:
Equation 13 is used to find the moment to later be used to find the net moment found by subtracting the fin moment(M_{α}) from the damping moment (M_{Damp}).



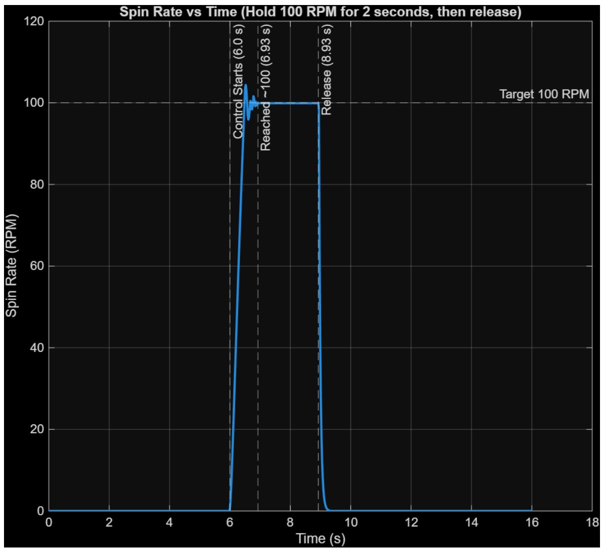
```
%% ==================== AERODYNAMIC LOADS ====================
    F_N = q * A * AoA_rad(i) * C_Nalpha;  % Normal Force
    M_alpha = 4 * F_N * y_bar;            % Moment

%% ==================== ADD DAMPING + NET TORQUE ====================
    M_net = M_alpha - M_damp;
%% ==================== ROTATIONAL DYNAMICS ====================
    alpha_w(i) = M_net / I_total;
    omega(i) = omega(i-1) + alpha_w(i)*dt;

% ONLY during the 2-second HOLD window, clamp omega so it doesn't exceed target
    if reached_target && ~hold_done && (Time(i) <= t_reach + hold_duration)
        if omega(i) > omega_target
            omega(i) = omega_target;
        end
    end
 % Store histories for Figure 3
    F_N_hist(i) = F_N;
    M_damp_hist(i) = M_damp;

end

rpm(end) = omega(end) * 60/(2*pi);
```

Figure 10: Applied Loads

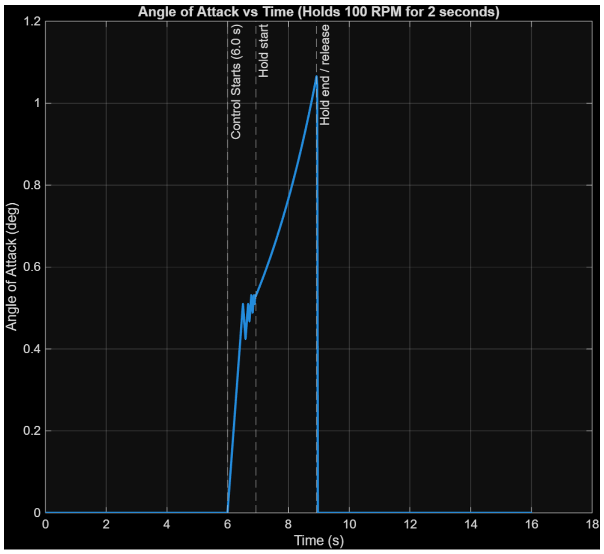
Rational Dynamics:
Once the target RPM has been reached, the angular velocity is then set during the hold window (2 seconds). The clamp prevents the pin rate from exceeding the target during the hold. It is not a physical torque; it is a numerical control safety guard that prevents the overshoot. The moment due to the fins and resistance is logged for later plots.

Plots:
Plot 1 shows the RMPs throughout the duration of the flight. It is set to graph when the controlled roll starts, when it keeps 100 RPMs, and when it ends. Plot 2 shows the change in degrees of the AoA during the start of the roll, hold, and end. Same for the moment forces throughout the flight due to the change in AoA in Plot 3. Plots 4, 5, 6, and 7 show the change in altitude, velocity, density, and AoA in radians with respect to time.

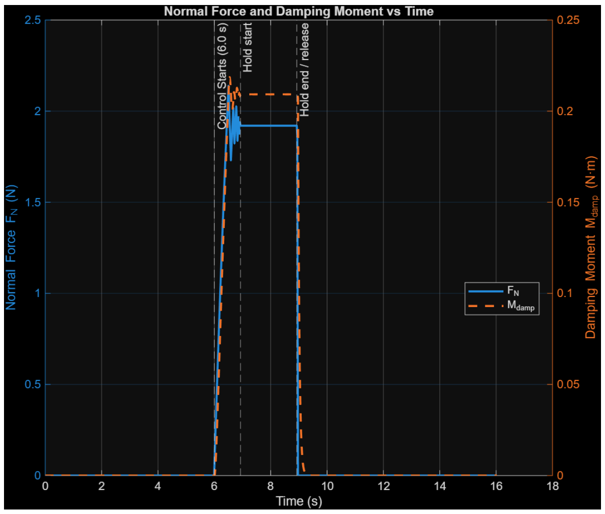
Result Analysis:
Using the values within Tables 1 and 2, it is shown that at the current design, ARC is able to induce a spin up to 100 RPM.



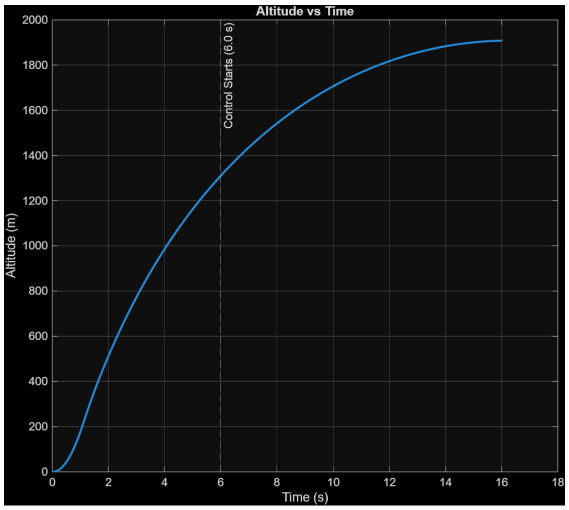
Plot 1: Spin Rate vs. Time



Plot 2: Angle of Attack Vs Time (Degrees)

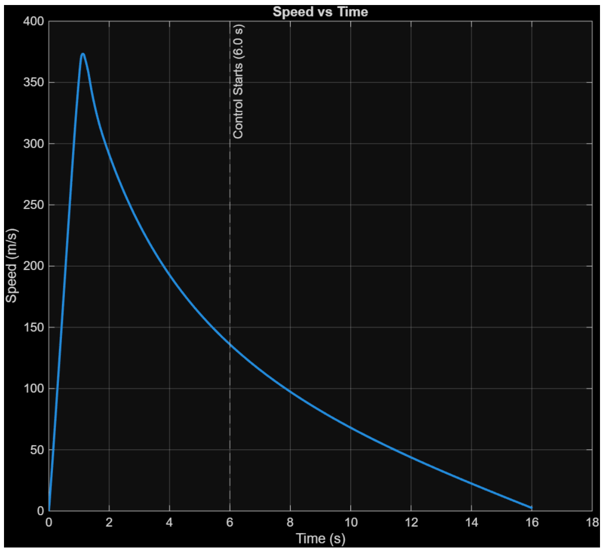


Plot 3: Normal Force and Damping Moment vs Time

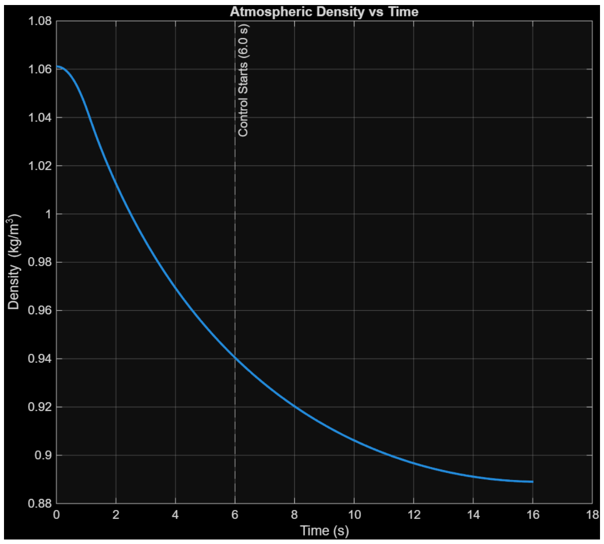


\

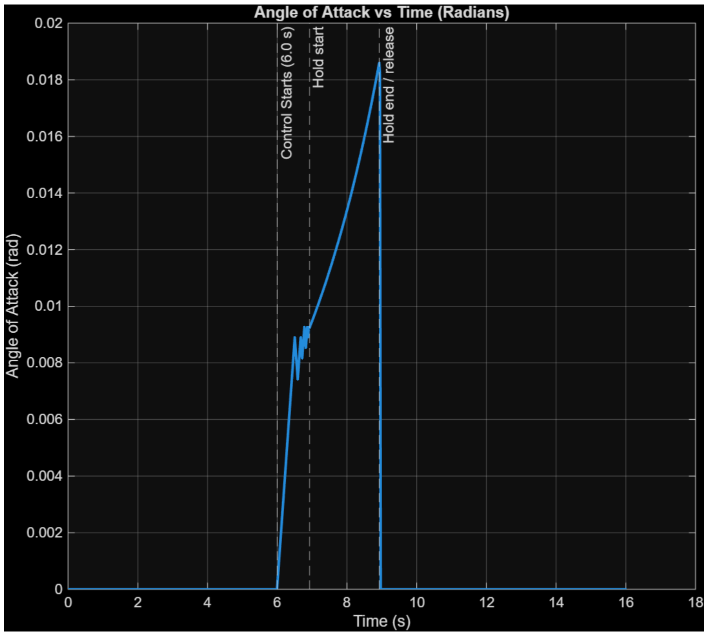
Plot 4: Altitude vs time



Plot 5: Speed vs Time



Plot 6: Atmospheric Density vs Time



Plot 7: Angle of Attack vs Time (Radians)

However, if the damping coefficient is increased, it will increase the dampening moment, which acts against the induced momentum, which reduces the RPM.

Conclusion and Future Work

The data will become more accurate of the future launch once the rocket is completely weighted and set up properly within RockSim. From there, further simulations can be run to see if any design changes need to be implemented. (Increase in fin area, an increase/decrease in weight, etc.) During a test launch at the end of February, the prototype rocket will have fins set with an AoA of 2 degrees. The measured values will be entered into the simulation to measure the accuracy of the simulation. Further FEA analysis will also be needed to measure a more accurate damping coefficient.

References:

[1] R. Nakka, “Introduction to Rocket Design: Fins,” Richard Nakka’s Experimental Rocketry Site, https://www.nakka-rocketry.net/RD_fin.htm (accessed Dec. 23, 2025).

[2] J. Barrowman, “James S. Barrowman (#6883) Judith A. ...,” The Practical Calculations of the Aerodynamic Characteristics of Slender Finned Vehicles, https://www.nakka-rocketry.net/articles/Barrowman.NARAM-8.pdf (accessed Jan. 22, 2026).

[3] “Area of trapezoid - formula: How to find the area of a trapezoid?,” Cuemath, https://www.cuemath.com/measurement/area-of-trapezoid/ (accessed Jan. 23, 2026).

[4] Admin, “Moment of inertia of a hollow cylinder,” BYJUS, https://byjus.com/jee/moment-of-inertia-of-a-hollow-cylinder/?utm_source (accessed Jan. 23, 2026).

[5] Admin, “Moment of inertia of cone,” BYJUS, https://byjus.com/jee/moment-of-inertia-of-a-cone/ (accessed Jan. 23, 2026).

[6] Alechner, “Refresher on the basics of angular acceleration and moment of inertia,” R+W Coupling Technology,https://www.rw-america.com/refresher-on-the-basics-of-angular-acceleration-and-moment-of-inertia/ (accessed Jan. 23, 2026)

[7] “Dynamics and control,” Proportional Integral (PI) Control, https://apmonitor.com/pdc/index.php/Main/ProportionalIntegralControl (accessed Jan. 29, 2026).

[8] R. Kilgore and J. Adock, *NASA TECHNICAL MEMORANDUM X-2555*. NASA.

[9] [1] S. Sandaram, ECE 380: Control Systems, chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://engineering.purdue.edu/~sundara2/misc/ece380_notes.pdf?utm_source=chatgpt.com (accessed Jan. 29, 2026).

[10] T. Milligan, “Apogeerockets,” Apogee Peak of Flight Newsletter, https://www.apogeerockets.com/education/downloads/Newsletter45.pdf (accessed Jan. 28, 2026).

Code:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Change in Angular Acceleration with Change in Angle of Attack

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;

clear;

format long;

%% ===================== IMPORT DATA =====================

Time       = readmatrix('ARC_RockSim_v1.xlsx','Range','B5:B759');  % s

Density     = readmatrix('ARC_RockSim_v1.xlsx','Range','G5:G759');  % kg/m^3

y_velocity  = readmatrix('ARC_RockSim_v1.xlsx','Range','H5:H759');  % m/s

%C_d  = readmatrix('ARC_RockSim_v1.xlsx','Range','J5:J759');

Time = Time(:);

Density = Density(:);

y_velocity = y_velocity(:);

%C_d = C_d(:);

C_d = 0.02;

%% ===================== CONTROL SETTINGS =====================

t_start_control = 6;    % controlled turns start at the values seconds

rpm_target = 100;         % target spin rate (RPM)

rpm_deadband = 0.25;      % +/- RPM tolerance

hold_duration = 2.0;      % holds the spin for 2 seconds

alpha0_deg = 0;           % AoA before control + after release (deg)

alpha_min_deg = -15;      % AoA maximum allowance

alpha_max_deg =  15;      % AoA minimum allowance
```

```
% Slowly adjusts the rate limit

alpha_rate_limit_deg_per_s = 1.0;  % deg/s

% PI gains

Kp = 0.06;       % deg/RPM

Ki = 0.02;       % deg/(RPM*s)

%% ===================== FIN GEOMETRY =====================

S   = 0.0613;    % Exposed fin semispan (m)

c_t = 0.0762;    % Fin tip chord (m)

c_r = 0.1524;    % Fin root chord (m)

x_t = 0.05162;   % Distance between fin root and fin tip (m)

f = 1;           % factor f

R = 0.039624;    % radius (m)

N = 4;           % number of fins

d = 2*R;         % diameter (or reference length) (m)

Theta = atan((1/S) * (x_t + 0.5*(c_t+c_r)));  %=Mid-chord sweep angle

l = S / cos(Theta);                           % Mid-chord length (m)

% C_Nalpha = Normal Force Coefficient

C_Nalpha = (1 + (f*R)/(S+R)) * (4*N*(S/d)^2 / ( 1 + sqrt( 1 + ( (2*l)/(c_r + c_t) )^2)));

disp('C_Nalpha =');

disp(C_Nalpha);

%% ===================== AREA OF THE TRAPEZOID =====================

a = c_t;     %Top part of Trapazoid

b = c_r;     %Bottom part of Trapazoid

h = S;       %height of Trapazoid

A = 0.5 * (a + b) * h;   % Area of the trapezoid

%% ===================== CENTER OF PRESSURE =====================
x_bar = (1/3) * ( x_t * (c_r + 2*c_t) / (c_r + c_t) ) ...
    + (1/6) * ( c_r + c_t - (c_r * c_t) / (c_r + c_t) );
y_bar = (S/3) * ( (c_r + 2*c_t) / (c_r + c_t) );
%% ====================== MOMENT OF INERTIA ======================
m_body = 0.78;       % Mass of the body (kg)
m_nose = 0.039624;   % Mass of the nose cone (kg)
I_Body = m_body * R^2;           %moment of inertia of body
I_Nose = (3 / 10) * m_nose * R^2; %moment of inertia of nose cone
I_total = I_Body + I_Nose;       %total moment of inertia
%% ====================== SPIN DAMPING ======================
C_damp = C_d;   % N*m per (rad/s)
%% ====================== STORAGE ======================
nPts = length(Time);
AoA_deg = zeros(nPts,1);
AoA_rad = zeros(nPts,1);
rpm     = zeros(nPts,1);
omega   = zeros(nPts,1);
alpha_w = zeros(nPts,1);
% For Figure 3
F_N_hist    = zeros(nPts,1);
M_damp_hist = zeros(nPts,1);
intErr = 0;
AoA_deg(1) = alpha0_deg;
AoA_rad(1) = deg2rad(AoA_deg(1));
omega_target = rpm_target * 2*pi/60;  % rad/s (only used during the 2-second HOLD)
% Hold-window tracking
```

```
reached_target = false;

t_reach = NaN;

hold_done = false;

%% ===================== MAIN LOOP =====================

for i = 2:nPts

  dt = Time(i) - Time(i-1);

  if dt <= 0

     dt = eps;

  end

  % Current RPM from previous omega

  rpm(i-1) = omega(i-1) * 60/(2*pi);

  rpm_now = rpm(i-1);

  % Always compute dynamic pressure for this time step

  q = 0.5 * Density(i) * y_velocity(i)^2;

  % Damping moment based on current omega

  M_damp = C_damp * omega(i-1);

  % ---- CONTROL LOGIC: Reach -> Hold for 2s -> Release ----

  if Time(i) >= t_start_control && ~hold_done

     % If we've never reached target and we are within the deadband, start hold timer

     if ~reached_target && abs(rpm_now - rpm_target) <= rpm_deadband

        reached_target = true;

        t_reach = Time(i);

     end

     denom = 4 * q * A * C_Nalpha * y_bar;

     if reached_target && (Time(i) <= t_reach + hold_duration) && abs(denom) > 1e-12

        % ===================== HOLD MODE (ONLY FOR 2 SECONDS)
 =====================
        AoA_des_rad = M_damp / denom;

        AoA_des_deg = rad2deg(AoA_des_rad);

      else

 % ===================== TRACK MODE (PI) UNTIL WE HIT TARGET
   =====================

 % (If we already held for 2 seconds, we "release" by ending control below.)

        err = rpm_target - rpm_now;

        if rpm_now < rpm_target

          intErr = intErr + err*dt;

        end

        intErr = max(-500, min(500, intErr));

        AoA_des_deg = alpha0_deg + Kp*err + Ki*intErr;

 % If above target (before reaching & holding), don't add more positive AoA

        if rpm_now > rpm_target

          AoA_des_deg = min(AoA_des_deg, 0);

        end

      end

 % If we already reached target AND we're past the 2-second hold window, RELEASE control

      if reached_target && (Time(i) > t_reach + hold_duration)

        hold_done = true;

        AoA_des_deg = alpha0_deg;  % release AoA back to baseline (0 deg here)

      end

% Clamp AoA to limits

      AoA_des_deg = max(alpha_min_deg, min(alpha_max_deg, AoA_des_deg));

% Rate limit AoA change

      maxStep = alpha_rate_limit_deg_per_s * dt;

      AoA_deg(i) = AoA_deg(i-1) + max(-maxStep, min(maxStep, AoA_des_deg - AoA_deg(i-1)));
```

```
  else
 % Before control starts OR after release: hold baseline AoA
      AoA_deg(i) = alpha0_deg;
  end
  AoA_rad(i) = deg2rad(AoA_deg(i));
%% ====================== AERODYNAMIC LOADS ======================
  F_N = q * A * AoA_rad(i) * C_Nalpha;  % Normal Force
  M_alpha = 4 * F_N * y_bar;            % Moment
%% ====================== ADD DAMPING + NET TORQUE ======================
  M_net = M_alpha - M_damp;
%% ====================== ROTATIONAL DYNAMICS ======================
  alpha_w(i) = M_net / I_total;
  omega(i) = omega(i-1) + alpha_w(i)*dt;
% ONLY during the 2-second HOLD window, clamp omega so it doesn't exceed target
  if reached_target && ~hold_done && (Time(i) <= t_reach + hold_duration)
      if omega(i) > omega_target
          omega(i) = omega_target;
      end
  end
% Store histories for Figure 3
  F_N_hist(i) = F_N;
  M_damp_hist(i) = M_damp;
end
rpm(end) = omega(end) * 60/(2*pi);
%% ====================== FIGURE 1: RPM ======================
figure;
```

```
plot(Time, rpm, 'LineWidth', 1.8);

grid on;

xlabel('Time (s)');

ylabel('Spin Rate (RPM)');

title('Spin Rate vs Time (Hold 100 RPM for 2 seconds, then release)');

yline(rpm_target,'--','Target 100 RPM');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

% Mark reach + release times if the target was reached

if ~isnan(t_reach)

  xline(t_reach,'--',sprintf('Reached ~100 (%.2f s)', t_reach));

  xline(t_reach + hold_duration,'--',sprintf('Release (%.2f s)', t_reach + hold_duration));

end

%% ===================== FIGURE 2: AoA =====================

figure;

plot(Time, AoA_deg, 'LineWidth', 1.8);

grid on;

xlabel('Time (s)');

ylabel('Angle of Attack (deg)');

title('Angle of Attack vs Time (Holds 100 RPM for 2 seconds)');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

if ~isnan(t_reach)

  xline(t_reach,'--','Hold start');

  xline(t_reach + hold_duration,'--','Hold end / release');

end

%% ===================== FIGURE 3: F_N and M_damp =====================

figure;
```

```
yyaxis left

plot(Time, F_N_hist, 'LineWidth', 1.8);

ylabel('Normal Force F_N (N)');

yyaxis right

plot(Time, M_damp_hist, '--', 'LineWidth', 1.8);

ylabel('Damping Moment M_{damp} (N·m)');

grid on;

xlabel('Time (s)');

title('Normal Force and Damping Moment vs Time');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

if ~isnan(t_reach)

  xline(t_reach,'--','Hold start');

  xline(t_reach + hold_duration,'--','Hold end / release');

end

legend('F_N','M_{damp}','Location','best');

%% ==================== PRINTS ====================

disp('x_bar =');

disp(x_bar);

disp('y_bar =');

disp(y_bar);

disp('I_Total')

disp(I_total)

disp('I_Body')

disp(I_Body)

disp('I_Nose')

disp(I_Nose)
disp('Theta = ')

disp(Theta)

disp('length =')

disp(l)

%% ===================== ALTITUDE CALCULATION =====================

altitude = cumtrapz(Time, y_velocity);   % meters (assuming y_velocity is m/s)

%% ===================== FIGURE 4: ALTITUDE vs TIME =====================

figure;

plot(Time, altitude, 'LineWidth', 1.8);

grid on;

xlabel('Time (s)');

ylabel('Altitude (m)');

title('Altitude vs Time');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

%% ===================== FIGURE 5: SPEED vs TIME =====================

figure;

plot(Time, y_velocity, 'LineWidth', 1.8);

grid on;

xlabel('Time (s)');

ylabel('Speed (m/s)');

title('Speed vs Time');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

%% ===================== FIGURE 6: DENSITY vs TIME =====================

figure;

plot(Time, Density, 'LineWidth', 1.8);

grid on;
```

```
xlabel('Time (s)');

ylabel('Density (kg/m^3)');

title('Atmospheric Density vs Time');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

disp('A =');

disp(A);

%% ===================== FIGURE 7: AoA (Radians) =====================

figure;

plot(Time, AoA_rad, 'LineWidth', 1.8);

grid on;

xlabel('Time (s)');

ylabel('Angle of Attack (rad)');

title('Angle of Attack vs Time (Radians)');

xline(t_start_control,'--',sprintf('Control Starts (%.1f s)', t_start_control));

if ~isnan(t_reach)

  xline(t_reach,'--','Hold start');

  xline(t_reach + hold_duration,'--','Hold end / release');

end
```