Eric Reyes

2 February 2026

Active Rocket Controls

Expanding Control Systems - Flight Computer Simulink Implementation

ME 486C - Section 002

Introduction

The Active Rocket Control (ARC) capstone project focuses on the design and implementation of an actively controlled rocket capable of maintaining roll stability during flight. The system uses a flight computer interfaced with an onboard inertial measurement unit (IMU) to measure angular motion and command fin actuators to regulate the rocket's orientation. Active fin control allows the rocket to remain within a ±2° roll envelope about the z-axis, improving stability beyond what is achievable through passive aerodynamic methods alone.

Previous analytical work established the feasibility of closed-loop roll-rate control using gyroscopic feedback and fin actuation; however, the control logic was primarily expressed through analytical models and MATLAB scripts. To better evaluate the interaction between the plant dynamics, controller logic, and actuator behavior, the control system needed to be represented in a unified simulation environment that allows time-domain analysis of the full closed-loop response.

This work focuses on the self-directed learning and application of MATLAB Simulink to model the ARC rocket's roll-control system. Video tutorials from the MATLAB YouTube channel, along with official MathWorks documentation, were used to learn Simulink's block-diagram modeling approach and simulation tools. Using these resources, previously developed analytical models and MATLAB-based flight computer code from team lead, Henry Benedictus, were translated into a closed-loop Simulink model that provides a clear and interpretable representation of the flight control system.

Skill Development

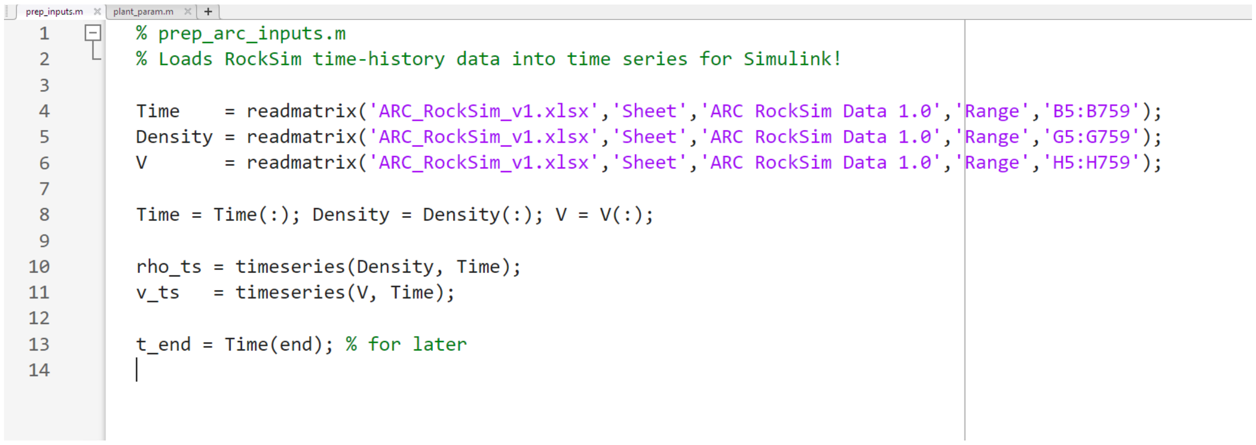
The primary skill developed during this self-learning effort was the ability to implement and validate a time-domain dynamic plant model in MATLAB Simulink, specifically for the roll dynamics of the ARC rocket. While previous work within the capstone established the feasibility of roll stabilization through analytical models and MATLAB scripts, those approaches did not provide a unified environment where plant dynamics, feedback effects, and future controller logic could be evaluated together over time. This gap became more apparent as the Electrical Engineering Division expressed interest in implementing the flight computer using Simulink-based workflows, with the long-term goal of setting logic to an Arduino board, acting as the flight computer. Our lead, Henry Benedictus, had already laid significant groundwork by developing a MATLAB-based flight computer that leveraged RockSim trajectory data, motivating the need to translate that foundation into a model-based simulation framework.

Learning Simulink addressed this need by shifting the focus from isolated calculations to system-level behavior. Rather than simply computing roll rates or moments at discrete points, Simulink forces the designer to think in terms of signal flow, state evolution, and feedback. This directly develops design attributes such as systems integration, dynamic reasoning, and model-based verification. The self-learning process relied on MATLAB YouTube tutorials and official documentation to understand Simulink's block-diagram paradigm, parameter handling, and numerical integration tools. These resources were not followed verbatim but instead used as references while building, breaking, and debugging the model. The emphasis throughout was on understanding *why* the system behaved the way it did, rather than simply reproducing example models.

While instructional tutorials provided a foundation for using Simulink, a key part of the learning process occurred during debugging. An early issue involved the plant producing a zero output despite nonzero inputs, which was traced to parameter handling within the model. A gain block referencing a workspace variable with an unintended zero value effectively canceled the normal force calculation. Identifying this issue reinforced the importance of probing intermediate signals and validating parameter values when implementing dynamic models in Simulink. In addition to reviewing instructional material, several hours of self-learning were spent troubleshooting and validating the Simulink plant, as early implementation issues required iterative debugging and verification of intermediate signals.

Application

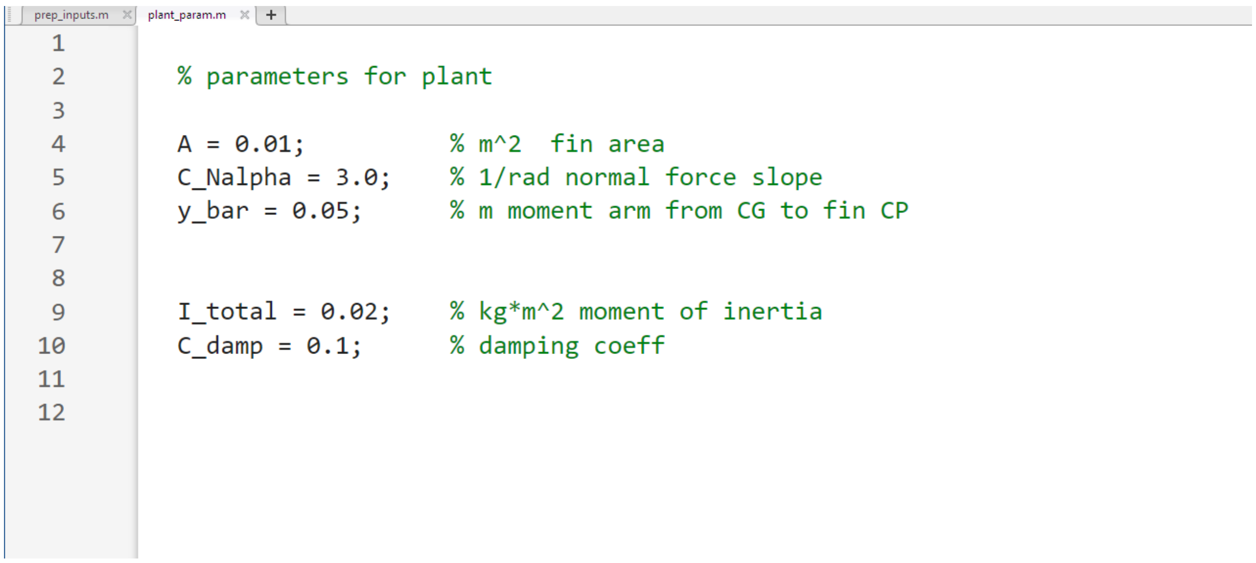
The learned Simulink skills were applied by constructing a roll-axis plant model that represents the ARC rocket's rotational dynamics using first-principles physics. The implementation began with initializing time-domain inputs derived from RockSim flight data. Density and velocity profiles were exported from RockSim to Excel, then imported into MATLAB and converted into workspace variables suitable for Simulink's time-based simulation. This step is shown in the MATLAB scripts included in the screenshots, where RockSim data is read, processed, and prepared for use as simulation inputs. Separating this data preparation step from the Simulink model itself made it easier to verify correctness and ensured that the plant was driven by realistic, time-varying flight conditions rather than constant or idealized inputs.



```
% prep_arc_inputs.m
% Loads RockSim time-history data into time series for Simulink!

Time    = readmatrix('ARC_RockSim_v1.xlsx','Sheet','ARC RockSim Data 1.0','Range','B5:B759');
Density = readmatrix('ARC_RockSim_v1.xlsx','Sheet','ARC RockSim Data 1.0','Range','G5:G759');
V       = readmatrix('ARC_RockSim_v1.xlsx','Sheet','ARC RockSim Data 1.0','Range','H5:H759');

Time = Time(:); Density = Density(:); V = V(:);

rho_ts = timeseries(Density, Time);
v_ts   = timeseries(V, Time);

t_end = Time(end); % for later

```

Figure 1: MATLAB Script to initialize ARC RockSim data into workspace for Simulink System

With the time-domain inputs prepared, plant parameters such as reference area, normal force coefficient slope, moment arm, total roll inertia, and damping coefficient were initialized in the MATLAB workspace. These parameters were then passed into the Simulink plant subsystem. Within the model, dynamic pressure was computed from density and velocity, aerodynamic normal force was calculated using angle of attack and aerodynamic coefficients, and that force was converted into a roll moment using the appropriate moment arm. A damping moment proportional to angular rate was included to capture rotational resistance, introducing a natural feedback path within the plant. The net moment was divided by inertia to compute angular acceleration, which was integrated to obtain angular velocity. Angular velocity was converted to RPM and routed to a scope for visualization. The Simulink screenshots illustrate this signal flow clearly, with intermediate signals such as dynamic pressure, moment, and angular rate available for probing and validation.



```

% parameters for plant

A = 0.01;           % m^2  fin area
C_Nalpha = 3.0;     % 1/rad normal force slope
y_bar = 0.05;       % m moment arm from CG to fin CP


I_total = 0.02;     % kg*m^2 moment of inertia
C_damp = 0.1;       % damping coeff


```

Figure 2: MATLAB Script to initialize plant variables

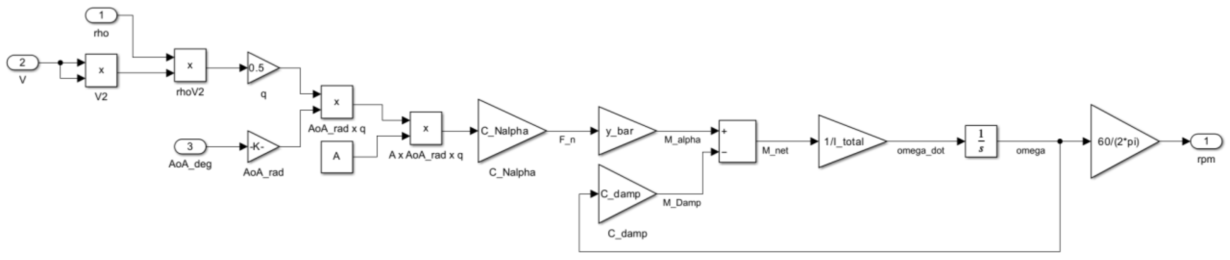


Figure 3: Actual Simulink Plant Diagram

Results and Conclusion

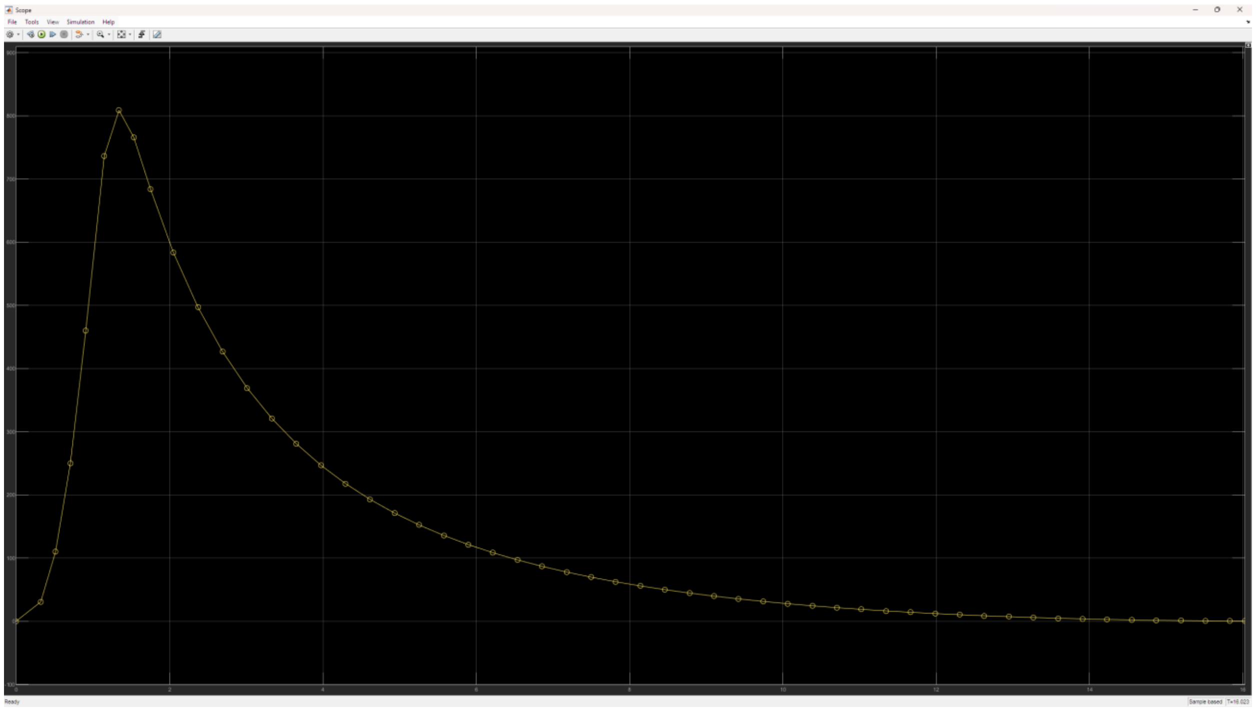


Figure 4: Scope Response of RPM in Time-Domain from Plant Loop

After resolving several implementation issues related to parameter scoping and signal flow, the Simulink plant produced a valid and nonzero time-domain response. When the simulation was run using RockSim-derived inputs, the scope output showed angular velocity increasing initially due to aerodynamic forcing and then decreasing as damping effects and changing dynamic pressure reduced the net roll moment. This RPM-versus-time plot, shown in the included figures, represents the open-loop response of the rocket's roll dynamics and confirms that the plant behaves as expected for a first-order rotational system.

The shape of the response provided immediate physical insight that was not apparent in previous script-based analyses. The rising portion of the curve corresponds to periods of higher dynamic pressure and aerodynamic moment, while the decay reflects damping and reduced excitation later in the flight. This validated that the plant correctly captures the interaction between aerodynamic forces, inertia, and damping. More importantly, it confirmed that the plant produces a meaningful output signal that can later be used by a controller to counteract unwanted roll motion. At this stage, the goal was not to tune performance or achieve stability, but simply to demonstrate that the plant responds dynamically and consistently to realistic inputs. This was seen as the realistic goal since the plant implementation was based on RockSim data and how to control that. In the actual flight computer, dealing with data in real time, a process similar to the RockSim derivative would be suitable.

References

[1] R. Wyatt, "Active Stabilized Rocket," AeroCon Systems, 2017. Available: aeroconsystems.com/tips/Active_Stabilized_rocket_Wyatt.pdf

[2] International Journal of Current Advanced Research, "Active Control Systems for Rockets," Int. J. Curr. Adv. Res., 2018. Available: internationalpubls.com/index.php/cana/article/download/2701/1610/4819

[3] California State University, Los Angeles, "Active Rocket Stabilization Project," Senior Design Project Slides, 2019. Available: calstatela.edu/sites/default/files/team_1_slides.pdf

[4] MathWorks, "MATLAB Simulink Tutorial for Beginners (Step-by-Step!)," YouTube, 2020. Available: youtube.com/watch?v=vxzR3W2BcRk

[5] MathWorks, "Getting Started with Simulink for Controls," YouTube, 2020. Available: youtube.com/watch?v=bE179wgm164

[6] MathWorks, "Everything You Need to Know About Control Theory," YouTube, 2020. Available: youtube.com/watch?v=lBC1nEq0_nk