# Active Control: MIT Rocket Team

***Marc D. NICHITIU, Conrad M. CASEBOLT***

*Active Control Algorithms and Simulation Testing for MIT Rocket Team.*

This repository contains tools and code for advanced simulation and control of rocket flight using both Java (OpenRocket) and Python. The workflow integrates custom Java simulation logic with Python scripting and visualization.

## 1 Overview

A Python-based tool for rocket flight simulation that integrates with OpenRocket to analyze and visualize flight characteristics. This project focuses on processing and visualizing rocket flight data, particularly useful for analyzing pitch rates and velocities during flight.

## 2 Prerequisites

- Python 3.12+
- Java Development Kit (JDK) 17
- `matplotlib`
- `numpy`
- `jpype`

## 3 Environment Setup

Python Dependencies: `pip install numpy matplotlib jpype1`

Contact: `nichitiu@mit.edu`

## 4 Usage Instructions

### 4.1 Compiling and Running

- **Use the `OR_interfaceTest + JAR` run configuration** in your IDE to:
  - Compile the Java code (including your modifications).
  - Run the Python script that interfaces with the compiled Java classes.

### 4.2 Editing Simulation and Control Logic

- **Simulation Flow:**
  Edit only `ModifiedEventSimulationEngine.java` to manage the overall simulation flow, such as event handling, simulation stepping, and integration with listeners.
- **Controller Behavior:**
  Edit `NewControlStepListener.java` to modify the controller's behavior and properties. This file is responsible for how the control system interacts with the simulation at each step.
- **Python Scripting:**
  Edit `openRocketInterface.py` (located in `sim/src/py/` or similar) to change how the Python script interacts with the Java code. This script is used to run specific tests, automate simulation runs, and collect results.

## 5 Python Script: `openRocketInterface.py`Python Script: openRocketInterface.py

This script uses `jpype` to launch the JVM and interface directly with the compiled Java classes. It allows you to:

- **Control Java Variables:**
  The script can set and get various simulation parameters in Java, such as:

  - Time step size (variable `prefDt` in `openRocketInterface.py`)
  - Controller gains and setpoints
  - Initial conditions (e.g., launch angle, velocity)
  - Simulation duration and event triggers
- **Run and Automate Simulations:**
  You can script multiple runs, parameter sweeps, or custom test scenarios by calling Java methods from Python.
- **Graphing and Visualization:**
  The script collects simulation output (e.g., altitude, velocity, control surface deflections, error signals) and generates plots for analysis. The default graph plots two panels, showing
  - Altitude, Velocity vs. time
  - Control output (fin cant) and rotational velocity vs. time

# 6 Repository File Structure

This repository is organized as follows:

- `README.md` Project documentation and usage instructions.
- `bib/`
  Reference materials and research papers.
- `canard/`, `ctrl/`, `wing/`
  Subdirectories for specific rocket components, control models, and aerodynamic studies.
- `clone/`
  Contains OpenRocket source code and related files:
  - `openrocket/` and `openrocket-release-24.12.RC.01/`
    OpenRocket Java source, build scripts, and documentation. `openrocket/` is a symlink to the openrocket version used, currently `openrocket-release-24.12.RC.01/`.
- `land.tbd/`
  Landing simulation (tbd - i.e. in progress) documentation and related files.
- `sim/`
  Main simulation code and data:
  - `src/py/`
    Python scripts for simulation and Java-Python integration (e.g., `openRocketInterface.py`).
- `src/java/`
  A symlink to the Java source of the used openrocket distribution - for ease of navigation.
- `dat/`
  Simulation results and data output.
- `stabil/`
  Additional files pertaining to the roll controller, mostly ideation.
- `test/`
  Test scripts and utilities.

Each directory contains files relevant to its purpose, such as source code, data, documentation, or research materials. The main simulation workflow is in the `sim/` directory, with integration to Java code in `clone/openrocket-release-24.12.RC.01/`.

# 7 Sample Output

## 7.1 Setup

Below is a comparison of the `RK4` and `RK6` integration methods for controller stability. The control coefficients used are:

- $k_P = 2.0$
- $k_I = 0.75$
- $k_D = 0.226$
- $v_{\min} = 15$ m/s

The controller will thus use a PID controller (see below) when the rocket's vertical velocity is larger than 15 m/s. We also note that the controller is only applied on ascent. Here, the controller's "ideal" rotational velocity is **zero**. Let $f(t)$ be the fin cant at some time $t$. Let the simulation step be $dt$. Let $\omega_0$ be the desired rotational velocity (here 0), and let $\omega(t)$ be the measured rotational velocity. Thus, the controller uses the following equation to calculate the next fin cant:

$$f(t+dt) = (\omega_0 - \omega(t))k_P + (\omega(t-dt) - \omega(t))k_D + k_I \sum_j (\omega_0 - \omega(t-jdt)) \tag{1}$$

Note that fin cant is capped at 15° in the `openrocket` framework.

As a sample rocket, we use that described by the file 'sim/dat/ork/canard1.ork'.

## 7.2 Results

Upon simulation with varying timesteps, it is possible to observe an ameliorated modeling of the controller. In the graphs below, at left is an `RK4` simulation, and at right is an `RK6` simulation. Note that in each graph, the angular velocity scale (bottom panel, left side) uses a `linlog` scale that is logarithmic for values larger than $10^{-3}$ in absolute value and linear for values less than $10^{-3}$ in absolute value. Also note that the horizontal black dotted line in the upper panel denotes the minimum velocity threshold for controller engagement, noted above to be 15 m/s.

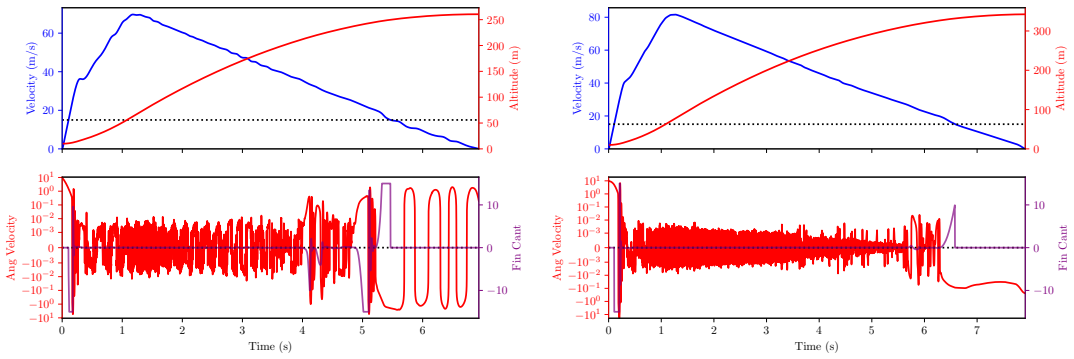


**Figure 1.** $dt = 10^{-3}$

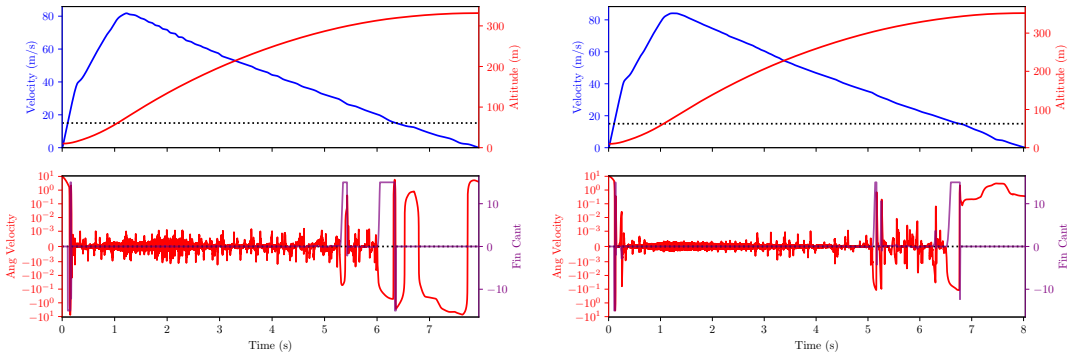


**Figure 2.** $dt = 10^{-4}$

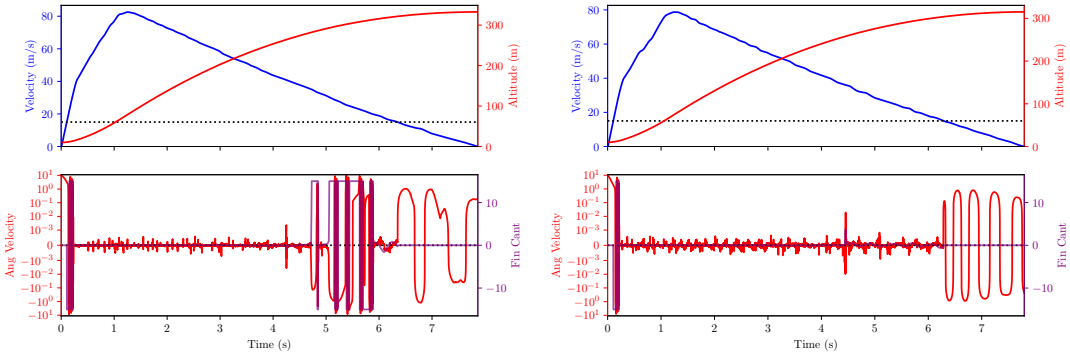


**Figure 3.** $dt = 10^{-5}$