GNCSIM: A High-Fidelity 6-DOF Simulation Framework for Closed-Loop Rocket Navigation and Control

Karsten Caillet*, David P. Reynolds†, Kandarp Vadia‡, Matthew Foley§, Aditya Swami¶, Andrés Chiva Ruiz ‖
Georgia Institute of Technology, Atlanta, GA, 30332

GNCSIM is a MATLAB simulation framework designed to support rapid development, testing, and validation of state estimation and control algorithms for sounding and high-powered rockets. It is developed by the Georgia Tech Guidance Navigation and Controls (GNC) team as part of the Ramblin' Rocket Club (RRC). It leverages a modular software architecture that allows for a high degree of customization and can produce accurate flight trajectories for any combination of rocket geometry, fin geometry, thrust curve, parachute model, actuator model, sensor model, and controller. It features accurate dynamic mass variations and leverages Digital DATCOM, a semi-empirical method for fast aerodynamic coefficient estimation. GNCSIM natively supports linear time-invariant controller design and simulation by providing automatic reference trajectory generation, linearization, gain scheduling, and model reduction capabilities. GNCSIM also provides built-in support for sensor modeling coupled with a navigation stack, enabling evaluation of pre-flight and in-flight state estimation performance. Six-degree-of-freedom rigid-body equations of motion are used to propagate the rocket states through time. To simulate closed-loop control in flight, the controller calculates a desired actuator deflection at every time step based on the latest state estimate. The actuator states are then propagated accordingly based on their dynamics. The simulation incorporates launch rail dynamics, closed-loop control, and parachute dynamics, spanning from motor ignition to landing. Finally, GNCSIM supports large-scale Monte Carlo simulations to quantify the robustness and statistical performance of state estimation and control algorithms under uncertainty.

I. Introduction

The Guidance, Navigation and Control (GNC) rocket club at the Georgia Institute of Technology aims to research and test active attitude stabilization in high powered rockets of various sizes and designs. Current projects of the club include a 65 inch reaction-wheel roll stabilized rocket, a 50 inch thrust-vectored gimbaled rocket, and a 114 inch fin-tabs rocket. For each of these rockets, a high fidelity simulation is needed to inform the engineers of the expected performance of the hardware and software ahead of flight. Publicly available rocket simulation software like RocketPy [1], OpenRocket [2], or RASAero II [3] exist, but none natively support the integration of closed-loop state estimation and controls. The lack of publicly available simulation software suited to the club's needs, combined with the wide variety of rockets and actuators being tested, drives the need for a custom, flexible and scalable simulation framework.

GNC Simulation, Integration, and Modeling (GNCSIM) is a MATLAB-based simulation framework developed to address this need over the long term. GNCSIM is designed to support fast navigation and controller algorithms development, simulation and validation while remaining scalable for future requirements. It is built with modularity and closed-loop simulation capabilities at its core, and written entirely in MATLAB to promote accessibility to engineering students, regardless of their technical background. A separate companion paper details the specific application of GNCSIM to the Fin-Tabs Rocket, with its controller and navigation algorithm design, testing and validation. The remainder of the paper details GNCSIM's architecture, capabilities and general results.

*GNC Controls Co-Lead, Undergraduate, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member 1537725
†GNC Controls Co-Lead, Graduate, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member 1789019
‡Controls Team Member, Undergraduate, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member 116988
§Controls Team Member, Undergraduate, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member 1931532
¶Controls Team Member, Undergraduate, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member 1930856
‖Controls Team Member, Undergraduate, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member 1930708

II. Software Architecture

The GNCSIM class architecture is displayed on Fig. 1. The core of the simulation is handled by the `Flight` class, including the six-degrees-of-freedom (6-DOF) equations of motion (EOM), the Runge-Kutta 4 numerical solver, and the state machine logic. The `Flight` class is complemented by the `Environment` class to provide wind, air density, and speed of sound at the current vehicle altitude. The `Rocket` class computes and aggregates all forces and moments acting on the vehicle at any given time in the simulation, and holds the mass, moments of inertia (MOI) and the center of gravity (CG). Its capabilities are extended by the `Body` and `Fins` classes for aerodynamics forces and moments, and the `Parachute` class for the descent phase of flight. Before the simulation begins, an optimal trajectory is generated and then linearized by the `Breakpoints` class and populated with gains by the `Controller` class, which supports any Linear Time Invariant (LTI) controller.

At every simulation step, the `Flight` class feeds the vehicle states to the sensor models, which generate raw, noisy readings, which are then passed to the `Navigation` class to perform state estimation [4]. The state estimate is then passed to the `Controller` class through the `Flight` class. The controller then computes the control input at each time step based on the latest state estimate. From the control input, the `Actuator` class generates the corresponding forces and moments.

Finally, for each flight, the results are recorded in `Log` objects. The `MonteCarlo` class leverages the capabilities of the `Flight` class to run many simulations with randomized vehicle and environmental parameters. The `Analysis` class then plots the results from either a single flight or a batch of flights and generates a comprehensive simulation report.

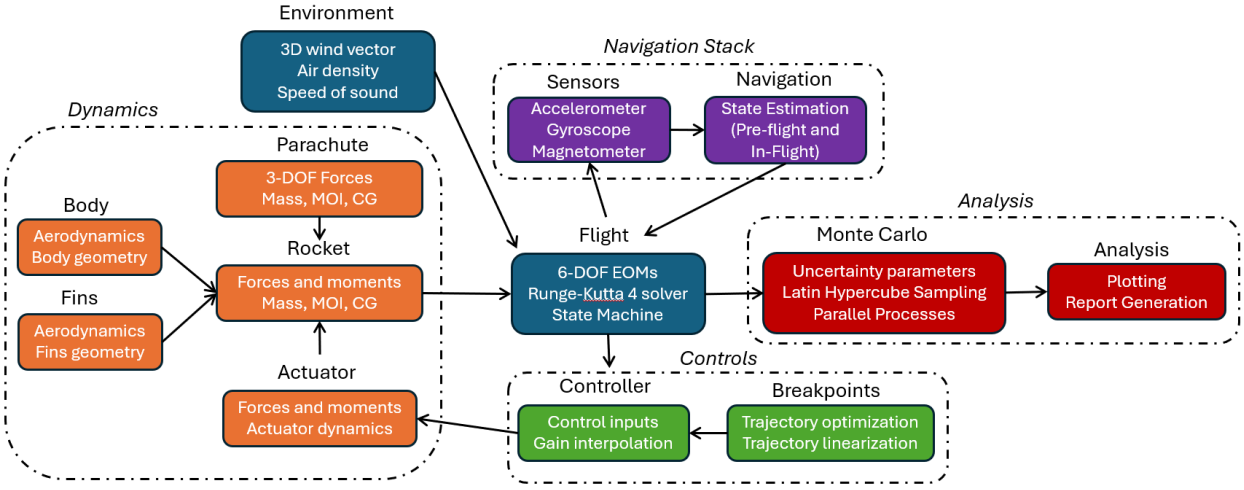


Fig. 1 GNCSIM Software Architecture Diagram.

III. Flight Simulation

A. Equations of Motion

The core of the `Flight` class holds the equations of motion, which mathematically define how the rocket evolves in 6 dimensions (translational and rotational) given a set of reference frames. The inertial frame coordinates are defined in East (x), North (y), Up (z) (ENU). The body frame is defined with x pointed out the nose, y pointed north on the launch rail, and z completing the right hand rule. Both the body and inertial frame are shown in Fig. 2.

The rocket equations of motion are defined as a set of time varying nonlinear ordinary differential equations (ODE). As high-powered rockets travel over limited range and altitude, several key assumptions can be made to simplify these equations: the Coriolis acceleration due to Earth's rotation is assumed negligible, gravity is assumed to be constant, and rigid body dynamics, except for constant mass, are assumed to apply.

The state vector is composed of the body position expressed in the inertial frame ($\vec{p}_I$), the body velocity expressed in the body frame ($\vec{v}_B$), a quaternion representing the orientation from the inertial frame to the body frame ($\vec{q}_{IB}$) and the angular rate of the body frame with respect to the inertial frame, expressed in the body frame ($\vec{\omega}_B$). Mass (m) is added as an extra state since it varies as a function of thrust ($T(t)$) and total impulse (I_{total}).

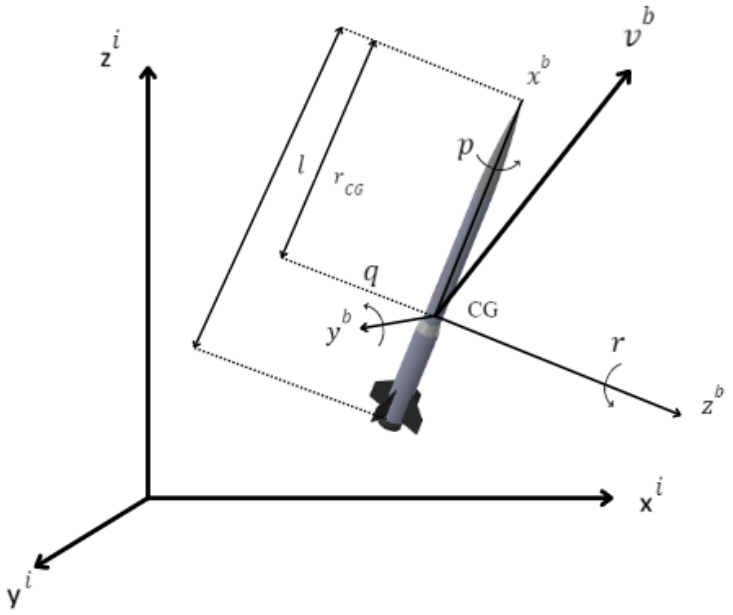


Fig. 2 Sign and frame conventions for the Fin Tabs Rocket.

$$\dot{m} = -\frac{T(t)\left(m_{\text{wet}} - m_{\text{dry}}\right)}{I_{\text{total}}} \tag{1}$$

$$\dot{\vec{p}}_I = C_{BI}(\vec{q}_{IB})v_B \tag{2}$$

$$\dot{\vec{v}}_B = \frac{\vec{F}_B}{m} - \vec{\omega}_B \times \vec{v}_B \tag{3}$$

$$\dot{\vec{q}}_{IB} = \frac{1}{2}\Omega(\vec{\omega}_B)\vec{q}_{IB} \tag{4}$$

$$\dot{\vec{\omega}}_B = I_B^{-1}\left(\vec{M}_B - \vec{\omega}_B \times (I_B\vec{\omega}_B)\right) \tag{5}$$

In the equations of motion, $C_{BI}(\vec{q}_{IB})$ is defined as a rotation matrix from the inertial to the body frame parameterized by the quaternion $\vec{q}_{IB}$.

$$C_{BI}(\vec{q}_{IB}) = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{6}$$

Furthermore, $\Omega(\vec{\omega}_B)$ is defined as a skew symmetric matrix which maps the 3D angular velocity into the 4D quaternion space.

$$\Omega(\vec{\omega}_B) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix} \tag{7}$$

Finally, I_B is the inertia tensor of the vehicle expressed in the body frame. $\vec{F}_B$ denotes the total external forces acting on the vehicle expressed in the body frame, and $\vec{M}_B$ denotes the total external moments acting on the vehicle expressed in the body frame. Forces and moments calculation are detailed in Section IV.

B. Simulation

Once the ODEs are defined, they are integrated through time to yield the rocket's trajectory. The Runge-Kutta 4 numerical solver is chosen for its balance between speed and flexibility. The simulation supports all stages of flight, including pre-launch, launch rail, powered ascent, coasting ascent, descent, and landing. The simulation behavior during each state is described below:

1) Pre-launch: The rocket remains idle and waits for motor ignition. Pre-flight state estimators can be implemented at this stage to yield an initial state estimate, an example implementation for the fin-tabs rocket is described in [4].
2) Launch rail: Once the motor ignites, the rocket's attitude remains constrained to the launch rail's orientation, and only total forces in the body x axis are applied until the launch rail is cleared.
3) Powered ascent: All forces and moments apply until motor burnout. Gimbal actuators can be implemented at this stage.
4) Coasting ascent: All forces and moments except thrust apply until apogee, mass is kept constant.
5) Descent: Only parachute forces apply, angular rates and attitude are zeroed to improve numerical stability and solver speed.
6) Landing: Rocket reaches starting altitude, simulation is stopped and logged.

C. Environment

The final piece to complete the core of the simulation is the environment model. It provides atmospheric wind and air property data as a function of altitude.

The mean wind speed profile is modeled using the empirical power law for the atmospheric boundary layer [5], where U_0 is the reference wind speed at reference height z_{ref}, and α is the terrain-dependent shear exponent, set to $\alpha = 0.14$ for open terrain under neutral stability conditions.

$$U(z) = U_0 \left(\frac{z}{z_{\text{ref}}} \right)^{\alpha} \tag{8}$$

Wind direction varies with altitude through a logarithmic veer profile, where θ_0 is the base wind direction and k is the veer coefficient, with the total direction change limited to $\pm 45°$.

$$\theta(z) = \theta_0 + k \ln\left(\frac{z}{z_{\text{ref}}} + 1 \right) \tag{9}$$

The full 3D mean wind vector is then

$$\vec{W}_{\text{mean}}(z) = \begin{bmatrix} U(z) \cos\theta(z) \\ U(z) \sin\theta(z) \\ 0 \end{bmatrix} \tag{10}$$

Turbulence is modeled as vertically correlated noise using a first-order auto-regressive (AR(1)) process, where ρ is the vertical correlation coefficient and $\sigma(z)$ is the altitude-varying turbulence amplitude scaled proportionally to the local mean wind speed, with turbulence intensity I_0.

$$n_i = \rho\, n_{i-1} + \sqrt{1 - \rho^2}\, \sigma(z_i)\, \xi_i, \quad \xi_i \sim \mathcal{N}(0, 1) \tag{11}$$

$$\sigma(z) = I_0\, U(z) \tag{12}$$

The vertical turbulence component is attenuated relative to the horizontal by a scale factor $k_w \leq 1$. The total 3D wind vector is,

$$\vec{W}(z) = \vec{W}_{\text{mean}}(z) + \begin{bmatrix} n_x(z) \\ n_y(z) \\ k_w\, n_z(z) \end{bmatrix} \tag{13}$$

Speed of sound a and density ρ_{air} are obtained as functions of altitude from the International Standard Atmosphere (ISA) model [6], accessed via MATLAB's `atmosisa` function.

IV. Rocket Dynamics and Modeling

A. Acting Forces and Moments

Forces and moments on the rocket originate from various sources at any given time t in the simulation. The `Rocket` class sums all the individual contributors to yield a net total force and moments expressed in the body frame and applied at the center of mass.

- Gravity: defined in the inertial frame as $\vec{F}_I^g = [0,\ 0,\ -m(t)g]^T$, where $m(t)$ is the current vehicle mass and $g = 9.81$ m/s^2. Gravity produces no moment as it acts through the center of mass.
- Thrust: modeled in the body frame as $\vec{F}_B^T = [T(t),\ 0,\ 0]^T$, where $T(t)$ is obtained from the thrust curve of the motor. It produces no moment either as it is assumed to act through the center of mass.
- Passive aerodynamics: force $\vec{F}_B^a$ and moment $\vec{M}_B^a$ are computed from the rocket and fin geometry via Digital DATCOM, detailed in Section IV.B.
- Actuator Model: active forces $\vec{F}_B^d$ and moments $\vec{M}_B^d$ on all three axes, generated by the `Actuator` class, detailed in Section IV.C.

With $C_{BI}(\vec{q}_{IB})$ representing the rotation from the inertial to the body frame, the net force and moment vectors applied at the center of mass are

$$\vec{F}_B = C_{BI}(\vec{q}_{IB})\vec{F}_I^g + \vec{F}_B^a + \vec{F}_B^T + \vec{F}_B^d \tag{14}$$

$$\vec{M}_B = \vec{M}_B^a + \vec{M}_B^d \tag{15}$$

B. Aerodynamics Modeling (DATCOM)

Digital DATCOM is a computer program made by the US Air Force in the 1970s used to calculate the static stability and dynamic derivative characteristics of fixed-wing aircraft. While originally intended to study the aerodynamics of aircraft, by treating the rocket fins as "wings", Digital DATCOM can also be applied to generate aerodynamic look-up tables for rockets.

While there is built-in compatibility between MATLAB and DATCOM, primarily through MATLAB's Aerospace Toolbox, the implementation of the tool still requires data formatting. First, the geometric parameters of the rocket are formatted into a valid DATCOM input file. These parameters are split into two groups: the body geometry, containing the rocket nose length, body length, and body diameter, and the fin geometry, containing root chord, tip chord, fin span, and sweep angle. The input file also includes an array of combinations of various vehicle speeds, altitudes, and angles of attack that the user is interested in.

After creating this input file, DATCOM completes its calculations and returns an output file containing aerodynamic coefficients. The parts of interest include the pitching moments, normal coefficients, and axial coefficients. It is important to note that these DATCOM calculations assume planar symmetry, providing only pitching moment coefficients with negligible sideslip.

Once these aerodynamic coefficients are found, it is possible to calculate the forces and moments acting on the rocket, which need to be first computed in the fin component frame, as shown in Fig. 3. The two angles used are θ, the position of the fin around the rocket body, and δ the deflection angle of the fin. Note that δ defaults to 0 if the fin is not implemented as an active control surface through the `Actuator` class.

The equations are shown in order of calculation in Fig. 4. Firstly, the vehicle's true airspeed v_B is computed and expressed in the body frame. Then, the true airspeed and angular velocity are converted to the component frame using θ and δ. From there, the fin Mach number M and angle of attack α are computed and used on the DATCOM generated lookup table to obtain the relevant aerodynamics coefficients. Next, the aerodynamics forces and moments components are calculated using dynamic pressure q_c and fin geometry. Finally, the forces and moments vectors are rotated back to the body frame. This information can now be fed into the overall rocket dynamics model for further use.

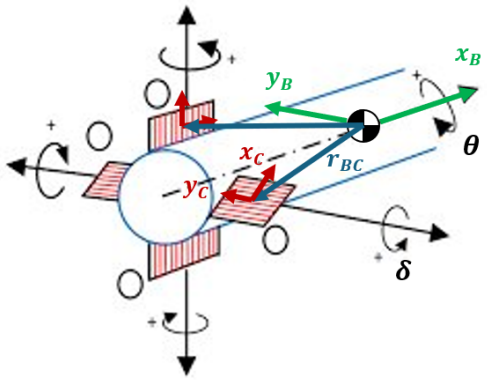


Fig. 3 Rocket body frame's relation to fin component frame.

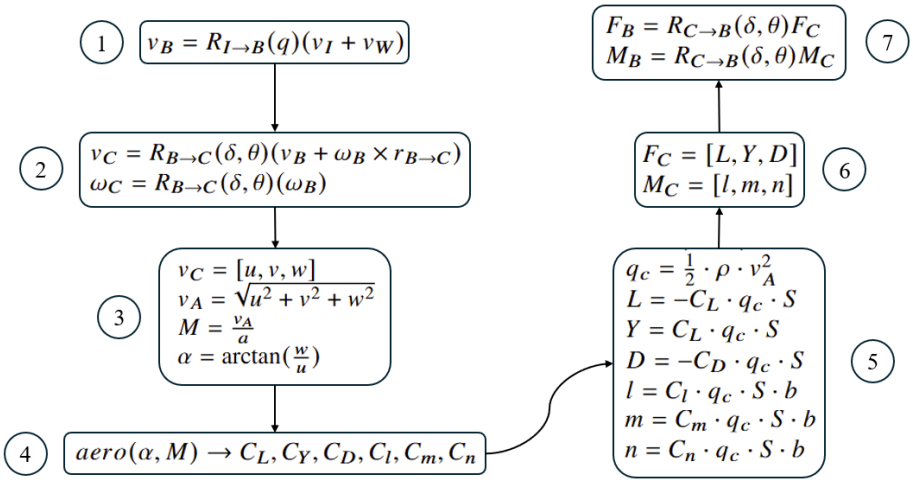


Fig. 4 Aerodynamic modeling derivation equations.

C. Actuator Model

The `Actuator` class provides a standard interface to support various actuator models, accommodating a wide range of control mechanisms such as actuated fins, gimbaled motors, reaction wheels, and RCS thrusters. As such, it is very flexible, and gives full freedom to the users to define how actuator states translate into forces and moments, and how control inputs govern actuator dynamics.

D. Parachute Dynamics

The `Parachute` class enables the modeling of parachutes during the descent phase of the rocket. This class enables the creation of parachute objects for a given rocket and computes the resulting forces. The inputs required include canopy diameter, canopy mass, drag coefficient, and deployment altitude for any secondary parachute. Each parachute is implemented within a linked-list structure, allowing multiple parachutes to be modeled independently and to be deployed one after another robustly. This class outputs aerodynamic forces in the rocket body-fixed reference frame, which are then used in the `Rocket` class to inform trajectory propagation.

The parachute model is simplified to a 3-DOF translational formulation. Rotational coupling between the canopy and rocket body is neglected, and aerodynamic moments are not modeled. Moreover, the angle of attack of the canopy is considered negligible. Small canopy oscillations and turbulence-induced moments are assumed to average out over the descent and therefore have a negligible impact on landing position accuracy. These assumptions significantly reduce

computational cost while preserving the dominant physics governing descent.

Under these assumptions, the parachute has only one effect, which is the drag force opposing the direction of the relative air velocity vector. The aerodynamic force is computed in the ENU frame and rotated into the body-fixed reference frame before being applied in the equations of motion.

$$F^b = B_i^b \left(-\frac{1}{2} \rho C_D A \left\| v_{rel}^i \right\| v_{rel}^i + F_g \right) \tag{16}$$

where F^b is the total force applied in the rocket body-fixed frame, B_i^b is the rotation matrix from the Flat Earth frame to the body-fixed frame, ρ is the atmospheric density, C_D is the parachute drag coefficient, A is the parachute reference area, and F_g is the gravitational force vector in the inertial frame.

E. Center of Gravity and Moments of Inertia

Finally, the `Rocket` class also handles changes in center of gravity and moment of inertia. As propellant is consumed, both the center of mass x_{cg} and the inertia tensor I_B vary linearly with mass between their wet and dry values,

$$x_{cg}(m) = x_{cg}^{\text{dry}} + \frac{m - m_{\text{dry}}}{m_{\text{wet}} - m_{\text{dry}}} \left(x_{cg}^{\text{wet}} - x_{cg}^{\text{dry}} \right) \tag{17}$$

$$I_B(m) = I_B^{\text{dry}} + \frac{m - m_{\text{dry}}}{m_{\text{wet}} - m_{\text{dry}}} \left(I_B^{\text{wet}} - I_B^{\text{dry}} \right) \tag{18}$$

V. Controller Implementation and Features

A. Trajectory Generation

Generating a nominal trajectory requires solving an optimization problem with specified rocket dynamics, path constraint, and boundary constraint. Additionally, there are path bounds on the state and on control, as well as bounds on the initial and final time and state of the rocket. To reach a solution, the problem needs to be rewritten from a continuous-time trajectory to a constrained parameter, finite-dimensional nonlinear program. This transcription is achieved via direct collocation, specifically using the Hermite-Simpson Collocation [7].

The collocation approximates our objective function and our dynamics as quadratics.

$$\int_{t_0}^{t_F} w(\tau)\, d\tau \approx \sum_{k=0}^{N-1} \frac{h_k}{6} \left(w_k + 4 w_{k+\frac{1}{2}} + w_{k+1} \right) \tag{19}$$

$$x_{k+1} - x_k = \frac{1}{6} h_k \left(f_k + 4 f_{k+\frac{1}{2}} + f_{k+1} \right) \tag{20}$$

This implies that the state is a cubic polynomial which, for each time step, will match the endpoints and the slope of the dynamics at those endpoints. Additionally, the state derivative at the midpoint is constrained to be equal to the cubic state trajectory by completing a Hermite cubic interpolation.

$$x_{k+\frac{1}{2}} = \frac{1}{2}(x_k + x_{k+1}) + \frac{h_k}{8} (f_k - f_{k+1}) \tag{21}$$

The implementation of this solution method in GNCSIM required the use of OptimTraj [8], a MATLAB library for solving nonlinear problems via direct collocation. The inputs include state and control bounds, an initial guess for trajectory, and the objective and constraint functions. The solver then outputs a nominal trajectory for the simulation to utilize.

B. Trajectory Linearization

Partitioning the trajectory into segments that can be linearized requires the use of breakpoints. These points are indexed by the fraction of the accumulated velocity at that point rather than on the basis of metrics such as time. This is done since the rocket's velocity accumulation is a better representation of its flight progress than elapsed time, and it is

mostly proportional to momentum, capturing both the mass decrease and speed increase. These breakpoints are used in gain scheduling, utilizing the stored state information at the points to develop linear controllers that will switch during flight. The linearization of the system dynamics is contained within the controller used. For the fin tabs rocket [4], a reduced order model linearization is implemented.

C. Controller Implementation

GNCSIM utilizes a linear feedback controller for its attitude tracking. At each simulation time step, the controller determines the vehicle's progress based on the fractional accumulated velocity. The controller then linearly interpolates the neighboring breakpoints to get an approximate linear model of the system and controller. It is important to note that the gain scheduling is independent of the controller allowing support for a wide variety of linear control strategies, such as Linear Quadratic Regulators (LQR) or Proportional-Integral-Derivative (PID) controllers. By decoupling the gain scheduling from the control logic GNCSIM allows for flexible testing environment.

VI. Sensor Modeling and Navigation

GNCSIM implements a navigation stack for both pre-flight and in-flight state estimation using an inertial measurement unit (IMU) sensor model, GPS model, and an extended Kalman filter (EKF). The navigation subsystem is a modular architecture with each sensor modeling inheriting from a common class to schedule outputs with a main `Navigation` class to perform the sensor data fusion and state estimation. This allows individual sensor models to be swapped without modifying the estimation algorithm.

A. Sensor Architecture

Each sensor in GNCSIM inherits from a common `Sensor` class. The sensors store an output rate, output period, and timestamps for the past and next scheduled output. This design separates the sensor model from the simulation timestep, allowing each sensor to individually determine whether a new measurement is available based on its sample rate and the current simulation time. This allows the models to replicate real sensors, as commercial GPS receivers typically operate anywhere between 1-10 Hz while commercial IMUs operate around 100 Hz [9]. The sensors are also initialized with noise covariance and observation matrices to be passed to the EKF for use in the measurement update step. The parameters for each sensor are constructed in the main rocket builder file to fit the needs of each rocket.

B. IMU Modeling

1. Gyroscope

The gyroscope model measures the angular velocity in the body frame, ω_B, directly from the simulated state vector. Two error sources, a constant bias vector, scale factor, and stochastic white noise derived from the Angular Random Walk (ARW), are applied to achieve a realistic measurement. The noisy gyroscope output is calculated as:

$$\tilde{\omega} = diag(1 + SF)\omega_B + b_g + \frac{ARW}{\sqrt{\Delta t}} \cdot \eta, \qquad \eta \sim \mathcal{N}(0, I) \tag{22}$$

where b_g is the gyroscope bias vector, delta t is the output period, and n is a white noise vector. ARW characterizes the power spectral density as it scales with time; dividing by the square root of the interval converts the density into a per-timestep noise vector.

2. Accelerometer

The accelerometer model measures specific force at the IMU location. Because the IMU is generally not placed at the rocket's center of gravity, which itself will change throughout flight, a lever-arm correction must be applied before state estimation. The true specific force at the IMU is derived from the simulated acceleration at the center of gravity and corrected for the body's angular rate and lever-arm from the center of gravity to the IMU shown below [10].

$$F_{s,IMU} = a_B - \omega_B \times v_{CG} - \omega_B \times (\omega_B \times r_{IMU}) - C_{BI} \cdot g_I \tag{23}$$

Here, a_B is the center-of-gravity acceleration expressed in the body frame, C_{BI} is the direction cosine matrix which rotates vectors from the inertial to the body frame, ω_B is the body angular rate, r_{IMU} is the lever-arm radius vector, v_{CG}

is the velocity from the movement of the center of gravity, and $g_I = \begin{bmatrix}0 & 0 & -9.81\end{bmatrix}^T \frac{m}{s^2}$ is the gravity in the inertial, East, North, Up (ENU) frame. The $\omega \times (\omega \times r)$ is the dominant lever-arm contribution and $\omega \times v_{CG}$ is the contribution from the movement of the center of gravity.

After the lever-arm correction, sensor errors are applied. The accelerometer model includes a scale-factor matrix derived from a scale factor vector, a bias vector b_a, and stochastic noise characterized by the Velocity Random Walk (VRW).

$$\tilde{a} = \text{diag}(1 + \text{SF})\, F_{s,IMU} + b_a + \frac{\text{VRW}}{\sqrt{\Delta t}}\,\eta, \qquad \eta \sim \mathcal{N}(0, I) \tag{24}$$

VRW is analogous to ARW for the accelerometer and the scale factor matrix captures errors along each of the axes independently. Figure 5 shows an example of a raw accelerometer output for a sinusoidal input. This includes a bias, scale factor, and VRW term. The gyroscope output would behave similarly with different terms for bias, scale factor, and ARW with values shown below.

$$b_a = \begin{bmatrix}0.05 & 0.05 & 0.05\end{bmatrix}^T, \quad SF = \begin{bmatrix}0.01 & 0.01 & 0.01\end{bmatrix}^T, \quad VRW = \frac{1}{60}\begin{bmatrix}0.033 & 0.039 & 0.039\end{bmatrix}^T$$

3. Magnetometer

The magnetometer model simulates a three-axis magnetic field sensor used for preflight attitude estimation. The vehicle position is maintained in an ENU frame relative to a specified reference LLA. At the time of the preflight estimate, the current ENU position is converted to geodetic latitude, longitude, and altitude using a transformation of the WGS-84 Earth model.

The resulting LLA coordinates, along with the specified calendar date, are provided to MATLAB's World Magnetic Model function to compute the local geomagnetic field vector in the North, East, Down (NED) frame. This field is converted from NED to ENU coordinates. The ENU magnetic field represents the true inertial-frame magnetic vector at the rocket's location.

A standard magnetometer measurement model is used to account for calibration errors and disturbances. The measured magnetic field vector $\vec{\tilde{m}}$ is modeled as

$$\vec{\tilde{m}} = S_I^{-1}\left(\vec{m}_E + \vec{m}_{e(t)}\right) + \vec{b}_{HI} + \vec{m}_{i(t)} \tag{25}$$

where $\vec{m}_E$ is the Earth's magnetic field, S_I is the soft-iron calibration matrix, $\vec{b}_{HI}$ is the hard-iron bias, $\vec{m}_{e(t)}$ represents external magnetic disturbances, and $\vec{m}_{i(t)}$ is zero-mean Gaussian white measurement noise.

A corresponding compensation model is applied:

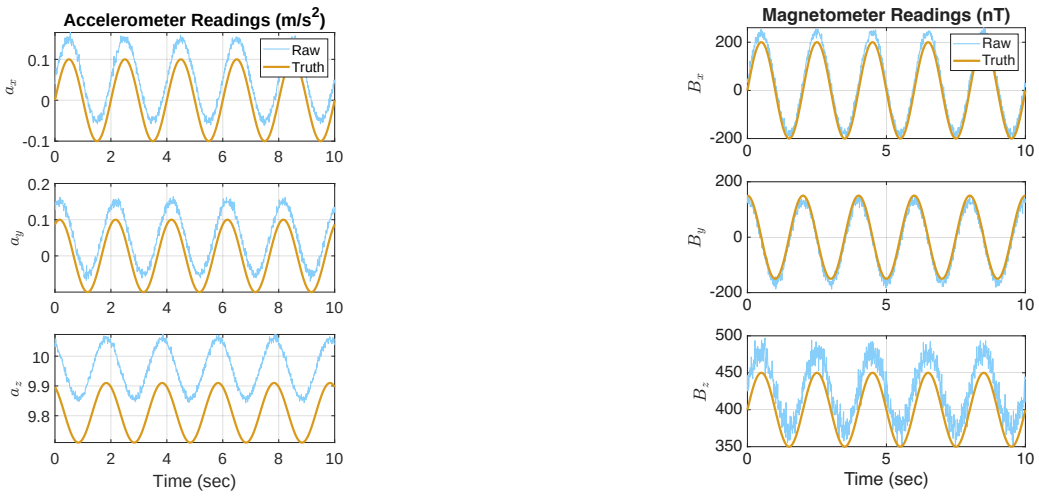
$$\vec{m}_c = S_I\left(\vec{\tilde{m}} - \vec{b}_{HI}\right) \tag{26}$$

The true inertial magnetic field vector is first rotated into the body frame using the current attitude quaternion. Soft-iron distortion is modeled using a 3×3 calibration matrix and hard-iron effects are modeled as an added bias vector. Zero-mean Gaussian white noise, with the sensor noise's covariance, is added to each axis to represent measurement uncertainty. The resulting three-axis body-frame magnetic field measurement is returned as the sensor output for use in preflight attitude determination.

Figure 5 shows raw magnetometer measurements for a sinusoidal input magnetic field. The raw outputs include soft-iron distortion modeled by

$$S_I = \begin{bmatrix}1.05 & 0.02 & 0.00\\ 0.02 & 0.98 & 0.01\\ 0.00 & 0.01 & 1.03\end{bmatrix},$$

a hard-iron bias of $\vec{b}_{HI} = \begin{bmatrix}30 & -20 & 15\end{bmatrix}^T$ nT, and additive zero-mean Gaussian white noise with $\sigma = 10$ nT.

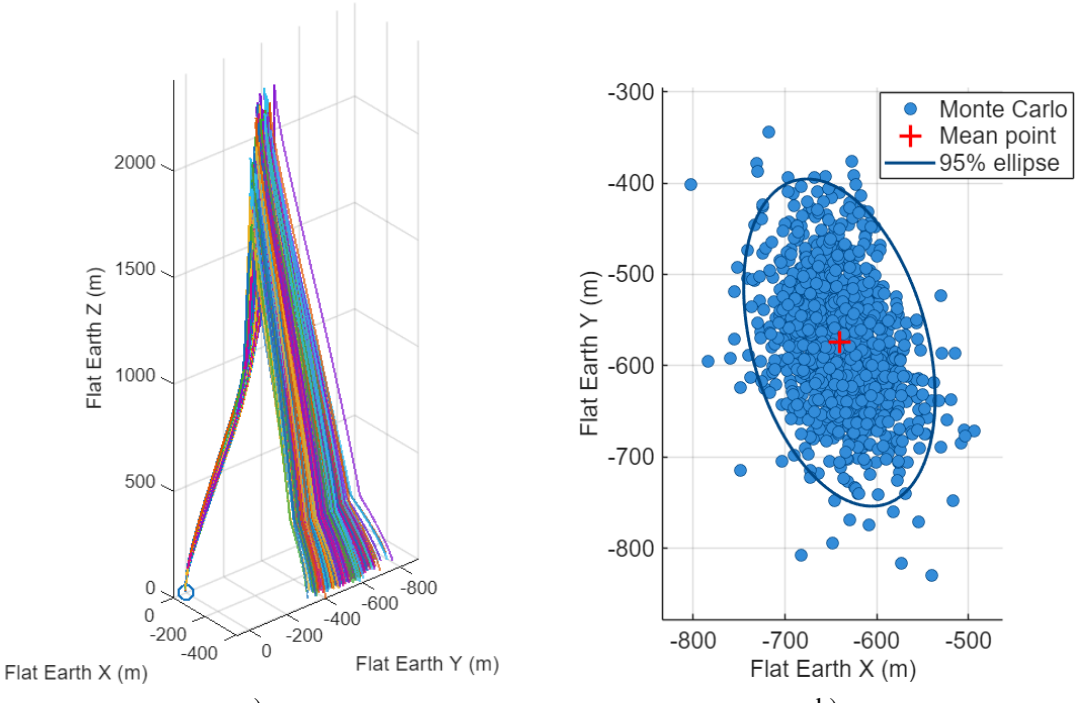


a) b)

Fig. 5 Raw simulated measurements from truth sinusoidal input for: a) Accelerometer, b) Magnetometer.

VII. Monte Carlo capabilities

To evaluate the robustness of guidance, navigation, and control algorithms, GNCSIM supports Monte Carlo simulations with efficient uncertainty quantification through Latin Hypercube Sampling (LHS). The framework can systematically vary multiple parameters simultaneously with user-defined uncertainty bounds.



a) b)

Fig. 6 a) Trajectory and b) Landing Dispersion for a 1000-sample Monte Carlo simulation.

Uncertainty can be attributed to 46 different parameters with uniform or gaussian (with 3σ deviation) distributions. The parameters that can be defined as uncertain are the following: aerodynamic scale factors (C_a, C_m, C_n), mass

properties (wet and dry mass, longitudinal and rotational moments of inertia, center of gravity locations), motor characteristics (total impulse, burn time, ignition time), geometric parameters (nose length, body length, rocket diameter), actuator dynamics (maximum deflection, rate limits, time constant), parachute drag coefficients, launch rail parameters (length and orientation), reference wind speed, base wind direction, turbulence intensity and sensor parameters (accelerometer noise σ_a, navigation covariance scale, GPS vertical and horizontal accuracy, and GPS output probability).

The framework automatically preserves critical inter-dependencies between parameters: when total impulse or burn time varies, the thrust curve scales appropriately in both magnitude and time while maintaining constant specific impulse I_{sp}. Similarly, dry mass recalculates based on the perturbed wet mass and total impulse to ensure physically consistent propellant mass across all simulations. Accounting for these uncertainties enables the establishment of confidence bounds on key mission outcomes, such as apogee altitude and landing dispersion radius. Figure 6 shows an example 3D trajectory and landing dispersion over a set of 1000 Monte Carlo samples, obtained using a Linear Quadratic Regulator [4] as the control strategy for a fin-tab rocket.

VIII. Conclusion

This paper details the design, capabilities and architecture of GNCSIM. It showcases GNCSIM's ability to serve both as a test bed for early-stage control and navigation algorithms, allowing users to iterate over ideas based on simulation data, and as a validation framework for mature algorithms. Its unique ability to combine a high-fidelity six-degrees-of-freedom closed-loop rocket simulation with a modular and flexible architecture makes it the tool of choice for student rocketry teams seeking to develop actively controlled rockets. Within the Georgia Tech Guidance, Navigation and Control team, it is already actively used for the simulation of three distinct rockets, demonstrating its versatility and ease of use. While early results show promising performance and extended capabilities, only real-flight data will prove GNCSIM's reliability in the long run. Future work will focus on creating native support for sensor calibration and system identification, as well as enabling hardware-in-the-loop testing for late-stage design validation.

References

[1] Ceotto, G. H., Schmitt, R. N., Alves, G. F., Pezente, L. A., and Carmo, B. S., "RocketPy: Six Degree-of-Freedom Rocket Trajectory Simulator," *Journal of Aerospace Engineering*, Vol. 34, No. 6, 2021. https://doi.org/10.1061/(ASCE)AS.1943-5525.0001331.

[2] Niskanen, S., "Development of an Open Source Model Rocket Simulation Software," Master's thesis, Helsinki University of Technology, 2009. URL https://openrocket.info/documentation.html.

[3] Rogers, C. E., and Cooper, D., "RASAero II: Aerodynamic Analysis and Flight Simulation Software," Rogers Aeroscience, 2019. URL https://www.rasaero.com.

[4] Reynolds, D. P., Caillet, K., Foley, C., Swami, A., and Chiva Ruiz, A., "Design, Simulation and Testing of a Linear Quadratic Regulator for Attitude Control of a Fin-Tab Rocket," *AIAA Student Conference*, 2026. Submitted.

[5] NOAA Chemical Sciences Laboratory, "LLLJP Wind Shear Formula (Power Law)," https://csl.noaa.gov/projects/lamar/windshearformula.html, 2025. Lamar Low-Level Jet Program (LLLJP). Accessed: 2026-02-26.

[6] NOAA, NASA, and USAF, "U.S. Standard Atmosphere, 1976," Tech. Rep. NOAA-S/T 76-1562, U.S. Government Printing Office, Washington, D.C., 1976.

[7] Kelly, M., "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, Vol. 59, No. 4, 2017, pp. 849–904. https://doi.org/10.1137/16M1062569.

[8] Kelly, M., "OptimTraj - Trajectory Optimization for Matlab," , 2017. URL https://github.com/MatthewPeterKelly/OptimTraj.

[9] Groves, P. D., *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, 2nd ed., Artech House, Norwood, MA, USA, 2013.

[10] Markley, F. L., and Crassidis, J. L., *Fundamentals of Spacecraft Attitude Determination and Control*, Springer New York, New York, NY, USA, 2014.