Closed Loop Analysis of Space Systems (CLASS): A Modular Test System for CubeSat Development

Marc Akiki, Michael Lembeck
University of Illinois at Urbana-Champaign
104 S. Wright St. Urbana, Illinois, 61801
makiki2@illinois.edu

ABSTRACT

Closed Loop Analysis of Space Systems (CLASS) is a small satellite test system engineered and maintained by the Laboratory for Advanced Space Systems at Illinois (LASSI), which is affiliated with the Department of Aerospace Engineering at the University of Illinois at Urbana-Champaign. CLASS is designed to be modular and user-friendly while providing the capability to perform accurate and reliable closed-loop tests. Commercial and academic CubeSat developers struggle to implement adequate testing procedures because of the relatively short development timeline and the sophisticated and inaccessible nature of space systems test equipment. Hardware-in-the-loop testing offers a convenient "test-as-you-fly, fly-as-you-test" validation and verification option for CubeSats. CLASS features a customizable real-time satellite orbital mechanics and rigid body dynamics simulation programmed to execute on the widely used Raspberry Pi 4 (RPi4). The RPi4 can be successfully interfaced with numerous hardware elements including the satellite's actual flight computer, sensors, and actuators. In case certain flight hardware components, such as the magnetometers and gyroscopes, are not available for testing, CLASS offers the option of using Arduino boards that are programmed to emulate satellite sensors and actuators. Using CLASS, closed-loop tests on the Attitude Determination and Control System of CAPSat, a LASSI 3U CubeSat, proved to be critical for the validation of a state feedback Earth-pointing controller.

INTRODUCTION

Since their establishment in 1999, CubeSats have revolutionized accessibility to space. In a period of approximately twenty years, more than one thousand nanosatellites have been launched internationally.[1] The modularity and affordability of these standardized small satellites has encouraged universities to develop their own CubeSat missions. This provides students with the unique opportunity to take part in a hands-on engineering experience in the discipline of space systems. Academic institutions developed the majority of the CubeSats launched before 2013.[2,3] As these small satellites started to support more sophisticated payloads and mission objectives, the commercial space industry overtook universities to become today's top producers of CubeSats (as of 2019, 57% of launched small satellites were developed by commercial space companies).[1]

Despite the increased reliance on small satellites for sending payloads into orbit, the mission success rate remains undesirably low. Academic CubeSat developers register a 45% success rate and commercial CubeSat developers register a 77% success rate.[4] The Electrical Power System (EPS), On-Board Computer (OBC), and Communication System (COM) are the three satellite subsystems that are regularly flagged as the main cause of mission failure.[5] But on many occasions, it is extremely difficult to determine what went wrong after deployment, even if communication was established. In fact, 33% of CubeSat developers that have experienced a mission failure after deployment declare that they have no evidence to help them identify what subsystem did not function properly.[5] While numerous failure scenarios can be hard or impossible to avoid, a satellite bus design can be improved by enhancing the accuracy, breadth, and frequency of testing during development.[6]

Twenty-three small satellite developers from industry and government, as part of a survey organized by leading companies in aerospace, have all highlighted the significant correlation between the quality of pre-launch system verification and validation and their mission success rate.[4] The survey results illustrate that many CubeSat developers struggle to execute comprehensive interface tests and software/hardware-in-the-loop tests. Small CubeSat developers lack the expensive and advanced testing equipment and the launch deadlines are placing many time constraints. This problem is especially eminent with universities[4].

Small to medium size developers have to resort to less sophisticated testing equipment, such as building their own emulators and simulators. Hardware-in-the-loop testing has proven to be an effective means of identifying critical design features such as resolution errors, hardware lag, and signal noise.[7] It is a common testing procedure for the Attitude Determination and Control System (ADCS).[8] Mainly because ADCS invovles the incorporation of a satellite dynamic model along with numerous sensors, actuators, and flight computers. Software and hardware fault injection is also gaining popularity in the verification and validation of CubeSats.[9]

While there are accessible and feasible verification and validation techniques for CubeSats, many of these techniques require tedious customization of the test setup. An accurate, modular, and easily customizable test setup would decrease the time and effort placed into creating test procedures and increase the satellite system's reliability.

OBJECTIVE

System Features

Small satellite developers are challenged by the general lack of test time, low test fidelity, and the high costs associated with spacecraft test equipment. To address this issue, the Laboratory for Advanced Space Systems at Illinois (LASSI) has developed the Closed Loop Analysis of Space Systems (CLASS), a modular satellite test system. The initial engineering development phase of CLASS is documented in detail.[10]

The test system provides real-time satellite attitude and orbital mechanics simulations to enable hardware-in-the-loop testing. The simulations are easily configurable and meet the requirements for various types of missions and orbits. Academic CubeSat developers rarely have access to advanced testing equipment and are sometimes forced to build their own custom simulations. An accurate general-purpose customizable simulation setup reduces the time and effort placed into building custom-made simulations.

The test system is modular. It offers the option of completing a comprehensive systems test or to test specific individual components or subsystems independently without causing interface issues. For example, the user is able to test the Electric Power System (EPS) with or without the Attitude Determination and Control System (ADCS). If tested together, then the ADCS collects information about available power from EPS and sends information about solar panel orientations to the EPS. If the EPS is tested individually, then, an emulation of the ADCS is made available within the test system's software.

CLASS relies on widely used and well documented processors and programming languages. This requirement makes the system compatible with different satellite hardware configurations and software. It also simplifies the system architecture and makes it user friendly.

The test system minimizes numerical inaccuracies and eliminates asynchronous executions. If the hardware being tested becomes asynchronous with the test system, accuracy is compromised. Within its design, CLASS protects against these issues.

Design

CLASS includes an accurate easily configurable real-time simulation of satellite rigid body dynamics and orbital mechanics. The present physical state of the satellite (i.e., attitude, orbital position, magnetic field vector etc.) can be extracted from the simulation for display to the operator. Actuator commands generated by control algorithms in the satellite under test (i.e., torque commands, propulsion commands etc.) are collected and integrated in the simulation in real-time. The architecture implemented by CLASS is represented in Figure 1.

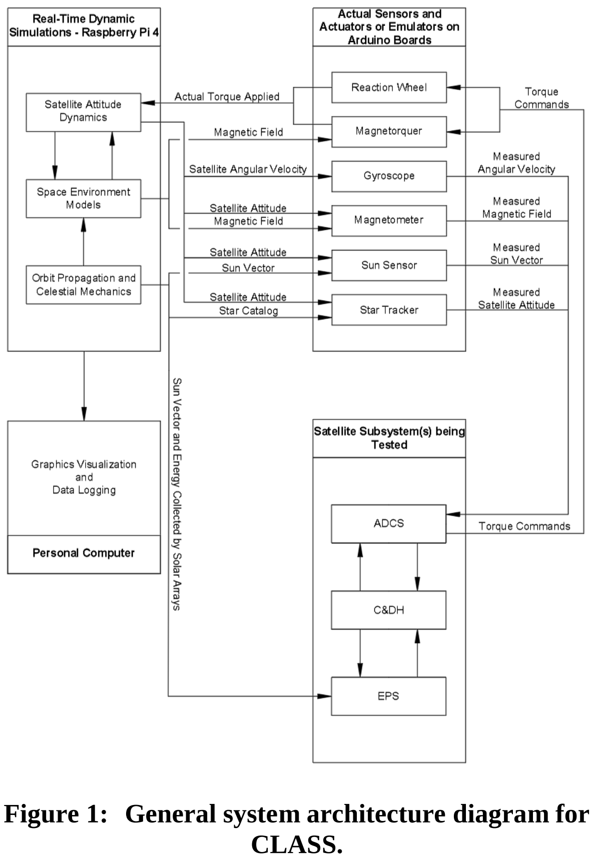
Real-Time Dynamic Simulations - Raspberry Pi 4
Actual Sensors and Actuators or Emulators on Arduino Boards
Satellite Attitude Dynamics
Space Environment Models
Orbit Propagation and Celestial Mechanics
Actual Torque Applied
Magnetic Field
Satellite Angular Velocity
Satellite Attitude
Magnetic Field
Satellite Attitude
Sun Vector
Satellite Attitude
Star Catalog
Reaction Wheel
Magnetorquer
Gyroscope
Magnetometer
Sun Sensor
Star Tracker
Torque Commands
Measured Angular Velocity
Measured Magnetic Field
Measured Sun Vector
Measured Satellite Attitude
Sun Vector and Energy Collected by Solar Arrays
Graphics Visualization and Data Logging
Personal Computer
Satellite Subsystem(s) being Tested
ADCS
C&DH
EPS
Torque Commands

Figure 1: General system architecture diagram for CLASS.

CLASS supports multiple test configurations. The dynamic simulation programmed in CLASS provides environmental data that can be used to create analog environments (e.g., magnetic fields) or translated into emulated satellite sensor outputs (e.g., gyro rates). In ground tests, the "static" outputs produced by sensors are replaced by the CLASS simulated outputs for the satellite flight software. For instance, the simulated instantaneous magnetic field vector value generated by CLASS can be sent to a Helmholtz Cage controller. The controller then commands the Helmholtz Cage to generate the magnetic field which is sensed by the satellite magnetometer placed inside of the cage. Attitude control software receives these sensor inputs, computes a control command, and commands the actuator to control the spacecraft attitude. The loop is closed when the commanded magnetorquer currents are received by the dynamics simulation, transformed into the corresponding torque values, and applied to the satellite dynamics. Analogously, a gyroscope can be setup in the same way using an air bearing table, a sun sensor can be coupled to a sun simulator, and a star tracker can be installed facing a wide-view display video of the relative star positions commanded by the dynamic simulation.

The microprocessor selected for the CLASS dynamic simulation is the Raspberry Pi (RPi) 4. Written in C++, the simulation software runs in real-time on the RPi 4 using the Linux patch PREEMPT_RT. The patch gives the RPi 4 the capability of executing tasks similar to a Real-Time Operating System. The easy access to RPi4 documentation as well as the processor's compatibility with various hardware and communication protocols make CLASS an easily configurable, compatible, and reliable test system.

If the actual sensor is not ready (as is often the case early in a project life cycle) to be included in a hardware-in-the-loop simulation, CLASS offers the option of implementing a customizable sensor emulator. Magnetometer and gyroscope emulators were developed on Arduino boards. The sensor emulators read the current attitude and orbit information from the dynamic simulation, add sensor noise and inaccuracies to it, apply the appropriate hardware timing delays, and then send the emulated sensor raw data to the flight computer.

As for actuator commands, depending on the configuration, the satellite attitude control system can command actual magnetorquers or reaction wheels in response to dynamic inputs. Such a configuration allows for testing satellite hardware performance and verifying response times. For example, test software in the satellite intercepts the digital torque commands being sent to the magnetorquers and forwards it to CLASS so that they can be integrated in the simulation.

SIMULATION ALGORITHMS

Frames of Reference

Multiple coordinate frames of reference are used in CLASS. The user can input any vector or matrix quantity using any of the following reference frames. CLASS then automatically converts the quantities into the appropriate reference frame for calculations and the display of results.

The satellite body frame is primarily defined by the user. The default body frame in CLASS is defined according to the CubeSat Design Specifications (CS) Rev. 13[11]. The origin of the coordinate frame is coincident with the center of mass of the CubeSat. A 3U CubeSat has its negative z-axis on the face that contains the deployment switch for the Poly-Picosatellite Orbital Deployer (P-POD) and the positive x-axis on the face that has the Access Ports in the P-POD. If desirable the user can alter the location of the body frame's origin and change the direction of the Cartesian axes.

The inertial reference frame used in class is the Earth Centered Inertial (ECI) J2000 frame. The *x-y* plane of the ECI J2000 frame is coplanar with the Earth's mean equator at midnight on the first day of January 2000. The positive x-axis points to the vernal equinox in the year 2000 and the positive z-axis points in the same direction as the celestial North Pole (or the Earth's spin axis).[12]

Another useful Earth centered reference frame in CLASS is the Earth Centered/Earth Fixed Frame (ECEF). It also has the *x-y* plane coplanar with the ECI J2000 frame and the positive z-axis along the Earth's spin axis. But, the ECEF frame rotates with the Earth making it convenient for certain types of analysis.

The Local Vertical/Local Horizontal (LVLH) frame is widely used for Earth pointing satellites because it maintains its z-axis nadir-pointing. As a result, it was also included in CLASS. The origin of the LVLH frame is located at the center of mass of the satellite. The basis vectors forming the coordinate frame are calculated by CLASS each time the satellite's orbital position is updated. The positive x-axis is along the satellite's orbital velocity vector in the inertial frame of reference. The positive y-axis is opposite to the normal vector of the orbit plane (i.e., opposite to the orbit angular momentum vector expressed in the inertial frame of reference). Finally, the z-axis is along the nadir-vector.

CLASS also includes the option of expressing the satellite position using Latitude, Longitude and Altitude (LLA). The spherical mean reference Earth radius (6371.2 km) is used to calculate altitude.

All conversions between the before mentioned reference frames are programmed in CLASS, offering the user the option of choosing the desired frame of reference when inputting parameters anytime during a test session.

Attitude Kinematics and Dynamics

Small satellites can be treated as rigid bodies.[13] Quaternions are used to calculate attitude in CLASS. All equation quantities that are expressed in **Bold** are vectors or matrices.

The first element of the quaternion vector is the scalar:

$$\mathbf{q_b^I} = [q_1 \; q_2 \; q_3 \; q_4]^T \tag{1}$$

The subscript **b** and superscript **I** indicate that the quaternion in Equation 1 represents the rotation from the body frame to the inertial frame. The corresponding rotation matrix in Equation 5 is populated accordingly:

$$\mathbf{R_{b_x}^I} = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 \\ 2(q_2 q_3 - q_1 q_4) \\ 2(q_2 q_4 + q_1 q_3) \end{bmatrix} \tag{2}$$

$$\mathbf{R_{b_y}^I} = \begin{bmatrix} 2(q_2 q_3 + q_1 q_4) \\ q_1^2 - q_2^2 + q_3^2 - q_4^2 \\ 2(q_3 q_4 - q_1 q_2) \end{bmatrix} \tag{3}$$

$$\mathbf{R_{b_z}^I} = \begin{bmatrix} 2(q_2 q_4 - q_1 q_3) \\ 2(q_3 q_4 + q_1 q_2) \\ q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix} \tag{4}$$

$$\mathbf{R_b^I} = \left[\mathbf{R_{b_x}^I} \quad \mathbf{R_{b_y}^I} \quad \mathbf{R_{b_z}^I}\right] \tag{5}$$

The kinematic equation for the quaternion vector is:[14]

$$\mathbf{\dot{q}_b^I} = \frac{1}{2}\mathbf{E^b}\left(\mathbf{q_b^I}\right)\boldsymbol{\omega}^{\mathbf{b}} \tag{6}$$

Where $\mathbf{\dot{q}_b^I}$ = time derivative of the quaternion vector; $\boldsymbol{\omega}^{\mathbf{b}}$ = satellite angular velocity expressed in the body frame and $\mathbf{E^b}\left(\mathbf{q_b^I}\right)$ is a matrix populated using Equation 7:

$$\mathbf{E^b}\left(\mathbf{q_b^I}\right) = \begin{bmatrix} -q_2 & -q_3 & -q_4 \\ q_1 & -q_4 & q_3 \\ q_4 & q_1 & -q_2 \\ -q_3 & q_2 & q_1 \end{bmatrix} \tag{7}$$

As for the attitude dynamics, the differential equation governing rotational rigid body motion is Euler's three-dimensional rotational equation:

$$\boldsymbol{\dot{\omega}}^{\mathbf{b}} = \mathbf{J^{b}}^{-1}\left[\mathbf{T_{ext}^b} - \boldsymbol{\omega}^{\mathbf{b}} \times \left(\mathbf{J^b}\boldsymbol{\omega}^{\mathbf{b}}\right)\right] \tag{8}$$

Where $\boldsymbol{\dot{\omega}}^{\mathbf{b}}$ = time derivative of the satellite angular velocity expressed in the body frame; $\mathbf{J^b}$ = satellite moment of inertia matrix expressed in the body frame; $\mathbf{T_{ext}^b}$ = external torques applied on the satellite expressed in the body frame.

Orbit Propagator

CLASS implements the Simplified General Perturbation 4 (SGP4) orbit propagator to calculate the orbital position and velocity of the satellite. The SGP analytical model was originally developed in the 1960s and since then underwent various modifications and updates. The simplified version of the SGP4 propagator used in CLASS was published by Lane and Cranford in 1970.[15] The SGP4 model was selected because it is widely used for Low Earth Orbit simulations, it has accessible documentation, and it has a satisfactory accuracy to run-time ratio.[15,16]

To initialize the simulation, the SGP4 algorithm determines the epoch orbital elements using a Two Line-Element (TLE) file. TLE files follow a standardized format for expressing the actual orbital properties of a satellite measured by the North American Aerospace Defense Command (NORAD). NORAD regularly provides the TLE file for orbiting satellites. Fundamentally, after setting the epoch elements, the SGP4 model integrates the equations of motion and compares the orbit state estimate with observations using the Least Squares Method and solving the Jacobian. When convergence is reached during every iteration, CLASS outputs the position and velocity of the satellite in real-time expressed in the ECI frame.

However, there are some limitations to the use of the SGP4 caused by the numerical inaccuracies accumulated due to long simulation times. When implemented as an on-board or ground station propagator it is recommended to frequently update the TLE file used in the algorithm to correct the error in orbit determination. But, for simulation times of one to two weeks, the numerical error remains acceptable. Nonetheless, more orbit propagators will be added to CLASS in the near future.

Magnetic Field Model

CLASS relies on the International Geomagnetic Reference Field (IGRF) model to calculate the Earth magnetic field at each orbital position. The IGRF model is updated every five years. The current version for 2020-2025 is the thirteenth. CLASS has both the 12th and 13th version programmed into it. All equations are programmed into the CLASS software in C++ for fast execution.

Aerodynamic Disturbance Torque

In Low Earth Orbit, the three most significant environmental disturbances are atmospheric drag, gravity gradient torques, and solar radiation pressure.[17] The aerodynamic torque is the most dominant of the three.

CLASS calculates the aerodynamic disturbance on a satellite by computing the aerodynamic drag force on each surface expressed in the body frame:[12]

$$\mathbf{F}^{\mathrm{b}}_{\mathbf{areo\ i}} = -\frac{1}{2}\rho C_D v^{\mathrm{b}}_{\mathrm{rel}} \mathbf{v}^{\mathbf{b}}_{\mathbf{rel}} S_i \max(\cos\theta_i, 0) \quad (9)$$

Where $\mathbf{F}^{\mathrm{b}}_{\mathbf{areo\ i}}$ = drag force on the ith plate of the satellite expressed in the body frame; ρ = air density; C_D = drag coefficient of the CubeSat (usually between 1.5 and 2.5); $\mathbf{v}^{\mathbf{b}}_{\mathbf{rel}}$ = the relative velocity of the satellite with respect to the Earth (i.e., taking into account the Earth's rotation); S_i = surface area of the ith plate; $\max(\cos\theta_i, 0)$ = the maximum value between either zero or the cosine of the angle theta which is the angle between the normal vector to the ith plate and the relative velocity vector in the body frame. The angle θ_i is calculated using:

$$\theta_i = \frac{\left(\mathbf{n}^{\mathbf{b}}_{\mathbf{i}} \cdot \mathbf{v}^{\mathbf{b}}_{\mathbf{rel}}\right)}{||\mathbf{v}_{\mathbf{rel}}||} \quad (10)$$

Where $\mathbf{n}^{\mathbf{b}}_{\mathbf{i}}$ = outward normal vector to the ith plate expressed in the body frame.

The aerodynamic disturbance torque on the satellite is then calculated using Equation 11:

$$\mathbf{T}^{\mathrm{b}}_{\mathbf{aero}} = \sum_{i=1}^{N} \mathbf{r}^{\mathbf{b}}_{\mathbf{cm_{cp}i}} \times \mathbf{F}^{\mathbf{b}}_{\mathbf{aero\ i}} \quad (11)$$

Where $\mathbf{T}^{\mathrm{b}}_{\mathbf{aero}}$ = total aerodynamic torque on the satellite expressed in the body frame; $\mathbf{r}^{\mathbf{b}}_{\mathbf{cm_{cp}i}}$ = vector between the satellite's center of mass and the center of pressure of the ith plate.

Gravity Gradient Torque

Any nonsymmetrical body in a non-uniform gravitational field will experience a disturbance torque caused by the gravity-gradient across the body.[12] CLASS computes the gravity gradient torque every time the satellite attitude is updated. Using the inverse square gravitational field:

$$\mathbf{g}(\mathbf{r}) = -\frac{\mu_{\mathrm{Earth}}}{r^3}\mathbf{r} \quad (12)$$

Where $\mathbf{g}(\mathbf{r})$ = Earth gravitational field as a function of the vector $\mathbf{r}$; $\mathbf{r}$ is the distance vector between the object of interest and the Earth's center; μ_{Earth} = Earth's standard gravitational parameter.

The gravity gradient torque can now be expressed using Equation 13:

$$\mathbf{T}^{\mathbf{b}}_{\mathbf{gg}} = \frac{3\mu_{\mathrm{Earth}}}{r^3}\left[\hat{\mathbf{n}}^{\mathbf{b}} \times \left(\mathbf{J}^{\mathbf{b}}\hat{\mathbf{n}}^{\mathbf{b}}\right)\right] \quad (13)$$

Where $\mathbf{T}^{\mathbf{b}}_{\mathbf{gg}}$ = gravity gradient torque on the satellite expressed in the body frame; $\hat{\mathbf{n}}^{\mathbf{b}}$ = unit nadir vector expressed in the body frame.

Solar Radiation Pressure

CLASS currently does not support Solar Radiation Pressure (SRP) calculations yet. However, the code infrastructure for SRP calculations is already uploaded in the CLASS software. Equation 14 is used to calculate the force on the ith plate caused by SRP:

$$\mathbf{F}^{\mathbf{b}}_{\mathbf{SRP\ i}} = P_{SRP} S_i \left[2\left(\frac{R_{\mathrm{diff\ i}}}{3} + R_{\mathrm{spec\ i}}\cos\theta_{\mathrm{SRP\ i}}\right)\mathbf{n}^{\mathbf{b}}_{\mathbf{i}} + \left(1 - R_{\mathrm{spec\ i}}\right)\mathbf{s}^{\mathbf{b}}\right]\max(\cos\theta_{\mathrm{SRP\ i}}, 0) \quad (14)$$

Where $\mathbf{F}^{\mathbf{b}}_{\mathbf{SRP\ i}}$ = force on the ith plate due to SRP expressed in the body frame; $\mathbf{s}^{\mathbf{b}}$ = unit Sun vector with respect to the satellite's center of mass; $R_{\mathrm{spec\ i}}$ = specular reflection coefficient; $R_{\mathrm{diff\ i}}$ = diffuse reflection coefficient; $\theta_{\mathrm{SRP\ i}}$ = angle between the normal vector $\mathbf{n}^{\mathbf{b}}_{\mathbf{i}}$ to the ith plate and the Sun vector $\mathbf{s}^{\mathbf{b}}$; P_{SRP} = solar radiation pressure calculated using:

$$P_{SRP} = \frac{\mathcal{F}}{c\,(r_{\mathrm{sat\ to\ sun}})^2} \quad (15)$$

Where $\mathcal{F} = 1{,}366\ \mathrm{W/m^2}$ solar constant defined as the flux density of solar irradiance received at 1 AU; c = $299{,}792{,}458\ \mathrm{m/s}$ speed of light; $r_{\mathrm{sat\ to\ sun}}$ = distance between the satellite and the Sun.

REAL-TIME SIMULATIONS

Software Task Scheduling

CLASS has the ability to perform satellite attitude dynamics and orbital mechanics simulations in real-time on a Raspberry Pi 4 (RPi4). The Linux Raspbian operating system is used. In its native configuration, Raspbian does not support real-time applications. To overcome this shortcoming, the Linux patch PREEMPT_RT was installed to provide the RPi4 the capability of emulating a Real-Time Operating System (RTOS). The execution model applied by PREEMPT_RT is reliant on POXIS threads or "pthreads". According to the *IEEE 1003.1c-1995* standard, each computational task is labeled as a thread and the POSIX Threads API manages the threads to execute according to the commanded scheduling policy.[18]

The scheduling policy used by the dynamic simulation in CLASS is SCHED_DEADLINE. This CPU scheduler guarantees that each satellite dynamics calculation is executed within a strict time interval. Avoiding cycle-slips and the desynchronization of the dynamic simulation with the satellite system being tested is central for achieving accurate results.

Calculating the satellite attitude is the task of highest priority. It is performed every five milliseconds. A new quaternion representing the rotation between the satellite body frame and the ECI frame is generated along with the satellite angular velocity and the magnetic field vector expressed in the body frame. In case the code is not able to calculate the new attitude before the five-millisecond deadline, the program overrides the task and switches to the next task of highest priority to ensure that no cycle-slips occur. But, in all test runs performed so far, the algorithm experienced no lag in calculating a new attitude within each time-step. After the satellite attitude is updated, the program switches tasks to calculate the new orbital position and velocity. The time-step for each orbital mechanics update is sixty seconds. Other environmental factors that depend on the orbital position and velocity are also calculated (such as the atmospheric density, magnetic field, sun's position etc.). The third and final thread is the idle task. Figure 2 illustrates the program execution time-steps and task flow for the CLASS satellite dynamics simulation software in real-time. The user has to input an initial quaternion, satellite angular velocity, an initial orbit TLE file, and the desired simulation parameters (run time, delays etc.) to start the simulation.

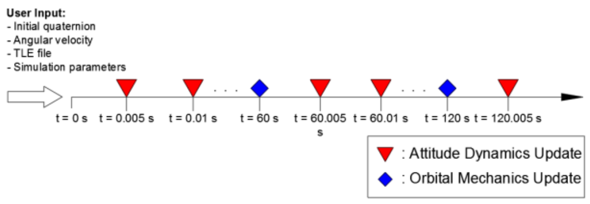

Figure 2: Task Flow Schematic for the CLASS Dynamics Simulation Software.

Attitude Dynamics Simulation

Satellite attitude is calculated by CLASS using Equations 6 and 8 and the Runge-Kutta 4 integration method. To validate the satellite attitude dynamics calculations, a 300-minute simulation was run on CLASS. The CubeSat used in the following demonstration has a moment of inertia equal to:

$$\mathbf{J^b} = \begin{bmatrix} 0.0275 & 0 & 0 \\ 0 & 0.04 & 0 \\ 0 & 0 & 0.0075 \end{bmatrix} \mathrm{kg.m^2} \quad (16)$$

In this demonstration, the CubeSat starts by rotating at a constant rate equal to 1.879 deg/s for 100 minutes. The initial conditions are:

$$\mathbf{q_b^I} = [0.836 \quad -0.029 \quad -0.52 \quad -0.173]^{\mathrm{T}} \quad (17)$$

$$\boldsymbol{\omega}^{\mathbf{b}} = [0.9 \quad -0.4 \quad -1.6]^{\mathrm{T}} \ \mathrm{deg/s} \quad (18)$$

$$\mathbf{T^b} = [0 \quad 0 \quad 0]^{\mathrm{T}} \ \mathrm{N.m} \quad (19)$$

Because no external torques are applied yet, the angular momentum expressed in the inertial frame should remain constant. After 100 minutes, an impulse torque, Equation 20, is applied on the CubeSat:

$$\mathbf{T^I} = [0.02 \quad -0.001 \quad 0]^{\mathrm{T}} \ \mathrm{N.m} \quad (20)$$

The impulse torque is expressed in the inertial reference frame. So, theoretically, in the inertial reference frame, the angular momentum along the positive x-direction will increase, the angular momentum along the negative y-direction will increase, and the angular momentum along the z-direction will remain the same. Figure 3 and Figure 4 illustrate the satellite's body to inertial frame quaternion vector and the angular velocity vector in the body frame vs time, respectively.

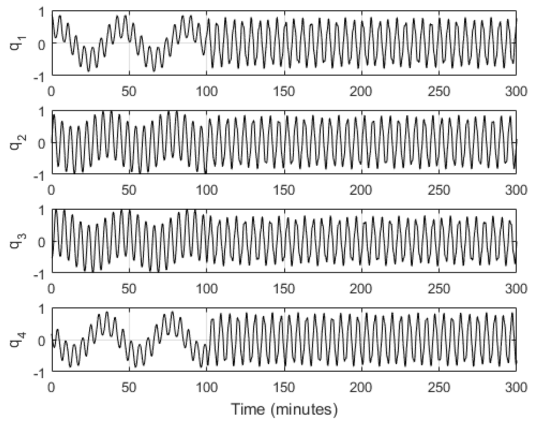

Figure 3: Simulation results for a spinning satellite in CLASS subject to an impulsive torque at time 100 minutes. Quaternion vector elements for body frame to inertial frame vs time (minutes).

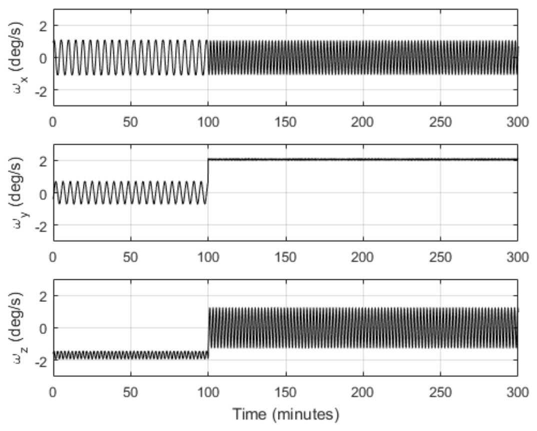

Figure 4: Simulation results for a spinning satellite in CLASS subject to an impulsive torque at time 100 minutes. Angular velocity (deg/s) vector elements expressed in body frame vs time (minutes).

Illustrated in Figure 5 is the angular momentum of the satellite in the inertial frame vs time. During the first 100 minutes, the angular momentum remained constant. At time 100 minutes, the x-component increased in the positive direction, the y-component increased in the negative direction, and the z-component remained unperturbed.

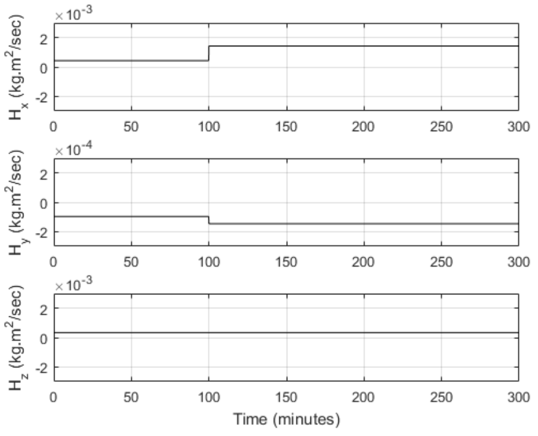

Figure 5: Simulation results for a spinning satellite in CLASS subject to an impulsive torque at time 100 minutes. Angular momentum (kg.m^2/s) vector elements expressed in the inertial frame vs time (minutes).

The angular momentum in the first 100 minutes should be equal to:

$$\mathbf{H^b} = \mathbf{J^b}\boldsymbol{\omega}^\mathbf{b} \tag{21}$$

$$\mathbf{H^I} = \mathbf{R^I_b}\mathbf{H^b} \tag{22}$$

The calculated expected values for Equation 22 are compared to the simulated values from Figure 5 in Table 1. The results match and the angular momentum remained conserved in the first 100 minutes.

Table 1: Comparison of the expected angular momentum value in the first 100 minutes to the simulated values.

Angular Momentum Vector Component	Expected Value before perturbation (kg.m^2/s)	Simulated Value before perturbation (kg.m^2/s)	Error (%)
H_x	0.0004291	0.0004291	0
H_y	-0.0000966	-0.00009657	0.031
H_z	0.0003392	0.00034	0.235

Table 2 confirms that the angular momentum did change as expected after 100 minutes simulation time.

Table 2: Change in the value of the angular momentum after the impulse torque is applied.

Angular Momentum Vector Component	Value before perturbation (kg.m^2/s)	Value after perturbation (kg.m^2/s)
H_x	0.0004291	0.001429
H_y	-0.00009657	-0.00001459
H_z	0.00034	0.00034

Orbit Propagation

To validate the orbit simulation that runs alongside the attitude simulation in real-time, a test run using a TLE file, obtained online, for the International Space Station (ISS) was performed. The orbit was propagated for 300 minutes (i.e., about three orbits and a third).

The results indicate an apogee of 6807 km (altitude of 429 km) and a perigee of 6790 km (altitude of 412 km). The orbital altitude is plotted vs time in Figure 6. The results are in accordance with documented orbits of the ISS at the date and time of the simulation.

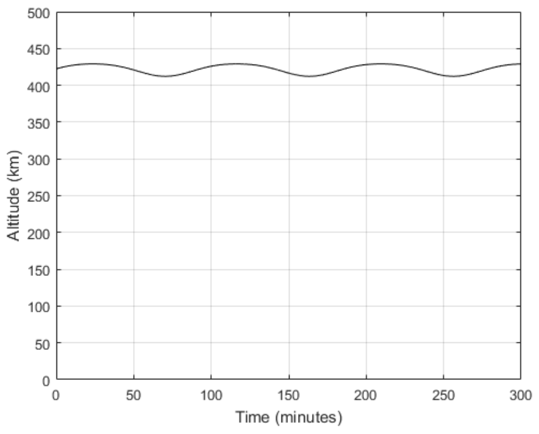

Figure 6: Simulation results for the ISS orbit using CLASS. The ISS orbit altitude (km) vs time (minutes).

The simulated orbital period is 92.83 minutes, which is in accordance with the expected ISS orbit period.[19] The simulated apogee and perigee velocities agree with hand-calculated results tabulated in Table 3.

Table 3: Comparison of the expected apogee and perigee velocity values to the simulated values.

Velocity	Expected Value (km/s)	Simulated Value (km/s)	Error (%)
Apogee	7.653	7.645	0.106
Perigee	7.663	7.664	0.013

More validation can be performed by examining the specific angular momentum of the orbit. Gravitational, aerodynamic, and other celestial perturbations cause satellites to de-orbit. Thus, the angular momentum is not conserved for the orbit. But, for the relatively short orbit time interval of 300 minutes, the perturbations are negligible, and it is expected that the specific angular momentum remain constant.[20] The expected specific angular momentum value is calculated using:

$$h = \sqrt{\mu_{\mathrm{Earth}} a(1 - e_0^2)} = 52033.679\ \mathrm{km}^2/s \qquad (23)$$

Where h = specific angular momentum of the satellite orbit; a = semi-major axis of the ISS orbit directly extracted from the TLE file; e_0 = epoch eccentricity of the ISS orbit directly extracted from the TLE file.

The simulation calculates the specific angular momentum using:

$$\mathbf{h}^{\mathbf{I}} = \mathbf{r}^{\mathbf{I}} \times \mathbf{v}^{\mathbf{I}} \qquad (24)$$

Where $\mathbf{h}^{\mathbf{I}}$ = specific angular momentum vector of the satellite orbit expressed in the inertial frame; $\mathbf{r}^{\mathbf{I}}$ = instantaneous orbital position of the satellite calculated by the simulation expressed in the inertial frame; $\mathbf{v}^{\mathbf{I}}$ = instantaneous orbital velocity of the satellite calculated by the simulation expressed in the inertial frame.

As illustrated in Figure 7, the simulated specific angular momentum for the first three orbits remains constant at 52070 km^2/s. The error between the simulated value and the expected value in Equation 23 is 0.07%.

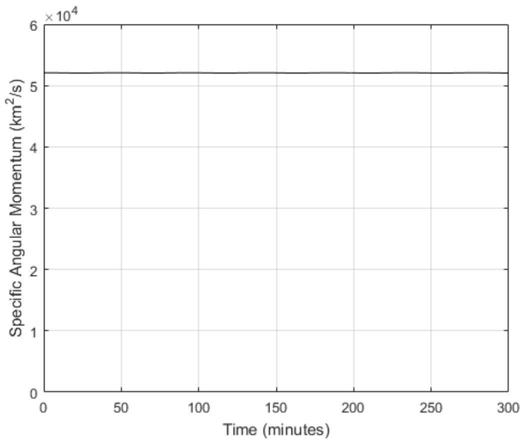

Figure 7: Simulation results for the ISS orbit using CLASS. The ISS specific angular momentum (km^2/s) vs time (minutes) remains constant at 52070 km^2/s.

CLOSED-LOOP TESTS

CLASS played a critical role during the development of CAPSat, a LASSI 3U satellite. CAPSat is due to launch in 2021 and has three payloads. The first payload is a radiator designed to experiment a new cooling technique. The second payload is a quantum annealing experiment prepared by the Physics Department at the University of Illinois. The third payload is a new satellite attitude pointing procedure performed by deforming the solar panels.

CAPSat's Attitude Determination and Control requirements are to de-tumble and then maintain the communication antenna Earth pointing. CLASS was used to run closed-loop simulations to test the ADCS design on CAPSat. While CLASS has the ability to directly interface with the CAPSat flight computer, using RS-422, and run a full systems test, the flight computer was not ready yet. So, an emulator of the ADCS flight computer was programmed inside the CLASS software to begin verification and validation early on during the development phase. The flight computer emulator sends the control torque commands to the dynamic simulation in CLASS, and the simulation, after applying the control torque, sends the current state of the satellite attitude and orbit to the flight computer emulator.

ADCS Emulation

The bus for CAPSat's ADCS has many hardware limitations that affect the control algorithm's performance. So, the emulator that was coded into CLASS was programmed to accurately incorporate hardware delays, communication delays, and sensor resolutions to achieve a more realistic and accurate closed-loop test.

The ADCS sensors are four three-axis magnetometers and four three-axis gyroscopes. The actuators for controlling the satellite's attitude are flat coil magnetorquers that were designed and built in the lab. When the flight computer emulation requests a sensor measurement, the CLASS dynamic simulation computes the magnetic field value and the satellite angular velocity and sends it to the emulator. Sensor noise is added as Gaussian Zero-Mean white noise. Sensor measurements are only performed with the magnetorquers off to avoid interference with the magnetometer readings. Also, measurement time delays are included. Each flight magnetometer measurement has been estimated to take about 0.4 seconds and each gyroscope measurement has been estimated to take about 0.2 seconds. CAPSat has only one serial port to handle ADCS sensor and command traffic. As a result, sensor measurements are picked up individually, resulting in a net sensor measurement time of 2.4 seconds. The emulator will only register measurements from the CLASS simulation that are greater or equal to the sensor resolutions.

The ADCS actuators are composed of six (i.e., two on each satellite face) single axis magnetorquers. The same single serial port that handles sensor data, also, commands all actuators. The time for each magnetorquer to receive and apply the commanded input current is measured to be 0.2 seconds. Consequently, the total time to turn on all actuators is 1.2 seconds. They are all kept on for 0.5 seconds and then are turned off one-by-one. So, another 1.2 seconds elapses as the magnetorquers are turned off.

The measurements and actuation cycles are reproduced about every seven seconds (i.e., the length of one cycle in the ADCS flight computer).

De-Tumbling

The first control requirement for CAPSat is to de-tumble or de-spin when deployed from the P-POD. The control algorithm used for de-tumbling is the *B-dot* control law:

$$\mathbf{u^b} = -\mathrm{K}\frac{\mathrm{d}\mathbf{B^b}}{\mathrm{dt}} \tag{25}$$

Where $\mathbf{u^b}$ = control input; K = proportional control gain; $\frac{\mathrm{d}\mathbf{B^b}}{\mathrm{dt}}$ = time derivative of the measured magnetic field vector expressed in the satellite body frame.

Earth-Pointing Controller

After a successful de-tumble (i.e., satellite angular velocity is almost zero), the controller that will point CAPSat's antenna along nadir is activated.

To determine the satellite's attitude, a seven state Extended Kalman Filter (EKF) was developed with the help of an on-board orbit propagator and attitude simulation. The EKF eliminates inaccuracies in measurement data and determines a quaternion representing a rotation from the satellite body frame to the inertial frame. Simultaneously, the on-board flight computer is calculating the attitude of the LVLH coordinate reference frame. The controller attempts to align the satellite body frame with the LVLH frame because the antenna is along the satellite body frame z-axis. Aligning the two frames will maintain the antenna Earth pointing because the LVLH frame z-axis is along the nadir vector.

The control law used is a state feedback control law:

$$\mathbf{u^b} = -\mathrm{k_p}\mathbf{q_b^{LVLH}}(\mathbf{2}\!:\!\mathbf{4}) - \mathrm{k_d}\boldsymbol{\omega}^{\mathbf{b}}_{\mathbf{b/r}} \tag{26}$$

Where $\mathbf{u^b}$ = control input; $\mathrm{k_p}$ = proportional control gain; $\mathrm{k_d}$ = derivative control gain; $\mathbf{q_b^{LVLH}}(\mathbf{2}\!:\!\mathbf{4})$ = vector containing the last three elements (the non-scalar) of the quaternion describing the rotation between the body frame and the LVLH frame; $\boldsymbol{\omega}^{\mathbf{b}}_{\mathbf{b/r}}$ = angular velocity of the body coordinate frame with respect to the LVLH coordinate frame expressed in the satellite body frame.

Results

The performance of the de-tumbling algorithm is displayed first. CAPSat, will be deployed from ISS altitude, so, the orbit propagator was initialized with orbital elements similar to that of the ISS (altitude of 400 km). A high spin rate of 9.6 deg/s is selected to be the satellite's initial rate. Such a high deployment spin rate is rarely reached, so, this test should provide a good understanding of the *B-dot* algorithm's capabilities. The initial conditions are:

$$\boldsymbol{\omega}^{\mathbf{b}} = [5.9 \quad 4.2 \quad -6.3]^{\mathrm{T}}\ \mathrm{deg/s} \tag{27}$$

$$|\boldsymbol{\omega}^{\mathbf{b}}| = 9.6\ \mathrm{deg/s} \tag{28}$$

$$\mathbf{q_b^I} = [0.24 \quad 0.189 \quad 0.917 \quad -0.258]^{\mathrm{T}} \tag{29}$$

The magnitude of the satellite angular velocity decrease from 9.6 deg/s to 0.065 deg/s in 117.2 minutes (i.e., almost after one orbit). The change of the angular velocity with time is illustrated in Figure 8. Dense oscillations in the values of the angular velocity occur when de-tumbling. They are the result of the hardware constraints that were discussed earlier: slow response of the actuators, the single serial port that causes time delays, and the under-actuated nature of magnetic torquers.

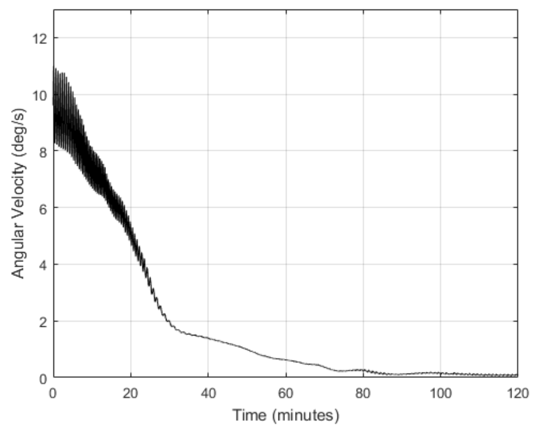

Figure 8: De-tumbling results for CAPSat. Satellite angular velocity (deg/s) vs time (minutes). Angular velocity reaches values near zero in less than one orbit. Oscillations are apparent because of hardware limitations.

Figure 9 illustrates the applied torque for an entire control cycle of seven seconds, arbitrarily selected at time $399\ \text{sec} < t < 406\ \text{sec}$. The torque is applied in steps. During the first 2.4 seconds, the torque is zero because sensor data is being read. Next, the magnetorquers are activated according to the following sequence: positive x-axis, positive y-axis, positive z-axis, negative x-axis, negative y-axis, and negative z-axis. Then, they are deactivated in the same sequence. This justifies why the torque seems to double after 0.6 seconds, keep a constant magnitude for 0.5 seconds, and go back to zero in steps. Regarding the small steps (the "wrinkles" that are most apparent in the y-component of the torque), they are caused by a physical property of magnetorquers. The torque applied by a magnetorquer is equal to:

$$\mathbf{T^b} = \boldsymbol{\mu}^\mathbf{b} \times \mathbf{B^b} \tag{30}$$

Where $\mathbf{T^b}$ = applied torque expressed in the body frame; $\boldsymbol{\mu}^\mathbf{b}$ = the magnetic moment vector expressed in the body frame; $\mathbf{B^b}$ = the magnetic field vector expressed in the body frame of the satellite.

The magnetic field vector seen by the satellite is continually varying while the satellite rotates. Consequently, for the same magnetic moment command from the ADCS, for a single cycle, the applied torque will not remain constant and will have these slight variations.

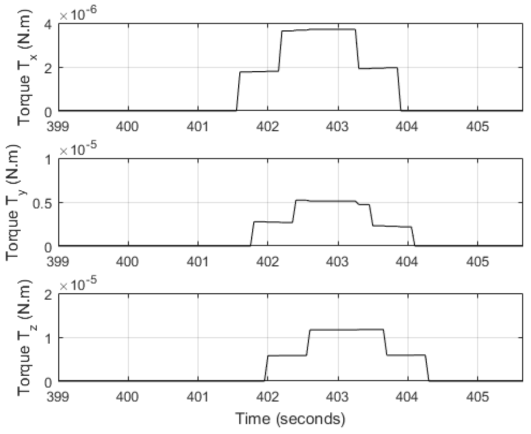

Figure 9: De-tumbling results for CAPSat. Applied torque (N.m) vs time (seconds) for one control cycle.

Error! Reference source not found. illustrates the commanded magnetic moment vector for the same control cycle as the one in Figure 9. The commanded magnetic moment remains constant per control cycle. If testing of the ADCS algorithm were to occur without taking into consideration the hardware delays and constraints discussed earlier (i.e., just a simple software simulation), then, the behavior of the applied torque in Figure 9, and consequently the behavior of the angular velocity in Figure 8 would not have appeared in the results.

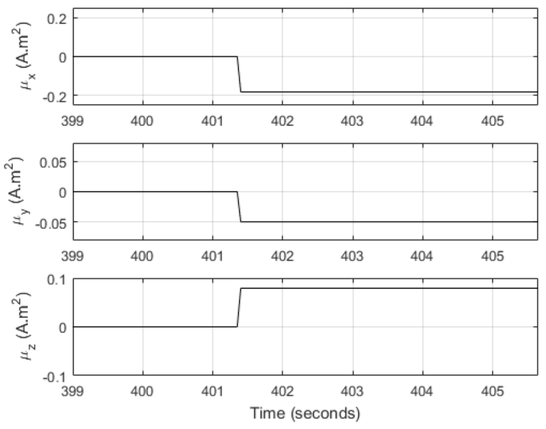

Figure 10: De-tumbling results for CAPSat. Commanded magnetic moment (A.m^2) vs time (seconds) for one control cycle.

After de-tumbling, the Earth pointing controller is activated. The arbitrary attitude reached after de-tumbling is:

$$\mathbf{q}_\mathbf{b}^\mathbf{I} = [0.427 \quad 0.468 \quad 0.137 \quad 0.762]^\mathrm{T} \qquad (31)$$

Which corresponds to a 113-degree angle between the satellite antenna and the nadir vector. The controller is able to decrease this angle difference to 0 ± 6 degrees in 142.2 minutes (a satisfactory pointing accuracy for CAPSat's antenna). Figure 11 illustrates the change of the angle between the antenna and the nadir vector vs time. Small oscillations are present, and they are also due to the hardware constraints and slow response.

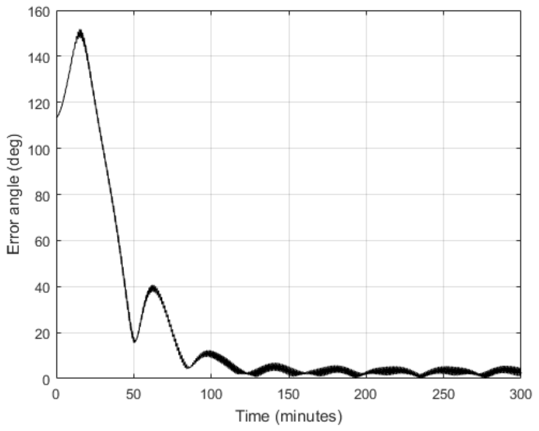

Figure 11: Earth-pointing results for CAPSat. Error angle (degrees) between the antenna and the nadir vector vs time (minutes).

In LEO, satellites have an almost circular orbit. Thus, the LVLH frame is rotating inertially at about the same angular velocity as the circular orbit angular velocity (or period):

$$\omega_\mathrm{circ} = \frac{360°}{\mathrm{T}_\mathrm{CAPSat}} = \frac{360°}{92.83} = 0.065 \text{ deg/s} \qquad (32)$$

To keep the antenna vector and the nadir vector aligned, the satellite must maintain an angular velocity almost equal to that of the LVLH frame, calculated in Equation 32. Figure 12 shows that the satellite angular velocity changed with time and settled at values around the 0.065 deg/sec mark. The oscillations are ± 0.005 (7.7%) in magnitude. The difficulty in strictly maintaining a steady state value equal to 0.065 deg/s stems from the presence of aerodynamic and gravity gradient disturbance torques coupled with the slow response of the magnetorquers.

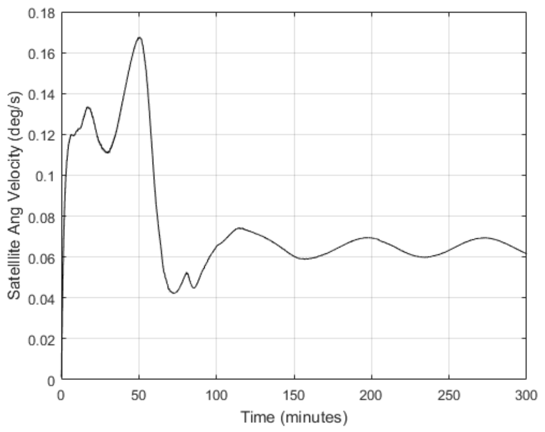

Figure 12: Earth-pointing results for CAPSat. Satellite angular velocity (deg/s) vs time (minutes).

Figure 13 illustrates the quaternion vector representing the rotation between the satellite body frame and the LVLH frame changing with time. For both frames to be aligned, the quaternion vector must reach a value of $[\pm 1 \quad 0 \quad 0 \quad 0]$. The desired steady state is reached in 142.2 minutes with a variation of $\pm 3\%$.

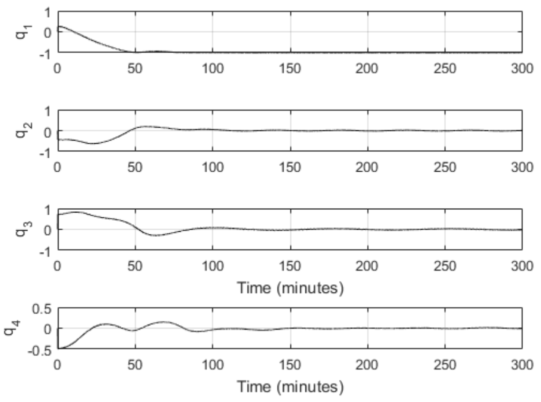

Figure 13: Earth-pointing results for CAPSat. Quaternion vector elements representing the rotation from the body frame to the LVLH frame vs time (minutes).

CLASS proved to be instrumental in identifying several software and hardware errors in the control algorithms previously developed for CAPSat. These errors include a sign flip within the algorithm, another sign flip in the installation of the magnetorquers, and insufficient magnetometer resolution.

CONCLUSION

The Laboratory for Advanced Space Systems at Illinois (LASSI) completed the initial engineering development stage for a modular satellite test system called: Closed

Loop Analysis of Space Systems (CLASS). CLASS is an easily configurable and reliable test system for CubeSat verification and validation. By providing an accurate real-time satellite attitude dynamics and orbital mechanics simulation on the easily configurable and widely used Raspberry Pi microprocessor, CLASS is increasing LASSI's ability to perform closed-loop tests of actual flight software and hardware. The aim of CLASS is to address the issue of high CubeSat mission failure rates in academia and industry. Its modularity cuts the time and resources spent on the design and customization of a simulation or test set-up for a specific subsystem. When test setups become very specific to the subsystem or mission they are testing, they are rarely used again or recycled in the future. Furthermore, CLASS offers the option of implementing customizable sensor emulators programmed on Arduino boards. Such emulators ensure that closed-loop test procedures, even early on during the development stage, include hardware constraints to maximize result reliability.

Critical errors in the previous software and hardware setup of the CAPSat, a LASSI satellite, attitude determination and control system were identified using CLASS. Identifying these errors early on during the development stage has saved time and resources and potentially might avoid a mission failure. The current and future LASSI missions rely on CLASS as an iterative tool for satellite bus design.

LASSI seeks to enhance CLASS's capabilities in the near future, starting off with a real-time visualization of the satellite's motion in orbit. The rapid and reliable execution of the dynamics simulation software allows the implementation of a live graphics visualization set-up on a separate personal computer. The user will be able to see the satellite respond to control commands and analyze the data in real-time. The addition of more sensor emulators is also desirable, such as sun sensors and star trackers. The lab is also including more dynamic and orbit models into the CLASS software.

References

1. T. Villela, C. A. Costa, A. M. Brandao, F. T. Bueno and R. Leonardi, "Towards the Thousandth CubeSat: A Statistical Overview," International Journal of Aerospace Engineering, vol. 2019, pp. 1-13, 2019.
2. W. J. Pang, B. Bo, X. Meng, X. Z. Yu, J. Guo and J. Zhou, "Boom of the CubeSat: A Statistics of CubeSats Launch in 2003-2015," Proceedings of the International Astronautical Congress, Guadalajara, 2016.
3. M. Swartwout, "A Statistical Survey of Rideshares (and Attack of the CubeSats, Part Deux)," Proceedings of the IEEE Aerospace Conference, Big Sky, 2012.
4. C. C. Venturini, M. Tolmasoff and R. Delos Santos, "Improving Mission Success of CubeSats," U.S. SPACE PROGRAM MISSION ASSURANCE IMPROVEMENT WORKSHOP, El Segundo, 2017.
5. M. Langer and J. Bouwmeester, "Reliability of CubeSats - Statistical Data, Developers' Beliefs and the Way Forward," Proceedings of the AIAA/USU Conference on Small Satellite, Logan, 2016.
6. A. Alanazi and J. Straub, "Statistical Analysis of CubeSat Mission Failure," Proceedings of the AIAA/USU Conference on Small Satellites, Logan, 2018.
7. S. Corpino and F. Stesina, "Verification of a CubeSat via Hardware-in-the-Loop Simulation," IEEE Transactions on Aerospace and Electronic Systems, vol. 50, no. 4, p. 2819, 2014.
8. J. Kiesbye, D. Messmann, M. Preisinger, G. Reina, D. Nagy, F. Schummer, M. Mostad, T. Kale and M. Langer, "Hardware-In-The-Loop and Software-In-The-Loop Testing of the MOVE-II CubeSat," Aerospace, vol. 6, no. 130, 2019.
9. C. L. G. Batista, A. C. Weller, E. Martins and F. Matiello-Francisco, "Towards increasing nanosatellite subsystem robustness," Acta Astronautica, vol. 156, pp. 187-196, 2019.
10. M. Akiki, "Closed Loop Analysis of Space Systems (CLASS) a Modular Test System for Small Satellite Verification and Validation," IDEALS, 2020
11. The CubeSat Program, "CubeSat Design Specification Rev 13," California Polytechnic State University.
12. F. L. Markley and J. L. Crassidis, Fundamentals of Spacecraft Attitude Determination and Control, Springer, New York, 2014.
13. W. Hu, Fundamental Spacecraft Dynamics and Control, John Wiley & Sons, Singapore, 2015.
14. NASA Mission Planning and Analysis Division, "Euler Angles, Quaternions, and Transformation Matrices," Lydon B. Johnson Space Center, Houston, 1977.
15. F. Hoots, P. W. Schumacher and R. A. Glover, "History of Analytical Orbit Modeling in the US Space Surveillance System," Journal of Guidance, Control, and Dynamics, vol. 27, no. 2, pp. 174-185, 2004.

16. D. Vallado and P. Crawford, "SGP4 Orbit Determination," Proceedings of the AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Honolulu, 2008

17. J. R. Wertz, Spacecraft Attitude Determination and Control, Kluwer Academic Publishers, Dordrecht, 1978.

18. IEEE, "POSIX Ada Language Interfaces," IEEE Standard for Information Technology, 1996.

19. NASA Johnson Space Center, "Reference Guide to the International Space Station," National Aeronautics and Space Admnistration, Houston, 2015.

20. J. E. Prussing and B. A. Conway, Orbital Mechanics, Oxford University Press, New York, 2013.