

© 2020 Marc Akiki

CLOSED LOOP ANALYSIS OF SPACE SYSTEMS (CLASS) A MODULAR TEST SYSTEM  
FOR SMALL SATELLITE VERIFICATION AND VALIDATION

BY

MARC AKIKI

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Aerospace Engineering  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Adviser:

Associate Professor Michael Lembeck

## ABSTRACT

Closed Loop Analysis of Space Systems (CLASS) is a modular, closed-loop satellite test system developed at the Laboratory for Advanced Space Systems at Illinois (LASSI). This thesis presents the technical details of the initial engineering development stages of CLASS. The CubeSat concept is now twenty-one years old, yet a significantly high number of mission failures (i.e., 33% for commercial developers and 55% for academic developers) are still being experienced. By employing hardware-in-the-loop testing driven by closed-loop simulations, critical aspects of validation and verification can be achieved with improved fidelity in an attempt to enhance the mission success rate of CubeSats.

CLASS is composed of real-time satellite orbital mechanics and rigid body dynamics simulations executing on a Raspberry Pi 4. The satellite attitude dynamics, orbital mechanics, and space environment properties are computed by the CLASS software reliably and rapidly. It has the ability to interface with the satellite's flight computer, sensors, and actuators. If individual flight hardware elements are not available, as may be the case early in a CubeSat's integration flow, emulators for such components as magnetometers and gyroscopes, executing on Arduino boards, are easily configurable by the user to match the behavior and properties of the actual hardware.

The theory behind the dynamic simulation programmed in CLASS is presented. CLASS updates the attitude of the satellite and environmental properties every five milliseconds and the orbital elements every sixty seconds. The performance and validity of the simulation algorithms are also presented. Finally, the results from a closed-loop test for the attitude determination and control system of one of the LASSI CubeSats, CAPSat, is demonstrated. CLASS is playing a critical role in the development of CAPSat by validating the design of a state feedback controller for nadir pointing, identifying hardware limitations, and enabling the correction of software errors.

CLASS will serve as a general purpose and easily configurable test system for all CubeSats and space systems developed at the University of Illinois.

## **ACKNOWLEDGEMENTS**

Thank you Dr. Michael Lembeck for giving me the opportunity to work in the Laboratory for Advanced Space Systems at Illinois. Dr. Lembeck saw in me a strong passion for space exploration and space engineering. I learnt a lot under his leadership and guidance. He certainly played an immense role in shaping my professional career.

I want to thank my teammates and friends working on spacecraft attitude determination and control systems: Hongrui Zhao, Eric Alpine and Walker Dimon as well as everyone else who contributed to our CubeSat missions.

While completing my Master's Degree, I was blessed with an inordinate amount of support from my mother, Fadia Salame, my father, Elie Akiki, my brother, Ralph Akiki and all my other dear relatives. Thank you for all the support despite being 6200 miles away.

*To my mother, Fadia Salame, for her exceptional support. To space exploration, one of humanity's greatest adventures*

## TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION .....	1
CHAPTER 2: DESIGN .....	14
CHAPTER 3: DYNAMIC MODEL .....	19
CHAPTER 4: REAL TIME SIMULATIONS .....	43
CHAPTER 5: CLOSED LOOP TESTS.....	55
CHAPTER 6: CONCLUSION.....	69
REFERENCES .....	71

## **CHAPTER 1: INTRODUCTION**

### **1.1 OBJECTIVE**

The objective of this research is to develop a hardware-in-the-loop test system for CubeSats being developed at the Laboratory for Advanced Space Systems at Illinois (LASSI). LASSI is affiliated with the Aerospace Engineering Department at the University of Illinois in Urbana-Champaign. By taking advantage of readily available, inexpensive processors, modern simulation algorithms, and a unique test system for satellite interfaces, CubeSats can be tested the way they are intended to fly and fly the way they have been tested. This approach improves end-to-end reliability of both flight and ground elements of the satellite and mission operations.

In this introduction, the history and configuration of CubeSats is introduced and approaches for the testing of satellites are reviewed. Later, an approach for closed loop testing of CubeSats, including the test and emulation subsystem hardware, is described. Results from simulations produced by the system are also provided.

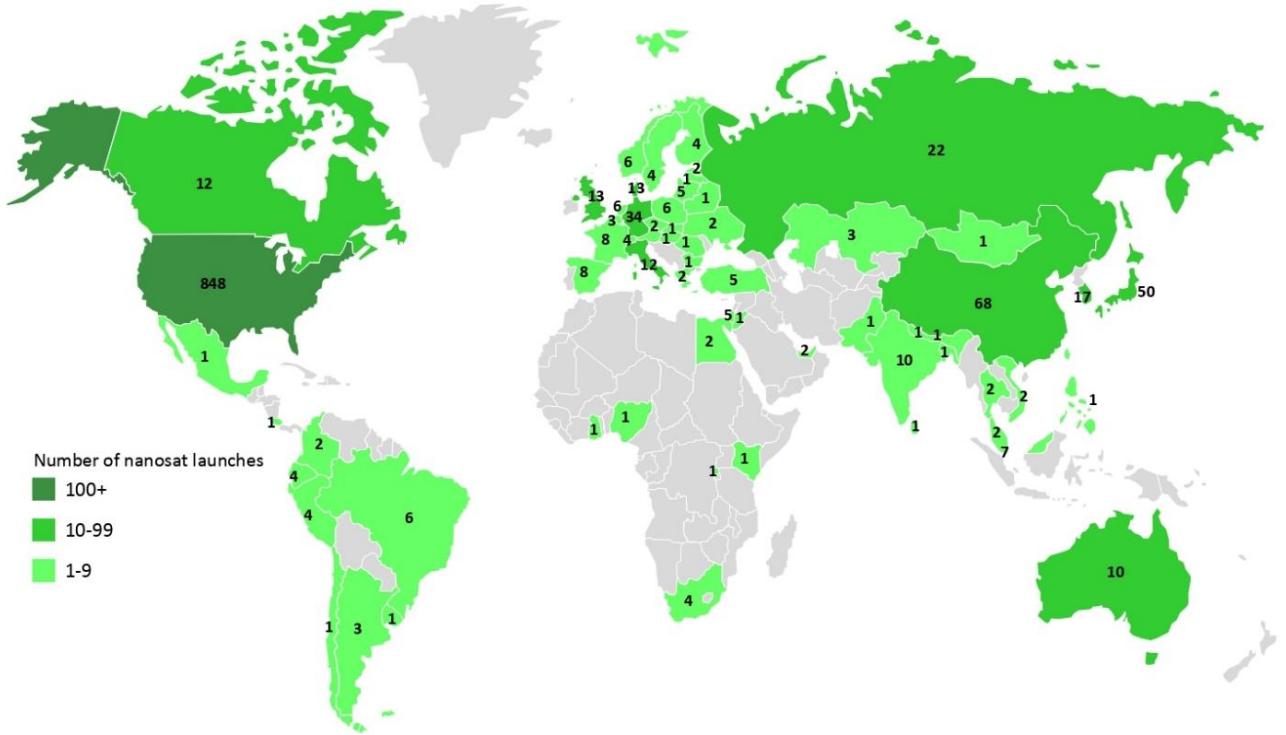
### **1.2 BACKGROUND**

#### **1.2.1 CubeSats**

With the goal of reducing the time frame of small satellite development and increasing the number of satellites per launch vehicle, researchers at California Polytechnic State University, under the guidance of Professor Jordi Puig-Suari and Professor Robert Twiggs, established the standards for CubeSats in 1999 [1] [2]. The customary CubeSat size is designated as 1U (dimensions: 10 cm × 10 cm × 10 cm). As of 2018, the most common (64%) CubeSat size is the 3U [3] (dimensions: 30 cm × 10 cm × 10 cm). The size standards, as of 2019, range from 0.25U to 27U [4] and all sizes maintain the face size of 10 cm × 10 cm [5]. Their mass ranges from 0.2 kg to 40 kg.

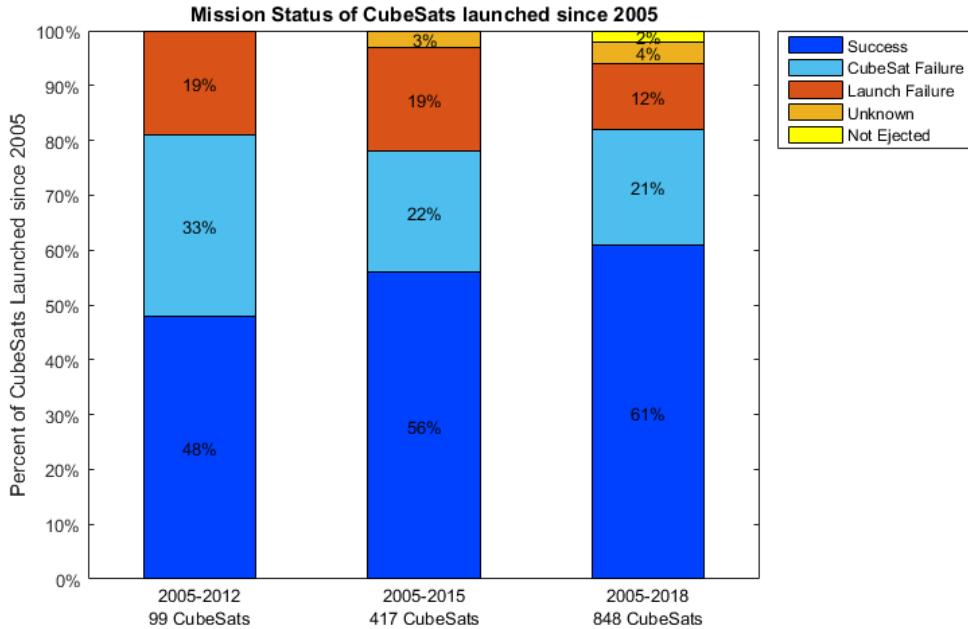
The standardized modular design of CubeSats has gained tremendous popularity in the small satellite industry and academia in the past twenty years. From 1999 to 2015, 425 international CubeSats were launched according to [6]. From 2016 to 2018, 663 CubeSats were launched [4]. In a period of three years only, there was a 150% increase in the number of CubeSats launched. Small satellites have become popular academic projects providing students the opportunity to gain practical experience on the design and development of spacecraft and facilitating access to space for small payloads [7]. Before 2013, universities were the leading developers of CubeSats [6] [8]. However, the modularity, the short development timeline, and affordability of these nanosatellites attracted the interest of the commercial space industry. As a result, universities were surpassed by the commercial sector (57% of all CubeSats are commercially developed) as the leading end-users of today's deployed CubeSats [3].

Due to size constraints, CubeSats cannot carry large payloads but, fortunately, innovations in instrumentation and electronics are reducing the amount of physical space needed to perform mission tasks. Common CubeSat mission objectives are atmospheric studies, Earth/deep space observation, and the demonstration of new space equipment. The discipline of small satellites is revolutionizing the space industry and increasing accessibility to space internationally (see Figure 1 for the number of nanosatellite launches per country as of October 2019).



**Figure 1:** World map indicating the number of launched nanosatellites per country as of October 2019 according to Erik Kulu, Nanosats Database [9].

CubeSat technology has evolved significantly as a result of the increased launch rates. However, reliability is still lacking as the success rate of CubeSat launches is 45% for academia and 77% for industry [10]. As documented in [3] and illustrated in Figure 2, the mission success rate (a mission is defined as successful if the CubeSat performs its primary mission objective and downlink/uplink the required data, if applicable) of all 848 CubeSats launched during the period of 2005-2018 is at 61%. Despite the sharp increase in the number of CubeSats launched from 2012 till 2018 (850% increase in 6 years), the success rate is still considerably low.



**Figure 2:** Mission status of CubeSats launched since the year 2005. Data extracted from T. Villela, C. A. Costa, A. M. Brandao, F. T. Bueno and R. Leonardi, "Towards the Thousandth CubeSat: A Statistical Overview" [3].

Avoidable failures caused by poor system design and inadequate testing could benefit from increased attention to verification testing, especially in academia [11]. According to [12], the three leading CubeSat subsystems that are commonly identified as the primary contributors to mission failure are Electrical Power System (EPS), On-Board Computer (OBC), and Communication System (COM). But, the most concerning statistic is that 33% of developers that experience a CubeSat failure right after deployment report that they do not know and do not have any way of identifying the subsystem/event causing the failure. CubeSats that fail to execute any useful mission objectives ultimately end up as space debris and eventually decay in the atmosphere, making failure analysis an arduous task. For failures that occur 30+ days after deployment, EPS is identified as the leading contributor to mission failure (44%). But, even after 90 days of nominal CubeSat behavior and despite having a decent amount of logged flight data before failure, the percentage of unknown mission failure causes does not drop below 10% [12].

A survey conducted by representatives from leading aerospace companies, documented in [10], aims at identifying what are the leading factors in the development process that lead to critical subsystem failures and consequently mission failure. Twenty-three CubeSat developers from academia, government and industry were interviewed. The authors compiled the main factors that the interviewees identified as the leading causes of mission failures (or the key contributors to mission success) into eight themes. The theme that encompasses the importance of pre-launch testing was underlined by all interviewed organizations. Particularly, there were concerns on the payload-to-bus interface testing and the lack of hardware/software in the loop testing. Although this result is not surprising, it serves as a practical emphasis on the importance of a comprehensive systems test, starting with system software functional verification, to individual subsystems qualification testing, to a full CubeSat performance verification,

### 1.2.2 Satellite Subsystems

A satellite's system components and functional capabilities is driven by the primary mission objective: accommodating the payload. The satellite bus provides the utilities, configuration control, and maneuvering capabilities necessary for the payload to successfully achieve its goals [13]. The satellite subsystems, making up the bus, are expressed in Table 1.

**Table 1:** Satellite Subsystems.

Subsystem Name	Function	Example(s) of hardware and/or software
Attitude Determination and Control System (ADCS)	Maintain the required satellite attitude using sensor readings, signal processing, filters, and actuator controllers.	<ul style="list-style-type: none"> <li>Sensors: magnetometers, gyroscopes, star trackers, sun sensors etc.</li> <li>Actuators: magnetometers, reaction wheels, thrusters etc.</li> <li>ADCS software containing the controllers and the algorithms</li> </ul>

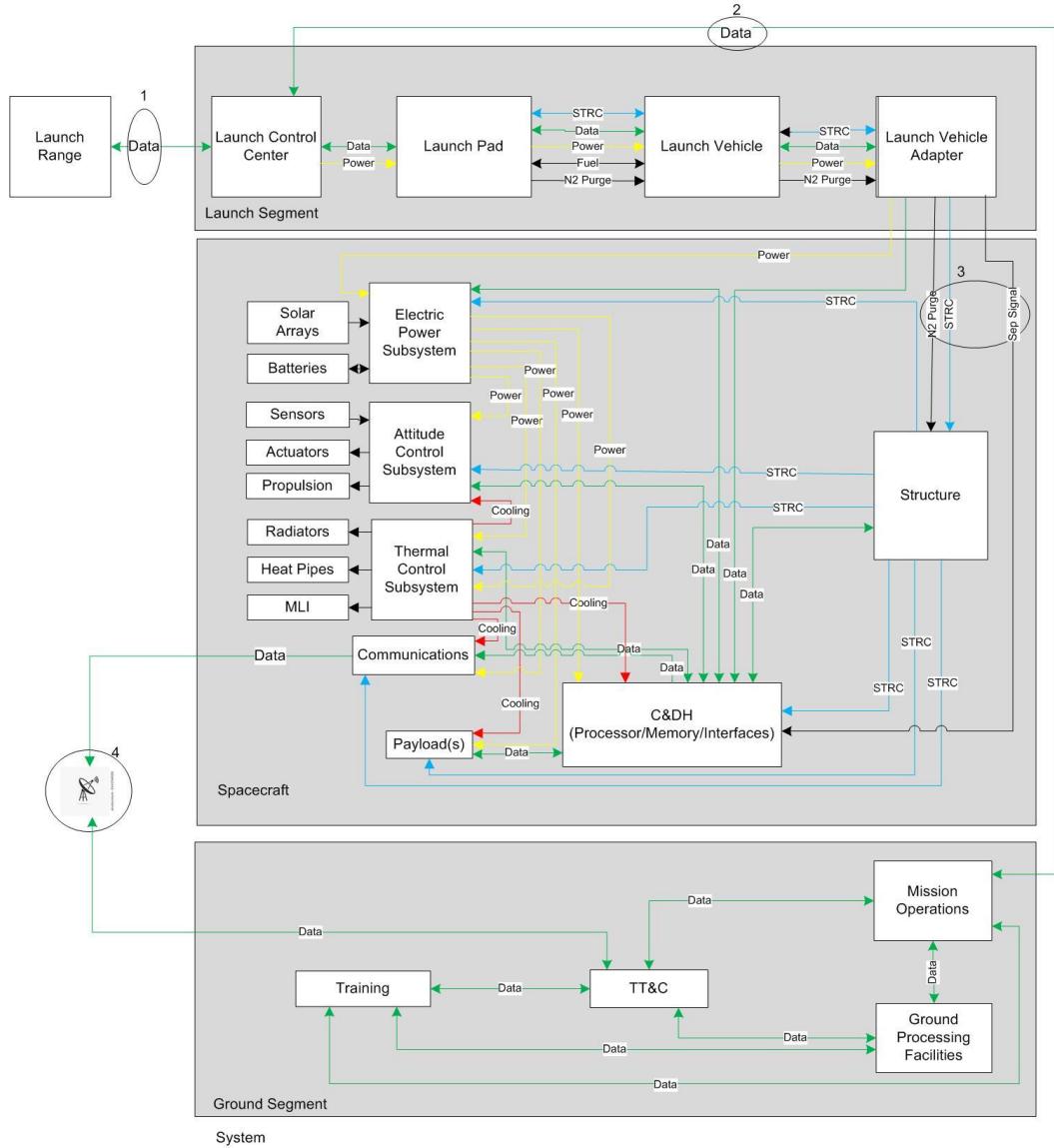
*Table 1 (cont.)*

Subsystem Name	Function	Example(s) of hardware and/or software
Orbital Mechanics and Space Trajectories	Determine the appropriate orbital trajectory and elements (e.g., orbital plane inclination, altitude etc.) the satellite needs to be in at any given instance of the mission	<ul style="list-style-type: none"> <li>• On-board orbit propagators</li> <li>• Thrusters</li> <li>• Trajectory optimization and gravity assist</li> </ul>
Command and Data Handling (C&DH) and Telemetry	Manages the data sent and received by the satellite and controls the sequence of operations and the data transfer between the different subsystems	<ul style="list-style-type: none"> <li>• Data storage and downlink protocols</li> <li>• Main controller of the flight software</li> </ul>
Electric Power System (EPS)	Provide electric power for the bus and the payload	<ul style="list-style-type: none"> <li>• Solar arrays</li> <li>• Batteries</li> <li>• Radioisotope Thermoelectric Generator</li> <li>• Power distributors</li> </ul>
Thermal Control System (TCS)	Maintain the satellite's component temperatures within the appropriate and operational range	<ul style="list-style-type: none"> <li>• Radiator and cooling fluids</li> <li>• Multilayer insulation</li> <li>• Heaters</li> </ul>
Structure and Mechanisms	Provides support for satellite systems and any deployables	<ul style="list-style-type: none"> <li>• Solar panels</li> <li>• Payload volume</li> <li>• Material</li> <li>• Robotic arms and deployables</li> </ul>
Propulsion	Provides the commanded thrust necessary for changing or maintaining a satellite's trajectory, orbit, or attitude during a mission	<ul style="list-style-type: none"> <li>• Chemical thrusters and the propellant</li> <li>• Electric thrusters</li> <li>• Ion thrusters</li> </ul>

Clearly, the design parameters for all these subsystems have some interdependencies with each other. These relationships can vary between spacecraft, but Figure 3 provides a generic illustration of the different interfaces between a satellite's subsystems.

The launch vehicle and ground station are also two integral segments of a space mission. The lifting capabilities, the volume available, the orbit insertion specifications, the vibration environment and cost are a few of the many launch vehicle characteristics that affect the satellite's design [14]. As for the ground station, its primary purpose is to send commands to the spacecraft

and receive telemetry data (i.e., to evaluate the health status of the spacecraft) and payload data [13]. The ground station is composed of the necessary communication equipment and a team of trained individuals that follow a pre-defined mission operations procedure [14].



**Figure 3:** Space mission segments and interfaces.

### 1.2.3 Space Systems Testing

While the development of complex systems such as a satellite is a laborious process, testing the system performance and its outputs can be equally as demanding. Even if some components

or software might seem simple and straightforward, it is critical that each module is tested individually and as a system. In 1999, the \$327 million Mars Climate Orbiter developed by NASA took an unintended trajectory while approaching Mars and burned up in the Martian atmosphere [15]. The principal trigger for that costly error was a miscommunication between two engineering and operations teams. One team assumed the propulsion subsystem was using Newtons for units and the other assumed they were using pound-force for units. While the failure was lack of proper team communication, a test incorporating direct comparisons with an expected model/set of results of the mission sequence would have highlighted the error [15].

“Adequate testing,” as understood by engineers, usually means undertaking a process known as Verification and Validation (V&V) testing [16]. The exact definition of each term in (V&V) differs depending on the engineering discipline it is being applied to. The *IEEE-610* standard for computer systems defines [17]:

- **Verification:** the steps necessary to assess whether the products of a certain development phase of a system or component meet the specifications set prior to the beginning of the phase (i.e., does it do the “thing” right?).
- **Validation:** the steps necessary, during or after the development phase, to assess whether the system or components meet the specifications (i.e., does it do the right “thing”?).

NASA engineers have complied in the past with the above definitions for V&V, according to a 2002 survey [18]. Space systems engineers *verify* a system by making sure it produces the right outcome and *validate* a system by making sure it conforms to the specifications and requirements set for it.

The second revision of the NASA Systems Engineering Handbook divides a space program/project life cycle into three high-level categories: Program Pre-Formulation, Program

Formulation and Program Implementation [19]. The three categories are then divided into Phases A to F. Phase D: System Assembly, Integration and Test, and Launch is where 50% of the mission life-cycle costs lie. However, the report clearly states that verification and validation does not only occur during Phase D. Product V&V should be performed extensively during the development phase to reduce the risk of making costly design adjustments during Phase D and to increase the system's reliability.

CubeSats are paving the way for rapid and scalable spacecraft development. Yet, some universities are struggling to keep up with fast-approaching launch deadlines [10]. Having a modular test bed that can be readily configured depending on the satellite bus configuration, payload, or other components being tested can significantly enhance mission reliability of a CubeSat. A comprehensive systems test involves emulating the operating and environmental conditions of the product or system [16]. With spacecraft, this is a significant challenge. Space industry leaders have worked for years to develop state-of-the-art testing facilities such as the NASA Glenn Research Center (e.g., micro-gravity testing capabilities, vacuum chamber, electromagnetic interference etc. [20]) and the ESA European Space Research and Technology Centre (e.g., antenna test facilities, thermal data handling facilities, Large Space Simulator etc. [21]). These facilities are usually not available for small satellite developers such as start-ups and university labs. In fact, one of the major issues brought up by universities, in the interviews conducted by [10], is that the lack of access to testing equipment and the little amount of time they have to develop emulators is forcing them to limit the amount of systems test time available before launching.

However, performing a spacecraft systems test does not necessarily have to involve sophisticated and costly equipment. While NASA was getting ready to launch the Galileo

spacecraft in the 1980s, engineers at JPL, in [22], developed one of the first closed-loop test systems to thoroughly test the spacecraft's Attitude and Articulation Control Subsystem (ACAS). The celestial sensors and inertial sensors were stimulated using a four-body dynamic simulation. A dynamic load, applied using an analog control loop commanding a DC motor, provided the appropriate inertial load on the two major Galileo actuators in response to the simulation. As a result, the actuator hardware and pointing control was accurately tested without having the massive spacecraft spin on frictionless air bearing tables. Additionally, external interfaces, such as the Retro-Propulsion Module, Command and Data Subsystem, Power and Pyro Subsystem, and the Radio Frequency Subsystem, with the ACAS system were emulated using hardware panels and computer software. The closed-loop test system developed for Galileo [22] is an example of the potential of designing a full systems test bed without the need for sophisticated test equipment.

Today, a variety of CubeSat test systems can be found in the literature and are mostly utilized in university labs. Hardware-in-the-loop (HWIL) testing provides engineers with a means of testing the real hardware responses and in the process provide the capability to identify sensor resolution errors, hardware lag, and signal noise [23]. When it comes to CubeSats, ADCS HWIL testing is a popular topic in the literature because it involves a variety of hardware (e.g., sensors and actuators) and the need for an accurate dynamic simulation or emulation [24]. In an effort to demonstrate satellite formation control using Global Positioning System (GPS), the authors in [25] developed a simulation that generates a GPS signal and sends that signal to two GPS receivers. To properly simulate a GPS signal, a measurement model was implemented. The two GPS receivers send the signal picked up to the inter-satellite communication model and the flight computers which interface with the external controller to determine the current satellites' dynamics. The external controller sends the control information to the GPS signal simulator, thus, closing the

loop. Similarly, an academic CubeSat development team [26] tested the spin stabilization capabilities of one of their CubeSats using HWIL. An attitude and orbit dynamics simulation commands, in real-time, a Helmholtz cage to stimulate the ADCS magnetometers. The on-board ADCS flight processor then computes the necessary voltage command sent to the torque coils. The team then used this voltage command to calculate the torque that would have been experienced by the CubeSat and apply it to the dynamic simulation. The test helped the team modify a critical aspect of their design: the magnetometers did not have enough time to properly sample a reading before the torque coils turn on and corrupt the magnetometer readings. Such a design issue would not have come to light in a software-only simulation.

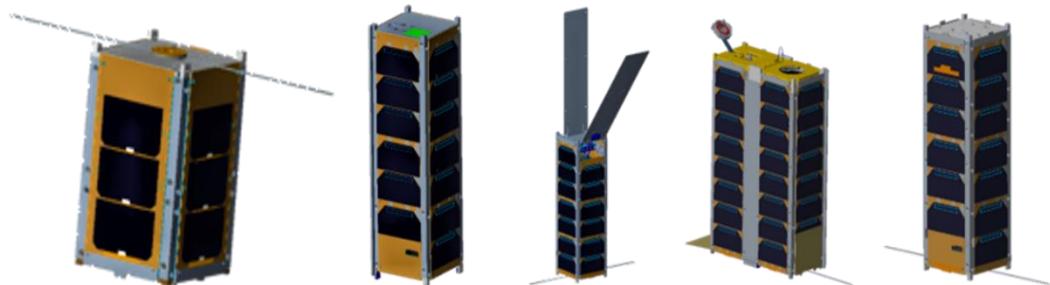
In conclusion, an accurate and reliable test system, with HWIL, for CubeSats could be used to improve overall system reliability [24] [25]. The desired functions would be to simulate the space environment (e.g., orbits, celestial illumination, etc.), the satellite dynamics (e.g., attitude inertial sensor stimulation etc.), emulate missing or unavailable bus hardware during test (i.e., provide the responses of an unavailable magnetometer at the satellite bus interface using a common piece of computing hardware like an Arduino or Raspberry Pi), and display telemetry for operators.

### 1.3 LABORATORY FOR ADVANCED SPACE SYSTEMS AT ILLINOIS

#### 1.3.1 Lab Missions

The Laboratory for Advanced Space Systems at Illinois (LASSI) is a multi-disciplinary lab for the research and development of small satellites and other space systems engineering projects. It is part of the Aerospace Engineering Department at the University of Illinois at Urbana-Champaign. While providing students from different majors a hands-on practical experience with the design, development and testing of CubeSats, the lab also assists customers from industry with some of their space systems through research and testing equipment.

**Table 2:** CubeSat missions at LASSI as of May 2020.



CubeSat Mission Name	CubeSail	SASSI^2	CAPSat	LAICE	SpaceICE
Size	2 × 1.5U	3U	3U	6U	3U
Mission Objective	Test of a solar sail concept: Ultra-sail developed by CU Aerospace	Spectrometer analysis of chemical elements and reactions during atmosphere re-entry	Radiator for active cooling, quantum annealing experiment and attitude control by solar panel deformation	Gravity waves analysis in the atmosphere	Measurement of particle behavior in freeze-casting experiments
Collaborator(s)	JPL CU Aerospace	Purdue University	JPL ARC NSF	Northwestern University	Virginia Polytechnic
Sponsor	NASA ELaNa	NASA USIP	NASA USIP	NASA ELaNa	NSF

The first satellite built by LASSI was launched in 2018 and was called CubeSail consisting of two 1.5U CubeSats held together by a 20 sq. meter solar sail. The goal was to test a solar sailing technique developed by the space company CU Aerospace. The next satellite, SASSI<sup>2</sup>, was a 3U and was launched in April 2019 to perform spectrometer analysis of chemical reactions during atmosphere re-entry. Next in line is CAPSat, another 3U satellite, due to launch in the summer of 2020. CAPSat has three payloads for three different research groups. The first payload is a radiator to test a novel active cooling technique. The second payload is a quantum annealing experiment designed by the Physics Department at the University of Illinois. The third payload is the

implementation of a fine pointing attitude control technique achieved by creating controlled solar panel deformations. After CAPSat, two more CubeSat launches are planned: LAICE 6U and SpaceICE 3U. LAICE will study atmospheric gravity waves and SpaceICE will perform freeze-casting experiments for the purpose of analyzing particle behavior. This thesis is intended to significantly expand the team's capabilities and systems testing abilities for those satellites.

### 1.3.2 Lab Equipment

The lab facility at the University of Illinois is equipped with the following test equipment [27]:

Clean Room: A 100K (cleanliness factor) room devised for the test and assembly of spaceflight hardware.

Thermal Vacuum Chamber: The  $60.96 \times 60.96 \times 60.96$  cm (2 ft  $\times$  2ft  $\times$  2ft) chamber, made of ASTN A36 tempered steel, can achieve internal vacuum pressure levels of  $1 \times 10^{-10}$  Torr. It can also provide an internal temperature environment between  $\pm 100$  °C to mimic the dramatic shifts in temperature a satellite experiences during orbit.

Sun Simulator: Full spectrum sunlight is provided for testing and calibrating solar powered systems.

Helmholtz Cage: Six magnetic coils used to generate a 3-axis magnetic field simulating the environment satellites experience in orbit. The cage can generate a custom time varying magnetic field value when commanded by an orbit propagator.

## CHAPTER 2: DESIGN

### 2.1 DRIVING REQUIREMENTS

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 Closed Loop Analysis of Space Systems (CLASS), a modular satellite test system. Some of the driving requirements that influenced the design are provided below.

The closed-loop test system needs to provide a real-time dynamic simulation of the orbital mechanics and satellite rigid body dynamics required to enable hardware-in-the-loop testing. Academic CubeSat developers seldom have access to sophisticated six degrees of freedom air bearing tables inside vacuum and magnetic chambers that emulate the physical behavior and environment of the satellite in orbit. Thus, an accurate software dynamics simulation running in real-time interfaced with the hardware is a satisfactory, accessible tool for testing the satellite systems.

The closed-loop test system must also be easily configurable by the user. Different subsystems within a satellite can require different testing configurations. For example, the user could be testing the Attitude Determination and Control System (ADCS) separately or jointly with the Electrical Power System (EPS). This would require the test system to either provide only satellite attitude data to the ADCS or convert the attitude representation into a Sun vector for the EPS to determine the energy collected from the solar arrays. A modular test system decreases the amount of time spent customizing simulations and validation procedures.

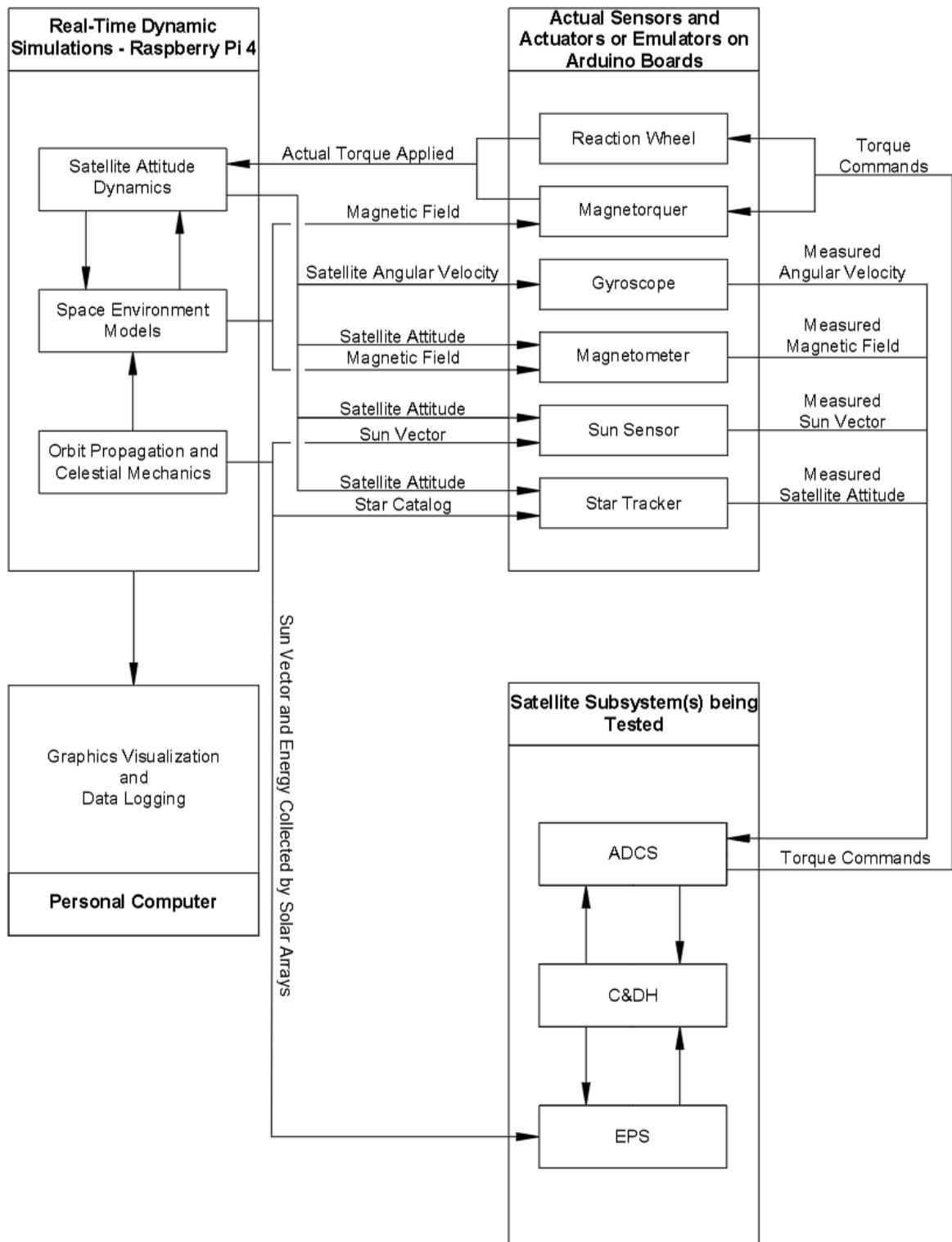
The closed-loop test system must rely on widely used and well documented processors, hardware, and programming languages. This feature makes the test system compatible with a

majority of satellite processors and hardware. It also decreases the amount of time and effort to become familiar with the architecture for customization purposes.

The closed-loop test system must run at least at real-time speed and mitigate numerical inaccuracies and asynchronous execution delays. If the dynamics simulation software can be executed at faster than real-time speeds, providing excess computational capabilities, the option of implementing a live graphics and data visualization feature becomes feasible.

## 2.2 CLASS SPECIFICATIONS

Closed Loop Analysis of Space Systems (CLASS) is a modular satellite test system developed at the Laboratory for Advanced Space Systems at Illinois (LASSI). CLASS provides an accurate real-time simulation of orbital mechanics and satellite dynamics using the algorithms and theory described in Chapter 3. Data describing the instantaneous state of the satellite (i.e., attitude, orbital position, magnetic field vector etc.) can be extracted from the simulation and commands generated by control software in the satellite under test (i.e., torque commands, propulsion commands etc.) can be sent to the simulation in real-time. The architecture implemented by CLASS is represented in Figure 4.



**Figure 4:** Architecture diagram for CLASS. Orbital mechanics and attitude dynamics simulations close the loop around real and emulated satellite control algorithms.

Several test configurations are made possible by CLASS. The simulation 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 test, sensors typically produce “static” outputs as a result of sitting on a table or bench. These outputs are replaced by the CLASS simulated outputs for the satellite flight software to use in its control computations. 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 apply the magnetic field value which is picked up by the satellite magnetometer sensor placed inside of the cage. Attitude control software receives these sensor inputs, computes a control command, and commands the three degrees of freedom magnetorquer currents in an attempt to point the spacecraft in the desired direction. The loop is closed when the commanded magnetorquer currents are received by the dynamics simulation, converted into torques, and the simulated satellite moves in the desired direction. 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.

A microprocessor of sufficient speed to calculate the six degrees of freedom dynamics simulation of a satellite in Low Earth Orbit was required to successfully implement CLASS. A Raspberry Pi (RPi) 4 has been used to successfully execute the aforementioned orbital mechanics and dynamic simulations. Simulation software, written in C++, executes in real-time on the RPi 4 using the Linux patch PREEMPT\_RT. RPi 4 is a well-documented and widely used processor making CLASS an easily configurable, compatible, and reliable test system.

If installation of the actual sensor is not feasible or the sensor is not available (as might be the case early in an integration flow), CLASS offers the option of implementing a customizable

sensor emulator. For the initial version of CLASS, emulators were developed on Arduino boards for a magnetometer and a gyroscope. The emulator boards interface with the dynamic simulation and the satellite being tested. The emulators are provided with the current attitude and orbit information from the dynamic simulation, transform it into the corresponding calculated raw sensor data format with noise and appropriate hardware timing delays added to it, and then sent to the satellite being tested. Test code in the satellite transparently replaces any real sensor data with the emulated data, allowing the flight software to compute flight-like control responses as a result.

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 it can be applied in the simulation.

The real-time simulation has been proven to be reliable and the code execution occurs in real-time. The following chapters provide insight into the design and implementation of CLASS. Verification test results are also provided. While CLASS now provides an operational capability for the LASSI lab to perform closed-loop tests on CubeSats, opportunities for enhancing user interfaces and other performance upgrades are identified in the concluding chapter.

## CHAPTER 3: DYNAMIC MODEL

CLASS provides a real-time simulation of a satellite's orbital motion and rigid body dynamics that close the loop around the attitude determination and control software executing in the satellite's command and data handling system. In this section, the different dynamic and environmental models that are coded into the real-time simulation are presented. CLASS is implemented on a Raspberry Pi 4 using PREEMPT\_RT, a Linux kernel patch that emulates a Real Time Operating System (RTOS).

All equation quantities expressed in **Bold** are vectors or matrices and all quantities expressed in regular font are scalars or magnitudes. This format is applicable throughout the entire thesis.

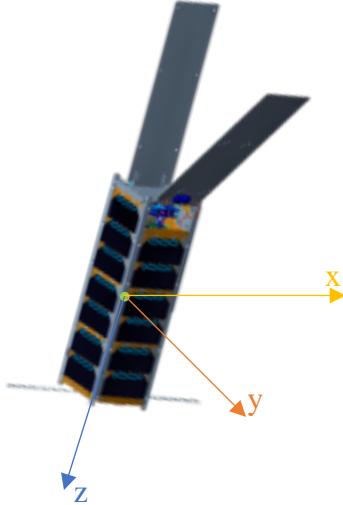
### 3.1 FRAMES OF REFERENCE

To express the many vector quantities that define a satellite's orbital position and velocity, as well as its inertial attitude and rates, several frames of reference are used in CLASS. All time representations and dates along with the different rotation matrices of the different coordinate frames are included in the CLASS software and can be called by the user.

#### 3.1.1 Satellite Body Frame

The satellite body frame of reference is a coordinate system that is fixed to the satellite. Usually, the origin of the coordinate frame is at the center of mass of the satellite. It is also common practice to define the direction of the three Cartesian axes according to the CalPoly CubeSat standards [28]. According to the CubeSat Design Specifications (CDS) Rev. 13, the satellite's coordinate frame shall follow the standard assigned for each size. For example, a 3U satellite, like CAPSat in Figure 5, has the  $-z$ -axis on the face that has the deployment switch for the Poly-

Picosatellite Orbital Deployer (P-POD) and the  $+x$ -axis on the face that has the Access Ports in the P-POD.



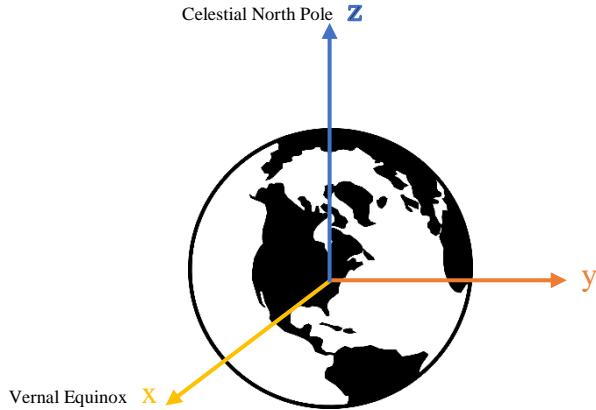
**Figure 5:** Example of the satellite body frame on CAPSat.

In CLASS, the user can choose to define the origin at any location desirable (e.g., center of mass, center of geometry, center of pressure, or any other point of interest). Also, the user can create multiple body frames with customized directions for the axes if it facilitates specific kinematic analysis. The satellite body frame is used to determine the satellite's attitude using either Euler angles or quaternions that describe the rotation between the body frame and the inertial frame. Details about the expression of satellite attitude can be found in Section 3.2.2.

### 3.1.2 Inertial Reference Frame

An inertial reference frame is described as relatively *fixed* and Newton's laws only apply in such a frame [29]. The standard reference frame used for celestial mechanics is the *International Celestial Reference System Frame* (ICRF) [30]. The axes of this frame are fixed relative to extragalactic radio waves sources and its origin is at the mean center of mass of the solar system [30]. However, there is a much simpler approximation for a fixed reference frame when it comes

to satellites. For a satellite, a *fixed* frame means it does not rotate with the Earth. So, a group of Earth-centered reference frames known as Earth Centered Inertial (ECI) or Geocentric-Equatorial Coordinate System (GES) are commonly used [31]. In CLASS, the ECI J2000 reference frame is used as the inertial reference frame. The *x-y plane* is aligned with the Earth's mean equator at 12:00AM on the first day of January 2000 and the coordinate system's center is coincident with the Earth's center of mass. The *x-axis* points to the vernal equinox in the year 2000. The *z-axis* is aligned with the celestial North Pole (the Earth's spin axis).



**Figure 6:** Earth centered inertial reference frame J2000.

### 3.1.3 Earth Centered/Earth Fixed Frame (ECEF)

The Earth Centered/Earth Fixed Frame (ECEF) frame is convenient for satellites because this frame rotates with the Earth [30]. The *x-axis* points from the center of mass of the Earth to the prime meridian. The *z-axis* is the same as that of the ECI frame (pointing along the celestial North Pole). With the *y-axis* completing the right-hand rule, the transformation matrix for expressing an inertial vector  $\mathbf{r}^I$  in the ECEF frame is:

$$\mathbf{r}^{\text{ECEF}} = \begin{bmatrix} \cos(\theta_{\text{GMST}}) & \sin(\theta_{\text{GMST}}) & 0 \\ -\sin(\theta_{\text{GMST}}) & \cos(\theta_{\text{GMST}}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{r}^I \quad (1)$$

Where  $\mathbf{r}^{\text{ECEF}}$  is a  $3 \times 1$  vector in the ECEF frame and  $\theta_{\text{GMST}}$  is the angle in the  $x$ - $y$  plane between the ECI and ECEF frames.  $\theta_{\text{GMNST}}$  is referred to as the Greenwich Mean Sidereal Time (GMST) because it is time dependent. The angle is computed using the Julian Day (JD), the number of days since January 1, 4713 BC [32]. JD is a continuous representation of the date, so, it is commonly used in astrodynamics. JD is calculated as follows:

$$\begin{aligned} \text{JD} = & 367 \times \text{year} + 1,721,013.5 - \text{INT} \left[ 7 \left( \frac{\text{year} + \text{INT} \left[ \frac{\text{month} + 9}{12} \right]}{4} \right) \right] + \\ & \text{INT} \left[ \frac{275 \times \text{month}}{9} \right] + \text{day} + \frac{60 \times \text{hour} + \text{minute} + \frac{\text{second}}{60^*}}{1440} \end{aligned} \quad (2)$$

Where:

year = the year number in yyyy format

month = the month number

day = the calendar day

hour = the hour of the current time

minute = the minutes of the current time

second = the seconds of the current time with  $60^*$  meaning that 61 seconds are used for days with a leap second

$\text{INT}[\dots]$  = rounded to the largest integer

After computing the Julian Date, the number of Julian Centuries elapsed,  $T_0$ , since the J2000 mark is computed [30]:

$$T_0 = \frac{2451545.0 - 2451545}{36525} \quad (3)$$

Which is then followed by the computation of the angle  $\theta_{\text{GMST}}$  in units of seconds:

$$\theta_{GMST} = 24110.54841 + 8640184.812866T_0 + 0.093104T_0^2 - (6.2 \times 10^{-6})T_0^3 \\ + 1.002737909350795(3600 \times \text{hour} + 60 \times \text{minute} + \text{second})$$

The above quantity must then be transformed to be a value between 0 and 86400 (1 day = 24 hours = 86400 seconds) by adding or subtracting multiples of 86400. Finally,  $\theta_{GMNST}$  is obtained in degrees:

$$\theta_{GMST} = \theta_{GMST_{sec}} \frac{1}{240} \quad (4)$$

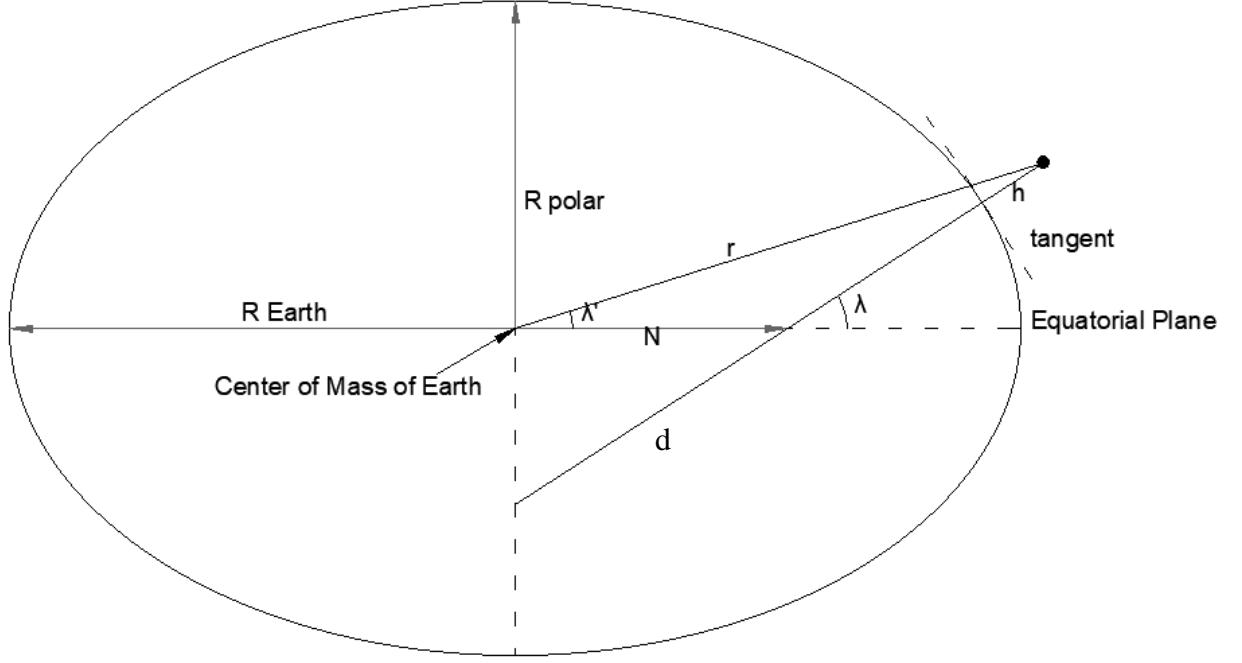
Calculating  $\theta_{GMST}$  allows us to determine the rotation matrix in Equation 1 to determine the expression of an inertial vector in the ECEF frame. Again, such a function is coded into the CLASS software and can be used for graphics display, debugging, and general calculations.

### 3.1.4 Latitude, Longitude, and Altitude (LLA)

Geographical coordinates are usually provided by Earth referenced latitude and longitude. For orbiting satellites, the tracking coordinates can sometimes be given in terms of latitude, longitude, and altitude from the Earth's surface. CLASS has functions that transform the Latitude, Longitude, and Altitude (LLA) representation of position into a more calculation-friendly representation such as the ECEF representation. To understand the transformation equations, it is important to distinguish between two commonly used representations for latitude: geodetic latitude and geocentric latitude [30].

The geocentric latitude  $\lambda'$  forms the angle between the equatorial plane and the position vector from the geometric center of the Earth to the spacecraft (indicated as  $\mathbf{r}$  in Figure 7). Since, the Earth is not a perfect sphere, its geoid is approximated as an ellipsoid. This reference ellipsoid allows us to determine the geodetic latitude. The ellipsoid model implemented in CLASS is the World Geodetic System 1984 model (WGS-84) with an eccentricity determined as  $e = 0.0818$ .

The geodetic latitude  $\lambda$  is the angle between the equatorial plane and the line normal to the surface of the ellipsoid intersecting with the satellite (indicated as **d** on Figure 7).



**Figure 7:** WGS-84 Ellipsoid Datum for Geodetic and Geocentric Latitude.

The distance  $N$  between the polar axis and the line **d** is calculated as:

$$N = \frac{R_{\text{Earth}}}{\sqrt{1 - e^2 \sin^2 \lambda}} \quad (5)$$

It is important to note that CLASS processes a positive latitude  $\lambda$  as the angle from the equator to the north and a negative  $\lambda$  as the angle to the south. Similarly, for the longitude  $\phi$ , a positive angle is from the Prime Meridian to the west and a negative angle is towards the east. Finally, the ECEF coordinates of an orbiting object are calculated using [30]:

$$r_x^{\text{ECEF}} = (N + h) \cos \lambda \cos \phi \quad (6)$$

$$r_y^{\text{ECEF}} = (N + h) \cos \lambda \sin \phi \quad (7)$$

$$r_z^{\text{ECEF}} = (N(1 - e^2) + h) \sin \lambda \quad (8)$$

If it is required to transform a geocentric latitude to a geodetic latitude, a function is provided in CLASS using a first order approximation from [30]. The equation used is:

$$\lambda = \lambda' + \frac{R_{\text{Earth}} - R_{\text{polar}}}{R_{\text{Earth}} + h} \sin 2\lambda' \quad (9)$$

### 3.1.5 Local Vertical/Local Horizontal Frame

The Local Vertical/Local Horizontal Frame (LVLH) is especially useful for Earth pointing satellites [30]. The LVLH frame is centered at the satellite's center of mass and has the *z-axis* always nadir-pointing. The *x-axis* points along the satellite's orbit velocity vector. Hence, the *y-axis* is normal to the orbit plane, opposite to the orbital angular momentum vector. The LVLH frame is included in CLASS to test the Earth pointing performance of nadir pointing satellites.

To form the basis vectors of the LVLH coordinate frame in CLASS, knowledge of the inertial position and velocity of the spacecraft (i.e., position and velocity in the ECI frame) is required.  $\mathbf{e}_1$  is the *x-axis*,  $\mathbf{e}_2$  is the *y-axis*, and  $\mathbf{e}_3$  is the *z-axis*:

$$\mathbf{e}_1 = \mathbf{e}_2 \times \mathbf{e}_3 \quad (10)$$

$$\mathbf{e}_2 = -\frac{\mathbf{r}^I \times \mathbf{v}^I}{\|\mathbf{r}^I \times \mathbf{v}^I\|} \quad (11)$$

$$\mathbf{e}_3 = -\frac{\mathbf{r}^I}{\|\mathbf{r}^I\|} \quad (12)$$

## 3.2 CUBESAT ATTITUDE

### 3.2.1 Attitude Dynamics

Treating small satellites as rigid bodies for attitude dynamics is a common practice in the literature [30] [31]. This assumption is accurate when dealing with 3U CubeSats with no deployables. For a rigid body rotating around its center of gravity (CG), the rotational dynamics can be expressed using Euler's rigid body rotational equation [30] [31]:

$$\dot{\omega}_{\text{bl}}^b = J^{b^{-1}} [T_{\text{ext}}^b - \omega_{\text{bl}}^b \times (J^b \omega_{\text{bl}}^b)] \quad (13)$$

Where:

$\dot{\omega}_{\text{bl}}^{\text{b}}$  = angular acceleration vector of the satellite body frame with respect to the inertial frame expressed in the body frame

$\mathbf{J}^{\text{b}}$  =  $3 \times 3$  matrix principal moment of inertia of the satellite expressed in the body frame (usually a diagonal matrix)

$\mathbf{T}_{\text{ext}}^{\text{b}}$  = external torques applied on the satellite expressed in the body frame

$\omega_{\text{bl}}^{\text{b}}$  = angular velocity vector of the satellite body frame with respect to the inertial frame expressed in the body frame

The assumption that a CubeSat is a rigid body has its limitations. To make CLASS a more generic simulation tool, additional options for dynamic rotational models, such as deployables and non-rigid body dynamics that include solar sails, will be added in the future.

### 3.2.2 Attitude Kinematics

CLASS utilizes quaternions to model a satellite's attitude. While Euler angles are the more intuitive method of expressing orientation, quaternions avoid gimbal lock (i.e., a singularity that can occur with Euler angles). The quaternion expression in CLASS takes the first element as the scalar:

$$\mathbf{q}_{\text{b}}^{\text{I}} = [q_1 \ q_2 \ q_3 \ q_4]^T \quad (14)$$

This means that the rotation matrix is populated in the following way [33]:

$$\mathbf{R}_{\text{b}}^{\text{I}} = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2q_3 + q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 + q_1q_3) & 2(q_3q_4 - q_1q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix} \quad (15)$$

The kinematic equation for quaternions is [34] [35]:

$$\dot{\mathbf{q}}_{\text{b}}^{\text{I}} = \frac{1}{2} \mathbf{E}^{\text{b}}(\mathbf{q}_{\text{b}}^{\text{I}}) \boldsymbol{\omega}^{\text{b}} \quad (16)$$

Where:

$$\mathbf{E}^b(\mathbf{q}) = \begin{bmatrix} -q_2 & -q_3 & -q_4 \\ q_1 & -q_4 & q_3 \\ q_4 & q_1 & -q_2 \\ -q_3 & q_2 & q_1 \end{bmatrix} \quad (17)$$

### 3.3 ORBITAL MECHANICS

Ever since Gauss's famous orbit determination theory, which was applied to calculate the orbit of Ceres, many algorithms have been developed to improve uncertainty estimations and position/velocity predictions [36]. Various models are used in publicly available orbit propagator software and have been proven to be accurate and reliable. Such analytical models include PPT2, SGP, SGP4, SGP8, SDP8, ANODE, HANDE, ASOP, AOPP [37]. Numerical models include SPEPH, POD, TRAJ, TMPEST [37].

The Simplified General Perturbation (SGP) analytical model was developed in the 1960s [38] [39] and is widely used for Low Earth Orbit satellites [40]. Many versions of the SGP model have been released since then [41]. An improved and simplified version, SGP4, was released in 1970 by Lane and Crawford [42]. CLASS utilizes the SGP4 orbit model because of its popularity with LEO simulations, its accessible documentation, and its satisfactory accuracy to run-time ratio [41]. A limitation of the SGP4 model is its numerical inaccuracies for long simulation times (over two weeks of simulation time). To utilize it as a long-term on-board propagator or a ground station propagator during the mission, it is recommended to regularly update the simulation with the actual measured satellite position provided by the North American Aerospace Defense Command (NORAD).

#### 3.3.1 SGP4 Model

The SGP4 model is based on a Differential Correction method that fundamentally compares the orbit state estimate with observations and checks for convergence using the Least

Squares Method and solving the Jacobian [41]. It was coded into the CLASS software using C++. The propagator is reliable for the current LASSI satellites, which are all LEO satellites for now. Testing and development of the LASSI satellites, as of now, requires only propagating the orbit for a few days. As a result, the long-term numerical drift of the SGP4 algorithm does not affect the current LASSI requirements.

The SGP4 model initializes orbit positions using a Two-Line Element (TLE) file. Small satellite TLE files are usually provided by NORAD, making it possible to use the SGP4 as an onboard orbit propagator. The algorithm, developed in CLASS extracts the following orbital elements from the TLE file and converts them to the appropriate units:

$t_0$  = epoch time in minutes

$n_0$  = mean motion in radians/minute

$e_0$  = eccentricity

$i_0$  = inclination in radians

$\omega_0$  = argument of perigee in radians

$\Omega_0$  = right ascension of ascending node in radians

$M_0$  = mean anomaly in radians

$B^*$  = atmospheric drag coefficient in  $1/R_{\text{Earth}}$  units

From now on, all terms that represent distance have units of Earth equatorial radii (i.e.,  $R_{\text{Earth}} = 6378.135 \text{ km} = 1 \text{ Earth Radii}$ ).

The algorithm then computes the following time-independent constant terms [40]:

$$a_1 = \left( \frac{k_e}{n_0} \right)^{\frac{2}{3}} \quad (18)$$

$$\delta_1 = \left( \frac{3k_2}{2a_1^2} \right) \left[ \frac{3 \cos^2 i_0 - 1}{(1 - e_0^2)^{\frac{3}{2}}} \right] \quad (19)$$

$$a_0 = a_1 \left( 1 - \frac{1}{3} \delta_1 - \delta_1^2 - \frac{134}{81} \delta_1^3 \right) \quad (20)$$

$$\delta_0 = \left( \frac{3k_2}{2a_0^2} \right) \left[ \frac{3 \cos^2 i_0 - 1}{(1 - e_0^2)^{\frac{3}{2}}} \right] \quad (21)$$

$$n_0'' = \frac{n_0}{1 + \delta_0} \quad (22)$$

$$a_0'' = \frac{a_0}{1 - \delta_0} \quad (23)$$

Where:

$$J_2 = 1.082616 \times 10^{-3} \text{ Earth oblateness perturbation constant}$$

$$k_e = \sqrt{\mu_{\text{Earth}}} = 0.0743669161 \left[ \text{units of Earth} \frac{\text{Radii}^{1.5}}{\text{minute}} \right]$$

$$k_2 = \frac{1}{2} J_2 R_{\text{Earth}}^2 = 5.41308 \text{ [units of Earth Radii}^2\text{]}$$

Next, the secular effects of atmospheric drag are initialized. These are based on a power law density function from [43].

$$\theta = \cos i_0 \quad (24)$$

$$\zeta = \frac{1}{a_0'' - s} \quad (25)$$

$$\beta_0 = \sqrt{1 - e_0^2} \quad (26)$$

$$\eta = a_0'' e_0 \zeta \quad (27)$$

$$C_2 = (q_0 - s)^4 \zeta^4 n_0'' (1 - \eta^2)^{-\frac{7}{2}} \\ \left[ a_0'' \left( 1 + \frac{3}{2} \eta^2 + 4e_0 \eta + e_0 \eta^3 \right) + \frac{3}{2} \frac{k_2 \zeta}{1 - \eta^2} \left( -\frac{1}{2} + \frac{3}{2} \theta^2 \right) (8 + 24\eta^2 + 3\eta^4) \right] \quad (28)$$

$$C_1 = B^* C_2 \quad (29)$$

$$C_3 = \frac{(q_0 - s)^4 \zeta^5 (-J_3 R_{\text{Earth}}^3) n_0'' R_{\text{Earth}} \sin i_0}{k_2 e_0} \quad (30)$$

$$C_4 = 2n_0'' (q_0 - s)^4 \zeta^4 a_0'' \beta_0^2 (1 - \eta^2)^{-\frac{7}{2}} \\ \left[ \left( 2\eta(1 + e_0 \eta) + \frac{1}{2} e_0 + \frac{1}{2} \eta^3 \right) - \frac{2k_2 \zeta}{a_0''(1 - \eta^2)} 3(1 - 3\theta^2) \left( 1 + \frac{3}{2} \eta^2 - 2e_0 \eta - \frac{1}{2} e_0 \eta^3 \right) \right. \\ \left. - \frac{2k_2 \zeta}{a_0''(1 - \eta^2)} \frac{3}{4} (1 - \theta^2) (2\eta^2 - e_0 \eta - e_0 \eta^3) \cos(2\omega_0) \right] \quad (31)$$

$$C_5 = 2(q_0 - s)^4 \zeta^4 a_0'' \beta_0^2 (1 - \eta^2)^{-\frac{7}{2}} \left[ 1 + \frac{11}{4} \eta(\eta + e_0) + e_0 \eta^3 \right] \quad (32)$$

$$D_2 = 4a_0'' \zeta C_1^2 \quad (33)$$

$$D_3 = \frac{4}{3} a_0'' \zeta^2 (17a_0'' + s) C_1^3 \quad (34)$$

$$D_4 = \frac{2}{3} a_0''^2 \zeta^3 (221a_0'' + 31s) C_1^4 \quad (35)$$

Where:

$$J_3 = -0.253881 \times 10^{-5} \text{ Earth oblateness perturbation constant}$$

$$q_0 = \frac{120}{R_{\text{Earth}}} + R_{\text{Earth}} = 1.01881427721 \text{ [units of Earth Radii]} \quad \text{altitude parameter}$$

constant (120 km + 1 Earth radius) from the power law density function [43]

$$s = \left( \frac{78}{R_{\text{Earth}}} + R_{\text{Earth}} \right) [\text{units of Earth Radii}] \text{ if altitude is greater than or equal to 156 km}$$

Or

$$s = \left( \text{perigee} - \frac{78}{R_{\text{Earth}}} + R_{\text{Earth}} \right) [\text{units of Earth Radii}] \text{ if altitude is greater than or equal to 98 km but less than 156 km}$$

Or

$$s = \left( \frac{20}{R_{\text{Earth}}} + R_{\text{Earth}} \right) [\text{units of Earth Radii}] \text{ if altitude is less than 98 km}$$

Next, the secular effects that are caused by the variations in Earth's gravity model are initialized using the following equations:

$$k_4 = -\frac{3}{8} J_4 R_{\text{Earth}}^4 \quad (36)$$

$$\dot{M} = n_0'' \left[ \frac{3k_2(-1 + 3\theta^2)}{2a_0''^2 \beta_0^3} + \frac{3k_2^2(13 - 78\theta^2 + 137\theta^4)}{16a_0''^4 \beta_0^7} \right] \quad (37)$$

$$\dot{\omega} = n_0'' \left[ -\frac{3k_2(1 - 5\theta^2)}{2a_0''^2 \beta_0^4} + \frac{3k_2^2(7 - 114\theta^2 + 395\theta^4)}{16a_0''^4 \beta_0^8} + \frac{5k_4(3 - 36\theta^2 + 49\theta^4)}{4a_0''^4 \beta_0^8} \right] \quad (38)$$

$$\dot{\Omega} = n_0'' \left[ \frac{-3k_2\theta}{a_0''^2 \beta_0^4} + \frac{3k_2^2(4\theta - 19\theta^3)}{2a_0''^4 \beta_0^8} + \frac{5k_4\theta(3 - 7\theta^2)}{2a_0''^4 \beta_0^8} \right] \quad (39)$$

Where:

$$J_4 = -1.65597 \times 10^{-6} \text{ Earth oblateness perturbation constant}$$

After the initialization of the previous constants and at each iteration of the simulation, there are multiple terms that are updated. The secular Earth zonal gravity and drag effect terms are updated in the algorithm first:

$$M_D = M_0 + n_0''(t - t_0) + \dot{M}(t - t_0) \quad (40)$$

$$\omega_D = \omega_0 + \dot{\omega}(t - t_0) \quad (41)$$

$$\Omega_D = \Omega_0 + \dot{\Omega}(t - t_0) \quad (42)$$

If the epoch perigee altitude is less than 220 km or for deep space applications, the variation terms in Equations 45 and 46 are dropped [40]. Otherwise, they are added:

$$\delta M = -\frac{2}{3}(q_0 - s)^4 B^* \zeta^4 \frac{R_{\text{Earth}}}{e_0 \eta} [(1 + \eta \cos M_D)^3 - (1 + \eta \cos M_0)^3] \quad (43)$$

$$\delta\omega = B^* C_3 (\cos \omega_0) (t - t_0) \quad (44)$$

This leads to the first update:

$$M = M_D + \delta\omega + \delta M \quad (45)$$

$$\omega = \omega_D - \delta\omega - \delta M \quad (46)$$

$$\Omega = \Omega_D - \frac{21 n_0'' k_2 \theta}{2 a_0'^2 \beta_0^2} C_1 (t - t_0)^2 \quad (47)$$

Atmospheric drag also introduces secular effects. If the perigee height is less than 220 km, these effects are represented by:

$$e = e_0 - B^* C_4 (t - t_0) \quad (48)$$

$$a = a_0'' [1 - C_1 (t - t_0)]^2 \quad (49)$$

$$IL = M + \omega + \Omega + n_0'' \left[ \frac{3}{2} C_1 (t - t_0)^2 \right] \quad (50)$$

But if the perigee height is greater than 220 km, additional higher order terms are included:

$$e = e_0 - B^* C_4 (t - t_0) - B^* C_5 (\sin M - \sin M_0) \quad (51)$$

$$a = a_0'' [1 - C_1 (t - t_0) - D_2 (t - t_0)^2 - D_3 (t - t_0)^3 - D_4 (t - t_0)^4]^2 \quad (52)$$

$$n_0'' \left[ \begin{aligned} IL &= M + \omega + \Omega + \\ &\left[ \frac{3}{2} C_1 (t - t_0)^2 + (D_2 + 2C_1^2)(t - t_0)^3 + \frac{1}{4} (3D_3 + 12C_1 D_2 + 10C_1^3)(t - t_0)^4 \right. \\ &\left. + \frac{1}{5} (3D_4 + 12C_1 D_3 + 6D_2^2 + 30C_1^2 D_2 + 15C_1^4)(t - t_0)^5 \right] \end{aligned} \right] \quad (53)$$

The effects of Earth's gravity on the long period terms are added using the following:

$$a_{xN} = e \cos \omega \quad (54)$$

$$\beta = \sqrt{1 - e^2} \quad (55)$$

$$IL_L = \frac{-J_3 R_{\text{Earth}}^3 \sin i}{8k_2 a \beta^2} (e \cos \omega) \left( \frac{3 + 5 \cos i}{1 + \cos i} \right) \quad (56)$$

$$a_{yNL} = \frac{-J_3 R_{\text{Earth}}^3 \sin i}{4k_2 a \beta^2} \quad (57)$$

$$IL_T = IL + IL_L \quad (58)$$

$$a_{yN} = e \sin \omega + a_{yNL} \quad (59)$$

To add the short period terms, the algorithm first solves Kepler's equation:

$$U = IL_T - \Omega \quad (60)$$

Then the iterative equation is:

$$(E + \omega)_{i+1} = (E + \omega)_i + \Delta(E + \omega)_i \quad (61)$$

With the initial condition:

$$(E + \omega)_1 = U \quad (62)$$

And:

$$\Delta(E + \omega)_1 = \frac{U - a_{yN} \cos(E + \omega)_i + a_{xN} \sin(E + \omega)_i - (E + \omega)_i}{1 - a_{yN} \sin(E + \omega)_i - a_{xN} \cos(E + \omega)_i} \quad (63)$$

The convergence tolerance for  $(E + \omega)$  in CLASS was chosen to be  $10^{-6}$ . Once the iterations converge, the following quantities are used to calculate short term periodics:

$$e \cos E = a_{xN} \cos(E + \omega) + a_{yN} \sin(E + \omega) \quad (64)$$

$$e \sin E = a_{xN} \sin(E + \omega) - a_{yN} \cos(E + \omega) \quad (65)$$

$$e = \sqrt{a_{xN}^2 + a_{yN}^2} \quad (66)$$

$$p_L = a(1 - e^2) \quad (67)$$

$$r = a(1 - e \cos E) \quad (68)$$

$$\dot{r} = k_e \frac{\sqrt{a}}{r} e \sin E \quad (69)$$

$$r\dot{f} = k_e \frac{\sqrt{p_1}}{r} \quad (70)$$

$$\cos u = \frac{a}{r} \left[ \cos(E + \omega) - a_{xN} + \frac{a_{yN}e \sin E}{1 + \sqrt{1 - e^2}} \right] \quad (71)$$

$$\sin u = \frac{a}{r} \left[ \sin(E + \omega) - a_{yN} - \frac{a_{xN}e \sin E}{1 + \sqrt{1 - e^2}} \right] \quad (72)$$

$$u = \tan^{-1} \left( \frac{\sin u}{\cos u} \right) \quad (73)$$

$$\Delta r = \frac{k_2}{2p_L} (1 - \cos^2 i) \cos 2u \quad (74)$$

$$\Delta u = -\frac{k_2}{4p_L^2} (7 \cos^2 i - 1) \sin 2u \quad (75)$$

$$\Delta \Omega = \frac{3k_2 \cos i}{2p_L^2} \sin 2u \quad (76)$$

$$\Delta i = \frac{3k_2 \cos i}{2p_L^2} \sin i \cos 2u \quad (77)$$

$$\Delta \dot{r} = -\frac{k_2 n}{p_L} (1 - \cos^2 i) \sin 2u \quad (78)$$

$$\Delta r\dot{f} = \frac{k_2 n}{p_L} \left[ (1 - \cos^2 i) \cos 2u - \frac{3}{2} (1 - 3 \cos^2 i) \right] \quad (79)$$

The osculating quantities are computed as:

$$r_k = r \left[ 1 - \frac{3}{2} k_2 \frac{\sqrt{1 - e^2}}{p_L^2} (3 \cos^2 i - 1) \right] + \Delta r \quad (80)$$

$$u_k = u + \Delta u \quad (81)$$

$$\Omega_k = \Omega + \Delta \Omega \quad (82)$$

$$i_k = i + \Delta i \quad (83)$$

$$\dot{r}_k = \dot{r} + \Delta \dot{r} \quad (84)$$

$$r\dot{f}_k = r\dot{f} + \Delta r\dot{f} \quad (85)$$

A set of unit vectors are then used to calculate the new orbit position:

$$\mathbf{M} = [-\sin \Omega_k \cos i_k \quad \cos \Omega_k \cos i_k \quad \sin i_k]^T \quad (86)$$

$$\mathbf{N} = [\cos \Omega_k \quad \sin \Omega_k \quad 0]^T \quad (87)$$

$$\mathbf{U} = \mathbf{M} \sin u_k + \mathbf{N} \cos u_k \quad (88)$$

$$\mathbf{V} = \mathbf{M} \cos u_k - \mathbf{N} \sin u_k \quad (89)$$

Finally, the new orbital position and velocity vectors in the ECI J2000 frame are calculated by CLASS.

$$\mathbf{r}^I = r_k \mathbf{U} \quad (90)$$

$$\mathbf{v}^I = \dot{r}_k \mathbf{U} + r \dot{\phi}_k \mathbf{V} \quad (91)$$

The equations in this section describe the algorithm for the SGP4 propagator coded in CLASS. Other implementation of various orbit models could prove to be more accurate at high altitudes (500 km +) but were deemed unnecessary for applications to LEO missions with periods of approximately 90 minutes. No perturbation models from the Moon's and the Sun's gravity were added to the algorithm as they only become significant for orbit periods higher than 225 minutes [40]. For applications to future missions at higher altitudes, the addition of more perturbation models may be desirable. Also, the addition of a more reliable long duration orbit propagator is desirable.

### 3.4 MAGNETIC FIELD MODEL

#### 3.4.1 International Geomagnetic Reference Field

Since 1965, a global collaborative effort, led by the International Association of Geomagnetism and Aeronomy (IAGA), has produced and maintained several versions of the International Geomagnetic Reference Field (IGRF) [44]. The IGRF is a collection of mathematical models that are used to calculate the theoretical Earth magnetic field values in a certain geographical location. In addition to the mathematical models, the IGRF relies heavily on data collected using ground magnetic observatories and satellites.

The magnetic field produced by the Earth's core has secular variation terms. As a result, the IGRF model is updated almost every five years [44]. The current version is the 13<sup>th</sup> such iteration, valid from 1900-2025. The predecessor, the 12<sup>th</sup> version, is valid for 1900-2020. CLASS contains both the 12<sup>th</sup> and the 13<sup>th</sup> IGRF version.

The magnetic field vector is expressed as the gradient of a potential function, much like gravity [45]:

$$\mathbf{B} = -\nabla V \quad (92)$$

And in spherical coordinates, the potential function is [44]:

$$V(r, \theta, \phi, t) = a \sum_{n=1}^N \sum_{m=0}^n \left(\frac{a}{r}\right)^{n+1} [g_n^m(t) \cos(m\phi) + h_n^m(t) \sin(m\phi)] P_m^n \cos \lambda' \quad (93)$$

Where:

$a$  = 6371.2 km Earth mean reference spherical radius according to the geomagnetic convention

$r$  = radial distance from the Earth's center in km

$\lambda'$  = geocentric latitude (in CLASS, positive value for Northward from equator)

$\phi$  = longitude (in CLASS, positive value for Eastward from equator)

$P_m^n(\cos \theta)$  = Schmidt quasi-normalized associated Legendre functions of order  $n$  and degree  $m$

$N$  = the truncation degree set at 13 to exclude crustal magnetic field influences [46]

The coefficients  $g_n^m(t)$  and  $h_n^m(t)$  are referred to as the Gauss coefficients. They change over time and are updated in every release of the IGRF model. Their change during an interval of five years is assumed to be linear [44]. The tabulated values for  $g(t)$ ,  $h(t)$ ,  $n$ , and  $m$  with respect to time can be found in [47]. The coefficients for both the IGRF 12<sup>th</sup> and 13<sup>th</sup> generations are included in CLASS.

As documented in [45], the gradient of each term in Equation 92 has the following form:

$$\mathbf{B} = \sum_{n=1}^N \sum_{m=0}^n \mathbf{B}_{n,m} \quad (94)$$

$$\mathbf{B}_{n,m} = \frac{K_{n,m} a^{n+2}}{r^{ECEF^{n+m+1}}} \left\{ \begin{array}{l} \frac{g_{n,m} C_m + h_{n,m} S_m}{r^{ECEF}} [(\sin \lambda' A_{n,m+1} + (n+m+1) A_{n,m}) \hat{\mathbf{r}}^{ECEF} - A_{n,m+1} \hat{\mathbf{e}}_3] \\ -mA_{n,m} [(g_{n,m} C_{m-1} + h_{n,m} S_{m-1}) \hat{\mathbf{e}}_1 + (h_{n,m} C_{m-1} - g_{n,m} S_{m-1}) \hat{\mathbf{e}}_2] \end{array} \right\} \quad (95)$$

Where:

$r^{ECEF}$  = magnitude of the satellite position (or any position of interest) expressed in the ECEF frame

$\hat{\mathbf{r}}^{ECEF}$  = normalized satellite position vector expressed in the ECEF frame

$\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3$  = unit standard basis vectors of the ECEF frame [1,0,0], [0,1,0], [0,0,1]

$K_{n,m}$  = coefficients described in [45] as:

- $K_{1,0} = 1$
- $K_{1,1} = 1$
- $K_{n,0} = 1$
- $K_{n,1} = \sqrt{\frac{n-m}{n+m}} K_{n-1,1}$
- $K_{n,m} = [(n+m)(n-m+1)]^{-\frac{1}{2}} K_{n,m-1}$  where  $m = [2, n]$  and  $n = [2, 13]$

$S_m, C_m$  = defined in [45] as

- $S_0 = 0$
- $C_0 = 1$
- $S_1 = \mathbf{r}^{ECEF} \cdot \hat{\mathbf{e}}_2$
- $C_1 = \mathbf{r}^{ECEF} \cdot \hat{\mathbf{e}}_1$
- $S_m = S_1 C_{m-1} + C_1 S_{m-1}$  where  $m > 1$
- $C_m = C_1 C_{m-1} + S_1 S_{m-1}$  where  $m > 1$

$A_{n,m}, A_{n,m+1}$  are the derived Legendre degree  $n$  and degree  $m + 1$  polynomials [45]. The polynomials can be expressed as functions of an argument  $u$  (ie:  $A_{n,m}(u)$ ). The polynomial general expression is:

$$A_{n,m}(u) = \frac{1}{n - m} [(2n - 1)uA_{n-1,m}(u) - (n + m - 1)A_{n-2,m}(u)] \quad (96)$$

Where  $m = 0, \dots, \infty$  and  $n \geq (m + 1)$ .

Note that if  $n = m$ , then, the expression in Equation 96 is not a function of  $u$ :

$$A_{n,n} = (2n - 1)A_{n-1,n-1} \quad (97)$$

All iterations and computations of the IGRF algorithm are performed in CLASS and the software outputs a magnetic field vector expressed in any desired reference frame for any given position input (also expressed in any reference frame).

### 3.5 SPACE ENVIRONMENT

When it comes to satellite attitude dynamics, the space environment can exert external disturbance torques on the satellite body. In Low Earth Orbit the three most dominant environmental disturbances are atmospheric drag, gravity gradient torques, and solar radiation pressure. It is important to include these disturbances in the dynamic simulation because they can have significant effects on the satellite dynamic response in orbit.

#### 3.5.1 Aerodynamic Torque

According to [48], the aerodynamic disturbance torque is the most dominant environmental disturbance torque for orbit altitudes below 800 km. In fact, its dominance increases exponentially as satellites get closer to the Earth's surface [48]. The task of modeling satellite drag in orbit is a tedious undertaking [49]. Aerodynamic drag on a satellite depends on the drag coefficient, the molecular behavior of the LEO atmosphere affecting air density, the relative velocity vector of the satellite, and the area of the satellite surface perpendicular to the relative velocity vector.

When assessing the orbit lifetime of a satellite, a commonly used parameter is the *Ballistic Coefficient (BC)* [50] [49] [48]:

$$BC = \frac{m}{A \cdot C_D} \quad (98)$$

Where

$m$  = mass of the satellite

$C_D$  = coefficient of drag

$A$  = cross sectional area of the satellite

The Ballistic Coefficient is an indicator of the satellite's ability to resist air drag [48]. So, a larger coefficient normally correlates to a larger orbit lifetime.

To calculate the aerodynamic disturbance, CLASS begins by calculating the velocity of the satellite relative to the atmosphere. According to [50], it is common to assume that the atmosphere is a body rotating at the same rate as Earth's rotation rate. So, the relative velocity of the satellite is expressed in the inertial frame as the vector sum of the satellite's inertial velocity and the atmosphere's inertial velocity at the orbital position of the satellite:

$$\mathbf{v}_{\text{rel}}^I = \mathbf{v}^I + \mathbf{r}^I \times \boldsymbol{\omega}_{\text{Earth}} \quad (99)$$

$\boldsymbol{\omega}_{\text{Earth}}$  is the Earth's angular rate and it is equal to  $[0 \ 0 \ 0.000072921158553]^T$  rad/s. The relative velocity is then expressed in the body frame:

$$\mathbf{v}_{\text{rel}}^B = \mathbf{R}_I^B \mathbf{v}_{\text{rel}}^I \quad (100)$$

The drag force is calculated on all faces of the CubeSat. So, the surface area of the  $i$ -th plate is defined as  $S_i$  and the angle between the relative velocity vector and the plate  $\theta_i$  is calculated as:

$$\theta_i = \frac{(\mathbf{n}_i^B \cdot \mathbf{v}_{\text{rel}}^B)}{||\mathbf{v}_{\text{rel}}^B||} \quad (101)$$

Such that  $\mathbf{n}_i^b$  is the unit vector normal to the  $i$ -th plate. This unit vector is obtained in CLASS using the attitude simulation of the CubeSat. During every iteration, CLASS computes the estimated aerodynamic force on each of the CubeSat's surfaces using:

$$\mathbf{F}_{\text{aero } i}^b = -\frac{1}{2} \rho C_D V_{\text{rel}}^b \mathbf{v}_{\text{rel}}^b S_i \max(\cos \theta_i, 0) \quad (102)$$

Where

$\rho$  = atmospheric density depending on the air density model adopted

$C_D$  = coefficient of drag of CubeSats is usually experimentally determined to be between 1.5 and 2.5 [50]. In CLASS, for a 3U CubeSat, the coefficient of drag is assigned to be 2.2 because it is typical for spacecraft in LEO [51] [52].

The  $\max(\cos \theta_i, 0)$  term determines whether the normal vector to the  $i$ th plate is projected opposite to (i.e., for a negative  $\cos \theta$  where the plate is not facing the flow, then drag should be zero) or along the relative velocity vector (i.e., a positive  $\cos \theta$ ).

The total disturbance torque caused by aerodynamics is then computed by CLASS as the sum of the torques caused by every drag force on each plate:

$$\mathbf{T}_{\text{aero}}^b = \sum_{i=1}^N \mathbf{r}_{\text{cm}_{\text{cp}} i}^b \times \mathbf{F}_{\text{aero } i}^b \quad (103)$$

$\mathbf{r}_{\text{cm}_{\text{cp}} i}^b$  = vector between the spacecraft's center of mass and the center of pressure of the  $i$ th plate

### 3.5.2 Gravity Gradient Torque

Nonsymmetrical rigid bodies placed in non-uniform gravitational fields will experience gravity-gradient torques [30] [48]. This environmental torque is caused by the fundamental property of the Newtonian gravity model: the force of gravity between two objects varies inversely with the square of the distance between them. The relatively small size of CubeSats does not rule

out the effects of gravity gradient torques on attitude dynamics. The torque is commonly modelled in CubeSat attitude simulations and the gravity field is approximated as being spherically symmetric:

$$\mathbf{g}(\mathbf{r}) = -\frac{\mu_{\text{Earth}}}{r^3} \mathbf{r} \quad (104)$$

Where:

$\mathbf{r}$  = position vector from the center of the Earth to the particle of interest

The gravitational forces on each mass particle of the rigid body spacecraft are summed to derive the below final expression for gravity gradient torque in the satellite body frame [48]:

$$\mathbf{T}_{\text{gg}}^b = \frac{3\mu_{\text{Earth}}}{r^3} [\hat{\mathbf{n}}^b \times (\mathbf{J}^b \hat{\mathbf{n}}^b)] \quad (105)$$

$\mathbf{n}^b$  = unit vector pointing in the nadir direction and expressed in the body frame

The unit nadir vector can be obtained using the LVLH frame  $z$ -axis representation expressed in Equation 12. Keep in mind that Equation 12 expresses the nadir vector in the inertial reference frame. To obtain the vector  $\mathbf{n}^b$  the below frame transformation is applied:

$$\hat{\mathbf{n}}^b = \mathbf{R}_I^b \hat{\mathbf{o}}_3 = \mathbf{R}_I^b \left( -\frac{\mathbf{r}^I}{||\mathbf{r}^I||} \right) \quad (106)$$

### 3.5.3 Solar Radiation Pressure

Currently, CLASS does not contain an environmental model that can emulate torque disturbances caused by solar radiation pressure (SRP). According to [30] [48] [53], SRP is dominated by aerodynamic disturbance torques at altitudes lower than 800 km. In order to add a simulation of SRP, the following expression for a force vector on the  $i$ th plate can be used:

$$\mathbf{F}_{\text{SRP } i}^b = P_{\text{SRP}} S_i \left[ 2 \left( \frac{R_{\text{diff } i}}{3} + R_{\text{spec } i} \cos \theta_{\text{SRP } i} \right) \mathbf{n}_i^b + (1 - R_{\text{spec } i}) \mathbf{s}^b \right] \max(\cos \theta_{\text{SRP } i}, 0) \quad (107)$$

Where

$S_i$  = surface area of the  $i$ th plate

- $\mathbf{n}_i^b$  = outward normal unit vector to the *i*th plate expressed in the body coordinate frame  
 $\mathbf{s}^b$  = unit vector representing the Sun position with respect to the satellite expressed in the body frame (i.e., the Sun vector)  
 $\theta_{SRP\ i}$  = angle between the normal vector  $\mathbf{n}_i^b$  to the *i*th plate and the Sun vector  $\mathbf{s}^b$  calculated using  $\cos \theta_{SRP\ i} = \mathbf{n}_i^b \cdot \mathbf{s}^b$   
 $R_{spec\ i}$  = specular reflection coefficient  
 $R_{diff\ i}$  = diffuse reflection coefficient  
 $P_{SRP}$  = solar radiation pressure

To determine the solar radiation pressure value, the orbit simulation in CLASS must provide the position vector  $\mathbf{r}_{sat\ to\ sun}^l$  between the satellite and the Sun in the ECI frame by computing the satellite's position relative to the Sun, Moon, and the Earth's horizon [30]. The pressure is computed using:

$$P_{SRP} = \frac{\mathcal{F}}{c (r_{sat\ to\ sun})^2} \quad (108)$$

Where:

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

The external torque due to the solar radiation pressure expressed in the body coordinate reference frame is calculated as:

$$\mathbf{T}_{SRP}^b = \sum_{i=1}^N \mathbf{r}_{cm_{csrp\ i}}^b \times \mathbf{F}_{SRP\ i}^b \quad (109)$$

$\mathbf{r}_{cm_{csrp\ i}}^b$  = vector between the satellite center of mass and the center of pressure of SRP

## CHAPTER 4: REAL TIME SIMULATIONS

### 4.1 OVERVIEW OF THE DYNAMIC SIMULATION SOFTWARE

A Raspberry Pi (RPi) 4 was selected as the microprocessor to run the satellite dynamics simulation in real-time. However, in its native configuration, the Raspbian operating system installed on the RPi is not capable of running applications in real-time. To overcome this shortcoming, a Linux patch called PREEMPT\_RT was installed to configure the kernel to perform real-time operations. The patch provides the necessary source codes to transform the current operating system into an emulation of a Real Time Operating System (RTOS) [54].

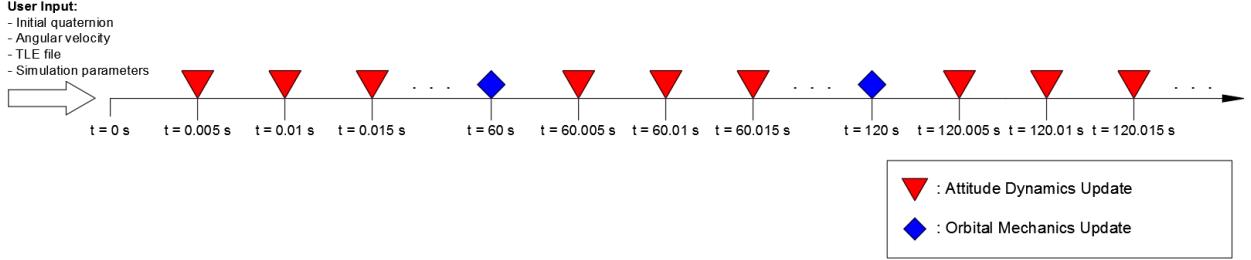
A task, running on an operating system, refers to a singular computer program being executed, and the information associated with that program. In a generic operating system, the tasks are scheduled according to a “policy” that leads to a balanced and stable performance [55]. In an RTOS, the scheduling policy is designed so that tasks meet timing deadlines and respond with satisfactory speed. The RTOS scheduler switches between tasks with the goal of executing the tasks in decreasing order of priority and within the expected time deadline. The execution model implemented by PREEMPT\_RT is based on POSIX threads or “pthreads” [56]. Defined by the *IEEE 1003.1c-1995* standard, each task is called a thread and the initiation and management of the threads is performed using the POSIX Threads API [57]. This execution model allows the Raspbian operating system to act as an RTOS.

There are three threads in the simulation code. The CPU Linux scheduler used is SCHED\_DEADLINE to insure that each satellite dynamics calculation is performed before a strict time-step deadline. The first thread, which has the highest execution priority, consists of the attitude dynamics integration and the calculation of the magnetic field vector expressed in the satellite body frame. This thread is of highest priority. It has the smallest time-step and the least

amount of time to be executed. The second thread is the orbit propagator. And the third thread is an idle task to allow overhead functions to execute when higher priority tasks have completed.

Every five milliseconds, the attitude of the satellite is updated (i.e., a new quaternion for the rotation from the body frame to the ECI frame is generated and the angular velocity in the body frame is calculated) and the magnetic field vector experienced by the satellite in the body frame is calculated. If the program failed to fully execute before the five-millisecond deadline, the SCHED\_DEADLINE scheduling policy overrides the current task and moves on to execute the next task. This configuration avoids the situation of having the code stuck on a task of highest priority longer than expected and causing the satellite dynamics simulation to become out of sync with the real-time subsystem being tested.

Usually, the code updates the new attitude in less than five milliseconds. So, after an attitude integration is complete, the code switches the task execution to the orbit propagator, which is the task with the second highest priority. The time-step for an orbit position and velocity update is sixty seconds. After each update of the attitude and orbit states, the states are sent to the serial buffer, making the data available to be extracted in real-time by the ADCS sensor emulations and another PC for data logging and live dynamics visualization. Additionally, in every iteration of the attitude dynamics, the code checks for any torque commands sent to the real-time simulation through the serial port to apply it to the dynamics. The third thread is simply an idle task. Figure 8 is a schematic representing the code execution cycles and dynamics updates with time.



**Figure 8:** Real-Time Dynamics Simulation Software Task Flow.

## 4.2 ATTITUDE DYNAMICS SIMULATION

The satellite attitude dynamics simulation runs in real-time. Using Equations 13 and 14, the quaternion and angular velocity vectors are integrated over a time-step of five milliseconds using the Runge-Kutta 4 integration method. The following sections demonstrate how the equations of motion are computed at each time-step. The moment of inertia of a hypothetical CubeSat used in the subsequent demonstrations is:

$$\mathbf{J}^b = \begin{bmatrix} 0.0275 & 0 & 0 \\ 0 & 0.04 & 0 \\ 0 & 0 & 0.0075 \end{bmatrix} \text{ kg}\cdot\text{m}^2$$

### 4.2.1 Attitude Dynamics Simulation Demonstration 1

The simulation in this demonstration is executed with no external torques applied to the CubeSat. The expected behavior is identical to a rotating rigid body in space with no acceleration applied.

The initial conditions are:

$$\mathbf{q}_b^I = [0.427 \quad 0.468 \quad 0.137 \quad 0.762]^T$$

$$\boldsymbol{\omega}^b = [2.3 \quad -0.5 \quad 1.2]^T \text{ deg/sec}$$

$$\mathbf{T}^b = [0 \quad 0 \quad 0]^T \text{ N.m}$$

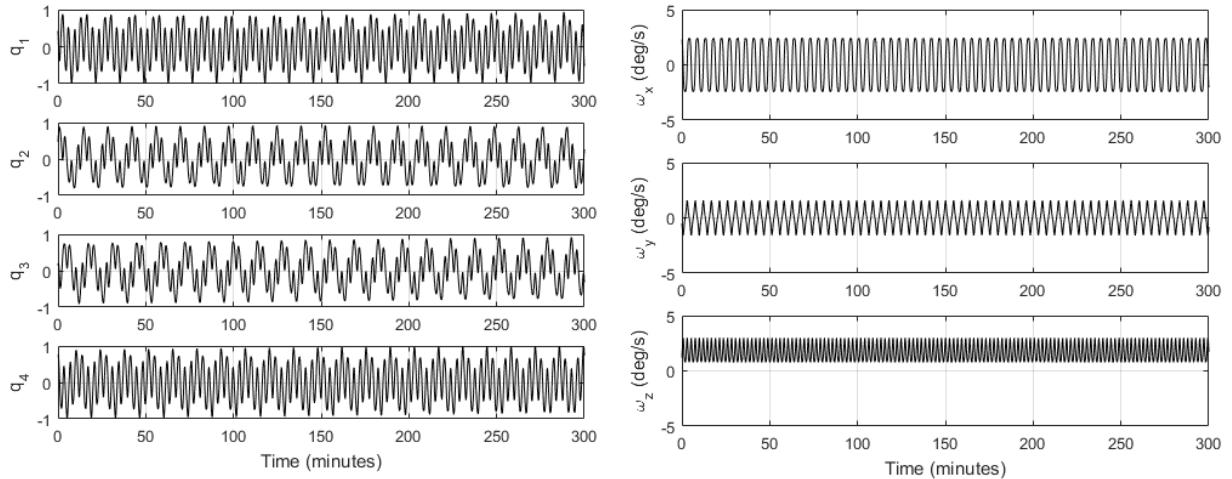
No external torque is applied. Angular momentum expressed in an inertial reference frame should be conserved because no external force is acting on the rigid body. The angular momentum expressed in the body frame is equal to:

$$\mathbf{H}^b = \mathbf{J}^b \boldsymbol{\omega}^b \quad (110)$$

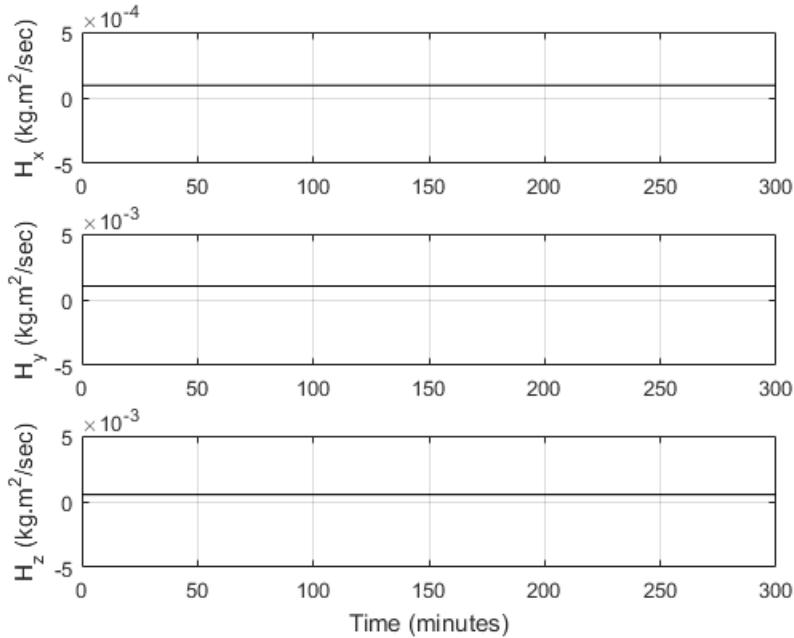
The Cartesian components of the angular momentum and angular velocity expressed in the body frame vary with time because the body axes are inertially rotating. When expressed in the inertial frame, using Equation 111, where  $\mathbf{R}_b^I$  is obtained instantaneously as the quaternion vector  $\mathbf{q}_b^I$  is updated in the simulation by populating the expression in Equation 15, the angular momentum is conserved. The initial angular momentum is equal to:

$$\mathbf{H}^I = \mathbf{R}_b^I \mathbf{H}^b = [0.0000941 \quad 0.0010388 \quad 0.0005282] \text{ kg.m}^2/\text{sec} \quad (111)$$

As demonstrated in Figure 10 and Table 3, the angular momentum was conserved in this simulation, thus, validating the attitude dynamics model in CLASS. The simulated angular momentum value is equal to the manually calculated value in Equation 111.



**Figure 9:** Simulation results for Demonstration 1. Quaternion vector elements for body frame to inertial frame rotation vs time (minutes) (left). Angular velocity (deg/sec) vector elements expressed in the body frame vs time (minutes) (right). These results are for a torque-free rotation of a satellite in CLASS.



**Figure 10:** Simulation results for Demonstration 1. Angular momentum (kg.m<sup>2</sup>/sec) vector elements expressed in the body frame vs time (minutes). The angular momentum of the torque-free rotating satellite in CLASS is conserved with time as expected.

Table 3 provides the numerical residual error values for this demonstration. The small error values are the result of numerical precision “noise” and are adequate for the purposes of this demonstration.

**Table 3:** Simulation Results for Demonstration 1. Angular momentum (kg.m<sup>2</sup>/sec) values from manual calculations compared to simulation results. The expected hand-calculated values of the angular momentum match the constant values in the simulation.

	Expected Value	Value from Simulation	Error (%)
$H_x$ (kg.m <sup>2</sup> /sec)	0.0000941	0.0000941	0
$H_y$ (kg.m <sup>2</sup> /sec)	0.0010388	0.0010381	0.07
$H_z$ (kg.m <sup>2</sup> /sec)	0.0005282	0.0005279	0.06

#### 4.2.2 Attitude Dynamics Simulation Demonstration 2

The simulation in this demonstration represents a CubeSat initially rotating at a constant rate up until an impulse torque is applied at  $t = 100$  min. The expected behavior is a change in the angular momentum of the CubeSat in the direction of the disturbance torque.

The initial conditions are:

$$\mathbf{q}_b^I = [0.61 \quad 0.453 \quad -0.618 \quad -0.204]^T$$

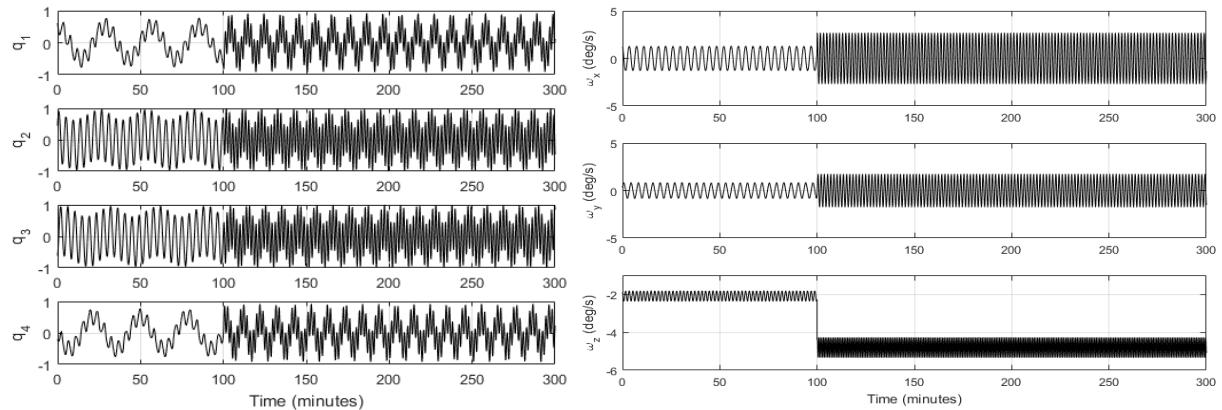
$$\boldsymbol{\omega}^b = [1.2 \quad 0.3 \quad -1.9]^T \text{ deg/s}$$

$$\mathbf{T}^b = [0 \quad 0 \quad 0]^T \text{ N.m}$$

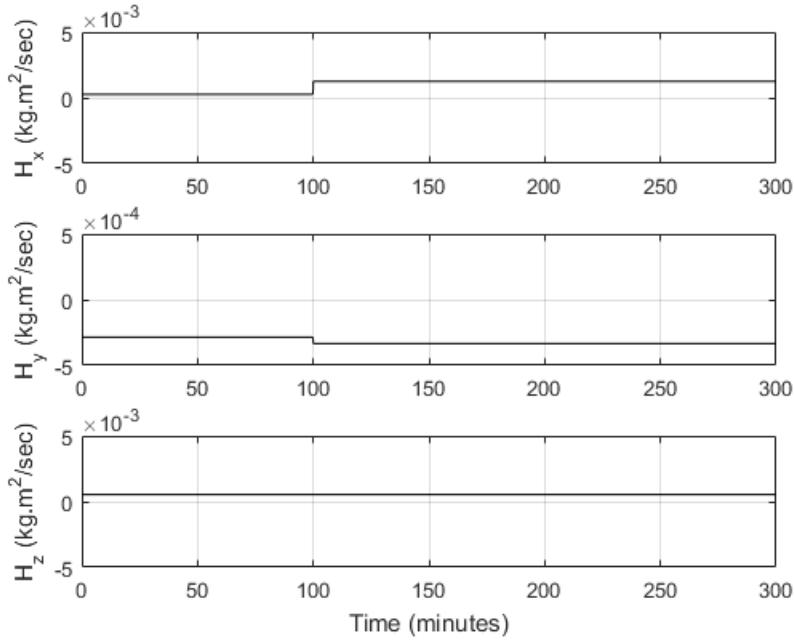
In this demonstration, at time  $t = 100$  min, an impulse torque is applied, and it is equal to:

$$\mathbf{T}^I = [0.02 \quad -0.001 \quad 0]^T \text{ N.m}$$

The above torque vector is expressed in the inertial reference frame. This means that angular momentum along the inertial x-direction will be increased in the positive direction, the angular momentum along the inertial y-direction will be increased in the negative direction, and the angular momentum along the inertial z-direction will remain the same. This expected behavior is demonstrated in Figure 12 and Table 4.



**Figure 11:** Simulation results for Demonstration 2. Quaternion vector elements for body frame to inertial frame rotation vs time (minutes) (left). Angular velocity (deg/sec) vector elements expressed in the body frame vs time (minutes) (right). These results are for a spinning satellite that undergoes an external impulsive torque at  $t = 100$  min.



**Figure 12:** Simulation results for Demonstration 2. Angular momentum ( $\text{kg} \cdot \text{m}^2/\text{sec}$ ) vector elements expressed in the body frame vs time (minutes). Due to the impulsive external torque applied at  $t = 100$  min, as expected the  $x$  and  $y$  components of the angular momentum changed accordingly while the  $z$  component remains constant because no torque in the  $z$ -direction was applied.

Table 4 provides the numerical values for the angular momentum before and after the impulse torque was applied.

**Table 4:** Simulation Results for Demonstration 2. Angular momentum ( $\text{kg} \cdot \text{m}^2/\text{sec}$ ) values before and after torque was applied.

	<b>Value at <math>t &lt; 100</math> min</b>	<b>Value at <math>t &gt; 101</math> min</b>
$H_x (\text{kg} \cdot \text{m}^2/\text{sec})$	0.0002567	0.001257
$H_y (\text{kg} \cdot \text{m}^2/\text{sec})$	-0.0002846	-0.0003343
$H_z (\text{kg} \cdot \text{m}^2/\text{sec})$	0.000539	0.000539

#### 4.3 ORBITAL MECHANICS SIMULATION

As mentioned in Chapter 3, the orbit propagator algorithm used in CLASS is SGP4. The initial orbital elements are entered by uploading a TLE .txt file to initialize the real-time orbit

simulation software in CLASS. To demonstrate the performance of the orbit propagator, the following TLE file for the International Space Station (ISS) in 2020 was uploaded to CLASS:

```
ISS (ZARYA)
1 25544U 98067A 20107.22393519 .00001546 00000-0 36590-4 0 9997
2 25544 51.6447 295.1279 0003731 120.2243 215.8424 15.48698545222403
```

**Figure 13:** TLE file for the International Space Station obtained from [58]

The epoch orbital elements are extracted appropriately by CLASS:

$$t_0 = 154402 \text{ minutes}$$

$$n_0 = 0.0675747 \text{ rad/min} = 3.871745^\circ/\text{min}$$

$$e_0 = 0.0003731$$

$$i_0 = 0.90137 \text{ rad} = 51.6447^\circ$$

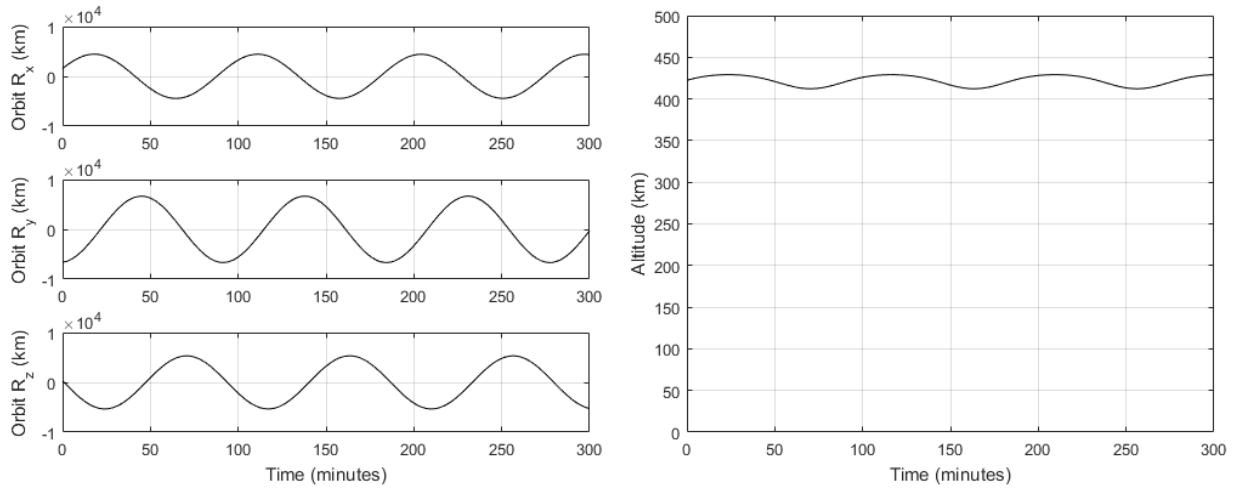
$$\omega_0 = 2.09831 \text{ rad} = 120.2243^\circ$$

$$\Omega_0 = 5.15095 \text{ rad} = 295.1279^\circ$$

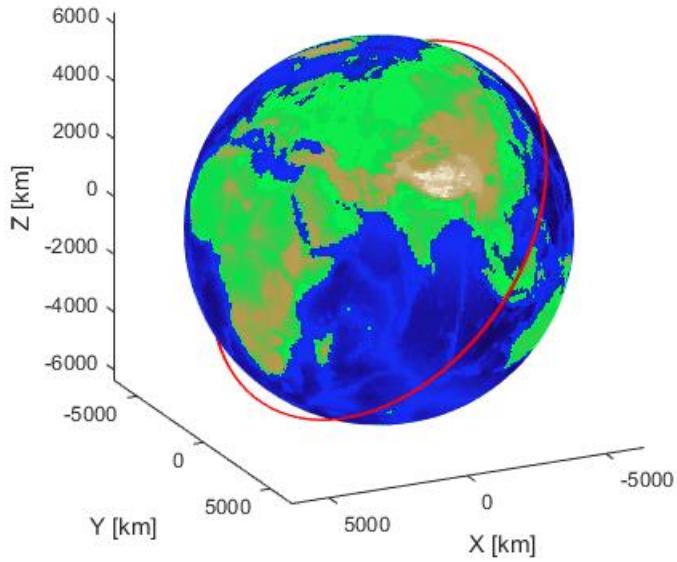
$$M_0 = 3.76716 \text{ rad} = 215.8424^\circ$$

$$B^* = 3.659 \times 10^{-5} [\text{1/R}_{\text{Earth}} \text{ units}]$$

The simulation results for the first 300 minutes indicate an apogee of 6807 km (i.e., an altitude of 429 km using the Equatorial Earth radius 6378 km) and a perigee of 6790 km (i.e., an altitude of 412 km). The ISS operates at a mean altitude of 400 km [59]. Plots of the simulated orbit position and altitude vs time are illustrated in Figure 14 and a visualization of the orbit is drawn in Figure 15. Online ISS tracking websites indicate, as of mid-April 2020 (i.e., the time this simulation was run), an apogee height of 423 km and a perigee height of 418 km [60]. The orbital period determined by CLASS is 92.83 minutes, which is the expected ISS orbit period at such altitudes [59].



**Figure 14:** Simulation results for ISS orbit in CLASS. The orbital position vector elements (km) expressed in the ECI J2000 Inertial Frame vs time (minutes) (left). The simulated altitude (km) of the ISS using the Equatorial radius of the Earth (6378 km) vs time (minutes) (right). The results show the first three orbits and the perigee, apogee, and period values agree with the known ISS orbit parameters.



**Figure 15:** Simulation results for ISS orbit in CLASS. 3D visualization of the orbital position expressed in the ECI J2000 Inertial Frame.

The semi-major axis for the first three orbits in the first 300 minutes can be calculated using the perigee and the epoch eccentricity (which is not expected to change much after only three orbits) [61]:

$$a = \frac{r_{\text{perigee}}}{1 - e_0} = 6792.534295 \text{ km} \quad (112)$$

Using the calculated semi-major axis, one form of validating the orbit simulation is to check the calculated orbit period:

$$T = 2\pi \sqrt{\frac{a^3}{\mu_{\text{Earth}}}} = 5571.331 \text{ sec} = 92.85 \text{ min} \quad (113)$$

Which is compliant with the orbital period simulated in CLASS (i.e., 92.83 min).

The expected orbital velocities at apogee and perigee calculated using Equations 114 and 115 respectively also match the simulated values in CLASS as shown in Table 5.

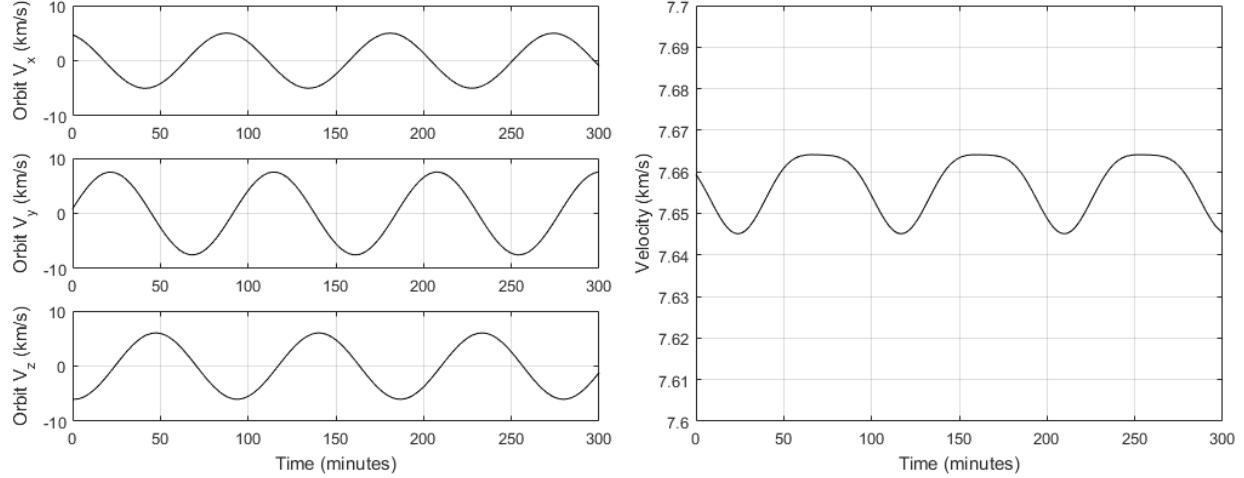
$$v_{\text{apogee}} = \sqrt{\frac{\mu_{\text{Earth}}(1 - e_0)}{r_{\text{apogee}}}} \quad (114)$$

$$v_{\text{perigee}} = \sqrt{\frac{\mu_{\text{Earth}}(1 + e_0)}{r_{\text{perigee}}}} \quad (115)$$

**Table 5:** Simulation results for ISS orbit in CLASS. Orbital velocity (km/sec) values for apogee and perigee from manual calculations compared to simulation results. The expected hand-calculated values of the orbital velocities match the values in the simulation.

	Expected Value	Simulated Value	Error (%)
<b>Apogee Velocity (km/sec)</b>	7.653	7.645	0.106
<b>Perigee Velocity (km/sec)</b>	7.663	7.664	0.013

Figure 15 demonstrates plots of the orbital velocity vector elements expressed in the inertial frame vs time and the orbital velocity magnitude vs time.



**Figure 16:** Simulation results for ISS orbit in CLASS. The orbital velocity vector elements (km/sec) expressed in the ECI J2000 Inertial Frame vs time (minutes) (left). The simulated velocity magnitude (km/sec) of the ISS vs time (minutes) (right). The results show the first three orbits and the perigee and apogee velocities agree with the expected ISS orbit velocities.

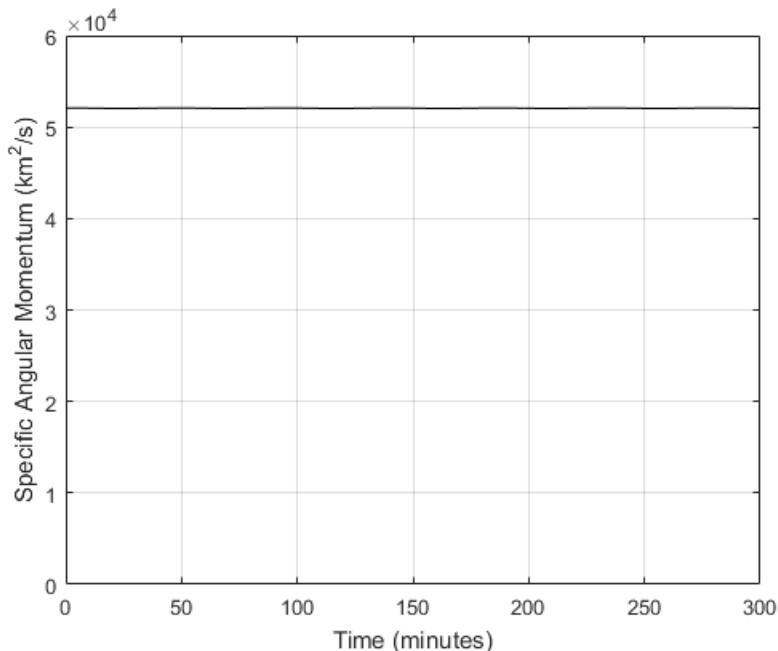
Further orbital mechanics validation is done by examining the angular momentum of the first three orbits. Because of gravitational, atmospheric, and solar perturbations, the orbit of a satellite precesses with time and the angular momentum does not remain constant [61]. However, for the first three orbits, the specific angular momentum (angular momentum per unit mass) can be considered constant because the secular variation of the orbital elements is negligible over such a short time frame. The magnitude of the angular momentum per unit mass is expected to be [61]:

$$h = \sqrt{\mu_{\text{Earth}} a (1 - e_0^2)} = 52033.679 \text{ km}^2/\text{sec} \quad (116)$$

The simulation is programmed to calculate the angular momentum using:

$$\mathbf{h}^I = \mathbf{r}^I \times \mathbf{v}^I \quad (117)$$

As indicated in Figure 17, the specific angular momentum appears to remain conserved with time for the first three orbits.



**Figure 17:** Simulation results for ISS orbit in CLASS. Specific angular momentum ( $\text{km}^2/\text{sec}$ ) of the orbital motion vs time (minute) for the first three orbits. For only three orbits, the environmental and gravitational perturbations should negligibly affect the orbital parameters and the angular momentum is not expected to change much.

Table 6 demonstrates the agreement between the expected specific angular momentum and the simulated specific angular momentum. The small error values are the result of numerical precision “noise” and are adequate for the purposes of this demonstration.

**Table 6:** Simulation results for ISS orbit in CLASS. Specific angular momentum ( $\text{km}^2/\text{sec}$ ) of the orbital motion generated by the simulation matches the expected value and appears to remain conserved with time. This is applicable for the first three orbits because the perturbation terms are negligible for such a short time frame.

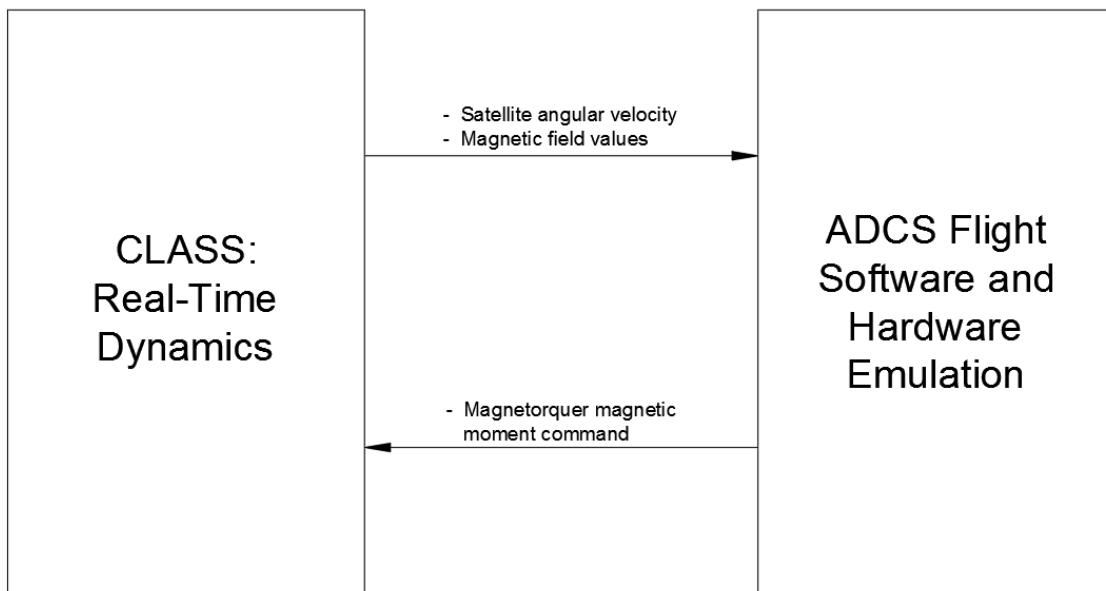
	<b>Expected Value</b>	<b>Simulated Value</b> $t < 90 \text{ min}$	<b>Simulated Value</b> $90 \text{ min} < t < 180 \text{ min}$	<b>Simulated Value</b> $180 \text{ min} < t < 270 \text{ min}$
<b>Specific Angular Momentum (<math>\text{km}^2/\text{sec}</math>)</b>	52033	52070	52070	52060

## CHAPTER 5: CLOSED LOOP TESTS

To demonstrate CLASS's capabilities, an ADCS performance test for LASSI's CAPSat is presented.

### 5.1 TEST SETUP

CAPSat's ADCS software runs in a daemon on the Sitara AM3354 flight computer. CLASS has the ability to interface with the flight computer using RS-422. However, because of limited access to the flight computer, an emulation of the ADCS flight software source code was encapsulated inside the CLASS software architecture and interfaced with the real-time dynamic simulation. Figure 18 illustrates the interface diagram between the ADCS emulation and CLASS.



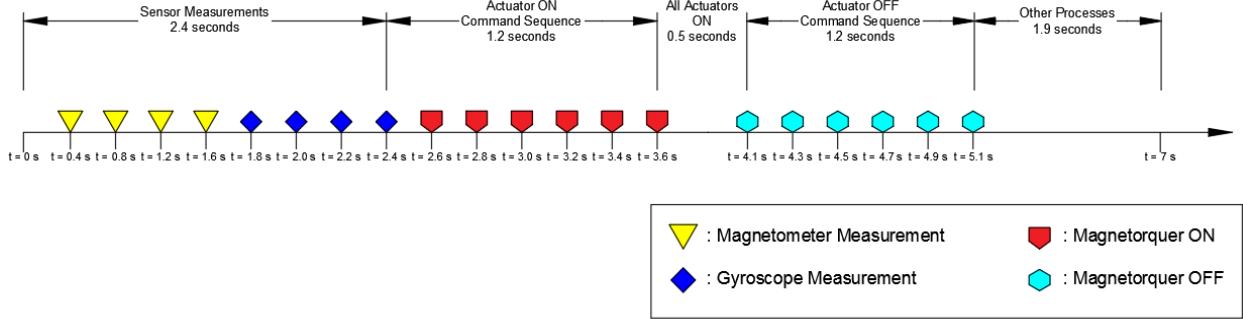
**Figure 18:** Data interface diagram between the ADCS flight computer emulation and CLASS dynamic simulation.

CAPSat's ADCS sensors include four 3-axis magnetometers and four 3-axis gyroscopes. The actuators used to control the satellite's attitude are magnetorquers. To emulate a sensor measurement, the magnetic field value and the satellite angular velocity are computed by the

CLASS dynamic simulation and sent to the ADCS algorithm when it requests a measurement. Sensor noise is added using Gaussian Zero-Mean white noise. Sensor measurements are made when the magnetorquers are off so that they do not interfere with the magnetometer readings. 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. The serial bus on CAPSat has only one serial port for all ADCS sensor and command traffic. As a result, sensor measurements are picked up one at a time, resulting in a total sensor measurement time of 2.4 seconds. The emulator is programmed to only pick up measurements from the CLASS simulation that are greater than the actual sensor resolutions.

The ADCS actuators include six (i.e., two on each satellite face) single axis magnetorquers. They are commanded by a single serial port. The estimated time for each magnetorquer to receive and apply the commanded input current is 0.2 seconds. Consequently, the total time for all magnetorquers to be on 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 previous measurements and actuation cycles are repeated about every seven seconds in the ADCS flight computer. Figure 19 illustrates the chronological sequence of tasks being executed by the ADCS on every computation cycle. The emulator executes these tasks in real-time.



**Figure 19:** Chronological sequence of events in one ADCS execution cycle. The total cycle duration is seven seconds and is repeated as long as C&DH is commanding ADCS to determine the satellite's attitude and control it.

## 5.2 ADCS ALGORITHM

CAPSat is programmed to de-tumble after deployment from the International Space Station and to then maintain nadir pointing for communications with the ground. Six magnetorquers (i.e., two on each body axis) with a maximum magnetic moment of  $0.18106 \text{ A} \cdot \text{m}^2$  are used to control CAPSat's attitude. The de-tumble control law used is the *B-dot* control law which uses the time derivative of the measured magnetic field vector multiplied by the gain K:

$$\mathbf{u}^b = -K \frac{d\mathbf{B}^b}{dt} \quad (118)$$

In discrete steps, it is equal to the difference between the current measured magnetic field vector and the previously measured magnetic field vector divided by the time difference between both measurements:

$$\mathbf{u}^b = -K \frac{\mathbf{B}_{t=\tau}^b - \mathbf{B}_{t=\tau-\Delta\tau}^b}{\Delta\tau} \quad (119)$$

The control command  $\mathbf{u}^b$  is then converted to the appropriate magnetorquer magnetic moment command and scaled down using the hardware properties such as maximum moment and resolution.

After a successful de-tumble (i.e., angular rates are reduced to values near zero), the ADCS is then commanded to begin pointing the satellite communication antenna along the nadir

direction. To determine the satellite attitude, a seven state Extended Kalman Filter (EKF) has been implemented. Using an on-board orbit propagator and attitude integrator, the EKF is able to determine the quaternion between the body and inertial frame and filter out inaccuracies in the angular velocity measurements using the magnetometer and gyroscope raw data. Therefore, the dynamic simulation only provides the emulator with the angular velocity and the magnetic field value.

The ADCS algorithm also computes the inertial direction of the Local Vertical/Local Horizontal (LVLH) frame discussed in Chapter 3. By determining the instantaneous attitude of the LVLH frame, the algorithm can determine the quaternion that describes the rotation from the body frame to the LVLH frame. The *z-axis* of the body frame happens to be aligned with the antenna. So, to point the antenna nadir, the body frame *z-axis* must be aligned with the LVLH *z-axis*. One possible way of achieving this, is by setting the quaternion describing the rotation from the body frame to the LVLH frame to be  $[\pm 1 \ 0 \ 0 \ 0]^T$  (i.e., aligning the two coordinate systems). The control law used is a state feedback control law (or PD controller):

$$\mathbf{u}^b = -k_p \mathbf{q}_b^{LVLH}(2:4) - k_d \boldsymbol{\omega}_{b/r}^b$$

Where:

$k_p$  = proportional gain

$\mathbf{q}_b^{LVLH}(2:4)$  = vector containing the last three elements (the non-scalar) of the quaternion describing the rotation between the body frame and the LVLH frame

$k_d$  = derivative gain

$\boldsymbol{\omega}_{b/r}^b$  = angular velocity of the body coordinate frame relative to the LVLH coordinate frame expressed in the body frame

## 5.3 RESULTS

When deployed from the P-POD, the satellite has no attitude information and might be spinning at high rates. For system verification tests, the initial conditions for the satellite dynamic simulation were set to relatively high spin rates (on the order of 10 deg/s) about all three axes. The initial conditions for the following results were:

$$\boldsymbol{\omega}^b = [6.3 \quad -5.5 \quad -6.1]^T \text{ deg/s}$$

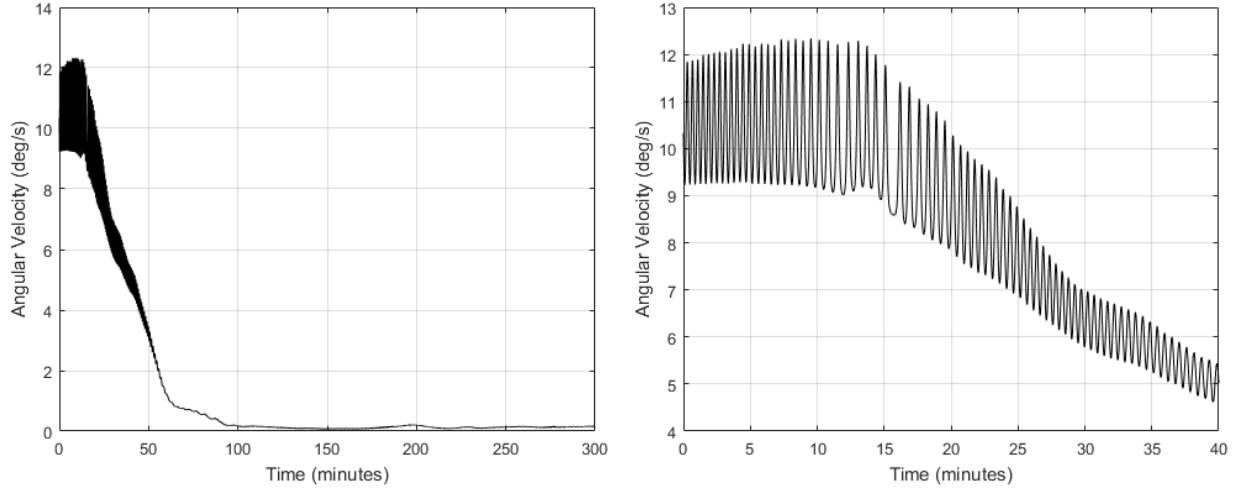
$$|\boldsymbol{\omega}^b| = 10.35 \text{ deg/s}$$

$$\mathbf{q}_b^I = [0.24 \quad 0.189 \quad 0.917 \quad -0.258]^T$$

The orbit propagator was initialized at an altitude of 400 km because CAPSat is expected to be deployed from the ISS. Representative aerodynamic and gravity gradient disturbance torques were also initialized in the simulation.

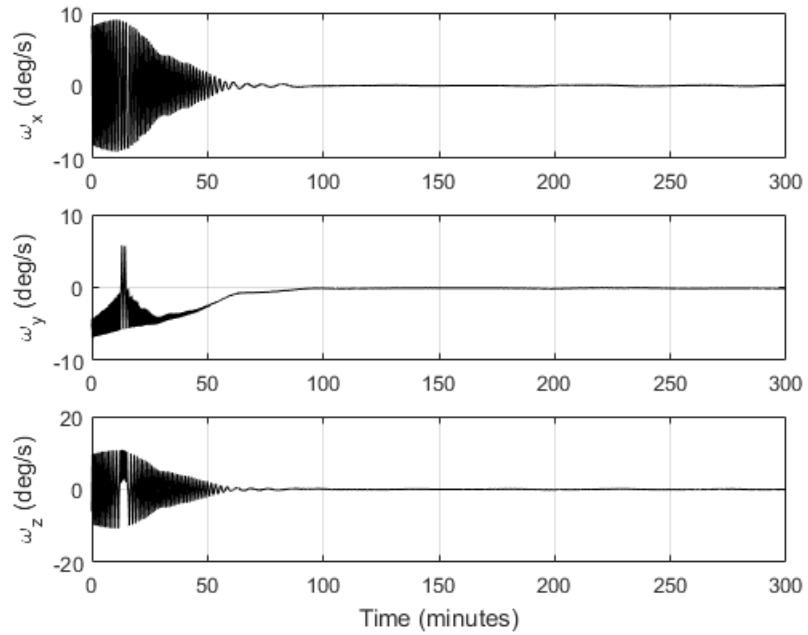
### 5.3.1 De-tumbling

After the simulation was initialized and run, the magnitude of the satellite angular velocity decreased from 10.35 deg/s to 0.15 deg/s in 105.8 minutes (i.e., after one orbit). As indicated in Figure 20, the angular velocity experienced oscillations while decreasing to zero. This is mainly due to the hardware constraints discussed in Section 5.1. That includes the application of the control torque every seven seconds, the gradual turn on and off of the actuators, and the under-actuated nature of magnetorquers. Otherwise, the de-tumbling algorithm is effective at de-spinning the satellite in an acceptable amount of time.



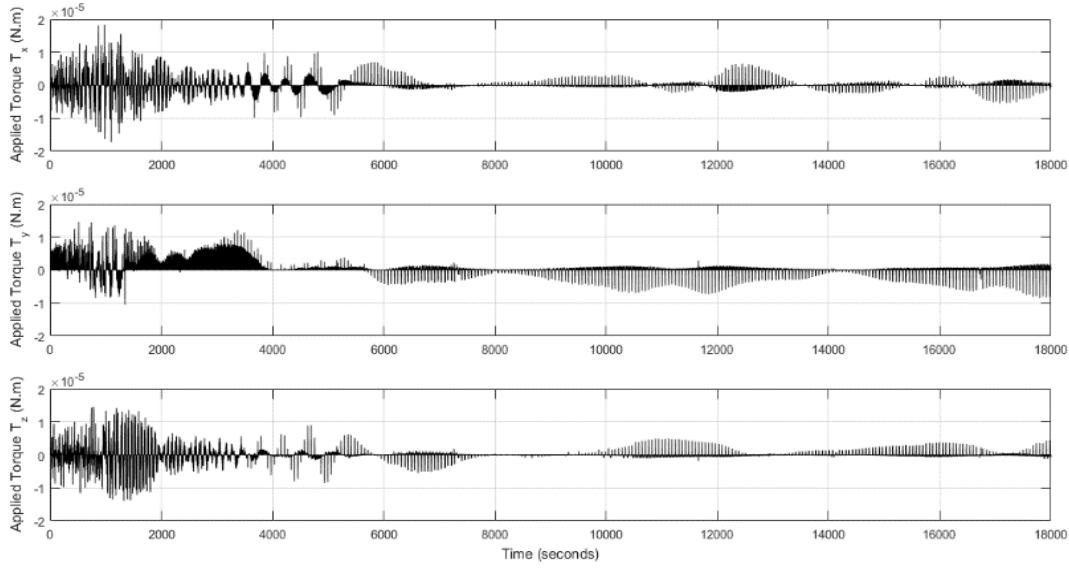
**Figure 20:** Closed-loop tests on CAPSat’s ADCS algorithm for de-tumbling. The magnitude of the angular velocity (deg/s) vs time (minutes) (left) and a close-up of the plot (right). After approximately one orbit, the satellite has an angular velocity near zero. The right-hand-side plot demonstrates the dense oscillations that occur because of the delays in applying torque due to hardware constraints.

Figure 21 illustrates the Cartesian components of the angular velocity expressed in the body frame vs time. Oscillations are clearly present but eventually dampen to zero after one complete orbit. The under-damped nature of the system is clearly demonstrated in Figure 21. In some cases, these oscillations might be undesirable. Reaction wheels provide a more stable response because they do not rely on the magnetic field vector or any other instantaneous environmental factor. However, CAPSat’s data communication hardware constraints, mainly the single serial port issue, is the greatest cause for this delayed and under-damped response.



**Figure 21:** Closed-loop tests on CAPSat's ADCS algorithm for de-tumbling. The angular velocity (deg/s) expressed in the body frame vs time (minutes). After approximately one orbit, the satellite stops spinning in all three axes.

Figure 22 illustrates the applied torque on the satellite vs time. The vector elements of the torque evidently have opposite signs compared to the angular velocity. Figure 23 offers a closer look on the behavior of the applied torque with time.

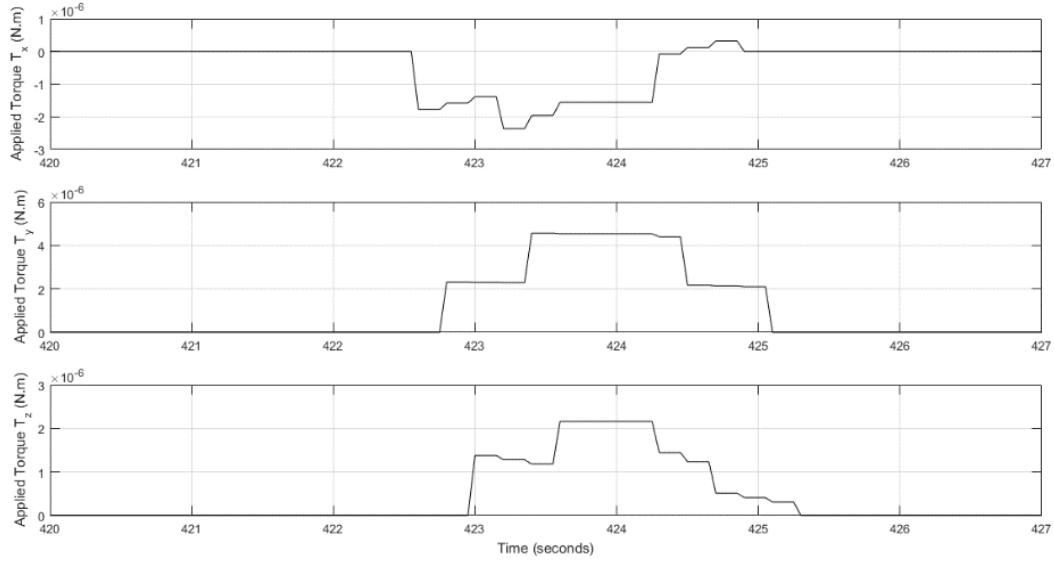


**Figure 22:** Closed-loop tests on CAPSat’s ADCS algorithm for de-tumbling. The vector components of the applied torque (N.m) expressed in the body frame vs time (minutes). Torque is not immediately applied to the satellite because of hardware constraints during each control cycle and it is almost never equal to the torque computed by the ADCS algorithm.

Figure 23 illustrates the applied torque for an entire control cycle of seven seconds, arbitrarily selected at time  $420 \text{ sec} < t < 427 \text{ sec}$ . 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 switched on in the following sequence: positive  $x$ -axis, positive  $y$ -axis, positive  $z$ -axis, negative  $x$ -axis, negative  $y$ -axis, and negative  $z$ -axis. They are then switched off in the same sequence. This explains why the torque seems to double after 0.6 seconds, maintain a magnitude for 0.5 seconds, and go back to zero in steps (the  $y$ -component of the torque in Figure 23 portrays this behavior the clearest). As for the small steps (the small “wrinkles” that are most apparent in the  $x$ -component), they are caused by a physical property of magnetorquers. The torque applied by a magnetorquer is equal to:

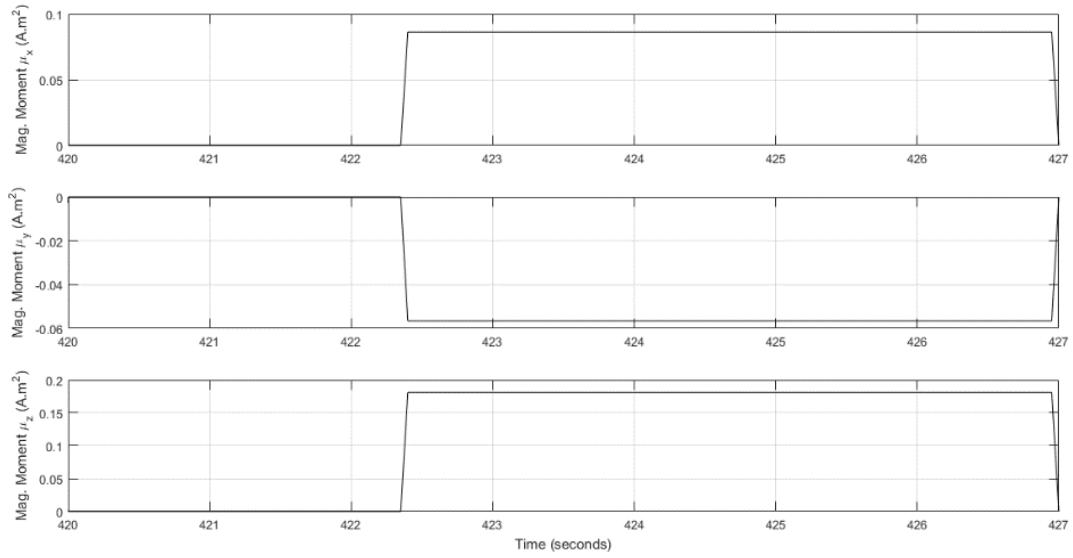
$$\mathbf{T}^b = \boldsymbol{\mu}^b \times \mathbf{B}^b \quad (120)$$

Where  $\mu^b$  is the magnetic moment and  $\mathbf{B}^b$  is the magnetic field vector expressed in the body frame of the satellite. Since, the magnetic vector of the satellite is constantly changing when the satellite is rotating, then for the same magnetic moment command from the ADCS, for a single cycle, the applied torque will not be constant and will have these slight variations and “bumps”.



**Figure 23:** Closed-loop tests on CAPSat’s ADCS algorithm for de-tumbling. The vector components of the applied torque (N.m) expressed in the body frame vs time (minutes). The plots are a close-up of an arbitrary control cycle. The step-like nature of the torque being applied is due to the hardware constraints of the CAPSat ADCS system and the dependence of magnetorquer-induced torque on the instantaneous magnetic field.

Figure 24 is a plot of the commanded magnetic moment vector for the same control cycle as the one in Figure 23. 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 23, and consequently the behavior of the angular velocity in Figure 20, would not have appeared in the results.



**Figure 24:** Closed-loop tests on CAPSat’s ADCS algorithm for de-tumbling. The vector components of the commanded magnetic moment ( $\text{A} \cdot \text{m}^2$ ) expressed in the body frame vs time. The plots are a close-up of the same arbitrary control cycle as Figure 23. The magnetic moment is the control command calculated by the ADCS software. Disregarding hardware and physical constraints would lead to an inaccurate representation of the actual torque applied.

CLASS proved to be instrumental in identifying several software and hardware errors in the de-tumbling algorithm 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.

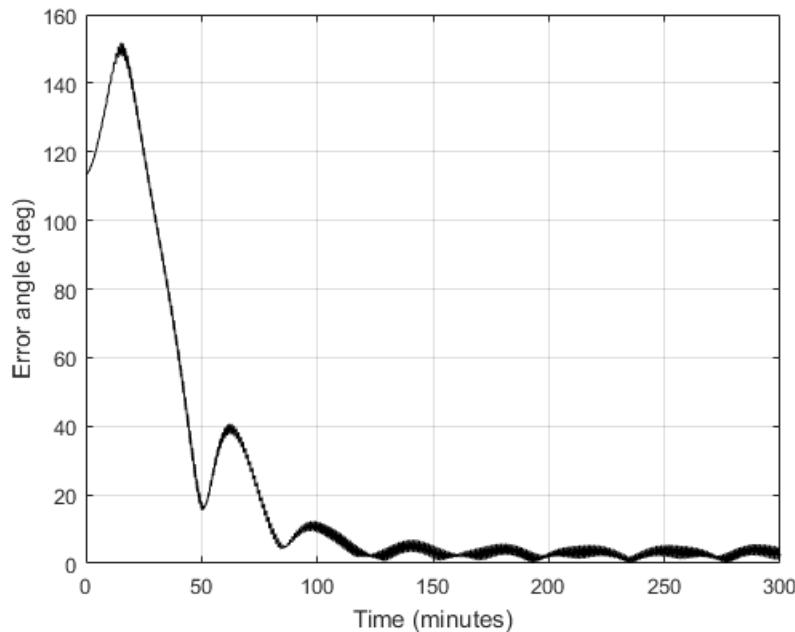
### 5.3.2 Earth-Pointing

After the satellite reaches a near-zero spin rate, the control system starts to command the satellite to point and maintain the antenna aligned with the nadir vector. An arbitrary attitude is reached in the simulation after de-tumbling:

$$\mathbf{q}_b^I = [0.427 \quad 0.468 \quad 0.137 \quad 0.762]^T$$

This corresponds to an angle of 113 degrees between CAPSat's antenna and the nadir vector. The controller is only activated after the satellite has de-tumbled so, the initial conditions for the angular velocity can be assumed to be a value near zero.

Figure 25 plots the error angle between the antenna vector and the nadir vector vs time. The angle decreases from 113 degrees to 0 degrees  $\pm 6$  degrees in 142.2 minutes (i.e., about an orbit and a half). The pointing accuracy is satisfactory according to CAPSat's antenna specifications.

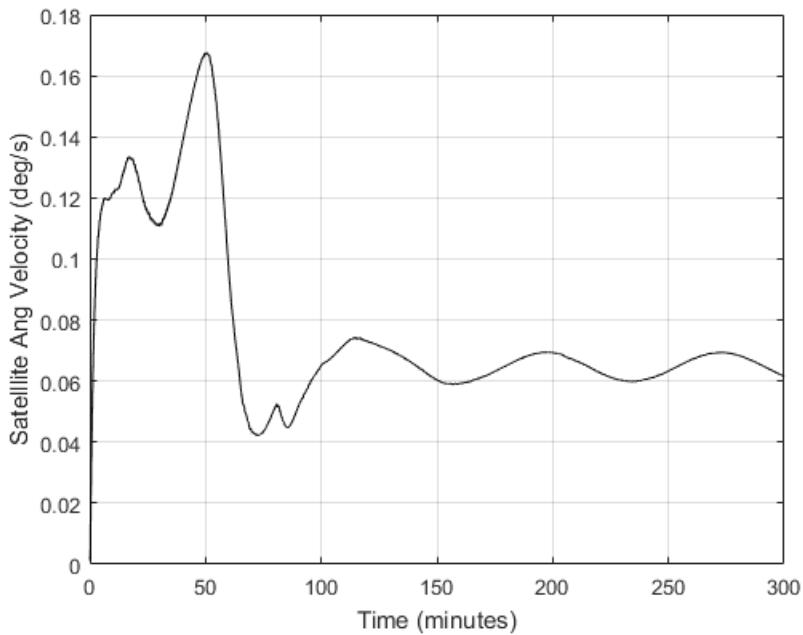


**Figure 25:** Closed-loop tests on CAPSat's ADCS algorithm for Earth-pointing. The error angle (degrees) between the antenna vector and nadir vector vs time (minutes). In about one orbit, the ADCS control algorithm is able to align CAPSat's antenna vector with the nadir vector. After the initial alignment, the controller also proves to be effective at maintaining the error angle at values  $\pm 6$  degrees.

For a satellite in LEO with a small eccentricity, the orbit is almost circular. Hence, the LVLH frame (or the nadir vector) is rotating inertially at about the same angular velocity as the orbit angular velocity (or period):

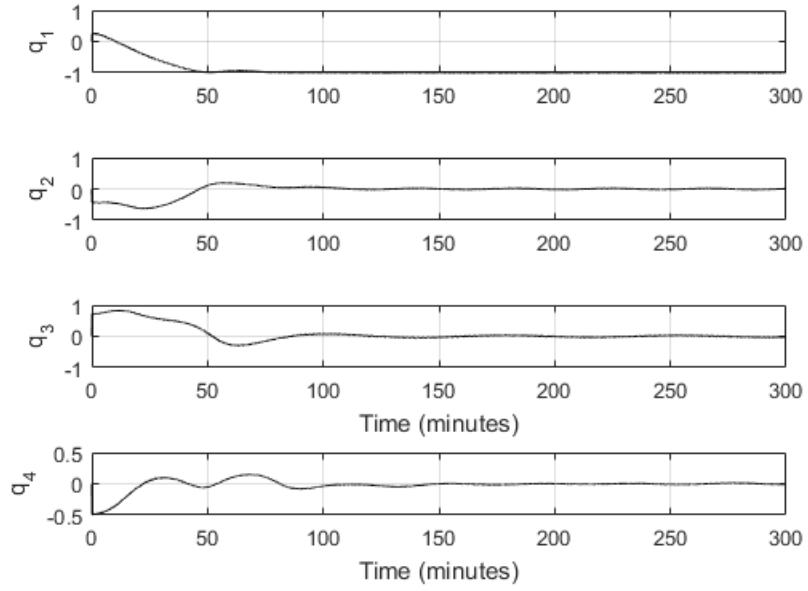
$$\omega_{\text{circ}} = \frac{360^\circ}{T_{\text{CAPSat}}} = \frac{360^\circ}{92.83} = 3.88 \frac{\text{deg}}{\text{min}} = 0.065 \frac{\text{deg}}{\text{sec}} \quad (121)$$

To maintain the alignment of the antenna vector and the nadir vector, the satellite must maintain an angular velocity equal to the LVLH frame angular velocity in Equation 121. As illustrated in Figure 26, the satellite angular velocity does eventually settle to oscillating values around the 0.065 deg/sec mark. The oscillations are  $\pm 0.005$  (7.7%) in magnitude. It is extremely difficult to strictly maintain a steady state value equal to 0.065 deg/sec due to the aerodynamic and gravity gradient torques that constantly disturb the satellite attitude, given the slow response of the magnetorquers.



**Figure 26:** Closed-loop tests on CAPSat's ADCS algorithm for Earth-pointing. The magnitude of the satellite angular velocity (deg/s) vs time (minutes). The velocity settles down to a value that almost matches the circular orbit period of the satellite, indicating a successful tidal lock.

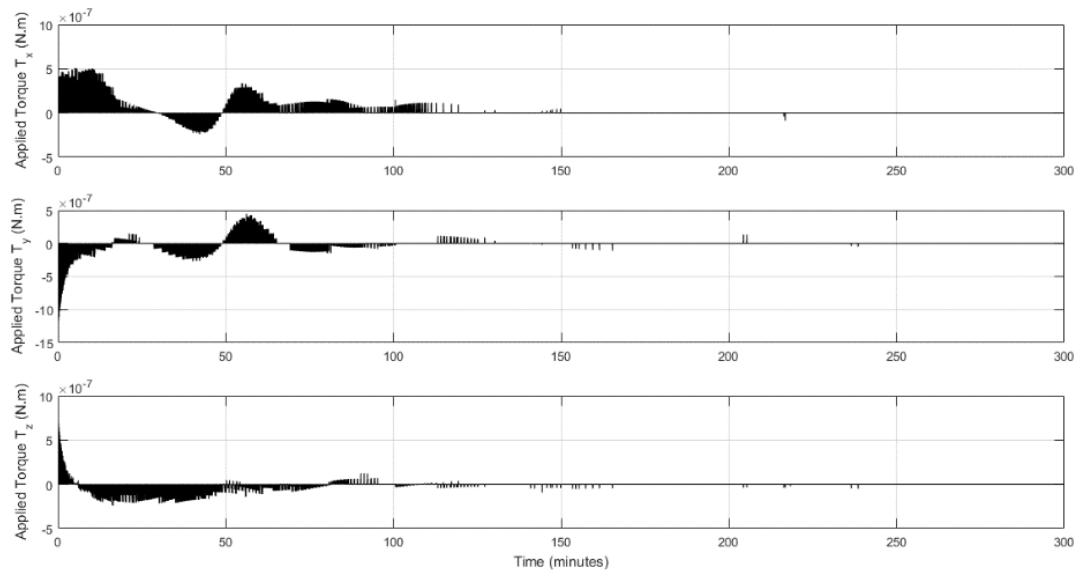
Figure 27 illustrates the quaternion vector representing rotation from the body frame to the LVLH frame vs time. To align the two body frames, the quaternion must go to  $[\pm 1 \ 0 \ 0 \ 0]^T$ . It achieves the desired steady state with a variation of  $\pm 3\%$  in 142.2 minutes.



**Figure 27:** Closed-loop tests on CAPSat’s ADCS algorithm for Earth-pointing. The quaternion vector representing the rotation from the body frame to the LVLH frame vs time (minutes). The quaternion vector reaches a steady state value of  $[-1 \ 0 \ 0 \ 0] \pm 3\%$  in 142.2 minutes. This means the satellite body frame and LVLH frame are aligned.

Figure 28 illustrates the applied torque on the satellite in the body frame vs time. As is the case with the de-tumbling phase, the torque is applied in steps and is also limited by the hardware constraints discussed previously.

CLASS has provided LASSI with an easily configurable and accurate test setup for verifying the performance of the ADCS algorithms developed for CAPSat. The test system greatly simplified the design of the state feedback controller and the tuning of control gains. Future LASSI spacecraft will similarly benefit from this enhanced capability.



**Figure 28:** Closed-loop tests on CAPSat's ADCS algorithm for Earth-pointing. The vector components of the applied torque (N.m) expressed in the body frame vs time (minutes). Just like the results for the de-tumbling test, the hardware properties of the ADCS have a major role in the dynamics and torque application.

## CHAPTER 6: CONCLUSION

The Laboratory for Advanced Space Systems at Illinois (LASSI) established requirements for a modular satellite test system called: Closed Loop Analysis of Space Systems (CLASS). CLASS provides CubeSat developers at the University of Illinois with an easily configurable and reliable test system for verification and validation. Avoidable mission failures caused by lack of testing remain significantly high in the CubeSat industry, mainly due to the difficulty in accessing sophisticated and costly space dynamics and environment emulators. 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. Its modularity cuts the time spent customizing a simulation or test set-up for a specific subsystem. As is usually the case in the integration flow of a CubeSat mission, the availability of the actual attitude sensors for testing purposes might not be feasible during an intermediate stage. CLASS offers the option of implementing a customizable sensor emulator developed on Arduino boards so that hardware constraints remain a present factor in the closed-loop tests to maximize result reliability.

CLASS played an instrumental role in promptly identifying critical errors in the previous version of the attitude determination and control system software for CAPSat, a LASSI satellite. It has also become the primary development tool for the design of the control algorithms and flight software of the current and future LASSI missions.

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. Furthermore, LASSI can take advantage of the lab's Helmholtz cage, using CLASS to dynamically simulate the Earth's magnetic field for a satellite's magnetometers.

The initial development phase of CLASS is now complete. In the future, CLASS will be enhanced as part of an on-going project within LASSI with contributions from graduate and undergraduate engineering students. The true benefit of CLASS will then be realized with successful CubeSat mission operations made possible by high fidelity hardware-in-the-loop testing.

## REFERENCES

- [1] J. Puig-Suari, C. Turner and W. Ahlgreen, "Development of the Standard CubeSat Deployer and a CubeSat Class PicoSatellite," in *IEEE Aerospace Conference*, Big Sky, 2001.
- [2] I. Nason, J. Puig-Suari and R. Twiggs, "Development of a Family of Picosatellite Deployers Based on the CubeSat Standard," in *IEEE Aerospace Conference*, Big Sky, 2002.
- [3] 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.
- [4] E. Kulu, "CubeSat," Nanosats Database, 2019. [Online]. Available: <https://www.nanosats.eu/cubesat>. [Accessed 2019].
- [5] W. A. Shiroma, L. K. Martin, J. M. Akagi, J. T. Akagi, W. L. Byron, B. A. Fewell and A. T. Ohta, "CubeSats: A Bright Future for Nanosatellites," *Central European Journal of Engineering*, vol. 1, no. 1, pp. 9-15, 2011.
- [6] 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," in *International Astronautical Congress*, Guadalajara, 2016.
- [7] S. A. Jacklin, "Small-Satellite Mission Failure Rates," NASA STI Program NASA Ames Research Center, Moffet Field, 2019.
- [8] M. Swartwout, "A Statistical Survey of Rideshares (and Attack of the CubeSats, Part Deux)," in *IEEE Aerospace Conference*, Big Sky, 2012.
- [9] E. Kulu, "Nanosats Database," October 2019. [Online]. Available: <https://www.nanosats.eu/>. [Accessed December 2019].
- [10] 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.
- [11] A. Alanazi and J. Straub, "Statistical Analysis of CubeSat Mission Failure," in *AIAA/USU Conference on Small Satellites*, Logan, 2018.
- [12] M. Langer and J. Bouwmeester, "Reliability of CubeSats - Statistical Data, Developers' Beliefs and the Way Forward," in *AIAA/USU Conference on Small Satellites*, Logan, 2016.
- [13] P. Fortescue, G. Swinerd and J. Stark, *Spacecraft Systems Engineering*, West Sussex: John Wiley & Sons, 2011.
- [14] G. Sebestyen, S. Fujikawa, N. Galassi and A. Chuchra, *Low Earth Orbit Satellite Design*, Cham: Springer, 2018.
- [15] D. E. Krutz and M. Lutz, "Bug of the Day: Reinforcing the importance of testing," in *IEEE Frontiers in Education Conference (FIE)*, Oklahoma City, 2013.
- [16] A. Engel, *Verification, Validation, and Testing of Engineered Systems*, Hoboken: John Wiley & Sons, 2010.
- [17] Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board, "A Compilation of IEEE Standard Computer 610," The Institute of Electrical and Electronics Engineers (IEEE), New York City, 1990.

- [18] C. Pecheur and S. Nelson, "Survey of NASA V&V Processes/Methods," NASA Ames Research Center, Moffett Field, 2002.
- [19] National Aeronautics and Space Administration, "NASA Systems Engineering Handbook," NASA Headquarters, Washington DC, 2007.
- [20] R. N. Sorge, "Space Power Facility - Capabilities for Space Environmental Testing With a Single Facility," in *Space Simulation Conference*, Annapolis, 2013.
- [21] European Space Agency, "Test Centre," [Online]. Available: [https://www.esa.int/Enabling\\_Support/Space\\_Engineering\\_Technology/Test\\_centre](https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Test_centre). [Accessed 2020].
- [22] M. F. Lembeck and N. D. Pignato, "Galileo Attitude and Articulation Control Subsystem Closed Loop Testing," in *AIAA Computers in Aerospace Conference*, Hartford, 1983.
- [23] 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.
- [24] 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.
- [25] E. Takuji and E. G. Lightsey, "A closed-loop hardware simulation of decentralized satellite formation control," *Advances in the Astronautical Sciences*, vol. 113, p. 13, 2003.
- [26] B. Bingham and C. Weston, "System Level Hardware-in-the-Loop Testing for CubeSats," in *Advances in Astronautical Sciences Guidance and Control Conference*, Breckenridge, 2014.
- [27] Engineering IT at the University of Illinois, "LABORATORY FOR ADVANCED SPACE SYSTEMS AT ILLINOIS (LASSI)," 2019. [Online]. Available: <https://aerospace.illinois.edu/research/research-facilities/laboratory-advanced-space-systems-illinois-lassi>. [Accessed January 2020].
- [28] The CubeSat Program, "CubeSat Design Specification Rev 13," California Polytechnic State University.
- [29] D. Gross, W. Hauger, J. Schroder, W. A. Wall and S. Govindjee, *Engineering Mechanics*, Berlin: Springer, 2011.
- [30] F. L. Markley and J. L. Crassidis, *Fundamentals of Spacecraft Attitude Determination and Control*, New York : Springer, 2014.
- [31] W. Hu, *Fundamental Spacecraft Dynamics and Control*, Singapore: John Wiley & Sons, 2015.
- [32] G. R. Hintz, *Orbital Mechanic and Astrodynamics: Techniques and Tools for Space Missions*, Cham: Springer, 2015.
- [33] E. Fresk and G. Nikolakopoulos, "Full Quaternion Based Attitude Control for a Quadrotor," in *European Control Conference*, Zurich, 2013.
- [34] J. Diebel, "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors," Stanford University, 2006.
- [35] NASA Mission Planning and Analysis Division, "Euler Angles, Quaternions, and Transformation Matrices," Lydon B. Johnson Space Center, Houston, 1977.
- [36] D. Vallado, *Fundamentals of Astrodynamics and Applications*, El Segundo: Space Technology Library, 2001.

- [37] B. Neta, "Partial List of Orbit Propagators," Naval Postgraduate School, Monterey.
- [38] C. G. Hilton and J. R. Kuhlman, "Mathematical Models for the Space Defense Center," PhilcoFord Publication, 1966.
- [39] Y. Kozai, "Mean Values of Cosine Functions in Elliptic Motion," *Astronomical Journal*, vol. 67, p. 311, 1962.
- [40] F. R. Hoots and T. Kelso, "Models for Propagation of NORAD Element Sets," Spacetrack, 1980.
- [41] D. Vallado and P. Crawford, "SGP4 Orbit Determination," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Honolulu, 2008.
- [42] 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.
- [43] M. H. Lane, P. M. Fitzpatrick and J. J. Murphy, "On the Representation of Air Density in Satellite Deceleration Equations by Power Functions with Integral Exponents," Armed Services Technical Information Agency , Arlington, 1962.
- [44] E. Thebault, C. C. Finlay and T. Zvereva, "International Geomagnetic Reference Field: the 12th generation," *Earth, Planet and Space*, vol. 67, 2015.
- [45] C. M. Roithmayr, "Contributions of Spherical Harmonics to Magnetic and Gravitational Fields," NASA Langley Research Center, Hampton, 2004.
- [46] R. A. Langel and R. H. Estes, "A Geomagnetic Field Spectrum," *Geophysical Research Letters*, vol. 9, no. 4, pp. 250-253, 1982.
- [47] NGDC NOAA, "International Geomagnetic Reference Field," December 2019. [Online]. Available: <https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html>. [Accessed 2020].
- [48] J. R. Wertz, *Spacecraft Attitude Determination and Control*, Dordrecht: Kluwer Academic Publishers, 1978.
- [49] D. A. Vallado and D. Finkleman, "A Critical Assessment of Satellite Drag and Atmospheric Density Modeling," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Honolulu, 2008.
- [50] L. F. Markley and J. L. Crassidis, *Fundamentals of Spacecraft Attitude Determination and Control*, New York: Springer, 2014.
- [51] D. L. Oltrogge and K. Leveque, "An Evaluation of CubeSat Orbital Decay," in *AIAA/USU Conference on Small Satellites*, Logan, 2011.
- [52] D. A. Vallado and D. Finkleman, "A Critical Assessment of Satellite Drag and Atmosphericing Density Modeling," *Acta Astronautica*, vol. 95, pp. 141-165, 2014.
- [53] NASA, "Spacecraft Aerodynamic Torques," NASA Space Vehicle Design Criteria (Guidance and Control), 1971.
- [54] The Linux Foundation, "HOWTO setup Linux with PREEMPT\_RT properly," 20 06 2017. [Online]. Available: [https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/preemptrt\\_setup](https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/preemptrt_setup). [Accessed 2020].
- [55] K. C. Wang, *Embedded and Real-Time Operating Systems*, Cham: Springer, 2017.

- [56] The Linux Foundation, "HOWTO build a simple RT application," 2017. [Online]. Available: [https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application\\_base](https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base). [Accessed 2020].
- [57] IEEE, "POSIX Ada Language Interfaces," IEEE Standard for Information Technology , 1996.
- [58] Celestrak, "ISS - Celestrak," 2020. [Online]. Available: <https://www.celestrak.com/NORAD/elements/stations.txt>. [Accessed April 2020].
- [59] NASA Johnson Space Center, "Reference Guide to the International Space Station," National Aeronautics and Space Administration, Houston, 2015.
- [60] C. Peat, "ISS-Orbit," Heavens Above, 2020. [Online]. Available: <https://heavens-above.com/orbit.aspx?satid=25544>. [Accessed April 2020].
- [61] J. E. Prussing and B. A. Conway, Orbital Mechanics, New York City: Oxford University Press, 2013.
- [62] P. Manikowski and M. A. Weiss, "The Satellite Insurance Market and Underwriting Cycles," *The Geneva Risk and Insurance Review*, vol. 38, no. 2, pp. 148-182, 2013.
- [63] H. Helvajian and S. W. Janson, Small Satellites: Past, Present, and Future, El Segundo: The Aerospace Press, 2008.
- [64] K. Fukuda, Y. Sakamoto, T. Kuwahara, K. Yoshida and Y. Takahashi, "Static Closed Loop Test System for Attitude Control System of Micro Satellite RISING-2," in *IEEE/SICE International Symposium on System Integration*, Kyoto, 2011.