CubeSat Attitude Control using Micronewton Electrospray Thruster Actuation

by

Mark David Van de Loo

S.B., Aerospace Engineering with Information Technology, Massachusetts Institute of Technology (2013)

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author ....

Department of Aeronautics and Astronautics
May 9, 2014

Signature redacted

Certified by ....

Sara Seager
Class of 1941 Professor of Physics and Planetary Science
Thesis Supervisor

Signature redacted

Accepted by ....

Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

CubeSat Attitude Control using Micronewton Electrospray Thruster Actuation

by

Mark David Van de Loo

Submitted to the Department of Aeronautics and Astronautics
on May 9, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Micronewton electrospray thrusters are a promising new actuator for CubeSat attitude control. Electrospray thrusters have advantages over current state of the art CubeSat attitude actuators in mass, volume, and their ability to produce translational acceleration in addition to control moments. An attitude determination and control system was designed for a 1U CubeSat assuming commercial-off-the-shelf attitude determination hardware components and six electrospray thrusters developed by the MIT Space Propulsion Laboratory. A high fidelity spacecraft dynamics simulation was constructed for analysis of the performance of the ADCS system. Attitude determination was tested with an engineering model of a 1U CubeSat, and the entire ADCS system was tested in simulation. Results of these preliminary tests show the use of electrospray thrusters as attitude actuators to be feasible, although significant work remains to complete a flight-ready ADCS system.

Thesis Supervisor: Sara Seager
Title: Class of 1941 Professor of Physics and Planetary Science

Academic Advisor: Paulo C. Lozano
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

There are many people I want to thank who have made the completion of this thesis possible. To Professor Sara Seager who has inspired me to dream and given me the opportunity to work on intriguing problems. To Professor Paulo Lozano for his invaluable advice and support throughout my time at MIT. To my TSat teammates Mary Knapp, Akshata Krishnamurthy, and Fernando Mier Hicks who made TSat engineering model testing a reality. To the Nexterra foundation for their generous sponsorship of the TSat project. To Alessandra Babuscia, Chris Pong, Ian Sugel, Josh Joseph, Justin Smith, Lars Blackmore, and Carolyn Major for their mentorship and life advice. To Marie Stuppard and Beth Marois who are two of the most helpful people I know. To my roommate Mitch Westwood for his constant encouragement and motivation. To Professor Sheila Widnall for her wisdom and overwhelming support, and to Bill Widnall for his inspiration and many great days of sailing in Marblehead. To Maggie McConnell who always helps me to see what is important in life. And finally, to my dad Dave, my mom Sue, my sisters Kate and Meg, and my brother Luke who have made me who I am.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 CubeSats: A Developing Platform for Space Exploration

1.1.1 The CubeSat Standard

A CubeSat is a type of satellite that meets design specifications developed by California Polytechnic State University in San Louis Obispo and the Space Systems Development Lab at Stanford University [18]. The CubeSat standard was developed with the intent of making space more accessible to small payloads. This is accomplished both by allowing faster development of small satellite missions through the creation of standardized commercial-off-the-shelf (COTS) components, and by streamlining the launch process as a secondary payload on a large launch vehicle [26]. The CubeSat specification requires that a "1U" CubeSat have external dimensions of 10 cm x 10 cm x 10 cm, and a maximum mass of 1.33 kg. A computer drawing of a 1U CubeSat is shown in Figure 1-1.

Two or three "1U" volumes may be fixed together to form a "2U" or "3U" CubeSat [26]. In the near future, a revised CubeSat specification will also include "6U" and possibly even "12U" options [25]. CubeSats are commonly deployed from a spring-loaded "P-POD" (PicoSatellite Orbital Deployer) attached to a launch vehicle aft of its primary payload, or

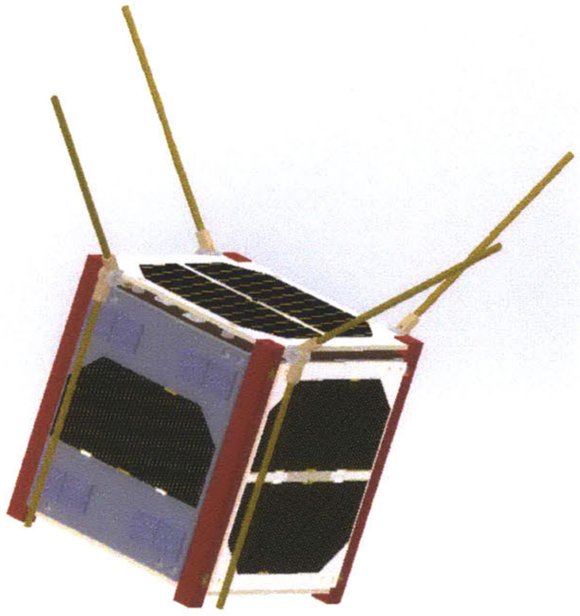

Figure 1-1: 1U CubeSat with deployed antennas. (credit: Mary Knapp)

from a JAXA deployer attached to the Japanese robotic arm of the International Space Station [26]. The P-POD deployer is shown in figure 1-2.

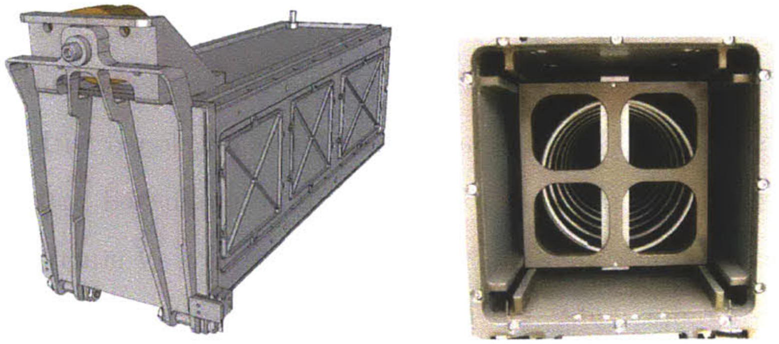

Figure 1-2: P-POD CubeSat Deployer [26].

1.1.2 Current CubeSat Mission Limitations

Over the last decade and a half, many academic institutions have developed CubeSats as educational tools and technology demonstration platforms. The low mission cost, usually less than $200,000 makes flight experience attainable in educational situations where it would not be with larger spacecraft. In addition, large organizations including NASA, The Aerospace Corporation, and Boeing have begun to explore the potential of CubeSats as a platform for science and commercial technology [3]. As of August 2013, 53 CubeSats were on orbit and actively tracked by the North American Aerospace Defense Command (NORAD), and more than 66 additional CubeSats were awaiting a launch opportunity as reported through the NASA CubeSat Launch Initiative [25].

As CubeSat platforms have come under consideration for Earth observation, atmospheric science, and astronomy, fundamental limitations due to weaknesses in the areas of CubeSat attitude control and translational ability have been recognized. Attitude control refers to control of the orientation and rotational rates of a satellite about its center of mass, while translation refers to maneuvers involving an acceleration of the satellite's center of mass. Nanosatellite propulsion technology is largely unproven, and there are no commercially available flight-demonstrated propulsion solutions available for CubeSats [3]. Without a proven and readily available propulsion system, CubeSat missions requiring formation flight or precision constellations are not possible. Possible objectives of interplanetary and asteroid science missions are limited by the absence of translational control, and the inability to compensate for atmospheric drag forces places limits on the minimum operating altitude and the mission lifetime of CubeSats in low Earth orbit. Though nanosatellite attitude control systems are more advanced than their translational control counterparts, a majority of nanosatellites are still designed without active attitude control [3]. Many are passively controlled using magnetorquers that interact with the geomagnetic field, and some have no attitude control at all. Reaction wheels that are compatible with the CubeSat form factor have been developed and tested, but only a few CubeSat missions have used them in flight.

Both attitude and translation control hardware have yet to become miniaturized to the point where achieving both attitude and translational control in a 1U satellite is reasonable. One state of the art design for a CubeSat guidance navigation and control module developed by the University of Texas at Austin offers translational and attitude capability in the volume of 1U, but leaves no room for payload unless a 2U or 3U structure is used [6]. Other designs that offer either attitude control or thrust capability (but not both) have been miniaturized to approximately one half of 1U each [21],[2]. A design requiring translational control must also have a means of controlling the spacecraft attitude since the thrust vector must be pointed in the direction opposite the desired motion. It can thus be reasonably concluded that designs using current state of the art hardware must have a size of at least 2U to support a mission that requires any translational control.

1.1.3 Motivation for Microthruster Actuation in CubeSat Attitude Control

Microthruster actuation presents a solution for supporting a mission that requires translational control in a 1U form factor. Since translational control requires attitude control as explained at the end of Section 1.1.2, all proposed designs for CubeSat translational control systems as of August 2013 contained reaction wheels for attitude actuation. Microthrusters may be used as both translational and attitude actuators, effectively eliminating the need for reaction wheels. The absence of reaction wheels in a microthruster-actuated attitude control scheme eliminates a volume and mass of approximately half of 1U. Attitude control about all rotational axes can be provided with a set of 6 microthrusters as presented in Section 2.2. The total mass and volume of required hardware, fuel, and electronics for a single microthruster are approximately 55 g and 0.042 liters respectively. Thus, microthruster actuation requires approximately 30% of 1U to provide full attitude and translational control. This leaves 0.7 liters and 880 g in a 1U form factor for other spacecraft subsystems and payload. In the case of a 2U or 3U design, the payload capacity is increased by 0.7 liters and 880 g from what it could be without microthruster actuation.

In addition to providing advantages in the area of payload capacity, microthrusters allow more precise and efficient translational control. Microthrusters can be mounted pointing in several different directions on a CubeSat structure instead of the single direction allowed by larger chemical and cold gas thrusters. Mounting thrusters facing several directions allows a CubeSat to translate without many of the slew maneuvers that are necessary when using a single large thruster for stationkeeping. The elimination of slew maneuvers enables stationkeeping that is both more efficient and more precise. Such stationkeeping could allow CubeSat constellations, formation flying, or even tethered missions that are not possible when a single thruster is used for translational control.

A final advantage of microthrusters over other propulsion solutions is that they can be designed without pressure vessels or volatile substances [16]. Since the majority of CubeSat missions are launched as cargo or secondary payloads, pressure vessels and volatile substances included in a design pose a risk not only to the mission success of the CubeSat but also to the mission success of the primary payload. Since primary payloads are often several orders of magnitude more expensive than secondary payloads, primary payload stakeholders are reluctant to allow CubeSats with pressure vessels or volatile substances to be launched on their launch vehicles. The absence of pressure vessels and the involatility of the fuel in some microthruster designs eliminates many of the safety concerns that primary payload stakeholders have with chemical propulsion systems.

1.1.4 Proposed CubeSat Missions Requiring Microthruster Actuation

Many CubeSat missions that require microthruster actuation have been proposed in published literature, and many more exist that have not been thoroughly explored. The goal of this section is to provide a few examples of CubeSat missions that require microthruster actuation in order to help motivate the need for development of microthruster actuated attitude control.

As mentioned earlier, CubeSats are in the midst of a transition from a status as primarily educational tools, to a common standardized sensor platform for commercial and government missions. There exist a plethora of important Earth observing missions that could be effectively accomplished by CubeSat constellations, as summarized by [21]. Proposed missions range from atmospheric sounding to disaster monitoring, and from surveillance to the gathering of data on the state of snow, ice, and oceans. The data gathered by these CubeSats might have implications in understanding weather and storm patterns, monitoring the health of terrestrial ecosystems, or ensuring national security.

One of the main advantages of CubeSats over larger monolithic satellites for missions such as those described above is the ability to send many satellites into orbit within a reasonable budget, as opposed to just one or two. The satellites in the constellation can be spaced out around one orbital plane, or in several different planes and at various altitudes. Constellations greatly increase the temporal resolution of collected data, which is very useful or critical in many cases [10]. In order to maintain a constellation, each of the satellites must be equipped with precision stationkeeping ability. As discussed in section 1.1.3, microthrusters seem to be the most attractive solution for providing the combination of translational and attitude actuation that is needed.

In addition to CubeSat constellations, another category of proposed CubeSat missions consists of those requring formation flight. Formation flying spacecraft fly in close proximity to one another, operating almost as a single larger spacecraft. Such a configuration could allow for modular system construction, with different satellites performing different functions and communicating among one another. For example, the communications module could fly separately from the camera and data processing modules. This would allow great flexibility in mission objectives and could potentially have great advantages in the area of mission robustness. Alternatively, the satellites in the formation might all be involved in performing the same task, such as forming the nodes of an array antenna or telescope. As just one example, a concept for a formation flying solar observatory is currently being developed by Knapp et al. [20]. Again, precision attitude control and stationkeeping are required.

Finally, CubeSat-sized spacecraft might serve as extremely useful tools in micro-gravity asteroid exploration. A team of small probes might be able to gather data over a much larger area than a single larger spacecraft. Microthruster actuation would be essential for operating in the delicate microgravity environment. If the body being explored was small enough that driving was not possible, microthrusters might even be used for moving from one data collection site to another.

1.2 Electrospray: A Microthruster Technology

1.2.1 Principles of Electrospray Thrusters

An electrospray thruster is a type of microthruster that produces thrust by emitting charged particles that have been accelerated through an electrostatic field. These charged particles are extracted from an ionic liquid that has been carried into a microfabricated array of porous tips by capillary action. The electrostatic field is created by placing an extractor grid above the porous tips and generating an electrical potential difference between the grid and the ionic liquid. The grid is aligned such that there is an opening directly above each tip, through which the ions from that tip are emitted. A schematic drawing of an Electrospray Thruster assembly is given in Figure 1-3.

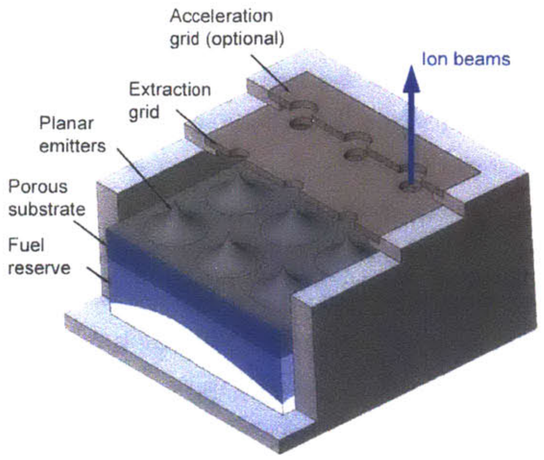

Figure 1-3: Drawing of an electrospray thruster assembly [4].

Common metrics used to assess the capabilities of any space propulsion system include thrust force and specific impulse. Electric propulsion systems are also typically assessed using an electrical power efficiency metric [4]. The thrust force, F, produced by a thruster is given by equation 1.1. Here, $\bar{c}$ is the average exhuast velocity of emitted particles, and $\frac{dm}{dt}$

is the propellant mass flow rate.

$$F = \bar{c}\frac{dm}{dt} \tag{1.1}$$

Propellant economy of a thruster is represented by the thruster's specific impulse or I_{sp}, given by Equation 1.2. Here, g is the acceleration due to the Earth's gravity at sea level and the other symbols represent the same quantaties as in Equation 1.1.

$$I_{sp} = \frac{F}{g\frac{dm}{dt}} = \frac{\bar{c}}{g} \tag{1.2}$$

The electrical power efficiency, η, is given by Equation 1.3, and represents the effectiveness of the thruster in converting consumed electrical power into thrust. In Equation 1.3, P represents the input power to the thrusters, or the product of the input current and voltage supplied by the thruster electronics to the thruster.

$$\eta = \frac{\frac{1}{2}\bar{c}^2\frac{dm}{dt}}{P} \tag{1.3}$$

In comparison to chemical propulsion systems, electrospray thrusters produce an extremely low thrust, on the order of μN. The low thrust levels are a result of the small mass (on the order of 100 amu) of the ions that are emitted. One advantage of using electrospray thrusters is that their low thrust levels can be used to achieve much finer translation and rotation control than the higher thrust levels of their chemical propulsion counterparts. Electrospray thrusters also have an advantage over other propulsion systems in propellant economy. The I_{sp} of electrospray thrusters is on the order of 2500s in comparision to the 500 s - 600 s specific impulse of the most efficient chemical rockets.

The greatest advantages of electrospray thrusters as applied to CubeSats are their low mass and volume, as well as the fact that they do not require compressed or volatile substances that could pose a risk to other satellites sharing the same launch vehicle. Table 1.1 gives a summary of the positive and negative characteristics of micropropulsion technologies available for small spacecraft.

Propulsion Technology	Mono-propellant	Cold gas	Bipropellant	Pulsed plasma thruster	Micro ion engines	Electrospray thruster
I_{sp} (s)	220	$\approx$65	$\approx$300	$\approx$270	2500-5480	2500-3500
Fuel	Hydrazine	N_2	LOX/hydrocarbon	Teflon	Argon/ Xenon	EMI-BF4
Moving parts		Yes	Yes	No	No	No
High temperature	Yes	No	Yes	Yes	Yes	No
Flammable fuel	Yes	No	Yes	No	No	No
Pressure vessel	Yes	Yes	Yes	No	Yes	No
High voltage	No	No	No	Yes	Yes	Yes

Table 1.1: Summary of characteristics of propulsion technologies available for small spacecraft compiled from [13]. The main advantages of electrospray thrusters include high efficiency and the absence of pressure vessels and volatile propellants. The main disadvantage of electrospray thrusters is the high driving voltage required for operation. (credit: MIT Seager Group INVEST proposal)

1.2.2 Electrospray Thrusters as CubeSat Actuators

Over the last several years, large strides have been made in the development of Electrospray Thrusters packaged as CubeSat actuators. The ion Electrospray Propulsion System (iEPS) being developed by the MIT Space Propulsion Laboratory consists of thruster modules of the dimension 12 mm x 12 mm x 2.5 mm, as well as supporting thruster electronics. Each thruster module contains a porous emitter array with approximately 600 porous tips [11]. Two thruster modules are shown in figure 1-4, and a scanning electron microscope image of a porous emitter array is shown in Figure 1-5. In the iEPS design, thruster modules are positioned in pairs. The two modules of each pair are operated with opposite polarity, and thus the pair emits an equal number of positive and negative ions. This is done so that the thruster exhaust will have a net neutral charge.

Eight pairs of thrusters, along with thruster drive electronics fit in 1/3 of a 1U cubesat [11], equivalent to approximately 300 cm^3 and 0.4 kg. Each 12 mm x 24 mm thruster pair

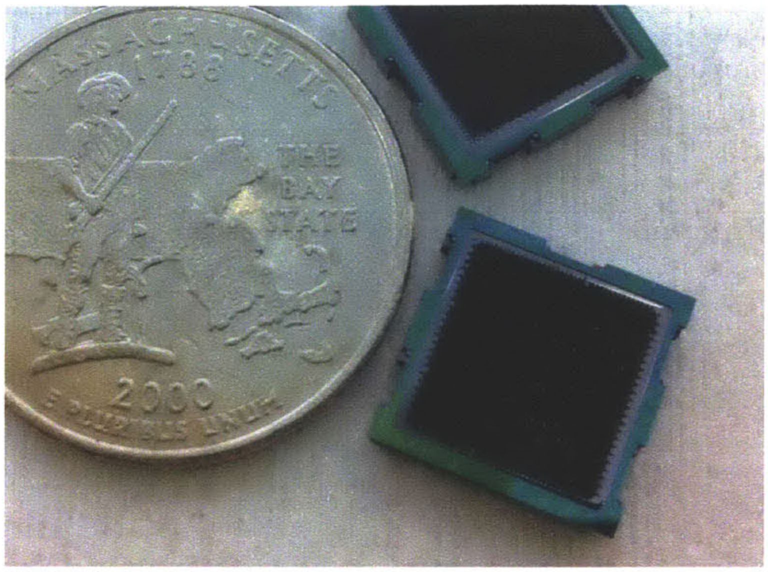

Figure 1-4: Two assembled iEPS thruster modules [11].

produces approximately 20 μN of thrust at full throttle. The thrusters are throttleable by varying the voltage applied between the ionic liquid and the extractor grid.

The current iEPS design documented in [11] shows one possible positioning of the thrusters, but they could be repositioned as needed to produce the desired control torques and forces for a given vehicle. For a 1U CubeSat, a single one of the eight thruster pairs is capable of producing a torque about the spacecraft center of mass of about 1μN-m. This seems small, but with the small mass and inertia of a 1U CubeSat, 1μN-m will produce an angular acceleration of approximately 0.12 rad/s^2, or almost 7 deg/s^2. Thus, though Electrospray Thrusters in their current form would likely not be effective as primary attitude actuators of large spacecraft, they do present a promising case for use on CubeSat-sized platforms.

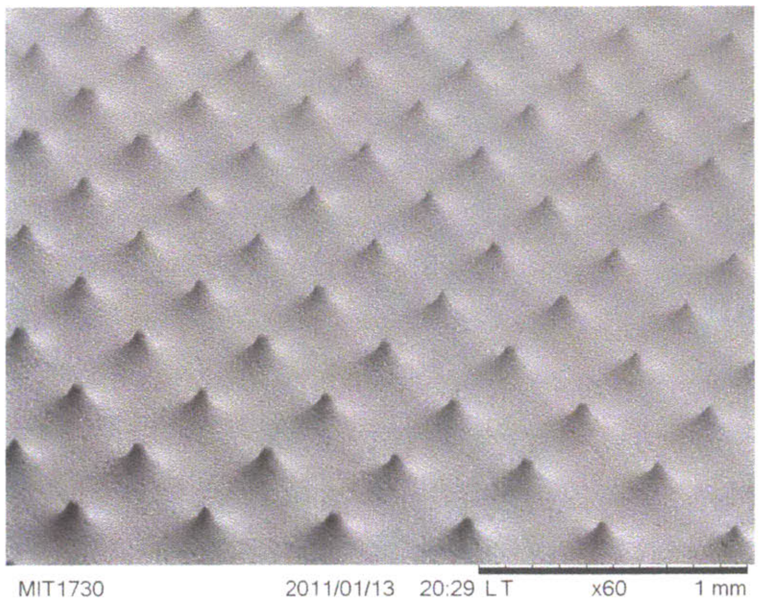

Figure 1-5: Scanning electron microscope image of a porous emitter array [4].

1.2.3 Comparison to Current State of the Art CubeSat Attitude Actuators

State of the art CubeSat attitude actuators include reaction wheels and magnetorquers. Reaction wheels are a set of flywheels spun up by electric motors that store some of the angular momentum of the spacecraft and can thus adjust its attitude. Magnetorquers consist of coils of wire that generate magnetic dipoles when current is passed through them. These dipoles interact with the Earth's magnetic field to produce a moment on the spacecraft. The Maryland Aerospace MAI-201 [12] reaction wheel assembly and the magnetorquers embedded in the NanoPower solar panels manufactured by GomSpace [5] are compared with the iEPS electrospray thruster actuation system in Table 1.2. The MAI-201 and a NanoPower solar panel are shown in Figures 1-6 and 1-7.

Reaction wheels are capable of producing more torque than electrospray thrusters, however they have disadvantages in the areas of mass and volume. As discussed in Section 1.1.2, improvement in these areas is key in increasing the variety of missions available to CubeSats. Further miniaturization of reaction wheel assemblies is difficult, due to the large number of

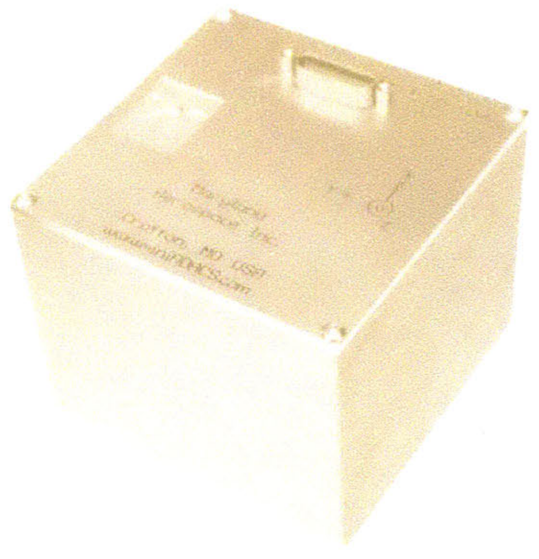

Figure 1-6: MAI-201 CubeSat reaction wheel assembly [12].

moving parts and electromechanical systems involoved. Additionally, since reaction wheels operate on the principle of momentum storage, it is possible for them to become saturated and unable to produce control torque. For this reason, systems that use reaction wheels as actuators must also have a means of removing momentum from the wheels. Magnetorquers or thrusters are thus required in addition to reaction wheels for most applications. Since magnetorquers cannot be used in deep space, due to the absence of a geomagnetic field, the tendency of reaction wheels to saturate presents a particular disadvantage when designing

	Reaction Wheels (MAI-201)	Magnetorquers (GomSpace)	Electrospray Thrusters (iEPS)
Mass (kg)	0.73	0.03	0.44
Volume (l)	0.44	0.01	0.30
Max Torque (Nm)	0.005	0 to 3.25e-6	2.0e-6
Power Consumption (W)	2.4	0.09	5.0
Lifetime Limitations	mechanical wear	none	fuel consumption

Table 1.2: Comparison of CubeSat attitude actuators.

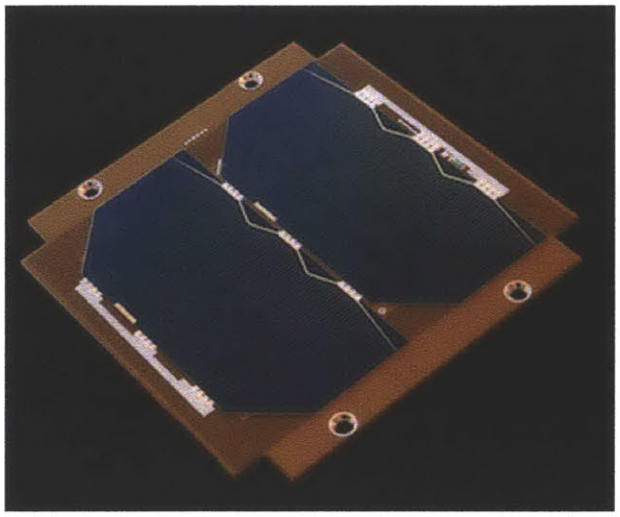
Figure 1-7: NanoPower solar panel [5].

interplanetary or asteroid exploration missions.

Magnetorquers are an attractive solution in the areas of mass, volume, and power consumption. Their main disadvantage is the irregularity in the amount of torque they can produce. Since magnetorquers operate by producing a dipole that tends to align with the geomagnetic field, the torque available is completely dependent on the direction and strength of the geomagnetic field at the magnetorquer's position in space. The torque produced by the interaction between the dipole and the magnetic field can be represented as the cross product of the dipole vector with the external magnetic field vector. Thus, magnetorquers can never produce moments on the spacecraft about the direction of the geomagnetic field. This is a particular problem for satellites in low inclination orbits, since the direction of the geomagnetic field changes very little along the path of the orbit. Though high inclination orbits do allow for torques to be applied about any axis, the set of available torques varies over the course of an orbit, so attitude adjustments may take a long time since they can only be made at a specific points along the orbit. A final drawback of magnetorquers is that they are not useful for deep space exploration missions.

The main disadvantage of electrospray thrusters is their finite lifetime due to the exhaustion of propellant. Though electrospray thrusters also have a lower maximum torque and a higher power consumption than state of the art actuators, they avoid the pitfalls of saturation and inconsistency in the magnitudes and directions of available control torques due to reliance on the geomagnetic field. Electrospray thrusters present advantages over reaction wheels in the areas of mass and volume, opening up more space for payloads and effectively making it possible to fit a mission that requires precision pointing in 1U. Finally, electrospray thrusters have the added benefit of being able to produce delta-V and thus to act as translational control actuators. Where reaction wheels would require a separate propulsion module on the order of 1U in size to be able to perform translation, electrospray thrusters require no additional equipment. The only increase in mass and volume from the attitude actuation system would be the amount of fuel needed to perform whatever translational maneuvers were required.

1.3 TSat: A Demonstration Mission for Electrospray Thrusters

ThrusterSat (TSat) is a 1U CubeSat demonstration mission for iEPS electrospray thruster technology being designed by the Seager Group in the Department of Earth, Atmospheric, and Planetary Sciences at MIT, in collaboration with the MIT Space Propulsion Laboratory. The goals of the TSat mission are to provide data that will be useful to future users of the iEPS thrusters, and to create a technological foundation for future formation flying CubeSat science missions.

1.3.1 Mission Overview

The TSat spacecraft is a 1U CubeSat, designed primarily using commercial-off-the-shelf CubeSat components. Either six or eight pairs of iEPS electrospray thrusters will be positioned along the edges of the cube in a configuration that is capable of producing control torques about all axes. The spacecraft will be transported to the International Space Station by a cargo re-supply vehicle, and will be deployed via the NanoRacks CubeSat deployer attached to the Japanese robotic arm. Upon exiting the ISS keep-out sphere, TSat will perform a series of tests to characterize the performance of the iEPS electrospray thrusters in the space environment. Once thruster performance is well characterized, additional tests will demonstrate the capabilities of the thrusters in attitude control and the production of delta-V for orbit changes. Data will be downlinked to a ground station at MIT.

1.3.2 Spacecraft Subsystems

Much of the TSat spacecraft is made up of commercial components. Power is provided by six body-mounted solar panels manufactured by GomSpace. The panels each include a 1-axis MEMS gyro, and a photodiode Sun sensor for attitude determination. The panels also include embedded magnetorquers that may be used as attitude actuators if the thrusters are not operating. The "NanoMind" flight computer and the "NanoPower" power supply are also purchased from GomSpace. The structure is a custom aluminum unibody design manufactured at MIT, and the communications subsystem consists of a Radiometrix BHX transciever and GomSpace deployable antennas. A GPS receiver made by Surrey will provide navigation data to the satellite. The iEPS thrusters are mounted on a PCB that also contains the high-voltage thruster drive electronics.

1.3.3 Preflight Testing

A magnetic levitation balance was constructed in the MIT Space Propulsion Lab to help characterize the performance of the iEPS thrusters once integrated with the TSat structure [16]. The balance consists of an electromagnet and two laser alignment sensors suspended from an aluminum frame. A permanent magnet is attached to the CubeSat structure. The electromagnet imparts a force on the permanent magnets, suspending it without physical contact to any external supports. The laser sensors measure the vertical displacement, which is used as feedback to a controller that adjusts the magnetic field of the electromagnet so that the CubeSat will levitate in a stable manner. The ability of the CubeSat to "float" without contacting any external supports allows the structure to rotate without friction. This is essential because friction could be a significant source of error since the thrust levels and moments being measured are so low. A CAD drawing of the balance is shown in figure 1-8. More technical details of the magnetic levitation balance are given in Section 5.1.1. The first objective of testing in this levitation balance is to prove that the iEPS package can be effectively integrated with a CubeSat structure. Interactions of the thrusters with each other and with the CubeSat structure, as well as their overall performance characteristics will be measured.

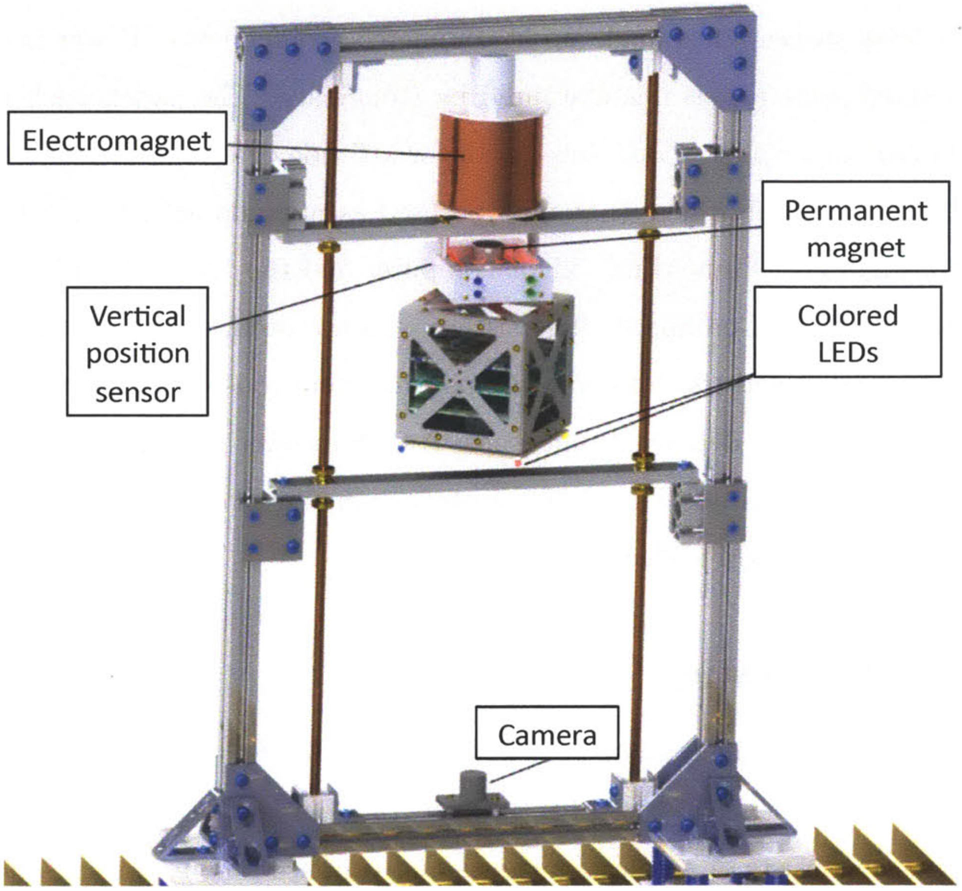

Figure 1-8: CAD drawing of the SPL magnetic levitation balance. (credit: Fernando Mier Hicks).

A second objective of the levitation balance is to test and verify the performance of attitude determination and control algorithms. The balance only allows rotation about one axis, so only single-axis control will be tested. The tests will use the same three-axis attitude estimator and controller that will be used on orbit, with outputs for the two unused axes set to zero. Demonstrating single axis control using electrospray thrusters as actuators is an important step toward the demonstration of full three-axis attitude control in space. Preparation for this levitation balance attitude determination and control test represents much of the work of this thesis.

1.3.4 Electrospray Thruster Attitude Control

As mentioned in the previous sections, TSat will demonstrate attitude control using electrospray thrusters as actuators. By addressing a fundamental limitation of current CubeSat technology, this demonstration will expand the variety of missions accessible to the CubeSat platform.

The following chapters of this thesis document the design and analysis of the TSat attitude determination and control system. Topics addressed include system hardware design, system software design, simulation and analysis, and engineering model testing. The goal of this thesis is to provide a foundation upon which microthruster attitude determination and control systems for future CubeSats may be built.

Chapter 2

System Hardware

The TSat attitude determination and control system is made up of both standard commercial-off-the-shelf (COTS) and custom hardware components. Attitude determination hardware consists strictly of COTS parts, since the focus of this research is on control actuation. Suppliers of the components include GomSpace, a Danish company specializing in Cube-Sat hardware and software, and Surrey of the United Kingdom specializing in small satellite technology. Attitude control hardware consists entirely of custom manufactured components still in the final stages of research and development. The components are manufactured by the MIT Space Propulsion Laboratory, and by Espace, a small company specializing in miniaturized high voltage electronics for space applications.

2.1 Attitude Determination Sensors

The attitude determination sensors consist of a single 3-axis magnetometer, six photodiode Sun sensors (one on each face of the spacecraft), six 1-axis MEMS gyros (primary and redundant in three orthogonal axes), and a GPS receiver. The Sun sensors and magnetometers are used to measure the orientation or attitude of the spacecraft, while the gyros measure angular rates of rotation of the spacecraft, referred to as spacecraft body rates. The GPS receiver supplies information about the spacecraft's position in space to the attitude determination software. This information must be known in order to effectively use the measurments from the magnetometer and Sun sensors. A detailed description of how the data from the sensors and magnetometer is used by attitude determination software to generate estimates of the spacecraft attitude and body rates is given in Chapter 3.

2.1.1 3-Axis Magnetometer

A 3-axis magnetometer manufactured by Honeywell is built into the GomSpace NanoMind flight computer.

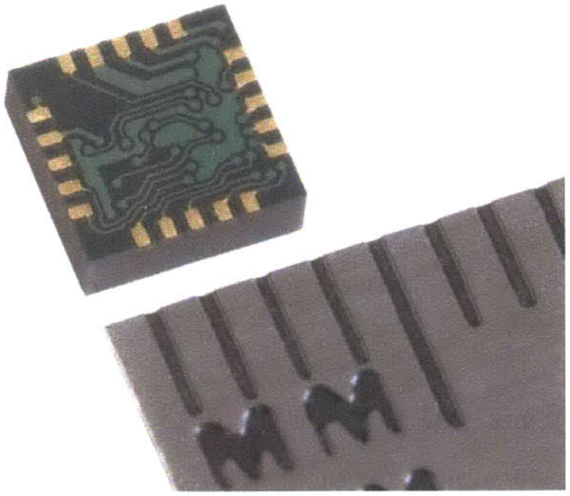

Figure 2-1: Honeywell HMC5843 magnetometer (credit: Honeywell).

The HMC5843 magnetometer, shown in Figure 2-1 measures the magnitude and directon of the Earth's magnetic field in the range from 1 nanoTesla to 0.4 milliTesla. For reference, the magnitude of the geomagnetic field expected to be experienced by TSat is between 20 and 60 microTesla. Figure 2-2 shows a plot of the magnitude of the magnetic field experienced by TSat over one orbital period as simulated by the TSat GNC simulation described in Chapter 4. The simulation was run based on a sample orbit with parameters similar to the orbital parameters of the International Space station.

The magnetometer uses Honeywell's Anisotropic Magnetoresistive technology, which provides measurements with a resolution of 7 milliGauss, or 0.7 microTesla, and a signal to noise

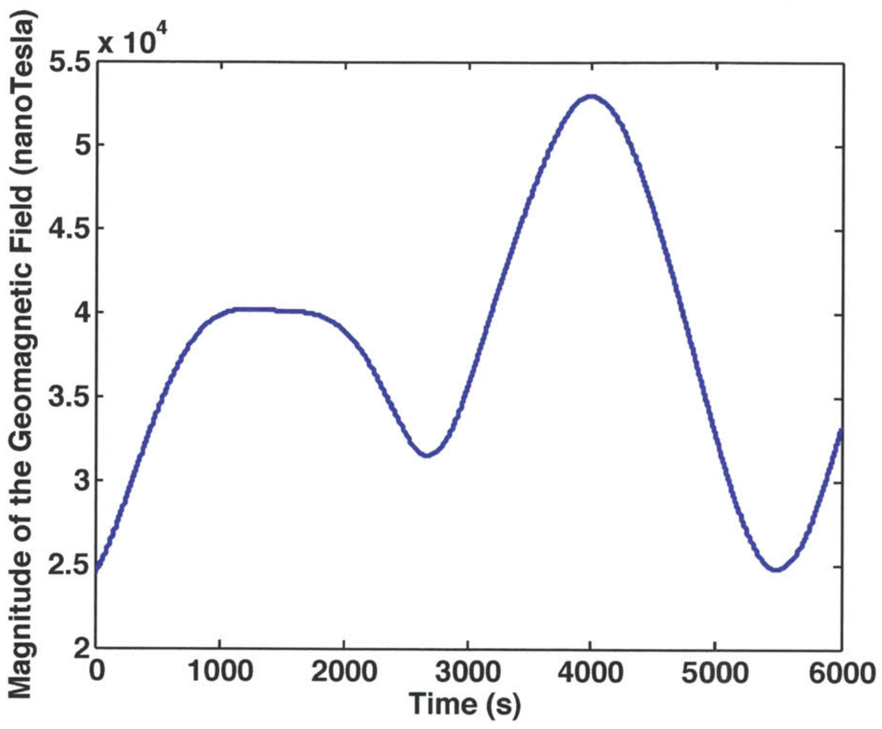

Figure 2-2: Magnitude of the magnetic field experienced by TSat over one orbital period as calculated by the TSat GNC simulation. The simulated orbit is similar to the orbit of the International Space Station.

ratio of 70 dB. Measurements are sent to the flight computer via an I^2C interface at a rate of 100 Hz. Figure 2-3 shows a sample magnetometer output during a lab test where the TSat engineering model was subjected to the ambient geomagnetic field in the lab and rotated through 180 degrees about the Spacecraft Body Frame (SBF) z-axis. A detailed description of SBF and all other coordinate systems used is given in Chapter 3.

2.1.2 Photodiode Sun Sensors

A planar photodiode is embedded in the center of the GomSpace solar panels mounted on each face of the Tsat structure. Manufactured by Silonex, the SLCD-61N8 photodiode produces a voltage proportional to the cosine of the angle between the direction of the Sun and the outward normal to the solar panel. This angle β is computed according to equation 2.1, where V is the output voltage of the photodiode in milliVolts and C is a constant that

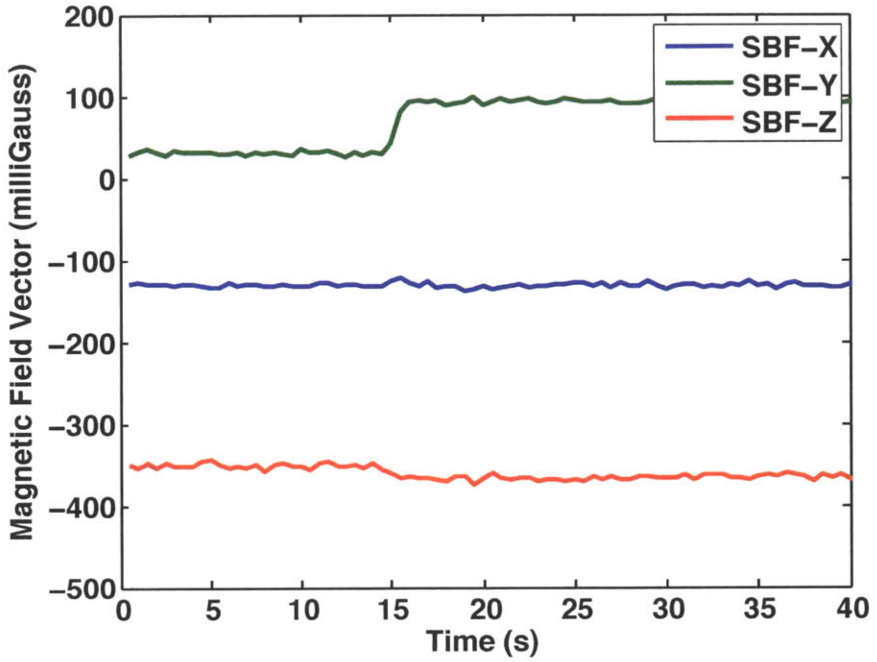

Figure 2-3: Sample magnetometer output when subjected to the geomagnetic field in the lab and rotated through 180 degrees.

depends on the intensity of the light source. For the Sun, C is assumed to be the open circuit voltage of the photodiode given by [24] of 400 mV. This constant will be measured before flight. For the light source used for engineering model testing in the lab, the value of C was determined to be $C = 42.0$. Since the data of all of the Sun sensors is processed using the same value of C to produce a vector measurement of the position of the Sun, and that measurement is subsequently normalized, the value of C used does not impact attitude determination.

$$\beta = \cos^{-1}\left(\frac{V}{C}\right) \tag{2.1}$$

The measured angles from all of the Sun sensors are combined by the attitude determination software, as described in Section 3.3, to produce a coarse measurement of the direction of the Sun. This Sun vector measurement is combined with measurements from the magnetometer to compute a measured spacecraft attitude quaternion.

Figure 2-4 shows a sample output of two Sun sensors. In this test, the TSat engineering model was rotated such that two adjacent faces were pointed toward a light source, one after the other. A noisy bias was observed in Sun sensor readings when the sensors were not pointed toward the light source. The bias is attributed to the presence of ambient light in the lab, and reflection of the light source by the lab bench and components of the TSat engineering model structure. A photograph of the test setup is shown in Figure 2-5.

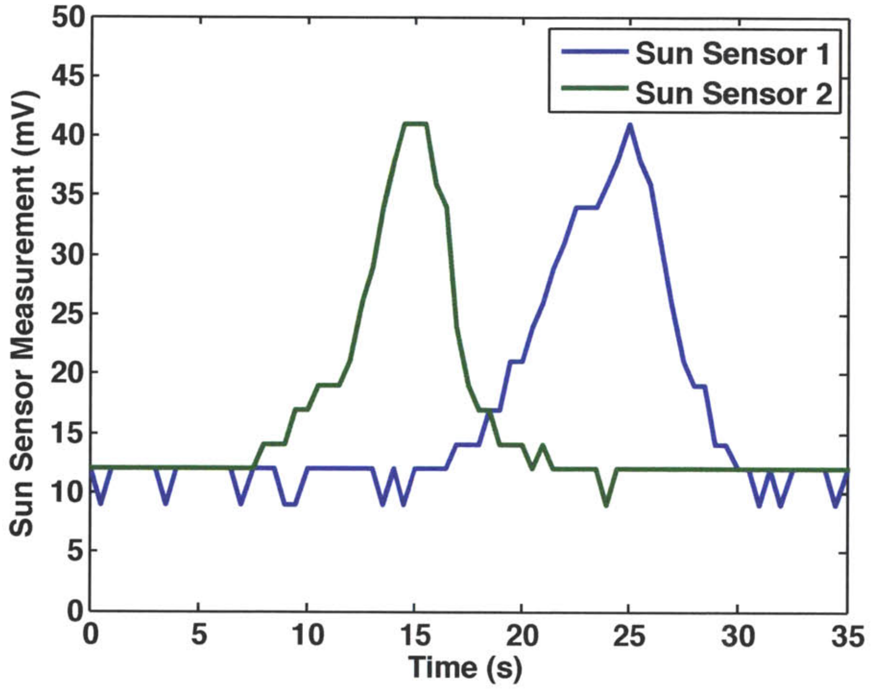

Figure 2-4: Sample output of two photodiode Sun sensors when rotated past a light source. The bias value of approximately 12 mV is attributed to ambient light in the lab and reflection of the light source.

The output of each Sun sensor is sent to the NanoMind flight computer via an analog-digital converter. The "typical" error in the angle measured by each Sun sensor, as reported by GomSpace is approximately 1.85 degrees, and the "maximum" error is 3.5 degrees.

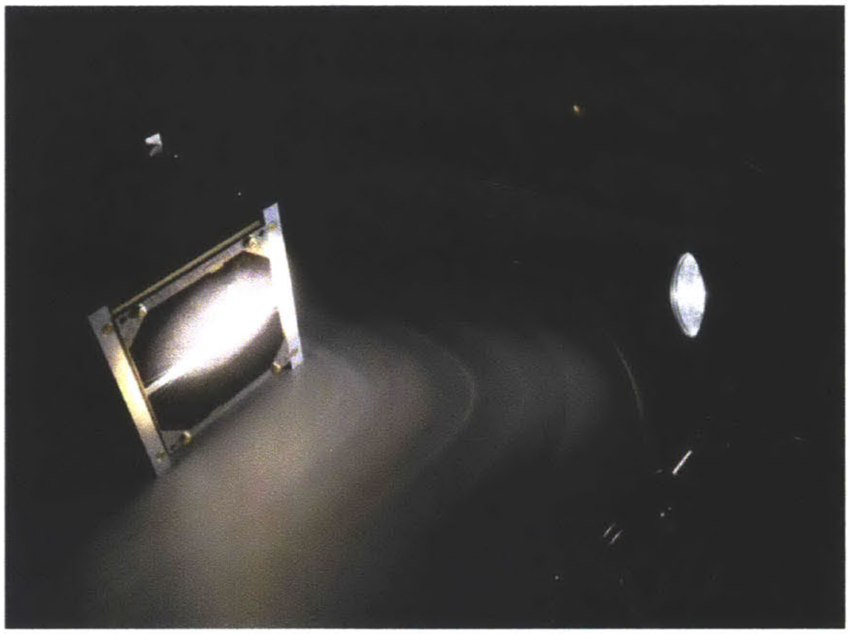
Figure 2-5: Sun sensor test setup used to record the data given in Figure 2-4.

2.1.3 MEMS 1-axis Gyros

A 1-axis MEMS gyro is mounted to the back of the GomSpace solar panels on each face of the TSat structure. The ADIS16251 gyro is manufactured by Analog Devices and measures rotation rates about the axis normal to the solar panel on which it is mounted. The gyro is also capable of integrating the rates to compute an angle of rotation. The ADIS16251 gyro is shown attached to the back of a GomSpace solar panel in Figure 2-6.

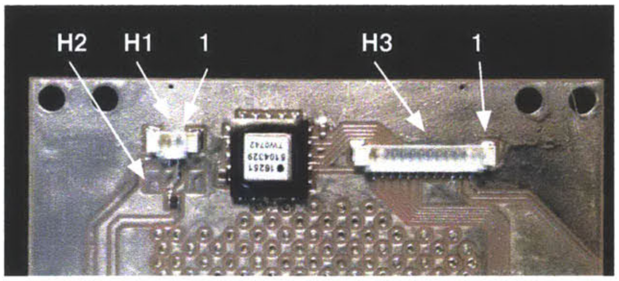

Figure 2-6: ADIS16251 gyro attached to a GomSpace solar panel (credit: GomSpace).

The ADIS16251 has a resolution of 0.00458 deg/sec and data is reported to the NanoMind flight computer at a rate of 10 Hz. The range of rates over which the measurement is accurate is ± 20 deg/sec. Noise is expected to be approximately white and Gaussian with an RMS value of 0.14 deg/sec. The gyro consumes approximately 90 mW of power in normal operation.

Figure 2-7 shows a sample output of a gyro mounted to the TSat engineering model structure. The test began with the structure at rest, and then experiencing a near-constant rotation about the axis measured by the gyro.

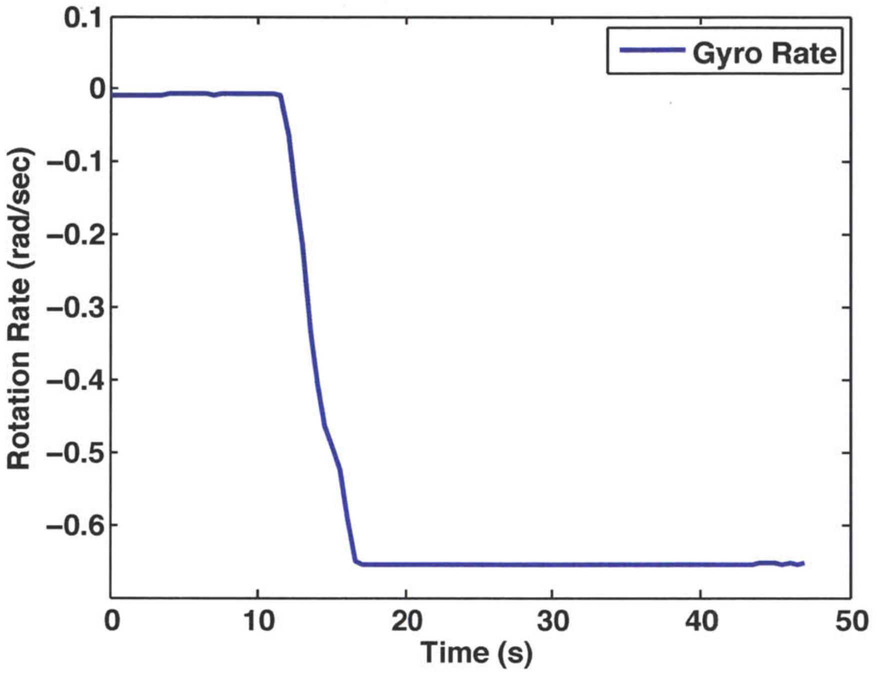

Figure 2-7: Sample output of an ADIS 16251 gyro.

2.1.4 GPS Receiver

The TSat design includes an SGR-05U GPS receiver built by Surrey Satellite Technology Ltd. of the United Kingdom. The receiver will provide position measurements to the attitude determination software. These position measurements will be used for calculation of the expected or "true" magnetic field and Sun vectors as described in Chapter 3. The operation of this GPS receiver and the interface with the GomSpace Nanomind are mainly outside the scope of this document. However, it is assumed that the receiver will provide position information with a "typical" error on the order of 10 m [9]. The SGR-05U is shown in Figure 2-8.

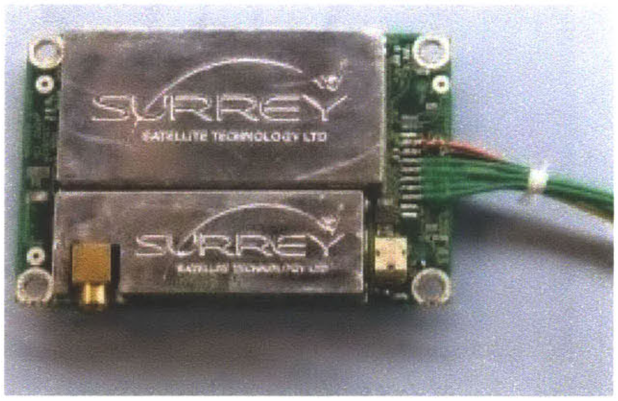

Figure 2-8: Surrey SGR-05U GPS receiver [9].

2.2 Electrospray Microthruster Actuators

TSat will use either six or eight electrospray microthruster pairs as attitude control actuators. The decision of how many thrusters to use will be made in the future, based on the availability of flight-ready thruster hardware. Six thrusters are sufficient for providing a control torque about any rotational axis, but an eight-thruster configuration is preferred. From here on, it will be assumed that mention of a single "thruster" refers to a pair of iEPS modules positioned next to each other and operating with opposite polarity, as discussed in Section 1.2.2.

2.2.1 Thruster Configuration

A trade study was carried out to determine the number of thrusters necessary to complete TSat on-orbit demonstrations. The demonstrations include effective 3-axis attitude control and the ability to control rotation with no net force on the spacecraft. The ability to rotate without imparting a net force is important for formation flight missions where precise position control is necessary. Control torque with no net force may be achieved by positioning two thrusters such that their applied moment is in the same direction, but their applied forces are in opposite directions, as shown in Figure 2-9.

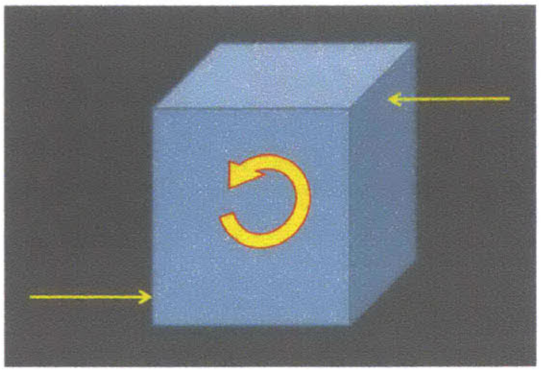

Figure 2-9: Conceptual illustration of two thrusters that produce a control moment on the spacecraft with zero net force. (credit: Akshata Krishnamurthy)

For purposes of the trade study, a distinction was made between "balanced thrust," where thrusters are paired as in Figure 2-9, and "unbalanced thrust," where control moments lead to extraneous net forces on the spacecraft. The number of thrusters required was determined for all combinations of "balanced" and "unbalanced" control axes. It was determined that three-axis control, as well as the ability to produce a control moment with no net force could be demonstrated with eight thrusters. The eight-thruster configuration consists of two "unbalanced" control axes with two thrusters each, and one "balanced" axis with four thrusters. A summary of the trade study is given in Figure 2-10.

In the event that eight flight-ready thrusters are not available for TSat, a six-thruster configuration may be used. The six-thruster configuration consists of three "unbalanced" control axes, with two of the four thrusters removed from the axis that is "balanced" in the eight-thruster configuration. The six-thruster configuration will demonstrate 3-axis attitude control, but is not capable of producing a control moment without net forces on the spacecraft.

From the perspective of attitude determination and control design, the six-thruster configuration is identical to the eight-thruster configuration, with the exception of ADCS software configuration constants. Since the six-thruster configuration is less complex, the design and analysis in the remainder of this document will assume the six-thruster configuration

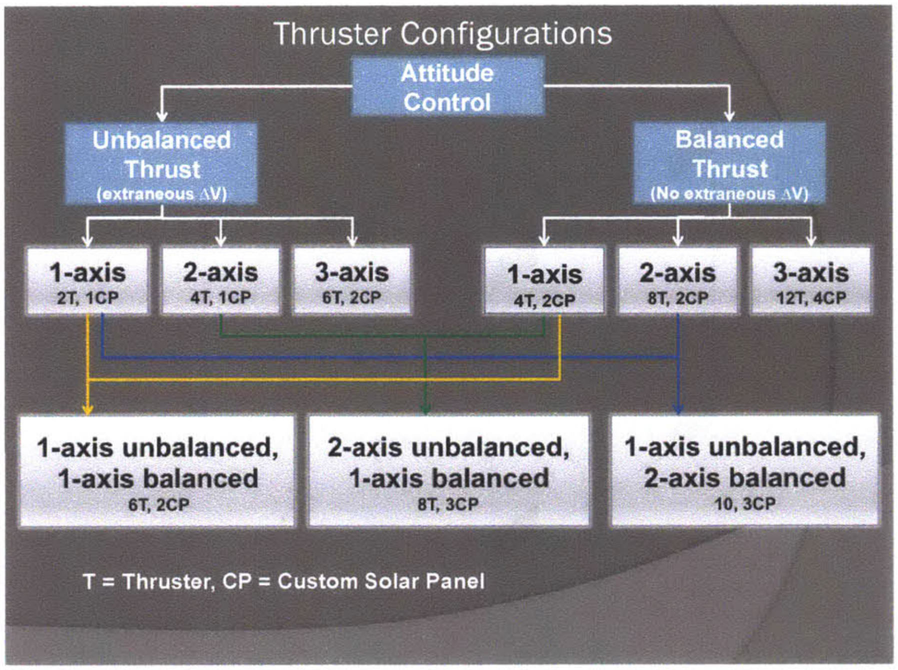
Thruster Configurations
Attitude Control
Unbalanced Thrust (extraneous ΔV)
Balanced Thrust (No extraneous ΔV)
1-axis 2T, 1CP
2-axis 4T, 1CP
3-axis 6T, 2CP
1-axis 4T, 2CP
2-axis 8T, 2CP
3-axis 12T, 4CP
1-axis unbalanced, 1-axis balanced 6T, 2CP
2-axis unbalanced, 1-axis balanced 8T, 3CP
1-axis unbalanced, 2-axis balanced 10, 3CP
T = Thruster, CP = Custom Solar Panel

Figure 2-10: Summary of the thruster configuration trade study. Three-axis attitude control, with control about one axis that produces no net force on the spacecraft is possible using eight thrusters. Additionally, the number of custom built solar panels required for each thruster configuration was determined. The implications of the number of custom built solar panels are beyond the scope of this document. (credit: Akshata Krishnamurthy)

shown in Figure 2-11. The software modifications required to support eight, or any number of thrusters are straightforward. These modifications are noted in the description of the attitude determination and control software design given in Chapter 3.

2.2.2 Fundamentals of Operation

As mentioned in Section 1.2, one key component of electrospray thrusters is an array of porous emitter tips. These tips are filled with an ionic liquid, which is a salt that is molten in the operating temperature range of the thrusters. Many ionic liquids with varying densities, ion masses, and other properties have been tested by electrospray researchers, and a summary is given in [4]. The ionic liquids being considered by the MIT Space Propulsion Lab as propellant for the iEPS thrusters are EMI-BF_4 and EMI-Im. The majority of thruster

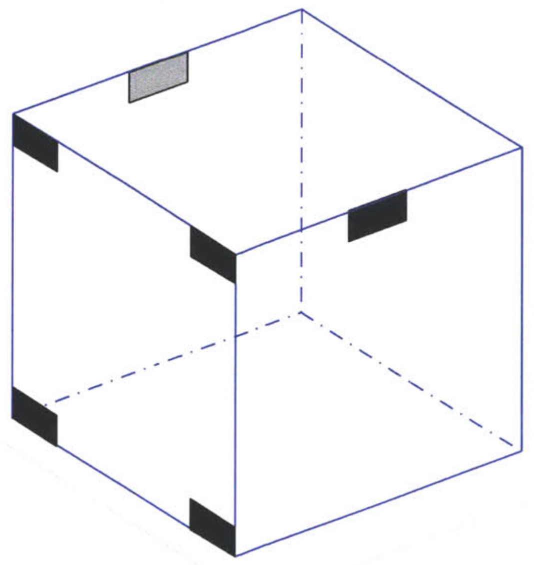
Figure 2-11: TSat six-thruster configuration provides control moments about any rotational axis.

testing to date has used EMI-BF_4 as a propellant, so this document will assume that the TSat thrusters will use EMI-BF_4 unless otherwise noted. The use of EMI-Im is under consideration as a possible upgrade to the current system, since it is more hydrophobic than EMI-BF_4. Water contamination of the propellant decreases the performance and lifetime of a thruster, so the use of EMI-Im could alleviate the need for special packaging to shield the thruster from humidity prior to launch. Both propellants have a vapor pressure close to zero, so they will not evaporate or boil in the vacuum of space. In addition, they have a surface tension sufficient to allow capillary action to carry the liquids into the porous emitter arrays from fuel tanks located on the back side of the arrays.

A potential difference on the order of 1-2 kV is applied between the ionic liquid inside the porous emitters and a grid positioned tens of microns above the emitter array [11]. This causes the ionic liquid to form a sharp cone on the top of each emitter tip. This phenomenon, called a Taylor cone, is caused by the interplay of electrostatic forces and the surface tension of the ionic liquid. The sharp tip of the cone produces a near-singularity in the electric field, and results in the extraction of charged droplets from the liquid that can be as small as a single ion in size [4]. The size of the extracted droplets affects the performance characteristics of the thruster, and is determined by the properties of the propellant as well as the materials used and alignment of the thruster components. A detailed discussion of these effects is given in [4]. In nominal operation, the iEPS thrusters extract pure ions from the ionic liquid, and accelerate them through the openings in the extractor grid. Figure 2-12 is a drawing showing a porous emitter tip with ionic liquid forming a Taylor cone and releasing ions. The thrust produced by a single Taylor cone is approximately 0.1 μN [11]. However, this thrust does not necessarily scale linearly with the number of emitters due to inefficiencies arising from misalignment of the thruster components and interactions between emitter beams.

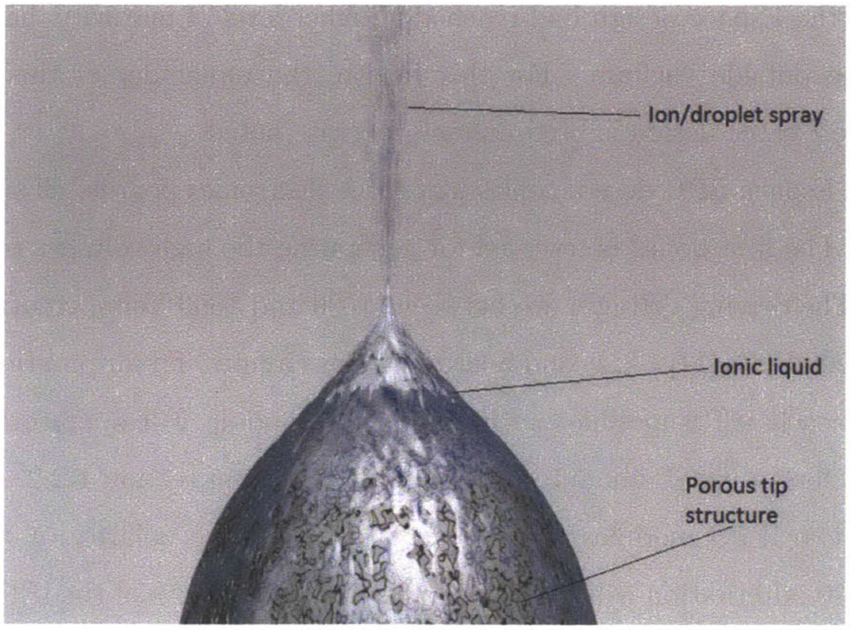

Figure 2-12: Drawing of a porous emitter tip showing the formation of a Taylor cone and extraction of ions from the ionic liquid. (Credit: MIT SPL)

Detailed specifications of the current iEPS thruster assembly design, as well as information on fabrication processes is given by [16].

2.2.3 Propulsion Power Unit

The propulsion power unit (PPU) is a key component of electrospray thruster actuation systems, since it generates the large electrical potential differences to accelerate ions and create thrust. Fitting the high voltage electronics that are required inside the CubeSat form factor is particularly challenging, since precautions must be taken against electrical shorts and arcing. The PPU must be fabricated in such a way that it operates reliably in the vacuum of space and is robust to the vibrations and loads of launch.

Two PPU implementations are being independently developed for potential use with TSat. One is being developed by Micro Aerospace Solutions of Melbourne, Florida and the other by Espace Inc. of Hull, Massachusetts. The two designs have similar requirements, and a future determination will be made as to which will be selected for flight based on the demonstrated performance of the two designs in vacuum chamber testing. At the time of this document, the Espace design had reached a higher level of maturity, having undergone succesful testing outside vacuum. For this reason, the remainder of this document will assume that the Espace design is used unless otherwise noted.

The current Espace PPU design consists of three electronics boards, all sized to fit inside a 1U CubeSat. The first board is reserved for generating the high voltages required to drive the thrusters. The driving voltages are between -1600 and 1600 Volts, creating a maximum differential of 3200 Volts [11]. A second board contains circuitry for automatically alternating the polarity of each iEPS module, such that every module will operate in positive and negative modes for an equal amount of time. As discussed in section 1.2.2, each "thruster" is made up of two iEPS modules that operate with opposite polarity at any given time. Thus, the polarity alternation will happen simultaneously for both of the iEPS modules that make up a given thruster. The rate of polarity alternation is configurable, but is expected to be on the order of 1 Hz. The third of the three electronics boards provides the interface and connections to eight thrusters. The thrusters and fuel tanks are mounted to this board as shown in Figure 2-13. Figure 2-13 represents an example arrangement of the thrusters, but they may be repositioned as desired to produce the desired control moments on the spacecraft.

The inputs from the flight computer to the Espace PPU are voltage commands with 16 bit resolution [11]. The current design supports six independent command channels. This

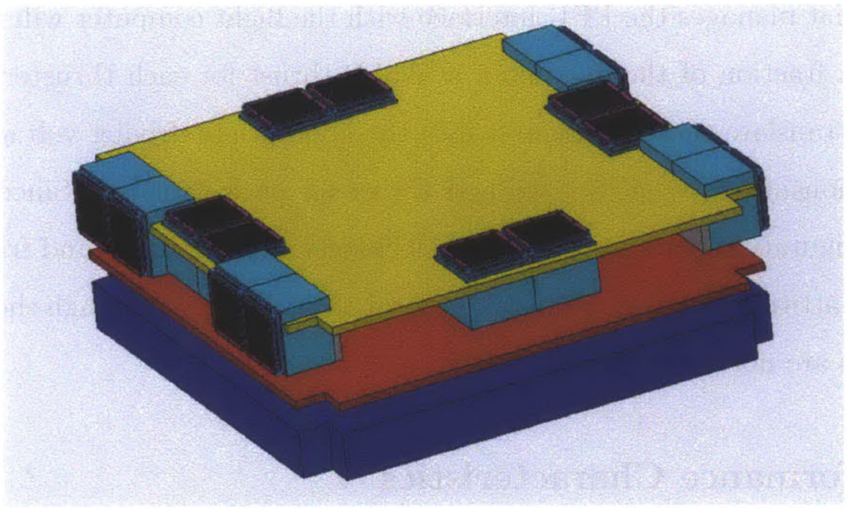

Figure 2-13: Conceptual sketch showing eight iEPS thrusters mounted on top of a stack of three Espace PPU electronics boards [11].

means that even though power is supplied for eight thrusters, there will effectively be only six control actuators. In the eight thruster case, the two channels assigned to the "balanced" control axis will drive two thrusters each. Thrusters on the same channel will have equal and opposite applied forces, and will thus impart no net force on the spacecraft, acting only as attitude actuators incapable of aiding in translational maneuvers. For simplicity, the software design presented in Chapter 3 assumes that only six thrusters are used, with each having its own individual control channel. Expansion to the eight-thruster case requires a change to the configuration parameters of the software.

The relationship between the voltage applied to a thruster and the amount of thrust produced is not precisely known, and will be determined with further characterization of the iEPS thrusters by the MIT Space Propulsion Lab. The Espace PPU feeds back measurements of the actual voltage applied to each thruster as well as the current drawn by each thruster to the flight computer. With further characterization of thruster performance, this data will be used to estimate the actual thrust produced. This thrust estimate will be used as an input to attitude determination algorithms described in Chapter 3 to reduce attitude determination error.

Since the relationship between thruster input voltage and thrust is not precisely known, an algorithm that manages the PPU interface with the flight computer will recieve thruster commands as a fraction of the maximum avaliable thrust for each thruster. The interface algorithm will translate these commands into the appropriate driving voltage based on the measured relationship between voltage and thrust for each thruster. Since the maximum thrust is a configuration parameter in the ADCS software, this command framework allows development of attitude determination and control algorithms even though the characteristics of the thrusters are not yet precisely known.

2.2.4 Performance Characteristics

Though precise performance characteristics of the iEPS electrospray thrusters remain yet to be determined through more extensive testing, the theoretical principles of thruster performance and efficiency are well developed. The thrust produced by a thruster in steady state is described by Equation 2.2, where $\dot{m}$ is the propellant mass flow rate and c is the exhaust velocity as described in Section 1.2. V is the voltage applied to the thruster in Volts, I is the currrent drawn by the thruster in Amps, $\frac{q}{m}$ is the charge density of the ionic liquid propellant in units of Coulombs per kg, and ϕ is an efficiency parameter. The maximum value $\phi = 1$ represents an idealized thruster with no efficiency losses, and the minimum value $\phi = 0$ represents an inoperable thruster. The thrust F is given in Newtons.

$$F = \dot{m}c = \phi I \sqrt{\frac{2V}{\frac{q}{m}}} \tag{2.2}$$

The efficiency parameter ϕ is only expected to reach 0.95 in the ideal case, because some losses due to interception of the ion beam by the extractor grid are unavoidable. Interception occurs when ions extracted from the emitter tips hit the extractor grid and are absorbed instead of exiting the thruster through an opening in the grid. Many factors impact the number of ions intercepted, including the quality of thruster fabrication and alignment, defects in the materials of the thruster, and the driving voltage applied to the thruster.

Thruster materials may also degrade over time, causing a thruster's efficiency to decrease over its lifetime. Determining the rate of degradation due to internal and environmental factors is one of the TSat mission objectives.

The relationship between the current drawn by a thruster and the voltage applied, referred to as the I-V characteristic, is measured by applying a sweeping range of voltages to the thruster and plotting the current drawn by the thruster at each voltage. This relationship may be stored in the form of a polynomial approximation or a lookup table on the spacecraft flight computer. Along with Equation 2.2, it will be relied upon by the thruster interface algorithm described at the end of Section 2.2.3 to translate thrust commands from the attitude control software to PPU voltage commands. Examples of measured I-V characteristics for several different thrusters are shown in Figure 2-14. The curves have different shapes due to the different efficiencies of each of the thrusters. All of the curves are flat in the center, showing the startup voltage of 700 to 800 Volts required for ions to begin to be extracted. The maximum thrust of a thruster is limited by the maximum voltage available from the PPU. Efficient thrusters such as Boro 45 have steep current-voltage profiles, and can draw high currents (thus creating high thrust by Equation 2.2) when the maximum driving voltage is applied. Less efficient thrusters such as Boro 37 have less-steep current-voltage profiles, and will reach a lower maximum current (and thus a lower maximum thrust) when the maximum driving voltage is applied. One important note is that since the thrust is directly proportional to the current as seen in equation 2.2, and current is a continuous function of voltage, there is no theoretical minimum value of the thrust that can be produced. This is applicable in cases where ultra-fine control is desired.

In addition to steady state performance, characteristics of transients in thrust during thruster startup and commanded throttle level changes are important knowlege for precision attitude control design. Transient behavior is expected to be dominated by PPU transients, since transients in the thruster itself arise mainly from the dynamics of Taylor cone formation, and are thus on the order of microseconds. The PPU voltage rise time is a tuneable parameter, where shorter rise times produce more overshoot and oscillation. The chosen

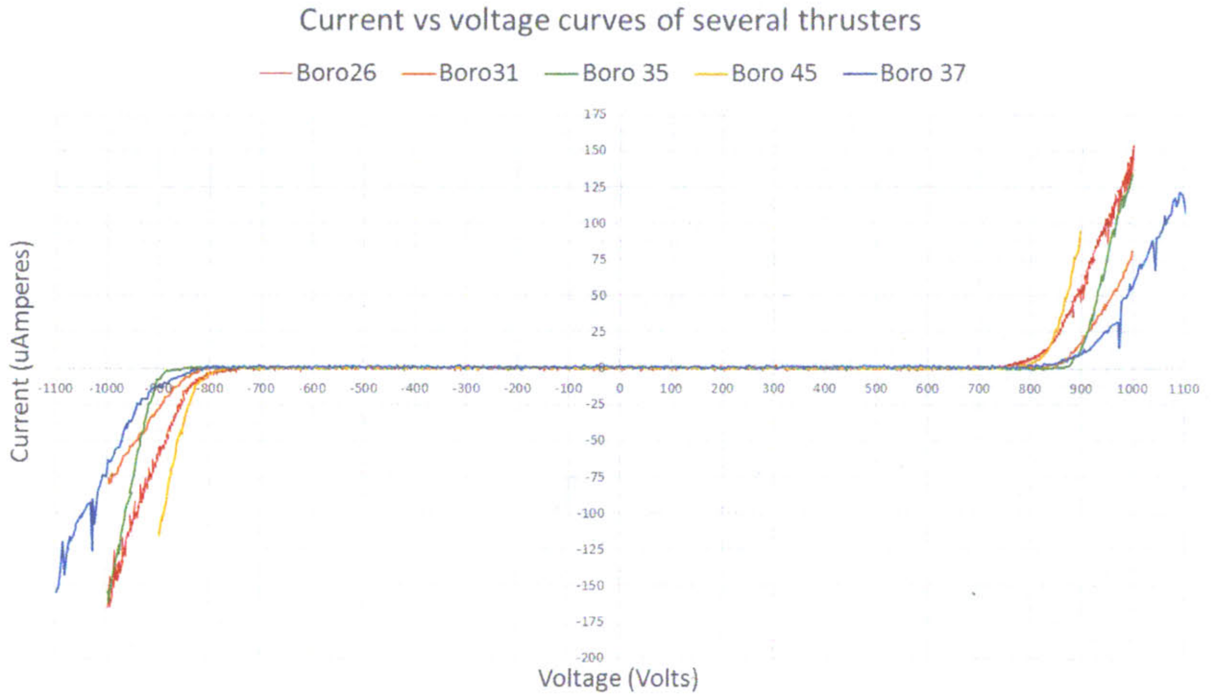

Figure 2-14: Sample I-V characteristics of several thrusters. Efficient thrusters such as Boro 45 have steep I-V curves, and reach high current levels at the maximum driving voltage. Less efficient thrusters such as Boro 37 have shallower I-V curves, and do not reach the same high currents at the maximum driving voltage. Since current is a continuous function of the driving voltage, there is no theoretical minimum thrust value. The measured I-V curves of the thrusters used for flight will be programmed into the PPU interface algorithm to convert thruster commands from the ADCS software into driving voltages. (credit: Fernando Mier Hicks)

rise time is expected to be on the order of several hundred milliseconds from zero to the maximum voltage of $\pm$ 1600 V. The length of the transient will be related to how large a voltage jump is commanded.

Chapter 3

System Software

This chapter presents the functionality and mathematical basis of attitude determination and control algorithms designed for TSat. It is written as a basic guide for implementation of ADCS flight software, and seeks to provide a foundation upon which CubeSat attitude determination and control systems may be constructed for future missions. Only nominal operation is considered, with many edge cases and failure cases ignored. These special cases will vary greatly depending upon the concept of operations and requirements of a given mission, and handling of the cases by ADCS software will be designed once those constraints are known.

The ADCS flight software was implemented in C for use in simulation and engineering model testing. The C code is listed in appendix A, along with pseudocode that describes the steps of each algorithm in detail. References are made to the sections of Appendix A where appropriate.

Many of the algorithm names contain the acronym "GNC," which stands for guidance, navigation, and control. GNC software consists of the general control logic of a spacecraft, including translational navigation and translational control as well as attitude determination and control. Thus, all of the attitude determination and control algorithms described in this chapter are classified as part of GNC, which is why they are identified as such.

3.1 Software Overview

TSat attitude determination and control software is structured as a sequence of five tasks. The sequence is repeated iteratively at a rate specified by the control cycle period configuration parameter. Each iteration of the sequence is referred to as a "control cycle." In this document, the control cycle rate is chosen to be 10 Hz.

The master attitude determination and control routine, *tsat_gnc*, calls a wrapper function for each of the five tasks in every control cycle. Each of the wrappers is identified by an algorithm name containing underscores. At the beginning of each control cycle, the sensor processing task (*gnc_sensor_processing*) reads and processes raw data from the ADCS sensors. This sensor data is incorporated by the attitude determination task (*gnc_attitude_determination*) to estimate the spacecraft attitude and body rates. The guidance task (*gnc_guidance*) processes ground commands and determines the desired spacecraft behavior based on the current mode of operation. Using this desired spacecraft behavior, as well as the estimated spacecraft attitude and body rates, the attitude control law task (*gnc_attitude_cl*) computes the control moment required to produce the desired spacecraft behavior. Finally, this control moment is converted into actuator commands by the actuator command preparation task (*gnc_cmd_prep*). The actuators fire, and the cycle repeats, beginning with the sensor processing task to read in new sensor data. Figure 3-1 shows a diagram of the overall software structure. All of the tasks update and use the values of flight software state variables stored as members of the struct fsw. Information about the flight software variables is given in Section 3.1.2.

3.1.1 Coordinate Systems and Labels

The coordinate systems referenced in the following sections are shown in table 3.1. The SBF coordinate frame is illustrated in Figure 3-2, which shows the levitation testbed configuration with only two thrusters.

The numbering of spacecraft faces is given in Figure 3-3. Solar panels, gyros, Sun sensors, magnetorquers, and any other components existing only once per face are indexed and

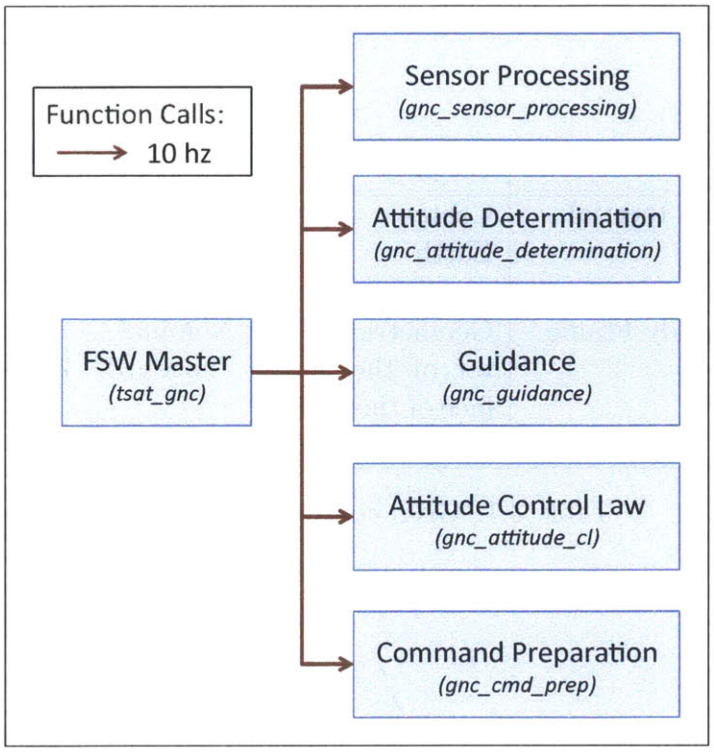

Figure 3-1: Structure of simulated software.

referred to by the number of the face on which they are mounted.

The numbering of thrusters is given in Figure 3-4, assuming a six thruster configuration.

	Full Name	Origin	X-axis	Y-axis	Z-axis
ECI	Earth Centric Inertial	Center of the Earth	Vernal Equinox	cross(Z,X)	Rotational Axis of Earth
SBF	Spacecraft Body Frame	Geometric center of the "aft" face of the spacecraft	Nominal "Forward"	Nominal "Down"	Nominal normal to orbit plane

Table 3.1: GNC coordinate systems.

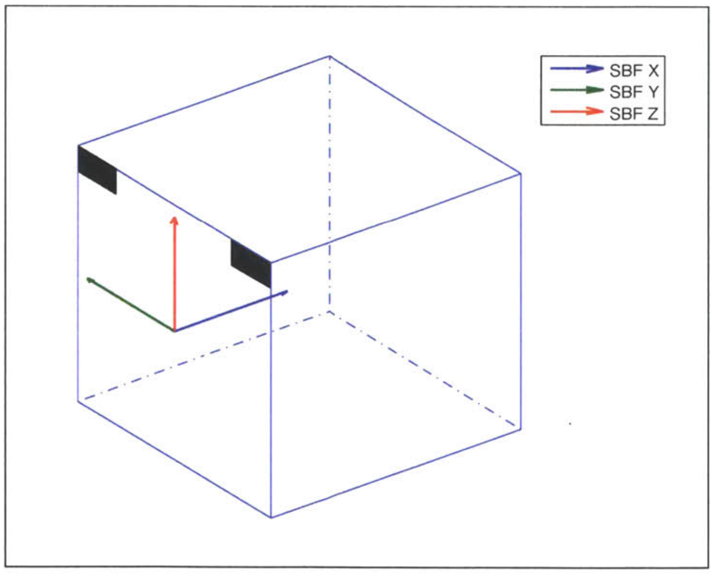

Figure 3-2: TSat Spacecraft Body Frame showing thrusters in the testbed configuration.

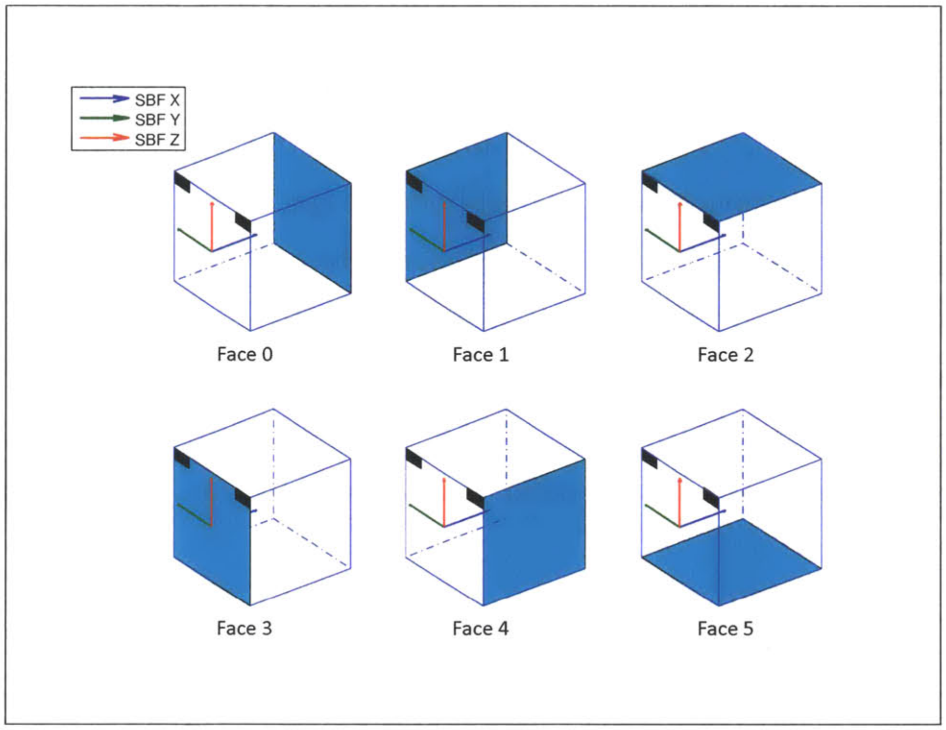

Figure 3-3: TSat spacecraft faces.

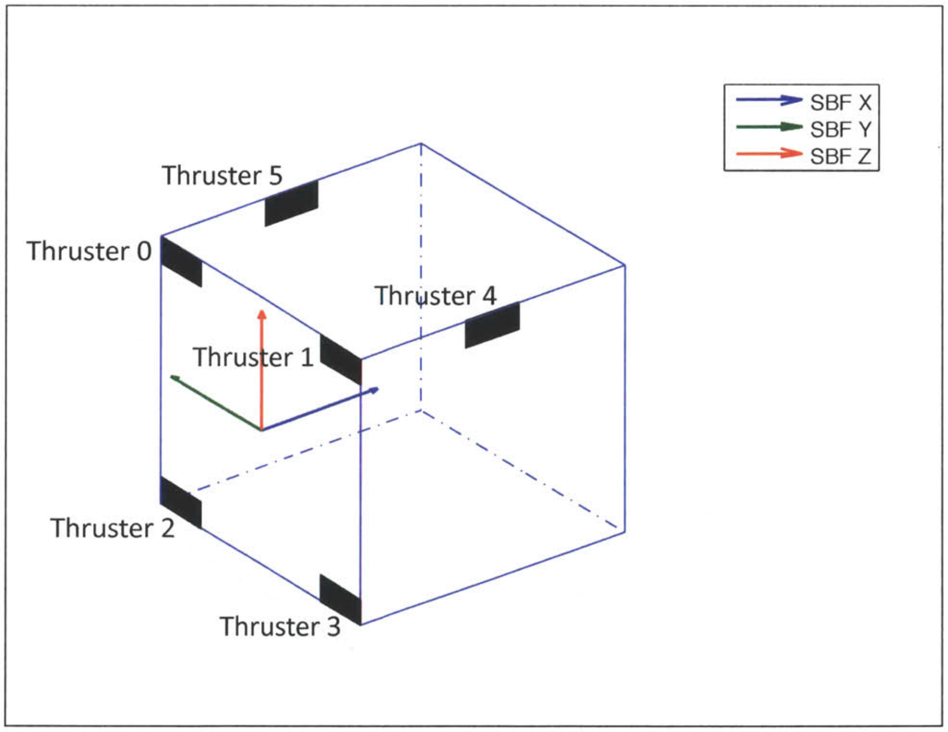

Figure 3-4: Thruster numbering.

3.1.2 Flight Software State Variable Structure

All of the flight software variables that are shared among various ADCS tasks are stored in a single data structure, fsw. The flight software time and other status variables are updated by tsat_gnc. Algorithm input values are read from the fsw struct, and outputs are written to the fsw struct for use in algorithms downstream. The values in the fsw struct remain in memory until they are overwritten by new values, and are thus constant from the end of one control cycle until the beginning of the next. Section A.8 contains a listing of the code that defines the fsw struct for the C implementation of the attitude determination and control algorithms. The listing gives a list of variables included in the fsw struct, along with short descriptions and units.

3.2 Sensor Processing

The sensor processing task (gnc_sensor_processing) converts raw data from the GNC sensors (Sun sensors, magnetometer, gyros) into a format that can be used by the attitude determination task (gnc_attitude_determination) to create an estimate of the spacecraft attitude state. The magnetometer data is processed by the nanomind flight computer before being ingested by the GNC software, so only Sun sensor processing and gyro processing are handled by gnc_sensor_processing. The gnc_sensor_processing wrapper calls gnc_process_ss and gnc_process_gyro as shown in Figure 3-5. Pseudocode for gnc_sensor_processing is given in Section A.2.1, and the C implementation used for simulation and testing is given in Section A.2.2.

3.2.1 Sun Sensor Processing

Sun sensor processing (gnc_process_ss) computes a measured Sun vector by combining data from the six Sun sensors. The output of each Sun sensor is assumed to be a voltage with some maximum value in direct sunlight. The output is expected to fall off as the cosine of the angle between the direction of the Sun and the normal to the solar panel on which the Sun

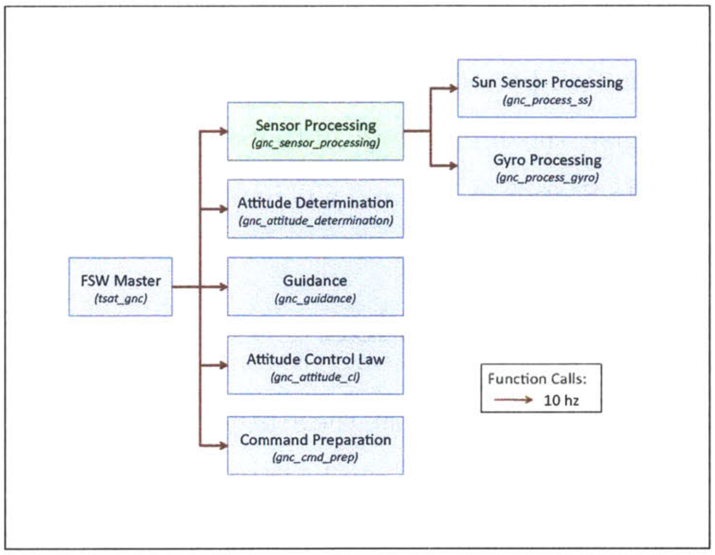

Figure 3-5: Context of the sensor processing task. Sensor processing is the first task called in each control cycle. It processes raw data from the Sun sensors and gyros, preparing inputs for the attitude determination task.

sensor is mounted. The algorithm uses this relationship to compute the cosine of the angle between the Sun vector and each of the spacecraft body frame axes. The algorithm checks the magnitudes of the computed cosines to determine whether at least one Sun sensor can see the Sun. If at least one sensor sees the Sun, the algorithm sets the flag *valid_sunvec_meas* equal to 1 and computes an SBF unit vector in the direction of the Sun, based on the computed cosines from each Sun sensor. Checks for Sun sensor failure will be added in the future.

Tables A.1 and A.2 give the inputs and outputs of gnc_process_ss. Pseudocode for the algorithm is given in Section A.2.4 and the C implementation used for simulation and testing is given in section A.2.5. Characteristics of the Sun sensors may be found in [5].

3.2.2 Gyro Processing

Gyro processing (gnc_process_gyro) performs any necessary processing or downselection of gyro data to create attitude rate measurements for the attitude filter. In its current state, Gyro processing is a placeholder, passing raw gyro data through directly to gnc_attitude_determination.

Filtering and/or smoothing of the gyro outputs may be added in the future if necessary to improve attitude determination performance.

Tables A.3 and A.4 give the inputs and outputs of this algorithm. Pseudocode for gnc_process_gyro is given in Section A.2.7, and a C implementation is given in Section A.2.8.

3.3 Attitude Determination

If a valid Sun vector measurement is available, the attitude determination task (gnc_attitude_determinati calls gnc_TRIAD, which computes a measured attitude quaternion from Sun vector and magnetic field measurements, and then calls gnc_attitude_kf which estimates the spacecraft attitude and rates. The implementation of this wrapper in section A.3.2 was created for use in simulation and for engineering model testbed testing. Some details of the algorithm will have to be adjusted for flight. The context of the algorithm is given in Figure 3-6. and pseudocode is given in section A.3.1.

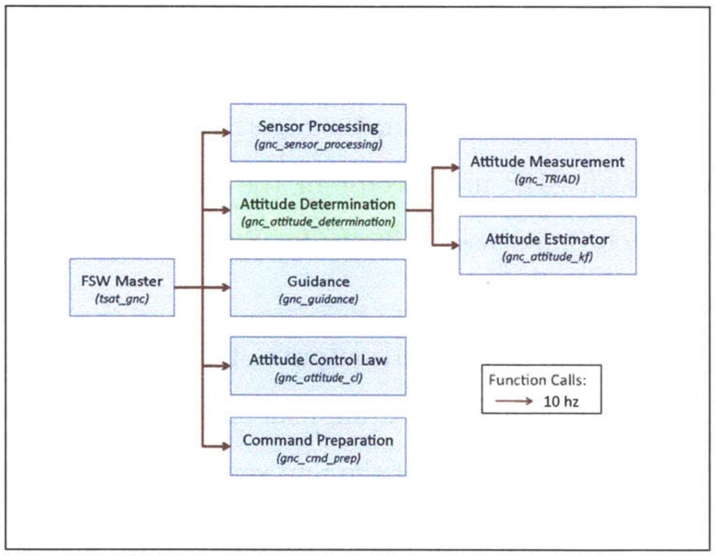

Figure 3-6: Context of the attitude determination task. Attitude determination runs immediately after sensor processing. It incorporates the processed measurements from ADCS sensors with propagated estimates to compute an estimated attitude quaternion and body rates.

3.3.1 Attitude Measurement

The attitude measurement algorithm, gnc_TRIAD, calculates a measured value of the spacecraft attitude quaternion based on GNC sensor data. The TRIAD algorithm is a standard way of fixing an attitude based on two vector measurements, as described by [22] and [1].

The algorithm first computes the "actual" direction of the geomagnetic field at the spacecraft's current position using a sixth-order spherical harmonic model described in Appendix H of [15]. The steps of the calculation are described in detail by the pseudocode in Section A.3.4. International Geodetic Reference Frame coefficients are read in from [14] to the flight software state variable struct during initialization of the flight software.

After computing the "actual" magnetic field vector, *B_actual*, the algorithm computes the unit vector *sunvec_actual* representing the "actual" direction of the Sun in the ECI frame. The calculation propagates the position of the Sun in the ECI frame from the J2000 epoch as described in [27]. The position of the spacecraft is then subtracted from the position of the Sun, and the resulting vector is normalized to form *sunvec_actual*.

Finally, the algorithm implements the TRIAD method to produce a spacecraft attitude measurement. TRIAD compares the "actual" geomagnetic field vector and Sun vector with measurements of the magnetic field and Sun vector from gnc_sensor_processing to calculate *att_quat*, the measured value of the ECI to SBF quaternion. *att_quat* becomes an input to the attitude filter.

Tables A.5 and A.6 give the inputs and outputs of the algorithm. Pseudocode is given in section A.3.4, and a C implementation is given in Section A.3.5.

3.3.2 Attitude Estimator

This algorithm implements an Unscented Kalman Filter (UKF) to estimate the attitude quaternion and body rates of the spacecraft. As described by [28], the UKF requires similar computational resources to the more common Extended Kalman Filter, but handles nonlinear dynamics in a more robust way. The inputs to the UKF include attitude quaternion measurements produced by gnc_TRIAD and rate measurements from the gyros. The measurements are combined with quaternion and body rate estimates that have been statistically propagated from the previous measurement update to produce a new estimate each time a new measurement is available. The propagation is done by creating a set of points ("sigma points") in the state space with a mean and variance represented by the statistics of the filter state. These points are propagated to the time of the next new measurement using a fourth order Runge-Kutta propagator and the nonlinear dynamics of the spacecraft system. The mean and variance of the filter estimate is then re-calculated from the distribution of propagated sigma points. The new measurement is used to update the mean and variance of the estimate with standard discrete Kalman Filter measurement update equations.

The algorithm implementation includes three helper functions: $modified\_chol$, $propagate$, and d, a dynamics function. The $modified\_chol$ function produces the offset of the sigma points from the estimate mean at the beginning of each propagation step. This includes a Cholesky decomposition of the filter covariance matrix. The sigma points are chosen according to the method proposed by Wan and Van der Merwe in [29]. The $propagate$ function propagates each sigma point using the d function, which contains the dynamics of the system.

Tables A.7 and A.8 give the inputs and outputs of gnc_attitude_kf. Pseudocode is given in section A.3.7 and a C implementation is given in section A.3.8.

3.4 Guidance

The algorithms called by gnc_guidance determine what the high-level behavior of the satellite should be based on the current mission phase, mode of operation, ground commands, and other factors. The current implementation was created for engineering model testing and only handles simple ground commands. For flight, the functionality will be expanded to handle spacecraft modes of operation and science mission test plans. The context of gnc_guidance is shown in Figure 3-7. Pseudocode is given in section A.4.1, and a C implementation is given in Section A.4.2.

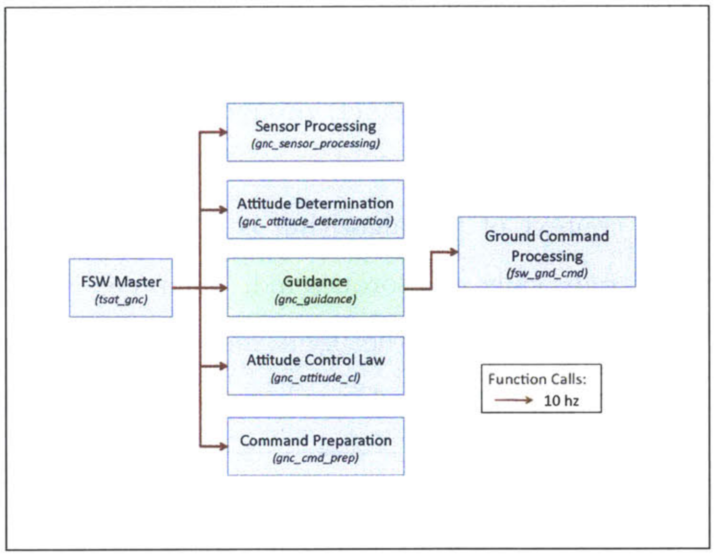

Figure 3-7: Context of the guidance task. Guidance uses the spacecraft state estimated by attitude determination to decide the desired behavior of the satellite. Ground commands and the spacecraft mode of operation are also incorporated in this decision.

3.4.1 Ground Command Processing

The fsw_gnd_cmd algorithm allows commands to be sent to the flight software while it is running. The current implementation is designed for simulation, where command sequences and timing are known before the simulation run begins. This implementation could be used for testbed testing as well, but a real-time command framework will need to be developed for flight.

In its current implementation, the algorithm looks through all of the commands read in from the fswcmd configuration file. If there is a command that is due for execution, the algorithm executes it. In TSATsim, the fswcmd file is read by a function called read_fswcmd. A sample fswcmd file is given in Section A.7.2 and a listing of the read_fswcmd function is given in Section A.7.1.

The inputs and outputs of fsw_gnd_cmd are given in tables A.9 and A.10. Pseudocode is given in section A.4.4 and a C implementation is given in section A.4.5.

3.5 Attitude Control Law

The attitude control law task (gnc_attitude_cl) produces a torque command based on the current attitude determined by gnc_attitude_determination, and the commanded attitude from gnc_guidance. The context of the algorithm is given in Figure 3-8. Currently the attitude control law task only calls gnc_torque_cmd, but serves as a placeholder so that additional algorithms may be added for the flight version of the software. Section A.5.1 gives pseudocode for gnc_attitude_cl, and a C implementation is given in Section A.5.2.

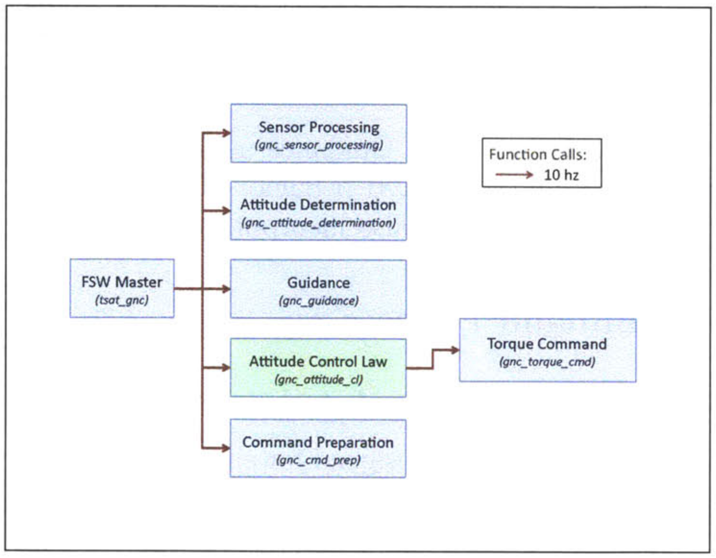

Figure 3-8: Context of the attitude control law task. The attitude control law takes in the estimated attitude and body rates from the attitude determination task, and the commanded attitude and body rates from the guidance task. It then computes the actuator torque that will move the spacecraft to the commanded state from the current state.

3.5.1 Torque Command

The gnc_torque_cmd algorithm implements a PID controller that calculates a commanded actuator torque. This torque is used by command preparation to determine commands to be sent to the thrusters or magnetorquers. The algorithm first computes the error between the commanded and estimated attitude and body rates. Using these errors, the algorithm implements a standard PID controller modeled after the control law in [17]. The output of the PID controller is a commanded torque vector, with each component representing a torque about one of the SBF axes. This commanded torque is used by actuator command preparation to produce commands for the thrusters and torque coils.

Tables A.11 and A.12 give the inputs and outputs of gnc_torque_cmd. Pseudocode is given in section A.5.4 and a C implementation is given in section A.5.5.

3.6 Actuator Command Preparation

The actuator command preparation task (gnc_cmd_prep) prepares individual commands for each of the spacecraft actuators based on the torque command computed by the attitude control law. The context of the algorithm is given in Figure 3-9. The current implementation of gnc_cmd_prep includes only allocation of the commanded torque as thruster commands. For flight, gnc_cmd_prep will be expanded to include allocation of commanded torque as magnetorquer commands, or possibly as a combination of thruster and magnetorquer commands. Additionally, the flight version of command preparation will incorporate actuator commands for translational acceleration of the spacecraft. Pseudocode is given in section A.6.1 and a C implementation is given in Section A.6.2.

3.6.1 Thrust Allocation

The thrust allocation algorithm, gnc_thrust_alloc, determines the thrust levels (as a percentage of maximum thrust) for each thruster to produce a commanded torque on the spacecraft. The most common method of thrust allocation requires that each thruster have a counterpart that produces an opposite torque to that of the original thruster. Positioning thrusters in this way allows straightforward thrust allocation by means of matrix inversion. In CubeSats, however, it is undesireable to require that thrusters be placed in a precise location relative to one another. Such requirements increase the complexity of structural design and fabrication, and decrease the number of prefabricated COTS components that may be used. Additionally, since many of the operational characteristics of electrospray thruster actua-

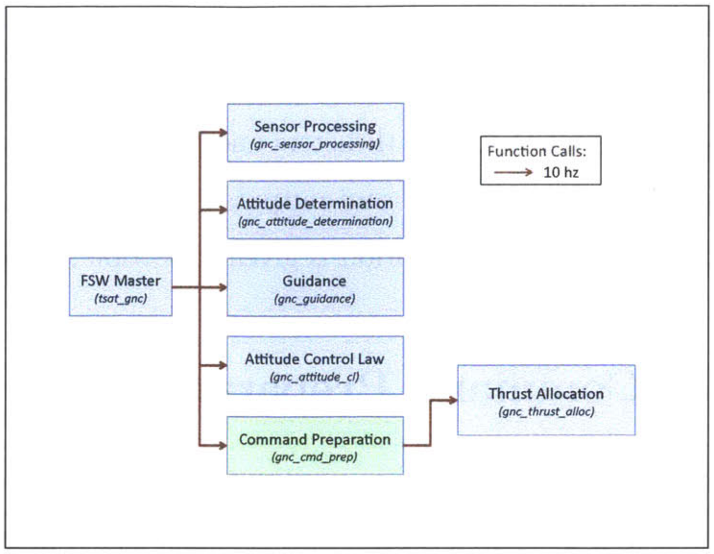

Figure 3-9: Context of the actuator command preparation task. Command preparation allocates the commanded torque from the control law to the appropriate actuators. Actuator commands are computed and sent to the actuators to impart control moments on the spacecraft.

tors are unknown, and performance and alignment may vary from thruster to thruster, the assumption that each thruster could be given an opposite-torque counterpart is unwise.

An original method of thrust allocation was developed that requires only that the torque produced by each thruster is precisely known, and that the thrusters are positioned such that no plane exists that bounds the entire set of torque vectors when their magnitudes are extended to positive infinity. The second condition ensures that the thrusters are capable of producing a torque about any axis. This thrust allocation method is robust to thruster failure where opposite-torque counterpart methods are not. A failed thruster is simply removed from the list of available torque vectors, and as long as the remaining vectors are not bounded by any plane, full control is retained.

The algorithm begins by projecting the torque command along each of the available thruster torque vectors, and choosing the thruster that produces the torque closest to the direction of the torque command. A level of thrust is commanded of that thruster that will produce a torque equal to the projection of the torque command vector along that thruster's torque vector. The algorithm then calculates the error vector between the torque command,

and the actual torque that will be produced by the thruster. This error vector is projected along each of the available thruster torque vectors, and the thruster that produces torque in the direction nearest the error is chosen. A level of thrust determined by the projection of the error vector on the thruster's torque vector is added to any thrust that has already been accumulated on that thruster. The process is repeated iteratively until the norm of the error vector is below a threshold value, or a maximum number of iterations is reached. Finally, the algorithm scales all of the commanded thrust levels by the maximum commanded thrust level so that the actual applied torque will be in the commanded direction even if the actuators are saturated.

The pseudocode in Section A.6.4 outlines the steps of the thrust allocation algorithm in detail. Tables A.13 and A.14 give the inputs and outputs of the algorithm, and a C implementation is given in Section A.6.5.

Chapter 4

High Fidelity Simulation (TSATsim)

A simulation of spacecraft dynamics, the space environment, and TSat attitude determination and control software was developed for analysis and testing of the algorithms described in Chapter 3. The purpose of the simulation is to verify attitude determination and control performance, and to demonstrate the behavior of the spacecraft in various orbits and on-orbit scenarios. The fundamental components of the simulation (TSATsim) are simulated flight software and a model of real world dynamics. These two components run completely independently, with actuator commands and sensor data passed back and forth to simulate the data interface between real flight hardware and software. Both the flight software and the real world or "truth" components of the simulation are designed in a modular way, so that the implementation of improvements to increase the fidelity of any part of the model are straightforward. In addition, the truth side of the simulation was designed to be adapted for use as a hardware in the loop simulator.

4.1 Structure of the Simulation

TSATsim is made up of two parts: a "truth side" model of spacecraft dynamics and the space environment, and a "flight software side" implementation of TSat attitude determination and control algorithms. The only data that is shared between the two sides of the simulation are the sensor outputs simulated by the truth model and the actuator commands produced by the flight software. This exchange of data imitates the interface between the real flight software and flight hardware. The simulation is managed by a main loop that runs each of the components of the truth model. The main loop also keeps track of the truth time, and runs the flight software master routine at the rate specified by the flight software control cycle period. The control cycle rate is configurable as discussed in Section 3.1, but is assumed to be 10 Hz for purposes of this document. The structure of TSATsim is illustrated in Figure 4-1, where the "Run Flight Software" block represents the flight software side, and the rest of the blocks make up the truth side.

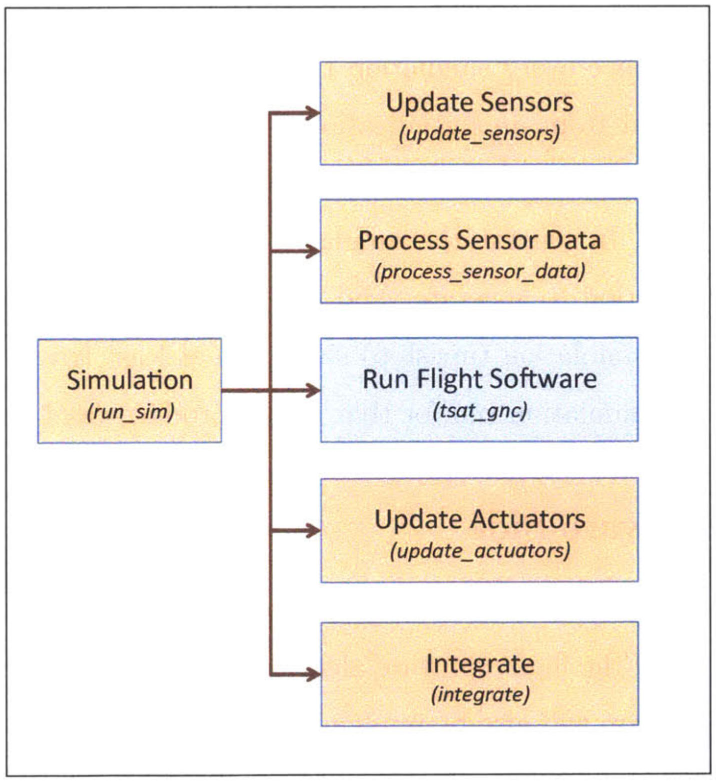

Figure 4-1: Structure of TSATsim. The "Simulation" block represents the main simulation loop, which calls all of the other blocks in order at each simulation timestep. The "Run Flight Software" block represents the simulated ADCS flight software or "flight software side" of the simulation, while all of the other blocks model the real world as part of the "truth side" of the simulation.

Each iteration of the main simulation loop first logs data from both the truth and flight software state data structures for plotting. The sensor update is called, using truth state information to simulate magnetometer, Sun sensor, and gyro outputs. In the case of hardware in the loop simulation, the simulated sensor outputs may be replaced by real sensor data. The sensor outputs are transferred from truth state variables to flight software state variables, and if an entire control cycle period has elapsed since the last time the flight software was run, the flight software uses sensor data to produce actuator commands as described in Chapter 3. Next, the actuator update is called, using the actuator commands to calculate the resulting truth forces and moments imparted on the spacecraft. Finally, the truth state is propagated forward to the next simulation timestep, incorporating moments and forces from the actuators and the environment, and the truth time is updated.

The main loop runs once every simulation timestep. The simulation timestep is a configuration parameter read from an initialization file (rwparam), and represents the time resolution of the simulation. For this document, the simulation timestep was set to 10 ms, or 1/10 of a control cycle. In general, the simulation timestep must be several times smaller than the control cycle period to generate valid simulation results. In addition, the simulation rate (inverse of the simulation timestep) should be at least twice the rate of any rates modeled in a particular simulation run for that run to produce valid results.

4.1.1 Flight Software Side

The flight software side of the simulation is made up of the algorithms described in Chapter 3 and Appendix A. All of the flight software algorithms are implemented in C for TSATsim since the real flight software will also be implemented in C. The implementation of attitude determination and control algorithms in the real flight software will thus be nearly identical to the implementation of the algorithms that will be tested and verified in TSATsim before flight. This workflow seeks to minimize the number of errors and bugs that arise in attitude determination and control algorithms during the process of implementing the algorithms on the flight computer.

4.1.2 Truth Side

The truth side of the simulation consists of algorithms that model each of the TSat sensors and actuators and the dynamics of the space environment. The truth side algorithms are implemented in C++. The algorithms are organized as functions, each of which model a specific spacecraft component or environmental factor. These functions are managed by the wrapper functions shown on the right side of Figure 4-1. For components with multiple instances, such as the gyros, the wrapper functions call the corresponding component model multiple times.

Truth state variables are stored in a single data structure. The truth state data structure contains all variables used by more than one component of the truth side simulation. These variables include information about the space environment, such as the position of the Sun, and the magnetic field experienced by the spacecraft, as well as spacecraft hardware states and sensor outputs. A full listing of variables included in the truth state data structure is given in Appendix B.

The truth side architecture was designed to be easily adaptable to incorporate design changes and new information about the performance characteristics of components. Since the electrospray thrusters, the thruster electronics, and the spacecraft are all still in the design process, many of the component models have been simplified until designs have been finalized and a more precise knowledge of the component characteristics is available from testing. When more precise performance characteristics for a particular component become available, it will be straightforward to update the corresponding component model. Such updates will increase the fidelity of the simulation by means of non-invasive and easily trackable code changes.

4.2 Simulation of Attitude Determination Hardware

TSATsim models the ADIS16251 1-axis MEMS gyro described in Section 2.1.3, the SLCD-61N8 planar photodiode described in Section 2.1.2, and the HMC5843 3-axis magnetometer of Section 2.1.1. The sensor models are called by the sensor update wrapper, as illustrated in figure 4-2. The sensor models compute simulated sensor outputs based on the current spacecraft and environment state stored in the thruth state data structure. The outputs are written to their own variables of the truth state data structure where they are made available to be read in by the flight software side.

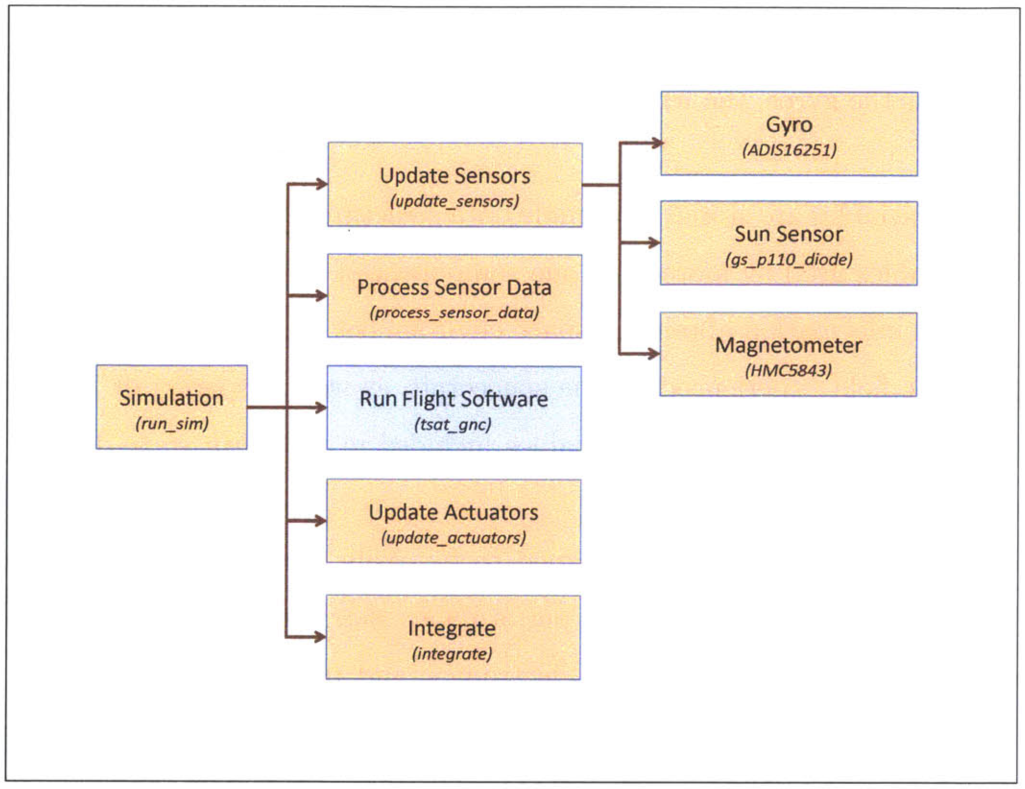

Figure 4-2: Context of TSATsim sensor models. The "Update Sensors" wrapper calls the gyro, Sun sensor, and magnetometer models, and their outputs are written to the truth state data structure.

4.2.1 Gyros

The model of the ADIS16251 1-axis MEMS gyro in TSATsim models all relevant output data of the real gyro, including "alarm" and "new data" Boolean flags. Each time the model runs, "alarm" and "new data" flags are defaulted to false, and the model then checks to see whether the internal sampling period of the gyro has elapsed. If a new sample is due, the model transforms the truth spacecraft body rates to the gyro axes and extracts the gyro frame z-axis rate. Gaussian noise is added to the output and the final value replaces the old rate value in the gyro rate register. The contents of the gyro rate register are preserved until they are overwritten, as in the real system, so there is a potential for stale data in the event of a gyro anomaly. A sample output of one simulated gyro is shown in Figure 4-3.

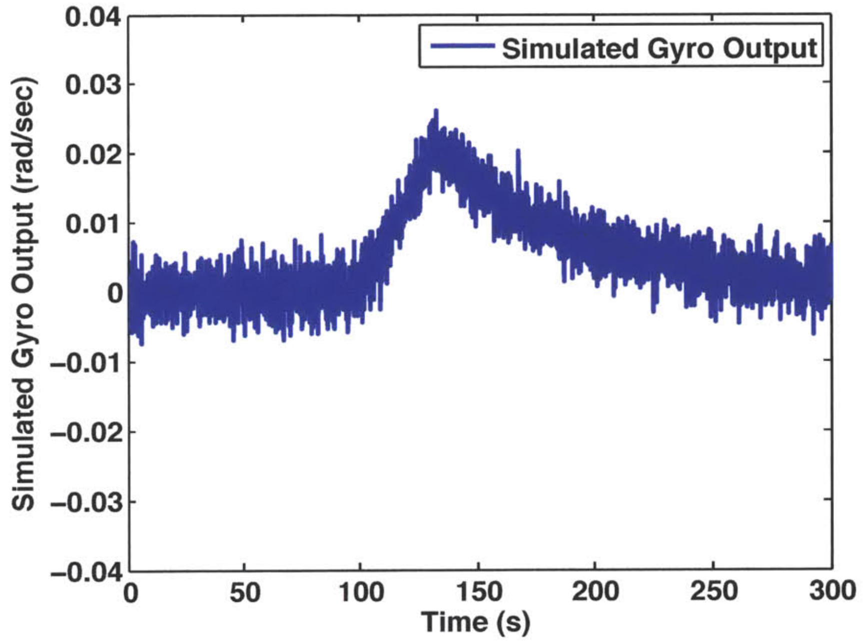

Figure 4-3: Sample output of a simulated gyro. At a time of 100 seconds, a moment is applied about the positive axis of rotation of the gyro for a duration of approximately 30 seconds. A moment is then applied in the opposite direction until the rate of rotation is close to zero.

Some improvements that would increase the fidelity of the gyro model include modeling latency in the availability of gyro data to the attitude determination software, and modeling quantization of the output. Additionally, the gyro noise model may be improved with more test data from the real gyros.

4.2.2 Sun Sensors

The TSat design includes six photodiode Sun sensors, one on each face of the spacecraft. The photodiode Sun sensor model in TSATsim computes the angle measured by a Sun sensor based on the spacecraft attitude and position relative to the Earth and the Sun. The truth state data structure contains a Boolean "eclipse" flag that is true if the Earth is between the spacecraft and the Sun. In the case of eclipse, the angle read by the Sun sensor is set to $\pi/2$ radians since this value signals to the flight software that the Sun sensor does not detect the Sun. If the spacecraft is not in the shadow of the Earth, the Sun sensor model calculates the position vector of the Sun relative to the spacecraft in the ECI coordinate frame. Positions of the Sun and spacecraft in the ECI coordinate frame are maintained in the truth state data structure as described in more detail by section 4.4. The position vector of the Sun is transformed to the SBF coordinate frame by means of the truth attitude quaterion, and then to the solar panel frame of the appropriate Sun sensor. The angle between the Sun vector and the normal to the solar panel is calculated by means of a dot product, and if the angle is greater than the edge of the Sun sensor field of view ($\pi/2$) it is set equal to $\pi/2$. Gausian noise is added to the angle, and finally the angle is converted to a voltage that will be an input to the flight software. A sample output of one simulated Sun sensor facing the Sun and then rotating away is shown in Figure 4-4.

Possible future updates to increase Sun sensor model fidelity include modeling quantization and latency introduced by the analog-digital converter that processes the photodiode voltage and accounting for the effects of thermal variation on the performance of the photodiode.

4.2.3 Magnetometer

The TSATsim magnetometer model is the simplest of the sensor models. The model transforms the magnetic field vector from the truth state to the SBF coordinate frame by means of the truth attitude quaternion, and then to the magnetometer frame. Gaussian noise is added to each component of the magnetic field measurement, and the components are con-

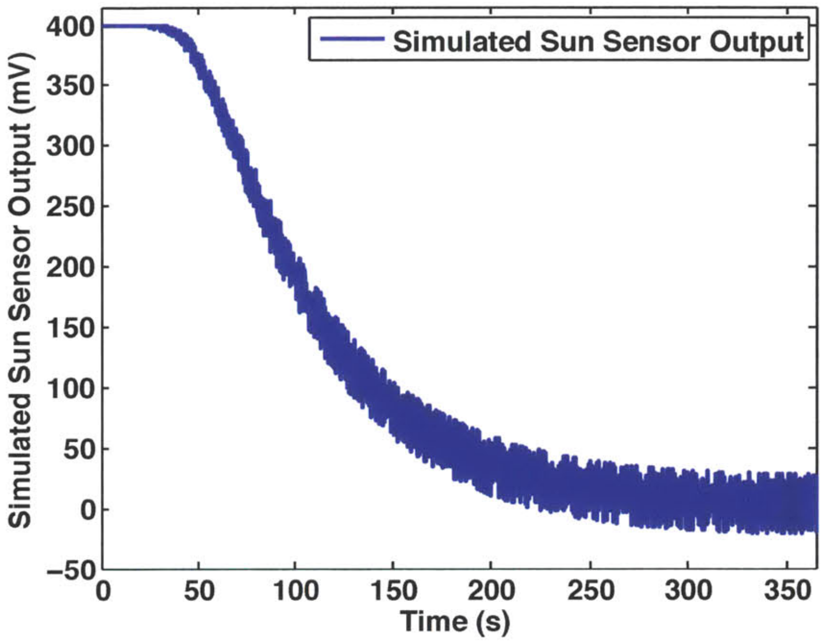

Figure 4-4: Sample output of a simulated Sun sensor. The sensor begins facing the Sun, and is then rotated away.

verted from units of Tesla to Gauss. The model maintains a count of the time since it was last run, and only updates the output vector in the truth data structure at the rate that the real flight computer samples magnetometer data. A sample of simulated magnetometer data is given in Figure 4-5.

Updates that would increase the fidelity of the magnetometer model include a more accurate noise model based on test data, and modeling the effects of magnetic fields produced by other spacecraft components such as magnetorquers.

4.3 Simulation of Attitude Control Hardware

The actuator models in TSATsim compute forces and moments imparted on the spacecraft based on the commands received from flight software. These forces and moments are combined by the actuator update wrapper function and eventually used by the integrate function to propagate the truth state of the spacecraft forward. The iEPS electrospray thruster is

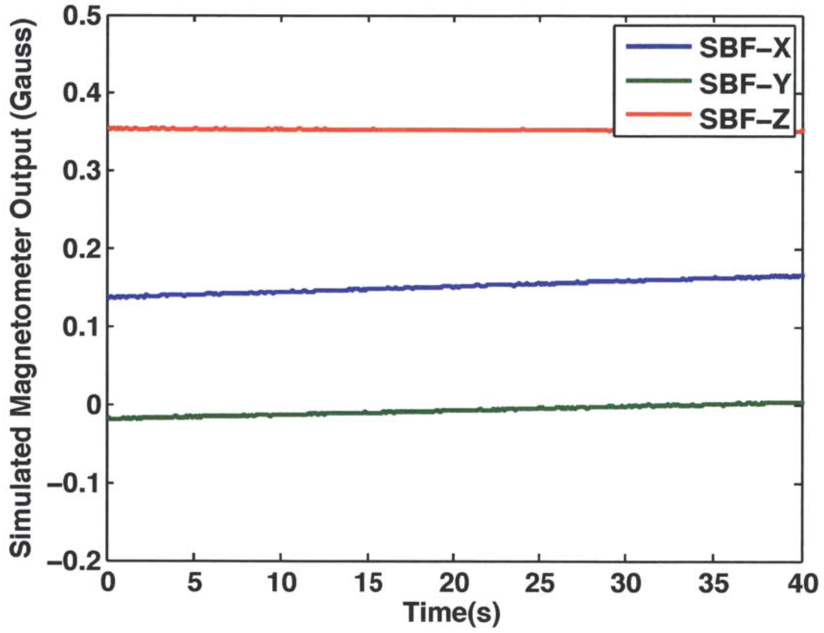

Figure 4-5: Sample output of a simulated magnetometer. Slow variation is observed as the magnetometer moves in an orbit around the Earth.

the only actuator currently modeled in TSATsim. A magnetorquer model should be added in the future. The context of actuator models is illustrated in Figure 4-6.

4.3.1 Electrospray Thrusters

The current electrospray thruster model in TSATsim simply assumes that the commanded thrust is produced precisely. Many critical improvements to this model are required to accurately represent the behavior of the actuators. Transients in thrust at thruster startup, shutdown, and during throttle level changes must be accounted for. In addition, uncertainty in thruster alignment and thruster placement must be modeled. Latency and quantization of thruster commands introduced by the PPU should also be considered. These changes were left as future work since the electrospray thrusters and PPU have yet to be rigorously tested, and many of the performance characteristics are not precisely known.

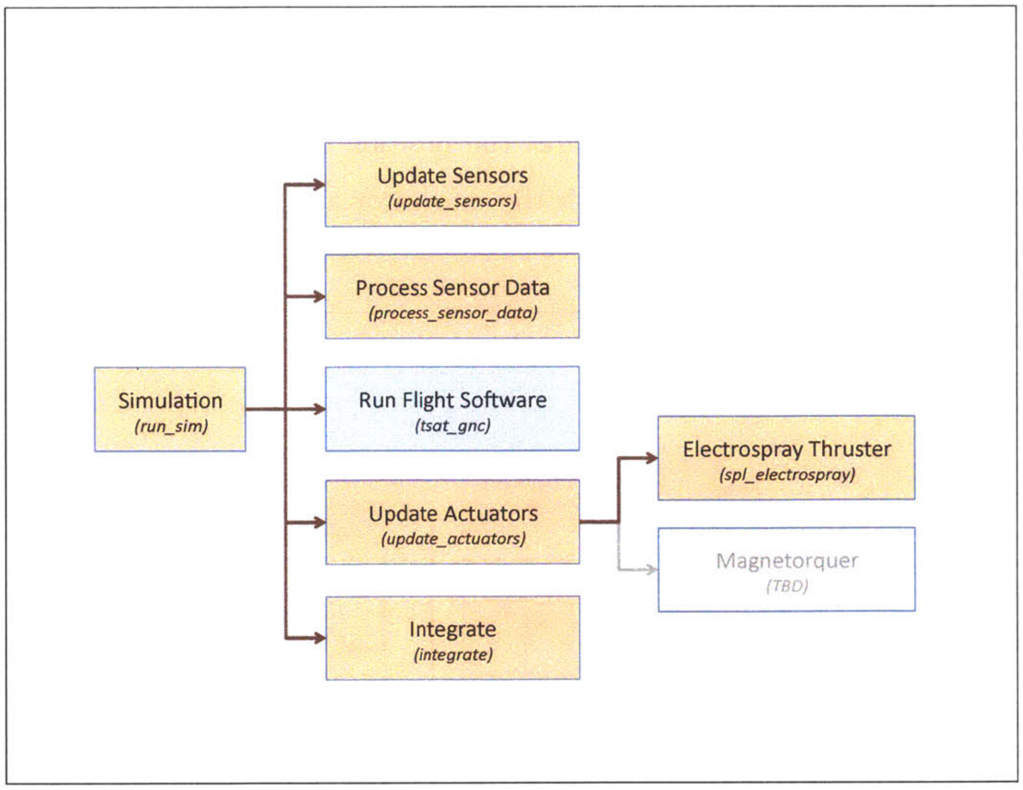

Figure 4-6: Context of TSATsim actuator models. The actuator models are updated based on commands from the simulated flight software.

4.4 Simulation of Dynamics and the Space Environment

TSATsim models the dynamics of the spacecraft and the space environment by propagating the kinematic truth states forward each simulation timestep using a fourth order Runge-Kutta integrator. The fourth order Runge-Kutta method is widely used as a computationally efficient algorithm for numerically solving second order differential equations, such as those of Newtonian dynamics [15]. The accumulated error is on the order of $O(h^4)$, where h is the integration timestep. The 10 ms simulation timestep used for this document results in error of the propagated spacecraft position on the order of millimeters. During each integration, the truth state variables that represent the space environment are updated four times, once for each intermediate state in the Runge-Kutta algorithm. The wrapper function that updates all of the space environment states is *update_environment*. Each call to *update_environment* produces forces and moments resulting from gravity, aerodynamics, and the geomagnetic field. It also updates the position of the Sun. The context of the *update_environment* function, along with the integrator and space environment models is shown in Figure 4-7.

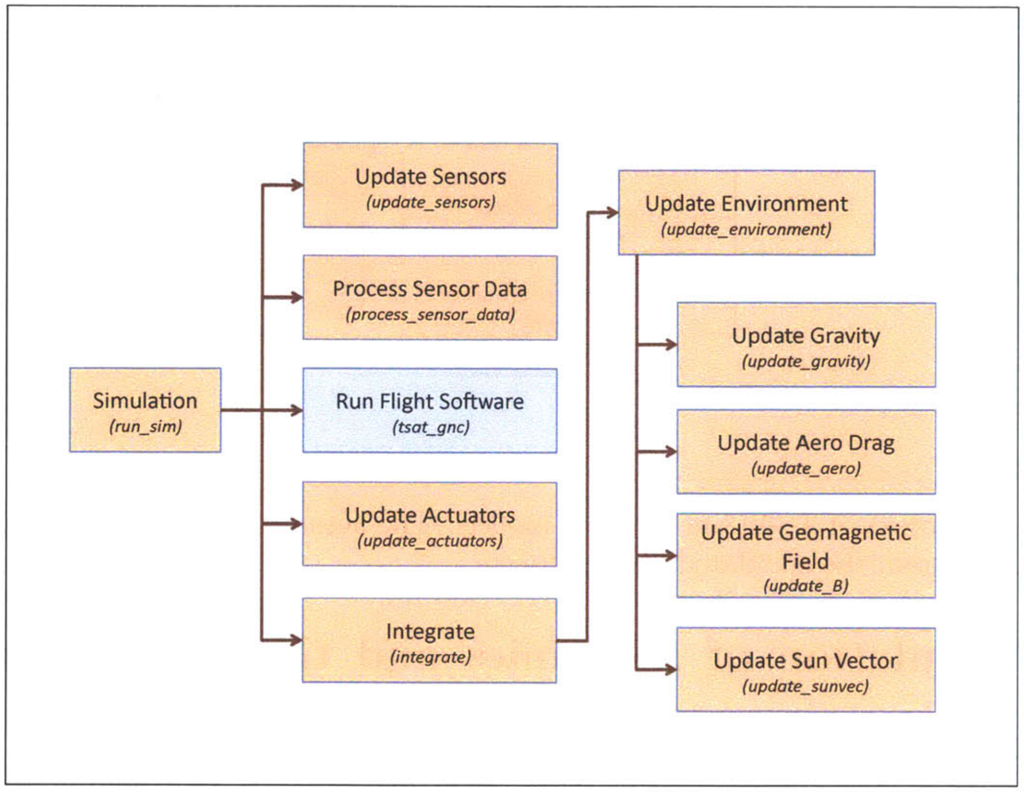

Figure 4-7: Context of TSATsim space environment models. Environment models are updated for each intermediate state of the four-stage Runge-Kutta integration.

4.4.1 Runge-Kutta Integrator

The TSATsim fourth order Runge-Kutta integrator propagates a thirteen-element state vector. The spacecraft position and velocity vectors contribute three elements each, and the ECI to SBF attitude quaternion and rotational body rates about the SBF axes make up the other seven elements. The Runge-Kutta integrator combines four Euler intergrations, each of which use a helper function that returns the derivative of each of the thirteen elements of the state vector. Since the values of the derivatives are dependent on the state, the *update_environment* function is called by the derivative helper function to update environmental forces and moments at each of the intermediate Runge-Kutta states. Thus, models of gravity, aerodynamic drag, the geomagnetic field, and the position of the Sun are updated four times during each Runge-Kutta integration. Since these updates are all based on intermediate integration states, *update_environment* is called one additional time after the entire Runge-Kutta integration is complete. Other parts of the simulation that depend on the simulated space environment thus use values calculated from the current spacecraft state, instead of from an intermediate state left over from the integration process.

4.4.2 Gravity

A J_2 model of the Earth's gravitational field, discussed in [23], calculates the gravitational force on the spacecraft. The gravitational potential of the Earth is approximated as equation 4.1, where μ is the product of the universal gravitational constant and the mass of the Earth, R_e is the radius of the Earth, J_2 is a zonal harmonic coefficient, and r and ϕ are the distance from the center of the Earth and the elevation above the equatorial plane. The force of gravity that acts on the spacecraft center of mass is the opposite of the gradient of the potential function, as shown in Equation 4.2.

$$\begin{aligned} U_{grav} &= \frac{\mu}{r}\left(-1 + \left(\frac{R_e}{r}\right)^2 J_2 \frac{1}{2}(3\sin^2(\phi) - 1)\right) \quad (4.1) \\ \sin(\phi) &= \frac{z}{r} = \frac{z}{\sqrt{x^2+y^2+z^2}} \\ F_{grav} &= -\nabla U(x,y,z) \quad (4.2) \end{aligned}$$

In its present state, the model ignores gravity gradient torque. In the future, the model should be augmented to simulate higher order zonal and tesseral harmonics, as well as moments imparted by gravity. These updates will increase the fidelity of the simulation, but in most scenerios their effect on the behavior and performance of the simulated spacecraft will be negligible.

4.4.3 Aerodynamic Drag

TSATsim uses a simple aerodynamic model to calculate the forces due to drag. The drag force is calculated according to equation 4.3, as described by [27]. C_d is the drag coefficient of the spacecraft, A is the cross section area , ρ is the density of the atmosphere, and v_{rel} is the velocity of the spacecraft relative to the air.

$$F_{drag} = -\frac{1}{2}(C_d A \rho ||v_{rel}||^2)(\frac{v_{rel}}{||v_{rel}||}) \tag{4.3}$$

In computing the cross section area, the model assumes that the shape of the spacecraft is a perfect cube with edges 10 cm in length. The velocity relative to the air is computed from the ECI spacecraft velocity and the rotation rate of the Earth. The atmosphere is assumed to be rotating with an angular velocity equal to that of the Earth's surface. The drag coefficient and the atmospheric density are read from a configuration file during initialization of the simulation. In order to model missions with a large variation in altitude (and thus in atmospheric density) the model could be adjusted to read in a lookup table of density based on the altitude. In the current model, moments imparted on the spacecraft by aerodynamic forces are ignored.

4.4.4 Geomagnetic Field

The geomagnetic field is modeled in TSATsim using the Spherical Harmonic Model described in Appendix H of [15]. This model assumes the Earth's magnetic field to be the gradient of a scalar potential function which is represented by a series of spherical harmonics. The potential, V is given by Equation 4.4, where a is the equitorial radius of the Earth, g_n^m and h_n^m are Gaussian coefficients, r is the distance from the center of the Earth to the spacecraft,

θ is the coelevation, and ϕ is the East Longitude from Greenwich [15].

$$V(r, \theta, \phi) = a \sum_{n=1}^{k} \left(\frac{a}{r}\right)^{(n+1)} \sum_{m=0}^{n} (g_n^m \cos m\phi + h_n^m \sin m\phi) P_n^m(\theta) \tag{4.4}$$

The International Geomagnetic Reference Field (IGRF) consists of a set of Gaussian coefficients g_n^m and h_n^m determined empirically by a least-squares fit to measurements of the field. TSATsim uses a 13th order model, so n ranges from 1 to 13 and m ranges from 0 to 13. The value of a adopted for use with the IGRF coefficients is 6371.2 km. New values for the IGRF coefficients are released periodically by the International Association of Geomagnetism and Aeronomy. The 11th generation values or IGRF-11 are the latest available, and were current as of December 2009. The rate of change of each of each of the coefficients is also published, and is used to correct for drift that has occured since the date of the coefficients themselves. TSATsim reads the IGRF Gaussian coefficients and their time derivatives from a text file during initialization of the simulation. The values in the text file were copied from [14].

4.4.5 Sun Position

TSATsim models the position of the Sun in the ECI coordinate frame with the mathematical basis described in Chapter 5 of [27]. The calculation begins with the Julian date, stored as part of the truth data structure, and proceeds to calculate the mean longitude of the Sun in a mean of date frame and the mean anomaly for the Sun. The mean longitude and mean anomaly are used to approximate the ecliptic longitude and to calculate the distance between the Earth and the Sun. The Earth-Sun distance becomes the magnitude of the Sun vector. The ecliptic longitude is combined with an approximation of the obliquity of the ecliptic to find the direction of the Sun vector.

4.5 Simulation Results

A 90-degree slew maneuver about the positive SBF Z-axis was simulated to demonstrate the capabilities of TSATsim and generate sample simulation results. The spacecraft was initialized with the positive SBF X-axis (face 0) facing the Sun and zero body rates. At a simulation time of 300 seconds, a commanded attitude quaternion with the negative SBF Y-axis (face 4) facing the Sun and the SBF Z-axis unchanged was sent to the spacecraft. This commanded attitude quaternion was retained for the remainder of the simulation. A summary of the commands sent to the spacecraft during the simulation is given in Table 4.1.

Simulation Time (sec)	Spacecraft Command
000.00	Attitude quaternion commanded to $[0.0000, -0.3596, 0.8295, 0.4272]$
000.00	Rotational body rates commanded to $[0.00, 0.00, 0.00]$ (rad/sec)
300.00	Attitude quaternion commanded to $[0.2542, 0.2542, -0.8886, 0.2844]$

Table 4.1: Summary of commands sent to the spacecraft during the 90-degree slew simulation. Commanded values remain active indefinitely unless they are overwritten by another call to the same command.

The spacecraft was placed in a simulated orbit similar to that of the International Space Station, with orbital elements given in Table 4.2. The slew maneuver took place during orbit day. The simulation was run for 6000 seconds, which is slightly longer than one orbital period.

Semimajor axis	6805.640 km
Eccentricity	0.001005
Inclination	51.6866 deg
Longitude of ascending node	43.6302 deg
Argument of perigee	-61.6666 deg
True anomaly	59.6506 deg

Table 4.2: Summary of the orbit used for the 90-degree slew simulation. The orbital elements in this table are the orbital elements of the spacecraft upon initialization of the simulation (at simulation time 0.0).

4.5.1 Truth Environment Data

Data from the truth side about the position and environment of the spacecraft validate that it is indeed in the intended orbit, and successfully completes the prescribed slew maneuver. Figure 4-8 shows a plot of each of the elements of a unit vector in the direction of the Sun, in the Spacecraft Body Frame (SBF). The Sun vector initially lies along the positive SBF-X axis, and at a time of 300 seconds begins a rotation toward the negative SBF-Y axis, where it remains for the rest of the simulation. A close-up view of the Sun vector during the slew maneuver is shown in figure 4-9. From Figure 4-9 it is clear that the duration of the maneuver is approximately 300 seconds, and that it takes place between the absolute simulation time of 300 seconds and 600 seconds.

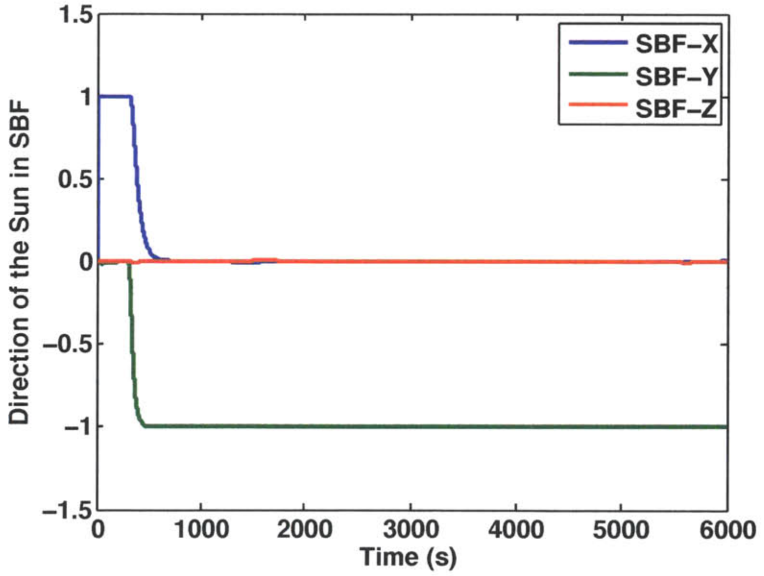

Figure 4-8: Elements of the unit vector in the direction of the Sun relative to the Spacecraft Body Frame origin. On the time scale of the simulation, the position of the Sun relative to the spacecraft may be treated as fixed, so this plot illustrates a 90-degree rotation of the spacecraft about the positive SBF-Z axis.

Figure 4-10 shows a plot of the TSATsim truth side eclipse flag. The value of this Boolean flag is 1 if the spacecraft is in the shadow of the Earth, and 0 otherwise. The eclipse flag

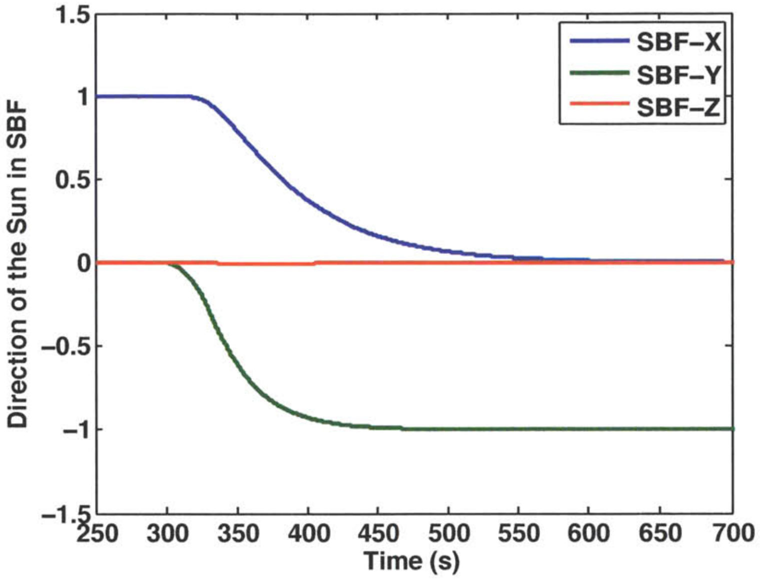

Figure 4-9: Elements of the unit vector in the direction of the Sun relative to the SBF origin for the duration of the slew maneuver.

shows that the maneuver takes place during orbit day, and that orbit night occurs between absolute simulation times of approximately 2800 and 5500 seconds.

Figures 4-11 and 4-12 show the position and velocity of the spacecraft relative to the center of the Earth in the ECI frame. Position and velocity data shows that the spacecraft was initialized in the desired orbit.

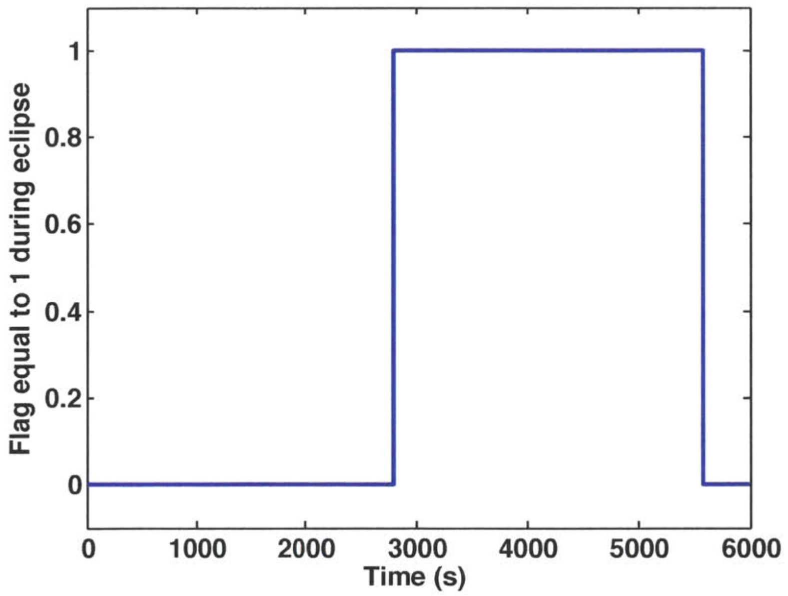

Figure 4-10: TSATsim Boolean eclipse flag. This flag is equal to 0 during orbit day and 1 during orbit night.

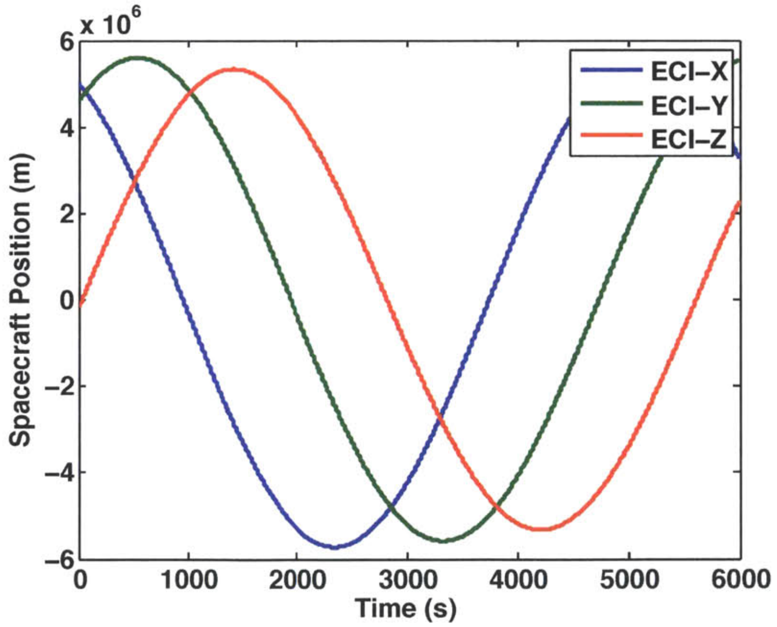

Figure 4-11: Inertial position of the spacecraft relative to the center of the Earth in the ECI frame. The simulated orbit is similar to the orbit of the International Space Station with the orbital elements given in Table 4.2.

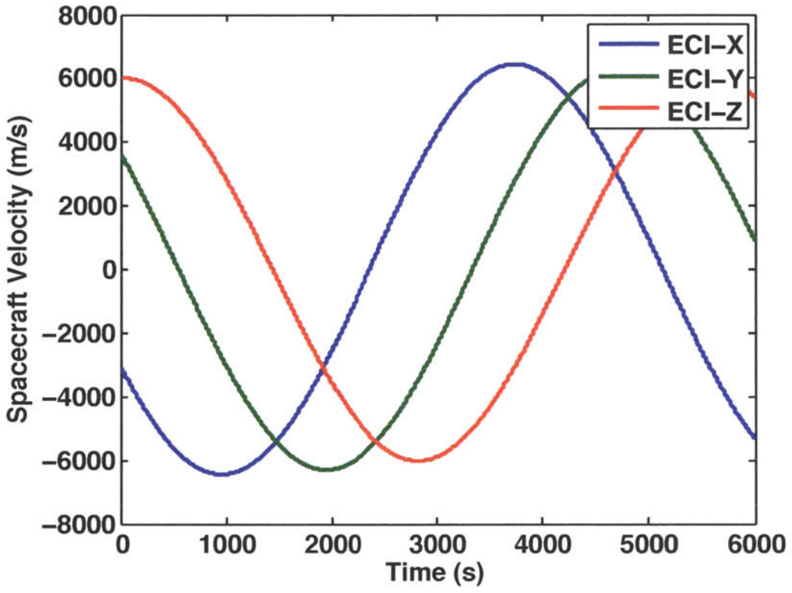

Figure 4-12: Inertial velocity of the spacecraft relative to the center of the Earth in the ECI frame. The simulated orbit is similar to the orbit of the International Space Station with the orbital elements given in Table 4.2.

4.5.2 Attitude Determination Performance

The data in this section illustrates the attitude determination performance of the spacecraft over the course of the simulation. The attitude determination error is visualized by computing the minimum angle of rotation between the truth attitude and the flight software attitude estimate. Error in estimation of the spacecraft body rates is computed by subtracting the estimated body rates from the truth body rates. Figure 4-13 shows the attitude determination error angle, and Figure 4-14 shows the error in estimation of the body rates. Note that units of degrees are used for these two figures instead of radians.

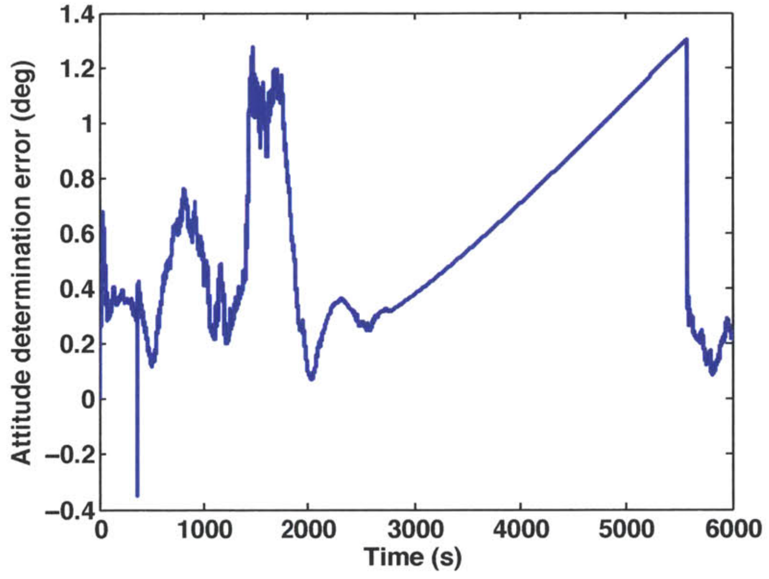

Figure 4-13: Attitude determination error. Computed as the smallest angle of rotation between the truth attitude and the flight software attitude estimate. The error is on the order of 1 degree for the entire simulation, which is acceptable for most CubeSat missions. The linear increase in error during the second half of the simulation is caused by a lack of attitude measurements during orbit night.

One feature of note in Figure 4-13 is the linear growth of the attitude determination error from shortly before 3000 seconds until well after 5000 seconds. This time period coincides with the time of eclipse, or orbit night shown in Figure 4-10. This occurs because the current formulation of the attitude determination algorithms does not allow UKF measurement

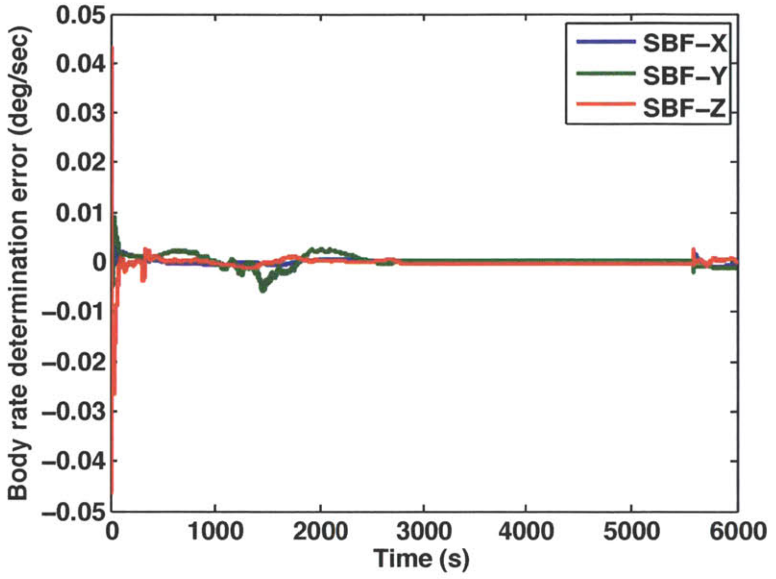

Figure 4-14: Body rate determination error. Computed as the difference between the flight software estimated body rates and the truth body rates. The error is less than 0.01 deg/sec, which is acceptable for most CubeSat missions.

updates unless a measurement of all the UKF states is available. During eclipse, the Sun is not visible and thus no quaternion measurement is available. The UKF covariance grows as seen in Figure 4-15, and the attitude estimate drifts according to any error in the estimated body rates. This behavior is likely tolerable for many simple CubeSat missions, especially given that power constraints may limit the use of thrusters during eclipse. When the spacecraft regains a line of sight to the Sun, the attitude determination error and UKF covariance decrease dramatically.

For missions requiring more precise attitude determination during orbit night, it may be beneficial to adjust the formulation of the UKF to use the available gyro data through the time of eclipse. One way that this might be implemented is called dynamic replacement, where the gyro biases are estimated by the Kalman filter instead of the body rates, and the estimated body rates are determined by combining the gyro measurments with the bias estimates. There are other potential advantages to using dynamic replacement that came to light during hardware testing and are discussed in Chapter 5. A more in-depth discussion of

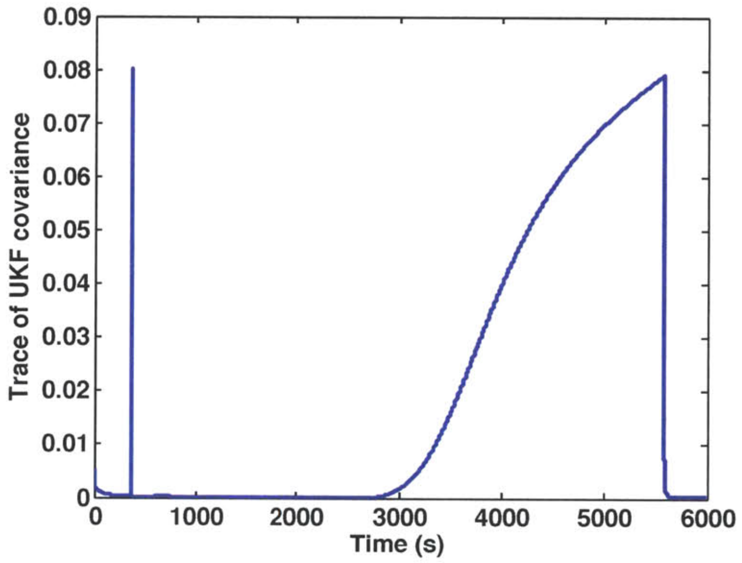

Figure 4-15: Trace of the Unscented Kalman Filter covariance. The covariance grows during orbit night, representing a growing uncertainty in the attitude estimate due to the lack of a recent attitude measurement.

the potential benefits and implementation details of an attitude filter with dynamic model replacement is given in the future work section of Chapter 6.

4.5.3 Attitude Control Performance

Attitude control performance is characterized by the attitude error and body rate error. These are not to be confused with the attitude *determination* error and the body rate *determination* error discussed in the previous section. The attitude error is the angle of rotation between the true spacecraft attitude and the commanded spacecraft attitude, while the body rate error is the difference between the actual and commanded body rates. Figure 4-16 shows the attitude error. The error starts small, spikes with the change in commanded attitude quaternion that initiates the slew maneuver, and then settles back toward zero as the spacecraft slews to the new commanded attitude. Figure 4-18 shows the spacecraft body rate error. The body rate error also spikes during the slew maneuver since the commanded body rates remain at zero, but the spacecraft must rotate to reach the new commanded attitude. The body rate error also settles toward zero as the slew maneuver is completed. The actual and commanded attitude quaternions and body rates are shown at the end of the section in Figures 4-22 through 4-25.

Figures 4-17 and 4-19 show the more precise features of the attitude and body rate errors during the steady state portion of the simulation, after the completion of the slew maneuver. The attitude error remains within 2 degrees, and the body rate error remains within $\pm$ 0.00025 radians per second. The attitude error grows linearly during the period of eclipse, due to the corresponding growth of the attitude determination error during that time. The body rate error does not grow during the period of eclipse but spikes at the beginning of the orbit day when the spacecraft corrects for the attitude error accumulated during eclipse.

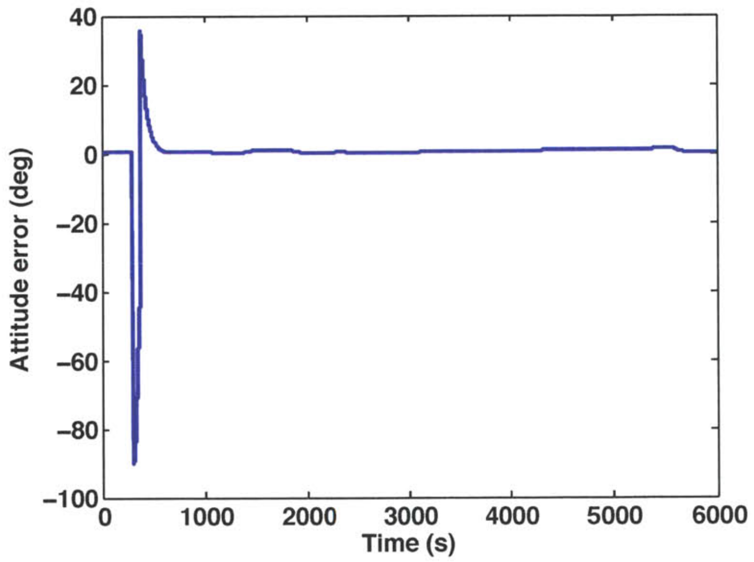

Figure 4-16: Attitude error angle. A sharp increase in attitude error is observed when a new attitude quaternion is commanded to initiate the slew maneuver. The attitude error returns toward zero as the maneuver is completed.

The actuator torque commanded by the flight software attitude control law is shown in Figure 4-20. A large torque in the SBF-Z direction is commanded by the controller at a simulation time of 300 seconds to begin the slew about the SBF-Z axis. Figure 4-21 shows

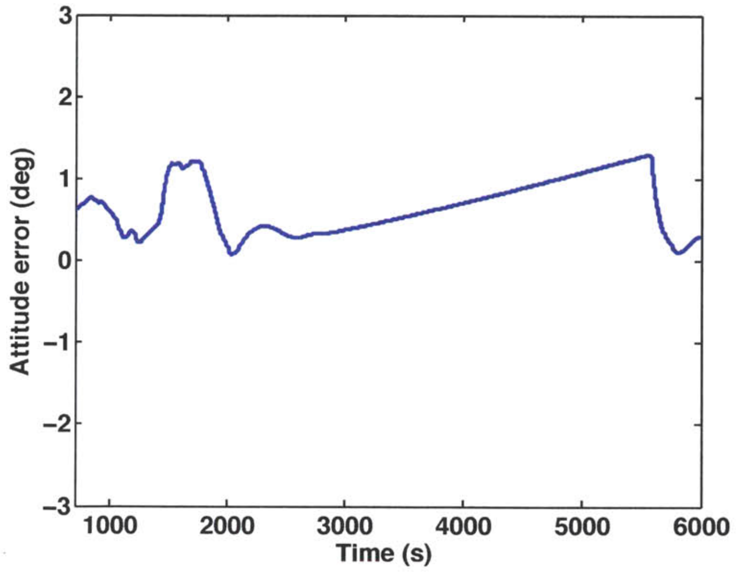

Figure 4-17: Attitude error angle during the steady state portion of the simulation after the slew maneuver is complete. In steady state, the attitude error is less than two degrees, which is acceptable for most CubeSat missions. The attitude error grows linearly during orbit night, as a result of the growing attitude determination error during that time.

the commands sent to each of the six thrusters. The thruster commands are the result of the thrust allocation law, and cause the thrusters to produce the commanded control torque.

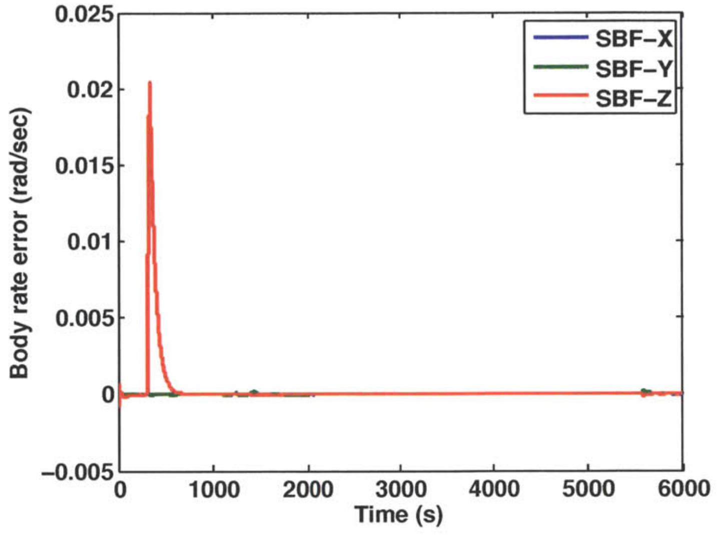

Figure 4-18: Body rate error increases during the slew maneuver, since zero body rates are commanded but the spacecraft must rotate in order to reach a new commanded attitude. Body rate error settles back toward zero as the slew maneuver is completed.

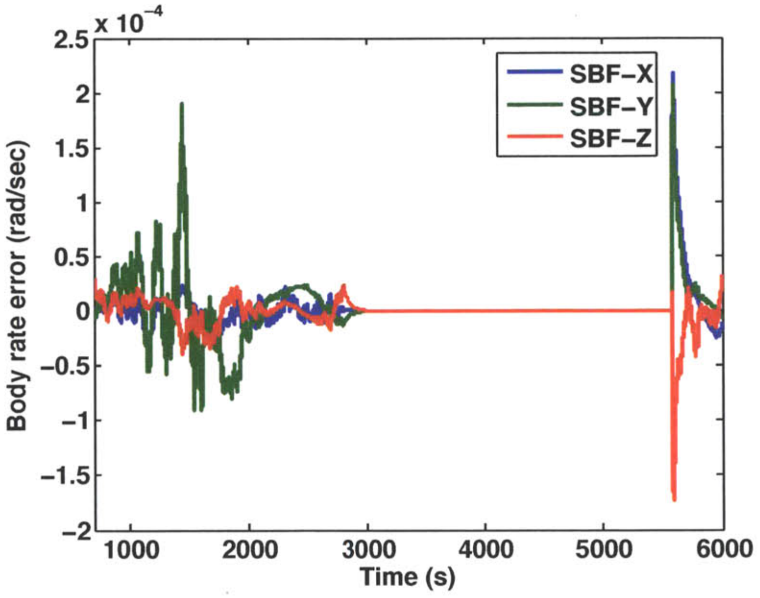

Figure 4-19: Body rate error during the steady state portion of the simulation after the slew maneuver is complete. In steady state, the body rate error remains within ± 0.00025 radians per second, which is acceptable for most CubeSat missions.

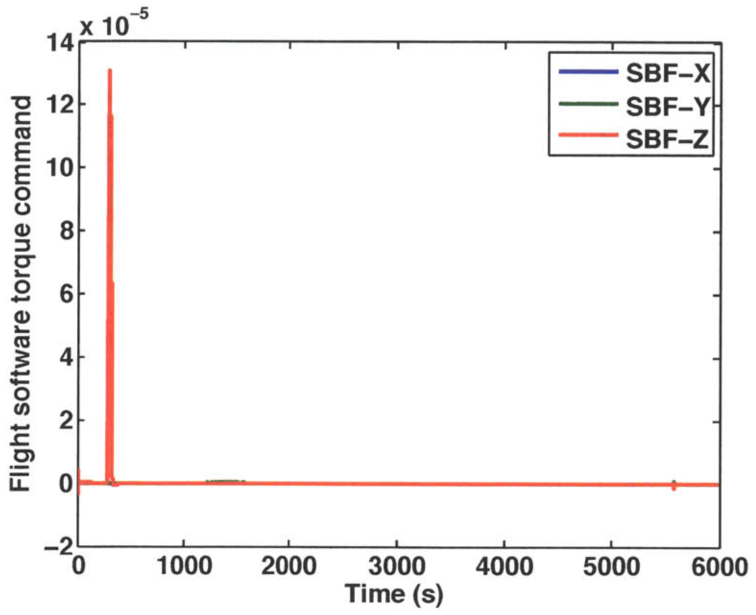

Figure 4-20: Torque command from the flight software attitude control law. A large torque is commanded about the positive SBF-Z axis to initiate the slew maneuver.

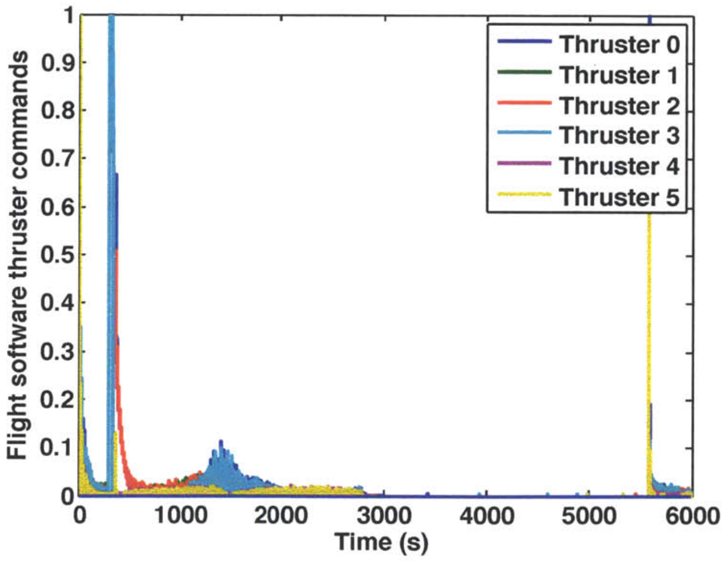

Figure 4-21: Thruster commands. Thrusters 1 and 3 are saturated to begin the rotation for the slew maneuver. Thrusters 0 and 2 are commanded to high values to slow the rate of rotation as the spacecraft reaches the new commanded attitude.

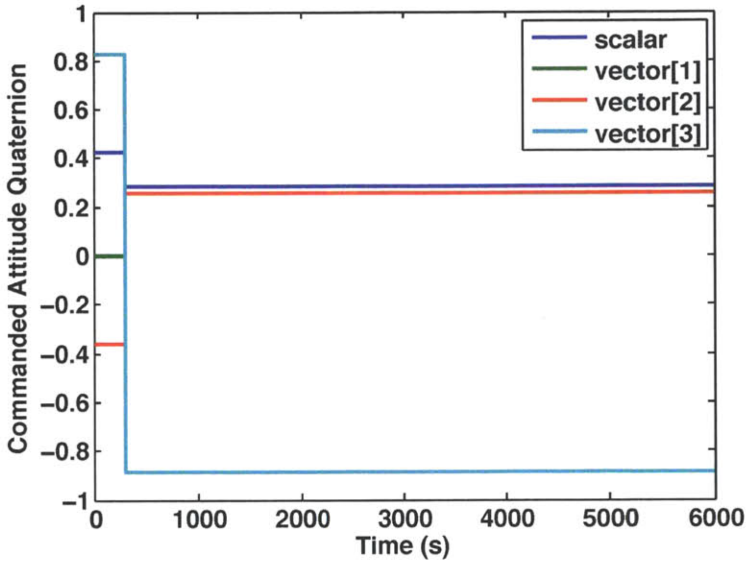

Figure 4-22: Flight software commanded attitude quaternion. A new quaternion command is sent to the spacecraft at a simulation time of 300 seconds to initiate the slew maneuver.

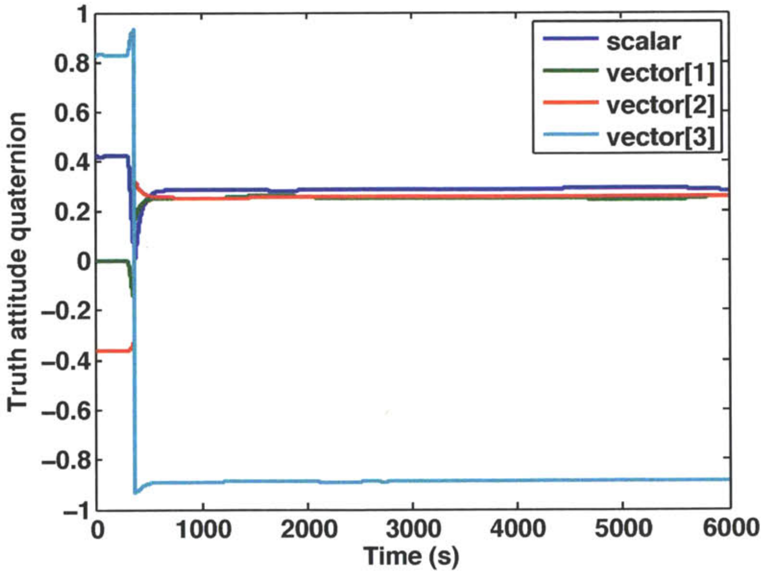

Figure 4-23: Actual attitude quaternion from the truth side model. The actual attitude quaternion converges to the commanded attitude quaternion.

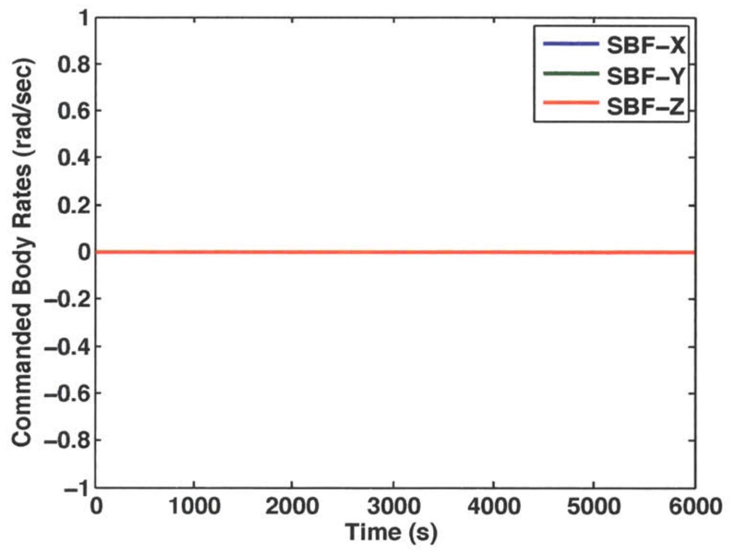

Figure 4-24: Flight software commanded body rates. Commanded body rates are zero throughout the simulation.

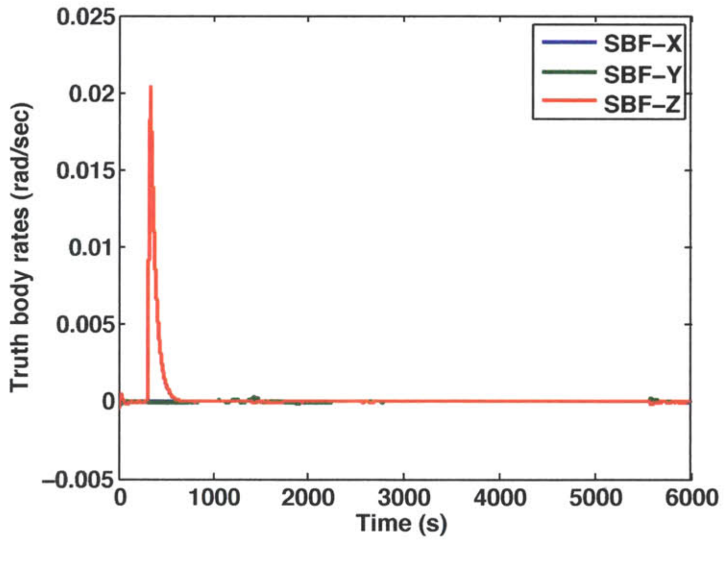

Figure 4-25: Actual body rates from the truth side model. Truth body rates are nonzero between 300 and 600 seconds to allow rotation to a new commanded attitude, but then converge to the commanded body rates.

Chapter 5

TSat 1-Degree of Freedom Testbed

The TSat 1-DOF levitation testbed is designed to test the performance of attitude determination and control hardware and software as an integrated system. The testbed consists of a magnetic levitation balance constructed by Fernando Mier Hicks of the MIT Space Propulsion Laboratory (SPL), a TSat engineering model, and a light source that simulates the Sun [16]. For attitude control tests, the apparatus will be placed inside the SPL Astrovac vacuum chamber so that the electrospray thrusters may be operated. Preliminary attitude determination tests have been attempted, while attitude control testing remains completely as future work.

5.1 Testbed Implementation

The TSat engineering model is suspended from the frame of the magnetic levitation balance using magnetic forces. There is no physical contact between the balance and the engineering model. The balance allows the engineering model to rotate freely about the SBF-Z axis. Translational motion of the engineering model and rotation about SBF-X and SBF-Y axes are not allowed by the testbed. The effects of small translational disturbances, and rotational disturbances about the SBF-X and SBF-Y axes are thought to be nearly negligible. The disturbances are thought to be damped out by magnetic moments that align the spacecraft along the SBF-Z axis. This alignment and damping of disturbances must be validated in future testing.

Separate from the balance/engineering model apparatus is a light source to simulate the Sun. For attitude determination testing, an LED flashlight was used to generate Sun sensor readings. For future attitude control testing, a more powerful Sun source may be required to generate power through the solar panels in order to power the thrusters for long duration tests.

5.1.1 Magnetic Levitation Balance

The magnetic levitation balance levitates the TSat engineering model by means of an electromagnet suspended from an aluminum frame. The electromagnet attracts a permanent neodymium magnet that is attached to a bracket on top of the engineering model. Two laser beams are projected into photodiode sensors just above and below an aluminum cylinder attached to the engineering model as shown on Figure 5-1. The photodiodes sense slight vertical motion of the engineering model, which is fed back to a PID controller that actuates the electromagnet. Since the aluminum cylinder that partially blocks the laser beams is symmetric, the control law is unaware of the rotation of the engineering model, and in turn the engineering model rotates freely. A CAD drawing of the levitation balance is shown in Chapter 1 as Figure 1-8, and a photograph of the levitation balance with the TSat engineering model is shown in Figure 5-4.

The levitation balance measures the angle of rotation of the engineering model by means of a camera positioned below the engineering model, looking up along the axis of rotation. The camera detects elements of color on the corners of the engineering model and tracks them as they rotate. Angular encoder software configured by Fernando Mier Hicks of MIT SPL then synthesizes an angle of rotation based on the varying positions of the colored elements. LEDs may be used, but for the attitude determination tests red, green, yellow, and blue ink circles were sufficient. The plate shown in Figure 5-2 was attached to the bottom face of the engineering model. The rotation angle measured by the levitation balance may eventually

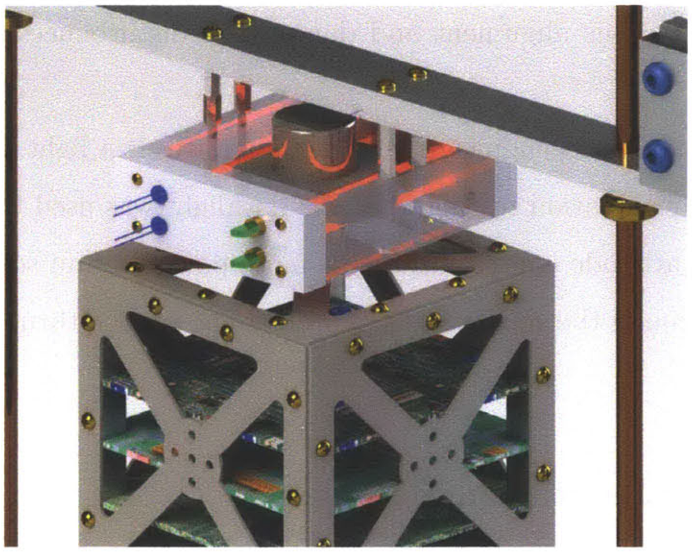

Figure 5-1: Illustration of the magnetic levitation balance laser sensing system. Lasers sense vertical displacement of the CubeSat engineering model and measurements are fed back to a control law that levitates the model by varying the strength of an electromagnet. (credit: Fernando Mier Hicks)

be used as a "truth" measurement for attitude determination testing. However, future work is needed to increase the accuracy of the measurement since the accuracy is currently on the order of $\pm$ 1 degree. This is the same order of magnitude as the expected attitude determination error.

5.1.2 TSat Engineering Model

The TSat engineering model was designed and constructed by systems engineer Mary Knapp. Almost all of the components of the engineering model are identical or extremely similar to the flight hardware components described in Chapter 2. The one major exception is the structural frame, which is modified to allow easy access to the internal components of the satellite and to support a bracket with a permanent magnet for interface with the levitation balance. Additionally, several of the sensors are absent from the engineering model. Only three of the model's solar panels are equipped with a Sun sensor and a gyro. Therefore, the three redundant gyros and three of the six Sun sensors are not available. The panels with Sun sensors and gyros are placed on face 2, face 3, and face 4 (+Z,-X,-Y). The absence of

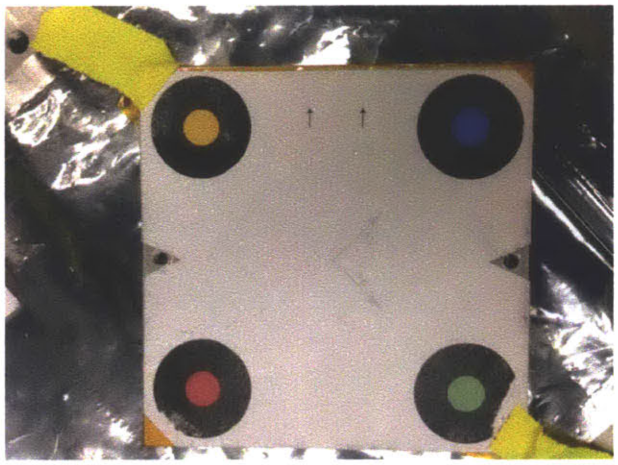

Figure 5-2: Plate attached to the bottom face of the engineering model to aid in the measurement of the angle of rotation by the levitation balance. Angular encoder software detects the colored circles to measure the angle of rotation of the engineering model.

Sun sensors on the other faces poses a particular challenge since a Sun vector measurement, and thus an attitude measurement is only available when the Sun is within the 90-degree field of view of the panels on face 3 and face 4, for 1-axis planar rotation. A photograph of the engineering model suspended by the levitation balance is shown in Figure 5-4. The electronics boards inside the engineering model structure, including the flight computer and power system, can be seen in Figure 5-3.

5.1.3 Flight Software Implementation

Micro Aerospace Solutions (MAS) of Melbourne, Florida was contracted to develop the TSat flight software platform to augment the software provided by the manufacturer. The flight software platform handles low level sensor and actuator interfaces, data logging, telemetry, and other system level tasks. The TSATSim C implementation of the attitude determination and control software was integrated with the TSat flight software platform by MAS, and slight changes were made for efficiency and interfaces with other parts of the software.

Several small engineering model-specific adjustments were made to the attitude determination software. First, panel 0 was swapped with panel 3 due to inadequate cable and harness length. The result is that Sun sensor 3 and gyro 3 are on spacecraft face 0. Sun sen-

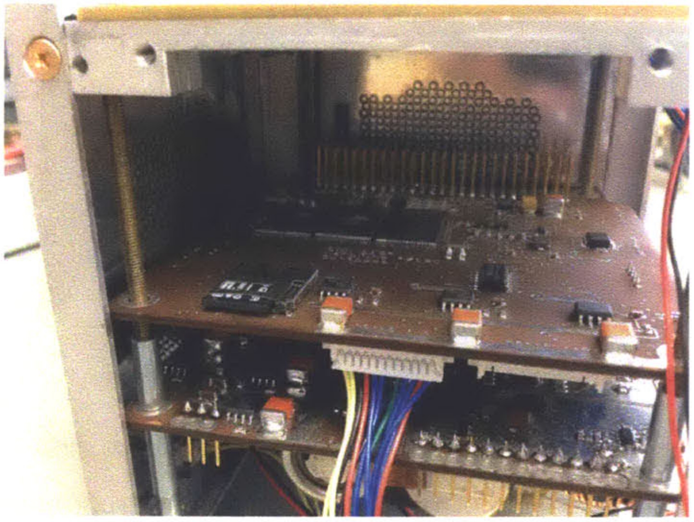
Figure 5-3: Interior of the TSat engineering model. Solar panels have been removed for interior access. The upper board contains the NanoMind flight computer and a memory card for data logging. NanoPower batteries are attached to the lower board.

sor 3 looks in the negative SBF-X direction, and gyro 3 measures rotation about the negative SBF-X axis. Sun sensor 0 and gyro 0 do not exist in the engineering model. The second adustment consists of an alteration to the dynamics equations of the UKF. When computing the derivative of a state, the engineering model UKF assumes the angular acceleration about the SBF-X and SBF-Y axes to be zero, regardless of the moments applied by actuators. As discussed in Section 5.1.1, the levitation balance is believed to counteract any moments that would cause rotation about the SBF-X and SBF-Y axes. Third, the "measured" magnetic field value is artificially fed to the software since the magnetic levitation creates a complex and noisy magnetic field that is not representative of the space environment. Finally, the Sun sensor readings of all Sun sensors except for Sun sensor 3 and Sun sensor 4 are automatically assumed to be zero. This is to reduce noise from ambient light in the room and from disconnected (missing) Sun sensors. Sun sensor 2 is assumed to read zero because the Sun source is in the plane normal to the SBF-Z axis.

During the attitude determination testing in Section 5.2, attitude determination and control software was run at a rate of 2 Hz. This was slowed from the design rate of 10 Hz due to constraints on the processing power of the flight computer. In the future, steps may

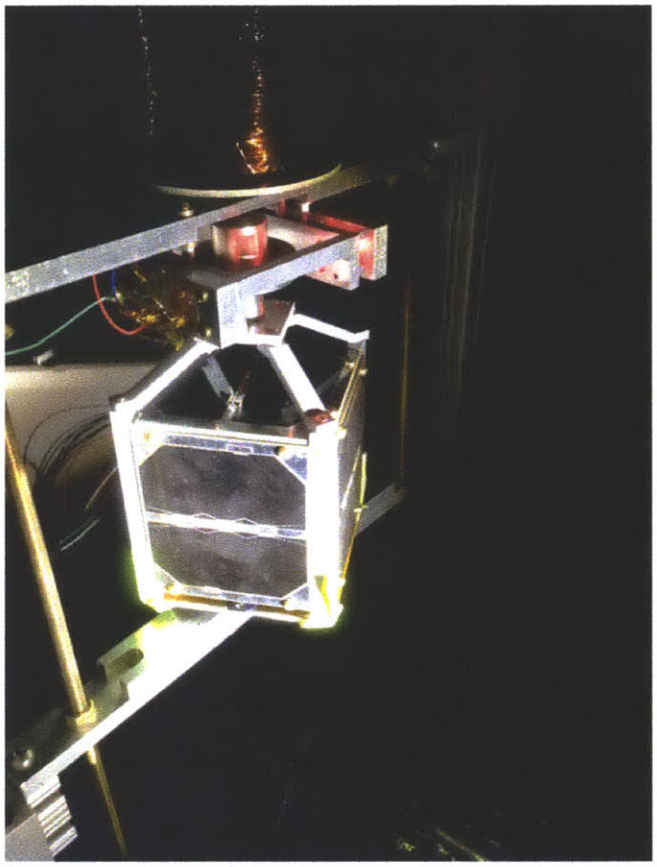
Figure 5-4: TSat engineering model levitated by the magnetic levitation balance.

be taken to increase the efficiency of the attitude determination and control software to allow a faster control cycle rate. The TSat flight software platform logs ADCS telemetry data to a removeable memory card during each control cycle for review.

5.1.4 Testbed Environment

For full attitude determination and control testing, the levitation balance and engineering model will be placed inside the Astrovac vacuum chamber in the MIT Space Propulsion Lab. A vacuum safe light source will also be placed in the chamber, away from the levitation balance apparatus. The lab and the inside of the chamber will be dark, except for light emitted from the light source "Sun." The vacuum chamber will be pumped down to simulate the vacuum of space and to allow electrospray thruster operation.

Several attitude determination tests were attempted outside the vacuum chamber as described in Section 5.2. The levitation balance and engineering model were placed in a darkroom, and an LED flashlight was placed approximately 1 meter away in the horizontal plane of the engineering model as a Sun source. The flashlight was directed at the axis of rotation of the engineering model. The testbed setup for this preliminary attitude determination testing is shown in Figure 5-5.

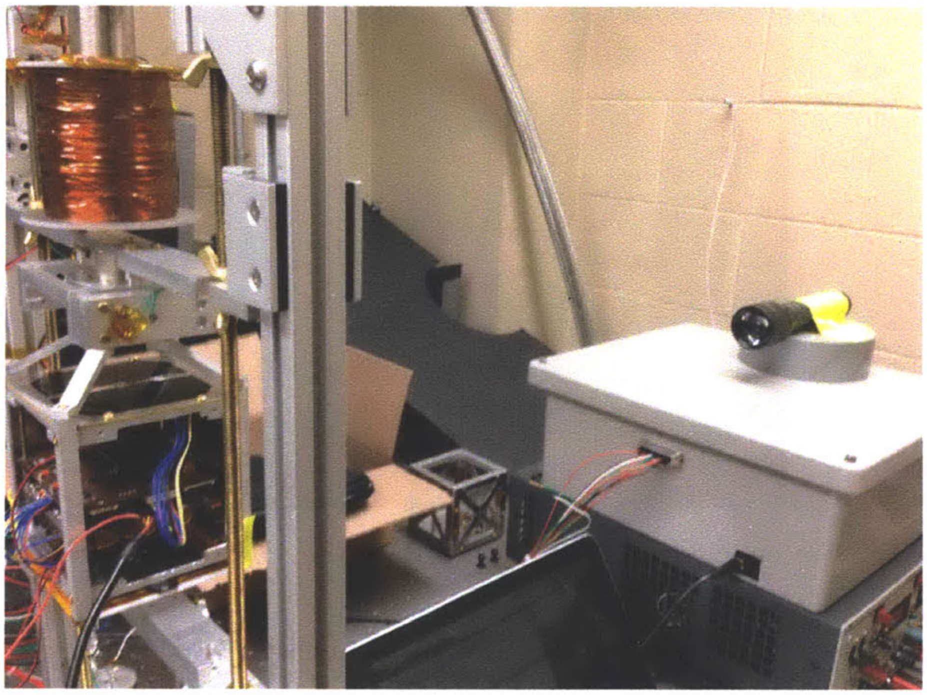

Figure 5-5: Darkroom testbed setup for preliminary attitude determination testing. The Sun source flashlight is visible on the right of the figure, and the TSat engineering model with solar panels removed for internal access is on the left.

5.2 Attitude Determination Test

Preliminary testing of TSat attitude determination hardware and software was attempted using the TSat 1-DOF levitation testbed. Thrusters were not mounted on the engineering model vehicle, and thus no control was tested. The test was carried out with the engineering model at rest, and showed convergence of the attitude filter to the true attitude and body rates. A second attempt with the engineering model in motion was less successful, prompting suggestions for improvements to the attitude filter and to characteristics of the external design.

5.2.1 Test Description

The engineering model was suspended from the levitation balance, with the negative SBF-X axis (spacecraft face 3) facing toward the Sun source. The engineering model remained stationary in this attitude for the duration of the test. The “measured” magnetic field was set to point along the positive SBF-Z axis for the entire test.

The attitude determination software on the engineering model was initialized with arbitrary estimates of the body rates and attitude quaternion. It was run at a rate of 2 Hz for a test duration of 5 minutes. The goal of the test was to observe convergence of the attitude filter to the true attitude quaternion and body rates. A secondary objective was to gather data on the performance and behavior characteristics of the attitude determination sensor hardware.

5.2.2 Test Results

The attitude filter successfully converged to the true attitude quaternion and body rates in a time of approximately 100 seconds. This is an acceptable duration for attitude acquisition. Figure 5-6 shows a plot of the attitude determination error angle in degrees, and Figure 5-7 shows a plot of the body rate determination error. Both errors converge to zero. The estimated and truth attitude quaternions are plotted in Figures 5-8 and 5-9.

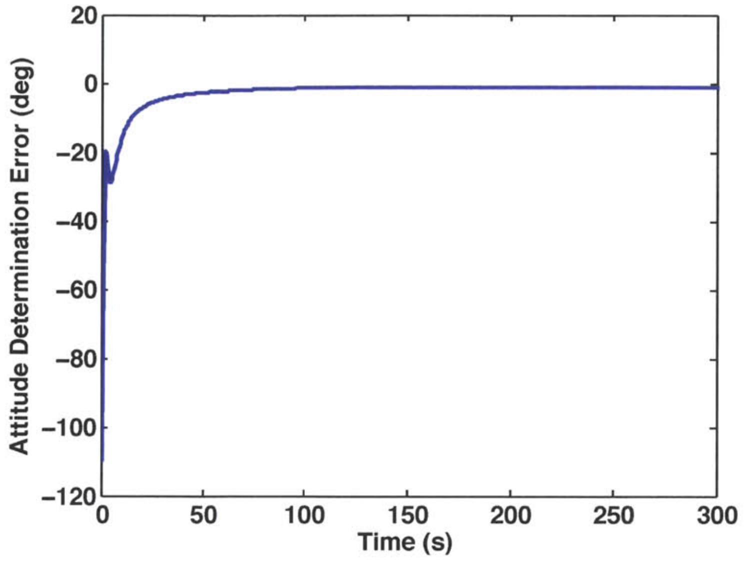

Figure 5-6: Attitude determination error. Calculated as the minimum angle between the estimated and truth attitude. Attitude determination error converges to zero in approximately 100 seconds.

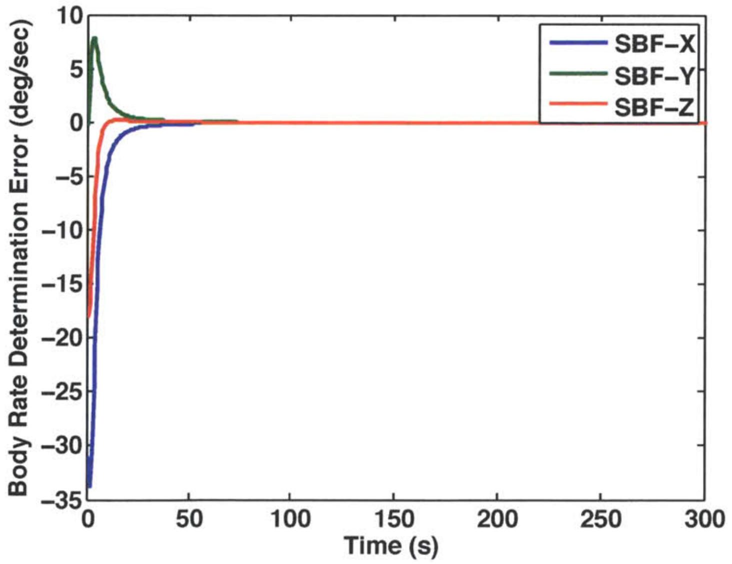

Figure 5-7: Body rate determination error. Calculated as the difference between the estimated and true spacecraft body rates. The body rate determination error converges to zero in approximately 100 seconds.

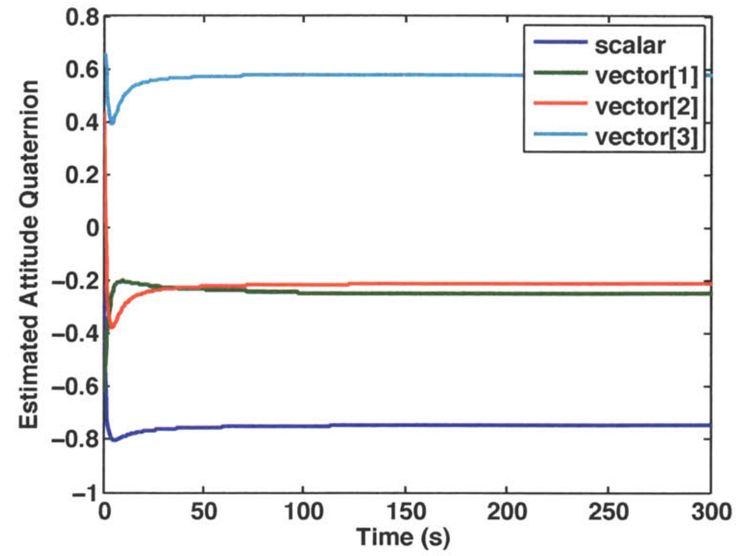

Figure 5-8: Attitude quaternion estimated by the TSat engineering model. The estimated attitude converges to the true attitude shown in Figure 5-9.

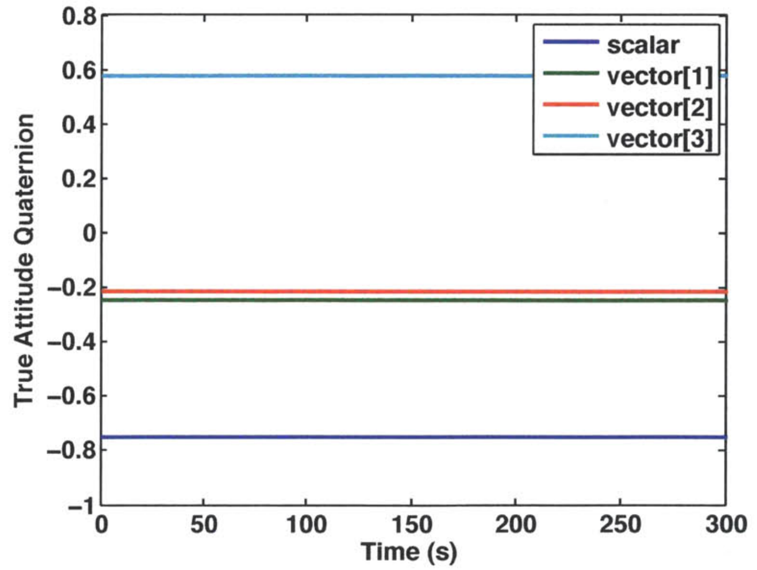

Figure 5-9: True attitude quaternion. The true attitude remains constant throughout the test.

5.2.3 Attempt with Constant Rotation

Another test was attempted with the engineering model set in motion, rotating at a near-constant rate about the SBF-Z axis. This test was not able to show attitude filter convergence because Sun vector measurements were too sparse. Due to the absence of Sun sensors on two of the four faces in the SBF X-Y plane, a Sun vector measurement was only theoretically possible during one quarter of the 360-degree rotation. The observed availability of Sun vector measurements was much less than this theoretical limit. The presence of a full set of Sun sensors in flight will make Sun sensor measurements theoretically possible at all times during orbit day. However, the factors that prevented Sun vector measurements while both engineering model Sun sensors faced the light source may still affect the flight system and must be mitigated.

The TSat engineering model structure has reflective aluminum edges as seen in Figure 5-10. This reflection affects the intensity of the light that is detected by the photodiode Sun sensors, particularly when the edge faces the light source directly. The reflective effects make Sun sensor angle measurements unpredictable, since the relationship between angle and light intensity becomes much more complicated than a simple cosine. In the constant rotation attitude determination test, the reflection was thought to prevent the Sun sensors from reaching their detection threshold at most angles of rotation. In future testing, the aluminum structural edges may be masked with a light absorbent coating to mitigate the reflective effects and produce more accurate Sun sensor measurements.

In addition to mitigating reflective effects of the structural frame, the attitude filter could be adjusted to be more robust to situations where Sun vector measurements are sparse. In its current configuration, the Unscented Kalman Filter only performs a measurement update when there is an available attitude quaternion measurement. The result is that constantly arriving gyro data is not used, except for at the comparatively few times that an attitude measurement is available. In flight, an attitude measurement will nearly always be available, except during eclipse. For the engineering model, however, attitude measurements are hardly ever available due to missing Sun sensors. In the event of a sensor failure, similar factors

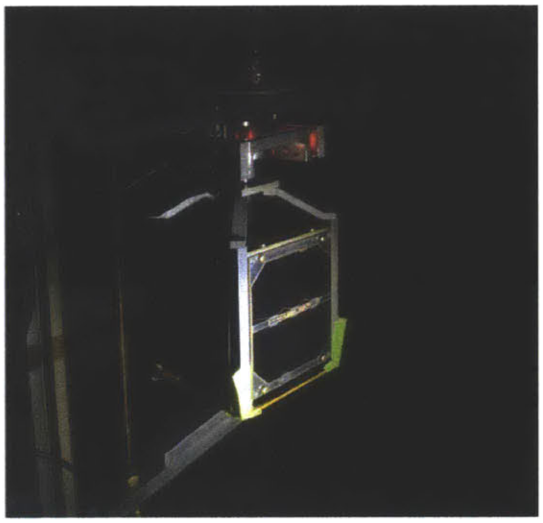

Figure 5-10: Reflection of light by the aluminum edges of the TSat engineering model structure. Reflections are thought to cause inaccuracies in Sun sensor measurements.

could arise in flight as well.

Dynamic replacement, an alternative UKF formulation described in Chapter 4 for the purpose of handling attitude determination during eclipse, also offers a solution to the problem of sparse Sun sensor data. The dynamic replacement UKF formulation is described in more detail in the future work section of Chapter 6. Since it relies more heavily on the gyro output, it may more easily mitigate situations where attitude sensor measurements are sparse.

5.3 Future Testing

Several future tests are planned using the TSat 1-DOF levitation testbed in the MIT SPL Astrovac vacuum chamber to validate the performance of TSat attitude determination and control hardware and software as an integrated system. These tests are representative of some of the maneuvers that will be demonstrated in flight. The capabilities to be demonstrated in the planned tests are applicable to many satellite missions and on-orbit scenarios.

5.3.1 Rate Nulling

The first of the planned tests is a body rate nulling test. The TSat engineering model will be initialized at an arbitrary constant rate of rotation about the SBF-Z axis. A commanded attitude quaternion and commanded body rates of zero will be sent to the flight software. Normal operation of the attitude determination and control system will proceed to remove the initial rotational rate, and position the engineering model with zero body rates in the commanded attitude. This test is similar to the detumble phase of a mission, but is also applicable to any part of the mission where a stable attitude is needed.

5.3.2 Angle Command Slew

The second planned test is a slew maneuver through a commanded angle. The TSat engineering model will be initialized with zero body rates and an arbitrary attitude. An attitude quaternion command corresponding to some angle of rotation about the SBF-Z axis will be sent to the flight software, and the engineering model will proceed to rotate to the commanded quaternion. This maneuver might be applicable in observation missions to turn a camera toward a point of interest, or in missions requiring rendezvous and docking.

5.3.3 Rate Command Slew

The third test planned is a slew maneuver at a commanded rate. The TSat engineering model will be initialized at an arbitrary attitude with zero body rates. A rate command about the SBF-Z axis, and a set of attitude commands corresponding to that rate will be sent to the flight software. The engineering model will slew about the SBF-Z axis at the commanded rate. This type of maneuver might be used for tracking of a ground station with a directional antenna.

5.3.4 Sun Tracking

The fouth planned test consists of Sun tracking. In this test, the Sun source will be slowly moved in a circular path around the TSat engineering model. The engineering model will be commanded to keep one face toward the Sun source, and will rotate as necessary to keep the face pointed directly at the Sun. This capability is not yet implemented in flight software, but might be very useful in a spacecraft safe mode to ensure adequate power generation.

Chapter 6

Future Work and Conclusion

Significant progress has been made toward a flight-ready attitude determination and control system using electrospray microthrusters as attitude actuators. In order to fully demonstrate the ability and advantages of electrospray thrusters, work must be continued in several areas. Improvements to attitude determination and control software are necessary to make the system more robust to the unknowns of the space environment and to handle hardware anomalies. Small hardware improvements are needed to help produce better quality sensor data and thus more accurate attitude determination performance. Improvements to modeling and simulation are needed to make ADCS analysis more credible, and further engineering model testing is necessary to demonstrate the performance of the complete attitude determination and control system on the ground. Finally, a flight demonstration must be carried out to definitively prove the effectiveness of electrospray thruster actuated attitude determination and control for CubeSats.

6.1 Software Improvements

Some features of the TSat attitude determination and control software have not yet been implemented, and several shortcomings of the current software have been discovered through simulation and testing. As discussed in the introduction of Chapter 3, the current implementation of the ADCS software ignores edge cases and hardware failures. These cases must be addressed in order to complete a robust ADCS system. Additionally, GPS data processing and the handling of disturbances by sensor processing algorithms must be implemented.

In testing of the attitude determination system on the TSat 1-DOF levitation testbed, shortcomings of the Unscented Kalman Filter formulation were exposed when Sun sensor data was sparse. In simulation, thruster command chatter resulting in potentially unnecessary use of propellant was observed. Addressing these shortcomings of ADCS software is an important step toward a successful and robust attitude determination and control system for flight.

6.1.1 Handling of Anomalies

A key characteristic of a robust attitude determination and control system is the ability to perform successfully in situations that fall outside the nominal operation of the spacecraft. Such anomalies might include the failure of an attitude determination sensor or an attitude actuator. The anomaly might also be the result of a failure of another spacecraft component that imposes limitations on the ADCS system, such as a solar panel failure limiting the available power for thrusters. Anomalies caused by the failure of attitude determination and control hardware components will likely have the greatest impact on the ADCS system. ADCS software should be updated to detect these anomalies, and to take an appropriate action based on the anomaly detected.

Checks for a failed Sun sensor should be added to the Sun sensor processing algorithm. Failure of a Sun sensor could be detected by comparing the output of sensors on opposite faces of the spacecraft, to ensure that both do not output a nonzero reading at the same time. If a Sun sensor failure is detected, measurements taken using the failed sensor should be ignored until the spacecraft is commanded otherwise following analysis of the situation on the ground. In addition to detecting failure of an individual Sun sensor, the Sun sensor processing algorithm should cross-check computed Sun vectors with previous Sun vector measurements. If the difference between the current and previous measurements is too large, the current measurement should be ignored. This will prevent large jumps in the measured attitude as a result of a noisy Sun vector measurement. In a similar manner, magnetometer measurements should be compared with previous magnetometer measurements and discarded if the difference is too large. This will prevent transient magnetic fields within the spacecraft from affecting the accuracy of the quaternion measurements used by the attitude filter.

Gyro failures should be detected in the gyro processing algorithm by comparing measured body rates with the body rate estimates computed by the attitude filter in the previous control cycle, as well as with the rates measured by the redundant gyros. If the redundant gyro measurement and the estimated body rate are consistent, but the primary gyro measurement is not, the redundant gyro should be used in place of the primary gyro until the spacecraft is commanded otherwise. In this situation, the original primary gyro would fill the role of the redundant gyro.

A new algorithm should be added to detect failure of a thruster or PPU channel. The thruster failure detection algorithm should compare the predicted angular acceleration computed by thrust allocation with the numerical derivative of the spacecraft body rates estimated by the attitude determination filter. In the event of a discrepancy, the failed thruster should be removed from the list of available control moments, so that it will not be used by thrust allocation until the spacecraft is commanded otherwise following analysis of the situation on the ground.

6.1.2 Sensor Processing

During nominal operation, characteristics inherent to the attitude determination sensors, as well as factors arising from integration of the sensors with the rest of the spacecraft will cause inaccuracies in sensor data. TSat sensor procesing should be improved to remove as many of the effects of these internal and external factors as possible.

The Sun sensors, gyros, and magnetometer will all be subject to thermal variations. The Sun sensors and gyros will be most affected since they are on the external faces of the spacecraft, while the temperature of the magnetometer may not vary greatly since it is mounted on an electronics board inside the spacecraft. In order to properly correct for the effects of temperature on gyro and Sun sensor readings, TSat sensor processing should use data from temperature sensors on the solar panels. The behavior of the Sun sensors and gyros as a function of temperature is provided by their manufacturers and must be verified through testing in a thermal chamber.

During preliminary testing of the TSat engineering model, the behavior of the Sun sensors was irregular. The output signals appeared highly quantized, and showed a low signal to noise ratio when the angle between the light source and the outward normal to the sensor was large. The resolution of the signal was not high enough to successfully measure angles more than approximately 30 degrees from the outward normal. When the engineering model was set in motion with a rotational rate of approximately 45 degrees per second, the Sun sensors were unable to detect the light source. The irregularity in Sun sensor behavior was attributed in part to nonuniformity and the lack of intensity of the LED flashlight used as a light source. However, the observed irregularity may also have been a result of reflections or structural dynamics that will be present in flight as well as in testbed testing. Future testing should use a light source more representative of the Sun, and should include improvements to the TSat engineering model structure to reduce reflection, as discussed in section 6.2. Once the behavior of the Sun sensors in the presence of the new light source is characterized, thresholds should be established in Sun sensor processing so that noise and stray reflected light will be ignored.

Gyro bias is ignored in the current implementation of TSat ADCS flight software. The bias of flight gyros should be measured prior to flight, and gyro processing should be updated to take this bias into account. In addition, gyro processing may be modified to estimate the gyro bias by means of a Kalman Filter, or by comparison with the redundant gyros in order to maintain a current bias value over the course of the mission. It is possible that gyro bias may also be included as a state in the main attitude determination filter if the filter is updated to use dynamic model replacement as recommended in Section 6.1.4.

Data from the magnetometer will potentially be altered by magnetic fields produced by other spacecraft components. A check should be added to sensor processing to ensure that magnetometer data is not used during operation of any components that significantly affect the credibility of the data. Components that have a significant effect on the magnetic field inside the spacecraft should be identified through engineering model testing.

6.1.3 GPS Processing

The attitude determination and control software design of Chapter 3 assumes that the inertial position of the spacecraft will always be known. The Surrey GPS receiver described in Section 2.1.4 will provide the position and velocity of the spacecraft to the flight software. This information will arrive at irregular intervals as it becomes available, and will be in an Earth-Centric-Earth-Fixed (ECEF) coordinate frame that is fixed to the rotating Earth. The GPS processing algorithm must translate the GPS position and velocity from the ECEF reference frame to the TSat ECI frame. GPS processing should use the position and velocity data to update the state of a Runge-Kutta propagator that will propagate the spacecraft position and velocity forward in time until the next GPS update. Attitude determination algorithms should use the propagated ECI position values as inputs where needed.

6.1.4 Attitude Determination Filter

As discussed in Chapter 5, attitude filter convergence was not achieved by the TSat engineering model when it was set in motion at a nearly constant rate of rotation on the 1-DOF levitation testbed. The failure to achieve attitude filter convergence was attributed partly to the sparsity of Sun sensor measurements. However, weaknesses of the attitude filter formulation were also recognized. First, the UKF in its current formulation is unable to perform a measurement update except when a valid attitude quaternion measurement is available. All of the constantly arriving gyro data is thus ignored while the filter waits for a Sun sensor measurement. Using all of the gyro data would reduce drift of the filter state away from the true attitude and body rates. Second, the UKF is highly dependent on a model of the physical spacecraft dynamics. The model is imperfect, especially in the testbed environment where air resistance and magnetic forces are present that are not accounted for. Model imperfections inhibit the convergence of the UKF.

A promising solution to both of the major weaknesses of the current attitude filter formulation is the use of dynamic model replacement. Discussed in detail by [19] and [8], dynamic model replacement uses the error state Kalman Filter formulation instead of the total state formulation. This means that instead of estimating the attitude, the filter estimates the attitude error. The "measurements" that are inputs to the filter are the difference between the measured attitude and an attitude estimate maintained by integrating the gyro rates. Use of the error state formulation with dynamic model replacement removes the reliance on modeling of the high frequency dynamics of the spacecraft system, and instead relies on the comparatively low frequency dynamics of the gyro biases. This is a common approach for spacecraft attitude determination systems. The UKF implemented by TSat attitude determination software should be updated to use an error state formulation with dynamic model replacement in order to allow filter convergence despite the sparsity of Sun sensor measurements and the presence of unmodeled spacecraft dynamics. The new UKF formulation will make attitude determination more robust, both in testbed testing and in flight.

6.1.5 Attitude Control Law

In simulation, thruster commands showed a noisy behavior of firing for very short durations, visible in Figure 4-21. The noisy firing behavior, referred to as "chatter," is a result of the control law attempting to maintain zero body rate and attitude errors while the attitude estimate drifts due to UKF dynamics. The controller believes that the attitude and body rates are changing at a higher frequency than they truly are, and thus corrects for these nonphysical dynamics only for the attitude estimate to drift back in the opposite direction. Chatter is undesirable because it uses fuel without increasing attitude accuracy. In order to remove the chatter, a controller deadband should be placed in the thrust allocation algorithm. If the commanded torque is less than the deadband threshold, no thrusters will be fired. This will allow the attitude estimate to drift to the order of magnitude of the attitude determination error and will avoid wasting propellant.

6.2 Hardware Improvements

As discussed in Section 6.1.2, the behavior of Sun sensors was unpredictable when the TSat engineering model was set in motion and rotated past the light source. One possible cause of the erratic Sun sensor data is reflection of light by the aluminum edges of the TSat structure as shown in Figure 5-10. This reflection could be a factor in flight, as well as in testbed testing. In attempt to reduce reflection and increase the quality of Sun sensor readings, the edges of the TSat structure should be covered with a light-absorbent coating. The light absorbing edges will allow more light to reach the Sun sensors instead of scattering into the environment when the edges of the structure point toward the light source.

6.3 Model Improvements

Model improvements that increase the fidelity of the TSATsim simulation would give more credibility to the analysis performed, and increase confidence that the attitude determination and control system will operate successfuly in flight. Currently, the simulation accurately models the behavior of the spacecraft and the environment, but ignores higher order dynamics. The eventual goal of the simulation is to mimic the behavior of each of the spacecraft components and the effects of environmental factors to the highest degree of accuracy possible. The simulation is structured so that models of spacecraft components and environmental phenomena may be upgraded individually in a way that is noninvasive to the rest of the simulation.

6.3.1 Sensor Models

Simulation of imperfections in sensor data should be added to each of the TSATsim sensor models. In the current implementation of TSATsim, all of the sensor models add random noise to sensor measurements. In addition, sensor biases and drift should be modeled for the Sun sensors, gyros, and magnetometer. Effects of the environment such as thermal variation of the Sun sensor outputs and the effects of other spacecraft components on the magnetic field experienced by the magnetometer should also be added to the attitude determination sensor models.

6.3.2 Actuator Models

In the current implementation of TSATsim, the simulated iEPS electrospray thrusters produce thrust exactly as commanded without latency or noise. Thruster command latency, transients following changes in the commanded thrust level, and uncertainty in the actual thrust force produced must be added to the thruster model. In addition, the behavior of the PPU and its interface with the flight computer should be modeled once the PPU design is finalized and tested. When more complete thruster performance data is available, thruster efficiencies and degradation over time should be added to the simulation as well.

6.3.3 Environmental Dynamics

The environmental dynamics model in the current implementation of TSATsim includes the forces imparted on the spacecraft by gravity and aerodynamic drag. The gravity model uses J_2 zonal harmonic terms. The fidelity of the environmental model should be increased by upgrading the gravity model to use J_4 or higher terms, and by simulating moments imparted on the spacecraft by gravity, aerodynamics, and the geomagnetic field. Simulation of forces and moments produced by solar radiation pressure should also be added.

6.4 Further Testing

Completion of TSat attitude determination and control system testing is the final step toward demonstrating the effectiveness and feasibility of electrospray thrusters as attitude control actuators for CubeSats. First, the performance of the TSat ADCS system in 1 axis must be verified on the 1-DOF levitation testbed described in Chapter 5. After the system has been debugged and verified, a flight demonstration will show that TSat is capable of performing a variety of attitude maneuvers in space.

6.4.1 1-DOF Levitation Testbed

Future testing on the TSat 1-DOF Levitation Testbed is discussed at the end of Chapter 5. First, the attitude determination system must be re-tested outside of the Astrovac vacuum chamber with a more uniform Sun source than the source used for the tests described in Chapter 5. The edges of the TSat structure should be covered with a light-absorbent coating, and the attitude filter should be revised to incorporate dynamic model replacement as discussed in Section 6.1.4. Planned attitude control tests inside the Astrovac vacuum chamber include a rate nulling test, a rate command slew, an angle command slew, and a Sun tracking test. The planned attitude control tests are summarized in Table 6.1.

Test Name	Description	Application
Rate nulling	Initialize engineering model at an arbitrary rate of rotation. Send a commanded quaternion and zero body rates to the flight software. Engineering model ADCS nulls body rates and holds engineering model in commanded attitude.	Detumble and precision pointing
Rate command slew	Initialize engineering model at an arbitrary attitude angle with zero body rates. Command a rotation about the SBF-Z axis. Engineering model rotates at constant rate.	Sensor scanning and ground station tracking
Angle command slew	Initialize engineering model at an arbitrary attitude angle with zero body rates. Command another attitude angle with zero body rates. Engineering model rotates to the commanded attitude.	Precision pointing and sensor positioning
Sun tracking	Move Sun source along a circular path around the engineering model. Engineering model tracks the Sun, rotating to keep a single solar panel pointed toward it.	Power generation and Sun observation

Table 6.1: Summary of attitude control testing on the 1-DOF Levitation Testbed in the Astrovac vacuum chamber. Tests are designed to resemble common on-orbit maneuvers and to verify the performance of the TSat attitude determination and control system about 1 axis.

6.4.2 Flight Demonstration

The flight demonstration of the TSat electrospray thruster actuated attitude determination and control system is designed to achieve four objectives. The objectives include operation of the electrospray thrusters in space, characterization of thruster performance, precision attitude control, and the ability to produce delta-V or translational acceleration. A series of eight tests will achieve these four objectives. The tests are shown in Figure 6-1 and are numbered in order of increasing "difficulty."

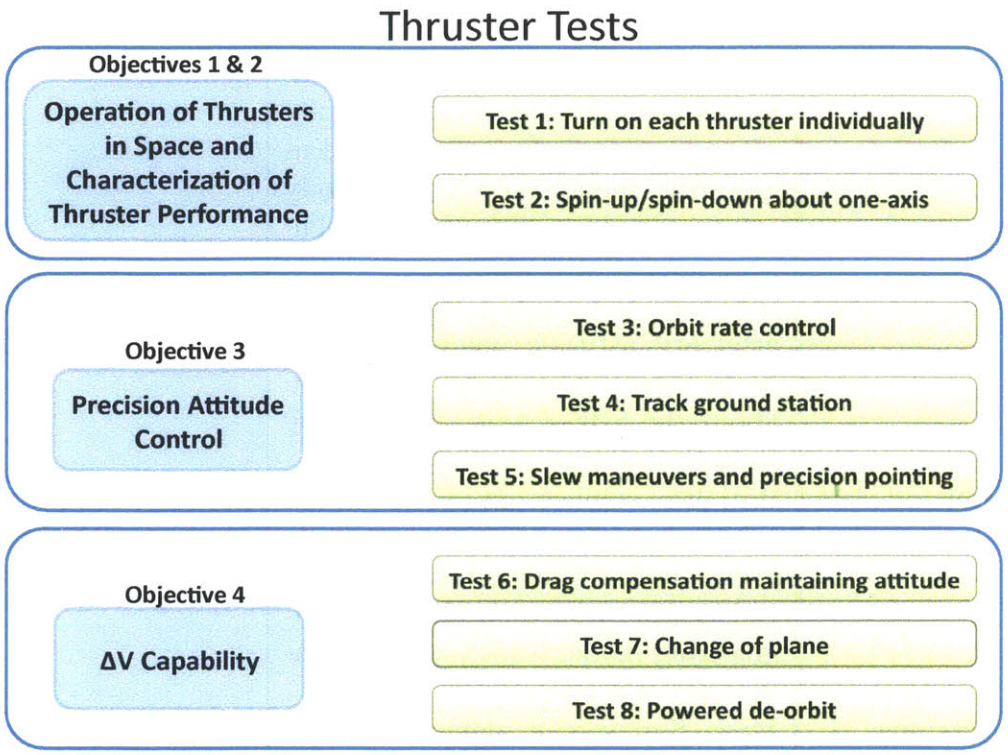
Thruster Tests
Objectives 1 & 2
Operation of Thrusters in Space and Characterization of Thruster Performance
Test 1: Turn on each thruster individually
Test 2: Spin-up/spin-down about one-axis
Objective 3
Precision Attitude Control
Test 3: Orbit rate control
Test 4: Track ground station
Test 5: Slew maneuvers and precision pointing
Objective 4
ΔV Capability
Test 6: Drag compensation maintaining attitude
Test 7: Change of plane
Test 8: Powered de-orbit

Figure 6-1: TSat flight demonstration tests and corresponding objectives. The flight demonstration tests will prove the ability of electrospray thrusters as CubeSat actuators by addressing objectives 3 and 4 on the left of the figure. (credit: Akshata Krishnamurthy)

In some of the earlier tests, thrusters are turned on and off manually, and the TSat attitude control algorithms are not used. These manual tests are classified as "open loop." Later tests incorporate the entire attitude determination and control system and are classified as "closed loop." Table 6.2 gives a description of each test, including whether it is open loop or closed loop.

Test Number	Events	Control Scheme
1	Turn on thruster n for 30 s. After 30 s, turn off and null all body rates using torque coils. Repeat for thrusters n = 0 to 7.	Open loop
2	Spin-up/Spin-down about SBF Y-axis: Turn on thrusters 0 and 1. When a pre-determined rate is reached, turn off thrusters 0 and 1, and turn on thrusters 2 and 3. When rate is close to zero or after maximum time, turn off all thrusters and null all body rates using torque coils.	Open loop
3	Set up and actively maintain orbit rate keeping one face pointed at the ground for an entire orbit.	Closed loop
4	Track ground stations pointing one face towards them for an entire orbit.	Closed loop
5	Point all 6 faces toward the Sun one after the other, stopping to minimize attitude error in each orientation. Ideally completed within 1 orbit day.	Closed loop
6	Compensate for the effects of drag with prograde thrust by operating thrusters on one face using off-modulation to maintain attitude along the velocity vector, over a period of approximately 1 week.	Closed loop
7	Alter orbit inclination.	Closed loop
8	Maintain retrograde pointing and lower orbit semi-major axis by as much as possible, for the length of a battery discharge cycle.	Closed loop

Table 6.2: Description of TSat flight demonstration tests. Tests will be executed in order of increasing "difficulty," beginning with open loop testing to measure the performance of the electrospray thrusters, and continuing with closed loop testing to demonstrate the capabilities of the complete attitude determination and control system.

6.5 Conclusion

Electrospray microthrusters will play a critical role in the future of space exploration. With advantages over reaction wheels and magnetorquers for missions requiring translational control or operation outside of low Earth orbit, electrospray thrusters are an attractive choice of attitude control actuators for CubeSats. Enabled by electrospray thrusters, constellations of CubeSats are capable of many Earth observation, astronomy, and even deep space science missions.

A CubeSat attitude determination and control system was developed using COTS hardware components for attitude determination, and electrospray microthrusters as attitude actuators. Preliminary testing in simulation showed attitude control accuracy on the order of 2-3 degrees. The attitude error was dominated by error in attitude determination, which could be reduced by using custom attitude determination hardware components.

Contributions of this research include a high fidelity simulation framework and a modular CubeSat attitude determination and control software implementation that will serve as a foundation for future development of microthruster actuated CubeSats. A novel thrust allocation algorithm was developed to address the possibility that each thruster might have unique characteristics, and to minimize the amount of precision alignment necessary during CubeSat fabrication. Attitude determination and control software was implemented on an engineering model flight computer, and preliminary testing of the engineering model gathered performance data for COTS sensors and suggested improvements to the preliminary ADCS design. Preliminary simulation results show that using electrospray microthrusters as CubeSat attitude actuators is feasible and effective.

Moving forward, revision of the ADCS design and engineering model testing of attitude control about one axis will produce a flight-ready attitude determination and control system. This work will lead directly to an on-orbit demonstration that will showcase the potential impact of electrospray thrusters in the field of CubeSat attitude control, and on the future of space exploration.

Appendix A

Flight Software Algorithm Pseudocode and Implementation for Analysis

A.1 FSW Master

A.1.1 Pseudocode for *tsat_gnc*

This section contains pseudocode for *tsat_gnc*, the master ADCS routine. An implementation of *tsat_gnc* in C is given in section A.1.2.

1. Read and process data from GNC sensors. (call *gnc_sensor_processing*)

2. Compute an estimate of the spacecraft attitude quaternion and body rates. (call *gnc_attitude_determination*)

3. Run guidance logic that determines the target behavior of the spacecraft. (call *gnc_guidance*)

4. Run the attitude control law. (call *gnc_attitude_cl*)

5. Prepare thruster commands based on the output of the control law. (call *gnc_cmd_prep*)

6. Update the software's knowledge of time and julian date.

A.1.2 TSATsim implementation of *tsat_gnc*

This section lists the TSat GNC analysis simulation implementation of *tsat_gnc*.

```
/* tsat_gnc.cc
   Main Functionality of TSat ADCS Flight Software
   Written for TSat ADCS analysis
   Mark Van de Loo, Spring 2013
*/


#include <stdio.h>

#include "tsat_gnc.h"


int tsat_gnc(fswstate *fsw)
{


  //read sensor data
  gnc_sensor_processing(fsw);

  //run attitude determination
  gnc_attitude_determination(fsw);

  //run guidance
  gnc_guidance(fsw);

  //run attitude control law
  gnc_attitude_cl(fsw);

  //run command preparation
```

```
    gnc_cmd_prep(fsw);


    fsw->time_cntr +=1;
    fsw->time = fsw->time_cntr*fsw->cyc_period;
    fsw->jd = fsw->jd0 + fsw->time_cntr*fsw->cyc_period/86400.0;


    return(0);
}
```

A.2 Sensor Processing

A.2.1 Pseudocode for *gnc_sensor_processing*

- Process sun sensor data. (call *gnc_process_ss*)
- Process gyro data. (call *gnc_process_gyro*)

A.2.2 TSATsim implementation of *gnc_sensor_processing*

This section lists the TSat GNC analysis simulation implementation of *gnc_sensor_processing*.

```
/* gnc_sensor_processing.c
   Algorithm for onboard processing of GNC sensor data. Processes data
   from sun sensors, magnetometer, and gyros.
   Written as part of TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


```

```c
#include <stdio.h>
#include <math.h>

#include "gnc_sensor_processing.h"


int gnc_sensor_processing(fswstate *fsw)
{

  //Magnetometer
  //B_magnetometer assumed to be updated by Nanomind black box for now.

  //Sun Sensors
  gnc_process_ss(fsw);

  //Gyros
  gnc_process_gyro(fsw);


  return(0);
}
```

A.2.3 Inputs and Outputs of *gnc_process_ss*

Table A.1: Inputs to gnc_process_ss

GNC Analysis Variable	Units	Description	Value comes from...
sunsensor_raw	milliVolts	Vector containing the output value of each sun sensor	GomSpace solar panels (rw/process_sensor_data.cc)
ss_thresh	cosine	Threshold value below which a sun sensor is considered not to see the sun	fsw_config

Table A.2: Outputs of gnc_process_ss

GNC Analysis Variable	Units	Description	Value goes to...
valid_sunvec_meas	boolean	=1 if there is a valid sunsensor measurement, =0 otherwise	Attitude Determination
sunvec_meas	unit vector	Direction of the Sun computed based on sensor readings	Attitude Measurement (gnc_TRIAD)

A.2.4 Pseudocode for *gnc_process_ss*

1. Calculate the cosine of the angle measured by each sun sensor. Angles are measured with respect to the outward normal to the sun sensor/solar panel plane.

$$\cos(\beta) = \frac{V}{400.0} \tag{A.1}$$

   Where V is the raw voltage reading from the sun sensor. This uses the value of the "typical" open circuit voltage of 400 milliVolts given by the photodiode datasheet, and assumes a voltage of 0 milliVolts when no light is being absorbed.

2. Determine whether at least one sun sensor gives a nonzero reading. (i.e. not in eclipse) Loop through sun sensors, and if the value of $\cos(\beta)$ calculated in the previous step is above a threshold value, set the *valid_sunvec_meas* flag to 1.

3. If all sensors give a zero reading, then skip all remaining sun-sensor-related steps and set $sunvec_{meas} = 0$. Else, continue with the remaining steps.

4. Calculate the SBF-X component of the sun vector. First compare the $\cos(\beta)$ values of sun sensors 0 and 3. If the reading of sun sensor 3 is greater than the reading of sun sensor 0, sensor 0 should read zero since faces 0 and 3 cannot see the sun simultaneously. The SBF-X component of the measured sun vector is then given by $-\cos(\beta)$, the opposite of the cosine of the angle between the sun and the outward normal to sensor 3. If the reading of sensor 0 is greater than that of sensor 3, then the SBF-X component of the measured sun vector is given by $\cos(\beta)$ of sensor 0. The number labeling of spacecraft faces is shown in figure ??. The number of a sun sensor

corresponds to the face on which it is mounted.

$$\begin{aligned}\text{if: } & \cos(\beta)_3 > \cos(\beta)_0 \\ \text{then: } & sunvec_{meas_x} = -\cos(\beta)_3 \\ \text{else: } & sunvec_{meas_x} = \cos(\beta)_0 \\ \\ \text{if: } & sunvec_{meas_x} < threshold \\ \text{then: } & sunvec_{meas_x} = 0 \end{aligned} \tag{A.2}$$

5. Repeat the previous step with sensor 1 taking the place of sensor 0 and 4 taking the place of 3 to find the SBF-Y component of the measured sun vector.

6. Repeat with sensor 2 taking the place of 0 and 5 taking the place of 3 to find the SBF-Z component.

7. Normalize the measured sun vector.

$$sunvec_{meas} = \frac{sunvec_{meas}}{\|sunvec_{meas}\|} \tag{A.3}$$

8. Check for faults. (TBR)

A.2.5 TSATsim implementation of *gnc_process_ss*

This section lists the TSat GNC analysis simulation implementation of *gnc_process_ss*.

```
/* gnc_sensor_processing.cc
   Algorithm for onboard processing sun sensor data.
   Written as part of TSat ADCS analysis
   Mark Van de Loo, February 2014
*/
```

```c


#include <stdio.h>
#include <math.h>

#include "gnc_process_ss.h"


int gnc_process_ss(fswstate *fsw)
{

  //Sun Sensors
  //convert from sensor output units to a cosine.
  //Value output by sensor in milliVolts.  400 is the "typical" open
  //circuit voltage from the photodiode data sheet. ss_meas is the sun
  //sensor measured value of the cosine
  //between the vector normal to the solar panel containing the sensor
  //(positive outwards) and the vector from the spacecraft cg to the
  //sun.
  double ss_meas [fsw->num_panels];
  int panel;
  for(panel = 0;panel < fsw->num_panels; panel++){
    ss_meas[panel] = (fsw->sunsensor_raw[panel]) / 400.0;
  }

  //Determine whether we can see the sun
  fsw->valid_sunvec_meas = 0;
  for(panel = 0;panel < fsw->num_panels; panel++){
    if(ss_meas[panel]>fsw->ss_thresh){
      fsw->valid_sunvec_meas = 1;
      break;
    }
  }

  //calculate sun vector from sun sensor cosine measurements:
  if(fsw->valid_sunvec_meas){
  //SBF-X component of Sun Vector
  if(ss_meas[3] > ss_meas[0]){
    fsw->sunvec_meas[0] = -ss_meas[3];
  }
  else{
    fsw->sunvec_meas[0] = ss_meas[0];
  }
  if(fabs(fsw->sunvec_meas[0]) < fsw->ss_thresh){
    fsw->sunvec_meas[0] = 0.0;
  }
  //TODO: add FAULT if panels 0 and 3 see sun at the same time


  //SBF-Y component of Sun Vector
  if(ss_meas[4] > ss_meas[1]){
    fsw->sunvec_meas[1] = -ss_meas[4];
  }
  else{
    fsw->sunvec_meas[1] = ss_meas[1];
  }
  if(fabs(fsw->sunvec_meas[1]) < fsw->ss_thresh){
    fsw->sunvec_meas[1] = 0.0;
  }
  //TODO: add FAULT if panels 1 and 4 see sun at the same time


  //SBF-Z component of Sun Vector
  if(ss_meas[5] > ss_meas[2]){
    fsw->sunvec_meas[2] = -ss_meas[5];
  }
  else{
    fsw->sunvec_meas[2] = ss_meas[2];
```

```
}
if(fabs(fsw->sunvec_meas[2]) < fsw->ss_thresh){
  fsw->sunvec_meas[2] = 0.0;
}
//TODO: add FAULT if panels 2 and 5 see sun at the same time


//normalize sunvec_meas
double sm_mag = sqrt(pow(fsw->sunvec_meas[0],2) + pow(fsw->sunvec_meas[1],2) +
                     pow(fsw->sunvec_meas[2],2));
if(sm_mag > 0.3){
  int i;
  for(i = 0;i<3;i++){
    fsw->sunvec_meas[i] = fsw->sunvec_meas[i]/sm_mag;
  }
}
else{
  fsw->valid_sunvec_meas = 0;
  int i2;
  for(i2 = 0;i2<3;i2++){
    fsw->sunvec_meas[i2] = 0.0;
  }
}

}
else{
  int i;
  for(i = 0;i<3;i++){
    fsw->sunvec_meas[i] = 0.0;
  }
}

// TODO:
//Add check for single sun sensor failure by comparing 2 sensor soln
```

```
    // with 3 sensor soln.



    return(0);
}
```

A.2.6 Inputs and Outputs of *gnc_process_gyro*

Table A.3: Inputs to gnc_process_gyro

GNC Analysis Variable	Units	Description	Value comes from...
gy_gyro_out	rad/sec	Vector containing the contents of the "gyro_out" register for each gyro	GomSpace solar panels (rw/process_sensor_data.cc)

Table A.4: Outputs of gnc_process_gyro

GNC Analysis Variable	Units	Description	Value goes to...
body_rates (TBR)	rad/sec	Spacecraft body rates in SBF computed from gyro readings (TBR)	Attitude Controller (TBU)

A.2.7 Pseudocode for *gnc_process_gyro*

1. Transfer the contents of the *gy_gyro_out* registers into the *body_rates* variable. Eventually this may be updated with some sort of filter. (TBR)

$$body\_rates(axis) = gy\_gyro\_out(axis) \tag{A.4}$$

A.2.8 TSATsim implementation of *gnc_process_gyro*

This section lists the TSat GNC analysis simulation implementation of *gnc_process_gyro*.

```
/* gnc_process_gyro.cc
   Algorithm for onboard processing of gyro measurements.
   Written as part of TSat ADCS analysis
   Mark Van de Loo, February 2014
*/


#include <stdio.h>
#include <math.h>
```

```

#include "gnc_process_gyro.h"


int gnc_process_gyro(fswstate *fsw)
{


  //Gyros
  int ind;
  for(ind = 0; ind < 3; ind++){
    fsw->body_rates_meas[ind] = fsw->gy_gyro_out[ind];
  }




  return(0);
}
```

A.3 Attitude Determination

A.3.1 Pseudocode for *gnc_attitude_determination*

This pseudocode corresponds to an implemention of gnc_attitude_determination suitable for analysis and engineering model testbed testing. It will have to be adjusted for flight.

1. **If** there is a valid sunvector measurement available from sensor processing this cycle: call *gnc_TRIAD*.

   Else: Set the measured attitude quaternion *att_quat_meas* to a zero rotation.

2. **If** *use_attitude_kf* $==$ *true*: call *gnc_attitude_kf*.

Else: Set the attitude quaternion and body rates estimates equal to the measured attitude quaternion and body rates.

A.3.2 TSATsim implementation of *gnc_attitude_determination*

This section lists the TSat GNC analysis simulation implementation of the algorithm. The current implementation of this wrapper is for analysis and engineering model testbed testing. It will have to be adjusted for flight.

```
/* gnc_attitude_determination.cc
   Wrapper for attitude determination tasks. Includes creating an
   attitude measurement, kalman filtering, related propagation, etc.
   Written as part of TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_attitude_determination.h"


int gnc_attitude_determination(fswstate *fsw)
{


  //use sensor data to compute attitude measurement
  if(fsw->valid_sunvec_meas){
    gnc_TRIAD(fsw);
  }
  else{
    int quatind;
```

```
      for(quatind = 0;quatind<3;quatind++){
        fsw->att_quat_meas[quatind] = 0.0;
      }
      fsw->att_quat_meas[3] = 1.0;
    }

    if(fsw->use_attitude_kf){
      //use KF to estimate attitude quaternion and rates
      gnc_attitude_kf(fsw);
    }
    else{
      int quatind1;
      for(quatind1 = 0;quatind1<4;quatind1++){
        fsw->att_quat[quatind1] = fsw->att_quat_meas[quatind1];
      }
      int wind;
      for(wind = 0;wind<3;wind++){
        fsw->body_rates[wind] = fsw->body_rates_meas[wind];
      }
    }


    return(0);
}
```

A.3.3 Inputs and Outputs of *gnc_TRIAD*

Table A.5: Inputs to gnc_TRIAD

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
B_magnetometer	Gauss	Geomagnetic field vector as measured by the 3-axis magnetometer in Spacecraft Body Frame	NanoMind (rw/process_sensor_data.cc)
sunvec_meas	unit vector	Direction of the Sun as computed from sun sensor measurements	gnc_sensor_processing
jd	days	Current Julian date	tsat_gnc
jd_igrfdata	days	Julian date corresponding to the timestamp of the IGRF data used in the spherical harmonic magnetic field model	fswparam

Continued on next page

Table A.5: Algorithm Inputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
pos_eci	meters	Position vector of spacecraft in ECI	GPS/propagator (TBU)
igrf_g	n/a	IGRF G coefficient for spherical harmonic magnetic field model	fswigrf
igrf_h	n/a	IGRF H coefficient for spherical harmonic magnetic field model	fswigrf
igrf_gdot	n/a	Time rate of change of IGRF G coefficient for spherical harmonic magnetic field model	fswigrf

Continued on next page

Table A.5: Algorithm Inputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
igrf_hdot	n/a	Time rate of change of IGRF H coefficient for spherical harmonic magnetic field model	fswigrf
PI	n/a	3.14159265359	initialize_fsw

Table A.6: Outputs of gnc_TRIAD

GNC Analysis Variable	**Units**	**Description**	**Value goes to...**
B_actual	nTesla	Geomagnetic field vector calculated from IGRF spherical harmonic model in ECI frame	Telemetry
sunvec_actual	unit vector	Direction of the Sun as computed from julian date	Telemetry

Continued on next page

Table A.6: Algorithm Outputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value goes to...**
att_quat (TBR)	quaternion	Spacecraft attitude quaternion. Transforms vectors from ECI to SBF.	Attitude Controller (TBU)

A.3.4 Pseudocode for *gnc_TRIAD*

1. Define the order of the spherical harmonic geomagnetic field model to be $k = 6$ (6th order).

2. Define the earth radius used by the IGRF as $a = 6371.2e3$. This is a parameter that was chosen when this particular model was constructed.

3. Define $difyear$ to be the time elapsed in units of years between the timestamp of the IGRF data and the current time:

$$(difyear) = \frac{(jd) - (jd\_igrfdata)}{365.25} \tag{A.5}$$

   Where jd is the current julian date (decimal value) and $jd\_igrfdata$ is the julian date of the IGRF data timestamp.

4. Calculate T, the time elapsed in units of julian centuries since the J2000 epoch (Vallado p.194):

$$T = \frac{(jd) - 2451545.0}{36525.0} \tag{A.6}$$

5. Calculate *gmst*, the current greenwich mean sidereal time in seconds (Vallado p. 194):

$$(gmst) = 67310.54841 + ((876600.0)(3600.0) + 8640184.812866)T + 0.093104T^2 - 6.2 \times 10^{-6}T^3 \tag{A.7}$$

6. Reduce *gmst* to fall between 0 sec and 86400 sec (per Vallado):

$$(gmst) = (gmst) \bmod 86400 \tag{A.8}$$

7. Convert *gmst* to radians. Define α_g to be the Greenwich mean sidereal time in radians:

$$\alpha_g = (gmst)\frac{1}{240}\frac{\pi}{180} \tag{A.9}$$

8. Let x, y, and z be the three components of the spacecraft position vector in ECI frame.

9. Let r be the magnitude of the ECI spacecraft position vector i.e. the distance from the spacecraft CG to the center of the earth.

10. Calculate the geocentric latitude (declination) of the spacecraft:

$$\delta = atan2(z, \sqrt{x^2 + y^2}) \tag{A.10}$$

11. Calculate the sidereal time (right ascension) of the spacecraft:

$$\alpha = atan2(y, x) \tag{A.11}$$

12. Define the coelevation.

$$\theta = \frac{\pi}{2} - \delta \tag{A.12}$$

13. Calculate the longitude of the spacecraft:

$$\phi = \alpha - \alpha_g \tag{A.13}$$

14. Calculate the factors $S_{n,m}$ to be used in the Schmidt functions according to Wertz eqn. H-7.

$$\begin{aligned} S_{0,0} &= 1 \\ S_{n,0} &= S_{n-1,0}[\frac{2n-1}{n}] \\ S_{n,m} &= S_{n,m-1}\sqrt{\frac{(n-m+1)(\delta_m^1+1)}{n+m}} \end{aligned} \tag{A.14}$$

Where δ_j^i is the Kronecker delta: $\delta_m^1 = 1$ if $i = j$, and 0 otherwise.

15. Combine the factors $S_{n,m}$ with the Gaussian coefficients as described by Wertz eqn. H-6.

$$\begin{aligned} g^{n,m} &= S_{n,m} g_n^m \\ h^{n,m} &= S_{n,m} h_n^m \end{aligned} \tag{A.15}$$

16. Calculate the $K^{n,m}$ coefficients according to Wertz eqn. H-9.

$$\begin{aligned} K^{n,m} &= \frac{(n-1)^2 - m^2}{(2n-1)(2n-3)} && \text{for } n > 1 \\ K^{n,m} &= 0 && \text{for } n = 1 \end{aligned} \tag{A.16}$$

17. Calculate the Gauss functions, $P^{n,m}$ according to Wertz eqn. H-8.

$$\begin{aligned} P^{0,0} &= 1 \\ P^{n,n} &= sin(\theta)P^{n-1,n-1} \\ P^{n,m} &= cos(\theta)P^{n-1,m} - K^{n,m}P^{n-2,m} \end{aligned} \tag{A.17}$$

18. Calculate the partial derivatives of $P^{n,m}$ with respect to θ as shown in Wertz eqn. H-10.

$$\begin{aligned} \frac{\partial P^{0,0}}{\partial \theta} &= 0 \\ \frac{\partial P^{0,0}}{\partial \theta} &= (\sin\theta)\frac{\partial P^{n-1,n-1}}{\partial \theta} + (\cos\theta)P^{n-1,n-1} \qquad \text{for } n \geq 1 \\ \frac{\partial P^{0,0}}{\partial \theta} &= (\cos\theta)\frac{\partial P^{n-1,m}}{\partial \theta} - (\sin\theta)P^{n-1,m} - K^{n,m}\frac{\partial P^{n-2,m}}{\partial \theta} \end{aligned} \tag{A.18}$$

19. Note from Wertz (eqn. H-11) that the number of sine and cosine evaluations can be reduced by using:

$$\begin{aligned} \cos m\theta &= \cos((m-1)\phi + \phi) \\ &= \cos((m-1)\phi)\cos\phi - \sin\phi\sin((m-1)\phi) \end{aligned} \tag{A.19}$$

20. Calculate the B field according to Wertz eqn. H-12. B_r is radial outward positive, B_θ is the coelevation component (South positive), and B_ϕ is the asimuthal component

with East positive.

$$
\begin{aligned}
B_r &= -\frac{\partial V}{\partial r} = \sum_{n=1}^{k} \frac{a}{r}^{n+2}(n+1)\sum_{m=0}^{n}(g^{n,m}\cos m\phi + h^{n,m}\sin m\phi)P^{n,m}(\theta) \\
B_\theta &= -\frac{1}{r}\frac{\partial V}{\partial \theta} = -\sum_{n=1}^{k} \frac{a}{r}^{n+2}\sum_{m=0}^{n}(g^{n,m}\cos m\phi + h^{n,m}\sin m\phi)\frac{\partial P^{n,m}(\theta)}{\partial \theta} \\
B_\phi &= -\frac{-1}{r\sin\theta}\frac{\partial V}{\partial \phi} = \frac{-1}{\sin\theta}\sum_{n=1}^{k} \frac{a}{r}^{n+2}\sum_{m=0}^{n} m(-g^{n,m}\sin m\phi + h^{n,m}\cos m\phi)P^{n,m}(\theta)
\end{aligned}
\tag{A.20}
$$

21. Convert to a less absurd coordinate system i.e. ECI (Earth Centric Inertial). (See Wertz eqn. H-14)

$$
\begin{aligned}
B_x &= (B_r\cos\delta + B_\theta\sin\delta)\cos\alpha - B_\phi\sin\alpha \\
B_y &= (B_r\cos\delta + B_\theta\sin\delta)\sin\alpha + B_\phi\cos\alpha \\
B_z &= (B_r\sin\delta - B_\theta\cos\delta)
\end{aligned}
\tag{A.21}
$$

22. Construct "actual" magnetic field unit vector in ECI from B calculated in the previous step.

$$B_{actual} = B/||B|| \tag{A.22}$$

23. Contstruct "measured" magnetic field unit vector in SBF from magnetometer output.

$$B_{meas} = B_{magnetometer}/||B_{magnetometer}|| \tag{A.23}$$

24. Caluclate the mean longitude of the sun in mean of date frame. (Vallado p. 297)

$$\lambda_M = 280.460 + 36,000.771 T_{UT1} \tag{A.24}$$

25. Calculate the mean anomaly for the sun using approximation T_{TDB} T_{UT1}. Units are degrees.

$$M = 357.5277233 + 35,999.05034T_{TDB}$$
$$M = \lambda_M \bmod 360 \tag{A.25}$$

26. Approximate the ecliptic longitude according to Vallado p.280.

$$\lambda_{ecliptic} = \lambda_M + 1.914666471\sin(M) + 0.019994643\sin(2M) \tag{A.26}$$

(Approximate ecliptic latitude to be 0.)

27. Calculate the distance from the earth to the sun in meters according to Vallado p. 281.

$$r_{mag} = 1.000140612 - 0.016708617\cos(M) - 0.000139589\cos(2M)$$
$$r_{mag} = r_{mag} * 149597870700 \tag{A.27}$$

28. Calculate the (approximate) obliquity of the ecliptic

$$\epsilon = 23.439291 - 0.0130042T_{TBD} \tag{A.28}$$

29. Compute the position of the sun in ECI frame. Units should be meters.

$$r_{sun1} = r_{mag}\cos(\lambda_{ecliptic})$$
$$r_{sun2} = r_{mag}\cos(\epsilon)\sin(\lambda_{ecliptic})$$
$$r_{sun3} = r_{mag}\sin(\epsilon)\sin(\lambda_{ecliptic}) \tag{A.29}$$

30. Compute "actual" ECI unit vector in the direction of the sun.

$$r_{sun_{actual}} = \frac{r_{sun}}{||r_{sun}||} \tag{A.30}$$

31. Let the vector r_1 represent the "actual" magnetic field unit vector.

$$r_1 = B_{actual} \tag{A.31}$$

32. Define the vector r_2 as the cross product of the "actual" magnetic field unit vector and the "actual" sun unit vector.

$$r_2 = B_{actual} \times r_{sun_{actual}} \tag{A.32}$$

33. Define the vector r_3 as the cross product of r_1 and r_2.

$$r_3 = r_1 \times r_2 \tag{A.33}$$

34. Define the vector s_1 as the "measured" magnetic field unit vector.

$$s_1 = B_{meas} \tag{A.34}$$

35. Define the vector s_2 as the cross product of the "measured" magnetic field unit vector and the "measured" sun unit vector.

$$s_2 = B_{meas} \times r_{sun_{meas}} \tag{A.35}$$

36. Define the vector s_3 as the cross product of s_1 and s_2.

$$s_3 = s_1 \times s_2 \tag{A.36}$$

37. Construct ECI to SBF transformation matrix A. (Shuster p. 115)

$$A = [s_1\ s_2\ s_3][r_1\ r_2\ r_3]^T \tag{A.37}$$

38. Construct ECI to SBF attitude quaternion q from transformation matrix A.

$$\begin{aligned}
q_4 &= \frac{1}{2}(1 + A_{11} + A_{22} + A_{33})^{\frac{1}{2}}(\text{scalar}) \\
q_1 &= \frac{1}{4q_4}(A_{23} - A_{32}) \\
q_2 &= \frac{1}{4q_4}(A_{31} - A_{13}) \\
q_3 &= \frac{1}{4q_4}(A_{12} - A_{21})
\end{aligned} \tag{A.38}$$

39. Normalize and properize the attitude quaternion. This is the "measured" attitude.

$$q = \frac{q}{\|q\|}$$

$$\text{If } q_4(\text{scalar}) < 0$$

$$q = -q$$

$$q_{meas} = q \tag{A.39}$$

A.3.5 TSATsim Implementation of *gnc_TRIAD*

This section lists the TSat GNC analysis simulation implementation of *gnc_TRIAD*.

```
/* gnc_TRIAD.cc
   Implementation of the TRIAD algorithm for computing a measured
   spacecraft attitude from two measured vectors. In this case the
   measured vectors are the geomagnetic field and the direction of the sun.
   Implemented for TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_TRIAD.h"
#include "custom_math.h"


int gnc_TRIAD(fswstate *fsw)
{

  //Compute "true" magnetic field vector in ECI

  int k = 6; //order of model
  double a = 6371.2e3; //IGRF Earth Radius (not the same as mean earth radius)
  double difyear; //number of years between IGRF data timestamp
                  //(01/01/2010) and present date.

  difyear = (fsw->jd - fsw->jd_igrfdata)/365.25;

  double T = (fsw->jd -2451545.0)/36525.0; //Julian Centuries since J2000
                                           //(Vallado p. 194)
  double gmst = 67310.54841 + (876600.0*3600.0 + 8640184.812866)*T
    + 0.093104*pow(T,2.0) - 6.2e-6 * pow(T,3.0); //Vallado p.194
  gmst = fmod(gmst,86400);

  double alphag = gmst *(1.0/240.0)*(fsw->PI/180.0); //converting from seconds
                                                     //to radians - this is the
                                                     //Greenwich mean sidereal time
  //For convenience and to make things easier to read:
```

```
double x, y, z, r;
x = fsw->pos_eci[0];
y = fsw->pos_eci[1];
z = fsw->pos_eci[2];
r = sqrt(pow(x,2)+pow(y,2)+pow(z,2));

double delta, alpha, theta, phi;
delta = atan2(z,sqrt(pow(x,2)+pow(y,2))); // geocentric latitude or declination
alpha = atan2(y,x); // sidereal time (right ascension)
theta = fsw->PI/2.0 - delta;
phi = alpha - alphag;  // longitude

int n,m;

double S[6+1][6+1];//see Wertz eqn (H-7)
S[0][0] = 1;
for(n = 1;n<k+1;n++){
  S[n][0] = S[n-1][0]*((2.0*n - 1.0)/n);
  for(m = 1;m<n+1;m++){
    S[n][m] = S[n][m-1] * sqrt(((n - m + 1.0)*((m==1)+1.0)) / (n+m));
  }
}

double g[6+1][6+1];//see Wertz eqn (H-6)
double h[6+1][6+1];
for(n = 0;n<k+1;n++){
  for(m = 0;m<n+1;m++){
    //also account for changing IGRF coefficients
    g[n][m] = S[n][m] * (fsw->igrf_g[n][m] + (fsw->igrf_gdot[n][m] * difyear));
    h[n][m] = S[n][m] * (fsw->igrf_h[n][m] + (fsw->igrf_hdot[n][m] * difyear));
  }
}

double K[6+1][6+1]; //see Wertz eqn (H-9)
```

```
   for(n = 1;n<k+1;n++){
     for(m = 0;m<n+1;m++){
       if(n==1) K[n][m] = 0.0;
       else{
         K[n][m] = (pow((n-1),2) - pow(m,2)) / ((2.0*n - 1.0)*(2.0*n - 3.0));
       }
     }
   }

 double P[6+1][6+1]; //see Wertz eqn (H-8)
  P[0][0] = 1.0;
  for(n = 1; n<k+1 ; n++){
    P[n][n] = sin(theta) * P[n-1][n-1];
    for(m = 0; m<n ; m++){
      if(n==1){
      P[n][m] = cos(theta) * P[n-1][m];
      }
      else{
      P[n][m] = cos(theta) * P[n-1][m] - K[n][m]*P[n-2][m];
      }
    }
  }

 double dP[6+1][6+1]; //see Wertz eqn (H-10)
  dP[0][0] = 0.0;
  for(n = 1; n<k+1 ; n++){
    dP[n][n] = (sin(theta) * dP[n-1][n-1]) + (cos(theta) * P[n-1][n-1]);
    for(m = 0; m<n ; m++){
      if(n==1){
        dP[n][m] = (cos(theta) * dP[n-1][m]) - (sin(theta) * P[n-1][m]);
      }
      else{
        dP[n][m] = (cos(theta) * dP[n-1][m]) - (sin(theta) * P[n-1][m])
          - (K[n][m]*dP[n-2][m]);
```

```
            }
        }
    }


    double Br, Btheta, Bphi; //see Wertz eqn (H-12)
    double temp1;
    Br = 0.0;
    for(n=1;n<k+1;n++){
        temp1 = 0.0;
        for(m=0;m<n+1;m++){
            temp1 = temp1 + ((g[n][m] * cos(m*phi)) + (h[n][m] * sin(m*phi))) * P[n][m];
        }
        Br = Br + (pow((a/r),(n+2.0)) * (n+1.0) * temp1);
    }

    Btheta = 0.0;
    for(n=1;n<k+1;n++){
        temp1 = 0.0;
        for(m=0;m<n+1;m++){
            temp1 = temp1 + ((g[n][m] * cos(m*phi)) + (h[n][m] * sin(m*phi))) * dP[n][m];
        }
        Btheta = Btheta - (pow((a/r),(n+2.0)) * temp1);
    }

    Bphi = 0.0;
    for(n=1;n<k+1;n++){
        temp1 = 0.0;
        for(m=0;m<n+1;m++){
            temp1 = temp1 + m*((-g[n][m] * sin(m*phi)) + (h[n][m] * cos(m*phi))) * P[n][m];
        }
        Bphi = Bphi - (pow((a/r),(n+2.0)) * temp1);
    }
    Bphi = Bphi * (1/sin(theta));
```

```
//Calculate Magnetic Field (should be in nanoTesla)
double B[3];
B[0] = (Br*cos(delta) + Btheta*sin(delta))*cos(alpha) - Bphi*sin(alpha);
B[1] = (Br*cos(delta) + Btheta*sin(delta))*sin(alpha) + Bphi*cos(alpha);
B[2] = (Br*sin(delta) - Btheta*cos(delta));
double B_mag;
B_mag = sqrt(pow(B[0],2)+pow(B[1],2)+pow(B[2],2));
fsw->B_actual[0] = B[0]/B_mag;//unit vector
fsw->B_actual[1] = B[1]/B_mag;
fsw->B_actual[2] = B[2]/B_mag;




//Unit vector for Measured magnetic field
double B_meas_mag;
double B_meas[3];
B_meas_mag = sqrt(pow(fsw->B_magnetometer[0],2)+pow(fsw->B_magnetometer[1],2)
                    +pow(fsw->B_magnetometer[2],2));
B_meas[0] = fsw->B_magnetometer[0]/B_meas_mag;
B_meas[1] = fsw->B_magnetometer[1]/B_meas_mag;
B_meas[2] = fsw->B_magnetometer[2]/B_meas_mag;





 //Compute "true" sun vector in ECI
double lambda_M = 280.460 + 36000.771*T;// Mean longitude of the sun in MOD frame.
lambda_M = fmod(lambda_M,360.0);

```

```
double M = 357.5277233 + 35999.05034*T; // Mean anomaly for the sun.
                                        // Uses approximation T_TDB ~ T_UT1
M = fmod(M,360);

double lambda_ecl = lambda_M + 1.914666471*sin(M*(fsw->PI/180.0))
  + 0.019994643*sin(2*M*(fsw->PI/180.0)); // Ecliptic longitude approximation

double rmag = 1.000140612 - 0.016708617*cos(M*(fsw->PI/180.0))
  - 0.000139589*cos(2*M*(fsw->PI/180.0));// Distance from earth to sun in AU
rmag = rmag*149597870700; //convert to meters;

double eps = 23.439291 - 0.0130042*T;//Obliquity of the ecliptic

double sunvec[3];
sunvec[0] = rmag*cos(lambda_ecl*(fsw->PI/180.0));
sunvec[1] = rmag*cos(eps*(fsw->PI/180.0))*sin(lambda_ecl*(fsw->PI/180.0));
sunvec[2] = rmag*sin(eps*(fsw->PI/180.0))*sin(lambda_ecl*(fsw->PI/180.0));
//sun vector in ECI frame [meters]

double sunvec_mag;
sunvec_mag = sqrt(pow(sunvec[0],2)+pow(sunvec[1],2)+pow(sunvec[2],2));
fsw->sunvec_actual[0] = sunvec[0]/sunvec_mag;
fsw->sunvec_actual[1] = sunvec[1]/sunvec_mag;
fsw->sunvec_actual[2] = sunvec[2]/sunvec_mag;//unit vector in ECI frame






 //

//V are "actual" vectors, W are measured vectors

```

```

double r1[3];
double r2[3];
double r3[3];
double s1[3];
double s2[3];
double s3[3];

//r1 = V_actual_1
int i;
for(i=0;i<3;i++){
  r1[i] = fsw->B_actual[i];
    }

//r2 = cross(V_actual_1, V_actual_2)/magnitude(cross(V_actual_1, V_actual_2))
cross(fsw->B_actual,fsw->sunvec_actual,r2);
double r2_mag;
r2_mag = sqrt(pow(r2[0],2)+pow(r2[1],2)+pow(r2[2],2));
int j;
for(j=0;j<3;j++){
  r2[j] = r2[j]/r2_mag;
    }

//r3 = cross(r1,r2)
cross(r1,r2,r3);


//s1 = W_meas_1
int indk;
for(indk=0;indk<3;indk++){
  s1[indk] = B_meas[indk];
    }

//s2 = cross(W_meas_1,W_meas_2)/magnitude(cross(W_meas_1,W_meas_2))
```

```
cross(B_meas,fsw->sunvec_meas,s2);
double s2_mag;
s2_mag = sqrt(pow(s2[0],2)+pow(s2[1],2)+pow(s2[2],2));
int l;
for(l=0;l<3;l++){
   s2[l] = s2[l]/s2_mag;
      }

//s3 = cross(s1,s2)
cross(s1,s2,s3);


//ECI to SBF Transformation Matrix
double A[3][3];
double A_1[3][3];
double A_2[3][3];

int row;
int col;
for(row = 0;row<3;row++){
   A_1[row][0] = s1[row];
   A_1[row][1] = s2[row];
   A_1[row][2] = s3[row];
}

for(col=0;col<3;col++){
   A_2[0][col] = r1[col];
   A_2[1][col] = r2[col];
   A_2[2][col] = r3[col];
}

mult_3x3(A_1,A_2,A);

//From Wertz p 415:
```

```
fsw->att_quat_meas[3] = 0.5*pow((1+A[0][0]+A[1][1]+A[2][2]),0.5);
fsw->att_quat_meas[0] = 1.0/(4.0*fsw->att_quat_meas[3])*(A[1][2]-A[2][1]);
fsw->att_quat_meas[1] = 1.0/(4.0*fsw->att_quat_meas[3])*(A[2][0]-A[0][2]);
fsw->att_quat_meas[2] = 1.0/(4.0*fsw->att_quat_meas[3])*(A[0][1]-A[1][0]);

//Normalize and properize
double att_quat_meas_mag;
att_quat_meas_mag = sqrt(pow(fsw->att_quat_meas[0],2)+pow(fsw->att_quat_meas[1],2)
                    +pow(fsw->att_quat_meas[2],2)+pow(fsw->att_quat_meas[3],2));

int quatind;
for(quatind = 0;quatind<4;quatind++){
  fsw->att_quat_meas[quatind] = fsw->att_quat_meas[quatind]/att_quat_meas_mag;
}

if(fsw->att_quat_meas[3]<0.0){
  for(quatind = 0;quatind<4;quatind++){
    fsw->att_quat_meas[quatind] = -1*fsw->att_quat_meas[quatind];
  }
}


 return(0);
}
```

A.3.6 Inputs and Outputs of *gnc_attitude_kf*

Table A.7: Inputs to gnc_attitude_kf

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
att_quat	quat	ECI to SBF rotation quaternion	this algorithm
body_rates	rad/sec	Spacecraft body rates about SBF axes	this algorithm
Q_kf	matrix	Kalman Filter estimate covariance	this algorithm
k_kf	n/a	Constant design parameter for UKF	fswparam
alpha_kf	n/a	Constant design parameter for UKF	fswparam
beta_kf	n/a	Constant design parameter for UKF	fswparam
W_kf	matrix	Kalman Filter process noise covariance	fswparam
valid_sunvec_meas	bool	Valid sunvector measurement flag	gnc_TRIAD

Continued on next page

Table A.7: Algorithm Inputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
att_quat_meas	quat	Measured attitude quaternion	gnc_TRIAD
body_rates_meas	rad/sec	Measured body rates	gnc_sensor_processing
R_kf	matrix	Kalman Filter sensor noise covariance	fswparam
cyc_period	sec	Length of tSat GNC flight software cycle	fswparam
I	kg-m^2	Spacecraft inertia matrix	initialize_fsw
M_tot	N-m	Total moment imparted on spacecraft	gnc_thrust_alloc

Table A.8: Outputs of gnc_attitude_kf

GNC Analysis Variable	**Units**	**Description**	**Value goes to...**
att_quat	quat	ECI to SBF rotation quaternion	attitude controller, this algorithm

Continued on next page

Table A.8: Algorithm Outputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value goes to...**
body_rates	rad/sec	Spacecraft body rates about SBF axes	attitude controller, this algorithm
Q_kf	matrix	Kalman Filter estimate covariance	this algorithm

A.3.7 Pseudocode for *gnc_attitude_kf*

1. Define $n = 7$ as the dimension of the state space for this system. The state vector is the following, where the first four elements represent the attitude quaternion of the spacecraft, and the next three elements represent the spacecraft body rates.

$$\vec{x} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{A.40}$$

(A.41)

2. Form $2n + 1$ sigma points $x_{k-1}^{(i)}$ according to the following logic:

$$x^{(0)} = \hat{x} \tag{A.42}$$

$$x^{(i)} = \hat{x} + (\sqrt{(n+\lambda)Q})_i^T \qquad \text{for } i \in [1, n] \tag{A.43}$$

$$x^{(i+n)} = \hat{x} - (\sqrt{(n+\lambda)Q})_i^T \qquad \text{for } i \in [1, n] \tag{A.44}$$

Here, $(\sqrt{(n+\lambda)Q})_i^T$ represents the i^{th} column of the matrix $(\sqrt{(n+\lambda)Q})^T$, found through a Cholesky decomposition. Q represents the estimate covariance. The Cholesky decomposition may be done as follows, or in a more efficient way if one is known. The following steps represent the current implementation of the $modified\_cholesky$ helper function in the tSat simulation.

- Define $\tilde{X} = (\sqrt{(n+\lambda)Q})^T$. This is what we are solving for in the Cholesky decomposition.
- Set $\tilde{X}_{0,0} = \sqrt{Q_{0,0}}$. Where the subscripts denote the element in the zeroth row and the zeroth column of the matrix.
- **For** $i \in [1, n-1]$:

$$\tilde{X}_{i,0} = \frac{Q_{i,0}}{\tilde{X}_{0,0}} \tag{A.45}$$

- **For** $j \in [1, n-1]$:

$$\tilde{X}_{j,j} = \sqrt{Q_{j,j} - \sum_{k=0}^{j-1} \tilde{X}_{j,k}^2} \tag{A.46}$$

for $i \in [j+1, n-1]$: (A.47)

$$\tilde{X}_{i,j} = \frac{1}{\tilde{X}_{j,j}} \left(Q_{i,j} - \sum_{k=0}^{j-1} \tilde{X}_{i,k} \times \tilde{X}_{j,k} \right) \tag{A.48}$$

- Multiply each element of the current $\tilde{X}$ by the constant $\sqrt{n+\lambda}$ to find $\tilde{X} =$

$\sqrt{(n+\lambda)^T}$. λ is a combination of design parameters given by

$$\lambda = \alpha^2(n+k) - n. \tag{A.49}$$

α and k are read in from a config file and $n = 7$ is the length of the state vector given in equation A.41.

3. Propagate each of the sigma points $x_{k-1}^{(i)}$ forward one filter timestep using a fourth-order Runge-Kutta propagator. Assume the system dynamics to be the following:

$$\dot{q} = \mathbf{\Omega} q \tag{A.50}$$

$$\dot{\vec{\omega}} = \mathbf{I}^{-1}(\vec{\tau} - \vec{\omega} \times \mathbf{I}\vec{\omega}) \tag{A.51}$$

Where q is the ECI to SBF attitude quaternion, I is the inertia of the spacecraft about the center of mass, $\vec{\omega}$ is a vector representing spacecraft body rates about SBF-X, SBF-Y, and SBF-Z axes, and $\vec{\tau}$ represents the total moment applied to the spacecraft as calculated by GNC thrust allocation. $\mathbf{\Omega}$ is given by the following:

$$\mathbf{\Omega} = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix} \tag{A.52}$$

The fourth-order Runge-Kutta propagator requires a helper function that returns the instantaneous derivative of each of the system states based on the system dynamics. In the tSat GNC simulation, this helper function is called d. It takes in a state as represented by equation A.41 and returns the left hand side of equations A.50 and

A.51 based on that state.

Note about levitating testbed mode: The flight software must support a levitating testbed mode that will be used to run tests on an engineering model in the TSat levitating testbed. In the testbed, spacecraft motion will be constrained to rotation about the SBF-Z axis. Thus, in the levitating testbed mode, the *d* helper function should set the SBF-X and SBF-Y components of the body-rate derivative $\dot{\vec{\omega}}$ to zero before returning the derivative. Thus, **for the levitating testbed mode only**, equation A.51 should effectively become:

$$\dot{\vec{\omega}} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} * (\mathbf{I}^{-1}(\vec{\tau} - \vec{\omega} \times \mathbf{I}\vec{\omega})) \tag{A.53}$$

A flight software configuration flag should be created to select between levitating testbed and flight modes upon flight software initialization.

As in the case of Cholesy decomposition, the fourth-order Runge-Kutta propagator may be implemented by the most efficient method known, provided the system dynamics given in equations A.50 and A.51 are used. The following steps represent the current implementation of the *propagate* helper function in the tSat GNC simulation.

- Evaluate the derivative of the state represented by the sigma point $x_{k-1}^{(i)}$ using the *d* helper function, and define the constant k_1.

  $$k_1 = (dt) \times d(x_{k-1}^{(i)}) \tag{A.54}$$

  Where (dt) is the length of the filter timestep in seconds.

- Define a new intermediate state $x_{int,1}$ as the Euler propagation of the original state $x_{k-1}^{(i)}$ to a point half way through the filter timestep.

  $$x_{int,1} = x_{k-1}^{(i)} + 0.5k_1 \tag{A.55}$$

- Find the derivative of the intermediate state $x_{int,1}$, and define the constant k_2.

$$k_2 = (dt) \times d(x_{int,1}) \tag{A.56}$$

- Define another intermediate state $x_{int,2}$ as the Euler propagation of the original state $x_{k-1}^{(i)}$, **using the derivative of the first intermediate state**, $x_{int,1}$, to a point half way through the filter timestep.

$$x_{int,2} = x_{k-1}^{(i)} + 0.5k_2 \tag{A.57}$$

- Find the derivative of the intermediate state $x_{int,2}$, and define the constant k_3.

$$k_3 = (dt) \times d(x_{int,2}) \tag{A.58}$$

- Define a third intermediate state $x_{int,3}$ as the Euler propagation of the original state $x_{k-1}^{(i)}$, **using the derivative of the second intermediate state**, $x_{int,2}$, this time through the **full** filter timestep.

$$x_{int,3} = x_{k-1}^{(i)} + k_3 \tag{A.59}$$

- Find the derivative of the intermediate state $x_{int,3}$, and define the constant k_4.

$$k_4 = (dt) \times d(x_{int,3}) \tag{A.60}$$

- Define a final state $x_{int,4}$ in the following way.

$$x_{int,4} = x_{k-1}^{(i)} + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \tag{A.61}$$

- Normalize and properize the quaternion elements of $x_{int,4}$.

- Return the final state $x_{int,4}$ as the propagated sigma point $x_k^{(i)}$.

$$x_k^{(i)} = x_{int,4} \tag{A.62}$$

4. Use the propagated sigma points $x_k^{(i)}$ to approximate the filter estimate $\hat{x}_{k|k-1}$ and covariance $Q_{k|k-1}$ just before the measurement update.

$$\hat{x}_{k|k-1} = \sum_{i=1}^{m} W_s^{(i)} x_k^{(i)} \tag{A.63}$$

$$Q_{k|k-1} = \sum_{i=1}^{m} W_c^{(i)} (x_k^{(i)} - \hat{x}_{k|k-1})(x_k^{(i)} - \hat{x}_{k|k-1})^T + W_{k-1} \tag{A.64}$$

   where W_s and W_c are the unscented transform weights:

$$W_s^{(0)} = \frac{\lambda}{(n+\lambda)} \tag{A.65}$$

$$W_c^{(0)} = \frac{\lambda}{(n+\lambda)} + (1 - \alpha^2 + \beta) \tag{A.66}$$

$$W_s^{(i)} = W_c^{(i)} = \frac{1}{2(n+\lambda)} \tag{A.67}$$

   and α and β are design parameters. $n = 7$ is the size of the state vector. In the current implementation, the unscented transform weights are calculated each time through the algorithm. However, the weights may be calculated manually and read in as config values in the interest of conserving processor time. W_{k-1} is the Kalman Filter process noise covariance read in from a config value.

5. **If** a new valid measurement vector is ready from sensor processing: Update the filter estimate and covariance with quaternion and body rate measurements using the discrete Kalman Filter equations. The unscented transform is not necessary here since

the measurement is assumed to be linear.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + L_k(y_k - \hat{x}_{k|k-1}) \tag{A.68}$$

$$Q_{k|k} = (I - L_k)Q_{k|k-1}(I - L_k)^T + L_k R_k L_k^T \tag{A.69}$$

Where

$$L_k = Q_{k|k-1}(Q_{k|k-1} + R_k)^{-1} \tag{A.70}$$

y_k is the measurement vector, in the same format as the state vector of equation A.41. R_k is the Kalman Filter sensor noise covariance read in from a config file.

Else: Do not perform the measurement update.

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} \tag{A.71}$$

$$Q_{k|k} = Q_{k|k-1} \tag{A.72}$$

6. Update flight software state attitude estimate and Kalman Filter covariance.

$$\hat{x} = \hat{x}_{k|k} \tag{A.73}$$

$$Q = Q_{k|k} \tag{A.74}$$

7. Normalize and properize the attitude quaternion estimate (First four elements of the estimate vector).

$$q = \frac{q}{||q||}$$

$$\text{If } q_4(\text{scalar}) < 0$$

$$q = -q \tag{A.75}$$

A.3.8 TSATsim Implementation of *gnc_attitude_kf*

This section lists the TSat GNC analysis simulation implementation of *gnc_attitude_kf*.

```
/* gnc_attitude_kf.cc
   Implementation of an Unscented Kalman filter for estimating the
   spacecraft attitude quaternion and body rates.
   Implemented for TSat ADCS analysis
   Mark Van de Loo, November 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_attitude_kf.h"
#include "custom_math.h"

double  d(double state[], int i, fswstate *fsw);

int propagate(double state0[7], fswstate *fsw, double state4[7]);

void modified_chol(double Q[7][7], int n, double k,double alpha, double beta,
                   double x_til[7][7]);

int gnc_attitude_kf(fswstate *fsw)
{

  //form sigma points
  int n = 7;
  int num_sig_pts = 2*n+1;

  double x_hat[7] = {0.0};
  int stind = 0;
  for(stind = 0;stind<n;stind++){
    x_hat[stind] = 0.0;
  }
  x_hat[0] = fsw->att_quat[0];
  x_hat[1] = fsw->att_quat[1];
  x_hat[2] = fsw->att_quat[2];
  x_hat[3] = fsw->att_quat[3];
  x_hat[4] = fsw->body_rates[0];
  x_hat[5] = fsw->body_rates[1];
  x_hat[6] = fsw->body_rates[2];

  double x_til[7][7]= {0.0};

  double sig_pts[7][15] = {0.0};

  int i = 0;
  for(stind = 0;stind<n;stind++){
    for(i = 0;i<num_sig_pts;i++){
      sig_pts[stind][i] = 0.0;
    }
  }

  int initind1 = 0;
  int initind2 = 0;
  for( initind1=0;initind1 <7;initind1++){
    for( initind2=0;initind2 <7;initind2++){
      x_til[initind1][initind2] = 0.0;
    }
  }

  modified_chol(fsw->Q_kf,n,fsw->k_kf,fsw->alpha_kf,fsw->beta_kf, x_til);

  for(stind = 0;stind<n;stind++){
    for(i = 0;i<n;i++){
        sig_pts[stind][i] = x_hat[stind] + x_til[stind][i];
        sig_pts[stind][i+n] = x_hat[stind] - x_til[stind][i];
      }
      sig_pts[stind][14] = x_hat[stind];
    }

    //normalize and properize quaternion parts of each sigma pt
    double quat_mag = 0.0;
    int quatind = 0;
    for(i=0;i<num_sig_pts;i++){
      quat_mag = sqrt(pow(sig_pts[0][i],2)+pow(sig_pts[1][i],2)
                      +pow(sig_pts[2][i],2)+pow(sig_pts[3][i],2));
      for(quatind = 0;quatind<4;quatind++){
        sig_pts[quatind][i] = sig_pts[quatind][i]/quat_mag;
      }

      if(sig_pts[3][i]<0.0){
        for(quatind = 0;quatind<4;quatind++){
          sig_pts[quatind][i] = -1.0*sig_pts[quatind][i];
        }
      }
    }


    //propagate each sigma point
    double x_hat_bef[7] = { 0.0 };
    double x_hat_aft[7] = { 0.0 };
    for(i=0;i<num_sig_pts;i++){
      for(stind = 0;stind<n;stind++){
        x_hat_bef[stind] = sig_pts[stind][i];
      }
      propagate(x_hat_bef,fsw,x_hat_aft);
      for(stind = 0;stind<n;stind++){
        sig_pts[stind][i] = x_hat_aft[stind];
    }
  }

  //normalize and properize quaternion parts of each sigma pt
  for(i=0;i<num_sig_pts;i++){
    quat_mag = sqrt(pow(sig_pts[0][i],2)+pow(sig_pts[1][i],2)
                    +pow(sig_pts[2][i],2)+pow(sig_pts[3][i],2));
    for(quatind = 0;quatind<4;quatind++){
      sig_pts[quatind][i] = sig_pts[quatind][i]/quat_mag;
    }

    if(sig_pts[3][i]<0.0){
      for(quatind = 0;quatind<4;quatind++){
        sig_pts[quatind][i] = -1.0*sig_pts[quatind][i];
      }
    }
  }


  //approximate estimate and covariance pre-measurement
  double x_hat_pre[7] = {0.0};
  double x_hat_post[7] = {0.0};
  double Q_pre[7][7] = {0.0};
  double Ws[15] = {0.0};
  double Wc[15] = {0.0};
  double lamb = 0.0;

  lamb = (fsw->alpha_kf*fsw->alpha_kf*((double)n + fsw->k_kf) - (double) n);

  for(i = 0; i<num_sig_pts; i++){
      Ws[i] = 1.0/(2.0*((double)n+lamb));
      Wc[i] = 1.0/(2.0*((double)n+lamb));
      Ws[14] = lamb/((double)n+lamb);
      Wc[14] = lamb/((double)n+lamb) + (1.0-fsw->alpha_kf*fsw->alpha_kf + fsw->beta_kf
}

for(stind = 0;stind<n;stind++){
  x_hat_pre[stind] = 0.0;
  for(i = 0; i<num_sig_pts; i++){
    x_hat_pre[stind] = x_hat_pre[stind] + Ws[i]*sig_pts[stind][i];
  }
}
int qrow = 0;
int qcol = 0;
double xdif[7] = {0.0};
for(qrow = 0;qrow<n;qrow++){
  for(qcol = 0;qcol<n;qcol++){
    Q_pre[qrow][qcol] = 0.0;
    for(i = 0; i<num_sig_pts; i++){
      for(stind = 0;stind<n;stind++){
        xdif[stind] = sig_pts[stind][i] - x_hat_pre[stind];
      }
      Q_pre[qrow][qcol] = Q_pre[qrow][qcol] + Wc[i]*(xdif[qrow]*xdif[qcol]);
    }
    Q_pre[qrow][qcol] = (Q_pre[qrow][qcol]) + fsw->W_kf[qrow][qcol];
  }
}


//measurement update
if(fsw->valid_sunvec_meas){
  //y = [quat_meas body_rates_meas]'
  double y[7] = {0.0};
  for(stind = 0;stind <4;stind++){
    y[stind] = fsw->att_quat_meas[stind];
  }
  for(stind = 4;stind <7;stind++){
    y[stind] = fsw->body_rates_meas[stind -4];
    }


    //L = Q*(Q+R)^-1
    double Q_plus_R[7][7] = {0.0};
    for(qrow = 0;qrow<n;qrow++){
      for(qcol = 0;qcol<n;qcol++){
        Q_plus_R[qrow][qcol] = Q_pre[qrow][qcol] + fsw->R_kf[qrow][qcol];
      }
    }

    double Q_plus_R_inv[7][7] = {0.0};
    inv_7x7(Q_plus_R,Q_plus_R_inv);

    double L[7][7] = {0.0};
    mult_7x7(Q_pre,Q_plus_R_inv,L);

    //x_hat_post = x_hat_pre + L*(y - x_hat_pre)
    double res[7] = {0.0};
    for(stind = 0;stind<n;stind++){
      res[stind] = y[stind] - x_hat_pre[stind];
    }

    double Lres[7] = {0.0};
    mult_7x7_7x1(L,res,Lres);

    for(stind = 0;stind<n;stind++){
      x_hat_post[stind] = x_hat_pre[stind] + Lres[stind];
    }


    //Q_post = (I-L)*Q_pre*(I-L)' + L*R*L'
    double I_minus_L[7][7] = {0.0};
    for(qrow = 0;qrow<n;qrow++){
        for(qcol = 0;qcol<n;qcol++){
          if(qrow == qcol){
            I_minus_L[qrow][qcol] = 1.0-L[qrow][qcol];
          }
          else{
            I_minus_L[qrow][qcol] = 0.0-L[qrow][qcol];
          }
        }
      }

      double I_minus_L_T[7][7] = {0.0};
      transp_7x7(I_minus_L,I_minus_L_T);

      double term1_first[7][7] = {0.0};
      mult_7x7(I_minus_L,Q_pre,term1_first);

      double term1_all[7][7] = {0.0};
      mult_7x7(term1_first,I_minus_L_T,term1_all);

      double L_T[7][7] = {0.0};
      transp_7x7(L,L_T);

      double term2_first[7][7] = {0.0};
      mult_7x7(L,fsw->R_kf,term2_first);

      double term2_all[7][7] = {0.0};
      mult_7x7(term2_first,L_T,term2_all);

      for(qrow = 0;qrow<n;qrow++){
        for(qcol = 0;qcol<n;qcol++){
          fsw->Q_kf[qrow][qcol] = term1_all[qrow][qcol] + term2_all[qrow][qcol];
        }
      }
    }
    else{ //simply propagate if we can't see the sun
      for(stind = 0;stind<n;stind++){
        x_hat_post[stind] = x_hat_pre[stind];
      }
      for(qrow = 0;qrow<n;qrow++){
        for(qcol = 0;qcol<n;qcol++){
          fsw->Q_kf[qrow][qcol] = Q_pre[qrow][qcol];
        }
      }
    }


    //Transfer x_hat back to fsw variables
    for (quatind = 0;quatind<4;quatind++){
      fsw->att_quat[quatind] = x_hat_post[quatind];
    }
    //Normalize and properize
    double att_quat_mag = 0.0;
    att_quat_mag = sqrt(pow(fsw->att_quat[0],2)+pow(fsw->att_quat[1],2)
                        +pow(fsw->att_quat[2],2)+pow(fsw->att_quat[3],2));
    for(quatind = 0;quatind<4;quatind++){
      fsw->att_quat[quatind] = fsw->att_quat[quatind]/att_quat_mag;
    }
    if(fsw->att_quat[3]<0.0){
      for(quatind = 0;quatind<4;quatind++){
        fsw->att_quat[quatind] = -1*fsw->att_quat[quatind];
      }
    }

    int wind = 0;
    for(wind = 0;wind<3;wind++){
      fsw->body_rates[wind] = x_hat_post[wind+4];
    }

```

```

  return(0);
}





int propagate(double state0[7],fswstate *fsw,double state4[7])
{
  double dt = fsw->cyc_period;

  int N = 7;

  double state1 [7] = { 0.0 };
  double state2 [7] = { 0.0 };
  double state3 [7] = { 0.0 };
  double    k1 [7] = { 0.0 };
  double    k2 [7] = { 0.0 };
  double    k3 [7] = { 0.0 };
  double    k4 [7] = { 0.0 };          /* for Runge-Kutta  */
   int i = 0;

   double no_step = 0.0;
   double half_step = 0.5;
   double full_step = 1.0;


   for (i=0; i<N; i++) state1[i] = state0[i]+0.5*(k1[i]=dt*d(state0,i,fsw));
   for (i=0; i<N; i++) state2[i] = state0[i]+0.5*(k2[i]=dt*d(state1, i,fsw));
   for (i=0; i<N; i++) state3[i] = state0[i]+    (k3[i]=dt*d(state2, i,fsw));
   for (i=0; i<N; i++) k4[i] =                    dt*d(state3, i,fsw);

   for (i=0; i<N; i++) state4[i] = state0[i]+(k1[i]+2*k2[i]+2*k3[i]+k4[i])/6.0;


   //normalize and properize quaternion
     double mag = 0.0;
     mag = sqrt(state4[0]*state4[0] + state4[1]*state4[1] + state4[2]*state4[2] +
                 state4[3]*state4[3]);

     int quatind = 0;
     for (quatind = 0;quatind<4;quatind++){
       state4[quatind] = state4[quatind]/mag;
     }

     if(state4[3]<0.0){
       for(quatind = 0;quatind<4;quatind++){
         state4[quatind] = -1*state4[quatind];
       }
     }


  return(0);
}


double  d(double state[], int i, fswstate *fsw)
{

    //Quaternion time derivative from Wertz
  if (i == 0) return(0.5*((state[1]*state[6]) - (state[2]*state[5]) +
                          (state[3]*state[4])));
  if (i == 1) return(0.5*((-state[0]*state[6])+(state[2]*state[4])+(state[3]*state[5]))
  if (i == 2) return(0.5*((state[0]*state[5])-(state[1]*state[4])+(state[3]*state[6])))
  if (i == 3) return(0.5*((-state[0]*state[4])-(state[1]*state[5])-(state[2]*state[6]))

  if (i==4||i==5||i==6){
    //Angular rates time derivative from physics: w_dot = I^-1*(M_Tot -
    //body_rates X H)
    double body_rates[3] = {0.0};
    int wind = 0;
    for(wind = 0;wind<3;wind++){
      body_rates[wind] = state[wind+4];
    }

    double H[3] = {0.0};
    mult_3x3_3x1(fsw->I,body_rates,H);

    double I_inv[3][3] = {0.0};
    inv_3x3(fsw->I,I_inv);

    double w_X_H[3] = {0.0};
    cross(body_rates,H,w_X_H);

    double M_minus_w_X_H[3] = {0.0};
    vec_minus(fsw->M_tot,w_X_H,M_minus_w_X_H);

    double w_dot[3] = {0.0};
    mult_3x3_3x1(I_inv,M_minus_w_X_H,w_dot);


    if (i == 4) return(w_dot[0]);
    if (i == 5) return(w_dot[1]);
    if (i == 6) return(w_dot[2]);
  }


  return(9999); //ERROR


```

```
}




void modified_chol(double Q[7][7], int n, double k,double alpha, double beta,
                   double x_til[7][7])
    {
      int row2 = 0;
      int col2 = 0;
      for(row2 = 0;row2<n;row2++){
        for(col2 = 0;col2<n;col2++){
          x_til[row2][col2] = 0.0;
        }
      }

      x_til[0][0] = sqrt(Q[0][0]);

      int row1 = 0;
      for(row1 = 1; row1<n; row1++){
        x_til[row1][0] = (1/x_til[0][0])*Q[row1][0];
      }

      int col = 0;
      for(col = 1;col<n;col++){
        double sum1 = 0.0;
        sum1 = 0.0;
        int k = 0;
        for(k = 0;k<col;k++){
          sum1 = sum1 + (x_til[col][k]*x_til[col][k]);
        }
        x_til[col][col] = sqrt(Q[col][col] - sum1);
        int row = 0;
```

```
            for(row = col+1; row<n;row++){
              double sum2 = 0.0;
              sum2 = 0.0;
              int k1 = 0;
              for(k1 = 0;k1<col;k1++){
                sum2 = sum2 + x_til[row][k1]*x_til[col][k1];
              }
              x_til[row][col] = (1/(x_til[col][col]))*(Q[row][col] - sum2);
            }
          }

          int nind1 = 0;
          int nind2 = 0;
          double lamb = 0.0;
          lamb = alpha*alpha*((double)n + k) - (double)n;
          for(nind1 = 0;nind1<n;nind1++){
            for(nind2 = 0;nind2<n;nind2++){
              x_til[nind1][nind2] = sqrt((double) n + lamb)*x_til[nind1][nind2];
            }
          }

        }
```

A.4 Guidance

A.4.1 Pseudocode for *gnc_guidance*

The current implementation is extremely simple and is intended for ground testing. More tasks will be added for the flight version.

1. Execute ground commands. (call *fsw_gnd_cmd*)

A.4.2 TSATsim implementation of *gnc_guidance*

This section lists the TSat GNC analysis simulation implementation of the algorithm. The current implementation of this wrapper is for analysis and engineering model testbed testing. It will have to be adjusted to take on more functionality for flight.

```
/* gnc_guidance.cc
   Sets modes, high level behavior for tSat gnc.
   Written as part of TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_guidance.h"


int gnc_guidance(fswstate *fsw)
{


  //execute ground commands
  fsw_gnd_cmd(fsw);




  return(0);
}
```

A.4.3 Inputs and Outputs of *fsw_gnd_cmd*

Table A.9: Inputs to fsw_gnd_cmd

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
max_gnd_cmd	int	maximum number of commands that can be sent in one run of TSat GNC sim	fswparam
time	sec	fsw mission time	tsat_gnc
ground_cmd_time	sec	time to execute a given ground command	fswcmd
ground_cmd_executed	bool	flag, true if the corresponding command has been executed	this algorithm
ground_cmd_num	int	ID number of a given ground command	fswcmd
ground_cmd_dubarg	any	Double argument of a given ground command	fswcmd

Continued on next page

Table A.9: Algorithm Inputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
ground_cmd_intarg	any	Integer argument of a given ground command	fswcmd

Table A.10: Outputs of fsw_gnd_cmd

GNC Analysis Variable	**Units**	**Description**	**Value goes to...**
fsw	state registry	any flight software variable	any flight software algorithm
ground_cmd_executed	bool	execution flag for each ground command	this algorithm

A.4.4 Pseudocode for *fsw_gnd_cmd*

```
1. For each of the command times read in from the fswcmd file:
    • If the command time is greater than the current mission time and the command
      has not been executed already: execute the code associated with the command's
      ID number and arguments.
```

A.4.5 TSATsim Implementation of *fsw_gnd_cmd*

This section lists the TSat GNC analysis simulation implementation of *fsw_gnd_cmd*.

```
/* fsw_gnd_cmd.c
   Executes flight software ground commands
   Written for TSat ADCS analysis
   Mark Van de Loo, Spring 2013
*/


#include <stdio.h>

#include "fsw_gnd_cmd.h"


int fsw_gnd_cmd(fswstate *fsw)
{
  int index;
  for(index = 0;index<fsw->max_gnd_cmd;index++){

    if(fsw->time+fsw->eps>=fsw->ground_cmd_time[index] &&
       fsw->ground_cmd_executed[index]==0){

 //Command #1: THRUST_LEVEL[thrusternumber][thrustlevel]
      if(fsw->ground_cmd_num[index] == 1){
        fsw->thr_cmd[fsw->ground_cmd_intarg[index]] = fsw->ground_cmd_dubarg1[index];
        printf("%s␣%i␣%s␣%.0f%s␣%.3f␣%s␣\n","Thruster",fsw->ground_cmd_intarg[index]
               ,"Commanded␣to",100*fsw->ground_cmd_dubarg1[index],"%␣at",fsw->time,
               "seconds.");
        fsw->ground_cmd_executed[index] = 1;
      }

 //Command #2: ATT_QUAT[q0][q1][q2][q3(scalar)]
      if(fsw->ground_cmd_num[index] == 2){
        fsw->att_quat_cmd[0] = fsw->ground_cmd_dubarg1[index];
        fsw->att_quat_cmd[1] = fsw->ground_cmd_dubarg2[index];
        fsw->att_quat_cmd[2] = fsw->ground_cmd_dubarg3[index];
```

```
                fsw->att_quat_cmd[3] = fsw->ground_cmd_dubarg4[index];
                printf("%s␣%.6f␣%s␣%.6f␣%s␣%.6f␣%s␣%.6f␣\n","Attitude␣quaternion␣commanded␣to:␣
                        ,"",fsw->att_quat_cmd[1]
                        ,"",fsw->att_quat_cmd[2]
                        ,"",fsw->att_quat_cmd[3]);
                fsw->ground_cmd_executed[index] = 1;
            }




        }
    }



    return(0);
}
```

A.5 Attitude Control Law

A.5.1 Pseudocode for *gnc_attitude_cl*

This pseudocode represents the current implementation of the attitude control law wrapper.

1. **If** there is a valid sunvector measurement available from sensor processing this cycle: call *gnc_torque_cmd*.

   Else: Set the torque command to zero.

A.5.2 TSATsim implementation of *gnc_attitude_cl*

This section lists the TSat GNC analysis simulation implementation of the algorithm.

```
/* gnc_attitude_cl.c
   Attitude Control Law
   Written as part of TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_attitude_cl.h"


int gnc_attitude_cl(fswstate *fsw)
{

  /* if(fsw->valid_sunvec_meas){ */
  //Run control law to find torque command
  gnc_torque_cmd(fsw);
  /* } */
  /* else{ */
  /*   int i; */
  /*   for(i = 0; i<3; i++){ */
  /*   fsw->torque_cmd[i] = 0.0; */
  /*   } */
  /* } */

  return(0);
}
```

A.5.3 Inputs and Outputs of *gnc_torque_cmd*

Table A.11: Inputs to gnc_torque_cmd

GNC Analysis Variable	Units	Description	Value comes from...
att_quat	quat	Estimated ECI to SBF rotation quaternion	gnc_attitude_kf
body_rates	rad/sec	Estimated spacecraft body rates about SBF axes	gnc_attitude_kf
att_quat_cmd	quat	Commanded ECI to SBF rotation quaternion	gnc_guidance (TBR)
body_rates_cmd	rad/sec	Commanded spacecraft body rates about SBF axes	gnc_guidance (TBR)
cyc_period	sec	Length of tSat GNC flight software cycle	fswparam
I	kg-m^2	Spacecraft inertia matrix	initialize_fsw
K_p	n/a	Proportional control gain for torque controller	fswparam

Continued on next page

Table A.11: Algorithm Inputs (continued)

GNC Analysis Variable	Units	Description	Value comes from...
K_i	n/a	Integral control gain for torque controller	fswparam
K_d	n/a	Derivative control gain for torque controller	fswparam

Table A.12: Outputs of gnc_torque_cmd

GNC Analysis Variable	Units	Description	Value goes to...
torque_cmd	N-m	Total commanded actuator torque	gnc_cmd_prep

A.5.4 Pseudocode for *gnc_torque_cmd*

1. Calculate the angular body rate error.

$$\vec{\omega_{err}} = \vec{\omega_{est}} - \vec{\omega_{cmd}} \tag{A.76}$$

$\vec{\omega_{est}}$ is the estimated body rate vector from gnc_attitude_kf, and $\vec{\omega_{cmd}}$ is the commanded body rate vector from gnc_guidance (TBR).

2. Calculate the attitude quaternion error.

$$q_{err} = q^* \otimes q_{cmd} \tag{A.77}$$

Here, q^* is the quaternion conjugate of the estimated attitude quaternion, and q_{cmd} is the commanded attitude quaternion from gnc_guidance (TBR). The symbol $\otimes$ denotes quaternion multiplication, and thus q_{err} represents the rotation from the estimated attitude to the commanded attitude.

3. Normalize and properize the error quaternion.

$$q_{err} = \frac{q_{err}}{||q_{err}||}$$

$$\text{If } q_{err,4}(\text{scalar}) < 0$$

$$q_{err} = -q_{err}$$

4. Calculate the torque command with the PID controller.

$$\vec{\tau_{cmd}} = \vec{\omega} \times I\omega - 2K_p q_{err,vec} - 2K_i q_{err,vec}(\Delta t) - K_d \vec{\omega_{err}} \tag{A.78}$$

Here $\vec{\omega}$ is a vector of the spacecraft body rates, I is the spacecraft inertia, and $\times$ denotes a cross product. K_p, K_i and K_d are the controller gains, $\vec{\omega_{err}}$ is the rate error calculated in step 1, and $q_{err,vec}$ is the vector part of the error quaternion that was calculated in step 2. (Δt) is the TSat control cycle period.

A.5.5 TSATsim Implementation of *gnc_torque_cmd*

This section lists the TSat GNC analysis simulation implementation of the algorithm.

```
/* gnc_torque_cmd.cc
```

```
   PID controller that calculates a commanded torque to be sent to
   thrusters or torque coils.
   Implemented for TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_torque_cmd.h"
#include "custom_math.h"


int gnc_torque_cmd(fswstate *fsw)
{

  //

  //Calculate angular rate error
  double w_err[3];
  int ind;
  for(ind = 0; ind<3; ind++){
    w_err[ind] = fsw->body_rates[ind] - fsw->body_rates_cmd[ind];
  }


  //Calculate quaternion error
  double q_err[4];
  double att_quat_cmd_conj[4];

  //att_quat_conj = fsw->att_quat*
  quat_conj(fsw->att_quat_cmd, att_quat_cmd_conj);

```

```
//q_err = att_quat_cmd_conj (*) fsw->att_quat
mult_quat(att_quat_cmd_conj,fsw->att_quat,q_err);

//Normalize and properize q_err
double q_err_mag;
q_err_mag = sqrt(pow(q_err[0],2)+pow(q_err[1],2)
                     +pow(q_err[2],2)+pow(q_err[3],2));

int quatind;
for(quatind = 0;quatind<4;quatind++){
  q_err[quatind] = q_err[quatind]/q_err_mag;
}

if(q_err[3]<0.0){
  for(quatind = 0;quatind<4;quatind++){
    q_err[quatind] = -1*q_err[quatind];
  }
}



//Calculate gyroscopic coupling term
double gycouple[3];
double H[3]; //spacecraft angular momentum

//H = J*w
mult_3x3_3x1(fsw->I,fsw->body_rates,H);

//gycouple = cross(w,H)
cross(fsw->body_rates,H,gycouple);


//calculate integrand
int intind;
```

```
    for (intind = 0;intind<3;intind++){
      fsw->integ_term[intind] = fsw->integ_term[intind] + 2*q_err[intind]*fsw->cyc_period;
      if(fsw->integ_term[intind] > 100){
        fsw->integ_term[intind] = 100;
      }
    }

    //Assemble control law, calculate torque command
    fsw->torque_cmd[0] = gycouple[0] - (fsw->K_p*2*q_err[0]) -
      (fsw->K_i*fsw->integ_term[0]) - (fsw->K_d*w_err[0]);
    fsw->torque_cmd[1] = gycouple[1] - (fsw->K_p*2*q_err[1]) -
      (fsw->K_i*fsw->integ_term[1]) - (fsw->K_d*w_err[1]);
    fsw->torque_cmd[2] = gycouple[2] - (fsw->K_p*2*q_err[2]) -
      (fsw->K_i*fsw->integ_term[2]) - (fsw->K_d*w_err[2]);

    return(0);
}
```

A.6 Command Preparation

A.6.1 Pseudocode for *gnc_cmd_prep*

The current implementation of this wrapper is for analysis and engineering model testbed testing. It will have to be adjusted for flight. In particular, the current implementation does not consider the presence of magnetorquers.

1. Run the thruster allocation algorithm. (call *gnc_thrust_alloc*)

2. Add up the total moment produced by the spacecraft actuators. For the testbed configuration, the total moment is equal to the moment produced by the thrusters.

$$\vec{M_{tot}} = \vec{\tau_{thr}} \tag{A.79}$$

A.6.2 TSATsim implementation of *gnc_cmd_prep*

This section lists the TSat GNC analysis simulation implementation of the algorithm. The current implementation of this wrapper is for analysis and engineering model testbed testing. It will be adjusted for flight.

```
/* gnc_cmd_prep.c
   Command preparation, includes control allocation, etc.
   Written as part of TSat ADCS analysis
   Mark Van de Loo, September 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_cmd_prep.h"


int gnc_cmd_prep(fswstate *fsw)
{

  //Run thrust allocation
  gnc_thrust_alloc(fsw);

  int j;
  for(j = 0;j<3;j++){
    fsw->M_tot[j] = fsw->torque_thr[j];
    //TODO: Update to include all torques, and update calculation of
    //torque_actual to be more accurate.
  }

  return(0);
28 }
```

A.6.3 Inputs and Outputs of *gnc_thrust_alloc*

Table A.13: Inputs to gnc_thrust_alloc

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
num_thr	n/a	Number of thrusters on the spacecraft	fswparam
torque_cmd	N-m	Torque commanded by the attitude controller in CMF	gnc_torque_cmd
thr_alloc_tol	N-m	Tolerance on thrust allocation error	fswparam
thr_alloc_maxiter	n/a	Maximum number of iterations allowed during thrust allocation search	fswparam

Continued on next page

Table A.13: Algorithm Inputs (continued)

GNC Analysis Variable	**Units**	**Description**	**Value comes from...**
thr_torque_uvecs	matrix	Matrix with each row containing a unit vector in the direction of the torque (CMF) applied by the thruster corresponding to the row number	fswparam
thr_torque_mag	N-m	Vector containing the magnitudes of torques applied by each thruster running at full throttle	fswparam

Table A.14: Outputs of gnc_thrust_alloc

GNC Analysis Variable	**Units**	**Description**	**Value goes to...**
thr_cmd	%	Commanded thrust level for each thruster as a fraction of full throttle	thruster driver
torque_actual	N-m	Predicted actual torque vector (CMF) applied by the thrusters	gnc_attitude_kf

A.6.4 Pseudocode for *gnc_thrust_alloc*

1. Initialize the variable that holds the actual commanded thruster torque magnitudes for each thruster (thr).

$$\tau_{thr} = 0.0 \tag{A.80}$$

2. Initialize the variable that holds the actual commanded torque vector.

$$\vec{\tau}_{actual} = 0.0 \tag{A.81}$$

3. Set error vector equal to the torque command.

$$\vec{e} = \vec{\tau}_{cmd} \tag{A.82}$$

4. Check whether the norm of the error vector is greater than the tolerance threshold and the current iteration is less than the maximum number of iterations. If yes, continue with step 5. If no, go to step 10.

5. Project the error vector along each of the available thruster torque vectors. Select the thruster with the largest projected magnitude and note that thruster's id along with the projected magnitude.

   - Initialize the largest projected magnitude, $p_{max} = 0.0$ and the selected thruster to, $t_{sel} = 99$.
   - Loop over all of the thrusters, and do the following for each thruster (thr).

   $$p = \vec{e} \cdot \vec{u}_{thr} \tag{A.83}$$

   If: $p > p_{max}$

   $$p_{max} = p \tag{A.84}$$

   $$t_{sel} = \text{thr} \tag{A.85}$$

   Here, $\vec{u}_{thr}$ is the unit vector in the direction of the torque that can be applied by the thruster (thr).

6. If there is no available control torque within 90 degrees of the error vector, t_{sel} will still be equal to the initialized value $t_{sel} = 99$. If this is the case, log a fault (TBR). This should never happen if thrusters can produce a torque in any direction.

7. Increment the thruster torque command of thruster (thr) by the projected magnitude p_{max}.

$$\tau_{thr} = \tau_{thr} + p_{max} \tag{A.86}$$

8. Update the actual commanded torque, and compute a new error vector.

$$\vec{\tau}_{actual} = \vec{\tau}_{actual} + p_{max}\vec{u}_{thr} \tag{A.87}$$

$$\vec{e} = \vec{\tau}_{cmd} - \vec{\tau}_{actual} \tag{A.88}$$

9. Update the iteration counter.

10. If the norm of the final error vector $\vec{e}$ is less than the tolerance threshold, continue to the next step. Otherwise, throw a thrust allocation fault (TBR) and skip the remaining steps.

11. Convert the commanded thruster torques to percentage values by scaling by the maximum torque magnitude. Do this for each thruster (thr).

$$T_{thr} = \frac{\tau_{thr}}{\tau_{max,thr}} \tag{A.89}$$

Here, τ_{thr} is the thruster torque command for thruster (thr) and τ_{max} is the magnitude of the torque that the thruster (thr) produces when running at full throttle.

12. If any of the fractional thruster torque commands T_{thr} are greater than 1, normalize all of the torque commands by the largest command. This ensures that the actual torque produced will be in the direction of the commanded torque, even if the magnitude is less than desired.

If: any $T_{thr} > 1$

$$T_{thr} = \frac{T_{thr}}{\max(T_{thr})} \qquad \text{for all thrusters (thr)} \tag{A.90}$$

Where "max" denotes the maximum value over all the thrusters.

13. Send the thruster commands T_{thr} to the thruster driver. (TBR)

A.6.5 TSATsim Implementation of *gnc_thrust_alloc*

This section lists the TSat GNC analysis simulation implementation of the algorithm.

```
/* gnc_thrust_alloc.cc
   Control allocation algorithm. Determines thruster levels that will
   produce the desired torque.
   Implemented for TSat ADCS analysis
   Mark Van de Loo, October 2013
*/


#include <stdio.h>
#include <math.h>

#include "gnc_thrust_alloc.h"
#include "custom_math.h"

void project(double err_vec[3], fswstate *fsw,int *thruster,double *amount);

int gnc_thrust_alloc(fswstate *fsw)
{
  double err[3];
  double torque_actual[3];
  double thr_trq_cmd[fsw->num_thr];

  int thrind;
  for(thrind = 0; thrind<fsw->num_thr; thrind++){
    thr_trq_cmd[thrind] = 0.0;
  }

  //make initial error equal to entire torque command
  int j;
  for(j=0;j<3;j++){
      err[j] = fsw->torque_cmd[j];
    }

    //      printf("%s %f \n","err: ",err[0]);

    //calculate magnitude of error
    double norm_err;
    norm_err = sqrt(pow(err[0],2)+pow(err[1],2)+pow(err[2],2));

    // printf("%s %f \n","NORM ERR: ",norm_err);

      int l;
      for(l=0;l<3;l++){
        torque_actual[l] = 0.0;
      }


    int iter = 0;
    int thruster;
    double amount;
    //loop until error is within tolerance or until max iter is reached
    while((norm_err > fsw->thr_alloc_tol) && (iter < fsw->thr_alloc_maxiter)){
      //project error along each of the available torque vectors, choose
      //thruster that produces the torque closest to the direction of
      //the error, scale according to projection
      project(err,fsw,&thruster, &amount);

      //Thr 99 event - no available control torque within 90 degrees of desired
      if(thruster > fsw->num_thr){
        printf("%s \n","THRUSTER_99_EVENT");
        break;
      }

      //add prescribed thrust to the total thrust command vector
    thr_trq_cmd[thruster]  = thr_trq_cmd[thruster] + amount;

    //update actual commanded torque and error
    int k;
    for(k=0;k<3;k++){
      torque_actual[k] = torque_actual[k] +
        amount*fsw->thr_torque_uvecs[thruster][k];
      err[k] = fsw->torque_cmd[k] - torque_actual[k];
    }

    //calculate magnitude of error
    norm_err = sqrt(pow(err[0],2)+pow(err[1],2)+pow(err[2],2));

    //update iteration counter
    iter = iter+1;

  }

//  printf("%s %i \n","THRUSTER ALLOCATION ITER: ",iter);

  if(iter == fsw->thr_alloc_maxiter){
    printf("%s \n","THRUSTER_ALLOCATION_MAX_ITER");
  }

  if(norm_err < fsw->thr_alloc_tol){
    //scale thruster torque commands by maximum torque magnitude to produce a
    //percentage value for fsw->thr_cmd.
    int thr;
    double maxcmd;
    maxcmd = 1.0;
    for(thr = 0; thr<fsw->num_thr; thr++){
      fsw->thr_cmd[thr] = thr_trq_cmd[thr]/fsw->thr_torque_mag[thr];
      if(fsw->thr_cmd[thr] > maxcmd){
        maxcmd = fsw->thr_cmd[thr];
      }
    }

    if(maxcmd > (1.0 + fsw->eps)){
      int r;
      for(r=0;r<fsw->num_thr;r++){
        fsw->thr_cmd[r] = fsw->thr_cmd[r]/maxcmd;
      }
    }
  }
  else{
      int r;
      for(r=0;r<fsw->num_thr;r++){
        fsw->thr_cmd[r] = 0.0;
      }
      printf("%s␣%f␣\n","THRUST␣ALLOCATION␣FAULT,␣norm_err␣=␣",norm_err);
  }

  //update estimated thruster torque for use in attitude determination
  int q;
  int ind;
  for(ind = 0; ind<3; ind++){
    fsw->torque_thr[ind] = 0.0;
  }
  for(q = 0;q<fsw->num_thr;q++){
    for(ind = 0; ind<3; ind++){
      fsw->torque_thr[ind] = fsw->torque_thr[ind] + fsw->thr_cmd[q]*fsw->thr_torque_mag[
    }
  }

  return(0);
}


```

```

void project(double err_vec[3],fswstate *fsw ,int *thruster,double *amount)
{
  double proj;
  double proj_temp;
  int thr;

  proj = 0.0;
  thr = 99;

  int i;
  for(i=0;i<fsw->num_thr;i++){
    dot(err_vec,fsw->thr_torque_uvecs[i],&proj_temp);
    if(proj_temp > proj){
      proj = proj_temp;
      thr = i;
    }
  }

  *amount = proj;
  *thruster = thr;

}
```

A.7 ADCS Flight Software Initialization

A.7.1 Listing of read_fswcmd (from TSATsim)

```
/* read_fswcmd.cc
   Reads fswcmd file containing fsw ground commands
   Written for TSat ADCS analysis
   Mark Van de Loo, Spring 2013
*/


#include <stdio.h>
#include <string.h>

#include "read_fswcmd.h"
#include "initialize_fsw.h"



int read_fswcmd(fswstate *fsw)
{

  char path[200], file[50];
  strcpy(path, fsw->inputpath);
  strcpy(file, "/fswcmd");
  strcat(path, file);
  FILE *fp;
  fp=fopen(path, "r");


  char temp_var [100];
  int temp_vali, temp_vali2;
  double temp_valf, temp_valf2,temp_valf3,temp_valf4,temp_valf5;

  temp_vali = 0;
  temp_valf = 0;
  temp_vali2 = 0;
  temp_valf2 = 0;
  temp_valf3 = 0;
  temp_valf4 = 0;
```

```
temp_valf5 = 0;

int ind = 0;

int done = 0;

while(done==0){
  if(fscanf(fp,"%s %lf %i %i %lf %lf %lf %lf %*[^\n]", temp_var, &temp_valf, &temp_va
            &temp_vali2, &temp_valf2, &temp_valf3, &temp_valf4, &temp_valf5)==EOF){
    done = 1;
    return(0);
  }

  (*fsw).ground_cmd_time[ind] = temp_valf;
  (*fsw).ground_cmd_num[ind] = temp_vali;
  (*fsw).ground_cmd_intarg[ind] = temp_vali2;
  (*fsw).ground_cmd_dubarg1[ind] = temp_valf2;
  (*fsw).ground_cmd_dubarg2[ind] = temp_valf3;
  (*fsw).ground_cmd_dubarg3[ind] = temp_valf4;
  (*fsw).ground_cmd_dubarg4[ind] = temp_valf5;

  ind ++;

  temp_vali = 0;
  temp_valf = 0;
  temp_vali2 = 0;
  temp_valf2 = 0;
  temp_valf3 = 0;
  temp_valf4 = 0;
  temp_valf5 = 0;


}

```

```
  fclose(fp);

  return(0);
}
```

A.7.2 Sample fswcmd file

```
thr_0_on        90000     1     0     0     0     0     0
thr_1_on        90000     1     1     0     0     0     0
thr_2_on        90000     1     2     0     0     0     0
thr_3_on        90000     1     3     0     0     0     0

att_quat        300       2     0     0.254290931636596       0.254290931636596
−0.888675218932228     0.284479523745629

att_quat        300000       2     0     −0.046940049092611       −0.35654507094671
0.766683976609875     0.531853291974819
```

A.8 FSW State Variable Data Structure

This section contains a listing of *initialize_fsw.h*, which defines the *fsw* state variable data structure. Variable descriptions and units are given in the comments.

```
#ifdef __cplusplus
extern "C" {
#endif

#ifndef INITIALIZE_FSW_H
#define INITIALIZE_FSW_H
```

```

#include <stdio.h>


typedef struct {
  const char * inputpath;//path to input file directory

  double eps;//fsw epsilon (for double comparisons etc.)

  int time_cntr;//fsw integer time counter - incremented once for each
                //fsw cycle
  double time;//fsw time (sec)
  double thr_cmd[8];//Commanded thrust level for each thruster as a
                    //fraction -  0.0=off to 1.0=full throttle

  int num_thr;//number of thrusters onboard spacecraft
  double cyc_period;//length of acs fsw cycle (sec)

  int max_gnd_cmd;//maximum number of ground
                                         //commands. Need to update size
                                         //of all ground_cmd variables if
                                         //you update this!!
  double ground_cmd_time[1000];//time to execute ground command (sec)
  int ground_cmd_num[1000];//command number of ground command
  int ground_cmd_intarg[1000];//integer argument of ground command
  double ground_cmd_dubarg1[1000];//double argument of ground command
  double ground_cmd_dubarg2[1000];//double argument of ground command
  double ground_cmd_dubarg3[1000];//double argument of ground command
  double ground_cmd_dubarg4[1000];//double argument of ground command
  int ground_cmd_executed[1000];//flag 1=command already
                                          //executed, 0=command not
                                          //executed yet.

  double B_magnetometer[3]; //Earth's magnetic field as measured by the
```

```
                                //magnetometer (Gauss)
    double B_actual[3]; //Unit vector containing the direction of Earth's magnetic field a
                        //IGRF shpherical harmonic model
    double sunsensor_raw[6]; //[solarpanel] output value of sunsensor to
                             //nanomind (microAmps)
    double gy_gyro_out [6];// gyro "gyro_out" register
    double pos_eci [3];//ECI position (m)
    double body_rates [3];//body rates as estimated (rad/sec)
    double body_rates_cmd [3];//commanded (desired) body rates (rad/sec)
    double att_quat [4];//Transformation Quaternion from ECI to SBF
    double att_quat_cmd [4];//Commanded (desired) Quaternion from ECI to SBF
    int num_panels; //number of solar panels
    double ss_thresh; //threshold value of the cosine reading above
                      //which a sun sensor is considered to see the sun.
    double sunvec_meas [3]; //Unit vector containing the direction of
                            //the sun in SBF as computed by sun sensor
                            //data processing.
    double sunvec_actual [3]; //Unit vector containing the calculated
                              //direction of the sun in ECI

    double igrf_g[14][14]; //igrf g coefficient from
                           //http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html
    double igrf_gdot[14][14]; //igrf gdot coefficient
    double igrf_h[14][14]; //igrf h coefficient
    double igrf_hdot[14][14]; //igrf hdot coefficient
    double jd_igrfdata; //julian date referring to the time IGRF data
                        //in the igrf input file was published
    double jd; //Current Julian Date (days)
    double jd0; //Initial Julian Date (days)
    double PI; //3.14159265359
    int valid_sunvec_meas; //binary flag =1 if there is a valid
                           //sunsensor measurement, =0 otherwise.
    double I [3][3];//spacecraft inertia matrix (kg*m^2)
    double K_p; //proportional control gain for torque control law
```

```
double K_i; //integral control gain for torque control law
double K_d; //derivative control gain for torque control law
double integ_term [3]; //integral term in control law

double torque_cmd[3]; //torque commanded by attitude controller in
                      //CMF (N*m)

double thr_alloc_tol; //tolerance on thrust allocation error (N*m)
int thr_alloc_maxiter; //maximum iterations allowed during thrust allocation
double thr_torque_uvecs[8][3]; //matrix with each row containg a unit vector
//in the direction of the torque
//applied by the corresponding thruster (CMF).
double thr_torque_mag[8]; //magnitudes of torques applied by each
//thruster running at full throttle.
//(N*m) CMF
double torque_thr[3]; //predicted actual torque applied by
                          //thrusters in CMF (N*m)
double max_thrust; //thrust produced by a single thruster running at
                   //full throttle. (Newtons)
double M_tot[3]; //Total moment applied to spacecraft by actuators
                 //and disturbances. (N*m)
double Q_kf[7][7]; //Kalman filter estimate covariance
double W_kf[7][7]; //Kalman filter process noise covariance
double R_kf[7][7]; //Kalman filter sensor noise covariance
double body_rates_meas [3];//body rates measured by the gyros (rad/sec)
double att_quat_meas [4];//ECI to SBF attitude quaternion measured by the
//gnc_TRIAD algorithm
int use_attitude_kf;//1 = use KF, 0 = just treat measurements
                          //as truth.
double k_kf;//k value for UKF
double alpha_kf;//k value for UKF
double beta_kf;//k value for UKF
int kf_version;// 1 = normal, 2 = complicated
double sig_pts[7][14];
```

```
} fswstate;




void initialize_fsw(fswstate*);



#endif

#ifdef __cplusplus
}
#endif
```

Appendix B

Simulation Truth State Data Structure

The truth state data structure from the truth side of the TSATsim high fidelity simulation is defined in *initialize_rw.h*, listed here. The comments in the file contain variable descriptions and units.

```
#ifndef INITIALIZE_RW_H
#define INITIALIZE_RW_H

#include <stdio.h>

typedef struct{
  static const double eps = 1e-12;

  int time_cntr;//integer time counter incremented with each sim step
  double time;//realworld time (sec)
  double dt;//time increment since last time through main loop  (sec)
  double time_last;//realworld time of last run through main loop  (sec)
  double pos_eci [3];//ECI position (m)
  double vel_eci [3];//ECI velocity (m)
```

```
    double body_rates [3];//body rates (rad/sec)
    double att_quat [4];//Transformation Quaternion from ECI to SBF
    double m;//total spacecraft mass (kg)
    double I [3][3];//spacecraft inertia matrix (kg*m^2)
    double cm_loc [3];//location of spacecraft center of mass in SBF (m)
    double side_length;//length of a side of the cube, used for aero
                       //calcs (m)
    double H [3];//angular momentum of spacecraft (kg*m^2/s)
    double M_tot [3];//sum of all moments acting on spacecraft (N*m)
    double gy_new_data [6];//gyro "new_data" register
    double gy_alarm [6];// gyro "alarm" register
    double gy_gyro_out [6];// gyro "gyro_out" register
    double M_thr [8][3];// [thruster][x/y/z] Moment produced by each
                        // thruster (N*m)
    double F_thr [8][3];//[thruster][x/y/z] Force produced by each
                        //thruster(N)
    double thr_mag [8];//Magnitude of the force produced by each
                       //thruster (N)
    double M_coil [4][3];//[coil][x/y/z] moment produced by each torque
                         //coil (N*m)
    double F_pin [3];// Force applied to spacecraft by testbed pin (N)
    double M_pin [3];// Moment applied to spacecraft by testbed pin (N*m)
    double M_dist [3];// Moment applied to spacecraft by disturbances
                      // (N*M) ECI
    double F_dist [3];// Force applied to spacecraft by disturbances (N) ECI
    double F_grav [3];// Force applied to spacecraft by gravity (N) ECI
    double M_grav [3];// Moment applied to spacecraft by gravity (N) ECI
    double F_aero [3];// Force applied to spacecraft by aero drag (N) ECI
    double M_aero [3];// Moment applied to spacecraft by aero drag (N) ECI

    double gy_internal_sample_period; //Internal sample period of gyros (sec)
    double gy_noise_rms;//RMS value of gyro noise (rad/sec)
    int gy_time_cntr_last [6];//Time counter value of last gyro update
    double gy_rate_register [6];//TBD
```

```
int num_gyros;//number of gyros
int num_thr;//number of thrusters
int num_panels; //number of solar panels
double thr_loc [8][3];//[thruster][x/y/z] location of each thruster
                      //in SBF (m)
double max_thrust;//magnitude of maximum thrust in (N)
double num_coil;//number of torque coils
double dry_mass;//dry mass of spacecraft (kg)
double mass_fuel;//mass of fuel (kg)
double sbf_2_gyro [6][3][3];//[gyro][rows][columns] transformation
                            //matrix from SBF to gyro frame
double sbf_2_panel [6][3][3];//[solarpanel][rows][columns] transformation
                            //matrix from SBF to solar panel frame
double sbf_2_thr [8][3][3];//[thruster][rows][columns]
                           //transformation matrix from SBF to
                           //thruster frame
double sbf_2_magnetometer [3][3];//[rows][columns]
                           //transformation matrix from SBF to
                           //magnetometer frame
double mu;//gravitational parameter (m^3/sec^2)
double J2;//J2 gravity coeff
double G;//gravitational constant (m-k-s units)
double Me;//mass of the earth (kg)
double Re;//radius of the earth (m)
double pin_loc [3];//location of the testbed pin in sbf (m)
double Cd;//drag coefficient
double rho_atm;// density of the atmosphere (kg/m^3)
double w_earth; //angular velocity of the earth (rad/sec)

double F_actuators[3]; //total force applied by spacecraft actuators
                       //(N) in SBF
double M_actuators[3];// total moment applied by spacecraft
                      // actuators (N*m) in SBF
double F_environment[3];//total force applied on spacecraft by
```

```
                            //environment (N) in ECI
    double M_environment[3];//total moment applied on spacecraft by
                            //environment (N) in ECI

    double igrf_g[14][14]; //igrf g coefficient from
                           //http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html
    double igrf_gdot[14][14]; //igrf gdot coefficient
    double igrf_h[14][14]; //igrf h coefficient
    double igrf_hdot[14][14]; //igrf hdot coefficient
    double jd_igrfdata; //julian date referring to the time IGRF data
                        //in the igrf input file was published
    double jd; //Current Julian Date (days)
    double jd0;// Julian Date at sim initialization (days)

    double B[3]; //Earth's magnetic field observed by the spacecraft in
    //ECI (nanoTesla)
    double B_magnetometer[3]; //Earth's magnetic field as measured by the
                              //magnetometer (Gauss)
    double sunvec[3]; //Position vector of the Sun in ECI (meters)
    double sunvec_sbf[4]; //Unit vector pointing from SBF origin to the
                          //sun in SBF coordinates

    double ss_noise_std; //standard deviation of sun sensor noise (radians)
    double ss_noise_max; //max value of sun sensor noise (radians)
    double sunsensor_angle[6]; //[solarpanel] angle to sun measured by
                               //sun sensor on each panel (radians)
    double sunsensor_raw[6]; //[solarpanel] output value of sunsensor to
                             //nanomind (microAmps)
    int eclipse; //eclipse flag: 1=full or partial eclipse, 0 = direct
                 //line of sight to the sun
    double mag_internal_sample_period; //Internal sample period of magnetometer (sec)
    double mag_noise_std;//standard deviation of magnetometer noise (rad/sec)
    int mag_time_cntr_last;//Time counter value of last magnetometer update

```

```
} realworld;


void initialize_rw(realworld&);



#endif
```

References

[1] Harold D. Black. A passive system for determining the attitude of a satellite. *AIAA Journal*, 1964.

[2] William Blackwell, G Allen, C Galbraith, R Leslie, I Osaretin, M Scarito, Mike Shields, E Thompson, D Toher, D Townzen, et al. Micromas: A first step towards a nanosatellite constellation for global storm observation. *Proceedings of the AIAA/USU Conference on Small Satellites*, 2013.

[3] J. Bouwmeester and J. Guo. Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology. *Acta Astronautica*, 67(78):854 – 862, 2010.

[4] Daniel George Courney. *Ionic Liquid Ion Source Emitter Arrays Fabricated on Bulk Porous Substrates for Spacecraft Propulsion*. PhD thesis, Massachusetts Institute of Technology, 2011.

[5] GomSpace. Gomspace nanopower solar panels. `http://gomspace.com/index.php?p=products-p110`.

[6] Henri Kjellberg and E Glenn Lightsey. A constrained attitude control module for small satellites. *Proceedings of the AIAA/USU Conference on Small Satellites*, 2012.

[7] Mary Knapp, Mark Van de Loo, Alessandra Babuscia, Anna Walsh, and Sara Seager. Tsat preliminary design review and status update. June 2013.

[8] Ern J Lefferts, F Landis Markley, and Malcolm D Shuster. Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429, 1982.

[9] Surrey Satellite Technology Ltd. Sgr-05u gps receiver user interface manual. Feb 2011.

[10] A. Marinan, A. Nicholas, and K. Cahoy. Ad hoc cubesat constellations: Secondary launch coverage and distribution. In *Aerospace Conference, 2013 IEEE*, pages 1–15, 2013.

[11] Francois Martel, Louis Perna, and Paulo Lozano. Miniature ion electrospray thruster and performance test on cubesats. *Proceedings of the AIAA/USU Conference on Small Satellites*, 2012.

[12] Inc. Maryland Aerospace. Mai-400 product description. `http://www.miniadacs.com/miniadacs_012.htm`.

[13] Juergen Mueller, Richard Hofer, and John Ziemer. Survey of propulsion technologies applicable to cubesats. *Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration*, 2010.

[14] International Association of Geomagnetism and Aeronomy. 11th generation international geomagnetic reference field schmidt semi-normalised spherical harmonic coefficients. `http://www.ngdc.noaa.gov/IAGA/vmod/igrf11coeffs.txt`, 2011.

[15] Computer Sciences Corporation. Attitude Systems Operation and J.R. Wertz. *Spacecraft Attitude Determination and Control.* Astrophysics and Space Science Library : a series of books on the recent developments of space science and of general geophysics and astrophysics. Reidel, 1978.

[16] Louis Perna, Fernando Mier Hicks, Chase S. Coffman, Hanqing Li, and Paulo C. Lozano. Progress toward demonstration of remote autonomous attitude control of a cubesat using ion electrospray propulsion systems. *AIAA/ADSME/SAE/ASEE Joint Propulsion Conference & Exhibit*, 2012.

[17] Christopher Pong, Matthew W Knutson, David W Miller, Sara Seager, Sungyung Lim, Timothy C Henderson, and Shawn D Murphy. High-precision pointing and attitude determination and control on exoplanetsat. In *AIAA. Minneapolis, MN: AIAA Guidance, Navigation, and Control Conference, Aug*, 2012.

[18] The CubeSat Project. Cubesat mission statement. `http://www.cubesat.org/index.php/about-us/mission-statement`, 2013.

[19] Stergios I Roumeliotis, Gaurav S Sukhatme, and George A Bekey. Circumventing dynamic modeling: Evaluation of the error-state kalman filter applied to mobile robot localization. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1656–1663. IEEE, 1999.

[20] Rainer Sandau, Rei Kawashima, Shinichi Nakasuka, and Jerry Jon Sellers. *Innovative Ideas for Micro/Nano-Satellite Missions.* International Academy of Astronautics Book Series. International Academy of Astronautics, 2013.

[21] Daniel Selva and David Krejci. A survey and assessment of the capabilities of cubesats for earth observation. *Acta Astronautica*, 74(0):50 – 68, 2012.

[22] Malcolm D. Shuster. The triad algorithm as maximum likelihood estimation. *The Journal of the Astronautical Sciences*, 2006.

[23] M.J. Sidi. *Spacecraft Dynamics and Control: A Practical Engineering Approach.* Cambridge Aerospace Series, 7. Cambridge University Press, 1997.

[24] Silonex. Slcd-61n8 solderable planar photodiode data sheet.

[25] Garrett Lee Skrobot and Roland Coelho. ELaNa - Educational Launch of Nanosatellite. In *Small Satellite Conference*, 2012.

[26] California Polytechnic State University. Cubesat design specification, revision 12. `http://www.cubesat.org/index.php/documents/developers`, 2009.

[27] David A Vallado. *Fundamentals of astrodynamics and applications*, volume 12. Springer, 2001.

[28] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. IEEE, 2000.

[29] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter. *Kalman filtering and neural networks*, pages 221–280, 2001.