Development and Analysis of a Small Satellite Attitude Determination and Control System Testbed

by

Corey Whitcomb Crowell

B.S., Astronautical Engineering (2009)
United States Air Force Academy

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 2011

Author .....
Department of Aeronautics and Astronautics
May 16, 2011

Certified by .....
David W. Miller
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Eytan H. Modiano
Associate Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Development and Analysis of a Small Satellite Attitude Determination and Control System Testbed

by

Corey Whitcomb Crowell

Submitted to the Department of Aeronautics and Astronautics
on May 16, 2011, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Attitude Determination and Control Systems (ADCS) are critical to the operation of satellites that require attitude knowledge and/or attitude control to achieve mission success. Furthermore, ADCS systems only operate as designed in the reduced friction, micro-gravity environment of space. Simulating these characteristics of space in a laboratory environment in order to test individual ADCS components and integrated ADCS systems is an important but challenging step in verifying and validating a satellite's ADCS design.

The purpose of this thesis is to design and develop an ADCS testbed capable of simulating the reduced fiction, micro-gravity environment of space within the Massachusetts Institute of Technology's Space Systems Laboratory. The ADCS testbed is based on a tabletop style, three degree of freedom, rotational air bearing, which uses four reaction wheels for attitude control and a series of sensors for attitude determination. The testbed includes all the equipment necessary to allow for closed loop testing of individual ADCS components and integrated ADCS systems in the simulated inertial environment of space. In addition to the physical ADCS testbed, a MATLAB Simulink based model of the ADCS testbed is developed to predict the performance of hardware components and software algorithms before the components and algorithms are integrated into the ADCS testbed. The final objective of this thesis is to validate the operation of the ADCS testbed and simulation to prepare the tool for use by satellite design teams.

Thesis Supervisor: David W. Miller
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I have many to thank for the opportunity to complete this thesis as part of MIT's Masters program. I would like to begin with my family. My dad has always supported me and any decisions I have made for myself. He provided me with everything I needed and gave me the freedom to choose my own way. He was and still is an excellent role model, and I am who I am today because I strive to follow his example. My mom has also supported my every step in life. She worked harder than anyone I know just to provide my sister and I with a place to come home to and great food! Her love and support has been a constant in my life, and I am happy to know it always will be. My sister is to thank for many things in my life as well; first, she helped to toughen me up as a child (to include several sets of stitches!), and more recently she and her family have shown me the blessings that starting one's own family can bring. I look forward to having my own family someday, and I hope to be as good a parent as she and her husband are. I must also thank my fiancé Lauren who has supported me throughout my graduate school experience. She has celebrated my accomplishments and supported me through the many trials of the program.

To my thesis advisor, Professor David Miller, thank you for the support over the last two years. Your advise and suggestions regarding my research (some optional, some required!) have kept me on track and helped me accomplish things I would have thought impossible for myself prior to this program. The opportunity to study at MIT and work as a part of the Space Systems Laboratory will always be a highlight of my career regardless of where I go from here, and I am very thankful for it.

I must also thank those who contributed to my education. My instructors at the United States Air Force Academy helped me to realize my potential as a student, an engineer, and an Air Force officer. The Academy and those who define it are to thank for many wonderful opportunities in my life to include the chance to study at MIT. Finally, I thank the teachers at the beginning of my educational career who showed me that hard work and determination are key to achieving one's goals. Though almost all of my teachers are to thank, Carol Beasley was the first to push me to the limits of my own abilities and then help me expand them. Soon after, Jerry Thomas helped me to build a foundation in mathematics and engineering that I have since relied upon in my educational career.

To those above and the many others who have been influential to me, thank you.

Contents

List of Figures

List of Tables

List of Acronyms

AAR Auxiliary Arduino

AB Air Bearing

ABBF Air Bearing Body Fixed

ADCS Attitude Determination and Control System

ADIS Analog Devices Inertial Sensor

AFIT Air Force Institute of Technology

AFRL Air Force Research Laboratory

BRMS Bifocal Relay Mirror Spacecraft

CASTOR Cathode/Anode Satellite Thruster for Orbital Repositioning

CM Center of Mass

CMG Control Moment Gyroscope

CONOPs Concept of Operations

CUSat Cornell University Satellite

DANDE Drag and Atmospheric Neutral Density Explorer

DC Direct Current

DCFT Diverging Cusped Field Thruster

DCM Direction Cosine Matrix

DIO Digital Input/Output

DoD Department of Defense

DSACSS Distributed Spacecraft Attitude Control System Simulator

EAPS Earth, Atmospheric, and Planetary Sciences

EELV Evolved Expendable Launch Vehicle

EKF Extended Kalman Filter

EMFG External Magnetic Field Generator

EMIC Electromagnetic Ion Cyclotron

ESPA EELV Secondary Payload Adapter

FASTSAT Fast, Affordable, Science and Technology Satellite

FIR Fixed Inertial Reference

FUSE Far Ultraviolet Spectroscopic Explorer

GPS Global Positioning System

IC Initial Condition

I^2C Inter-Integrated Circuit

IDE Integrated Development Environment

IMAGE Imager for Magnetopause to Aurora Global Exploration

IMU Inertial Measurement Unit

ISS International Space Station

JPL Jet Propulsion Laboratory

KLC Kodiak Launch Complex

LED Light-Emitting Diode

LEO Low Earth Orbit

LQE Linear Quadratic Estimator

LQR Linear Quadratic Regulator

LVLH Local Vertical, Local Horizontal

MAR Main Arduino

MicroMAS Micro-sized Microwave Atmospheric Satellite

MISO Master In Slave Out

MIT Massachusetts Institute of Technology

MOSI Master Out Slave In

MPA Multi-Payload Adapter

NASA National Aeronautics and Space Administration

NFOV Narrow Field-of-View

NiMH Nickel-Metal Hydride

O/OREOS Organism/Organic Exposure to Orbital Stresses

ODTML Ocean Data Telemetry MicroSatLink

P-Pod Poly-PicoSatellite Orbital Deployer

Polar BEAR Polar Beacon Experiment and Auroral Research

PPR Pulse Per Revolution

PPT Pulsed-Plasma Thruster

PSI Pounds per Square Inch

RAX Radio Aurora Explorer

RPM Revolutions Per Minute

RTC Real Time Clock

RW Reaction Wheel

RWCS Reaction Wheel Coordinate System

SCLK Serial Clock

SDA Serial Data

SIV Standard Interface Vehicle

SNAP-1 Surrey Nanosatellite Applications Platform

SPEX Space Phenomenology Experiment

SPHERES Synchronized Position Hold, Engage, Reorient, Experimental Satellites

SPI Serial Peripheral Interface

SS Slave Select

SSL Space Systems Laboratory

SSTL Surrey Satellite Technology Ltd.

STP Space Test Program

TACT Triaxial Attitude Control Testbed

TERRIERS Tomographic Experiment using Radiative Recombinative Ionospheric Extreme ultraviolet and Radio Sources

TERSat Tethered Environmental Reconditioning Satellite

TIMED Thermosphere Ionosphere and Mesosphere Energetic and Dynamics

TOMS-EP Total Ozone Mapping Spectrometer - Earth Probe

TRL Technology Readiness Level

TTL Transistor-Transistor Logic

UART Universal Asynchronous Receiver/Transmitter

UHF Ultra High Frequency

UNP University Nanosat Program

USAFA United States Air Force Academy

USB Universal Serial Bus

WFOV Wide Field-of-View

Chapter 1

Introduction

1.1 Problem Statement

Developing and testing attitude determination and control systems (ADCS) in a university environment is a challenging task. Testing hardware in the loop ADCS systems in a laboratory is difficult due to the fact that ADCS subsystems often rely on the micro-gravity, reduced friction environment of space to perform as designed. Equipment needed to simulate the key characteristics of the space environment for ADCS testing has been previously designed and built by organizations for internal use, but such equipment is not available for purchase. However, without hardware in the loop testing in a simulated micro-gravity, reduced friction environment, ADCS engineers must rely solely on isolated component testing and simulation to validate the operation of the integrated ADCS subsystem. In this case, the subsystem will not be fully operational until it is on orbit where design and development errors likely cannot be corrected and can easily cause mission failure.

1.2 Motivation

Designing and developing an ADCS testbed for use within the Massachusetts Institute of Technology's Space Systems Laboratory (MIT's SSL) will provide student satellite design engineers the means to test ADCS hardware components, software algorithms, and integrated ADCS subsystems in a simulated micro-gravity, reduced friction environment. Many students at the undergraduate and even graduate level have no experience in satellite design, much less ADCS subsystem design. Therefore, providing these students with a means

to validate hardware in the loop ADCS systems after conceptual designs and simulations have been produced is critical to satellite mission assurance. Student engineers as well as university faculty will be able to gain confidence in a given ADCS subsystem once "test as you fly" results are available to verify the ADCS system's simulation results.

1.3 Thesis Objectives

- Design and develop a small satellite ADCS testbed capable of meeting the following requirements (further described in Section 2.5).
    - Provide class project support for estimation and control theory students.
    - Provide ADCS hardware component testing capability.
    - Provide ADCS estimation and control algorithm testing capability.
    - Provide a platform for integrated ADCS subsystem testing.
- Verify the individual hardware components and baseline software used in the ADCS testbed design.
- Validate the integrated ADCS Testbed and the supporting MATLAB based simulation.

1.4 Thesis Outline

- Chapter 2 covers the current state of satellite ADCS systems to include those of several small satellites being developed within the SSL. The chapter also discusses recent on-orbit ADCS failures. The chapter then covers the current state of ADCS testbed technology and concludes with the design requirements for the testbed developed as part of this thesis.
- Chapter 3 discusses the development of the rotational air bearing simulation. First, the chapter defines the coordinate systems used by the simulation and physical air bearing. The chapter then develops the air bearing's equations of motion. Finally, the chapter discusses the individual sections of the air bearing simulation, which include the reaction wheel and air bearing plant, the attitude estimation module, the attitude command module, and the attitude control module.

- Chapter 4 discusses the design and development of the physical air bearing to include the air bearing's structure, the power system, the avionics and communication system, the baseline set of attitude determination sensors, the baseline reaction wheel configuration, and the ground station computer. The chapter then covers the default software algorithms used by the air bearing's on-board processors. The chapter concludes by discussing the air bearing's external magnetic field generator (EMFG) and the benefits of integrating a SPHERES satellite on the air bearing.

- Chapter 5 covers the analysis of the air bearing and accompanying simulation. First, the chapter covers attitude sensor characterization. The chapter then discusses reaction wheel response characterization. Air bearing disturbances like center of mass misalignment, compressed air vibration, and friction are discussed, and the chapter concludes by analyzing the results of several integrated system tests.

- Chapter 6 summarizes the thesis and discusses future work to expand the capabilities of the ADCS testbed.

Chapter 2

Background

2.1 Attitude Determination and Control System Overview

A satellite's ADCS system is used to stabilize and orient the vehicle as required by the Concept of Operations (CONOPs) in the presence of external disturbance torques acting on the satellite. The ADCS system uses external references to determine the satellite's angular orientation with respect to a fixed inertial reference frame, usually an Earth centered, equatorial frame for Earth orbiting satellites. External attitude references include the Sun, Earth's horizon, the local magnetic field, and the stars. The satellite may also use inertial sensors like angular rate gyroscopes to measure angular rate and estimate the satellite's angular orientation between fixed inertial reference measurements, or while fixed inertial reference measurements are unavailable. For example, sun sensors cannot provide a fixed inertial reference measurement while the satellite is in eclipse [52].

For attitude control, ADCS systems impose torques on the satellite using a series of actuators. Thrusters of all shapes, sizes, and operating characteristics are used to produce external torques for attitude control, but they require an expendable fuel source. Magnetic torque coils and rods are also used to produce external torques on the satellite by acting against Earth's local magnetic field. Reaction wheels are common satellite attitude actuators because they produce internal torques that do not change the angular momentum of the satellite.

The ADCS system must maintain attitude control in the presence of constant disturbance torques on the satellite. Disturbance torques are a function of the inertial properties of a satellite and its orbital location. For small satellites in Low Earth Orbit (LEO), the

most common disturbance torques are caused by solar radiation pressure, interaction with Earth's local magnetic field, aerodynamic drag due to Earth's atmosphere, and gravity-gradient torque. These disturbances exert external torques, which build up angular momentum within the satellite. Reaction wheels may be used to maintain satellite stability and angular orientation in the presence of disturbance torques, but they can only store angular momentum and cannot dissipate momentum. When the reaction wheels are near their maximum angular momentum storage capability, external torques must be applied using thrusters or magnetic torquers to reduce the angular momentum stored in the wheels. This process is commonly referred to as desaturating the reaction wheels [52]

Satellite ADCS systems are designed to meet the requirements of their parent satellite making each ADCS system unique. Large satellites require attitude control actuators with large torque capabilities. Small satellites require actuators with less torque capability. Precision payloads may require ADCS systems capable of determining and controlling attitude to within an arc-second (1/3600 of a degree), while some payloads may only require attitude control to within twenty degrees, or possibly no attitude control at all. ADCS systems must be custom designed and built for each satellite program with mission success often relying on the flawless operation of the ADCS system.

2.2 State of Small Satellite ADCS Systems

2.2.1 First Three Axis Stabilized Small Satellite

The first three axis stabilized small satellite was the Surrey Nanosatellite Applications Platform (SNAP-1) developed and built by Surrey Satellite Technology Ltd. (SSTL). SNAP-1 was launched on 28 June, 2000 on-board a Russian Cosmos rocket [51]. The primary mission of the 6.5 kilogram satellite was to demonstrate three axis attitude control as well as orbital maneuvering [51]. The SNAP-1 satellite used a three axis magnetometer, an on-board camera, and a Kalman Filter for attitude estimation, and three magnetic torque rods in conjunction with a single pitch axis momentum wheel for attitude control [51]. SNAP-1 was also the first small satellite to use the Global Positioning System (GPS) for orbital position acquisition [51]. SNAP-1 was able to successfully control its attitude and use a butane fueled cold gas thruster to maneuver itself to within 2000 kilometers of its target satellite after starting more than 15000 kilometers away [51]. Though SNAP-1 did not rendezvous

with the target satellite, it proved that small satellites can achieve three axis stability and autonomously perform complex orbital maneuvers [51].

2.2.2 Space Test Program S26 Payloads

The current state of the art for small satellite ADCS systems is well represented by the payloads launched as part of the most recent Space Test Program (STP) mission. The mission, known as STP-S26 launched from Kodiak Launch Complex (KLC) in Kodiak, Alaska on 19 November, 2010 [?]. A converted intercontinental ballistic missile known as a Minotaur IV was the launch vehicle used for the mission [?]. The Minotaur IV carried seven satellites to orbit; four EELV Secondary Payload Adapter (ESPA) class satellites and three cubesats [?]. The four ESPA class satellites attached to the launch vehicle using the Minotaur IV Multi-payload Adapter (MPA) in its first ever flight [?]. Two of the three cubesats were attached directly to the launch vehicle using Poly-PicoSatellite Orbital Deployers (P-Pods) [?]. The remaining cubesat, NASA's Nanosail D cubesat was integrated into the FASTSAT ESPA class satellite. The remaining five satellites making up the STP-S26 mission include FalconSat-5, FASTRAC, STPSat-2, O/OREOS, and RAX [?]. Figure 2-1 shows the four ESPA class satellites integrated onto the MPA.

FalconSat-5 is the fifth satellite built by the Astronautical Engineering Department at the United States Air Force Academy (USAFA) and the second ESPA class satellite. The satellite is carrying four payloads, two of which are SERB ranked [37]. The primary mission of FalconSat-5 is to use its payloads to collect space weather measurements [37]. In order to operate its payloads as required, the satellite must achieve and maintain three axis stability. FalconSat-5 is equipped with four sun sensors, a three axis magnetometer, and an inertial measurement unit capable of measuring three axis angular rate. Using these sensors, FalconSat-5 estimates its attitude and uses three orthogonal reaction wheels for primary attitude control. Though three orthogonal reaction wheels allow for a fully controllable system, they provide zero redundancy. For reaction wheel desaturation, FalconSat-5 uses three orthogonal magnetic torque rods [37]. FalconSat-5's attitude estimation sensors and attitude control actuators represent a common ADCS configuration amongst current small satellites; a fully observable, fully controllable, zero redundancy system. FalconSat-5 is the far right satellite seen in Figure 2-1.

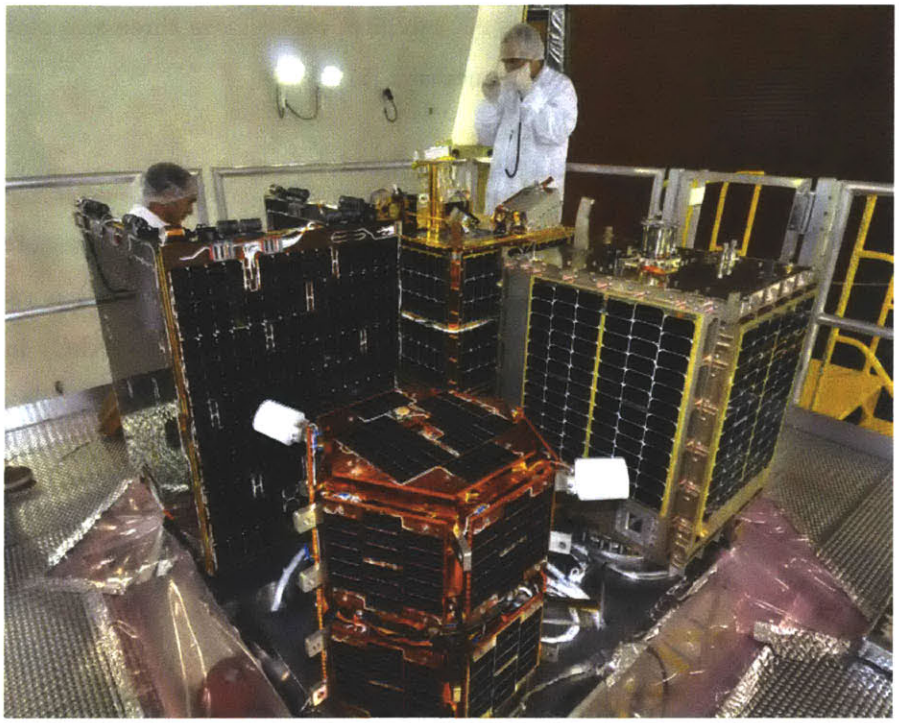


Figure 2-1: ESPA Class Satellites Mounted on Minotaur IV MPA For Launch [31]

FASTRAC is an ESPA class satellite designed and built by the University of Texas Nanosatellite Program at UT Austin [11]. The mission of FASTRAC is threefold. First, the satellite will demonstrate "on-orbit real-time GPS relative navigation solution[s] via real-time crosslink data exchange" [11]. Second, the satellite will demonstrate "on-orbit real-time attitude determination using a single frequency, C/A-code, reprogrammable GPS receiver" [11]. Finally, the satellite will demonstrate the use of a "micro-discharge plasma thruster" [11]. FASTRAC is made up of two sections that will separate in space [11]. Both require three axis attitude determination as a mission objective of FASTRAC, and they will each accomplish this using a single GPS receiver in combination with a three axis magnetometer [11]. Once separated, the two sections will attempt to control relative distance, though attitude is not controlled on either section [27]. One section carries a thruster that will be fired when the satellite's attitude is estimated to be within fifteen degrees of the desired thrust direction [11]. The other satellite carries an inertial measurement unit that will be used to estimate relative distance between the two sections [27]. FASTRAC's innovative means of attitude determination will provide additional attitude sensing capability to

future satellites with little additional cost since many satellites already carry GPS receivers and magnetometers. FASTRAC is the front satellite seen in Figure 2-1.

NASA's Fast, Affordable, Science and Technology Satellite (FASTSAT) is an ESPA class satellite designed and built at the Marshall Space Flight Center in Huntsville, Alabama [2]. The mission of FASTSAT is "to demonstrate the capability to build, design and test a microsatellite platform to enable governmental, academic and industry researchers to conduct low-cost scientific and technology experiments on an autonomous satellite in space" [2]. Though FASTSAT's mission objective is to be a cheap, quick, easy to develop spacecraft, the FASTSAT launched on STP-S26 carried several unique payloads, which include the first Nanosail to be deployed in low Earth orbit, a miniature star tracker, a thermosphere temperature imager, and a miniature imager for neutral Ionospheric atoms and Magnetospheric electrons [2]. Though the payloads will differ for each FASTSAT, the satellite bus will be primarily the same. The bus will be three axis stabilized with attitude estimation capability of 0.02 degrees and attitude control capability of two degrees [12]. FASTSAT is shown furthest back in Figure 2-1.

STPSat-2 is an ESPA class satellite built by Ball Aerospace [7]. STPSat-2 carries two payloads; SPEX (Space Phenomenology Experiment) will "evaluate sensor compatibility for the space environment" and ODTML (Ocean Data Telemetry MicroSatLink) will provide "two way data relay from terrestrial sensors to users" [14]. STPSat-2 is the first satellite to use STP's Standard Interface Vehicle (SIV), which uses a common satellite bus and payload integration system to allow for quick satellite design and build [7]. The satellite bus used by the SIV is ComTech AeroAstro's Astro-200 satellite bus, which provides all of the subsystems required to support the satellite's payloads [9]. The Astro-200 satellite bus carries the satellite's ADCS system, which uses three orthogonal reaction wheels to provide three axis stabilization [9]. The Astro-200 bus is designed to maintain Nadir pointing and can control its attitude to within 0.1 degrees. STPSat-2 is shown on the far left in Figure 2-1.

NASA's second payload on STP-S26 is a three unit cubesat named Organism/Organic Exposure to Orbital Stresses (O/OREOS) Nanosatellite, which was designed and built at the Ames Research Center located at Moffett Field, California [3]. The mission of O/OREOS is to "characterize the growth, activity, health and ability of microorganisms to adapt to the stresses of the space environment" [3]. The cubesat has no active ADCS system, but uses several passive attitude control components to manage angular rates. Permanent magnets

known as hysteresis rods are mounted to the satellite's structure, which will eventually align the satellite with Earth's magnetic field [13]. Since Earth's magnetic field changes slowly with a sinusoidal component having a period equal to the satellite's orbital period, alignment with the magnetic field will be a relatively stable state.

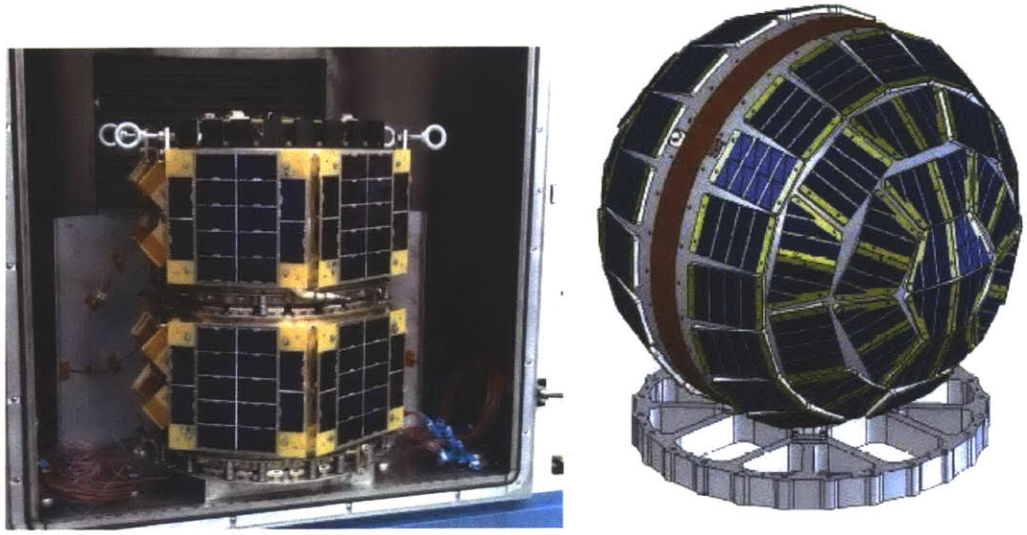
The Radio Aurora Explorer (RAX) satellite is a three unit cubesat designed and built by the University of Michigan and SRI International [18]. The mission of RAX is to "study formations and distribution of magnetic field-aligned plasma irregularities located in the lower ionosphere" [18]. To do this, RAX will carry a UHF radar receiver which will measure "dense plasma structures forming between E and F layers of the ionosphere" [18]. RAX will estimate its attitude using a three axis magnetometer, a three axis inertial measurement unit, and a set of sun sensors [18]. RAX will not be actively controlled, but will use passive magnetic stabilization just like O/OREOS [18]. RAX will align itself with Earth's magnetic field for attitude stability.

2.2.3 University Nanosat Program

The University Nanosat Program is an Air Force Research Laboratory (AFRL) sponsored program that supports the development of small satellites in university environments across the nation. The University Nanosat Program (UNP) sponsors roughly ten to twelve schools in the development of ESPA class satellites over a two year period. The UNP program provides financial support to each university over the two year development period, and will sponsor the winning university satellite from each UNP cycle for launch [15]. The UNP program is currently in the initial phases of its seventh two year satellite development cycle. The winners of previous UNP cycles are a good representation for state of the art ESPA class satellites. The first to be discussed is the winner of the third UNP cycle. The winning satellite of UNP 3 is FASTRAC, which was launched in November, 2010 as part of STP-S26 as discussed above. The winning satellites from UNP 4, 5, and 6 will be briefly discussed below.

The winning satellite from UNP 4 is the Cornell University Satellite (CUSat), which is an ESPA class satellite designed and built at Cornell University [43]. The mission of CUSat is to demonstrate formation flying capability precise enough to diagnose the structural health and configuration of another satellite after orbital rendezvous [43]. CUSat is made of up two functionally identical spacecraft that will separate from each other after being ejected

from the launch vehicle [43]. Using differential GPS with three separate GPS antennas, each satellite section will be able to estimate its attitude to within two degrees as well as estimate its orbital position to within a few meters [43]. Each satellite half will then use pulsed-plasma thrusters (PPTs), torque coils, and miniature reaction wheels for three axis attitude control and orbital maneuvering [43]. An image of CUSat in a thermal chamber is shown in Figure 2-2(a).



(a) CUSat in Thermal Chamber (b) Computer Image of DANDE

Figure 2-2: UNP 4 and UNP 5 Winning Satellites [15]

The winner of UNP 5 is the Drag and Atmospheric Neutral Density Explorer (DANDE) satellite, which is an ESPA class satellite designed and built by the University of Colorado [53]. DANDE's mission is to "explore the spatial and temporal variability of the neutral thermosphere at altitudes of 350 - 100 kilometers, and investigate how wind and density variability over 500 - 3000 kilometers scales translate to drag forces on satellites" [1]. For attitude estimation, DANDE will use a three axis magnetometer in combination with two horizon crossing indicators (HCIs) [53]. DANDE will use two magnetic torque rods for active attitude control with partial controllability [53]. The satellite will also use viscous fluid nutation dampers for passive nutation damping [53]. Figure 2-2(b) gives a computer generated image of the DANDE satellite.

The winner of the most recent UNP competition, UNP 6 is the Oculus satellite, which is an ESPA class satellite designed and built by Michigan Technological University [24]. The mission of the Oculus spacecraft is to "demonstrate vision-based attitude control for tracking resident space objects" [24]. Due to the satellite's mission, it requires inertial and vision based attitude control [24]. For inertial attitude estimation, the Oculus satellite uses a three axis magnetometer and a three axis rate gyroscope [24]. For vision based attitude control, Oculus uses two cameras; the wide field-of-view (WFOV) camera for general target location and the narrow field-of-view (NFOV) camera for target tracking [24]. Oculus uses three orthogonal reaction wheels for attitude control, and three orthogonal magnetic torque rods for reaction wheel angular momentum desaturation [24].

2.2.4 MIT Space Systems Laboratory Satellites

MIT's Space Systems Laboratory develops space hardware for operation within the Space Shuttle, the International Space Station (ISS), and as free flying satellite systems. The SSL has had several payloads fly on the Space Shuttle and currently operates three SPHERES satellites on the ISS. The SSL's free flying satellite program includes a series of small satellites in various stages of the satellite design process. This section will cover three of the SSL's small satellites, which are CASTOR, ExoPlanetSat, and TERSat.

Cathode/Anode Satellite Thruster for Orbital Repositioning (CASTOR) is an ESPA class satellite designed by undergraduate and graduate students within the SSL. CASTOR was designed as part of the UNP 6 small satellite competition. CASTOR's mission is to "validate the performance and application of Diverging Cusped Field Thruster (DCFT) technology" [8]. A DCFT is a high efficiency, low thrust system that ionizes an inert gas (Xenon in this case), and propels the ionized gas using a set of permanent magnets. Though similar to a Hall Effect thruster, the DCFT uses permanent magnets rather than electromagnets to create the magnetic field required to accelerate the ionized gas. Hall Effect thrusters experience degradation due to the extremely hot ionized gas passing through the engine nozzle. Due to differences in its structural design, the DCFT experiences significantly less degradation, which will allow the DCFT to operate at high efficiency for longer than a standard Hall Effect thruster [8].

The CASTOR satellite will provide the space support system necessary to operate the DCFT on orbit and increase the thruster's technology readiness level (TRL). A SolidWorks

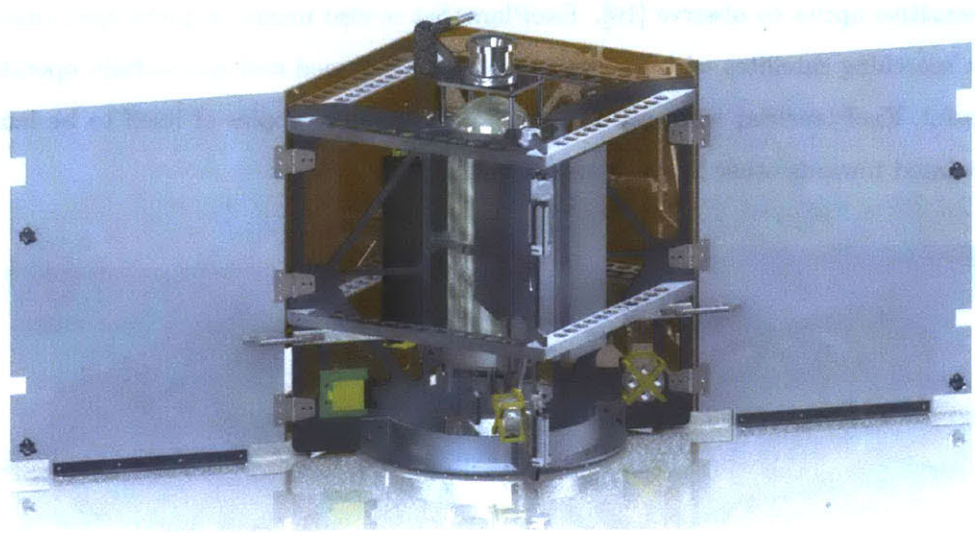
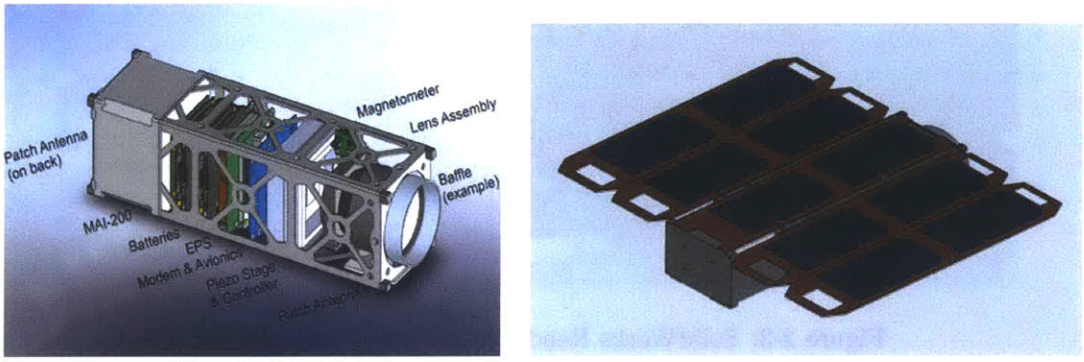


Figure 2-3: SolidWorks Rendering of CASTOR Satellite

model of the CASTOR satellite is shown in Figure 2-3. In order to provide the power necessary to operate the DCFT as well as point the engine in the desired thrust direction, CASTOR requires three axis stability. For attitude estimation, CASTOR uses four sun sensors, two three axis magnetometers supported by a GPS unit, and a three axis inertial measurement unit [8]. For attitude control, CASTOR uses three orthogonal reaction wheels, which provide full controllability but no redundancy in case of wheel failure. The reaction wheels are desaturated using three orthogonal torque coils [8]. An Extended Kalman Filter is used to provide CASTOR's attitude estimate using sensor inputs, and a linear quadratic regulator is used to provide attitude control commands to the reaction wheels. This ADCS system will allow CASTOR to point its solar panels towards the sun for battery charging, and point the engine in the desired thrust direction once per orbit as required by CASTOR's CONOPs [8].

ExoPlanetSat is a three unit cubesat being designed and built by the SSL in cooperation with MIT's Department of Earth, Atmospheric, and Planetary Sciences (EAPS). Figures 2-4(a) and 2-4(b) show ExoPlanetSat's internal component configuration and solar panel configuration respectively. The mission of ExoPlanetSat is to "be capable of detecting a transiting Earth-sized planet in the habitable zone of the brightest sun-like stars" [10]. Though there are large scale satellites on orbit whose mission is to detect exoplanets, ExoPlanetSat will be capable of looking at stars that are too bright for large satellites with

very sensitive optics to observe [10]. ExoPlanetSat is also meant to prove that small exoplanet searching satellites with low budgets can be designed and successfully operated. If successful, ExoPlanetSat will pave the way for many more copies of itself to be launched and pointed towards other bright stars in Earth's sky.



(a) ExoPlanetSat Component Configuration

(b) ExoPlanetSat Solar Panel Configuration

Figure 2-4: SolidWorks Rendering of ExoPlanetSat

ExoPlanetSat will detect exoplanets by staring at a single bright star for a long period of time using the camera assembly shown in Figure 2-4(a). If an exoplanet orbiting the target star passes between ExoPlanetSat and the target star, ExoPlanetSat's optical system will be able to detect the decrease in light intensity from the target star [10]. In order for this type of exoplanet detection to work, the satellite's ADCS system must be capable of keeping the target star in almost exactly the same location with respect to ExoPlanetSat's optical system. In order to achieve such precision three axis attitude control, ExoPlanetSat uses two attitude control loops; a course and fine loop. The first attitude control loop, the course loop, estimates and controls the entire satellite's attitude to within sixty arc-seconds [10]. ExoPlanetSat will use a three axis magnetometer and a three axis gyroscope for course loop attitude estimation. For course loop attitude control, ExoPlanetSat will use three orthogonal reaction wheels and three orthogonal torque coils [10]. The second attitude control loop, the fine loop, estimates and controls the attitude of the satellite's optical sensor in the two axes perpendicular to the vector pointing towards the target star [10]. The optical system estimates its attitude in the plane perpendicular to the target star vector using the pattern of stars surrounding the target star. This portion of the optical system works just like a star tracker. For fine loop attitude control, the optical system is mounted

on a piezoelectric stage that can move in both axes perpendicular to the target star vector. The fine attitude control loop can estimate and control the attitude of the optical system to within one arc-second, which is the mission requirement for stabilizing the optical system with respect to the target star [10]. If successful, ExoPlanetSat will demonstrate the most accurate three axis stabilized attitude control system ever attempted on a cubesat.

Tethered Environmental Reconditioning Satellite (TERSat) is a second ESPA class satellite being developed within the SSL. TERSat is being developed as part of the seventh iteration of the UNP small satellite competition. TERSat's mission is to remove high energy protons from the inner Van Allen Radiation Belt [41]. Energetic protons in the lower Van Allen Belt occur naturally but can also be caused by nuclear detonations in the upper atmosphere. High energy protons damage satellites by causing single event upsets and transistor charging. Single event upsets occur when protons impact a satellite's memory storage devices and cause a bit to flip from a one to a zero or visa versa. A bit flip can cause an entire section of code to malfunction, which can ultimately lead to the failure of a satellite's avionics system. Transistor charging occurs when protons impact the silicon surrounding a transistor. Over time, the charge builds up to a point where the transistor can no longer flip from low to high rendering the transistor useless [25].

Energetic protons travel back and fourth along the Earth's magnetic field lines, bouncing off the Earth's atmosphere at the points where the magnetic field lines intersect with the Earth, once in the Northern and once in the Southern Hemisphere. TERSat will attempt to remove these energetic protons by emitting Electromagnetic Ion Cyclotron (EMIC) waves, which will coax the energetic protons to pass into Earth's atmosphere at one of the two magnetic field line intersection points, rather than bouncing off the atmosphere and traveling back into space [41]. In order to emit EMIC waves, TERSat will need a four kilometer long antenna. To create this antenna, TERSat will release two tethers on opposite sides of the satellite. Each tether will extend out two kilometers creating the required antenna. Though the exact ADCS design has not been chosen, TERSat's primary structure will require three axis stabilization in order to align the tethers in the correct orientation for ejection. TERSat will likely use sun sensors in combination with a three axis magnetometer and three axis inertial measurement unit for primary body attitude estimation and three orthogonal reaction wheels and magnetic torque devices for attitude control [41].

Once extended several hundred meters from the primary structure, each tether will continue to deploy using gravity gradient force. However, during the initial phase of tether deployment, the gravity gradient force is not sufficient to deploy the tethers. Therefore, a short term ADCS system must be integrated into modules at the end of each tether. The sensor suite in each module must be able to determine the module's attitude and position with respect to the primary structure. The module will likely use cold gas thrusters to correctly position itself with respect to the primary structure, and deploy itself until the gravity gradient force is sufficient to continue tether deployment. TERSat will be the first ESPA class satellite to attempt tether deployment on the kilometer scale. If successful, TERSat will not only help remove satellite debilitating high energy protons from the Van Allen Belts, but it will also create a foundation for future complex ADCS systems in small satellites [41].

2.3 Satellite ADCS Failures

Satellite ADCS systems are often complex combinations of hardware components and software algorithms, and their flawless operation is a requirement for mission success. As can be expected with such complex systems, there are many examples of satellite ADCS systems failing either partially or completely once on orbit. Below are several examples of on-orbit ADCS failures for satellites of all sizes. They are listed in chronological order.

The first American satellite, known as Explorer I, was designed and built by the Jet Propulsion Laboratory (JPL) in Pasadena, CA and launched in January, 1958 [30]. Explorer I carried an instrument designed by James van Allen to measure the radiation environment in space [30]. Explorer I and James van Allen are famous for discovering the high energy particle radiation belts now known as the Van Allen Belts [30]. Though a successful mission overall, Explorer I suffered from a debilitating attitude control failure. The satellite was permanently fixed to the rocket's fourth stage, which was spun during launch to provide stability and even out any thrust imbalances in the engine [30]. The satellite also had four whip antennas mounted symmetrically about the spinning axis [30]. Once in orbit, the uncontrolled spinning satellite (and fourth stage) began to experience periodic communication loss [30]. This was eventually determined to be caused by the satellite's spin axis moving from the long, minimum inertia axis to the transverse, maximum inertia

axis [30]. Spinning satellites tend towards their minimum energy states, which is rotation about the axis of maximum inertia [30]. Spinning about the minimum inertia axis is an unstable equilibrium, and any source of energy loss will allow the satellite to change its rotation axis. Flexing in Explorer I's four whip antennas provided the energy loss necessary to transfer from the unstable minimum inertia axis to the stable maximum inertia axis, which led to periods of loss in communication due to incorrectly oriented antennas [30].

In November, 1986 The Polar Beacon Experiment and Auroral Research (Polar BEAR) spacecraft built by the Naval Research Laboratory was launched to measure auroral and ionospheric effects on radio frequency wave propagation [29]. The satellite was gravity gradient stabilized using a boom and end mass in combination with a pitch axis flywheel [29]. Due to solar heating of the boom, the satellite began to sway back and forth away from the gravity gradient orientation. Eventually, the satellite flipped over and stabilized in the inverted orientation [29]. The satellite operations team was eventually able to re-invert the satellite by allowing the momentum wheel to slowly spin down and then quickly spin up, producing enough angular rate to overcome the gravity gradient torque [29].

The Magellan spacecraft was deployed by the Space Shuttle in May, 1990 [30]. Magellan's mission was to travel to and map the surface of Venus. After achieving orbit around Venus and beginning the process of mapping Venus' surface using its synthetic-aperture radar, Magellan began to experience spontaneous increases in attitude error and angular rate about the spacecraft's body fixed x axis [30]. Engineers determined the problem to be in the solar panel pointing control loop [30]. When the solar panels were near their commanded angular orientation, the step size of the solar panel actuator was greater than the angular error between the estimated and commanded solar panel orientation [30]. This caused the solar panels to flip back and forth on either side of their commanded orientation [30]. The solar panel jitter excited a seven Hertz oscillation mode in the panels that led to the x axis attitude error [30]. Engineers were able to stabilize Magellan by adjusting the solar panel control algorithm [30].

The TOPEX-Poseidon satellite was designed in cooperation between the United States and France and launched in August, 1992 [30]. TOPEX-Poseidon's mission was to measure the topography of Earth's oceans in order to determine current flows and investigate the El Nino weather phenomenon [30]. TOPEX-Poseidon used two star trackers with thermoelectric coolers in combination with several other sensors for attitude control [30]. In order

to prevent damage to the star trackers, mechanical shutters would close if bright objects like the sun or moon moved to within twenty degrees of the star tracker's field of view [30]. After four months of operation, a single event upset caused the shutter control algorithm in one of the star trackers to always assume the star tracker was pointed towards a bright object and the shutter was closed [30]. Engineers reconfigured the control algorithm to use the remaining star tracker, which was successful until April, 1998 [30]. After six years of operation, degradation in the optical system caused an increase in background current. If the shutter were to close due to a bright object, it would continue to 'see' a bright object regardless of the shutter position, and it would not reopen [30]. Though engineers attempted to predict when the sun or moon would pass too close to the star tracker's field of view and maneuver the satellite to avoid such encounters, an unexpected reflection of the sun off of the star tracker's baffle caused the shutter to permanently close [30]. Engineers were able to save the satellite by rebooting, which cleared the single event upset in the first star tracker [30]. Since it had only been used briefly, the recovered star tracker operated like new [30].

The Total Ozone Mapping Spectrometer satellite was developed by NASA's Goddard Space Flight Center as part of its Earth Probe series, hence the name TOMS-EP [30]. TOMS-EP was launched in July, 1996 on-board a Pegasus rocket [30]. The satellite was three axis stabilized with sun sensors, a magnetometer, and two Earth horizon sensors for attitude estimation, and thrusters, magnetorquers, and reaction wheels for attitude control [30]. Soon after launch, two of the sun sensors began providing erroneous measurements and were found to be cross-wired from installation [30]. The problem was solved by switching the two sensor outputs in the software algorithm [30]. After errors during magnetorquer operations arose, the polarities of the magnetorquers were found to be inverted [30]. This problem was also mitigated by altering the software control algorithm [30].

The Lewis spacecraft was developed by NASA and launched from Vandenberg Air Force Base on 23 August, 1997 [30]. The Lewis spacecraft was designed to make high-resolution multi-spectral observations of the Earth from a 600 kilometer altitude [30]. After working out several avionics issues in the first few days on orbit, Lewis was placed in a zero rotation sun pointing safe mode [30]. Once in sun pointing mode, the operations crew retired for the evening. Lewis used two rate gyroscopes to measure angular rotation about the axes perpendicular to the sun pointing axis and thrusters to control its attitude. While maintaining its sun pointing orientation, a thruster imbalance induced a spin about Lewis' sun pointing

axis, which was unobserved by the gyroscopes in the other two axes [30]. Furthermore, Lewis' solar panel face was the satellite's intermediate inertial axis, which may not be a problem for a satellite with little to no angular rate. However, the spin induced by the thruster imbalance was unstable about the intermediate axis [30]. By the time the satellite operators returned, Lewis had precessed to a spin about its major axis of inertia, which caused the solar panels to be edge on to the sun [30]. The thrusters had depleted their fuel while trying to recover the satellite and the batteries were nearly dead and not receiving a charge [30]. Satellite operators permanently lost contact with Lewis on 26 August, 1997 just three days after launch [29].

The Tomographic Experiment using Radiative Recombinative Ionospheric Extreme ultraviolet and Radio Sources (TERRIERS) microsatellite was designed and built by Boston University and launched in May, 1999 by a Pegasus launch vehicle [30]. After being placed in a sun synchronous 550 kilometer orbit, the satellite could not face its solar panels towards the sun, drained its batteries, and shut down [30]. Engineers determined the problem to be a sign flip which inverted the polarity of one of the magnetorquers used for attitude control, similar to the problem faced by NASA's TOMS-EP satellite [30]. Using a ground engineering model, the operations team determined that the satellite could successfully reboot if the solar panel were to face the sun for a sufficient amount of time, and a software patch could be uploaded to correct the sign error [30]. After months of contact attempts, the operations team was never able to communicate with the TERRIERS satellite [30].

The Far Ultraviolet Spectroscopic Explorer (FUSE) spacecraft was designed and built by NASA and launched in June, 1999 [29]. FUSE was used to make observations of distant stars, galaxies, and other deep space objects [29]. The FUSE satellite used magnetic torque coils and reaction wheels for attitude control [29]. In 2001, two of the four reaction wheels failed, but engineers were able to reprogram the satellite's control algorithm to maintain attitude control using the two remaining reaction wheels and the magnetic torque coils [29]. After one of the six rate gyroscopes failed in 2001 and the other five showing degradation, engineers reprogrammed the satellite's attitude estimation algorithm to operate without gyroscopes completely [29]. Though several ADCS components were lost on the FUSE spacecraft, redundancy and clever engineering were able to extend the satellite's mission life.

NASA's Imager for Magnetopause to Aurora Global Exploration (IMAGE) spacecraft was launched from Vandenberg Air Force Base in March, 2000 [29]. The satellite was designed to be spin stabilized using a single magnetic torque rod [29]. IMAGE also used a passive ring nutation damper intended to remove any nutation due to external disturbances on the spinning satellite [29]. However, after the satellite's initial nutation did not dampen out as expected, engineers determined that the satellite's low spin rate of 0.5 RPM was not high enough to overcome the surface tension of the liquid mercury in the nutation damper, rendering the device useless at the commanded spin rate [29]. Satellite operators were able to stabilize the satellite by uploading an open loop nutation damping control algorithm using the magnetic torque rod [29].

The Thermosphere Ionosphere and Mesosphere Energetics and Dynamics (TIMED) satellite was built by NASA and launched from Vandenberg Air Force Base in December, 2001 by a Delta II rocket [30]. The TIMED satellite used magnetometers, star trackers, sun sensors and an inertial measurement unit for attitude estimation and reaction wheels and magnetorquers for attitude control [30]. In order to remove excess angular momentum in the satellite after being released by the launch vehicle, engineers decided to do a rate damping maneuver using the magnetorquers. However, measurements from the inertial measurement unit were signaling an increase in angular momentum [30]. Engineers determined that this was due to a sign error in the magnetorquers, again similar to the TOMS-EP and TERRIERS satellites [30]. After fixing the sign error and stabilizing the satellite, engineers noticed that TIMED was trying to point the wrong axis towards the sun [30]. After examining several photos of the satellite before launch, they determined that two of the sun sensors were mounted ninety degrees from their designed locations [30]. By reprogramming the attitude estimation algorithm to account for the incorrectly located sun sensors, engineers were able to stabilize the satellite and perform the mission [30].

FalconSat-3 was designed and built by the Astronautical Engineering Department at the United States Air Force Academy. FalconSat-3 was launched in March, 2007 as a secondary payload on-board an Atlas V from Cape Canaveral Air Force Station in Florida. FalconSat-3 was designed to be a three axis stabilized satellite using a gravity gradient boom to maintain Nadir pointing. For active attitude estimation, FalconSat-3 uses sun sensors and magnetometers. For active attitude control, the satellite has three orthogonal magnetic torque rods. After separation from the launch vehicle, FalconSat-3 was unable to

reduce stored angular momentum using the magnetic torque rods. This was believed to be due to sign errors, which caused the polarity of the torque rods to be reversed. Though the system's angular momentum had not been removed, the gravity gradient boom was deployed, which reduced the satellite's angular rates, but did not change the system's stored angular momentum. The problem was exacerbated by the sun sensors failing to operate due to software interface errors leaving the magnetometers as the only attitude sensor. Though not inverted, FalconSat-3 has yet to successfully remove excess angular momentum in the system, and has large angle oscillations about the Nadir pointing attitude orientation.

As shown by the above examples, ADCS failures can occur at any step of the satellite design process. Failures are due to insufficient modeling, incorrect construction, software errors, component malfunctions and more. Though not all, at least some of the above ADCS failure cases could have been identified and mitigated using devices like an ADCS testbed during the satellite's testing phase. Though time and cost budgets may be limited, increased testing of the ADCS system using devices like an air bearing could ultimately save the mission.

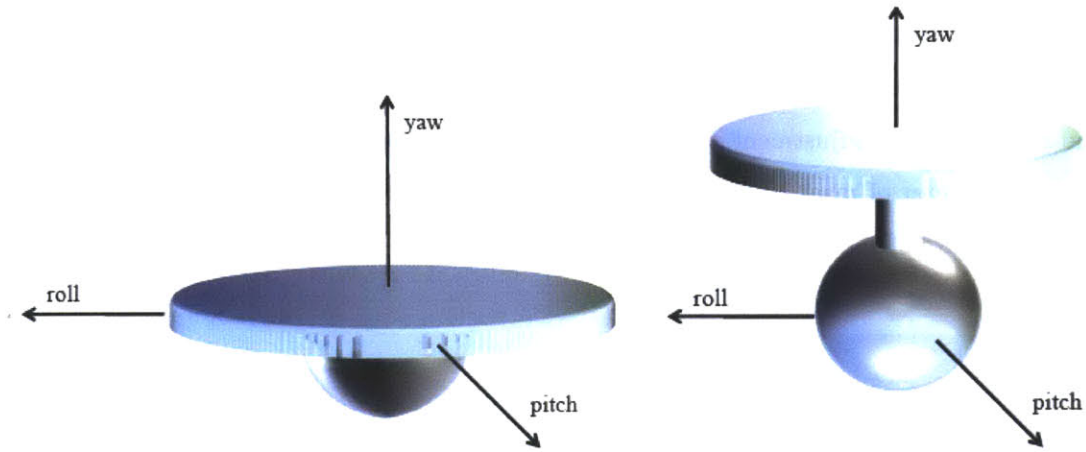
2.4 State of ADCS Air Bearing Testbed Technology

There are two main categories of air bearings used to develop ADCS testbeds. These categories include planar air bearings and rotational air bearings. Planar air bearings operate using a flat, planar surface upon which objects float using compressed gas. Planar air bearings allow for two translational degrees of freedom along the surface of the plane as well as one rotational degree of freedom about the axis perpendicular to the plane. The translational degrees of freedom are constrained by the dimensions of the plane, but the rotational degree of freedom is unconstrained. Planar air bearings can also support multiple devices simultaneously on the plane if space is available. Rotational air bearings operate by floating a spherical object above a concave structure that matches the the geometry of the sphere. Rotational air bearings also use compressed gas to float the spherical object just above the surface of the concave support structure. Though constrained in all three translational degrees of freedom, rotational air bearings provide three rotational degrees of freedom. Angular constraints are a function of the specific geometry of the rotational air

bearing. Though complex, planar and rotational air bearings can be combined to provide additional degrees of freedom for ADCS testing [49].

MIT's Space Systems Laboratory uses a planar air bearing for operating the SPHERES ADCS testbed. The operating plane is approximately 1.5 meters by two meters and can accommodate up to three SPHERES at once, which allows for formation flight testing with two constrained translational degrees of freedom and one unconstrained rotational degree of freedom for each SPHERE. The SPHERES satellites operate on the air bearing by interfacing with an air carriage. Each air carriage can accommodate one SPHERE and two compressed CO_2 tanks, which provide the gas necessary to float the flat bottom of the air carriage above the glass surface of the flat plane. Using the planar air bearing, engineers can develop ADCS algorithms that will eventually be tested using the three SPHERES located on the ISS. The ISS provides the ultimate ADCS test platform since it allows for all six degrees of freedom; three constrained translational degrees and three unconstrained rotational degrees of freedom.

Though planar air bearings provide a useful test platform for multiple objects, rotational air bearings often provide a better attitude determination and control test platform for a free-flying satellite because they allow for motion in all three rotational degrees of freedom. Two similar types of rotational air bearings are the tabletop and umbrella style air bearings, shown in Figures 2-5(a) and 2-5(b) respectively. Both of these types of air bearings allow for unconstrained rotation about the yaw axis and constrained rotation about the remaining two axes. The difference between the two is that the umbrella style air bearing allows for increased freedom of motion in the pitch and roll axes by mounting the component plate above and out of the way of the rotating portion.



(a) Tabletop Style Air Bearing

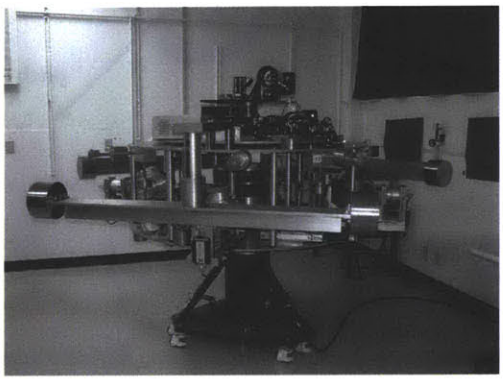
(b) Umbrella Style Air Bearing

Figure 2-5: Tabletop and Umbrella Style Rotational Air Bearings [50]

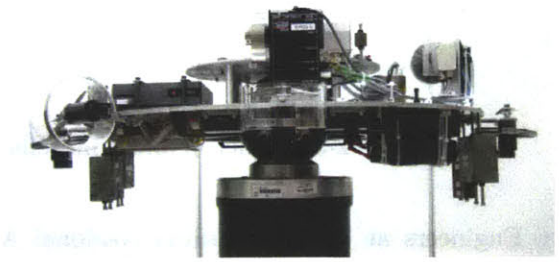
Engineers at the Universidad Nacional Autonoma de Mexico (National Autonomous University of Mexico) in Mexico City developed and operate a tabletop style rotational air bearing for ADCS testing. The rotational air bearing can rotate up to fifty degrees off nominal in both the pitch and roll axes and support up to eighty kilograms [45]. The air bearing has three orthogonal reaction wheels for attitude control and three orthogonal torque coils for reaction wheel desaturation or attitude control [45]. The testbed uses inertial measurement units and built in-house sun and Earth sensors for attitude estimation. Finally, the testbed communicates wirelessly with a nearby ground station computer for reprogramming and data storage [45]. The air bearing has manual as well as automatic center of mass adjustment devices used to reduce torque due to gravity on the system [45].

The Naval Postgraduate School in Monterey, California developed and operates a tabletop style rotational air bearing to test the Bifocal Relay Mirror Spacecraft (BRMS), which was designed to redirect laser light from a given source to distant targets either on the Earth or in space [35]. The BRMS rotational air bearing testbed is capable of floating an 800 kilogram mass with seventy pounds per square inch of air pressure and can rotate up to twenty degrees off nominal in the pitch and roll axes [35]. For attitude estimation, the testbed uses a three axis magnetometer, two inclinometers, a two axis sun sensor, and an inertial measurement unit [35]. The testbed is controlled using three control moment gyroscopes (CMGs) which operate by rotating a constant velocity wheel about an axis perpendicular

to its axis of rotation [35]. Rotating the wheel creates a gyroscopic torque that can be used to control the testbed's attitude. The air bearing testbed uses an automatic center of mass adjustment system made up of three masses attached to orthogonal linear actuators [35]. The automatic CM adjustment system operates in real time during ADCS tests in order to ensure the center of mass (CM) remains close to the center of rotation [35]. Figure 2-6(a) shows a picture of the BRMS air bearing testbed.



(a) BRMS Testbed [35]



(b) SimSat II Testbed [50]

Figure 2-6: Tabletop Style Air Bearing Examples

The Air Force Institute of Technology (AFIT) at Wright-Patterson Air Force Base, Ohio developed and currently operates a tabletop style rotational air bearing satellite simulator known as SimSat II [50]. The air bearing can rotate up to twenty degrees off nominal in the pitch and roll axes and support up to 136 kilograms [50]. SimSat II uses a Northrop Grumman LN-200 Fiber Optic Gyroscope Inertial Measurement Unit (IMU) as its only attitude estimation device, and three orthogonally mounted reaction wheels for attitude control [50]. An on-board Mini-Box PC serves as the air bearing's avionics processor, which wirelessly communicates with a ground station desktop computer for programming and data storage [50]. SimSat II uses manually placed counter-masses to move the system's center of mass to the center of rotation [50]. Figure 2-6(b) shows a picture of the SimSat II ADCS testbed.

A third type of rotational air bearing is the dumbbell style air bearing. The dumbbell style air bearing provides unconstrained rotational motion about the vehicle's yaw axis just like the tabletop and umbrella style air bearings. However, the dumbbell style air bearing

also provides unconstrained rotational motion about the vehicle's roll axis. Only the pitch axis is constrained. Drawbacks to this style are that the air bearing requires two relatively independent sections on either side of the rotating sphere that must be perfectly balanced with each other and can only be connected via wires passing through the center of the rotating sphere. Figure 2-7 shows the general design of a dumbbell style rotational air bearing.

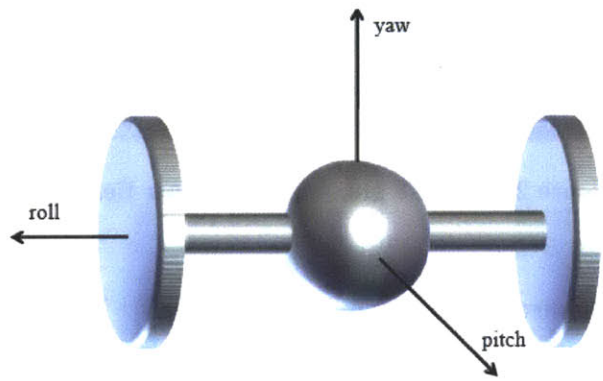


Figure 2-7: Dumbbell Style Rotational Air Bearing [50]

Engineers at the University of Michigan developed and operate a dumbbell style rotational air bearing testbed known as the Triaxial Attitude Control Testbed (TACT) [19]. Operating at seventy pounds per square inch of air pressure, the air bearing can support up to 360 pounds and rotate up to forty-five degrees off nominal in the pitch axis, which is the only constrained axis on the dumbbell style air bearing [19]. For attitude control, the TACT air bearing is designed to use several types of actuators. Developers designed the air bearing to accommodate reaction wheels, fans, or mass actuators [19]. Though fans obviously cannot be used in a space environment with no atmosphere, they can simulate torque similar to that of thrusters acting on an actual satellite. The mass actuators would not work in the space environment either, but on the air bearing, they can be manipulated to move the center of gravity with respect to the center of rotation and produce gravitational torques on the system that can be used for attitude control. Figure 2-8 shows a picture of the TACT dumbbell style air bearing configured to use reaction wheels for attitude control.

Engineers at the Virginia Polytechnic Institute & State University have developed a satellite ADCS test platform using two rotational air bearings; one tabletop style air bearing and one dumbbell style air bearing [48]. The set of air bearings are known as the Distributed

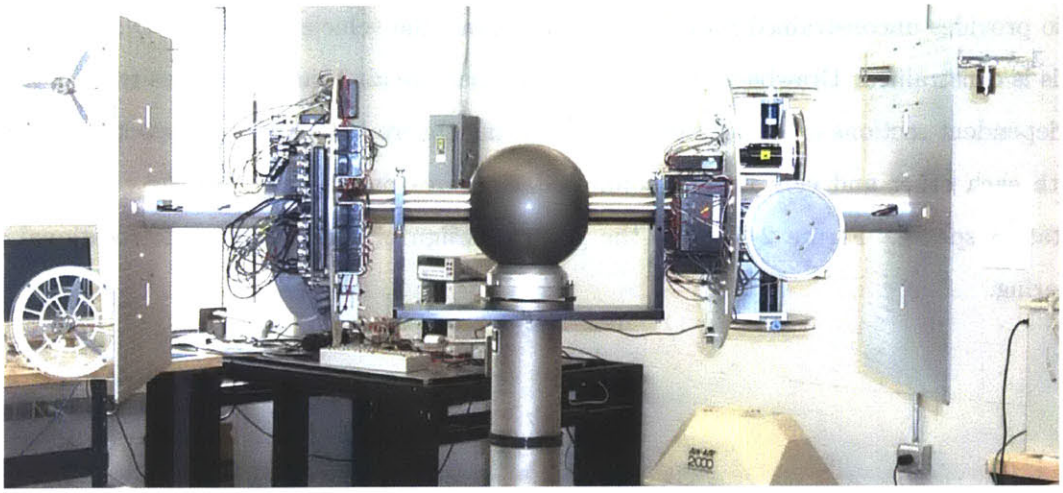


Figure 2-8: TACT Dumbbell Style Air Bearing [19]

Spacecraft Attitude Control System Simulator (DSACSS). The set of air bearings allow test developers to test single satellite attitude control systems as well as multiple satellite formation ADCS systems. Each air bearing is capable of supporting up to 300 pounds [48]. The tabletop air bearing can rotated up to five degrees off nominal in the constrained pitch and roll axes, and the dumbbell air bearing can rotate up to thirty degrees off nominal in the constrained pitch axis [48]. Both air bearings use three axis rate gyroscopes and three axis linear accelerometers for attitude estimation [48]. For attitude control, each air bearing is designed to use three orthogonal reaction wheels or a set of cold gas thrusters [48]. The design team has also developed a single CMG that can be attached as a control device [48]. The center of mass adjustment system is made up of three masses mounted upon orthogonal linear actuators [48]. Presently, the tabletop style air bearing is nearly complete, and the dumbbell style air bearing is in the design phase. Figure 2-9 shows the completed tabletop air bearing with the dumbbell air bearing base in the background.

Planar and rotational air bearings both provide useful testbeds depending on the requirements of the test scenario, though both have limitations due to constraints on certain degrees of freedom. Planar air bearings only provide one degree of rotational freedom, and rotational air bearings provide no translational degrees of freedom. However, combining the two air bearing styles can provide additional degrees of freedom that could not be achieved using either air bearing style individually. NASA's Marshall Space Flight Center developed a combined air bearing system using a rotational air bearing mounted upon a cylindrical

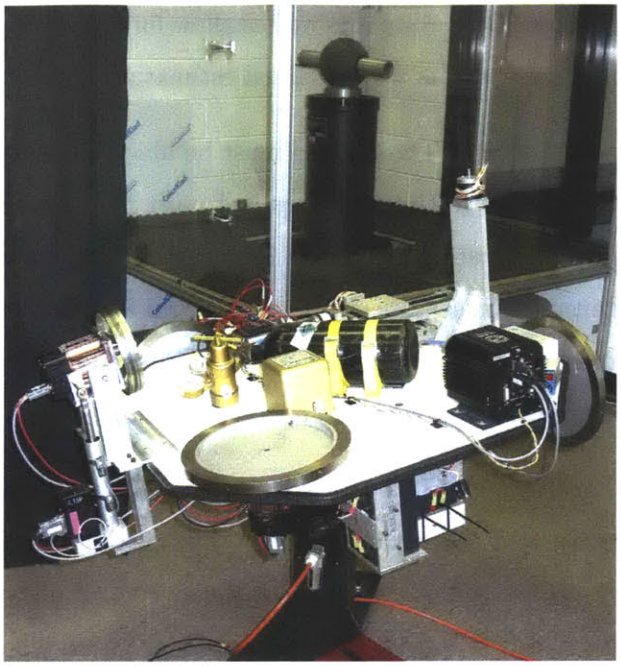


Figure 2-9: DSACSS Air Bearing Set [48]

air cushioned lift. The lift is mounted on a planar air bearing system floating above a flat floor [49]. The combined set of air bearings provide a 400 pound payload all six degrees of translational and rotational freedom [49].

2.5 Testbed Design Requirements

The ADCS testbed developed as described by this thesis will be a rotational tabletop style air bearing that will provide three rotational degrees of freedom with constraints in the pitch and roll axes. The ADCS testbed based on the tabletop style air bearing must provide a platform for testing small satellite attitude determination and control systems in a university environment. Because the testbed will operate within MIT's Space Systems Laboratory, the testbed must meet the needs of student groups studying and applying control theory in addition to specific ADCS tests required by satellite design teams. The following bullet list outlines the four key air bearing design requirements.

- Class Project Support - MIT's Aeronautical and Astronautical engineering department uses hands on tools to accompany estimation and control theory taught in the classroom. The testbed must have the required software and hardware to produce a

functional closed loop ADCS system as well as allow for high level operation to support projects developed by students with minimal estimation and control background.

- ADCS Component Testing - The testbed must be able to integrate individual ADCS hardware components for testing. The testbed must have hardware mounting capability and software integration capability for a variety of components, whether they are attitude determination sensors or attitude control actuators. The testbed must be able to power these components, communicate with them, and provide a closed loop ADCS system for these components to operate within.

- ADCS Algorithm Testing - The testbed must be able to incorporate a variety of software estimation and control algorithms. The testbed avionics system must be versatile enough to accommodate estimation and control algorithms developed for specific satellite projects in the lab or for classroom projects. The hardware sensors on the testbed must be able to provide measurements for use by the estimation algorithm, and the actuators must be able to provide control torques in response to control algorithm commands.

- Integrated ADCS Testing - The testbed must be able to perform integrated ADCS testing for specific satellite projects within the SSL. The testbed must be able to incorporate hardware components simultaneously with software algorithms for integrated ADCS testing. The testbed must provide wireless communication, electrical power, processing capability, and any additional support required to allow for closed loop ADCS testing.

Chapter 3

ADCS Testbed Model and Simulation

A dynamic model and simulation are created as part of the three degree of freedom rotational air bearing testbed. The simulation and integrated model allow the user to develop test scenarios for the air bearing testbed and simulate hardware or software adaptations to the air bearing. The simulation provides the user with a predicted response that allows the user to determine if the desired test scenario is within the air bearing's capabilities and if so, the simulated response can be used to validate the actual response of the air bearing during testing as well as provide key details for results analysis after a test is carried out on the air bearing.

This chapter focuses on the development of the air bearing model and simulation. First, the primary coordinate systems used to define the motion of the air bearing are developed. Using these coordinate systems, the equations of motion that define the dynamics of the air bearing are derived. These equations include the dynamic effects introduced by the integrated reaction wheels. From these equations, a simulation is created to model the dynamics of the air bearing. The simulation can be modified by the user to match changes in the physical air bearing. Modifications can be made to any module of the simulation depending on the specific changes required. The simulation is developed in the MATLAB Simulink environment. Each major module of the simulation is detailed to ensure a clear understanding of the simulation as well as aid in modification.

3.1 Dynamic Air Bearing Model

The air bearing allows for three rotational degrees of freedom and constrains all three translational degrees of freedom. Therefore, three equations of motion will be derived; one for each rotational degree of freedom. The first step in deriving these equations is to define the required three axis coordinate systems necessary to describe the system.

3.1.1 Coordinate Systems

The air bearing uses three primary coordinate systems. These include the fixed inertial reference (FIR) coordinate system, the air bearing body fixed (ABBF) coordinate system, and the reaction wheel coordinate system (RWCS). The first to be defined is the fixed inertial reference coordinate system that is fixed with respect to the laboratory at all times. Though the FIR coordinate system is assumed to be fixed in inertial space, the coordinate system rotates once per day with respect to the Earth's fixed inertial frame due to the laboratory's position on the surface of the Earth. The fixed inertial assumption introduces error into the air bearing system that will be discussed in more detail in Section 5.1.1.

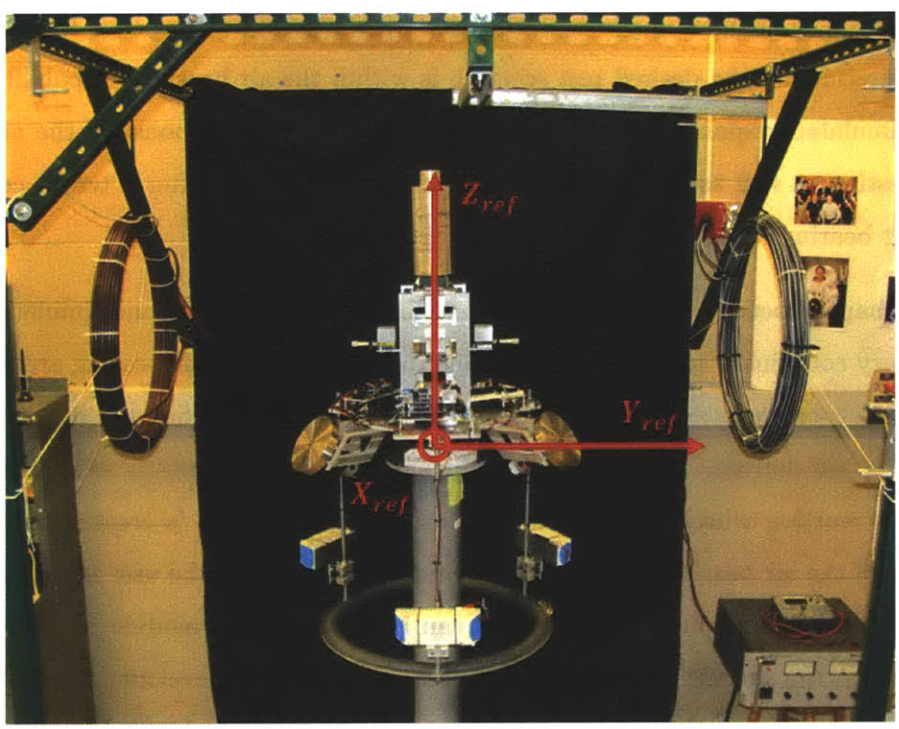


Figure 3-1: Air Bearing with Fixed Inertial Reference Coordinate System

Figure 3-1 depicts the fixed inertial reference coordinate system with respect to the green external equipment frame. The following are the defining characteristics of the fixed inertial reference coordinate system.

- **Fixed Inertial Reference Point of Origin** - The fixed inertial reference point of origin is located at the center of rotation of the air bearing.
- $\mathbf{X_{ref}}$ **Axis** - The X_{ref} axis is perpendicular the gravity vector and points away from the wall and toward the viewer as seen in Figure 3-1.
- $\mathbf{Y_{ref}}$ **Axis** - The Y_{ref} axis is perpendicular to the gravity vector and points to the right from the perspective of the viewer in Figure 3-1.
- $\mathbf{Z_{ref}}$ **Axis** - The Z_{ref} axis is the cross product of the X_{ref} axis with the Y_{ref} axis and completes the three axis right handed coordinate system.

The second coordinate system to be defined is the air bearing body fixed coordinate system. This coordinate system remains fixed with respect to the air bearing as it rotates through the FIR frame. The mass properties of the air bearing remain constant in a body fixed coordinate system and the principle axes of the air bearing are aligned with this coordinate system. Figure 3-2 depicts the ABBF coordinate system with respect to the air bearing. The ABBF coordinate system is represented in black. The following are the defining characteristics of the ABBF coordinate system.

- **Air Bearing Body Fixed Point of Origin** - The air bearing body fixed point of origin is located at the center of rotation of the air bearing.
- $\mathbf{x_{body}}$ **Axis** - The x_{body} axis is in the plane of the base plate of the air bearing and extends out towards the component mounting plate holding the avionics processor.
- $\mathbf{y_{body}}$ **Axis** - The y_{body} axis is also in the plane of the base plate of the air bearing and is perpendicular to the x_{body} axis.
- $\mathbf{z_{body}}$ **Axis** - The z_{body} axis is the cross product of the x_{body} axis with the y_{body} axis and completes the three axis right handed coordinate system. The z_{body} axis is perpendicular to the base plate of the air bearing and extends upward opposite of the hemisphere mounted beneath the base plate.

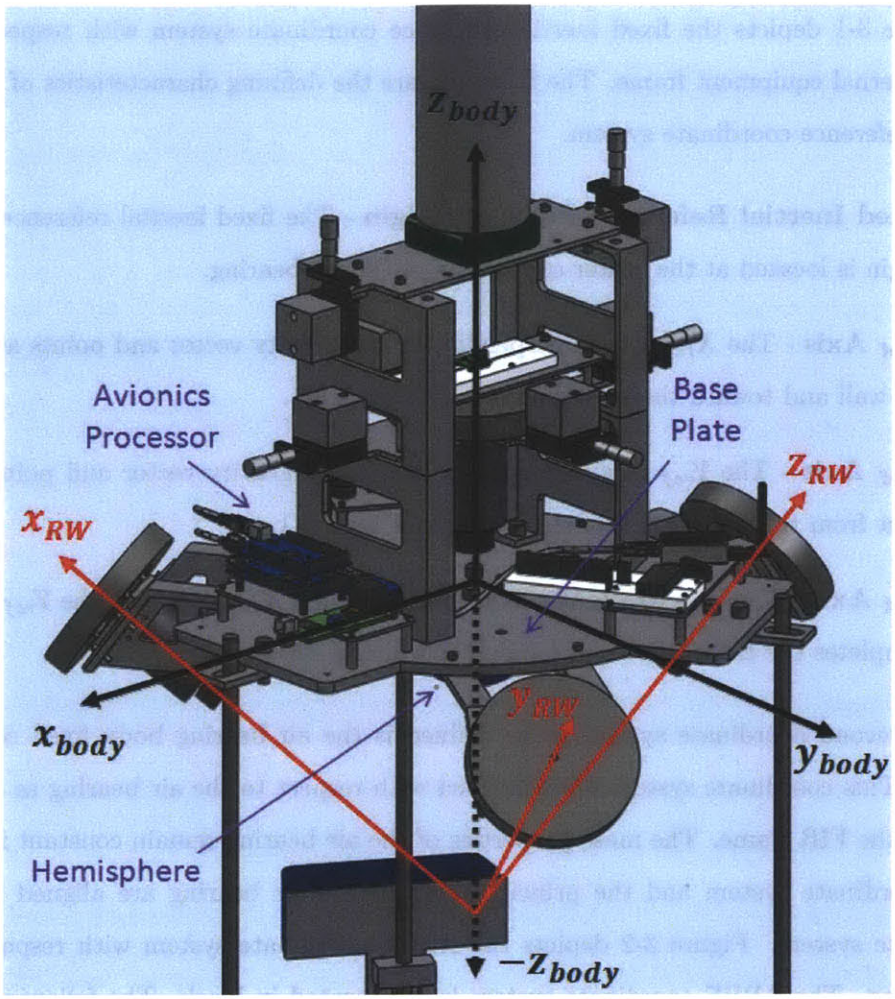


Figure 3-2: Air Bearing with Air Bearing Body Fixed and Reaction Wheel Coordinate Systems

The third coordinate system to be defined is the reaction wheel coordinate system. The three orthogonal reaction wheels are mounted symmetrically about the z_{body} axis in order to maintain air bearing symmetry while providing the ability to test the common three reaction wheel attitude control system used in many satellites as discussed in Section 2.2. The reaction wheel coordinate system is fixed to the air bearing body similarly to the ABBF coordinate system; however, the reaction wheel coordinate system is aligned with the rotation axes of the three orthogonal reaction wheels. The reaction wheel coordinate system is shown in Figure 3-2, and is represented in red. The following are the defining characteristics of the reaction wheel coordinate system.

- **Reaction Wheel Point of Origin** - The reaction wheel point of origin is located on the negative z_{body} axis below the air bearing's point of rotation. The origin is placed at the intersection of the three vectors that are coincident with each reaction wheel's rotation axis. Reference Figure 3-2 for clarity.

- **$\mathbf{x_{RW}}$ Axis** - The x_{RW} axis is coincident with reaction wheel one's rotation axis.

- **$\mathbf{y_{RW}}$ Axis** - The y_{RW} axis is coincident with reaction wheel two's rotation axis.

- **$\mathbf{z_{RW}}$ Axis** - The z_{RW} axis is coincident with reaction wheel three's rotation axis. The z_{RW} axis is also the cross product of the x_{RW} axis with the y_{RW} axis, which completes the three axis right handed coordinate system.

3.1.2 Equations of Motion

The air bearing can move in all three rotational degrees of freedom. Therefore, three rotational equations of motion must be derived to completely define the motion of the air bearing. The three equations of motion will be derived in the ABBF coordinate system described in Section 3.1.1 because the mass properties of the air bearing are constant in this frame. For simplicity, the equations will be derived in vector form. Equation 3.1 is the angular representation of Newton's second law of motion, which states that a mass subject to a force undergoes an acceleration. In this case, a mass, represented by its second moment of inertia I, subject to a torque undergoes an angular acceleration, $\dot{\bar{\omega}}$. The inertia times the angular acceleration is also equal to the change in angular momentum, $\dot{\bar{H}}$ [28].

$$\bar{T}_{ext} = I\dot{\bar{\omega}} = \dot{\bar{H}} \tag{3.1}$$

The angular acceleration and angular momentum relationship given in Equation 3.1 is assuming a fixed inertial reference frame. In order to represent the motion of the air bearing in the ABBF coordinate system that moves with respect to the FIR coordinate system, the additional cross product of the angular velocity vector $\bar{\omega}$ with the angular momentum vector $\bar{H}$ given in Equation 3.2 must be included [28].

$$\bar{T}_{ext} = \dot{\bar{H}} + \bar{\omega} \times \bar{H} \tag{3.2}$$

The angular momentum vector $\bar{H}$ represents the total angular momentum of the air bearing system. Because the air bearing uses reaction wheels for attitude control, it is important to differentiate between the angular momentum of the air bearing itself and the angular momentum of the reaction wheels. Equation 3.3 shows how the angular momentum of the system is broken down [40].

$$\bar{H}_{total} = \bar{H}_{AB} + \bar{H}_{RW} \tag{3.3}$$

Substituting Equation 3.3 in Equation 3.2 gives Equation 3.4. This equation differentiates between the angular momentum of the air bearing and that of the reaction wheels. This differentiation is an important step towards developing control laws that will manipulate the angular velocity of the reaction wheels in order to achieve the desired angular velocity of the air bearing while keeping the angular momentum of the system at a constant.

$$\bar{T}_{ext} = \dot{\bar{H}}_{AB} + \bar{\omega} \times \bar{H}_{AB} + \dot{\bar{H}}_{RW} + \bar{\omega} \times \bar{H}_{RW} \tag{3.4}$$

As shown in Equation 3.1, change in angular momentum is equal to inertia times angular acceleration. Making this substitution into Equation 3.4 gives Equation 3.5, which represents the complete vector equation of rotational motion for the air bearing system. In order to differentiate between the angular velocity of the air bearing ω, and the angular velocity of the reaction wheels, Ω will be used for reaction wheel angular velocity.

$$\bar{T}_{ext} = I_{AB}\dot{\bar{\omega}} + \bar{\omega} \times I_{AB}\bar{\omega} + DI_{RW}\dot{\bar{\Omega}} + \bar{\omega} \times DI_{RW}\bar{\Omega} \tag{3.5}$$

Equation 3.5 can be used to prove the important concept that reaction wheels are simply angular momentum storage devices. They have the capability to transfer angular momentum to and from the vehicle to which they are attached, in this case the air bearing. However, they do not change the total angular momentum of the combined vehicle/reaction wheel system. Equation 3.5 shows the only variable that can change the total angular momentum of the system is the sum of external torques. As a reminder, external torques can be intentional control torques like attitude thrusters and magnetic torque coils or unintentional disturbance torques like aerodynamic drag and solar pressure. If the sum of external torques is assumed to be zero, the angular velocity of the reaction wheels $\bar{\Omega}$ can be

manipulated to achieve the desired angular velocity of the air bearing $\bar{\omega}$ without changing the total angular momentum of the system.

The constants in Equation 3.5 are the inertia matrices, I_{AB} and I_{RW}, and the direction cosine matrix (DCM), D. The symbolic inertia matrix of the complete air bearing I_{AB} is given in Equation 3.6. The air bearing inertia matrix is given in the ABBF coordinate system. This is the same coordinate system in which the equations of motion given in Equation 3.5 are developed. Therefore, no rotation matrix is necessary to include the air bearing inertia matrix in the equations of motion. As stated in Section 3.1.1, the ABBF coordinate system is aligned with the principle axes of the air bearing, so the air bearing inertia matrix is a diagonal matrix. The actual values will be determined using a SolidWorks model of the air bearing and given in Section 4.1.1.

$$I_{AB} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} kg * m^2 \tag{3.6}$$

The reaction wheel inertia matrix I_{RW} represents the inertia of the rotating portions of the reaction wheels and is given symbolically in Equation 3.7. Numerical values are given in Section 4.1.5. The rotating portions of the reaction wheel include the flywheel itself plus the internal rotating portions of the motor. The reaction wheel inertia matrix is given in the reaction wheel coordinate system. Since each of the three orthogonal reaction wheels are aligned with one of the three axes of the reaction wheel coordinate system, the reaction wheel inertia matrix is a diagonal matrix. The inertial value for each of the flywheels is the same and is calculated using the mass and mechanical properties of the wheels. The inertial value for each of the motors is taken from the manufacturer's specification sheet. A rotation matrix is necessary to include the reaction wheel inertia matrix in the equations of motion because the reaction wheel coordinate system is not aligned with the ABBF coordinate system.

$$I_{RW} = I_{flywheel} + I_{motor} = \begin{bmatrix} I_a & 0 & 0 \\ 0 & I_a & 0 \\ 0 & 0 & I_a \end{bmatrix} kg * m^2 \tag{3.7}$$

The final constant term D in Equation 3.5 represents the DCM between the reaction wheel coordinate system and the ABBF coordinate system. A DCM is a three by three rotation matrix that can be used to transform a three dimensional vector from one coordinate system to another. In this case, the reaction wheels are aligned with a frame that is rotated with respect to the frame used to define the equations of motion as described in Section 3.1.1. In order to represent the angular velocity of each reaction wheel independently of the others while still writing the equations of motion in the ABBF coordinate system, the DCM between the two coordinate systems must be defined. Using the DCM in this way will allow a desired air bearing angular velocity about any of its primary axes to be related to an independent angular acceleration command for each of the three orthogonal reaction wheels.

Before deriving the DCM from the reaction wheel coordinate system to the ABBF coordinate system, the development of a general DCM is explained. Figure 3-3 depicts a unit vector in the first axis $\hat{y}_1$ of a rotated coordinate system within the unrotated coordinate system represented by the $\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ unit vectors.

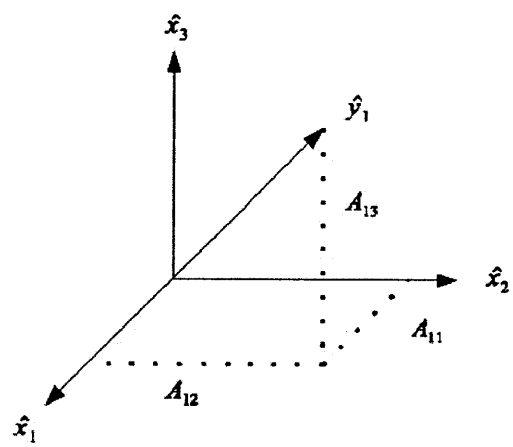


Figure 3-3: First Rotated Axis in Unrotated Frame [6]

The value A_{11} is the component of the $\hat{y}_1$ unit vector in the $\hat{x}_1$ direction. A_{12} is the component of the $\hat{y}_1$ unit vector in the $\hat{x}_2$ direction, and A_{13} is the component of $\hat{y}_1$ in the $\hat{x}_3$ direction. Because $\hat{x}_1$, $\hat{x}_2$, and $\hat{x}_3$ are orthogonal, the three components A_{11}, A_{12}, and A_{13} completely define the unit vector $\hat{y}_1$ in the $\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ coordinate frame as is shown in Equation 3.8 [6].

$$\hat{y}_1 = A_{11}\hat{x}_1 + A_{12}\hat{x}_2 + A_{13}\hat{x}_3 \tag{3.8}$$

The second and third unit vectors, $\hat{y}_2$ and $\hat{y}_3$, aligned with the second and third axes of the rotated coordinate frame respectively can be defined in the unrotated coordinate frame in a similar fashion as shown in Equations 3.9 and 3.10.

$$\hat{y}_2 = A_{21}\hat{x}_1 + A_{22}\hat{x}_2 + A_{23}\hat{x}_3 \tag{3.9}$$

$$\hat{y}_3 = A_{31}\hat{x}_1 + A_{32}\hat{x}_2 + A_{33}\hat{x}_3 \tag{3.10}$$

Placing Equations 3.8, 3.9, and 3.10 in matrix form gives Equation 3.11. A vector in the $\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ coordinate frame pre-multiplied by the matrix in Equation 3.11 gives the same vector represented in the $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ coordinate frame. Therefore, the matrix is a rotation matrix between the two coordinate frames.

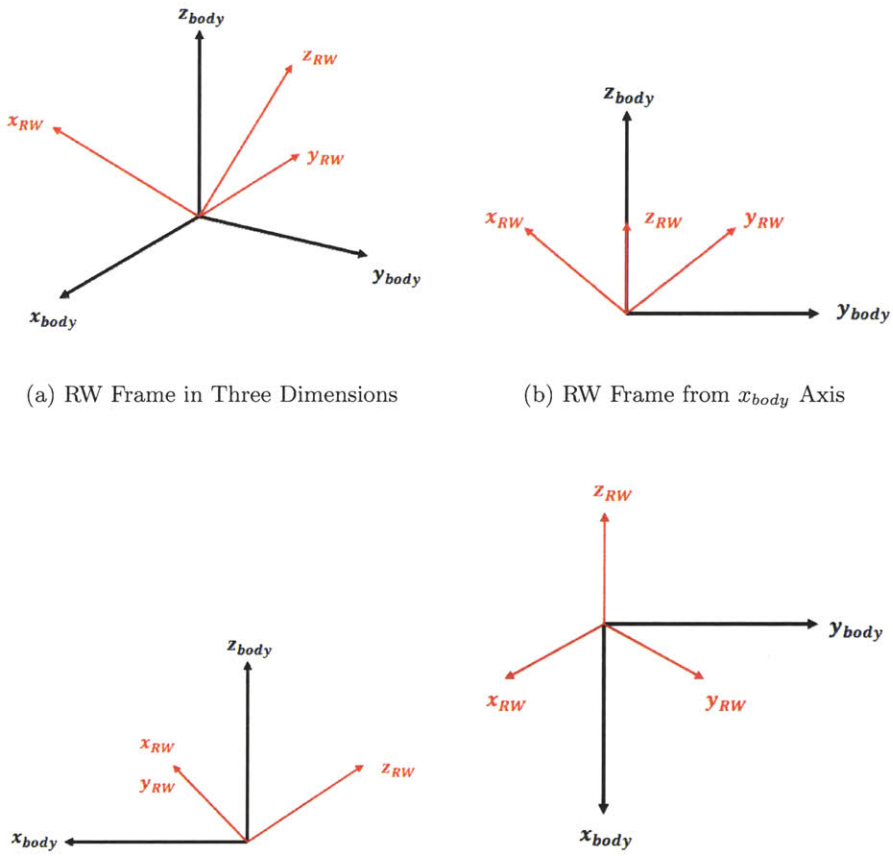
$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \hat{x}_3 \end{bmatrix} \tag{3.11}$$

As seen in Figure 3-3, A_{11} is the component of $\hat{y}_1$ projected in the $\hat{x}_1$ direction. This projection is defined as the cosine of the angle between $\hat{y}_1$ and $\hat{x}_1$ and is given in Equation 3.12 [54].

$$A_{11} = \cos\theta_{\hat{y}_1\hat{x}_1} \tag{3.12}$$

Using the relationship shown in Equation 3.12, all entries of the matrix given in Equation 3.11 are substituted to produce the DCM between the rotated and unrotated coordinate systems. The DCM is given in Equation 3.13. The DCM can be used to rotate any vector in the $\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ coordinate frame to the $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ coordinate frame and vise versa using the matrix transpose. However, in order to produce the DCM, all nine angles shown in Equation 3.13 must be known.

$$DCM = \begin{bmatrix} \cos\theta_{\hat{y}_1\hat{x}_1} & \cos\theta_{\hat{y}_1\hat{x}_2} & \cos\theta_{\hat{y}_1\hat{x}_3} \\ \cos\theta_{\hat{y}_2\hat{x}_1} & \cos\theta_{\hat{y}_2\hat{x}_2} & \cos\theta_{\hat{y}_2\hat{x}_3} \\ \cos\theta_{\hat{y}_3\hat{x}_1} & \cos\theta_{\hat{y}_3\hat{x}_2} & \cos\theta_{\hat{y}_3\hat{x}_3} \end{bmatrix} \tag{3.13}$$



(a) RW Frame in Three Dimensions

(b) RW Frame from x_{body} Axis

(c) RW Frame from y_{body} Axis

(d) RW Frame from z_{body} Axis

Figure 3-4: Reaction Wheel Frame in ABBF Frame

Figure 3-4 depicts the reaction wheel frame in the ABBF frame from several perspectives. Figure 3-4(d) makes clear that the orthogonal reaction wheel frame is symmetric about the z_{body} axis, and that the z_{RW} axis is in the negative x_{body} direction from this perspective. Using the information in the four subfigures of Figure 3-4, all nine required DCM angles can be determined.

Because the reaction wheel frame is symmetric about the z_{body} axis, a vector along the z_{body} axis is in the $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ direction in the reaction wheel frame. To find the angle

between any of the reaction wheel frame axes and the z_{body} axis (all three are the same), use the definition of the vector dot product given in Equation 3.14 [54].

$$A \bullet B = |A||B| \cos \theta_{AB} \tag{3.14}$$

Using Equation 3.14, substitute $A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$ to represent a vector along the z_{body} axis and $B = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ to represent a vector along the x_{RW} axis (both given in the reaction wheel frame). Solving this equation for θ results in the angle $\theta_{z_b x_R} = 54.74°$. Due to the symmetry about the z_{body} axis, the angle between z_{body} and the remaining two reaction wheel axes is also 54.74°. Using the dot product definition in a similar fashion, the remaining six angles can be found. All nine angles are listed in Table 3.1.

Table 3.1: Angles Between RW Axes and ABBF Axes

Angle Between:	Angle Label	Magnitude (deg)
x_{body} and x_{RW}	$\theta_{x_b x_R}$	65.9052°
x_{body} and y_{RW}	$\theta_{x_b y_R}$	65.9052°
x_{body} and z_{RW}	$\theta_{x_b z_R}$	144.7356°
y_{body} and x_{RW}	$\theta_{y_b x_R}$	135.0000°
y_{body} and y_{RW}	$\theta_{y_b y_R}$	45.0000°
y_{body} and z_{RW}	$\theta_{y_b z_R}$	90.0000°
z_{body} and x_{RW}	$\theta_{z_b x_R}$	54.7356°
z_{body} and y_{RW}	$\theta_{z_b y_R}$	54.7356°
z_{body} and z_{RW}	$\theta_{z_b z_R}$	54.7356°

Using the angles given in Table 3.1 and the general definition of a direction cosine matrix given in Equation 3.13, the DCM used to rotate a vector from the reaction wheel frame to the ABBF frame is given in Equation 3.15. The DCM is represented by D in the vector equation of motion and is given in numeric form below because this matrix will be constant regardless of software or hardware changes to the air bearing.

$$D = \begin{bmatrix} \cos 65.91° & \cos 65.91° & \cos 144.74° \\ \cos 135.00° & \cos 45.00° & \cos 90.00° \\ \cos 54.74° & \cos 54.74° & \cos 54.74° \end{bmatrix} = \begin{bmatrix} 0.40825 & 0.40825 & -0.81650 \\ -0.70711 & 0.70711 & 0.00000 \\ 0.57735 & 0.57735 & 0.57735 \end{bmatrix} \tag{3.15}$$

All constants in the vector rotational equation of motion given in Equation 3.5 are now defined. The equation can now be used to model the motion of the combined air bearing, reaction wheel system in the MATLAB simulation as well as be used to develop control algorithms in the simulation and in the avionics code used on the physical testbed.

3.2 MATLAB Simulation

The supporting MATLAB simulation of the ADCS testbed provides several advantages over operating the testbed without using a model. First of all, the testbed will likely need to be modified either slightly or significantly in order to accommodate a given test campaign. The simulation allows for those in charge of testing to determine the most efficient means of testbed modification that meet the requirements of the test campaign. The modifications, either hardware or software, can be integrated into the simulation first in order to predict the response of the modified system to a given input. If the simulation predicts that the modifications will meet requirements while remaining within the physical limitations of the air bearing, the modifications can then be implemented in the hardware or software.

The simulation also provides an expanded testbed state estimate. The baseline testbed uses a three axis inertial measurement unit and a three axis magnetometer as attitude sensors and four encoders to measure the magnitudes of the four reaction wheels' angular rates. These sensors provide a limited measurement of the testbed state. The simulation can be used to predict unobservable states like testbed angular acceleration, reaction wheel angular direction, reaction wheel electrical current and voltage, and the direction cosine matrix between the FIR frame and the true ABBF frame.

Once updated to match the current physical testbed configuration, the simulation can produce response plots to be compared with the measurable responses of the physical testbed to a given input. Matching responses between the model and the physical testbed provides supporting evidence so that the physical response is not unexplainable but rather an expected response based on vehicle dynamics. Furthermore, the additional state estimates produced by the simulation provide insight into exactly what characteristics of the testbed drive the response of the testbed. This information can be used to predict limitations of the system being tested as well as provide designers with information on how to best modify the system to meet requirements if the current configuration is found to be insufficient.

3.2.1 Simulation Development

The simulation is developed in the MATLAB Simulink environment. Simulink provides a simulated continuous real-time environment for which to run a dynamic model with feedback. Rather than using written code, Simulink uses a block structure similar to a flow chart. The block structure allows for easy to recognize loop organization and feedback paths. The block structure also provides ease of modification in order to update the simulation based on proposed testbed changes. Figure 3-5 is a screen-shot of the outer most layer of the Simulink simulation. Though details are hard to see in this picture, the looped block structure is clearly recognized. The block structure also allows for easy manipulation of the simulation. Certain aspects like sensor measurements or disturbance torques can be turned on and off simply by connecting or disconnecting an arrow in the Simulink environment. Making such a change in line by line code would likely be more time consuming and less obvious to recognize.

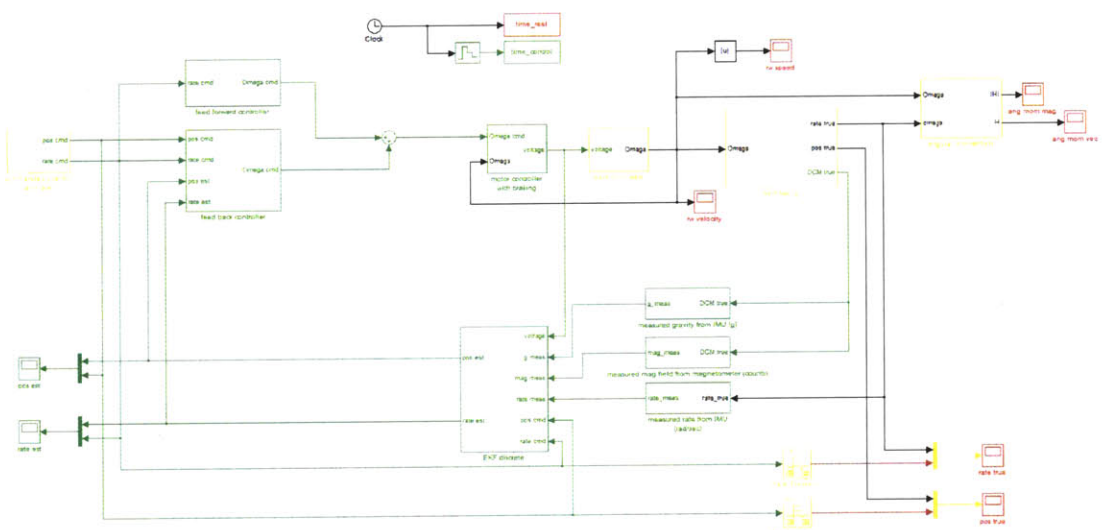


Figure 3-5: Simulation Screen-shot

Simulink has several advantages besides the block structure that make it the ideal platform to build the ADCS testbed simulation. First, the simulated continuous time environment allows for accurate estimation of the dynamic response of the air bearing to a given input. The physical air bearing responds to an input in continuous real time. Therefore, the simulation is most accurate if it can model the air bearing dynamics as a continuous system rather than a discrete system. Simulink also provides the ability to operate a model at dif-

ferent time steps within its environment. This is beneficial because it allows the simulation not only to model the continuous dynamics of the physical air bearing, but also to model the discrete attitude estimation and control algorithm running on the testbed's avionics computer. The physical computer runs the estimation and control algorithm at a discrete time step. The discrete nature of this process must be modeled in order to accurately predict the characteristics of the estimation and control algorithm. Using a continuous model of the air bearing dynamics and a discrete model of the estimation and control algorithm allows the simulation to provide the best estimate of the testbed state.

Among other useful tools, Simulink uses a color coding scheme to make clear which sections of the simulation are running in a given time step as can be seen in Figure 3-5. In this case, the blocks and lines appearing in black represent the continuous portions of the simulation like the air bearing plant and the angular momentum calculator. The blocks and lines appearing in green represent the discrete portions, which include the sensors, estimator, and controller. Yellow blocks represent subsystems containing blocks with different time steps, and red blocks represent the discretized output of the continuous states within the model. This discretization is necessary to create data arrays within the MATLAB environment, but it should occur at a higher frequency than the discrete control algorithm to accurately represent the continuous information. In general, discretization of continuous states should occur at no less than ten times the frequency of the control algorithm to maintain accuracy.

Though the Simulink environment uses a block structure, MATLAB code in the form of m files or embedded code can be called by any block within the simulation requiring the functions of a given piece of MATLAB code. This feature is useful when a series of mathematical operations are required that would be tedious to develop using the default block structure. Furthermore, MATLAB m files can be called before beginning the real time simulation to provide initial conditions used by the Simulink simulation. The simulation can also send data arrays as variables to the MATLAB workspace for post analysis. This feature allows the simulation to provide expanded testbed state estimation over the entire test duration.

3.2.2 Plant Module

The simulation plant module consists of the devices on the air bearing that undergo physical motion during a test. These include the three orthogonal reaction wheels and the air bearing

itself. The plant module is the only module operated in a continuous environment in the simulation in order to best model the actual motion of the reaction wheels and air bearing. The reaction wheel section comes first in the plant module because the reaction wheels receive commands from the control algorithm to achieve a commanded angular velocity. The reaction wheel angular velocity is the input into the air bearing section, since the torque due to the change in reaction wheel angular velocity is what drives motion in the air bearing.

3.2.2.1 Reaction Wheel Plant Block

The Pololu Trex motor controllers on the air bearing drive the reaction wheels by commanding a voltage. In order to accurately simulate this, the reaction wheel block receives three voltage inputs from the motor controller block within the control module. Each voltage input corresponds to each of the three reaction wheels. The simulated reaction wheel response can be found by applying the voltage input to the reaction wheel equations of motion. However, the reaction wheel equations of motion must be developed before they can be used. Figure 3-6 shows the simplified electrical representation of a direct current motor, which is the type of motor used for the reaction wheels.

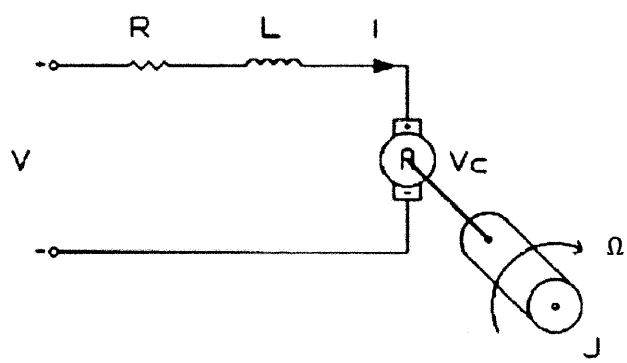


Figure 3-6: Electrical Diagram of Direct Current (DC) Motor [16]

Equation 3.16 gives Kirchoff's voltage law applied to the reaction wheel system. Kirchoff's voltage law states that the sum of all voltages around a loop must equal zero. V is the source voltage from the motor controller. The other voltages are defined in Equations 3.17 through 3.19 [16].

$$V - V_R - V_L - V_c = 0 \tag{3.16}$$

$$V_R = i * R \tag{3.17}$$

$$V_L = \dot{i} * L \tag{3.18}$$

$$V_c = K_e * \Omega \tag{3.19}$$

Substituting the above equations into Kirchoff's voltage law gives Equation 3.20, which is the dynamic equation representing the electrical characteristics of the reaction wheel. The equation is solved for $\dot{i}$.

$$\dot{i} = -\frac{K_e}{L}\Omega - \frac{R}{L}i + \frac{V}{L} \tag{3.20}$$

The equation representing the mechanical characteristics of the reaction wheel begin with Newton's second law for rotational bodies given in Equation 3.21, which states that the sum of torques on the system equals the inertia of the system times the system's angular acceleration. T_e and T_Ω are defined in Equations 3.22 and 3.23 [16].

$$T_e - T_\Omega = J\dot{\Omega} \tag{3.21}$$

$$T_e = K_t * i \tag{3.22}$$

$$T_\Omega = b * \Omega \tag{3.23}$$

Substituting the torque equations into Newton's second law gives Equation 3.24, which is the dynamic equation representing the mechanical characteristics of the reaction wheel. The equation is solved for $\dot{\Omega}$. The electrical and mechanical dynamic equations can be combined to model the motion of the three reaction wheels on the air bearing given voltage inputs from the motor controllers.

$$\dot{\Omega} = -\frac{b}{J}\Omega + \frac{K_t}{J}i \tag{3.24}$$

The constants in the electrical and mechanical dynamic equations are unique to each motor. Table 3.2 lists each constant and its value as given in the data sheet provided by the manufacturer, Motion Control Group. The specification sheet can be found at Motion Control Group's website.[1] The exceptions are the inertia J which is the sum of the motor inertia as listed in the the data sheet and the inertia of the attached fly wheel, and the damping ratio b which is empirically determined. The damping ratio is different than the value listed in the data sheet because of the attached fly wheel.

Table 3.2: Reaction Wheel Motor Constants

Constant	Name	Value	Units
J	Inertia	0.00725	$kg * m^2$
K_e	Voltage Constant	0.071	$V/rad/sec$
K_t	Torque Constant	0.070	Nm/A
b	Damping Ratio	0.000178	$Nm/rad/sec$
L	Inductance	0.00339	H
R	Resistance	1.52	Ω

Figure 3-7 shows the actual Simulink diagram representing the two reaction wheel equations of motion. The top portion is the block representation of the electrical characteristics of the wheels, and the bottom portion is the block representation of the mechanical characteristics of the wheels. The equations are coupled, which is why the integrated outputs are fed back to each other. Though the diagram only shows one set of equations, a three vector of voltages are provided at the input, and the model calculates the angular velocity of all three wheels simultaneously. The output is a three vector containing three reaction wheel velocities. The reaction wheel block is the first to be modeled in a continuous fashion. The voltage input is provided at the frequency of the control algorithm, and the input is modified to provide a continuous input to the reaction wheel equations of motion. The modification is done by the Rate Transition block in Figure 3-7.

[1] `http://www.ametektip.com/index.php?option=com\_catalog\&view=models&which=catalogs\&id=399\&Itemid=107\&lang=en`

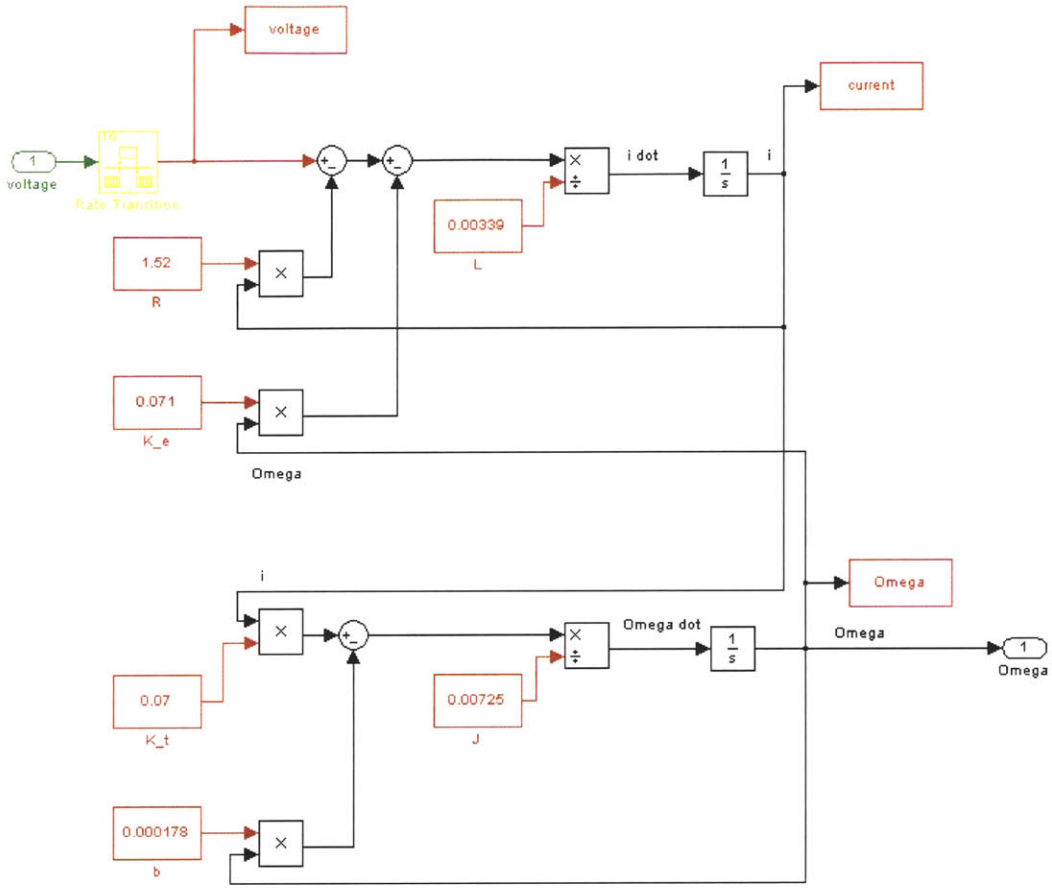


Figure 3-7: Reaction Wheel Plant Simulink Diagram

3.2.2.2 Air Bearing Plant Block

The vector equation of motion for the air bearing is developed in Section 3.1.2 and given in Equation 3.5. However, to be applied correctly, the equation must be solved for the desired output, which is the angular acceleration of the air bearing $\dot{\omega}$. For the baseline testbed, the reaction wheels are the only source of torque on the air bearing, and they do not provide external torques. There are also assumed to be no external disturbance torques on the system. Therefore, the left side of Equation 3.5 will be set to zero. The manipulated equation is given in Equation 3.25. If torque coils or thrusters are added for attitude control, their torques will be included in the external torque variable on the left side of Equation 3.5, and the below equation will have an additional term.

$$\dot{\bar{\omega}} = I_{AB}^{-1}\Big[-(\bar{\omega} \times I_{AB}\bar{\omega}) - DI_{RW}\dot{\bar{\Omega}} - (\bar{\omega} \times DI_{RW}\bar{\Omega})\Big] \tag{3.25}$$

Figure 3-8 shows the Simulink block diagram of Equation 3.25. The input is the three vector of reaction wheel angular velocities $\bar{\Omega}$, and the derivative of the reaction wheel angular velocity vector, which is the reaction wheel angular acceleration vector $\dot{\bar{\Omega}}$. The integrated output, which is the three vector of air bearing angular velocities $\bar{\omega}$ is fed back into the equation. The air bearing block is modeled in a continuous fashion similar to the reaction wheel block in order to best simulate the physical air bearing motion. The desired outputs are air bearing angular velocity and its integral, air bearing angular orientation. These outputs represent the simulated true attitude state of the air bearing.

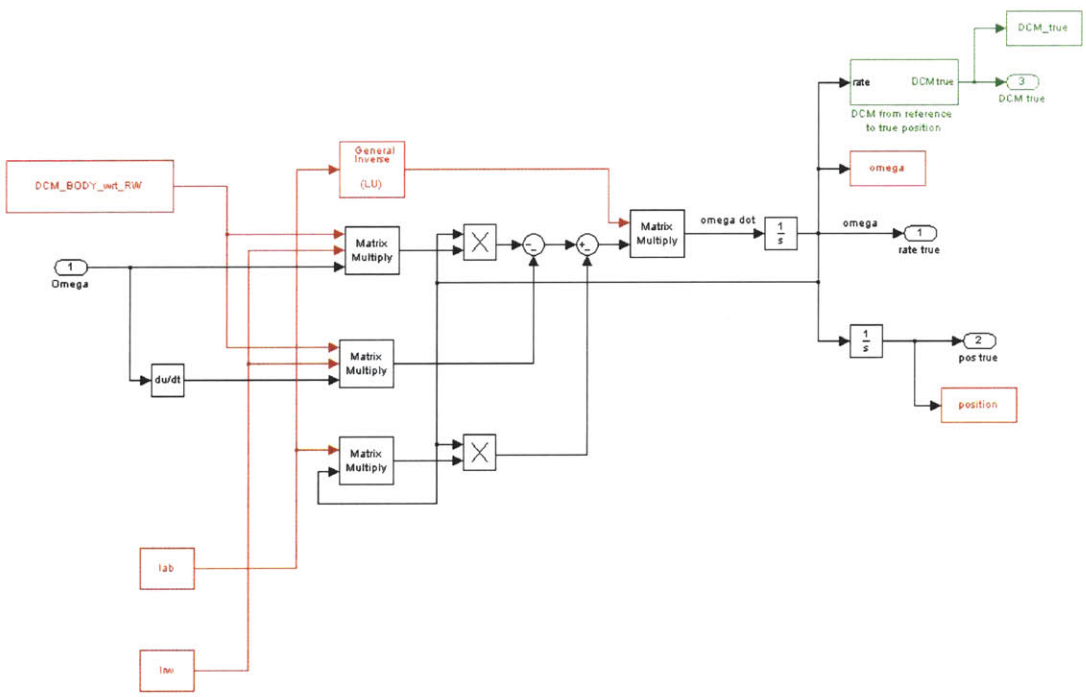


Figure 3-8: Air Bearing Plant Simulink Diagram

In the top right corner of Figure 3-8 is located the block calculating the DCM from the FIR frame to the air bearing's current angular orientation as defined by the true ABBF frame. The DCM is necessary to provide estimated attitude measurements similar to those provided by the IMU's accelerometers and the magnetometer. Since these measurement devices are used on the air bearing, they must be modeled in the simulation. As previously described, the DCM is a three by three matrix that can transform a vector in this case

from the FIR frame to the ABBF frame and visa versa. Equation 3.26 gives the means to discretely calculate the DCM using the air bearing's state [22].

$$DCM_{k+1} = e^{-skew(\bar{\omega}_k * dt)} * DCM_k \tag{3.26}$$

Figure 3-9 gives the Simulink diagram located within the DCM block that calculates the true DCM from the FIR frame to the ABBF frame. The diagram is a block representation of Equation 3.26. Due to the high computational requirement of a matrix exponential, this block operates at the discrete time step. Operating at the discrete time step introduces some amount of error in the system, though the benefit of simulation runtime speed outweighs the cost of introduced error.

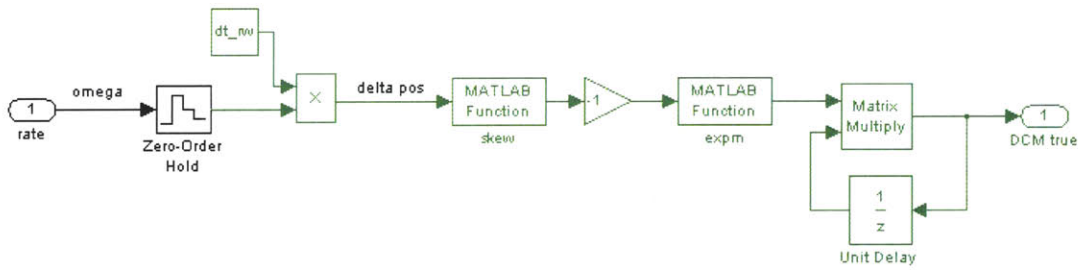


Figure 3-9: Direction Cosine Matrix Simulink Diagram

The true air bearing angular orientation and rate as well as the DCM from the FIR frame to the ABBF frame are the primary outputs of the plant module. These outputs feed into the sensor portion of the estimation module to provide a simulated state measurement and are used to test the accuracy of the estimation algorithm. The true state is also the most beneficial output to be used in comparison with actual estimated state data collected from various tests on the air bearing.

3.2.3 Estimation Module

The purpose of the simulation estimation block is to produce the best estimate of the air bearing's state. The estimation block first uses inputs from the control law in order to propagate a linear state space model of the reaction wheels and air bearing. The estimation block also takes inputs from sensors on-board the air bearing and uses the measurements to update the linear model. The combination of propagating the linear model and updating

the output of the model based on sensor measurements produces the best estimate of the air bearing's angular rate and orientation. This type of estimation process is commonly known as an Extended Kalman Filter (EKF).

3.2.3.1 Reaction Wheel State Space Block

The input to the reaction wheel state space block is the same as the input to the reaction wheel plant block, which is the three vector of commanded voltages. Ideally, the reaction wheel angular velocities would not need to be estimated since there are encoders located on each of the wheels. These encoders could provide reaction wheel angular velocity measurements that could be fed directly into the air bearing state space model. However, due to computational limitations on the Arduino processors used on the air bearing, only one encoder input from each reaction wheel can be processed. The single encoder input allows for the magnitude of the reaction wheel angular velocity to be measured, but not direction. Without direction, the measurement is insufficient as an input to the air bearing state space model. Therefore, the reaction wheel angular velocities must be estimated using the voltage inputs from the control law, which are available on the actual air bearing.

The reaction wheel equations of motion developed in Section 3.2.2 must be manipulated into a state space model for use in the estimation block. Equations 3.27 and 3.28 below represent the general continuous state space equations. The variable x represents the state of the system being modeled, and u represents the inputs to the system. The matrix A is the state matrix used to calculate the derivative of the state, which can be used to determine the state at a future point in time. The matrix B is the input matrix used to correctly scale and apply the inputs to the correct states. The matrix C is the output matrix used to correctly scale and determine observability of the states [39].

$$\dot{x} = Ax + Bu \tag{3.27}$$

$$y = Cx \tag{3.28}$$

From Equations 3.20 and 3.24 and the fact that there are three reaction wheels, the state vector x_{rw} is given in Equation 3.29. The derivative of the state vector consists of the

derivatives of each of the components of the state vector. The input vector u_{rw} consists of the three voltages and is given below in Equation 3.30.

$$x_{rw} = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \\ i_x \\ i_y \\ i_z \end{bmatrix} \tag{3.29}$$

$$u_{rw} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \tag{3.30}$$

The state matrix and input matrix must be formed so that each row of the matrices reproduces one of the dynamic equations of motion for each of the three reaction wheels. Because there are two dynamic equations of motion for each wheel and three wheels, there are six rows in the state and input matrices. Fortunately, the reaction wheel dynamic equations are linear so they can be directly converted to a state space model without being linearized. This allows for a more accurate model of the reaction wheel system and reduces the error induced by having to estimate the reaction wheel angular velocities instead of using direct measurements from the encoders. Equation 3.31 gives the state matrix A_{rw} as a function of the components of the electrical and mechanical dynamic equations.

$$A_{rw} = \begin{bmatrix} -\frac{b}{J} & 0 & 0 & \frac{K_t}{J} & 0 & 0 \\ 0 & -\frac{b}{J} & 0 & 0 & \frac{K_t}{J} & 0 \\ 0 & 0 & -\frac{b}{J} & 0 & 0 & \frac{K_t}{J} \\ -\frac{K_e}{L} & 0 & 0 & -\frac{R}{L} & 0 & 0 \\ 0 & -\frac{K_e}{L} & 0 & 0 & -\frac{R}{L} & 0 \\ 0 & 0 & -\frac{K_e}{L} & 0 & 0 & -\frac{R}{L} \end{bmatrix} \tag{3.31}$$

Equation 3.32 gives the input matrix B_{rw} as a function of the components of the electrical dynamic equation. The mechanical dynamic equation is not a function of the voltage input.

Therefore, the input matrix does not apply the voltage input to the mechanical portion of the state.

$$B_{rw} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{L} & 0 & 0 \\ 0 & \frac{1}{L} & 0 \\ 0 & 0 & \frac{1}{L} \end{bmatrix} \tag{3.32}$$

The desired outputs of the state space model are the angular velocities of the three reaction wheels. These values are just a portion of the full state, so the output matrix C_{rw} is constructed to pull just these values from the full state. Equation 3.33 gives the output matrix of the reaction wheel state space model.

$$C_{rw} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.33}$$

Substituting the populated state, input, and output matrices into the general state space equations gives the complete continuous reaction wheel state space model shown in Equations 3.34 and 3.35.

$$\begin{bmatrix} \dot{\Omega}_x \\ \dot{\Omega}_y \\ \dot{\Omega}_z \\ \dot{i}_x \\ \dot{i}_y \\ \dot{i}_z \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & 0 & 0 & \frac{K_t}{J} & 0 & 0 \\ 0 & -\frac{b}{J} & 0 & 0 & \frac{K_t}{J} & 0 \\ 0 & 0 & -\frac{b}{J} & 0 & 0 & \frac{K_t}{J} \\ -\frac{K_e}{L} & 0 & 0 & -\frac{R}{L} & 0 & 0 \\ 0 & -\frac{K_e}{L} & 0 & 0 & -\frac{R}{L} & 0 \\ 0 & 0 & -\frac{K_e}{L} & 0 & 0 & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \\ i_x \\ i_y \\ i_z \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{L} & 0 & 0 \\ 0 & \frac{1}{L} & 0 \\ 0 & 0 & \frac{1}{L} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \tag{3.34}$$

$$\begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \\ i_x \\ i_y \\ i_z \end{bmatrix} \tag{3.35}$$

The above equations represent the continuous state space model of the reaction wheels. In order to be coded as part of the estimator on the air bearing, the model must be converted from a continuous to a discrete model. Therefore, the simulated model must be discrete as well. Equations 3.36 and 3.37 represent the general discrete state space equations [42]. Rather than calculating the derivative of the state like the continuous state space equations, the discrete equations calculate the state at the next time step.

$$x_{k+1} = A_d x_k + B_d u_k \tag{3.36}$$

$$y_k = C x_k \tag{3.37}$$

As seen in the above discrete state space equations, the new matrices needed to apply these equations are the state matrix A_d and the input matrix B_d. These can both be calculated using the continuous state and input matrices along with the discretization time step. Equation 3.38 shows how to calculate the discrete state matrix [42].

$$A_d = e^{A*dt} \tag{3.38}$$

If the continuous state matrix A is nonsingular and the inverse can be determined, Equation 3.39 shows the quick method for calculating the discrete input matrix. However, if the state matrix does not have an inverse, convolution is necessary to find the discrete input matrix as shown in Equation 3.40 [42].

$$B_d = A^{-1}(A_d - I)B \tag{3.39}$$

$$B_d = \int_0^{dt} e^{A*\tau} d\tau * B \tag{3.40}$$

Because the matrix exponential used to calculate the discrete matrices is complex, the discrete matrices cannot be given in symbolic form. However, Equations 3.41 and 3.42 give the discrete state space equations for the three reaction wheels without expanding the state and input matrices. Note that there is no difference in the output equation between the continuous and discrete state space models. These equations will be directly coded into the Simulink simulation as the reaction wheel angular velocity estimator.

$$\begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \\ i_x \\ i_y \\ i_z \end{bmatrix}_{k+1} = A_{d\,rw} \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \\ i_x \\ i_y \\ i_z \end{bmatrix}_k + B_{d\,rw} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}_k \tag{3.41}$$

$$\begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \\ i_x \\ i_y \\ i_z \end{bmatrix}_k \tag{3.42}$$

Figure 3-10 shows the above equations transformed into the Simulink block diagram format. The voltage vector is the input and the reaction wheel angular velocity vector is the output. The model runs at the discrete time step to best simulate the same process on the air bearing processors. The angular velocity vector is immediately differentiated in order to produce the required input to the air bearing state space model, which is the three vector of reaction wheel angular accelerations.

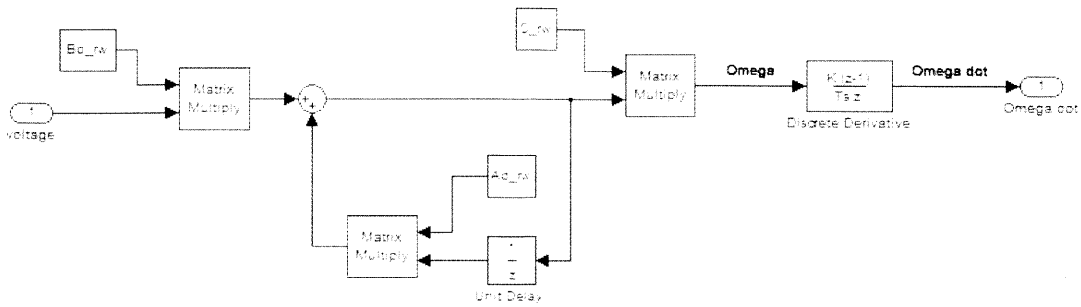


Figure 3-10: Reaction Wheel Discrete State Space Diagram

3.2.3.2 Air Bearing State Space Block

Just as with the reaction wheel equations of motion, the air bearing vector equation of motion must be manipulated into a state space model. However, the air bearing equation of motion will not be as simple as the reaction wheel equations because the air bearing equation is nonlinear. Several steps are required before the equation can be placed within a state space model. First, the equation must be expanded from the vector form given in Equation 3.25 to scalar form. Equations 3.43, 3.44, and 3.45 give the respective ABBF x, y, and z axis scalar equations of motion expanded from the vector form given in Equation 3.25.

$$\dot{\omega}_x = \frac{1}{I_{xx}}\Big[I_{yy}\omega_y\omega_z - I_{zz}\omega_z\omega_y - I_a(D_{11}\dot{\Omega}_x + D_{12}\dot{\Omega}_y + D_{13}\dot{\Omega}_z) + \omega_z I_a(D_{21}\Omega_x + D_{22}\Omega_y + D_{23}\Omega_z) - \omega_y I_a(D_{31}\Omega_x + D_{32}\Omega_y + D_{33}\Omega_z)\Big] \quad (3.43)$$

$$\dot{\omega}_y = \frac{1}{I_{yy}}\Big[I_{zz}\omega_z\omega_x - I_{xx}\omega_x\omega_z - I_a(D_{21}\dot{\Omega}_x + D_{22}\dot{\Omega}_y + D_{23}\dot{\Omega}_z) + \omega_x I_a(D_{31}\Omega_x + D_{32}\Omega_y + D_{33}\Omega_z) - \omega_z I_a(D_{11}\Omega_x + D_{12}\Omega_y + D_{13}\Omega_z)\Big] \quad (3.44)$$

$$\dot{\omega}_z = \frac{1}{I_{zz}}\Big[I_{xx}\omega_x\omega_y - I_{yy}\omega_y\omega_x - I_a(D_{31}\dot{\Omega}_x + D_{32}\dot{\Omega}_y + D_{33}\dot{\Omega}_z) + \omega_y I_a(D_{11}\Omega_x + D_{12}\Omega_y + D_{13}\Omega_z) - \omega_x I_a(D_{21}\Omega_x + D_{22}\Omega_y + D_{23}\Omega_z)\Big] \quad (3.45)$$

The next step is to linearize the scalar equations. To linearize an equation, the partial derivative must be taken with respect to each variable within the equation. The partials are then evaluated at an equilibrium point chosen to be at or near where the actual system is expected to be operating. The remaining coefficient of each evaluated partial derivative is then multiplied by the variable by which the partial derivative is taken to produce the linear component for each variable. The resulting equation is a function of linear components for each original variable. For example, Equation 3.47 shows the general process for producing the linearized function $g(m, n, h)$ from the nonlinear function $f(m, n, h)$ about the equilibrium points given in Equation 3.46 [21].

$$eq\ vec = \begin{bmatrix} m_{eq} \\ n_{eq} \\ h_{eq} \end{bmatrix} \quad (3.46)$$

$$g(m, n, h) = \frac{\partial f}{\partial m}\bigg|_{eq\ vec} m + \frac{\partial f}{\partial n}\bigg|_{eq\ vec} n + \frac{\partial f}{\partial h}\bigg|_{eq\ vec} h \quad (3.47)$$

The linearization process will be shown for the scalar equation of motion about the ABBF x axis and omitted for the other two equations since they follow a similar process. The final linearized equations will be given for all three equations of motion. Equation 3.49 gives the symbolic equation of motion for the ABBF x axis linearized about the equilibrium points given in Equation 3.48.

$$eq\,vec = \begin{bmatrix} \omega_{x\,eq} \\ \omega_{y\,eq} \\ \omega_{z\,eq} \\ \Omega_{x\,eq} \\ \Omega_{y\,eq} \\ \Omega_{z\,eq} \\ \dot{\Omega}_{x\,eq} \\ \dot{\Omega}_{y\,eq} \\ \dot{\Omega}_{z\,eq} \end{bmatrix} \tag{3.48}$$

$$\begin{aligned} \dot{\omega}_{x\,lin} = & \left.\frac{\partial \dot{\omega}_x}{\partial \omega_x}\right|_{eq\,vec} \omega_x + \left.\frac{\partial \dot{\omega}_x}{\partial \omega_y}\right|_{eq\,vec} \omega_y + \left.\frac{\partial \dot{\omega}_x}{\partial \omega_z}\right|_{eq\,vec} \omega_z \\ & + \left.\frac{\partial \dot{\omega}_x}{\partial \Omega_x}\right|_{eq\,vec} \Omega_x + \left.\frac{\partial \dot{\omega}_x}{\partial \Omega_y}\right|_{eq\,vec} \Omega_y + \left.\frac{\partial \dot{\omega}_x}{\partial \Omega_z}\right|_{eq\,vec} \Omega_z \\ & + \left.\frac{\partial \dot{\omega}_x}{\partial \dot{\Omega}_x}\right|_{eq\,vec} \dot{\Omega}_x + \left.\frac{\partial \dot{\omega}_x}{\partial \dot{\Omega}_y}\right|_{eq\,vec} \dot{\Omega}_y + \left.\frac{\partial \dot{\omega}_x}{\partial \dot{\Omega}_z}\right|_{eq\,vec} \dot{\Omega}_z \end{aligned} \tag{3.49}$$

The nine partial differentials given in the above equation must now be calculated. Equations 3.50 through 3.52 give the partial derivatives of $\dot{\omega}_x$ with respect to the air bearing angular velocities about the three ABBF axes.

$$\frac{\partial \dot{\omega}_x}{\partial \omega_x} = 0 \tag{3.50}$$

$$\frac{\partial \dot{\omega}_x}{\partial \omega_y} = \frac{1}{I_{xx}} \Big[I_{yy}\omega_z - I_{zz}\omega_z - I_a(D_{31}\Omega_x + D_{32}\Omega_y + D_{33}\Omega_z) \Big] \tag{3.51}$$

$$\frac{\partial \dot{\omega}_x}{\partial \omega_z} = \frac{1}{I_{xx}} \Big[I_{yy}\omega_y - I_{zz}\omega_y + I_a(D_{21}\Omega_x + D_{22}\Omega_y + D_{23}\Omega_z) \Big] \tag{3.52}$$

Equations 3.53 through 3.55 give the partial derivatives of $\dot{\omega}_x$ with respect to the angular velocities of the three reaction wheels.

$$\frac{\partial \dot{\omega}_x}{\partial \Omega_x} = \frac{1}{I_{xx}} \Big[\omega_z I_a D_{21} - \omega_y I_a D_{31} \Big] \tag{3.53}$$

$$\frac{\partial \dot{\omega}_x}{\partial \Omega_y} = \frac{1}{I_{xx}}\Big[\omega_z I_a D_{22} - \omega_y I_a D_{32}\Big] \tag{3.54}$$

$$\frac{\partial \dot{\omega}_x}{\partial \Omega_z} = \frac{1}{I_{xx}}\Big[\omega_z I_a D_{23} - \omega_y I_a D_{33}\Big] \tag{3.55}$$

Equations 3.56 through 3.58 give the partial derivatives of $\dot{\omega}_x$ with respect to the angular accelerations of the three reaction wheels.

$$\frac{\partial \dot{\omega}_x}{\partial \dot{\Omega}_x} = -\frac{I_a D_{11}}{I_{xx}} \tag{3.56}$$

$$\frac{\partial \dot{\omega}_x}{\partial \dot{\Omega}_y} = -\frac{I_a D_{12}}{I_{xx}} \tag{3.57}$$

$$\frac{\partial \dot{\omega}_x}{\partial \dot{\Omega}_z} = -\frac{I_a D_{13}}{I_{xx}} \tag{3.58}$$

Using these partial derivatives, the linearized equation of motion can be found about any set of equilibrium points. However, in order to simplify the linearized equations, the equilibrium points for the air bearing are all chosen to be zero. Not only does this choice simplify the linearized equations, but it also provides the best estimate of the air bearing's state in most cases. Though the air bearing may assume angular rates during a test, and the reaction wheels most certainly will assume some angular rate, these rates vary significantly above and below zero during most tests. Even if the initial velocity of the reaction wheels is a positive value, their velocity will likely reduce or switch directions during a test. If the model were linearized about the initial velocity, it would be less accurate once the wheels change direction. If the model is linearized about zero, the model is equally accurate regardless of reaction wheel direction. This argument is the same for air bearing body rates. Equation 3.59 gives the evaluated equation of motion for the ABBF x axis linearized about all zero equilibrium points.

$$\dot{\omega}_{x\,lin} = -\frac{I_a}{I_{xx}}\Big[D_{11}\dot{\Omega}_x + D_{12}\dot{\Omega}_y + D_{13}\dot{\Omega}_z\Big] \tag{3.59}$$

Equations 3.60 and 3.61 give the evaluated equations of motion for the ABBF y and z axes linearized about all zero equilibrium points respectively.

$$\dot{\omega}_{y\,lin} = -\frac{I_a}{I_{yy}}\left[D_{21}\dot{\Omega}_x + D_{22}\dot{\Omega}_y + D_{23}\dot{\Omega}_z\right] \tag{3.60}$$

$$\dot{\omega}_{z\,lin} = -\frac{I_a}{I_{zz}}\left[D_{31}\dot{\Omega}_x + D_{32}\dot{\Omega}_y + D_{33}\dot{\Omega}_z\right] \tag{3.61}$$

A state space model can now be defined using the above equations of motion. The state space model can be defined in many ways, but the most straightforward and beneficial definition in this case is to define the state vector as the three angular orientations with respect to the ABBF frame and the three angular rates with respect to the ABBF frame since these are the desired outputs of the model. The three reaction wheel angular accelerations will make up the input vector. Equation 3.62 gives the air bearing state vector and Equation 3.63 gives the air bearing input vector.

$$x_{ab} = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.62}$$

$$u_{ab} = \begin{bmatrix} \dot{\Omega}_x \\ \dot{\Omega}_y \\ \dot{\Omega}_z \end{bmatrix} \tag{3.63}$$

The state matrix that is derived from the above state vector and the three linearized equations of motion is given below in Equation 3.64. The matrix is sparse because the linearized equations of motion are not a function of the state. The only nonzero components in the state matrix equate the derivative of the three angular orientations to their respective angular rates, since these values are equal. Otherwise, the current state has no effect on the propagation of the state.

$$A_{ab} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.64}$$

The input matrix derived from the input vector and the linearized equations of motion is given below in Equation 3.65. The upper half of the input matrix is zero because the three angular orientations are not driven by the reaction wheel accelerations. The lower half of the input matrix is populated because the linearized equations of motion are functions of the reaction wheel angular accelerations.

$$B_{ab} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\frac{I_a D_{11}}{I_{xx}} & -\frac{I_a D_{12}}{I_{xx}} & -\frac{I_a D_{13}}{I_{xx}} \\ -\frac{I_a D_{21}}{I_{yy}} & -\frac{I_a D_{22}}{I_{yy}} & -\frac{I_a D_{23}}{I_{yy}} \\ -\frac{I_a D_{31}}{I_{zz}} & -\frac{I_a D_{32}}{I_{zz}} & -\frac{I_a D_{33}}{I_{zz}} \end{bmatrix} \tag{3.65}$$

All six states are desired outputs of the state space model so the output matrix is simply a six by six identity matrix given in Equation 3.66.

$$C_{ab} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.66}$$

Using the state, input, and output matrices, the complete continuous state space model of the air bearing can be constructed. Equation 3.67 gives the air bearing state equation and Equation 3.68 gives the air bearing output equation. Again, this model is based on the linearized equations of motion about an equilibrium point of zero for all states and inputs.

$$\begin{bmatrix} \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\frac{I_a D_{11}}{I_{xx}} & -\frac{I_a D_{12}}{I_{xx}} & -\frac{I_a D_{13}}{I_{xx}} \\ -\frac{I_a D_{21}}{I_{yy}} & -\frac{I_a D_{22}}{I_{yy}} & -\frac{I_a D_{23}}{I_{yy}} \\ -\frac{I_a D_{31}}{I_{zz}} & -\frac{I_a D_{32}}{I_{zz}} & -\frac{I_a D_{33}}{I_{zz}} \end{bmatrix} \begin{bmatrix} \dot{\Omega}_x \\ \dot{\Omega}_y \\ \dot{\Omega}_z \end{bmatrix} \tag{3.67}$$

$$\begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.68}$$

The above continuous state space model must now be transformed to a discrete state space model. This is done in the same fashion as the reaction wheel model using Equation 3.38 for the discrete state matrix. The continuous state matrix is clearly not invertible so the convolution equation is necessary to find the input matrix. This equation is listed in Equation 3.40. The discrete air bearing state space model can now be defined using the discrete state and input matrices. The discrete state equation is given below in Equation 3.69. The discrete output equation is the same as the continuous output equation given above. The discrete state and input matrices are not expanded due to the complexity introduced by the matrix exponential used to calculate them.

$$\begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_{k+1} = A_{d\,ab} \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_k + B_{d\,ab} \begin{bmatrix} \dot{\Omega}_x \\ \dot{\Omega}_y \\ \dot{\Omega}_z \end{bmatrix}_k \tag{3.69}$$

Figure 3-11 shows the above discrete state space model of the air bearing transformed into a Simulink block diagram. As with the reaction wheel model, this model runs at the discrete time step. The model is constructed slightly differently than the reaction wheel model seen in Figure 3-10 because this model is the propagation step in the Kalman Filter, which will be explained in the Extended Kalman Filter Block section below. The output matrix is also not used in this diagram because the matrix is identity and therefore not necessary.

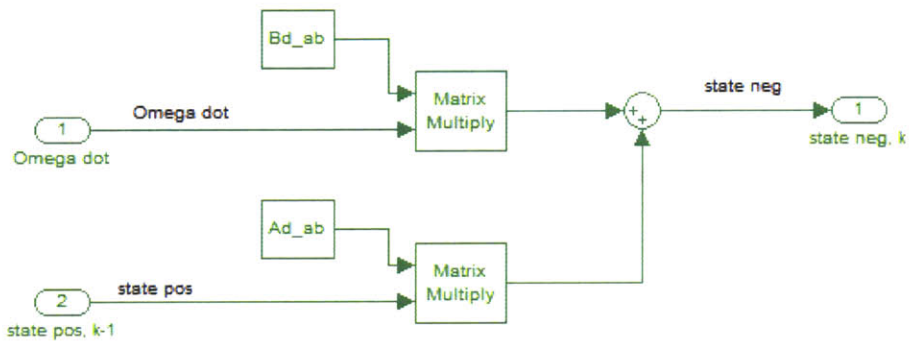


Figure 3-11: Air Bearing Discrete State Space Diagram

3.2.3.3 Measurement Blocks

The state space model of the system is one of two sets of inputs to the Kalman Filter. The other is the measurement vector, which is produced by creating simulated measurements of the true state as provided by the air bearing plant block. The measurement devices on the air bearing are an IMU which provides three axis rate measurements and three axis linear accelerometer measurements used to measure the gravity vector, and a magnetometer that provides three axis magnetic field measurements. The first measurement device to be covered is the magnetometer.

The first step to producing a simulated magnetometer measurement is knowing the magnetic field in the FIR frame. On the actual air bearing, this vector is measured by the magnetometer while it is co-aligned with the FIR frame at the beginning of each test. The measurement is then stored and used as the reference vector for the remainder of the test. In the simulation, the reference magnetic field vector is defined in the initial conditions section of the test_init.m file, which runs at the beginning of each simulation. The magnetic field

vector is measured in units of *counts* by the magnetometer. The *counts* can be converted to micro-Tesla using a conversion factor given in the magnetometer's data sheet.

The magnetic field vector in the FIR frame is pre-multiplied by the DCM from the FIR frame to the ABBF frame at each time step. The DCM from the FIR frame to the ABBF frame is calculated in the air bearing plant block as explained in Section 3.2.2. This multiplication produces the true magnetic field in the ABBF frame at the given time step. Random white noise with a mean of zero and an empirically determined variance is added to each component of the true magnetic field vector to produce the measured magnetic field vector at the given time step. The variance of the noise is found by testing the magnetometer, which will be covered in detail in Chapter 5. Figure 3-12 shows the process of calculating the simulated magnetic field vector measurement from the known magnetic field in the FIR frame.

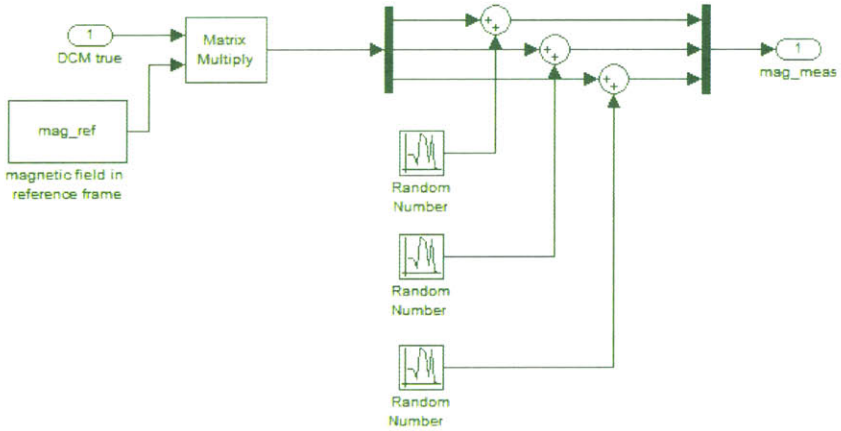


Figure 3-12: Magnetic Field Measurement Diagram

The simulated gravity vector measurement from the IMU's accelerometers is produced in the same way as the magnetic field vector measurement. The gravity vector in the FIR frame must be known at the beginning of the simulation. Due to the way the FIR frame is defined, the gravity vector is always in the negative FIR z axis. The gravity vector is measured in $g's$ by the accelerometers, which produces the reference gravity vector given in Equation 3.70.

$$grav_{ref} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{3.70}$$

The gravity vector in the FIR frame is pre-multiplied by the DCM from the FIR frame to the ABBF frame at each time step to produce the true gravity vector in the ABBF frame. Random white noise with a mean of zero and an empirically determined variance is added to each of the three components of the true magnetic field vector. The noisy vector is then quantized to values of 0.01 $g's$ to match the output of the actual accelerometers, which is limited to one one-hundredth of a g in each axis. Figure 3-13 shows the process of determining the simulated gravity vector measurement given the gravity vector in the FIR frame and the DCM from the FIR frame to the ABBF frame.

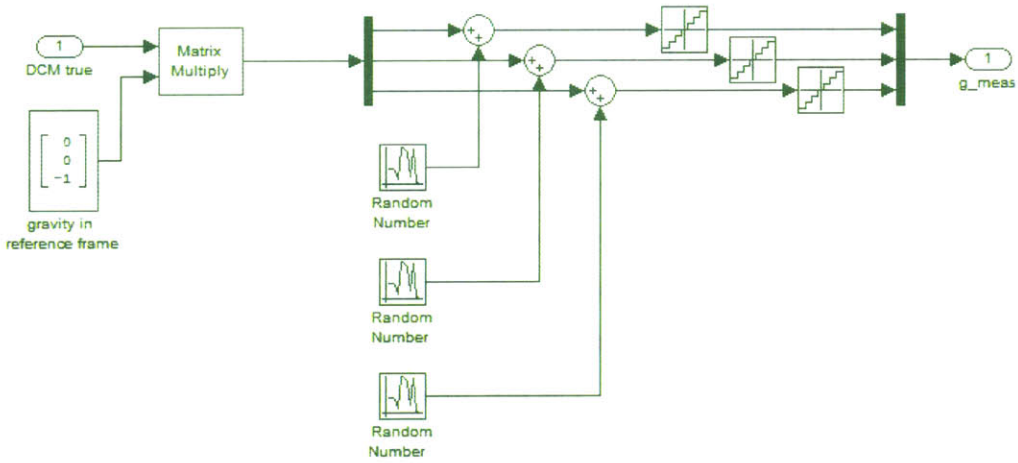


Figure 3-13: Gravity Vector Measurement Diagram

The most straightforward of the simulated measurements is that of the air bearing rate by the IMU's gyroscopes. Because the gyros are directly measuring part of the air bearing state, the only manipulation necessary to produce a simulated measurement is to add white noise to each component of the true rate vector provided by the air bearing plant block. The noise has a mean of zero and an empirically determined variance. The magnitude of the variance for the gyros is determined by testing the device and will be explained in Chapter 5. Finally, the rate measurement is discrete so the continuous true rate input from the air bearing plant must be discretized before noise can be added. Figure 3-14 shows how

the rate measurement is calculated in the Simulink block environment given the continuous true rate input.

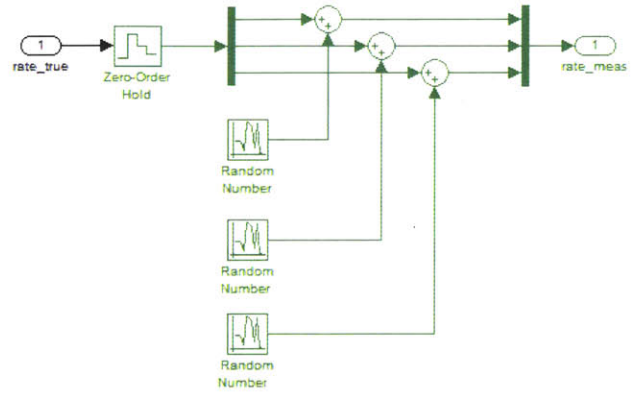


Figure 3-14: Air Bearing Angular Rate Measurement Diagram

3.2.3.4 Extended Kalman Filter Block

The Extended Kalman Filter can now be developed using the linearized model of the reaction wheels and air bearing in addition to the measurements from the IMU and magnetometer. The first step is to introduce the basics of a Kalman Filter. The goal of the Kalman Filter is to provide the best estimate of the system state by optimally combining the inputs from the system model with measurements of the system state. There are two major steps to the discrete Kalman Filter process. The first step is to propagate the system state from one time step to the next using the system model. The second is to update the propagated state using measurement inputs from the available sensors. In order for the Kalman Filter to provide an optimal state estimate, the filter must also propagate and update the error covariance matrix P, which contains information about the uncertainty of the system model and the individual sensor measurements. Figure 3-15 shows the basic two step process of the Kalman Filter over the course of three time steps. For consistent notation, a $\ominus$ sign will be used to denote propagated values that have not been updated with a measurement, and a $\oplus$ sign will be used to denote updated values that have not been propagated [34].

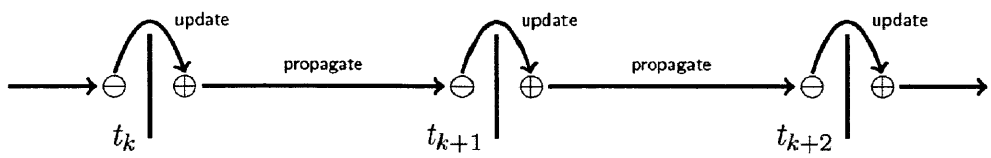


Figure 3-15: Discrete Kalman Filter Process [34]

The following will be a walk through of the two step Kalman Filter process for a general system. First, the system state will be propagated from time t_{k-1} to time t_k. Equation 3.71 gives the process for propagating the system state forward by one time step using the discrete state and input matrices. Note that this is essentially the same as the discrete state space equation previously derived.

$$\bar{x}_k^{\ominus} = A_d \bar{x}_{k-1}^{\oplus} + B_d \bar{u}_{k-1} \tag{3.71}$$

Equation 3.72 gives the process for propagating the error covariance matrix forward by one time step. This equation is a function of the discrete state matrix as well as the estimated noise covariance of the system model Q_{k-1}. The model noise covariance matrix is chosen to best describe the expected error induced by the system model. This matrix is often adjusted once the Kalman Filter is implemented to produce the best state estimate. The adjustment process is known as 'tuning' the Kalman Filter [34].

$$P_k^{\ominus} = A_d P_{k-1}^{\oplus} A_d^T + Q_{k-1} \tag{3.72}$$

The propagated system and error covariance matrix must now be updated using sensor inputs. Equation 3.73 gives the method to compute the Kalman gain, which is the optimal weighting factor used to mix the model input with the sensor inputs. The equation is a function of the error covariance matrix as well as the estimated sensor noise covariance R_k. The sensor noise covariance matrix is populated using the estimated noise introduced by each of the sensors. The sensor noise covariance matrix can also be adjusted similar to the model noise covariance matrix. Finally, the C_d matrix is not necessarily the same as the output matrix from the state space model, but is constructed to produce the correct estimated measurement vector from the state matrix [34].

$$L_k = P_k^{\ominus} C_d^T [C_d P_k^{\ominus} C_d^T + R_k]^{-1} \tag{3.73}$$

Equation 3.74 gives the method used to update the state estimate by introducing the sensor measurements. The vector y_k is the measurement vector, which includes all the sensor inputs. Equation 3.75 gives the process used to update the error covariance matrix based on the Kalman gain [34].

$$\bar{x}_k^{\oplus} = \bar{x}_k^{\ominus} + L_k(y_k - C_d \bar{x}_k^{\ominus}) \tag{3.74}$$

$$P_k^{\oplus} = (I - L_k C_d) P_k^{\ominus} \tag{3.75}$$

The above equations define the general two step Kalman Filter process to determine the optimal state estimate from the initial time to any future time. Though the error covariance matrix and Kalman gain are calculated at each time step, these values converge over time for a linear time-invariant system. The error covariance matrix and Kalman gain both converge to constant matrices. If the system state is to be estimated for an extended period of time, computation per control cycle can be reduced by using the steady state Kalman gain from the initial time step rather than waiting until the gain converges. Using the steady state Kalman gain from the start results in a less than optimal estimate of the state during the filter convergence period. However, once the filter converges, there is no difference between the estimate produced using the two methods.

The steady state error covariance matrix is found by solving the discrete Riccati Equation given below in Equation 3.76. The only unknown in the discrete Riccati equation is P_{ss}. Once solved, the steady state error covariance matrix can be used to calculate the steady state Kalman gain. However, the discrete Riccati equation is very difficult to explicitly solve. MATLAB offers a solution in the form of dlqr.m. This function solves the discrete Riccati equation and produces the steady state error covariance matrix as well as the steady state Kalman gain [34].

$$P_{ss} = A_d\Big(P_{ss} - P_{ss} C_d^T [C_d P_{ss} C_d^T + R]^{-1} C_d P_{ss}\Big) A_d^T + Q \tag{3.76}$$

Using the steady state Kalman gain eliminates the need to propagate and update the error covariance matrix. Therefore, the two step Kalman filter process is reduced to the following two equations. Equation 3.77 is used to propagate the state (unchanged from Equation 3.71) and Equation 3.78 is used to update the state with sensor inputs. Using the steady state Kalman gain significantly reduces the computation speed required to produce the optimal state estimate [34].

$$\bar{x}_k^{\ominus} = A_d \bar{x}_{k-1}^{\oplus} + B_d \bar{u}_{k-1} \tag{3.77}$$

$$\bar{x}_k^{\oplus} = \bar{x}_k^{\ominus} + L_{ss}(y_k - C_d \bar{x}_k^{\ominus}) \tag{3.78}$$

With a basic understanding of the steady state discrete Kalman Filter, the process will now be applied to the air bearing simulation. The filter used on the air bearing is known as an Extended Kalman Filter rather than a regular Kalman Filter simply because the equations of motion for the air bearing are linearized to create the state space model. Otherwise, the two estimators are the same. The first step of the EKF process; propagating the state using the state space model has already been explained above in the Air Bearing State Space Block section. Therefore, the remainder of this section will focus on the second step of the EKF process; updating the state estimate using sensor measurements.

The measurement vector y_k is based on what sensor measurements are available to update the state. For the air bearing, the measurement vector will contain twelve measurements. The first three values represent the angular orientation of the three ABBF axes measured by integrating the IMU rate gyroscopes. The next three values represent the ABBF angular rates measured directly by the IMU rate gyroscopes. Positions seven through nine of the measurement vector are populated by the angular orientation of the three ABBF axes as measured by the IMU accelerometers. The final three positions of the measurement vector are populated by the angular orientation of the three ABBF axes as measured by the magnetometer. Equation 3.79 gives the full twelve position measurement vector.

$$y_k = \begin{bmatrix} \theta_{x\,gyro} \\ \theta_{y\,gyro} \\ \theta_{z\,gyro} \\ \omega_{x\,gyro} \\ \omega_{y\,gyro} \\ \omega_{z\,gyro} \\ \theta_{x\,acc} \\ \theta_{y\,acc} \\ \theta_{z\,acc} \\ \theta_{x\,mag} \\ \theta_{y\,mag} \\ \theta_{z\,mag} \end{bmatrix}_k \tag{3.79}$$

The first six positions of the measurement vector can be filled by either integrating the rate gyro measurements (first three positions) or directly placing the rate gyro measurements in the vector (next three positions). The last six positions are not as easily filled. The accelerometers and magnetometer do not directly measure the air bearing's angular orientation but rather the gravity vector and magnetic field vector respectively. The first step towards producing a useful measurement from these two sensors is to calculate the DCM from the FIR frame to the commanded ABBF frame. This DCM is calculated just as the one described in the Air Bearing Plant Block section that relates the FIR frame to the true ABBF frame. The only difference is that this DCM is formed from the commanded angular rate to produce the rotation matrix from the FIR frame to the commanded ABBF frame.

The remainder of the process necessary to produce a measured angular orientation will be described for the linear accelerometers. The same process is used to produce a measured angular orientation from the magnetometers but will be omitted in this discussion. The gravity vector in the FIR frame described above in the Measurement Blocks section and given in Equation 3.70 is pre-multiplied by the DCM from the FIR frame to the commanded ABBF frame to produce the commanded gravity vector. The commanded gravity vector is defined as the vector that should be measured by the accelerometers if the air bearing is located in the exact angular orientation where it is commanded to be. Though the

commanded gravity vector and measured gravity vector are both known, a direct subtraction of the two does not produce useful attitude information. Equation 3.80 gives the process necessary to produce angular orientation information from the measured and commanded gravity vectors [46].

$$\bar{\theta}_{acc\,error} = \frac{g_{meas} \times g_{cmd}}{|g_{meas}||g_{cmd}|} \tag{3.80}$$

The cross product of the measured and commanded gravity vectors divided by their magnitudes produces a three vector of attitude errors in the ABBF frame. Each component of this vector is stating the angular error in radians between the commanded ABBF frame and the ABBF frame as measured by the accelerometers [46]. Adding the commanded angular orientation to the attitude error vector produced by Equation 3.80 gives the angular orientation in the ABBF frame as measured by the accelerometers. Equation 3.81 shows this process in equation form. The output of the below equation can now be placed within the measurement vector given in Equation 3.79.

$$\begin{bmatrix} \theta_{x\,acc} \\ \theta_{y\,acc} \\ \theta_{z\,acc} \end{bmatrix} = \begin{bmatrix} \theta_{x\,cmd} \\ \theta_{y\,cmd} \\ \theta_{z\,cmd} \end{bmatrix} + \frac{g_{meas} \times g_{cmd}}{|g_{meas}||g_{cmd}|} \tag{3.81}$$

The same process is used to provide the angular orientation of the air bearing as measured by the magnetometer. The only difference is that the measured magnetic field vector and the commanded magnetic field vector are used in place of the gravity vector as shown in Equation 3.82.

$$\begin{bmatrix} \theta_{x\,mag} \\ \theta_{y\,mag} \\ \theta_{z\,mag} \end{bmatrix} = \begin{bmatrix} \theta_{x\,cmd} \\ \theta_{y\,cmd} \\ \theta_{z\,cmd} \end{bmatrix} + \frac{B_{meas} \times B_{cmd}}{|B_{meas}||B_{cmd}|} \tag{3.82}$$

Figure 3-16 shows the block diagram interpretation of Equation 3.81, which uses the commanded and measured gravity vector to produce the air bearing angular orientation as measured by the IMU accelerometers. Again, a similar block diagram is used to evaluate the magnetic field measurement.

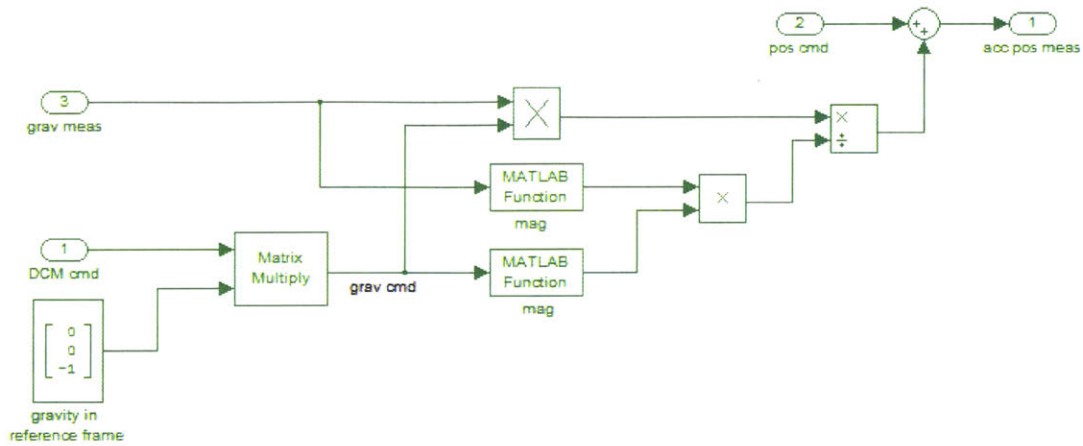


Figure 3-16: Angular Orientation Vector from Gravity Vector Measurement

The measurement vector given in Equation 3.79 has been completely populated and ready to be applied in the state update portion of the EKF process. However, the measurement vector hides several limitations inherent to the measurement process that should be explained. These limitations are the driving factor as to why the measurement vector includes three separate measurements of the air bearing angular orientation and only one of the air bearing angular rate.

Besides noise, measurement methods often have other limitations that introduce error to the measurement process. First, the IMU rate gyroscopes produce an angular orientation measurement by integrating the rate measurement. The rate gyroscope measurement is assumed to have zero mean, but in reality, the noise often has some nonzero mean known as bias. The bias usually has minimal effect when the measurement is used to simply estimate rate. However, if the rate measurement is integrated to produce an angular orientation measurement, the bias is integrated as well. This effect causes the angular orientation measurement to 'walk' in either the positive or negative direction depending on the sign of the bias even though the actual vehicle is not moving. This effect is difficult to correct in the state estimation process and therefore, integrating gyroscope measurements to produce an angular orientation measurement should be avoided if other sensors are available to provide an angular orientation measurement without integration.

The IMU accelerometers and magnetometer provide angular orientation measurements by finding the angular difference between a measured vector in three dimensional space and where that vector is supposed to be. Using the accelerometers as an example, the

error between the measured and commanded gravity vector is found. This process provides error information about two of the three orthogonal axes. However, the accelerometers cannot determine angular error about the true gravity vector because this information is not included in the cross product used to calculate angular error. For the air bearing, this means that the accelerometers cannot measure angular orientation error about the z_{ref} axis. The angular orientation measurement found from the accelerometers is still a three vector regardless of the unobservable rotation axis. However, the cross product of the measured and commanded gravity vector produces a zero error value about the unobservable axis. Therefore, the angular orientation measurement is assumed to be equal to the commanded angular orientation about the unobservable axis, and is corrected by the measurement only in the two observable axes.

The magnetometer faces the same limitation as the accelerometers because it uses essentially the same process as the accelerometers to find attitude error. The angular orientation about the true magnetic field vector is unobservable. However, by using both sensors in combination with each other, all three orthogonal axes are observable as long as the true gravity vector and true magnetic field vector are not co-aligned.

Equation 3.83 gives the EKF state update equation with the air bearing state and measurement vectors substituted in place of the general variables. Using this equation and a better understanding of the benefits and limitations of the air bearing sensors, the $C_{d\,ab}$ matrix can be formed to provide the best combination of measurements to apply to the state estimate.

Equation 3.84 gives the $C_{d\,ab}$ that is most commonly used in the simulation. The matrix must be a twelve by six matrix to operate within the state update equation. The below equation uses the angular rate measurement from the IMU gyroscopes and the angular orientation measurements from the IMU accelerometers and magnetometer. The matrix ignores the angular orientation measurement from the integrated IMU gyroscopes due to the bias error explained above. This matrix can be modified to meet the needs of a given test. For example, if the test requires that the accelerometers provide the only angular orientation measurement, the positions in the below matrix relating the the magnetometer measurements can be set to zero and visa versa.

$$\begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_k^{\oplus} = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_k^{\ominus} + L_{ss} \left(\begin{bmatrix} \theta_{x\,gyro} \\ \theta_{y\,gyro} \\ \theta_{z\,gyro} \\ \omega_{x\,gyro} \\ \omega_{y\,gyro} \\ \omega_{z\,gyro} \\ \theta_{x\,acc} \\ \theta_{y\,acc} \\ \theta_{z\,acc} \\ \theta_{x\,mag} \\ \theta_{y\,mag} \\ \theta_{z\,mag} \end{bmatrix}_k - C_{d\,ab} \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_k^{\ominus} \right) \tag{3.83}$$

$$C_{d\,ab} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.84}$$

The final unknown in the air bearing state update equation is the steady state Kalman gain L_{ss}. This is found using MATLAB's dlqe.m function to solve the discrete Riccati equation. Equation 3.85 gives the equation and inputs necessary to produce the steady state Kalman gain. The only unknown input is the six by six identity matrix. This is simply a scaling matrix between the model noise covariance and the sensor noise covariance and is set to identify because the scaling factor is included in the noise covariance matrices.

$$L_{ss} = dlqe(A_{d\,ab}, I, C_{d\,ab}, Q_{lqe}, R_{lqe}) \tag{3.85}$$

The state update step of the EKF is now ready to be applied to the simulation. Figure 3-17 shows the block diagram interpretation of the state update process. Beginning with the commanded angular rate and orientation as well as the sensor measurements and propagated state, the updated state is produced. The updated state represents the best estimate of the state at the current time step. This is the output of the EKF, which is provided as an input to the control module for comparison with the commanded state.

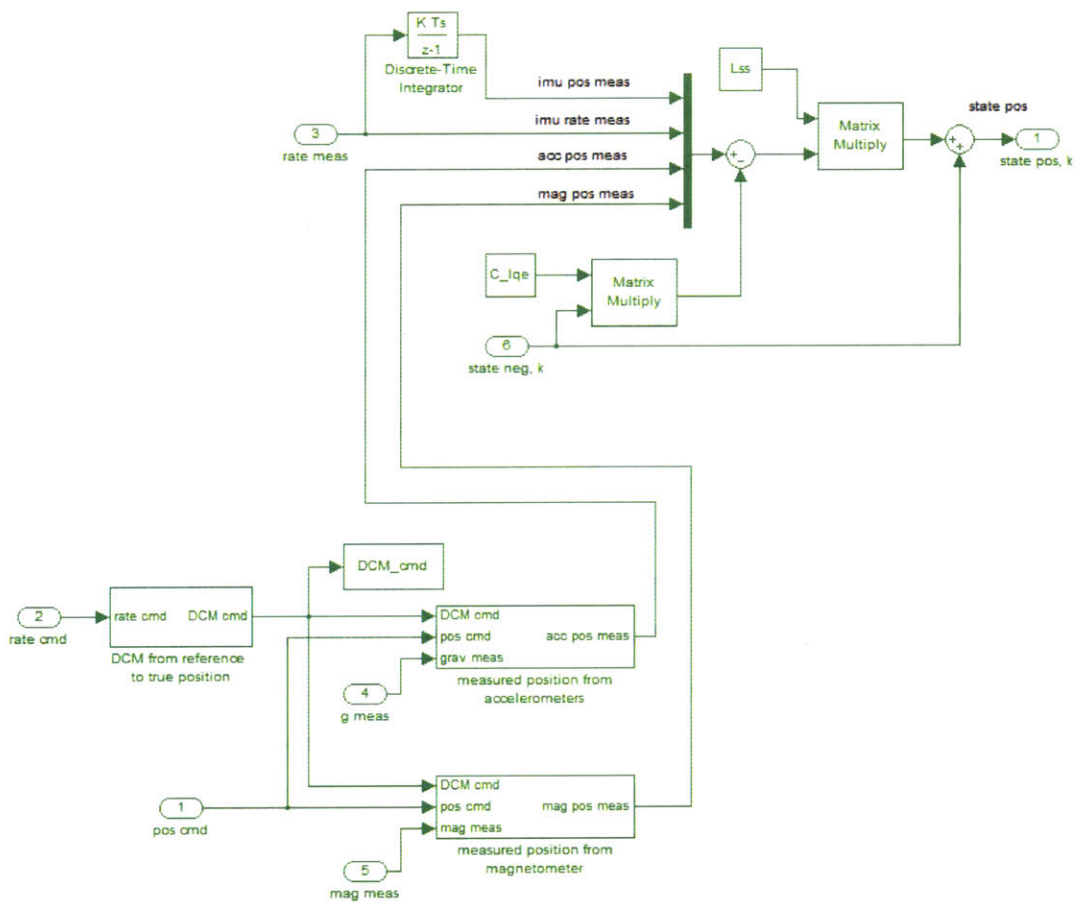


Figure 3-17: State Update Diagram within EKF

3.2.4 Command Module

The command module provides the three axis commanded angular orientation and rate to the air bearing control law. These commands are given in the ABBF frame. Providing the commands in the ABBF frame allows for the test designer to visualize the motion of the air bearing in response to the commands being provided since each commanded angular orientation or rate directly relates to commanded motion about one of the air bearing's principle axes. The DCM from the FIR frame to the commanded ABBF frame found in the EKF module can be used to relate the commanded angular orientation and rate to the FIR frame so that the commanded motion of the air bearing in the simulation can be compared with the motion of the actual air bearing as seen by a stationary observer.

A scalar command as a function of time must be provided for each ABBF axis. Three total commands are required to complete the three vector of angular orientation and rate. The commands can be given either in radians as the angular orientation command or in radians per second as the angular rate command. If the command is given in radians, the command is differentiated to produce the commanded angular rate in radians per second. If the command is given in radians per second, the command is integrated to produce the commanded angular orientation in radians.

In order to not saturate the control law with angular orientation or rate inputs that are significantly different than the estimated angular orientation and rate, commands should be continuous at a minimum and smooth if possible. For example, a step input should not be used as a command for either angular orientation or rate because at the instant the step is applied, the commanded angular value is significantly different than the estimated angular value. Furthermore, if the step input is applied as a commanded angular orientation, the commanded angular rate, which is the derivative of the commanded orientation will be an impulse. This is especially not desirable.

For commanded angular rate, ramp inputs are acceptable because even though they are not smooth, they are continuous, and their integral is smooth providing a smooth commanded angular orientation. Sinusoidal inputs are ideal because they are smooth and both their derivative and integral are smooth. Therefore, a sinusoidal input provided either as the commanded angular orientation or rate will guarantee smooth inputs for both commanded

values. In general, angular rate is commanded because ramp inputs can be used, which are then integrated to produce smooth commanded angular orientations.

Figure 3-18 shows an example of the command block. As seen in the figure, a scalar commanded rate is calculated for each ABBF axis using a series of ramp inputs. These commanded rates are concatenated to produce a three vector of commanded angular rates. The angular rates are integrated to produce a three vector of commanded angular orientations. These values are then output to the control module.

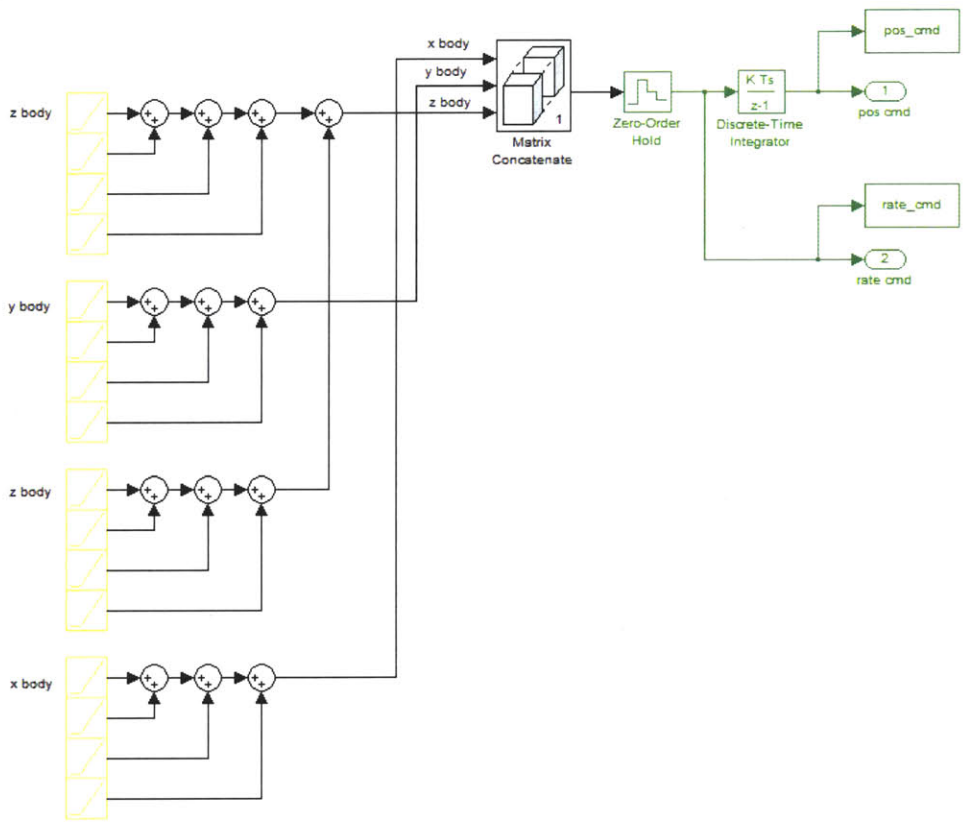


Figure 3-18: Commanded Angular Orientation and Rate Diagram

The ramp inputs can be set to start at any given time to produce the desired angular rate. By adding ramps with equal magnitude and opposite slope, the commanded rate can be stopped, reversed, returned to zero, etc. Figure 3-19 shows the three commanded rates produced by the series of ramps shown in Figure 3-18. The four ramps used for each ABBF axis produce the trapezoidal commanded rate seen in this figure. The z_{body} axis has two sets of four ramps which produce the two distinct angular rate commands seen in Figure 3-19.

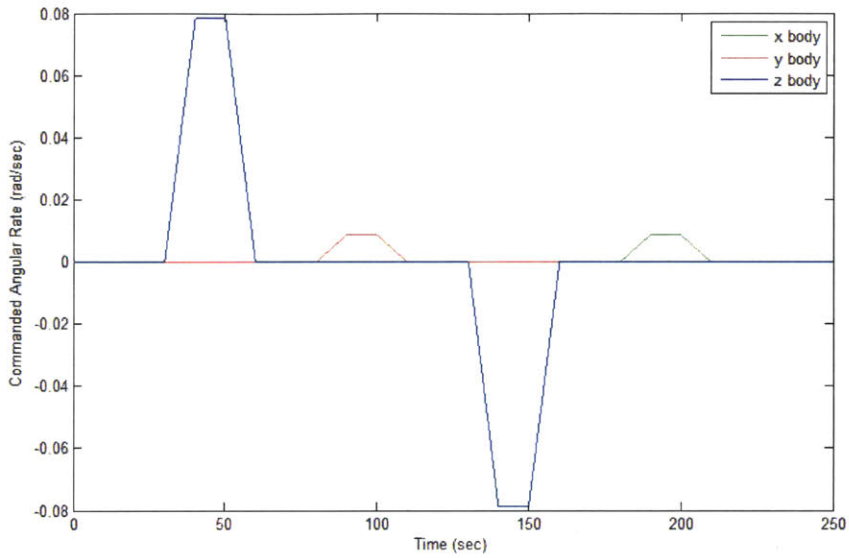


Figure 3-19: Commanded Angular Rate over Time

Integrating the commanded rate produces the commanded angular orientations seen in Figure 3-20. Unlike the commanded angular rates, the commanded angular orientations are smooth. This is because integrating the ramps in Figure 3-19 produce the parabolas connecting the linear portions of Figure 3-20. This series of commanded angular rates and orientations is a commonly used test scenario applied to validate the operation of the physical air bearing and simulation. These tests and results will be described in Chapter 5.

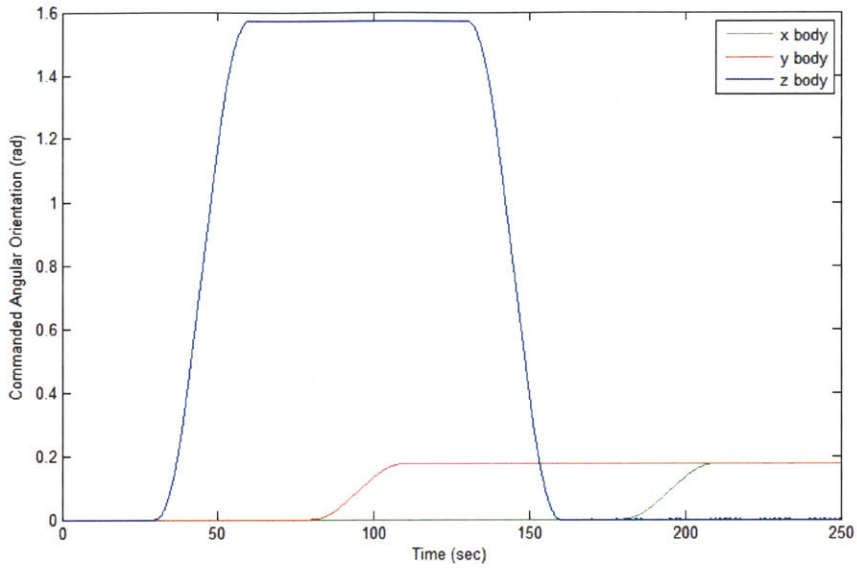


Figure 3-20: Commanded Angular Orientation over Time

The command module is the most flexible in the air bearing simulation. Commanded angular rates and orientations can be designed to meet the needs of almost any test scenario. The only limits to the commanded angular rates and orientations are those set by the physical limitations of the air bearing and reaction wheels.

3.2.5 Control Module

The control module uses several common control techniques in parallel to produce an effective control input to the system. Though the following section will cover the current set of control algorithms used on the air bearing, this module also lends itself to modification as easily as the command module. The control techniques applied can be adjusted throughout the test process to better meet the goals of the current test scenario.

The current control set includes a feedback portion consisting of a discrete linear quadratic regular (LQR) and a simple derivative gain for rate damping. The control set also includes a feedforward portion that uses the air bearing equations of motion to produce the reaction wheel angular rates required to achieve commanded maneuvers with or without stored angular momentum. The feedforward portion will be described first because this portion produces the primary set of commanded reaction wheel angular rates used to control the air bearing. The feedback portion accounts for error between the predicted torque from the

feedforward portion and the true torque required to produce the commanded air bearing angular orientations and rates.

The vector air bearing equation of motion that produces air bearing angular accelerations from reaction wheel angular accelerations and the current state of the system is given in Equation 3.25 in the Air Bearing Plant Block section. Knowing that this equation produces a given air bearing angular acceleration from a given set of reaction wheel angular accelerations and the current reaction wheel and air bearing angular rates, the equation can be manipulated to produce the commanded reaction wheel angular accelerations based on a set of commanded air bearing angular rates. Equation 3.86 gives the manipulated equation as a function of the commanded air bearing angular rate and its derivative as well as the commanded reaction wheel angular rate, which is found by integrative the output of this equation.

$$\dot{\bar{\Omega}}_{cmd} = I_{RW}^{-1} D^T \Big[- I_{AB}\dot{\bar{\omega}}_{cmd} - (\bar{\omega}_{cmd} \times I_{AB}\bar{\omega}_{cmd}) - (\bar{\omega}_{cmd} \times D I_{RW}\bar{\Omega}_{cmd}) \Big] \tag{3.86}$$

Figure 3-21 shows the above equation in the Simulink block diagram format. Some test cases require the angular velocity of the wheels to be set at a nonzero value as an initial condition. These nonzero reaction wheel angular velocities are placed into the simulation as the initial condition to the integrator within the feedforward block that produces commanded reaction wheel angular rate from commanded reaction wheel angular acceleration. By placing the reaction wheel initial conditions within this block, the feedforward equation above can account for the additional gyroscopic effects resulting from the nonzero reaction wheel angular rates.

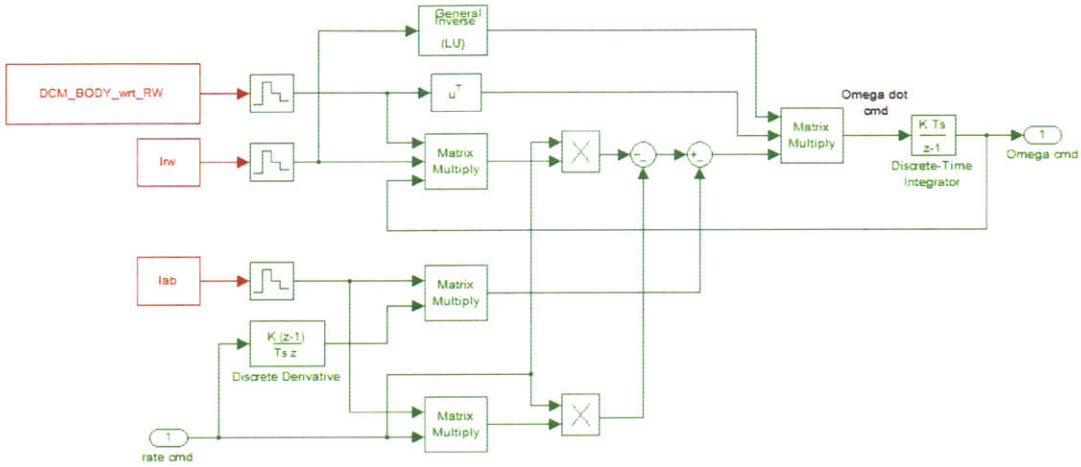


Figure 3-21: Feedforward Control Diagram

To show the effects of nonzero angular reaction wheel rates on the air bearing system, two commanded reaction wheel angular rate profiles will be calculated using the same commanded air bearing angular rate profile; one with zero initial reaction wheel angular rate and the second with nonzero initial reaction wheel angular rate. The two profiles will be compared in the following two figures. The commanded air bearing angular rate input to the feedforward equation will be the same as those given above in Figure 3-19. Figure 3-22 shows the commanded reaction wheel rates required to achieve the commanded air bearing angular rate given in Figure 3-19 assuming zero initial reaction wheel velocity. The wheels return to zero angular velocity after completing each maneuver.

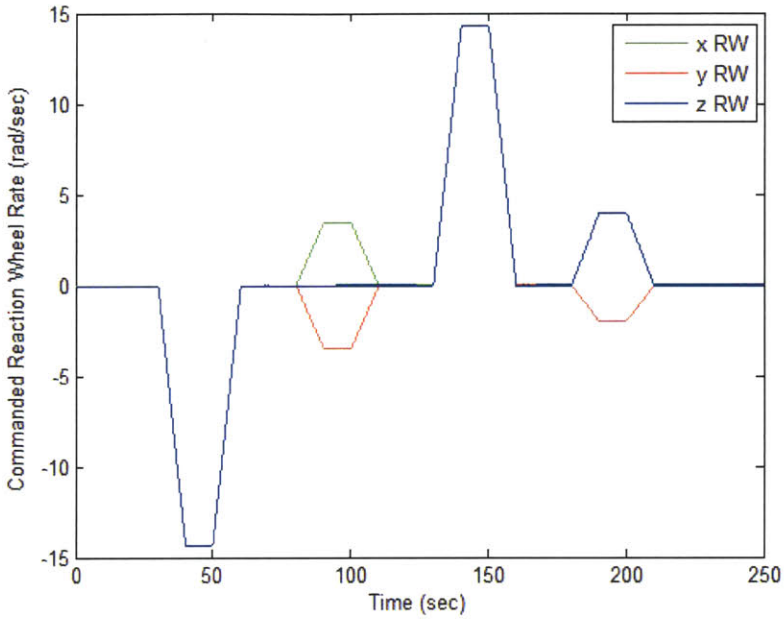


Figure 3-22: Commanded Reaction Wheel Rates - Zero IC

Figure 3-23 shows the commanded reaction wheel rates required to achieve the same commanded air bearing rate assuming that the reaction wheels have the initial conditions given below in Equation 3.87. The reaction wheels do not return to their initial conditions after completing each maneuver as seen in Figure 3-22. This is because the nonzero initial reaction wheel rates produce gyroscopic torques that must be accounted for by adjusting the commanded reaction wheel rates themselves. Therefore, even though the commanded air bearing angular rates are the same for both cases, the feedforward controller produces the commanded reaction wheel angular rates required to perform the commanded air bearing rates regardless of the initial or current state of the system.

$$\bar{\Omega}_0 = \begin{bmatrix} 40 \\ 40 \\ -80 \end{bmatrix} \frac{rad}{sec} \tag{3.87}$$

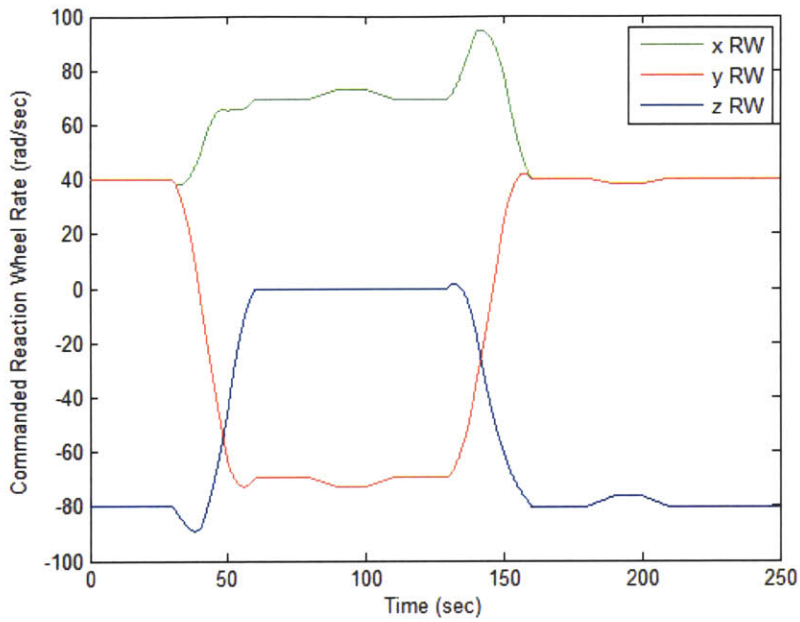


Figure 3-23: Commanded Reaction Wheel Rates - Nonzero IC

The feedback portion of the control law uses a linear quadratic regulator and constant derivative gain to account for errors between the commanded state and the estimated state from the EKF. The state error should be minimal due to the feedforward control portion. Errors are caused by inaccuracies in the system model used to produce the vector equation of motion as well as disturbance torques that are unknown to the model. Errors are also caused by not modeling reaction wheel lag in the feedforward equation.

First, the LQR controller will be described. An LQR controller uses the state space model to find a feedback gain matrix that provides the optimal full state feedback gains for a given set of state and input weighting matrices. These weighting matrices are constructed by ranking the importance of maintaining minimum error for each individual state and input. As will be seen in the following discussion, the process is very similar to the method for producing the Kalman gain by using weighting matrices and solving the discrete Riccati Equation [33].

The foundation of an LQR controller is the cost function, which is a function of the system state and inputs. Equation 3.88 gives the cost function used to develop an LQR controller. The state weighting matrix Q is used to set the importance of controlling each state. If one of the states is more important to control than the others, a large value can be

placed in the corresponding position within the state weighting matrix. This will place a larger cost on that particular state and drive the cost function to minimize that state more so than the others. The same holds for the input weighting matrix R. If control inputs should be kept to a minimum, high values can be set within the matrix R. Similar to the Kalman gain development process, these matrices can be tuned to provide the ideal control as desired by the control designer [33].

$$J = \int_0^\infty \left[\bar{x}^T C^T Q C \bar{x} + \bar{u}^T R \bar{u}\right] dt \tag{3.88}$$

To minimize the cost function, the derivative of the function is set to zero. The resulting equation given in Equation 3.89 is the algebraic Riccati Equation for control. This equation is very similar to the Riccati Equation used for finding the Kalman gain in the EKF [33].

$$0 = A^T P + PA + C^T Q C - PBR^{-1}B^T P \tag{3.89}$$

By substituting the system state space model and weighting matrices into the above equation, the P matrix can be found. Substituting P into Equation 3.90 gives the optimal feedback gain for the system. Multiplying this gain by the estimated state produces the optimal closed loop input to the system [33].

$$K_{lqr} = R^{-1}B^T P \tag{3.90}$$

The linearized state space model of the air bearing is developed for use in the EKF. Therefore the model is ready for use in finding the optimal LQR gain. The weighting matrices are the only missing components. Though the values within the weighting matrices are often tuned to fit the specific test scenario, the weight placed on the air bearing state is usually significantly higher than the input since the reaction wheels are not considered an expendable resource. They can be operated as much as necessary over the course of a given test scenario. If other actuators like thrusters were placed on the air bearing, the input weighting matrix may require adjustment.

Similar to the estimator Riccati Equation, MATLAB is used to solve the controller Riccati Equation and produce the LQR gain matrix. Equation 3.91 gives the command required to use MATLAB's discrete LQR function.

$$K_{lqr} = dlqr(A_{d\,ab}, B_{d\,ab}, Q_{lqr}, R_{lqr}) \tag{3.91}$$

LQR controllers are often used as the only control algorithm for a system. Because the air bearing uses the LQR controller in combination with a feedforward controller that produces the primary control inputs, the LQR controller is only required to account for state error. Therefore, the feedback gain is implemented differently than the typical method. Equation 3.92 gives the method for applying the feedback gain. The gain is multiplied by the state error rather than the state itself. This produces input commands relating to the state error regardless of the commanded state. These inputs due to error are eventually added to the inputs from the feedforward portion to produce the total commanded reaction wheel angular rate from the controller suite.

$$\dot{\bar{\Omega}}_{cmd\,lqr} = K_{lqr} * [\bar{x}_{cmd} - \bar{x}_{est}] \tag{3.92}$$

The last stage of the air bearing controller is the derivative gain. This portion finds the estimated air bearing angular acceleration by differentiating the estimated air bearing angular rate. The commanded acceleration is set to zero as a means to dampen air bearing rate commands and reduce oscillation. The acceleration error (the negative of the estimated acceleration) is multiplied by an adjustable gain to produce a commanded air bearing acceleration. This acceleration must be transformed from a commanded air bearing acceleration to a commanded reaction wheel acceleration using the linearized equations of motion. Equation 3.93 gives the process for determining the commanded reaction wheel acceleration from the derivative feedback portion [33].

$$\dot{\bar{\Omega}}_{cmd\,deriv} = -I_{RW}^{-1} D^T I_{AB} [Kd * (-\dot{\bar{\omega}}_{est})] \tag{3.93}$$

The commanded reaction wheel angular accelerations from the LQR controller and derivative controller are summed to produce the total commanded reaction wheel acceleration due to state error as seen in Figure 3-24. This is integrated and added to the commanded reaction wheel rate calculated by the feedforward portion of the controller to produce the total commanded reaction wheel angular rate from the series of air bearing control algorithms. The inputs are fed into the reaction wheel motor controller block.

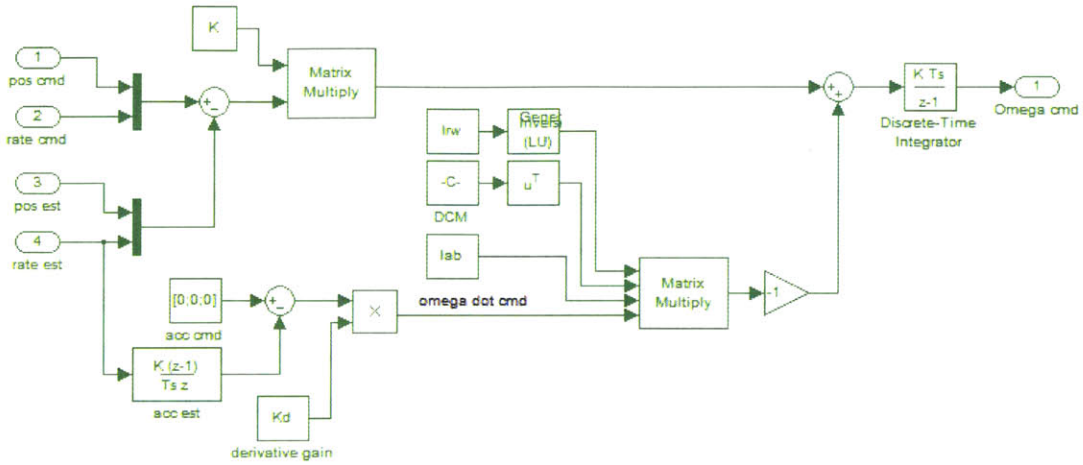


Figure 3-24: Feedback Control Diagram

The final feedback control used on the air bearing is the reaction wheel angular rate feedback into the motor controller algorithm. This closed loop operates within the larger air bearing control loop. The control loop is necessary due to the operation of the motor controllers used on the air bearing. The motor controllers command voltage to the reaction wheels. This allows the wheels to accelerate to a higher angular rate as predicted by their equations of motion. However, if the wheels are commanded to a lower angular rate in the same direction, the voltage change does not drive the motors to the new commanded speed. Rather, the motors coast until friction reduces their angular rate to the commanded value. In order to mitigate this problem, the fed back angular rate from the motor encoders is used to determine if the reaction wheels are spinning faster than they are commanded to spin. If their rate is five percent higher than the commanded value, the wheels are commanded to brake until they reduce their speed to within percent of the initially commanded speed. Braking the wheels means that the motor controller connects the positive and negative leads from the motor together. This eliminates the path for back EMF to travel and creates a force against the motion of the motor, causing it to reduce speed with a time constant similar to positive acceleration. The reaction wheel inner control loop forces the reaction wheels to behave similar to a motor control system using current commands that could evenly accelerate the wheels to higher or lower angular velocities. This is important because the remaining air bearing control process assumes the wheels can be commanded equally in both directions. Though this process provides a solution to the reaction wheel control problem,

there are disturbances introduced that are otherwise not accounted for, which adds error to the system.

3.2.6 Simulation Assumptions and Limitations

Like any simulation of a physical system, the testbed simulation described in this chapter has limitations due to assumptions made to create the simulation. These limitations affect the accuracy of the model, but as long as they are understood and accounted for, the results of the simulation can be quite beneficial. The assumptions made within the simulation can be grouped into two main areas; those affecting the results of the air bearing plant and those affecting the estimator algorithm.

The most sweeping assumption in the plant module is that the simulated true state of the reaction wheels and air bearing are calculated using the same equations of motion used to develop the state space model of the system. This artificially improves the accuracy of the state space model because this process assumes that the known equations of motion perfectly define the state of the system. In reality this is not the case. Though the general structure of the equations of motion for both the reaction wheels and air bearing are correct, the coefficients used within the equations have some unknown error.

The reaction wheel equations of motion use coefficients from the manufacturer's specification sheet. These same coefficients are used for all three orthogonal reaction wheels. The actual reaction wheel motors likely have values that are slightly different than these specifications and therefore do not react to the same input in the exact same way. For the air bearing, the defining coefficient within the equations of motion is the inertia of the vehicle. This is determined by the SolidWorks model of the air bearing. The SolidWorks model excludes some of the electronic devices on the air bearing and likely does not model every piece of hardware to exact specifications. Therefore, the true inertia of the vehicle is different than that found using the SolidWorks model. Overall, the difference in the simulated and true dynamics coefficients for the reaction wheels and air bearing means that the system model more accurately defines the motion of the simulated system than the true system. There are methods to more precisely determine the dynamics coefficients for the testbed, and though they have yet to be implemented, they can be used to more accurately match the simulated model with the true system.

An additional assumption causing error between the simulated plant module and the actual system is the assumption that there are no disturbance torques on the system. In simulation, there are no disturbance torques affecting the motion of the air bearing. However, in reality there are always disturbance torques on the system. The largest disturbance torque is due to the misalignment of the adjustable center of mass with the fixed center of rotation. These two points must be co-located in order to remove torque on the system due to gravity. Even the smallest misalignment causes gravity to torque the vehicle, which causes motion that is not modeled in the simulation. Other sources of disturbance torques include friction, which is significantly reduced by the air cushion that floats the vehicle but not eliminated, and atmospheric torques due to air moving around the testbed. These additional torques require additional control input from the reaction wheels that are not accounted for in the simulation. Though the control algorithm can usually handle these torques, they may cause attitude error to exceed the requirements of a given test or cause the reaction wheels to saturation much sooner than expected.

The estimator and controller algorithms within the simulation are also based on several assumptions that limit the precision between the actual and simulated systems. The EKF provides the optimal estimate based on the assumption that the sensors have zero mean noise. The actual IMU rate gyroscopes are known to not have zero mean noise. Tests can be run to characterize the noise and attempt to remove bias, though the bias itself does not seem to be constant. The simulation assumes that noise from all sensors has zero mean. Therefore, the simulated estimator is more accurate than the actual estimator on the air bearing.

Overall, the assumptions made to develop the testbed simulation produce inconsistencies between the simulated system and the actual system, but the benefits of having an operational system simulation outweigh the limitations of the system due to the assumptions. Furthermore, with additional testing, error induced by the above assumptions can be reduced, and as long as the assumptions are accounted for, the results of the simulation are a powerful tool in developing and evaluating test scenarios on the ADCS testbed.

Chapter 4

ADCS Testbed Development

The ADCS testbed is designed to provide a versatile platform for testing small satellite attitude determination and control systems from the component level to a fully integrated system as well as provide a hands on educational tool for students learning attitude estimation and control concepts. This chapter will discuss the detailed development of each of the physical testbed subsystems that come together to produce a robust, functional test platform capable of effectively evaluating complex control systems while also functioning as an easy to use educational tool.

Software development will also be discussed in order to provide a clear understanding of the testbed's avionics and communication protocol. Familiarization with the testbed's software operation will allow the test designer to effectively write new software algorithms that operate within the testbed's capabilities and are written to meet the needs of the required test scenario.

Finally, supporting equipment like the SPHERES satellites and the external magnetic field generator will be discussed. These devices provide expansion capability to the ADCS testbed. A SPHERES satellite can be mounted to the top of the air bearing to provide an external attitude estimate that can be sent directly to the ground station computer as additional attitude data for post processing. With some additional coding, the SPHERES can even send real time attitude information to the testbed avionics system and receive control commands from the avionics system using the SPHERES expansion port. In this configuration, the SPHERES' sensor measurements can be included in the EKF running

on the air bearing, and the air bearing control law can incorporate the SPHERES' cold gas thrusters for additional attitude control.

The external magnetic field generator can provide a well known magnetic field vector for attitude determination via on-board magnetometers as well as produce a magnetic field strong enough to perform magnetic attitude control tests within a laboratory environment where disturbance torques would prohibit magnetic torque maneuvers using Earth's magnetic field.

4.1 Testbed Subsystems

Though the air bearing is designed to be an ADCS testbed, it must incorporate all the major subsystems of a general satellite in order to support ADCS testing. Testing an integrated set of attitude sensors and actuators connected via estimation and control software requires support from other subsystems like power, avionics, communications, and of course structures. Because the system is an ADCS testbed, the supporting subsystems are not designed to test their respective counterparts on any specific satellite program. Rather, they are designed to best support the expected battery of ADCS test scenarios that can be performed on the air bearing testbed.

4.1.1 Structure

The primary objective of the air bearing structure is to provide a physical platform capable of interfacing with the hemispherical base as well as provide attachment points and stability for the components necessary to perform ADCS testing. The foundation of the structure is the hemisphere that floats on the cushion of compressed air. The hemisphere is the only portion of the vehicle that is in contact with the fixed inertial air bearing support column. However, the hemisphere is not actually touching the support column, but rather floating slightly above the column supported completely by a layer of compressed air. This support method is what gives the air bearing the ability to rotate in all three axes with significantly reduced friction while also being constrained to zero translational motion.

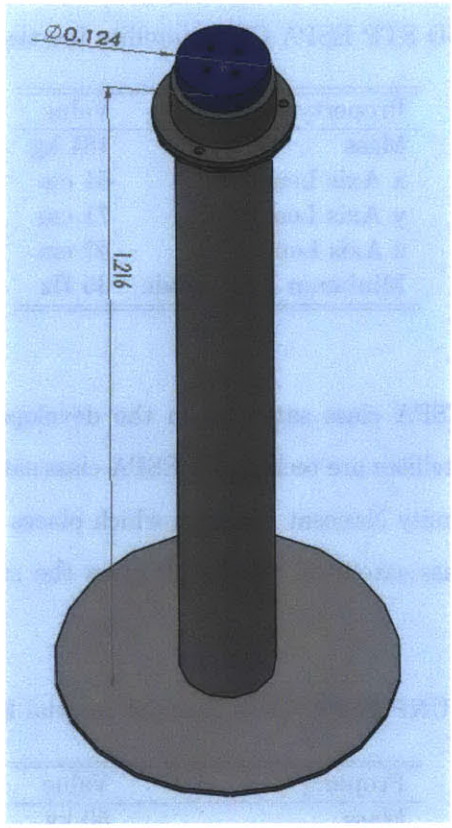


Figure 4-1: Air Bearing Support Column with Rotating Hemisphere

Figure 4-1 shows a SolidWorks rendering of the hemisphere positioned within the fixed inertial bowl on the support column. Compressed air is pumped between the gray bowl and the purple hemisphere to float the hemisphere and anything attached to it using the four threaded bolt holes on the top portion of the hemisphere. The hemisphere has a diameter of 0.124 meters and is positioned 1.216 meters above the base of the support column.

The structure attached to the rotating hemisphere is responsible for securing the components necessary to perform ADCS testing while ensuring safe operation of the air bearing. The structure is designed to roughly emulate the inertial properties of an ESPA class satellite, which is a common small satellite design used within the university environment. Smaller satellites like cubesats are also common in a university environment. Though the air bearing structure is larger than cubesat structures, cubesat ADCS components can still be mounted on the air bearing, and attitude control devices can be scale tested. The mass and volume requirements of an ESPA class satellite as defined by the Department of Defense (DoD) Space Test Program are listed in Table 4.1 [5].

Table 4.1: DoD STP ESPA Class Satellite Inertial Requirements

Property	Value
Mass	181 kg
x Axis Length	61 cm
y Axis Length	71 cm
z Axis Length	97 cm
Minimum First Mode	35 Hz

MIT's SSL has two ESPA class satellites in the development phase; CASTOR and TERSat. Though these satellites are technically ESPA class satellites, they are being developed under AFRL's University Nanosat Program which places additional mass and volume requirements on ESPA class satellites. Table 4.2 gives the more stringent UNP requirements [26].

Table 4.2: UNP ESPA Class Satellite Inertial Requirements

Property	Value
Mass	50 kg
x Axis Length	50 cm
y Axis Length	50 cm
z Axis Length	60 cm
Minimum First Mode	100 Hz

A rectangular satellite with the dimensions and uniformly distributed mass listed in Table 4.2 would have the principle axis inertia values listed in Equation 4.1 according to SolidWorks. In order to provide a comparable test environment with respect to the below inertial properties, the air bearing testbed should maintain inertial properties within an order of magnitude of these values and strive to be as close as possible.

$$I_{UNP\,Req} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} 2.54 & 0 & 0 \\ 0 & 2.54 & 0 \\ 0 & 0 & 2.08 \end{bmatrix} kgm^2 \tag{4.1}$$

The default configuration of the air bearing testbed has the mass given in Equation 4.2 and the ABBF aligned inertia matrix given in Equation 4.3. These values are taken from the SolidWorks model of the air bearing structure shown in Figure 4-2.

$$m_{ab} = 40.5kg \tag{4.2}$$

$$I_{ab} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} 4.08 & 0 & 0 \\ 0 & 4.14 & 0 \\ 0 & 0 & 2.32 \end{bmatrix} kgm^2 \tag{4.3}$$

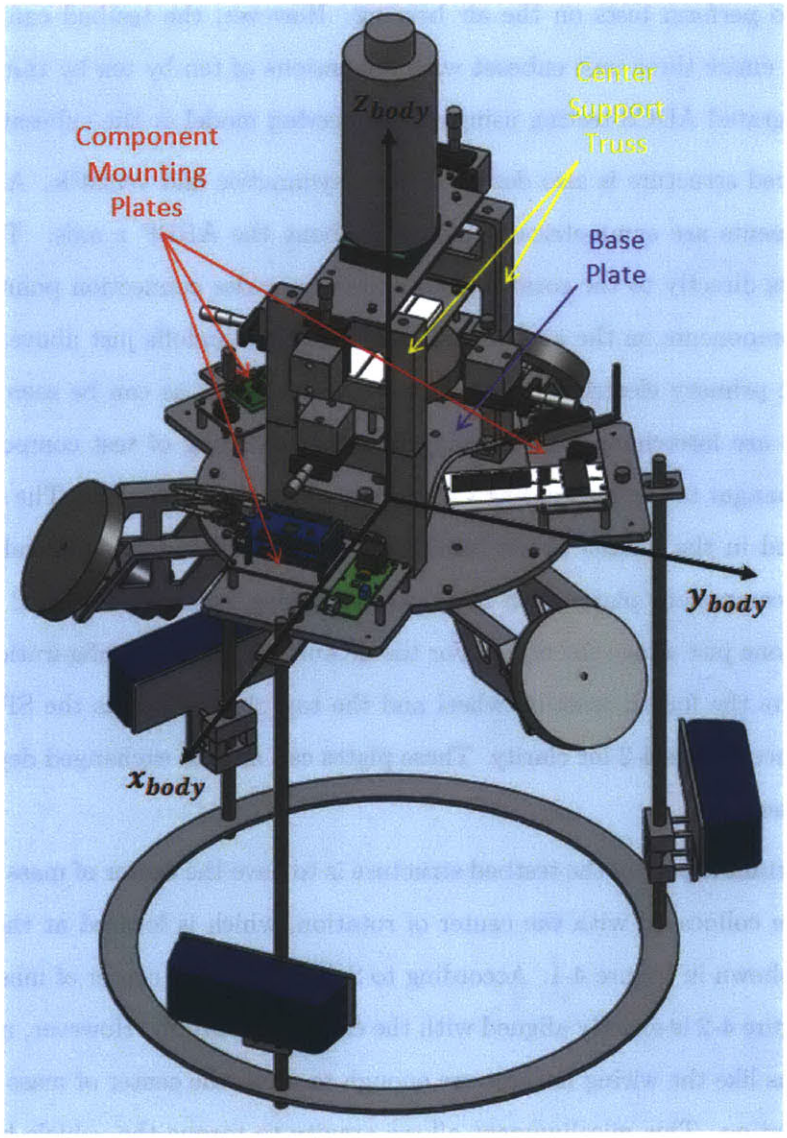


Figure 4-2: Air Bearing SolidWorks Model

The SolidWorks model of the air bearing does not show the support column, but just the physical components that are fixed to the rotating hemisphere. The electrical components

of the system are included in the model, but much of the wiring harness connecting the components is not included. This omission is a source of error in estimating the mass and inertia matrix of the air bearing, but this error is small since the mass of the wiring harness is much less than the mass of the modeled portion. The mass and inertia values given above are similar to those of a general rectangular UNP ESPA class satellite allowing for ADCS test scenarios to be performed with minimal scaling necessary between the satellite ADCS system and the testbed when testing ESPA class satellites. For cubesats, scaling is required to perform tests on the air bearing. However, the testbed can accommodate mounting an entire three unit cubesat with dimensions of ten by ten by thirty centimeters for fully integrated ADCS testing using an engineering model or the cubesat flight model.

The testbed structure is also designed to be symmetric and versatile. All major structural components are symmetrically mounted about the ABBF z axis. The base plate, which mounts directly to the rotating hemisphere provides connection points for all other structural components on the air bearing. Mounted on standoffs just above the base plate are the three primary electrical component mounting plates as can be seen in Figure 4-2. These plates are interchangeable allowing for easy switching of test components without significant changes to the air bearing's structure or inertial properties. The center support truss mounted in the middle of the base plate is made up of four identical A frame supports. Two component plates with identical interfacing holes are mounted between these four frames; one just above the other. For the default air bearing configuration, the bottom plate supports the fourth reaction wheel and the top plate supports the SPHERES interface. Reference Figure 4-2 for clarity. These plates can also be exchanged depending on the test requirements.

An important aspect of the testbed structure is to have the center of mass of the rotating portion to be collocated with the center of rotation, which is located at the center of the hemisphere shown in Figure 4-1. According to SolidWorks, the center of mass of the model shown in Figure 4-2 is exactly aligned with the center of rotation. However, modeling errors and omissions like the wiring harness are enough to move the center of mass away from the center of rotation. This misalignment allows gravity to torque the vehicle because gravity acts on the center of mass, and torque is created when a moment arm exists between the center of rotation and the point where gravitational force acts on the vehicle. In order to remove this torque even with inaccurate center of mass estimates, the structure has CM

adjustment fixtures. Large CM adjustments can be made (primarily in the ABBF z axis) by moving the three battery packs up or down their threaded support rods. The battery packs are shown in blue in Figure 4-3. The battery packs can be moved up or down to accommodate large component changes on the center support truss like adding or removing a SPHERES satellite. For smaller CM adjustments due to smaller component changes, trim masses can be added to the base plate at the six trim mass attachment points identified in Figure 4-3.

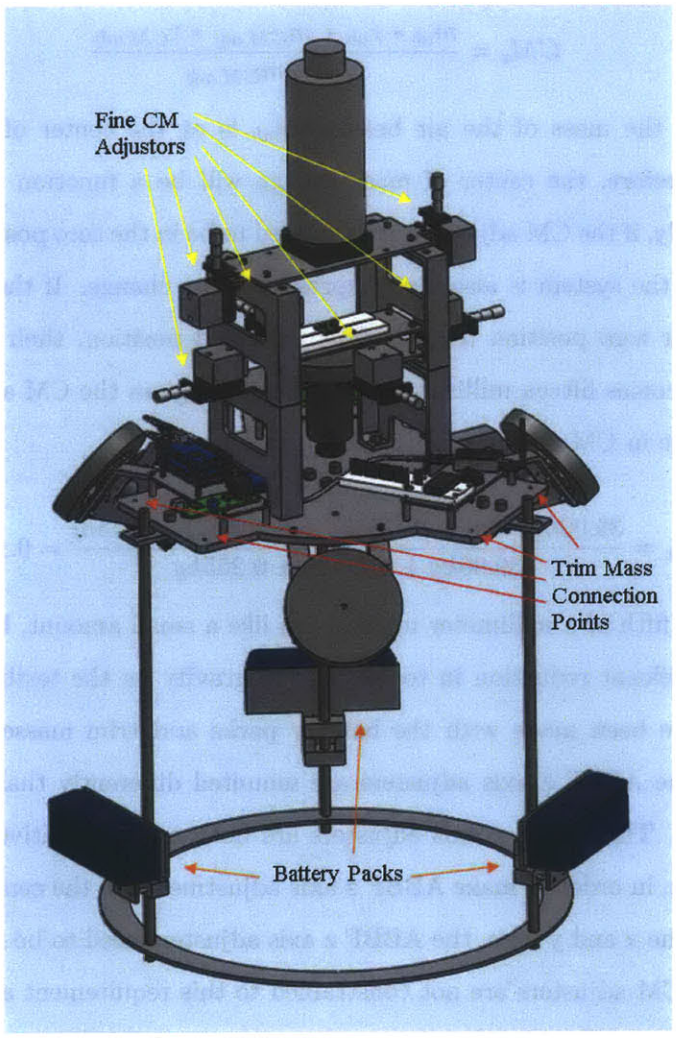


Figure 4-3: Air Bearing Center of Mass Adjusters

For fine CM adjustments necessary to reduce CM/center of rotation alignment error to an acceptable amount after making course adjustments with the battery packs and trim masses, the six CM adjusters shown in Figure 4-3 can be used. Twisting the handle of the

CM adjusters slides the mass linearly along the axis aligned with the handles. Each CM adjuster has a fifteen millimeter range of motion and the sliding portion has a mass of 0.255 kilograms. There are two CM adjusters located on the ABBF x axis. One on the positive side and one on the negative side. There are two more similarly located on the ABBF y axis. The total CM adjustment capability in the ABBF x axis due to the two CM adjusters can be found using Equation 4.4 [54]. This same equation can be used for the ABBF y and z axes.

$$CM_x = \frac{m_{ab} * x_{ab} + m_{CM\ adj} * x_{CM\ adj}}{m_{ab} + m_{CM\ adj}} \tag{4.4}$$

Assume that the mass of the air bearing m_{ab} is at the center of the ABBF x axis ($x_{ab} = 0$). Therefore, the center of mass change will be a function of moving the CM adjusters. Initially, if the CM adjusters are assumed to be in the zero position ($x_{CM\ adj} = 0$), the total CM of the system is also zero, representing no change. If the CM adjusters are moved from their zero position to their fully extended position, their position along the ABBF x axis becomes fifteen millimeters. Equation 4.5 gives the CM adjustment distance due to the change in CM adjuster position.

$$CM_x = \frac{39.99kg * 0m + (0.255kg + 0.255kg) * 0.015m}{39.99kg + 0.255kg + 0.255kg} = 0.189mm \tag{4.5}$$

Roughly one fifth of a millimeter might seem like a small amount, but this adjustment can make a significant reduction in torque due to gravity on the testbed once rough CM adjustments have been made with the battery packs and trim masses. As can be seen in Figure 4-3, the ABBF z axis adjusters are mounted differently than the ABBF x and y axis adjusters. The ABBF z axis adjusters are both on the positive size of the ABBF z axis. Therefore, in order to make ABBF z axis adjustments to the center of mass that are independent of the x and y axes, the ABBF z axis adjusters need to be moved equally. The ABBF x and y CM adjusters are not constrained to this requirement and can be adjusted independently as necessary.

The final structural component is the safety ring mounted at the base of the battery support rods. The ring is designed to prevent damage to the air bearing by contacting the inertial support column before the base plate touches the edge of the hemisphere support bowl. This scenario is shown in Figure 4-4.

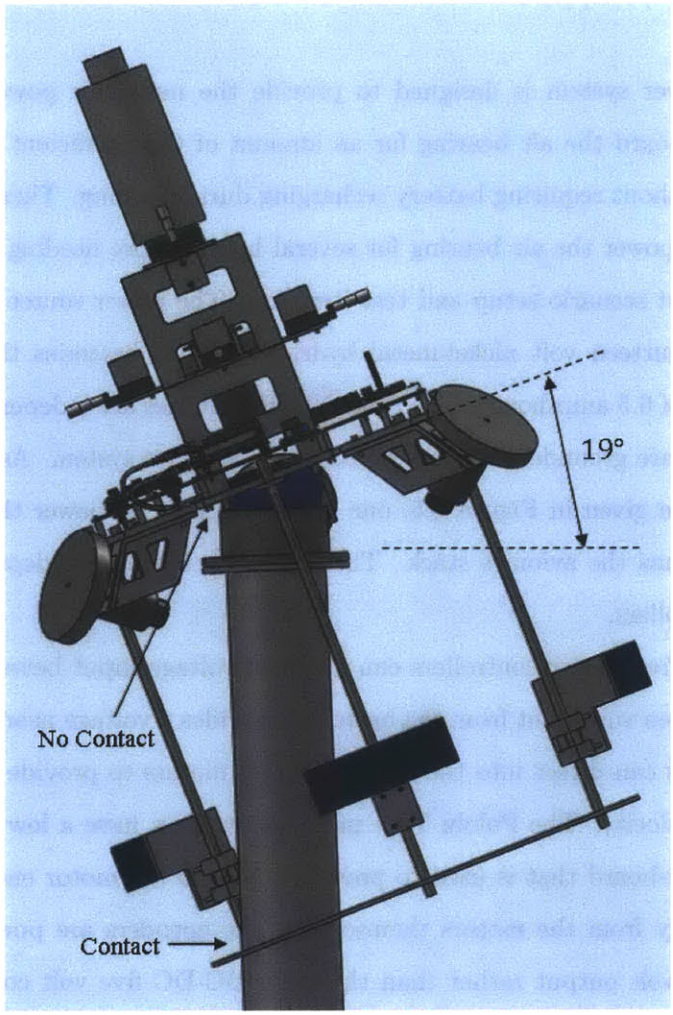


Figure 4-4: Air Bearing Maximum FIR x Axis Rotation

The safety ring is mounted 0.55 meters below the bottom of the base plate and has an inner diameter of 0.52 meters. The outer diameter of the inertial support column is 0.129 meters. Modeling these dimensions in SolidWorks produces an accurate angular rotation limit in the FIR x and y axes. The rotation limit due to the safety ring in the FIR x and y axes is nineteen degrees for each axis. There is no rotation limit about the FIR z axis.

The structure is designed to be versatile enough to accommodate a range or test components as well as provide support for those components. The structure also allows for center of mass adjustment to reduce gravity disturbance torques and allows for safe operation by eliminating the possibility of damage due to over-rotation.

4.1.2 Power

The testbed power system is designed to provide the necessary power to the electrical components on-board the air bearing for an amount of time sufficient to perform desired test scenarios without requiring battery recharging during testing. Therefore, the batteries must be able to power the air bearing for several hours before needing to be recharged to accommodate test scenario setup and test iteration. The power source for the air bearing includes three fourteen volt nickel-metal hydride (NiMH) batteries that each provide a charge capacity of 6.5 amp-hours. The three NiMH batteries are independent of each other, though all three are grounded together through the avionics system. As can be seen in the electrical diagram given in Figure 4-5, one battery is used to power the DC-DC five volt converter that runs the avionics stack. The other two batteries independently power the two motor controllers.

The Pololu Trex motor controllers can accept a voltage input between six and sixteen volts. The fourteen volt input from the batteries provides a voltage near maximum that the motor controllers can direct into the reaction wheel motors to provide maximum reaction wheel angular velocity. The Pololu Trex motor controllers have a low power DC-DC five volt converter on-board that is used to provide power to the motor encoders, which operate independently from the motors themselves. The encoders are powered by the motor controller's five volt output rather than the main DC-DC five volt converter so that the encoders are only on when the reaction wheels are in use. Reference Figure 4-5 for wiring clarification and Table 4.3 for the motor controller power requirements. Often the avionics stack must be powered to update or test software algorithms, and powering the reaction wheels in this case is unnecessary. Therefore, each battery has its own power switch so that power to the avionics stack can be controlled independently of the motor controllers.

Table 4.3: Pololu Trex Motor Controller Power Requirements

	Current (A)	Voltage (V)	Power (W)
MC One (with Encoders)	0.323	14.0	4.522
MC Two	0.089	14.0	1.246
Reaction Wheel (max torque)	5.9	14.0	82.6

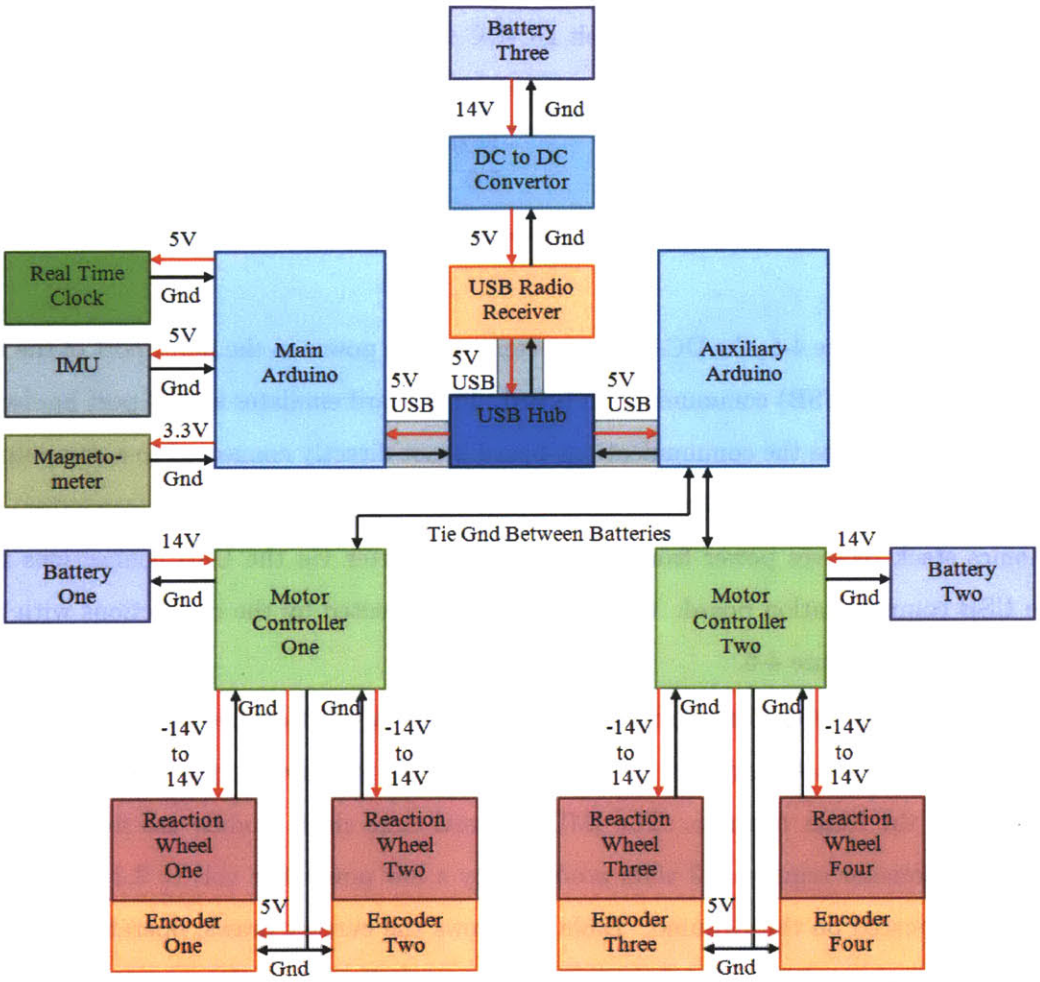


Figure 4-5: Air Bearing Electrical Diagram

The avionics stack requires a five volt input, and is powered by a ten watt DC-DC converter that can take an input from ten to thirty-six volts and provide a regulated five volt output with up to two amps of current draw. As shown in Table 4.4, the output from the DC-DC converter to the avionics stack is 0.514 amps, which is only about twenty-five percent of the DC-DC converter's operating limit. Table 4.4 also gives the DC-DC converter's power efficiency.

Table 4.4: Five Volt DC-DC Converter Efficiency

	Current (A)	Voltage (V)	Power (W)
Input	0.242	14.0	3.388
Output	0.514	5.0	2.570
Efficiency			75.8%

As seen in Figure 4-5, the DC-DC converter provides power to the input port of the Universal Serial Bus (USB) communication board. This board emulates a USB port hardwired to a computer. Since the communications board is not directly connected to a computer, it must receive power from another source; in this case, the DC-DC five volt converter. The avionics stack receives power from the five volt converter via the USB connections from the USB communication board. USB cables are represented by the connections with gray backgrounds in Figure 4-5.

The Main and Auxiliary Arduinos are powered via USB connections. The external components like the sensors and real time clock (RTC) module are powered from voltage outputs on the Main Arduino. The IMU and real time clock module use five volts, and the magnetometer requires 3.3 volts produced by a low power five volt to 3.3 volt DC-DC converter located on the Arduino. Table 4.5 shows the current draws, operating voltages, and power consumed by each of the components as well as the total power consumed by the avionics stack.

Table 4.5: Avionics Stack Power Requirements

	Current (A)	Voltage (V)	Power (W)
Main Arduino Mega	0.032	5.0	0.16
Aux Arduino Mega	0.032	5.0	0.16
IMU	0.072	5.0	0.36
Magnetometer	0.001	3.3	0.003
Total	0.514	5.0	2.57

As seen in the above table, the total current from the DC-DC five volt converter is much higher than the sum of the components listed. This is due to the additional components not listed in Table 4.5. These components include the USB communication board and the USB Hub. The communication board is responsible for the majority of the additional current

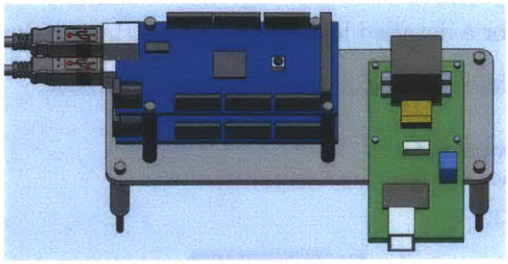
draw, and the USB Hub accounts for the remaining difference. Even with a total current draw of 0.514 amps at five volts, the battery is capable of providing power to the avionics stack for approximately thirteen hours assuming a fifty percent depth of discharge. For the motor controllers, each battery must power one controller and two reaction wheels. If the reaction wheels are all running at maximum torque, the batteries could sustain power for only about fifteen minutes assuming fifty percent depth of discharge. However, the batteries do not maintain constant maximum torque and have an average power draw of between 0.5 and one amp. At this rate of power consumption, each battery can provide power to the motor controllers for approximately 1.5 hours at fifty percent depth of discharge.

The air bearing power system provides the required power to operate the system for the length of time necessary to perform a complete test scenario to include test setup, modification, and iteration without needing to pause for battery charging. Therefore the test engineer can effectively ignore the power system during testing, which allows the engineer to focus on the test itself rather than supporting subsystems like power.

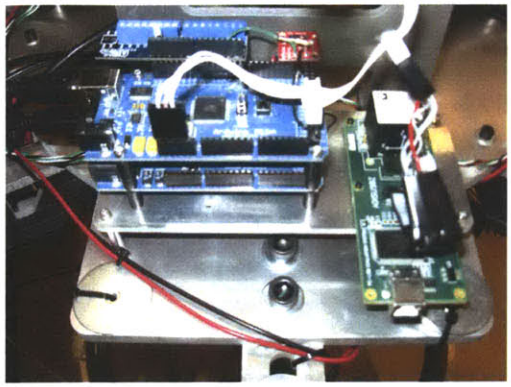
4.1.3 Avionics and Communication

The testbed avionics and communication system is made up of all the electrical components besides the reaction wheels themselves. The system includes two Arduino Mega Processors, two Pololu Trex Dual Motor Controllers, an Analog Devices Three Axis Inertial Measurement Unit, a MicroMag3 Three Axis Magnetometer, a Sparkfun Real Time Clock Module, and a TruLink Wireless USB Connector. These components work together to produce an operational avionics system capable of estimating the air bearing's inertial angular orientation and rate, determining the attitude control inputs necessary to move the air bearing to the commanded angular orientation and rate, and send those commands to the attitude control actuators; in this case the reaction wheels. Figure 4-6 shows the SolidWorks model of each of the component mounting plates next to images of the actual plates. All of the avionics and communication components listed above are located on the component mounting plates except for the magnetometer. The Main and Auxiliary Arduinos along with the IMU are located on component plate one in Figures 4-6(a) and 4-6(b). The communication board, the five volt DC-DC converter, and the avionics stack power switch are located on component plate two in Figures 4-6(c) and 4-6(d). The two motor controllers along with the power switches for each are located on component plate three in Figures 4-6(e) and 4-6(f).

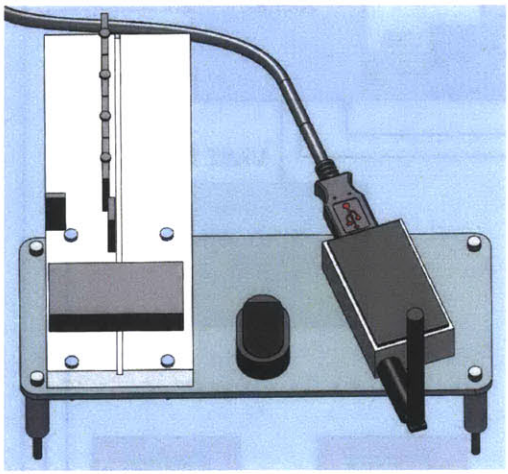
The magnetometer is mounted just above the fourth reaction wheel within the center support truss. The magnetometer can be seen on its white breadboard in Figure 4-3. The magnetometer is located in this position in order to reduce its translational motion while the air bearing is rotating. Due to the laboratory environment, the magnetic field around the air bearing is not constant. Reducing translational motion of the sensor helps to mitigate the effects caused by an inconsistent magnetic field. The magnetometer does not translate if the vehicle is rotating about the ABBF z axis. However, due to the fact that the magnetometer is mounted above the center of rotation, it does translate if the vehicle rotates about the ABBF x or y axes, though this movement is minimal since the vehicle is constrained to nineteen degrees off nominal rotation in these two axes.



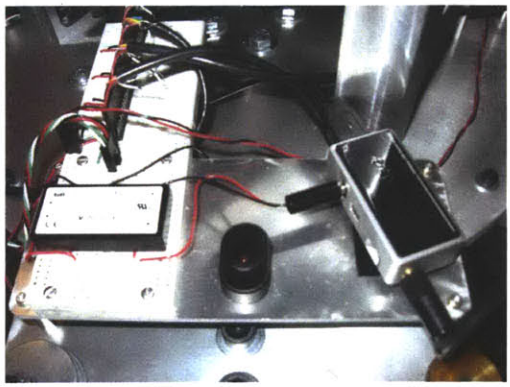
(a) Component Plate One - SolidWorks



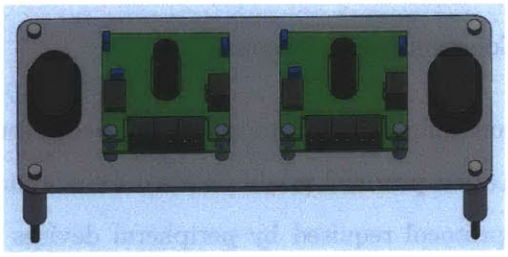
(b) Component Plate One - Actual



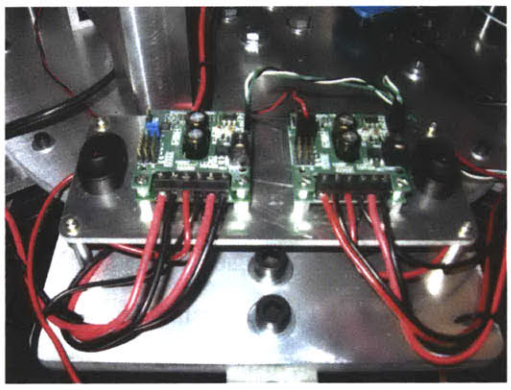
(c) Component Plate Two - SolidWorks



(d) Component Plate Two - Actual



(e) Component Plate Three - SolidWorks



(f) Component Plate Three - Actual

Figure 4-6: Avionics Component Plates

Each of the avionics and communication components listed above is connected via data communication lines. Figure 4-7 shows a block diagram of the data connections between the devices. This figure is similar to the electrical diagram given in Figure 4-5 but represents communication lines rather than power lines. For a detailed look at the air bearing's avionics connections, reference Appendix B for a complete testbed avionics wiring schematic.

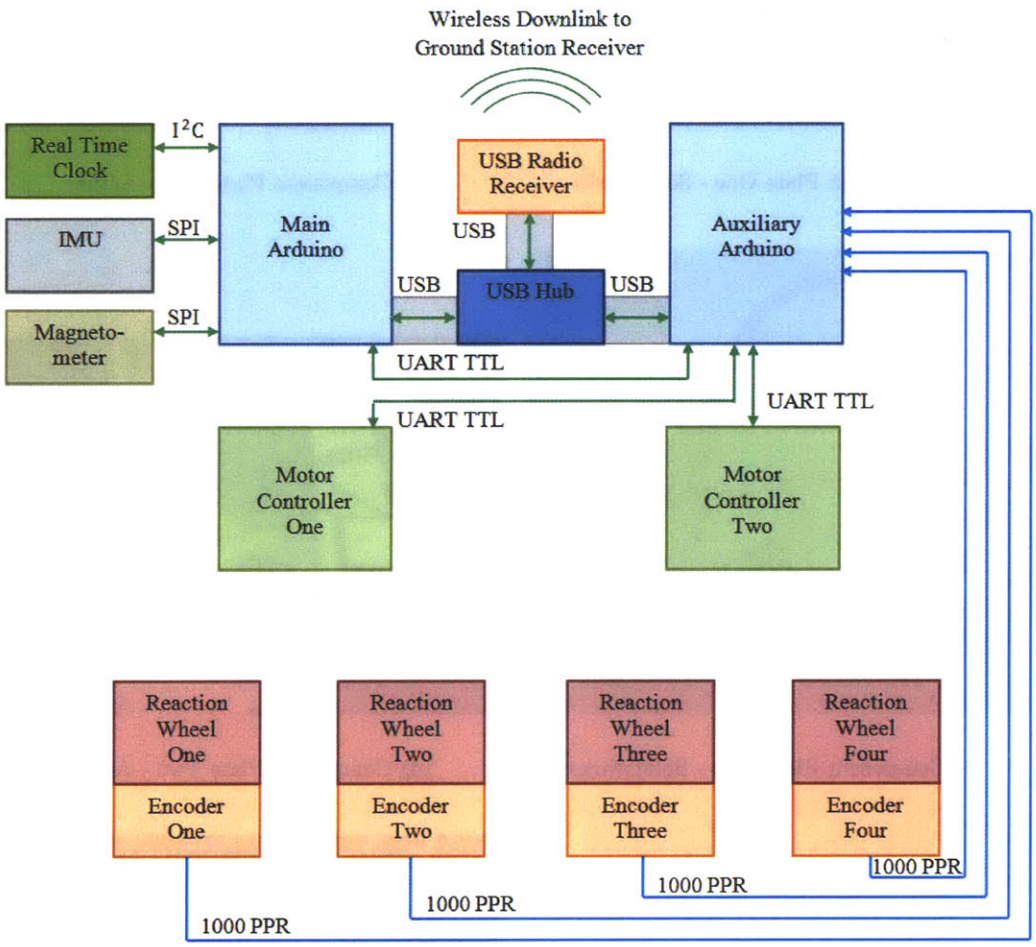


Figure 4-7: Air Bearing Communication Diagram

The Avionics stack uses several different communication protocols as can been seen in the above figure. Though using one communication protocol would be ideal for simplicity, the processors must use the communication protocol required by peripheral devices like the IMU and motor controllers. The following bullet list gives a brief explanation of each communication protocol used by the air bearing avionics system.

- UART TTL - Universal asynchronous receiver/transmitter operating at transistor-transistor logic (TTL) level. UART translates data from parallel to serial form for transmission across the communication line. This particular serial transmitter uses TTL level data, which consists of zero volts representing a digital zero bit, and five volts representing a digital one bit. UART requires two data lines for communication; one transmits serial data and the other receives serial data [17].

- SPI - Serial peripheral interface. SPI operates similarly to UART but allows for communication between a master device and multiple slave devices through the same communication port. SPI achieves this by using four lines for communication. The two data lines are Master Out Slave IN (MOSI) and Master In Slave Out (MISO). Unlike UART, SPI must provide a clock pulse to the slave device in sync with the data being sent or received. The clock pulse travels on the Serial Clock line (SCLK). The fourth line is the Slave Select line (SS). The master must have a slave select line for each device using SPI communication. The master selects which device to communicate with by setting the corresponding Slave Select line to a predetermined voltage, usually zero volts. The master keeps the other Slave Select lines at five volts which tells the other slave devices to ignore the communication [17].

- I^2C - Inter-Integrated Circuit. I^2C is another type of serial communication protocol using only two lines for communication. I^2C can also communicate with multiple devices using the master, slave format. The first line is the Serial Clock line (SCLK) just like the line used by SPI. The second line is the Serial Data Line (SDA), which handles all data transmitted or received. I^2C uses only one data transmission line by sending start and stop bytes with each data transfer that define what is being sent, who should receive the data, and what data is being requested [38].

- USB - Universal Serial Bus. USB uses serial communication in a 'black box' style. Using generic connectors, USB allows two line serial communication between two devices. USB also provides a five volt power line and a ground line, which can be used to power the peripheral device.

- 1000 PPR - One Thousand Pulses Per Revolution. This is not a communication protocol but rather the type of data being received from the motor encoders as shown

in Figure 4-7. The encoders produce a zero to five volt square wave at one thousand cycles per revolution of each reaction wheel. This data is interpreted by the Arduino processors to determine how far the wheel has traveled. Using time data, the processors can also determine each reaction wheel's angular velocity.

The heart of the avionics system are the two Arduino Mega processors located on the first component mounting plate as shown in Figure 4-6. These processors work together to provide real time attitude estimation and control by taking inputs from the real time clock module, IMU, magnetometer, motor encoders and each other, and then providing control commands to the reaction wheels as well as outputting state information to the ground station. The specific requirements of each Arduino will be discussed in Section 4.2.1 and 4.2.2. Table 4.6 lists several important specifications of the Arduino Mega processors. The complete Arduino Mega specification sheet can be found at Arduino's website.[1]

Table 4.6: Arduino Mega Specifications

Operating Voltage	5V
Clock Speed	16 MHz
Flash Memory	128 KB
UART Ports	4
SPI Ports	1
I^2C Ports	1
External Interrupts	6
PWM Output Pins	14
Analog Input Pins	16

The Arduinos have an internal time function, but this function is not accurate enough to provide real time operation. Therefore, an external real time clock module designed and built by Sparkfun Electronics is used to provide a real time clock input. The module uses the I^2C communication protocol and is powered via a five volt input. However, a battery on-board the module powers the clock when the system is turned off. The battery can power the clock for nine years before needing replacement according to the specification sheet. The RTC module provides a square wave at 4.0 KHz to one of the Main Arduino's external interrupt pins. By defining how many pulses must be received between control cycles, the programmer can set the time step of the control cycle to within 0.0002 seconds.

[1] `http://arduino.cc/en/Main/ArduinoBoardMega`

The Arduinos wirelessly communicate with the ground station computer via the USB communication board, made by TruLink. The USB communication board transmits and receives at frequencies between 3.1 and 4.8 GHz [4]. The receiver translates the wirelessly received data into serial format for transmission along a standard USB 2.0 port. This is very convenient because many devices communicate via USB or have conversion chips that allow communication via USB. Overall, the USB communication board allows the ground station computer to think it is hardwire connected to the Arduino Megas via a standard USB cable.

4.1.4 Attitude Determination Sensors

The two attitude sensors on the air bearing are the IMU and the magnetometer. The other feedback sensors are the four reaction wheel motor controllers. Though these sensors are not used to directly measure attitude, their measurements are used to determine reaction wheel speed, which is used in the estimation and control process. Therefore, all of these sensors are important to producing an accurate state estimation and the correct control inputs. The first sensor to be discussed is the IMU.

The three axis IMU is made by Analog Devices and has the specific part number ADIS 16365. The IMU communicates via an SPI port and can send a full data set at a maximum rate of 819 Hertz, which is much higher than any control cycle that would be used by the avionics system. The IMU is poled for data once per control cycle. The requested measurements from the IMU are the angular rates in each of the three axes, and the linear accelerations in each of the three axes. Though the IMU can be set to output angular rate at up to plus or minus 300 degrees per second, it is set to output rate at plus or minus seventy-five degrees per second since even this lowest setting is a higher rate than would be achieved by the air bearing. Using the lowest rate bandwidth, the IMU provides the highest precision at 0.0125 degrees per second resolution. The measurement from the IMU is given in degrees per second and immediately converted to radians per second for use in the estimation and control algorithm. The linear accelerometers measure acceleration in units of $g's$ (9.81m/s^2). The accelerometers have only one setting and can measure up to plus or minus eighteen $g's$ at a resolution of 3.38 $mg's$. Because Earth's gravity is the largest contributor of linear acceleration on the air bearing, the accelerometers only measure between plus or minus one g. The six measurements from the IMU are used to update the

state estimate once per control cycle. The complete specification sheet for the IMU can be found at Analog Device's website.[2]

The three axis magnetometer is known as the MicroMag3 and is made by PNI Corporation. The magnetometer also communicates using SPI protocol and can send a full data set at up to 2000 Hertz. Just like the IMU, the magnetometer is poled for data once per control cycle. The measurement consists of magnetic field strength readings in each of the three axes. The magnetic field strength can be read at up to plus or minus 1100 micro Tesla before the sensors saturate. The magnetometer provides measurement resolution of 0.032 micro Tesla. The three measurements from the magnetometer are combined with the IMU measurements to update the air bearing state estimate once per control cycle. The complete MicroMag3 data sheet can be found on Sparkfun.com.[3]

The four motor encoders are bearing style, three-channel encoders and are made by Motion Control Group, the same company that makes the reaction wheels. The encoders output a five volt square wave at 1000 cycles per revolution of the wheel. Though the Arduinos take only one pulse train as an input, each encoder has a total of four square wave outputs that can be read by the Arduinos. In addition to the single square wave that is used as an input to the Arduino, there is a second square wave that is the negative of the first. This can be used to reduce signal noise by comparing the inputs from these two signals. The additional two inputs are made up of a square wave and its negative that are ninety degrees out of phase with the first set of square waves. These inputs can be compared with the first set to determine reaction wheel direction in addition to angular rate. These inputs would be beneficial to the state estimator but due to processor limitations, only one input per motor encoder can be handled. Therefore, only reaction wheel speed and not direction can be measured. The complete data sheet for the motor encoders can be found at Motion Control Group's website.[4]

The driving requirement for using two Arduino processors is measuring the motor encoders while maintaining real time processing. The motor encoders are measured using interrupt pins that stop the processor at each new square wave input to increment a counter and keep up with the number of square waves that are measured. This process must be

[2] http://www.analog.com/en/mems/imu/adis16365/products/product.html

[3] http://www.sparkfun.com/datasheets/Sensors/MicroMag3\%20Data\%20Sheet.pdf

[4] http://www.ametektip.com/index.php?option=com_catalog\&view=models\&which=catalogs\&id=369

completed for each of the four wheels. At a maximum angular rate of approximately thirty revolutions per second, each encoder will output 30,000 square waves which equates to 30,000 processor interrupts. For four reaction wheels, there could be a maximum of 120,000 interrupts per second, which makes running a real time algorithm in addition to the interrupts difficult. Therefore, the Auxiliary Arduino is added with a primary purpose of taking inputs from the motor encoders and relaying reaction wheel commands from the Main Arduino to the motor controllers. The algorithm necessary to receive reaction wheel commands from the Main Arduino and relay them to the motor controllers is relatively simple and can be accomplished once per cycle despite the encoder interrupts. Once the Auxiliary Arduino relays reaction wheel commands to the motor controllers, it sends motor encoder information back to the Main Arduino, which is running the control algorithm in real time without being paused for motor encoder interrupts.

4.1.5 Attitude Control Actuators

The air bearing's attitude control actuators consist of two Pololu Trex dual motor controllers that each drive two reaction wheels. Three of the four wheels are oriented in an orthogonal set symmetric about the ABBF z axis to simulate a common small satellite attitude control configuration. The fourth reaction wheel is mounted in alignment with the ABBF z axis so that its axis of rotation has equal projections in the rotation axes of the other three orthogonal reaction wheels, which allows for reaction wheel redundancy and failure testing as well as allowing for the development of angular momentum storage techniques.

The Pololu Trex motor controllers operate at the voltage of the power source; fourteen volts in this case. Each motor controller receives speed and direction commands for both motors in packetized serial format via UART TTL protocol, and sets the voltage across the two motor leads for each motor depending on the received command. The motor controllers can discretize the voltage from zero to maximum (fourteen volts) in 127 settings. Therefore, using the current power source, the motor controllers control voltage with a resolution of 0.11 volts. The motor controllers can switch the polarity of the motor leads and provide the same voltage resolution in the opposite direction. Voltage polarity is set by the reaction wheel direction command and voltage level is set by the reaction wheel speed command. The motor controllers can also brake the motors to zero velocity by shorting the motor leads, which causes resistance to motor rotation by eliminating a path for the electrical

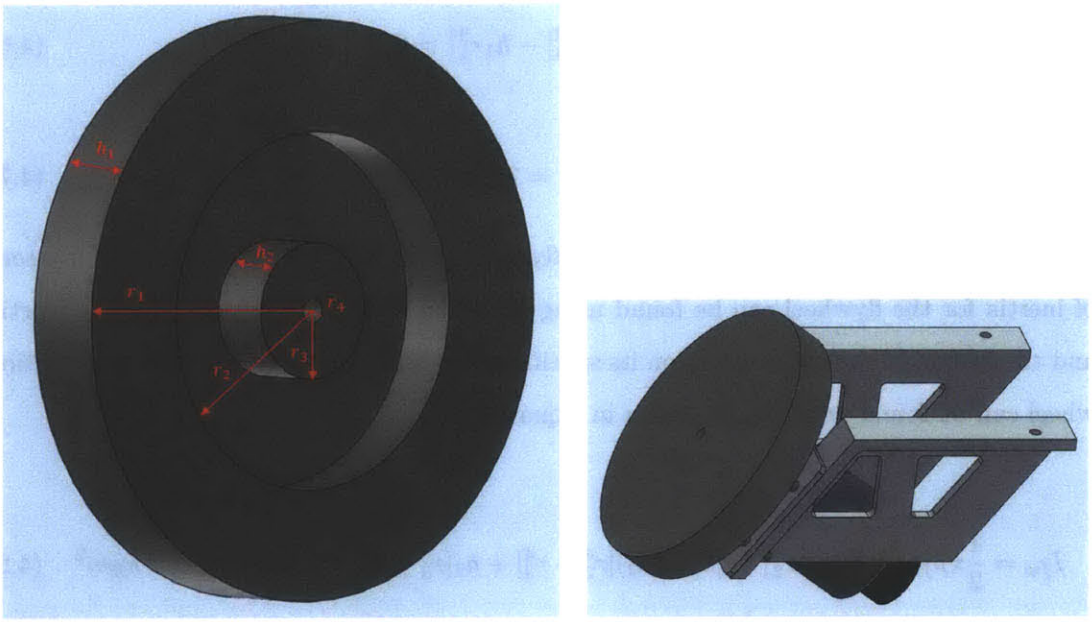
current induced by the motor's rotation to escape. The specification sheet for the Pololu Trex motor controllers can be found at Pololu's website.[5]

The reaction wheel motors are made by Motion Control Group and are rated (based on the specification sheet) to provide continuous torque of 0.147 Nm and a peak rated torque of 0.295 Nm. Maximum voltage according to the specification sheet is sixty volts, which is significantly higher than the fourteen volt input from the batteries. The no load maximum speed is 628 radians per second, which is also significantly higher than the achieved maximum speed on the air bearing of 194 radians per second, due to the lower voltage along with the increased inertia from to the attached flywheel. Other motor characteristics from the specification sheet are previously listed in Table 3.2 for use in developing the reaction wheel equations of motion. The complete motor specification sheet can be found at Motion Control Group's website.[6]

The flywheels attached to each of the four reaction wheels are designed and machined the same. Figure 4-8(a) gives a SolidWorks rendering of the flywheel and Figure 4-8(b) gives a rendering of the wheel as attached to the motor and reaction wheel support frame. The flywheels are relatively large compared to reaction wheels that might be used on ESPA class spacecraft. However, disturbance torques in the space environment have much less magnitude than the disturbance torques faced by the air bearing in operation. Therefore, the reaction wheels must be larger to compensate for these torques.

[5] http://www.pololu.com/catalog/product/777

[6] http://www.ametektip.com/index.php?option=com_catalog\&view=models&which=catalogs\&id=399\&Itemid=107\&lang=en



(a) Flywheel with Dimensions

(b) Reaction Wheel Assembly with Frame

Figure 4-8: Air Bearing Reaction Wheel Model

Table 4.7 lists the dimensions of the flywheel labeled in the figure above [20]. The dimensions are found by measuring the physical wheel rather than using the SolidWorks model for accuracy in determining inertial characteristics.

Table 4.7: Flywheel Mass and Dimensions

Description	Label	Value
Mass	m_{fw}	2.304 kg
Outer Radius	r_1	73.57 mm
Slot Outer Radius	r_2	45.77 mm
Slot Inner Radius	r_3	17.54 mm
Inner Shaft Radius	r_4	8.26 mm
Disc Height	h_1	23.01 mm
Slot Inner Height	h_2	16.51 mm

The volume of the flywheel can be found using Equation 4.6 and the dimensions listed above. The volume is necessary in finding the density of the flywheel as shown in Equation 4.7 [20].

$$V_{fw} = \pi\left[h_1 r_1^2 - h_2[r_2^2 - r_3^2] - h_1 r_4^2\right] = 2.942 \times 10^{-4}\, m^3 \tag{4.6}$$

$$\rho_{fw} = \frac{m_{fw}}{V_{fw}} = 7831 kg/m^3 \tag{4.7}$$

Using density and the dimensions of the flywheel given in Table 4.7, the second moment of inertia for the flywheel can be found using Equation 4.8 [20]. From the flywheel inertia and the inertia of the motor given on its specification sheet, the total inertia of the reaction wheel can be found and is given below in Equation 4.9.

$$I_{fw} = \frac{1}{2}\pi\rho_{fw}\left[h_1[r_1^4 - r_2^4] + (h_1 - h_2)[r_2^4 - r_3^4] + h_1[r_3^4 - r_4^4]\right] = 7.224 \times 10^{-3}\, kgm^2 \tag{4.8}$$

$$I_a = I_{fw} + I_{motor} = 0.00725\, kgm^2 \tag{4.9}$$

The total angular momentum storage capability of each reaction wheel based on the maximum angular rate and inertia of the wheel is given below in Equation 4.10.

$$H_{rw\,max} = I_a \Omega_{max} = 1.407\, Nms \tag{4.10}$$

The four air bearing reaction wheels provide the torque and angular momentum storage capability necessary to control the air bearing for enough time to complete a range of ADCS tests despite the disturbance torques associated with a laboratory environment. Furthermore, the four wheels are configured to allow for a series of tests on operating the wheels themselves to include angular momentum transfer techniques, orthogonal reaction wheel control, reaction wheel redundancy, and reaction wheel failure analysis.

4.1.6 Ground Station

The ground station can be any thirty-two bit computer capable of connecting with the TruLink wireless USB adapter, which connects using a standard USB port. Included software is necessary to install the drivers required to interface with the wireless USB adapter, and this software is stored with the current ground station desktop computer. The wireless

USB adapter has an advertised range of thirty feet; however the current ground station computer is located within six feet of the USB receiver on the air bearing.

The ground station computer must also be able to install and run the Arduino's coding environment and compiler, known as an Integrated Development Environment (IDE). The IDE is regularly updated by the manufacturers of the Arduino and will automatically recognize when new versions are released and suggest updating, which is beneficial. The IDE includes a series of functions and libraries that can be used to simplify the coding process, and new versions of the environment include additional functions. Instructions for downloading the Arduino's IDE are included in the air bearing user's manual.

With the avionics stack powered on and the Arduino coding environment open, the Arduino boards can be selected based on which communication port they are connected. The ports are arbitrarily numbered by the ground station computer so some investigation may be required to determine which communication port is associated with which Arduino. In the normal configuration, there will be two Arduinos connected to the ground station computer via USB comm ports. Using the coding environment, new code can be wirelessly uploaded to either Arduino at the click of a button, and information sent back via the wireless serial port can be viewed once received and stored in batch files after the test is completed.

Batch files from test scenarios can be analyzed at the ground station using Microsoft Excel and MATLAB. Several MATLAB script files have been developed for general data analysis, and if these scripts do not meet the needs of the test, the test developer can easily write new MATLAB scripts to evaluate data collected in the batch files. Overall, the ground station computer functions as the avionics interface to the air bearing testbed. The Arduinos can be reprogrammed from the ground station before a test, data can be collected during a test, and analyzed afterwards.

4.2 Testbed Software Development

Provided with the testbed is a baseline set of software for the Main and Auxiliary Arduinos. The software is meant to provide a foundation that can be built upon to meet the needs of future test scenarios. The provided software is designed to run estimation and control algorithms exactly as they are modeled in the air bearing testbed Simulink simulation

described in Chapter 3. If commanded angular orientation and rate profiles are all that need to be changed for a given test, modification to the provided software will be minimal. However, if a new sensor or actuator is added to the testbed, software modification will be more complex.

Arduinos use a C++ based language which is written in the Java based IDE. The IDE allows Arduino code to be written, compiled, and uploaded to the Arduino using one program. The IDE also has a serial window that allows data to stream back from the Arduino and be collected using the same program. As described in the Ground Station section, the IDE is provided by the Arduino's manufacturer and can be downloaded from their website at no cost.

Due to the hardware interrupt limitations discussed in the Attitude Determination Sensors section, the air bearing avionics stack uses two Arduino Megas to run the estimation and control algorithm. The primary functions of the baseline software that is provided for both the Main and Auxiliary Arduinos will be described below.

4.2.1 Main Arduino

The Main Arduino (MAR) runs the main attitude estimation and control algorithm. During each control loop, the MAR requests and receives inputs from the IMU and Magnetometer as well as motor encoder inputs from the Auxiliary Arduino (AAR). The MAR then propagates a model of the air bearing's state, finds the best estimate of the air bearing's state by running an EKF, propagates the commanded air bearing state, calculates commanded reaction wheel velocities using the same suite of control algorithms described in the Control Module section in the simulation development chapter, and sends reaction wheel velocity commands to the AAR. The MAR then sends requested data to the ground station before completing the estimation and control loop.

The provided software for the MAR is divided into four code files that are put together by the IDE's compiler. Each of these four files are represented by tabs in the IDE making it easy to view and edit each code file individually. The four code files for the MAR are briefly described in the bullet list below.

- mega_main.pde - This is the primary code file and calls the other three files to be included once compiled. The file has a '.pde' file type and can be opened using the Arduino IDE.

- functions.h - This header file includes all of the non-mathematic functions required by the mega_main.pde code file. Examples of these functions are the SPI communication functions for the IMU and magnetometer.

- matrix.h - This header file includes all the mathematical functions required by the estimation and control algorithm.

- init.h - This header file includes all of the initial condition variables that can be changed by the user. These variables include initial reaction wheel velocities and control algorithm gains.

Because the mega_main.pde file calls functions and variables from each of the header files, the header files will be described first. Beginning from the bottom of the bullet list, the init.h file simply defines a series of variables for use in the control algorithm. These variables include initial reaction wheel rates, initial air bearing state, and several key estimation and control matrices. The entire init.h file is developed by a MATLAB script called test_init.m, which is designed to take inputs from the user like initial reaction wheel speed and estimation/control algorithm changes. The MATLAB script calculates the required changes in the variables included in the init.h file, and then creates a new init.h file to incorporate the changes made by the user. The user can then replace the existing init.h file with the new one that includes the updates relating to the desired initial conditions. The process of using the MATLAB script not only makes defining initial conditions easier, but also makes the initial condition input method to the physical air bearing code practically the same as the input method to the Simulink simulation. Therefore, the same set of initial conditions can be used to run the Simulink simulation as well as the physical air bearing. The process for creating the init.h file using the test_init.m MATLAB script and replacing the previous init.h file in the Arduino's IDE will be described in detail in Section A.2.3 of the testbed user's manual given in Appendix A. Included in Section C.1 of Appendix C is a copy of the provided init.h code file. This copy of the init.h file is the output of the provided copy of the test_init.m MATLAB script in Section C.6 of Appendix C. Though

the provided init.h file is operational, the user should set the desired initial conditions in the test_init.m MATLAB script and generate an updated version of the init.h header file for use.

The matrix.h file includes all the mathematical functions that are used by the estimation and control algorithm that are not already included in the Arduino's coding environment. Most of these functions are vector and matrix functions since only scalar functions are included in the environment. The functions include three by three matrix multiplication, three by one vector cross product, three by one vector magnitude, and a numerical approximation of the the three by three matrix exponential. The numerical approximation of the matrix exponential is the most complex of the functions but is a requirement in determining an attitude estimate from the IMU accelerometers and magnetometer as described in Section 3.2.3. Equation 4.11 gives the numerical approximation used to find the matrix exponential [42]. A copy of the provided matrix.h file is included in Section C.2 of Appendix C.

$$e^A \approx \sum_{i=0}^{8} \frac{1}{k!} A^k \tag{4.11}$$

The functions.h file includes all additional functions called by mega_main.pde that are not included in the matrix.h file. These functions include the read and write functions associated with the IMU as well as the functions to read magnetometer measurements. This file also includes the functions used to send data back to the ground station computer during testing. The functions used to find the reference magnetic field vector and the function to set the initial reaction wheel velocity during the setup phase are also in this file. A copy of the provided functions.h file is included in Section C.3 of Appendix C.

The mega_main.pde file is broken into sections to help the user understand the algorithm's operation and aid in modification if necessary. The following numerical list gives each section of the code and a description about the section's operation. The sections listed below are labeled exactly the same in the provided software. A copy of the complete provided mega_main.pde file is included in Section C.4 of Appendix C.

1. Set Desired Outputs - The first section is where the user decides what data is to be sent back to the ground station during the test. Almost every variable in the estimation and control algorithm is available for output. In this section, the user can

also decide how to format spacing between data for clarity. For all variables chosen to be output during testing, the program will send back a header line during the setup phase that labels each column of data. Besides visually monitoring the air bearing during testing, the data sent to the ground station computer is the only information stored during each test.

2. Initialize Required Variables and Header Files - In this section, the three header files described above are included in the code. All variables used in the algorithm that are not previously defined in the header files are included as well.

3. Initialize Estimation/Control Loop - The setup loop is run in this section. The loop runs one time and initializes many of the Main Arduino's interfaces. The setup loop opens the UART and I^2C communication ports and defines their communication rate. The loop also defines the operation of each of the digital input/output (DIO) pins. The setup loop then poles the magnetometer for data to determine the reference magnetic field measurement for use throughout the remainder of the current test, and commands the reaction wheels to their required angular velocity before beginning the estimation/control algorithm. The setup loop requires approximately five seconds to determine a reference magnetic field. If the reaction wheels are given a nonzero initial angular velocity command, the setup loop pauses for ten seconds to allow the wheels to reach steady state before beginning the control loop. If the reaction wheel initial angular velocity is set to zero, the ten second pause is skipped.

4. Begin Estimation/Control Loop - This section highlights the beginning of the code that runs every time step. The first thing to be defined at the beginning of the control loop is the current time and the change in time since the last control cycle. The program begins the control loop when the counter incrementing at 4 KHz hits the desired number. This number is set by the user to determine the time step of the estimation/control algorithm.

5. Write Commanded Reaction Wheel Angular Rate to Motor Controllers - This section sends the reaction wheel speed and direction commands determined at the end of the previous cycle to the AAR. The reaction wheel commands are sent at the beginning of the control cycle because the AAR is triggered to begin its loop when it receives

data from the MAR. In this way, the MAR and AAR are operating in parallel. The AAR completes its loop by sending motor encoder information back to the MAR, which will read the data at the end of the estimation/control algorithm. Reference Figure 4-9 for clarity on the parallel operation of the two Arduinos.

6. Read IMU - This section calls the functions required to take a measurement from the IMU. The measurement includes the three angular rate values as well as the three linear acceleration values. These six measurements are stored for use in the EKF.

7. Read Magnetometer - This section calls the functions required to take a measurement from the magnetometer. The measurement includes the three magnetic field values and is stored for use in the EKF.

8. Commanded Angular Orientation and Commanded Angular Rate - This section is where the user defines the commanded angular orientation and rate of the air bearing for the extent of the test scenario. Just as with the Command Module in the simulation, the user can define commanded angular rate for all three ABBF axes and integrate the rate to obtain commanded angular orientation, or define angular orientation and take the derivative to obtain the rate.

9. Extended Kalman Filter - The EKF section operates just as the Extended Kalman Filter Block in the simulation. A linearized state space model of the reaction wheels and air bearing is propagated in this section, and the measurements from the IMU and magnetometer are combined with the propagated state model to produce an estimate of the current state to be compared with the commanded state defined above. This block is computationally expensive because it requires many of the matrix functions in the matrix.h file including the matrix exponential function.

10. Control Law - This section also operates just like its counterpart in the simulation. A feedforward control algorithm is used to determine commanded reaction wheel angular acceleration from the commanded air bearing state and reaction wheel velocity. The feedforward reaction wheel commanded acceleration is added to the commanded acceleration from the LQR to determine the best reaction wheel commanded acceleration.

11. Determine Reaction Wheel Angular Rate from Motor Encoders - This section analyzes raw motor encoder data received from the AAR to determine the measured reaction wheel speed of all four reaction wheels in radians per second.

12. Reaction Wheel Motor Controller - This section takes the commanded reaction wheel rate from the control law and converts it into the correct set of packets to be sent to the motor controllers via the AAR. This section also incorporates the closed loop braking algorithm that uses reaction wheel rate feedback from the motor encoders to determine if the wheels are spinning faster than commanded and slows them down if necessary.

13. Print Results - This section includes the function calls to send user requested data to the ground station at the end of each estimation/control cycle.

14. Read Encoder Information from Auxiliary Arduino - The last step in the estimation/control loop is the read the motor encoder information sent by the AAR. The AAR requires less time to complete its loop and send the encoder data, so the data is available to be read by the MAR once it reaches the end of its loop.

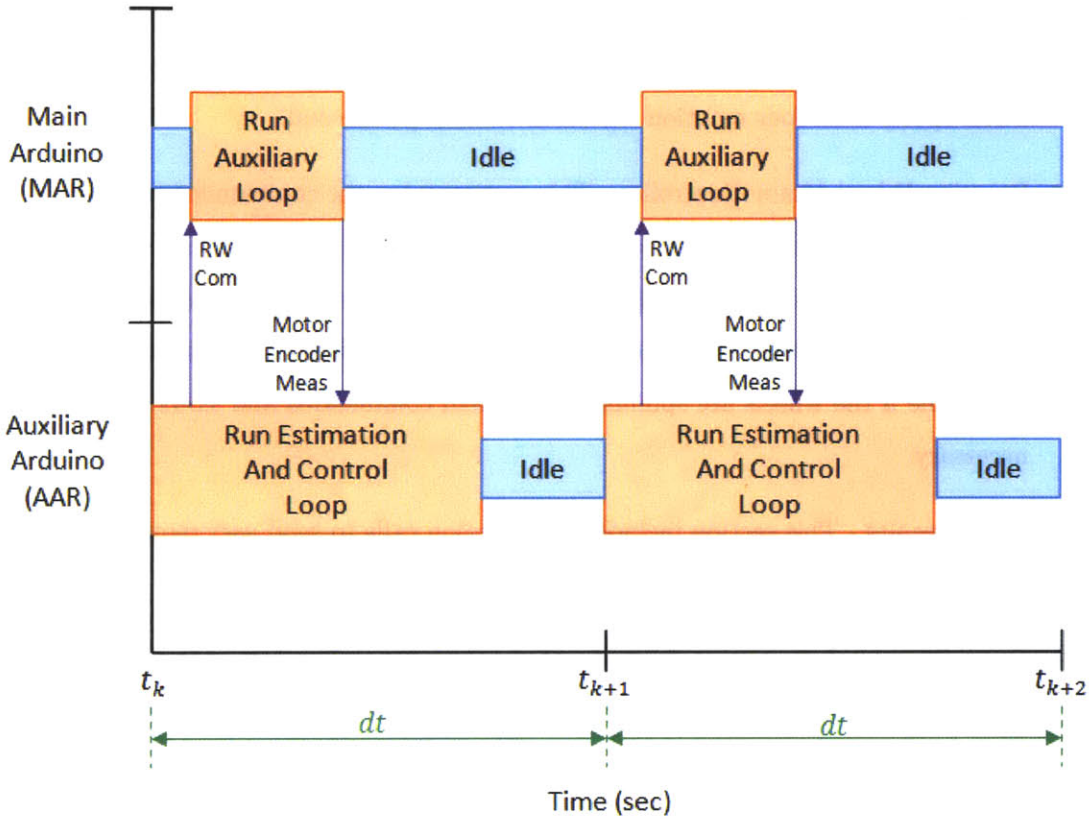


Figure 4-9: Main and Auxiliary Arduino Control Cycle Diagram

Once each step in the estimation/control loop is complete, the MAR waits and continues to increment the time counter. Once the time counter reaches the user defined value, the MAR enters the estimation/control loop again, and repeats steps four through fourteen above. The MAR continues running through the loop at each time step until it is reset or powered off and on again.

4.2.2 Auxiliary Arduino

The Auxiliary Arduino has fewer responsibilities in its loop than the Main Arduino. The primary reason for having minimal code in the AAR's loop is to set aside processor time to handle the large number of interrupts from the motor encoders. As previously described in the Attitude Determination Sensors section above, the motor encoders can interrupt the processor as much as 120,000 times per second. The interrupts occur continuously so the AAR's code should be minimal and not dependent on real time knowledge.

The provided software for the AAR is responsible for reading the interrupts from the four motor encoders and incrementing a counter after each interrupt to determine the number of pulses received from each encoder. Once during each control cycle, the AAR sends the current encoder counter value for each of the four motor encoders back to the MAR. The MAR is responsible for interpreting the encoder values and determining reaction wheel velocity since the velocity calculation requires real time knowledge. The AAR's second primary responsibility is the relay reaction wheel velocity commands from the MAR to the two motor controllers.

The provided software for the AAR is included in one file called mega_aux.pde. The following numerical list describes the different sections of the AAR code. The sections are labeled exactly as they are in the mega_aux.pde file. A copy of the complete provided mega_aux.pde file is included in Section C.5 of Appendix C.

1. Initialize Required Variables and Interrupt Pins - This section defines the variables used throughout the program as well as the pins used for motor encoder inputs.

2. Initialize Auxiliary Loop - This section opens all of the serial ports required for the AAR and defines their communication rate. All four hardware serial ports on the AAR are used. One port is used to communicate with the MAR, two more are used to communicate with the two motor controllers, and the fourth is used to communicate with the ground station computer. This section also sets the motor encoder input pins to function as interrupts.

3. Begin Auxiliary Loop - This section defines the beginning of the AAR's repeated loop. The AAR waits until it receives reaction wheel commands from the MAR via its serial port. Once these commands are received, the AAR enters its loop.

4. Write Motor Encoder Values to Main Arduino - Immediately after entering the loop, the AAR sends the current motor encoder information back to the MAR. Sending data requires time so the AAR sends the data first so it will be available to be read by the MAR when the MAR looks for the data at the end of its estimation and control loop.

5. Read Reaction Wheel Commands From Main Arduino - In this section, the AAR reads the reaction wheel data sent by the MAR and stores it in labeled variables.

6. Write Reaction Wheel Commands to Motor Controllers - The AAR then sends the data for the first two reaction wheels to motor controller number one, and the data for the second two reaction wheels to motor controller number two. This step completes the AAR's primary loop.

7. Define Motor Encoder Interrupt Functions - This section defines the functions that must be performed when a motor encoder interrupt occurs. These functions simply increment a counter after each motor encoder interrupt.

Once the AAR's primary loop is complete, it waits until it receives reaction wheel commands from the MAR before reentering the loop and completing steps three through six from the list above. Even when the processor is not running its primary loop, it is listening for interrupts from the motor encoders and incrementing the respective counters when interrupts occur.

4.2.3 Arduino Performance

In order for the avionics system to function properly, the Arduinos must first be able to store the complied code in their on-board flash memory, and they must be able to complete their software loops within the discrete time step set by the user. The Arduino Megas have 128 KB of flash memory, though some of this memory is required for formatting. With formating, each Arduino can store 126,976 bytes of information in flash memory. Table 4.8 lists the memory required to store the MAR's and AAR's provided software.

Table 4.8: Arduino Memory Requirements

	Software Size	Percent of Total
Main Arduino	52782	41.57%
Auxiliary Arduino	5396	4.25%

As can be seen in the above table, both the MAR and AAR are well within their memory storage limitations using the provided software. The next requirement is that the Arduinos complete their respective loops in less time than the user defined time step. The default time step used for air bearing validation testing described in Chapter 5 is 0.05 seconds, which equates to a twenty Hertz control cycle. Because the time step is defined by choosing

how many interrupts must occur from the 4096 Hz RTC module input between cycles, the time step must be given as an integer divided by 4096. The closest integer that solves this equation is 205. This process is given below in Equation 4.12.

$$0.05 seconds \approx \frac{205}{4096} = 0.05049 seconds \tag{4.12}$$

In order to ensure both Arduinos are capable of completing their control cycles within the required time step, they are tested in action using an oscilloscope. The test procedure begins by using a previously unassigned DIO pin on each Arduino. Immediately after entering the primary loop, the DIO pin is written to high voltage. The control loop then runs as usual and immediately before exiting the loop, the same DIO pin is written to ground. By placing an LED between the DIO pin and ground and placing an oscilloscope connection across the LED leads, the amount of time required to run the primary loop can be viewed on the oscilloscope output as a recurring pulse with a width that is some percentage of the recurrence period. For the normal control frequency of twenty Hertz, the recurrence period should be approximately 0.05 seconds. If the pulse representing the Arduino's run time per loop is a constant high voltage, the Arduino is unable to complete its loop within the time step provided and reenters the loop immediately upon completion because it is late. The estimation and control algorithm requires the MAR to run at a constant predetermined time step so reducing the time step to a value that is less than the time required for the Arduino to complete a control loop is unacceptable.

Figure 4-10 shows the results of the MAR timing test for a control cycle of twenty Hertz. The horizontal axis represents time and the vertical dotted lines are separated by five milliseconds. The control cycle is represented by the green line, and begins when the green line steps from zero volts to approximately 0.5 volts. Based on these results, the provided estimation and control loop requires forty-five milliseconds to complete, which is ninety percent of the available fifty millisecond time step. Notice that the next control cycle starts fifty milliseconds after the start of the first control cycle. At a control frequency of twenty Hertz, the estimation and control loop cannot be significantly expanded. If additional code must be written, the control frequency must be reduced to accommodate an extended runtime.

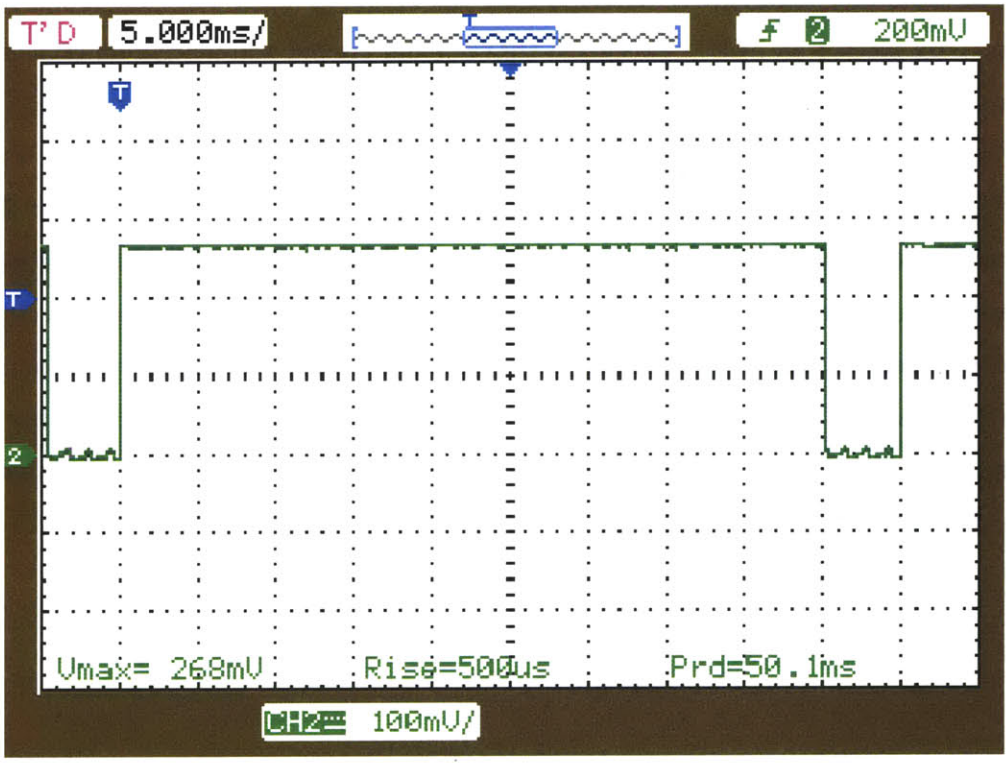


Figure 4-10: Main Arduino Program Runtime

In order to represent the process used to maintain real time operation, a second LED is pulsed each time the RTC module interrupts the MAR at 4096 Hz. Based on the interrupt frequency, the pulses should be counted once every 244 microseconds. Figure 4-11 shows the estimation and control loop in green and the pulses from the RTC module in blue. In this figure, the vertical dotted lines are separated by 100 microseconds. Though the spacing cannot be measured precisely, the blue pulses appear approximately 240 microseconds apart. Because the time step is 0.05 seconds, which equates to 205 pulses out of 4096 pulses per second as shown in Equation 4.12, the estimation and control loop begins just after the 205th RTC module interrupt is recorded.

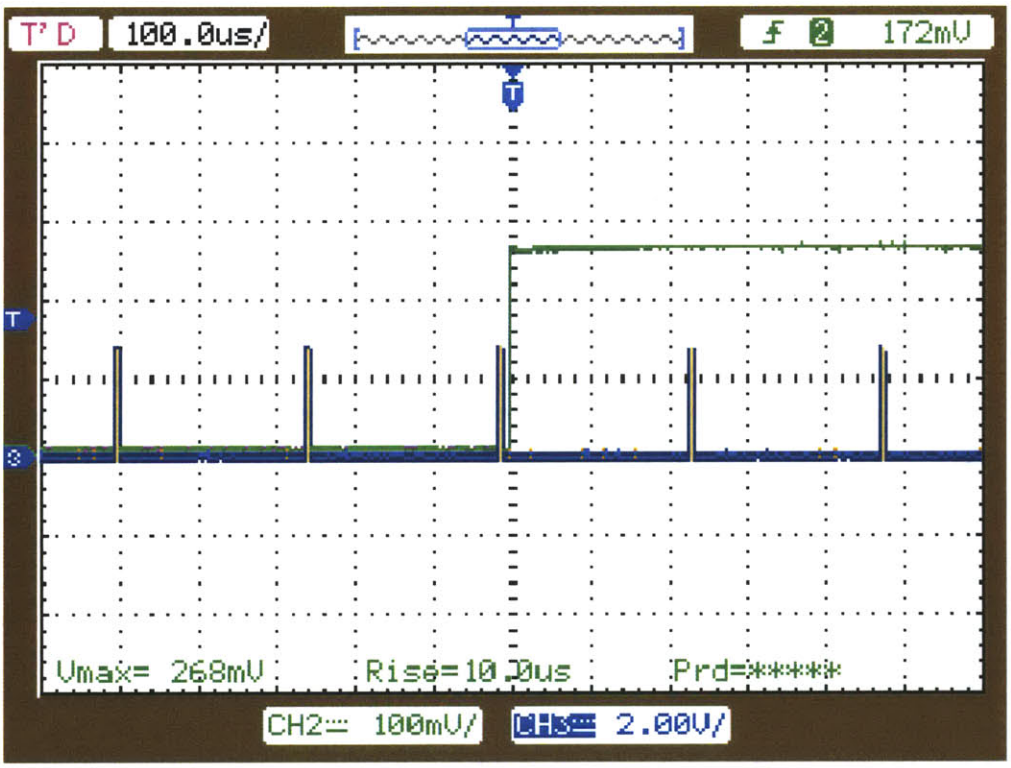


Figure 4-11: Main Arduino Program Start at Clock Pulse

As previously discussed, the AAR is triggered to run by inputs from the MAR during each control cycle. Figure 4-12 shows the MAR loop (green) and the AAR loop (blue) running in parallel. The figure is scaled just like Figure 4-10 with the vertical dotted lines separated by five milliseconds. Approximately one millisecond after entering the estimation and control loop, the MAR triggers the AAR to enter its loop by sending it reaction wheel commands. The AAR completes its loop in approximately three milliseconds. Though the AAR's loop runtime is significantly less than the MAR's runtime, the AAR must still listen for motor encoder interrupts, which are not represented in this figure.

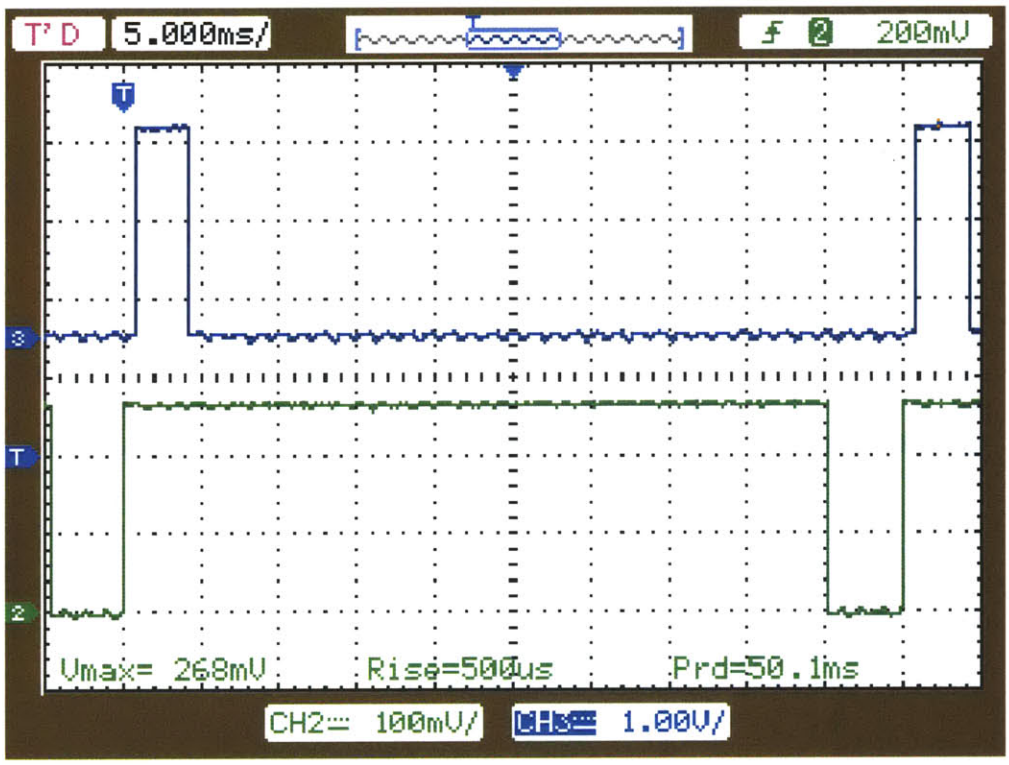


Figure 4-12: Main Arduino and Auxiliary Arduino Runtime

The pair of Arduinos work together to provide the processing power necessary to operate the ADCS testbed. Though they are operating at close to their maximum performance using the provided software and running at a twenty Hertz control cycle, they are flexible enough to handle significant changes in the size of the code running on both the MAR and the AAR, and the time step, which is easily adjusted by the user.

4.3 External Magnetic Field Generator

The ADCS testbed has a one axis external magnetic field generator that is used to produce a relatively consistent magnetic field in an otherwise noisy magnetic environment within the laboratory. The external magnetic field generator creates a field several times stronger than the ambient magnetic field, and the field can be used by the magnetometer as a means of attitude determination, and by a torque coil or torque rod as a means of attitude control. The increased strength of the generated magnetic field will allow the torque coil or torque rod to produce a high enough torque to overcome the disturbance torques within the laboratory. A magnetic torque device acting on Earth's ambient magnetic field would likely not be able to produce the torque required to overcome air bearing disturbance torques.

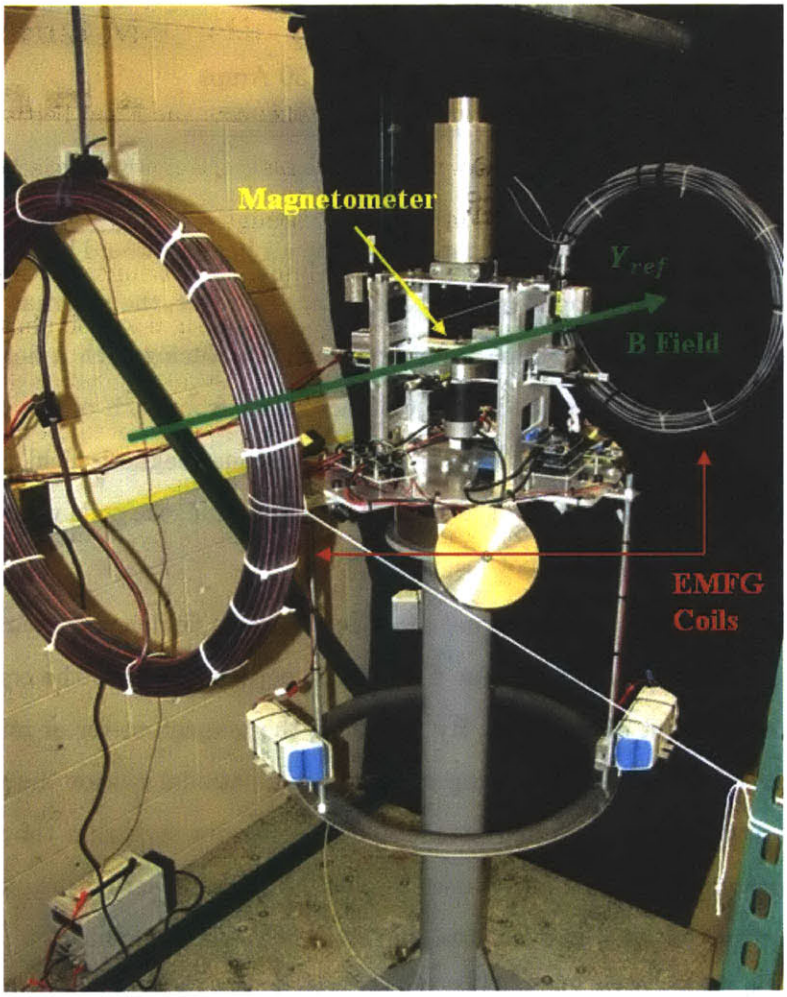


Figure 4-13: External Magnetic Field Generator

The external magnetic field generator is made up of two wire coils mounted on the external equipment frame surrounding the air bearing platform. Figure 4-13 points out the location of the coils with respect to the air bearing. The coils are mounted concentrically and perpendicular to the FIR y axis. By controlling the magnitude and direction of current flowing through the coils, which are wired in series with each other, a magnetic field vector of adjustable magnitude can be produced in either the positive or negative FIR y axis. Table 4.9 lists the specifications of the external magnetic field generator.

Table 4.9: External Magnetic Field Generator Specifications

Coil Radius	55 cm
Separation Distance	154 cm
Coil Turns	30
Wire Gauge	4
Normal Current	30 Amps
Normal Voltage	6 Volts

A uniform magnetic field environment around the air bearing would be ideal for attitude determination and control using magnetic equipment. However, the external magnetic field generator is not capable of providing a uniform field. Producing such a field requires the use of coils whose radius is equal to their distance apart from each other. Such coils are formally known as Helmholtz coils because of their unique magnetic field characteristics. To create a Helmholtz coil set using the same mounting frame, the coils would each require a radius of 1.54 meters, which is not feasible [23].

Though the current set of coils do not produce a uniform field across their full diameter, they are capable of creating a magnetic field vector perpendicular to the coils that passes through the center point of each coil. By mounting the magnetometer or magnetic torque device near the center of the air bearing to reduce translational motion during air bearing rotation, and mounting the external magnetic field generator such that the vector passing through the center of the coils also passes as close to the air bearing magnetic devices as possible, the error due to a non-uniform magnetic field can be reduced to an acceptable level. As seen in Figure 4-13, the magnetometer is mounted along the magnetic field vector produced by the coils. Due to the magnetometer's location on the air bearing, it does not translate during rotation about the ABBF z axis. If the air bearing rotates about the ABBF

x or y axes, the magnetometer will translate, but rotation about the ABBF x and y axes is constrained so the magnetometer will never translate far from the magnetic field vector between the center of the two coils. Characterization of the ambient magnetic field and the external magnetic field generator will be covered in Chapter 5.

4.4 SPHERES Overview

The ADCS testbed is designed to be able to integrate with the SPHERES testbed developed in the Space Systems Laboratory. The SPHERES testbed is a versatile ADCS testing platform that can be used to design and test ADCS systems for individual satellites as well as a close proximity satellite formation. Each SPHERES satellite uses a sonar metrology system to measure its inertial position. The satellite has twenty-four microphones located around its structure that listen for ultrasonic pulses produced by a series of ultrasonic emitters (known as beacons) located on the edges of the test volume. The satellite sends an infrared pulse that commands the beacons to produce an ultrasonic pulse one after another. By knowing where the beacon is located, when the beacon is supposed to emit a pulse, and measuring when the pulse is actually received by the microphones, the SPHERES satellite can determine its distance away from the beacon. By repeating this process with other beacons around the test volume, the SPHERES satellite can determine its position and angular orientation within the volume. The volume cannot exceed roughly ten cubic meters due to limitations in accurately measuring the ultrasonic pulses over larger distances. The SPHERES satellite also uses rate gyroscopes and linear accelerometers to support its attitude estimation algorithm. For attitude control, the SPHERES satellite has twelve cold gas thrusters that use compressed carbon dioxide gas as fuel. Two thrusters are mounted to produce pure torque in both the positive and negative rotational directions of all three orthogonal axes. In this way, the twelve thrusters provide complete attitude control [47].

A single SPHERES satellite can be mounted on top of the center support structure of the air bearing. Adding a SPHERES satellite to the air bearing testbed can significantly expand the functionality of the combined ADCS testbed system. The satellite can operate independently of the air bearing's avionics system and provide its own attitude estimate to the ground station computer for comparison with the air bearing's attitude estimate. This process could provide a means to test attitude estimation algorithms by using the

SPHERES' attitude estimate as a truth measurement since the SPHERES' attitude estimation system has been verified to a certain degree of accuracy. The SPHERES satellite can also be programmed to provide known external torques that could simulate external disturbance torques on the system that must be accounted for by the air bearing's attitude control system. With software updates, the SPHERES satellite can be integrated into the air bearing's estimation and control algorithm using its serial communication expansion port. The SPHERES satellite could provide state measurements to the main Arduino for use in the EKF, and the main Arduino could assign control torques to the SPHERES' thrusters during each control cycle. Information could be sent back to the ground station either by the SPHERES' wireless communication system, the air bearing's wireless USB system, or both.

Currently, the air bearing structure is configured to support a SPHERES satellite with a mass simulator in place of the satellite when it is being used elsewhere. The ground station computer is configured to communicate with the SPHERES satellite independently of the air bearing. The SPHERES satellite can be reprogrammed by the ground station computer and the computer can receive data from the satellite during testing. The green external component frame has mounting brackets for five metrology beacons, though the beacons themselves must be shared by other SPHERES test facilities. The provided software for the MAR and AAR do not include the option to communicate with a SPHERES satellite via a serial communication port, though the MAR has sufficient flash memory available to incorporate SPHERES communication software in the future. Even without direct communication with the air bearing's avionics system and the lack of beacons on the external frame, the SPHERES satellite can still provide a reasonable attitude estimate using its rate gyros and linear accelerometers, which can be used to validate attitude estimation software running on the air bearing's avionics system.

For more information about the SPHERES ADCS testbed please reference Dr. Alvar Saenz-Otero's Ph.D. thesis [47]. In order to set up and operate the SPHERES testbed, please reference the SPHERES user's manual for configuring a computer for communication with SPHERES as well as maintaining and operating the hardware [32].

4.5 Testbed Assumptions and Limitations

The ADCS Testbed is a valuable tool capable of verifying the operation of physical ADCS components as well as validating an integrated ADCS system consisting of hardware components and software algorithms. However, the testbed is subject to the disturbances associated with operating within a laboratory. The air bearing simulates the reduced gravity and friction environment of space while operating in a one g environment at sea level. In order to simulate reduced gravity, the air bearing's center of mass must be collocated with its center of rotation. Though there are several methods for adjusting the center of mass as previously described, the two points will never be exactly aligned, and their separation distance defines the disturbance torque due to gravity on the system. Even with fine adjustment of the center of mass, gravity torque is the most significant disturbance torque affecting the air bearing during testing. The air bearing operates by floating on a cushion of compressed air, which reduces friction between the air bearing and its support column. Though reduced, friction is not eliminated, which accounts for a second disturbance torque on the vehicle. The third significant disturbance torque is damping due to Earth's atmosphere at sea level. The air bearing operates in an atmosphere many orders of magnitude more dense than the space environment, and the increased drag due to the sea level atmosphere will dampen air bearing motion much faster than a satellite with similar inertial properties operating in space.

The air bearing is also subject to low amplitude, high frequency vibration caused by the high pressure air floating the vehicle above the support column. The air pressure must be high enough to float the vehicle above the support column because physical contact with the column can damage the rotating hemisphere. However, if the air pressure is too high, it causes an oscillating effect where the high pressure air increases the vehicle's height above the support column to allow the pressurized air to escape, which causes the vehicle to be too high. The vehicle then falls back down and is caught by the compressed air and the process repeats. This up and down motion is low amplitude and cannot be seen, but the motion induces a vibration that can be felt throughout the vehicle. Many ADCS tests may not be affected by the vibration, but some precision pointing tests like optical control systems might require an isolation mount to mitigate this vibration. The vibration can be reduced by tuning the air pressure based on the mass of the vehicle. However, the test designer

must be careful not to reduce the pressure to the point where the air bearing contacts the support column.

One of the largest limitations to operating in the laboratory environment is that the air bearing is not capable of providing full rotation about all three orthogonal axes. The air bearing can only rotate a total of thirty-eight degrees in the FIR x and y axes. This limitation means that full ADCS CONOPs tests in one test scenario are usually not feasible. However, with clever test design, almost any ADCS maneuver can be tested using several test scenarios in succession.

The testbed's avionics system is capable of testing almost any ADCS software algorithm planned for use on small satellites. However, due to the specific requirements of the Arduino processors as well as the satellite flight processors, copying complete code from the testbed for use on flight processors is unlikely to work. Some portions of the testbed software might be transferable with minor changes, but any copying of code must be done carefully to prevent bugs arising in the flight software.

The ADCS testbed is subject to limitations like increased disturbance torques, constrained rotational motion, and software compatibility. Though these limitations affect the functionality of the testbed, as long as they are accounted for, the air bearing can be used as a powerful tool for verifying and validating ADCS equipment.

Chapter 5

ADCS Testbed Analysis

The ADCS Testbed will be a valuable tool for both satellite developers and control theory students. However, the combined simulation and rotational air bearing testbed must first be tested itself to verify that the system operates as expected. Only after testbed component verification and system validation can the results from the testbed be used to verify and validate the operation of other ADCS components and algorithms. This chapter will focus on characterization of the ADCS components used in the baseline air bearing configuration as well as testing the integrated air bearing system. Once validated, the air bearing will be used to test an ADCS CONOPs scenario for the MicroMAS cubesat being developed within the Space Systems Laboratory.

5.1 Sensor Characterization

The first set of components to be tested on the air bearing are the attitude determination sensors. The rate gyroscopes and linear accelerometers within the IMU and the three axis magnetometer will be tested to determine their noise characteristics as well as any bias in their respective measurements due to the sensors themselves or external sources. This information will be used to improve the physical air bearing's attitude estimator and create an accurate model of the sensor in the air bearing simulation.

5.1.1 IMU Rate Gyroscopes

The three angular rate gyroscopes provide a measurement of the air bearing's angular rate in degrees per second once per control cycle. The orthogonal sensors provide a rate mea-

surement about each of the ABBF axes. The Analog Devices ADIS 16365 IMU's rate gyroscopes do not provide a perfect measurement of the air bearing's angular rates. Error is introduced to the measurement in several ways. First, random noise in the measurement causes the measured rate to vary from the true rate by some random value at each measurement. Second, offset bias causes the measured rate to differ from the true rate by some constant value at each measurement. Though noise cannot be removed, it should be accurately characterized to ensure the attitude estimation algorithm accounts for uncertainty in the rate gyroscope measurements. The offset bias can be measured and removed as long as it does not change with time.

In order to determine the noise and offset bias of the IMU, measurements are recorded from the device at twenty Hertz for a two hour period. During this time, the air bearing is fixed to the platform forcing it to remain fixed in the inertial reference frame. By fixing the air bearing, its angular rate is known to be zero in the FIR frame. Therefore, in a true FIR frame, the rate gyroscope measurements should consist only of noise and offset bias. However, the air bearing's FIR frame is fixed in the laboratory, which is rotating at the same rate as Earth's rotation. Even though the rate gyroscopes are fixed with respect to the FIR frame, their rate measurements should also include Earth's rotation in addition to noise and offset bias. Table 5.1 gives the average angular rates in degrees per second for each of the ABBF axes for the two hour test. Table 5.1 also gives the variance of the noise, which should not be affected by Earth's rotation because Earth's rotation should be constant in the air bearing's FIR frame.

Table 5.1: Average Rate and Variance of Gyroscope Measurements in Two Hour Fixed Test

Axis	Average Rate (deg)	Variance (deg^2)
x_{body}	0.0486	0.0556
y_{body}	0.1246	0.0605
z_{body}	0.1312	0.0835

Earth's rotation rate is 0.00417 deg/sec about its spin axis. Accounting for the laboratory's angular orientation with respect to Earth's spin axis, Earth's rotation rate in the air bearing's FIR frame is given in Equation 5.1.

$$\omega_{Earth} = \begin{bmatrix} 0.002083 \\ 0.003608 \\ 0 \end{bmatrix} \frac{deg}{sec} \tag{5.1}$$

The magnitude of each of the rate gyroscopes' offset biases (average rates in Table 5.1) is significantly higher than Earth's rotation rate, which suggests that the offset biases are not primarily caused by measuring Earth's rotation rate, but are due to inaccuracies in the gyroscopes themselves. In order to mitigate these biases, they must be subtracted from the measurements provided by the gyroscopes in order to gain the most accurate rate estimate. Subtracting the full offset biases ignores the components due to Earth's rotation, but these components are small relative to the offset biases, and they will have little impact during a test scenario. The adjusted rate measurements are provided to the attitude estimation algorithm.

5.1.2 IMU Accelerometers

The three linear accelerometers provide a measurement of the air bearing's linear acceleration in units of Earth's gravity *g's* which is equal to 9.81 m/s^2. Just like the rate gyroscopes, the three linear accelerometers are orthogonally mounted with one accelerometer in each of the ABBF axes. The accelerometers are also included in the ADIS 16365 IMU. On the air bearing testbed, the linear accelerometers are used to measure the gravity vector, which is used by the Extended Kalman Filter to determine an attitude estimate. Measurements from the linear accelerometers are subject to noise and offset bias similar to other sensors, but the linear accelerometers' measurements are dominated by discretization rather than noise or offset. Any noise or offset in the linear accelerometer's measurements is less than 0.01 *g's*, which is the smallest unit of measurement recorded from the linear accelerometers. This discretization is due to the way the measurement information is stored. Over the course of the two hour fixed air bearing test, the linear accelerometers measured the gravity vector in the ABBF z axis to be -0.99 to -1.00 *g's*, which is either exactly right or one discretization step off.

Measurement discretization has its own limitations regardless of noise or offset bias. Because the linear accelerometer measurements are discretized to relatively high step sizes, the attitude estimate found using the linear accelerometer measurements is limited. For

instance, the smallest angle that can be measured between the commanded and measured gravity vector is 0.58 degrees. Equation 5.2 restates Equation 3.80, which is the method used to find angular orientation error. Equation 5.2 is evaluated using $g_{cmd} = \begin{bmatrix} 0 & 0 & -1.00 \end{bmatrix}$ and $g_{meas} = \begin{bmatrix} 0 & -0.01 & -0.99 \end{bmatrix}$ to represent one linear accelerometer measurement step.

$$\bar{\theta}_{acc\,error} = \frac{g_{meas} \times g_{cmd}}{|g_{meas}||g_{cmd}|} = \frac{\begin{bmatrix} 0.01 & 0 & 0 \end{bmatrix}}{0.99 * 1.00} = \begin{bmatrix} 0.0101 & 0 & 0 \end{bmatrix} rad = \begin{bmatrix} 0.58 & 0 & 0 \end{bmatrix} deg \quad (5.2)$$

Another limiting factor in using the linear accelerometers to measure the gravity vector is centripetal acceleration. The linear accelerometers measure all accelerations, which include gravity and centripetal acceleration due to rotational motion. Centripetal acceleration contributes acceleration components that are not expected by the attitude estimation algorithm. The magnitude of centripetal acceleration is a function of the sensor's distance away from the center of rotation and the rate of rotation. Figure 5-1 shows the IMU's distance away from the center of rotation assuming rotation about the ABBF z axis, which is the only axis that could sustain high angular rates.

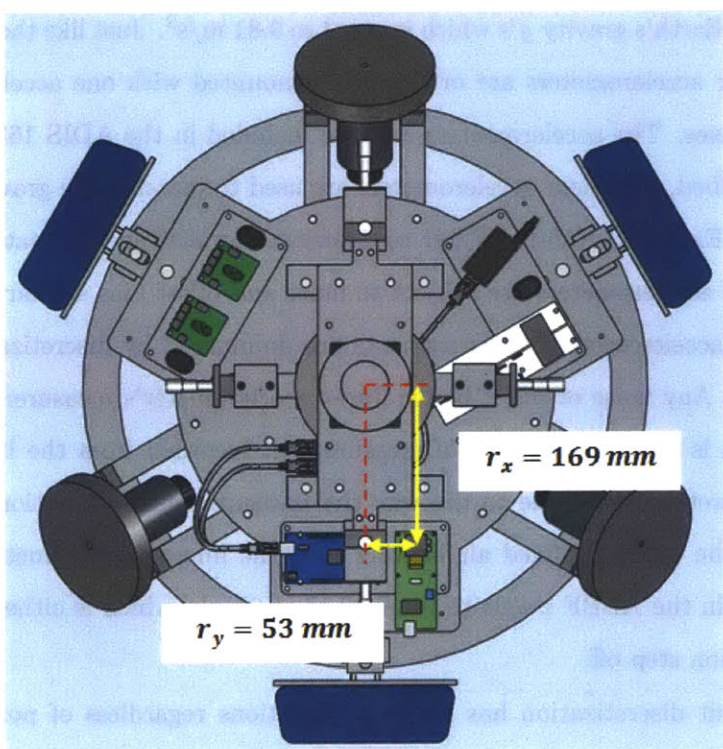


Figure 5-1: IMU Mounting Location

Equation 5.3 gives the formula for finding centripetal acceleration a_c using radial distance and angular rate [54].

$$a_c = \omega^2 r \tag{5.3}$$

The effects of centripetal acceleration will be minimal if the acceleration is less than what can be measured by the linear accelerometers, which is known to be 0.01 *g's* or 0.0981 m/s^2. The radial distance in the ABBF x axis is larger than the distance in the ABBF y axis, which means that the ABBF x axis linear accelerometer will experience greater centripetal acceleration for a given angular rate. Solving Equation 5.3 for ω assuming $a_c = 0.0981$ m/s^2 and $r = 169$ mm gives an angular rate of 0.762 rad/sec (43.7 deg/sec) in order to produce 0.01 *g's* of centripetal acceleration or one step in the ABBF x axis linear accelerometer's measurement. This rate is higher than the maximum rate achievable using the current set of attitude control actuators. Therefore, centripetal acceleration should not cause significant error in the the measurements produced by the linear accelerometers.

5.1.3 Magnetometer

The three axis magnetometer provides a measurement of the magnetic field vector in micro-Tesla μT. The measured magnetic field vector is used by the Extended Kalman Filter to determine an attitude estimation similar to the way the gravity vector provided by the linear accelerometers is used by the EKF. Just like other sensors, the magnetometer measurements are subject to noise and offset bias. In addition to these error factors, the magnetic field within the laboratory is not uniform as it is for a given location in space. In order to create a relatively constant magnetic field vector to be measured by the magnetometer, the EMFG is used to generate a strong magnetic field in the air bearing's FIR y axis as shown below in Figure 5-2. When in the zero attitude position, the magnetometer is mounted such that the magnetic field vector created by the EMFG passes directly through the magnetometer. Rotation about the ABBF z axis will not induce translational motion of the magnetometer, which means the magnetic field vector at that location should not change. However, rotation about the ABBF x and y axes will cause translational motion of the magnetometer and will lead to some change in the magnetic field vector and therefore some attitude estimation error.

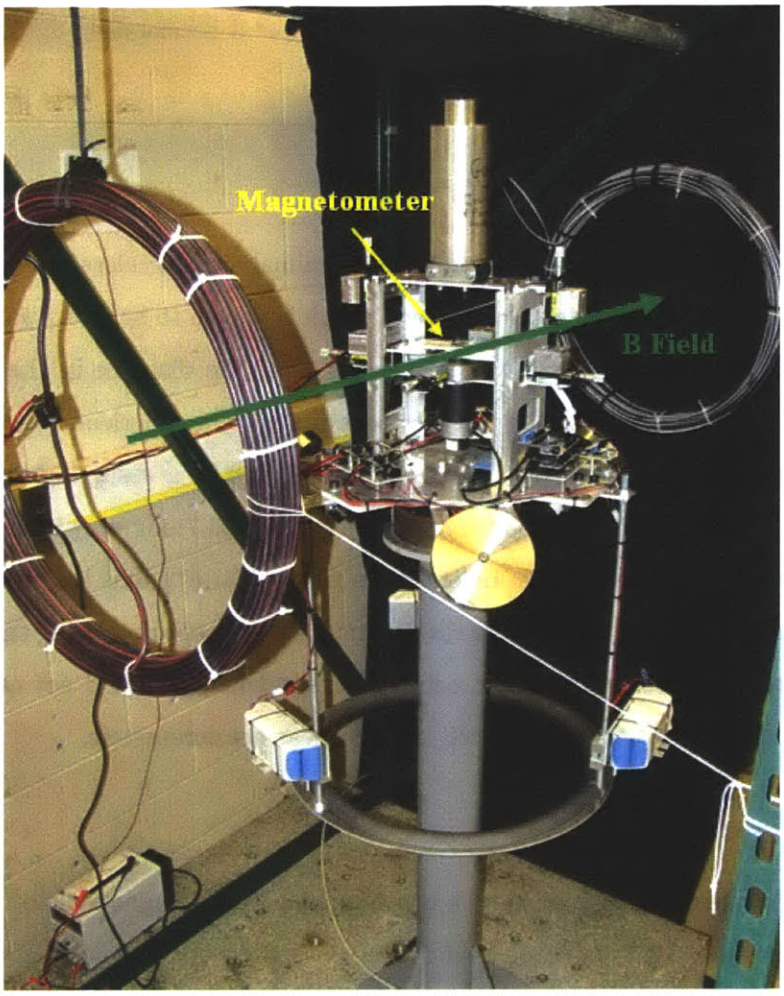


Figure 5-2: Magnetometer and EMFG Field Vector

In order to characterize noise, measurements from the magnetometer are taken at twenty Hertz for a two hour period with the EMFG off. During this period, the air bearing is fixed in the inertial reference frame. Therefore, any change in the magnetic field measurement is due either to noise or a change in the ambient magnetic field within the lab. Offset bias cannot be determined during this test because the direction of the total ambient magnetic field is unknown. Figure 5-3 gives the plot of magnetic field measurements over the two hour period. As can be seen in the figure, the ambient field does not change, which means that any high frequency deviations in the measurements are due to sensor noise. Table 5.2 gives the noise variance for each of the three orthogonal magnetic field sensors on the magnetometer. The magnitude of the ambient field shown in Figure 5-3 is approximately 85 μT, which is stronger than Earth's average magnetic field on the surface, which is approximately 50 μT.

The additional field is likely due to electronic equipment in the laboratory, which distorts the local magnetic field.

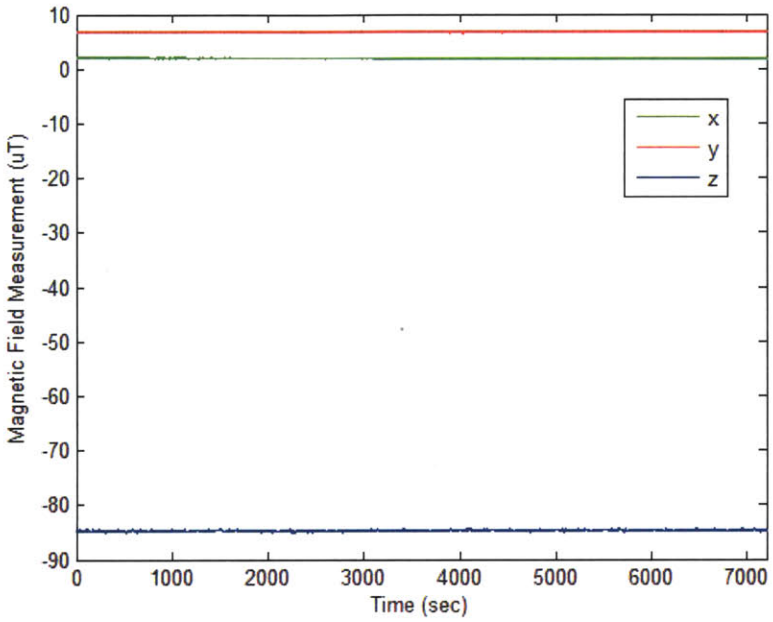


Figure 5-3: Magnetometer Measurements of Ambient Magnetic Field - EMFG Off

Table 5.2: Variance of Three Axis Magnetometer Measurements in Two Hour Fixed Test

Axis	Variance (μT^2)
x_{body}	0.0096
y_{body}	0.0025
z_{body}	0.0100

Offset bias must now be characterized in order for the magnetometer to produce a useful measurement. Offset bias can be caused either by differences in the three sensors on the magnetometer or external magnetic fields that are created by and fixed to the air bearing. In order to characterize these biases, the air bearing is commanded to complete ten thirty second rotations about the ABBF z axis with the EMFG powered on. Measurements from the magnetometer are recorded during this period and Figure 5-4 gives the results from this test.

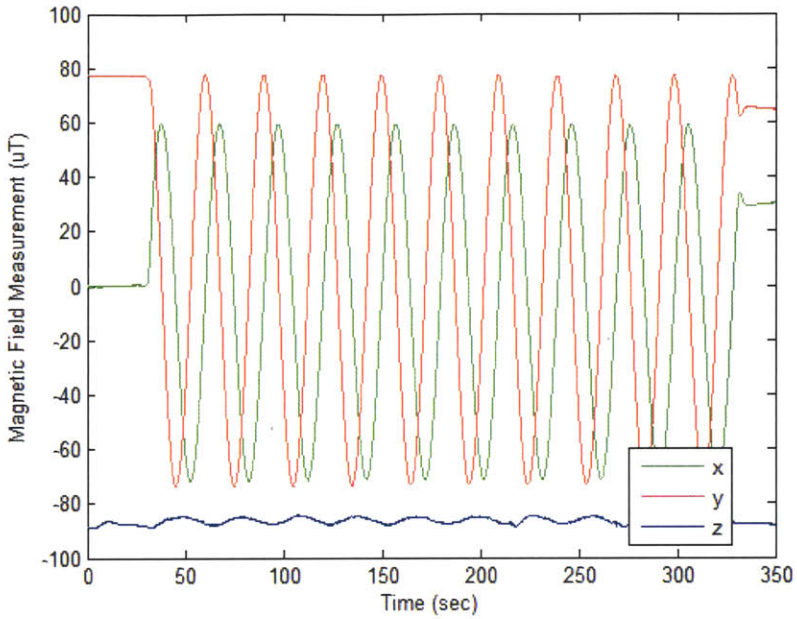


Figure 5-4: Magnetometer Measurements over Ten ABBF Z Axis Rotations - EMFG On, No Gains

The ABBF z axis magnetic field measurement has little change, which is expected since the air bearing is rotating about the ABBF z axis. The ABBF y axis magnetic field measurement is initially much larger than the ambient field ABBF y axis measurement because the ABBF y axis sensor is measuring the majority of the EMFG's generated magnetic field vector when the air bearing is in its initial position. The ABBF x axis sensor initially measures near zero, which is approximately the same as the ambient field ABBF x axis measurement. Once the air bearing begins its ABBF z axis rotation, the EMFG's field rotates in the ABBF frame. The ABBF y axis measurement should have the same magnitude in the positive and negative direction as it completes a rotation, and the ABBF x axis measurement should also have the same magnitude as the ABBF y axis measurement with a period ninety degrees out of phase with the ABBF y axis measurement. The phase shift in Figure 5-4 is correct, but the magnitudes of both the ABBF x and y axis measurements are incorrect. Gains are added to the positive and negative measurements of both the ABBF x and y axis magnetometers in order to correctly scale the measurements. The positive ABBF y axis magnetometer measurement is chosen as the baseline and assigned a gain of one. The four magnetometer gains are listed in Table 5.3.

Also listed in Table 5.3 is the ABBF z axis magnetometer bias value for the test. As seen in Figure 5-4, the ABBF z axis has a large magnetic field component that is not due to the EMFG's generated magnetic field vector. This component cannot be easily characterized like the ABBF x and y axis components because the air bearing cannot complete full rotations about the ABBF x or y axes. In order to mitigate error due to the ABBF z axis component, it is simply removed from the magnetic field measurement using a bias, which is calculated at the beginning of each test based on the current ABBF z axis magnetic field measurement value. The resulting magnetic field measurement provided by the magnetometer represents the sum of the large EMFG generated magnetic field vector in the FIR y axis and the small ambient field components, which have little effect on the magnetic field measurement.

Table 5.3: Magnetometer ABBF X, Y Axis Gains and Z Axis Bias

Axis	Gains/Bias
x_{body} Positive Gain	1.298
x_{body} Negative Gain	1.088
y_{body} Positive Gain	1.000
y_{body} Negative Gain	1.060
z_{body} Bias	-86.5 μT

The ABBF x and y axis gains and ABBF z axis bias are applied to the magnetometer measurements, and the ten rotation test is repeated. Figure 5-5 gives the adjusted magnetometer measurements resulting from the second test. As seen in this figure, the ABBF z axis measurement remains near zero with some oscillation due to attitude error. The ABBF x and y axis measurements oscillate between the same amplitude with the ABBF x axis measurement ninety degrees out of phase with the ABBF y axis measurement. This is the expected result of rotating the air bearing perpendicular to the magnetic field vector generated by the EMFG. The adjusted magnetic field measurements from the magnetometer are sent to the EKF to help determine an attitude estimate for the air bearing.

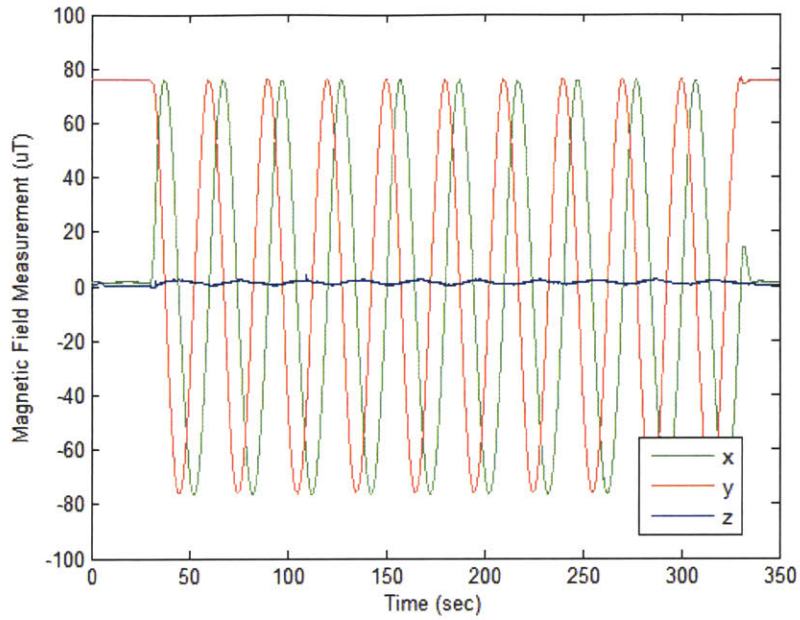


Figure 5-5: Magnetometer Measurements over Ten ABBF Z Axis Rotations - EMFG On, Gains Applied

5.2 Reaction Wheel Characterization

The air bearing's reaction wheels provide three axis attitude control by accelerating in the positive or negative directions as commanded. In order to provide accurate attitude control, the reaction wheels must react to voltage inputs as expected. In Section 3.2.3, a state space model of the reaction wheels is developed for use in the attitude estimation and control algorithm. The reaction wheel state space model must predict the physical response of the reaction wheels accurately enough to develop useful attitude estimation and control algorithms based on the model.

The first reaction wheel characterization test will be to determine how well the physical and simulated reaction wheels respond to a step input in the positive direction followed by a step input in the negative direction. Figure 5-6 shows a commanded reaction wheel angular velocity of 194 rad/sec (maximum angular velocity) as well as the simulated reaction wheel response and the actual reaction wheel response for the x_{RW} axis reaction wheel. Though not exactly alike, the response of each of the four reaction wheels is similar. Therefore, only response plots for the x_{RW} axis reaction wheel will be shown.

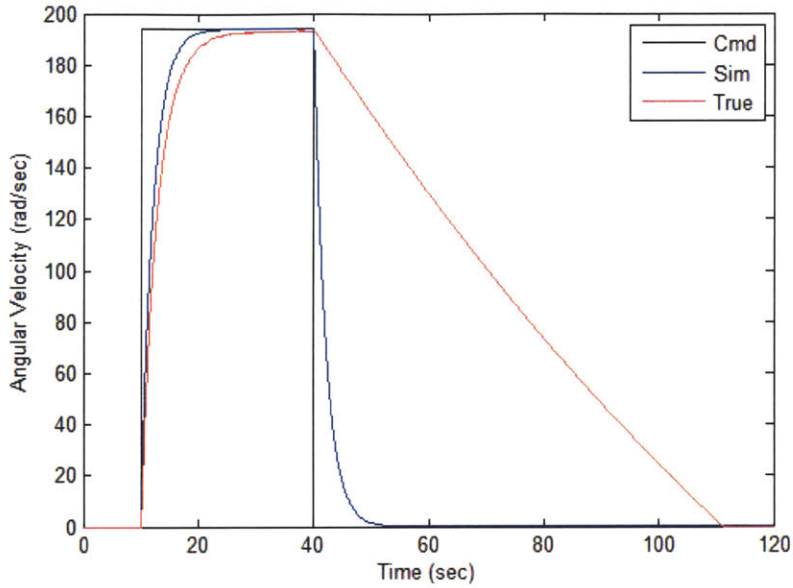


Figure 5-6: Maximum Step Input and Simulated/Actual Reaction Wheel Response - No Brake

The 10/90 rise time of the simulated response to the step input shown in Figure 5-6 is 4.61 seconds. The 10/90 rise time of the actual reaction wheel response to the same step input is 6.48 seconds, which is significantly longer. However, the actual response does not deviate from the simulated response until the reaction wheel nears its maximum velocity. The actual wheel appears to have additional damping coefficients that take effect as the wheel nears saturation. However, maximum step inputs to the actual reaction wheels should not be used during legitimate ADCS scenario tests. The steady state error of the actual reaction wheel is 0.52 percent. The most notable deviation of the actual reaction wheel response seen in Figure 5-6 is the slow return to zero angular velocity after a negative step input at forty seconds. As discussed in Section 3.2.5, this is due to the Pololu Trex motor controllers driving voltage across the reaction wheel motors rather than current through the motors. A braking algorithm is applied which uses reaction wheel rate feedback to determine when the reaction wheel velocity is five percent higher than its commanded velocity. If so, the reaction wheel is commanded to brake, which connects the positive and negative motor leads and quickly decelerates the motor until it is within five percent of its commanded value. Figure 5-7 shows the same commanded reaction wheel angular velocity shown in Figure 5-6. However, the reaction wheel responses have the braking algorithm applied.

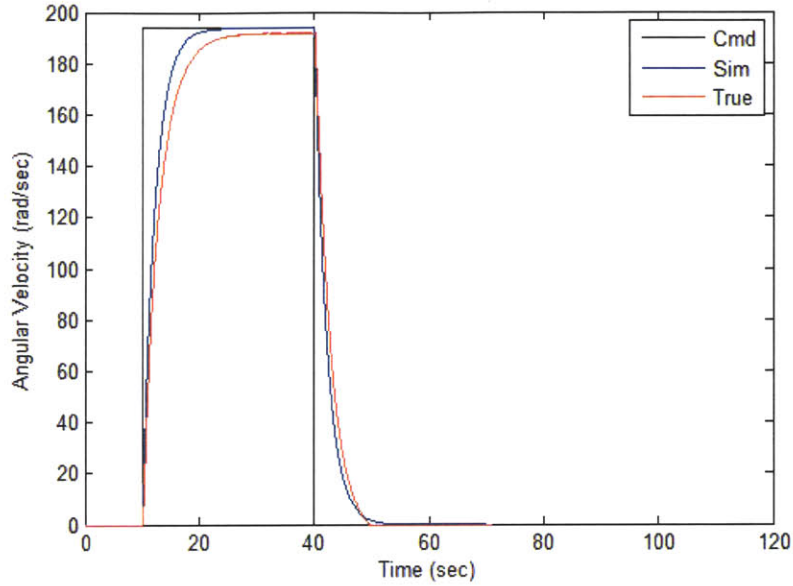


Figure 5-7: Maximum Step Input and Simulated/Actual Reaction Wheel Response - Braking Applied

The actual reaction wheel's response to a negative step input is significantly improved once the braking algorithm is applied. The 10/90 rise time of the braked reaction wheel response to the negative step input is 5.27 seconds, which is much less than the approximately fifty second 10/90 rise time of the unbraked reaction wheel response to the negative step input.

By differentiating the actual reaction wheel response to a maximum step input shown in Figure 5-7, the reaction wheel's torque profile can be found. Figure 5-8 shows the corresponding torque applied by the reaction wheel in response to a maximum step input. The peak torque is applied just as the reaction wheel receives the maximum step input. The reaction wheel's peak torque is found to be 0.545 Nm as it accelerates from zero and 0.491 Nm as it accelerates towards zero.

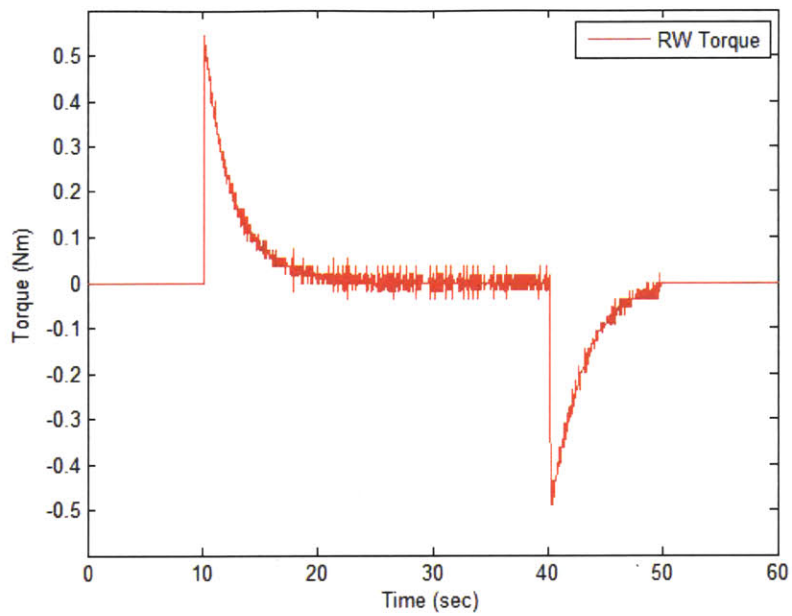


Figure 5-8: Reaction Wheel Torque

Figure 5-9 shows a commanded reaction wheel angular velocity of forty rad/sec as well as the simulated and actual reaction wheel responses. The actual response overshoots the commanded response in this case, but the closed loop braking algorithm helps to maintain the commanded angular velocity by braking the wheel for short periods to slow it down. Though the braking algorithm keeps the reaction wheel from exceeding five percent overshoot, the algorithm introduces jitter by oscillating the reaction wheel brake command. The jitter is clearly shown in Figure 5-9. Braking algorithm jitter may have an adverse affect on ADCS test scenarios. The need for a braking algorithm can be removed if regenerative motor controllers are used in place of the current Pololu Trex motor controllers. This topic will be discussed further in Section 6.2.1.

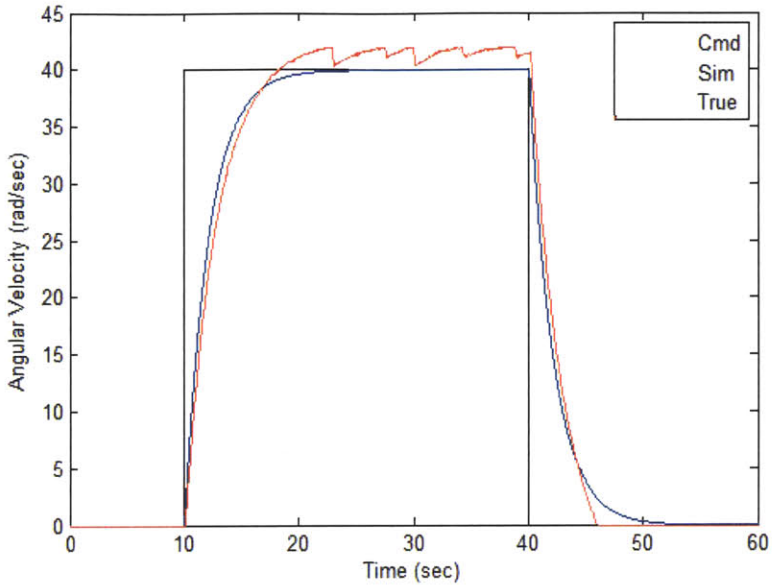


Figure 5-9: 40 Rad/Sec Step Input and Simulated/Actual Reaction Wheel Response - Braking Applied

To further compare the simulated reaction wheel model to the actual reaction wheel, the Bode gain and phase plots of the reaction wheel state space model are given below in Figure 5-10. Three points on the Bode plots are chosen for comparison with the actual wheel. The reaction wheel is driven by a sinusoidal input with an amplitude of 4.5 volts, which corresponds to sixty rad/sec. The three sinusoidal periods chosen for comparison are fifteen seconds, forty seconds, and 160 seconds. These three points are highlighted on the Bode plots below.

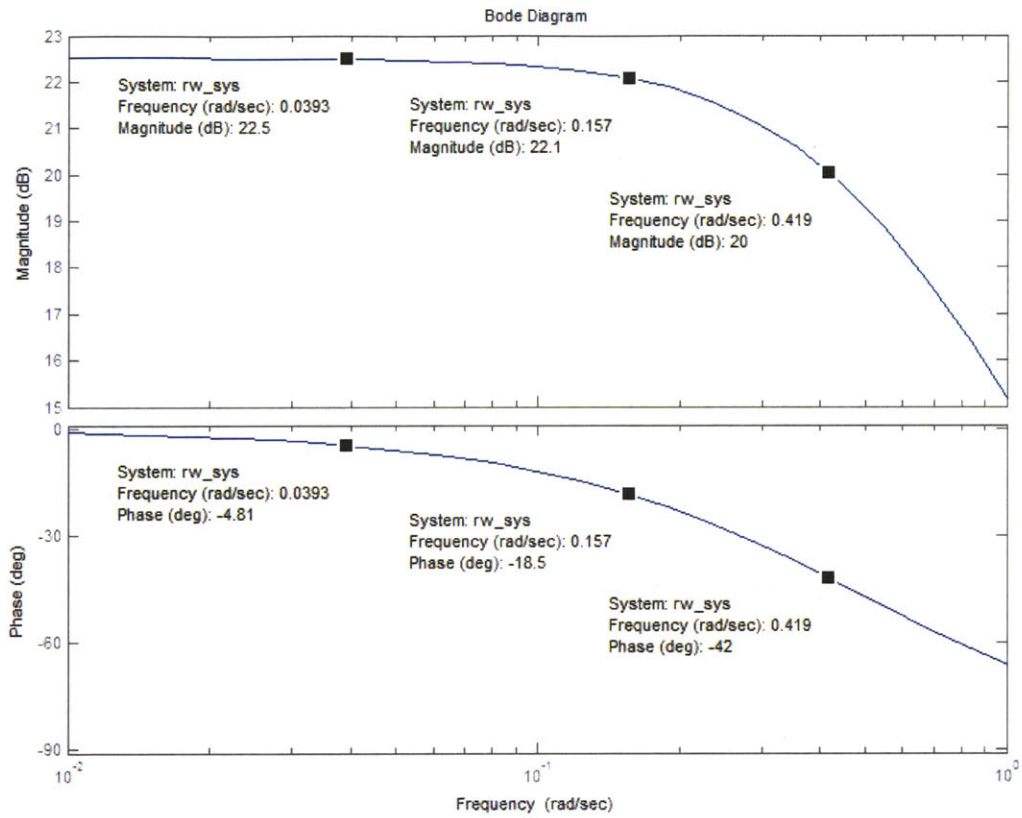
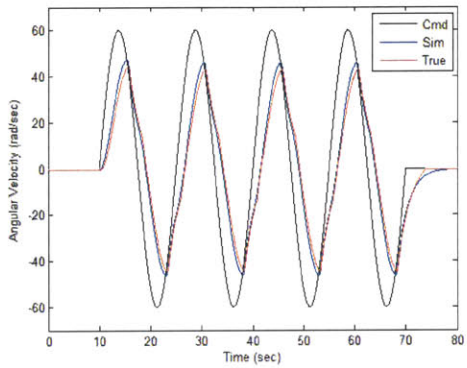


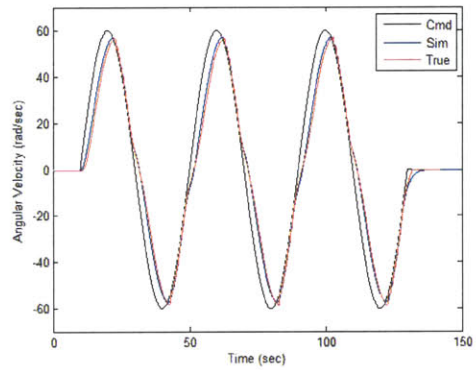
Figure 5-10: Reaction Wheel State Space Model Bode Plots

Figures 5-11(a), 5-11(b), and 5-11(c) show the sinusoidal inputs (in rad/sec) as well as the simulated and actual reaction wheel responses for the fifteen, forty, and 160 second period cases respectively. Table 5.4 gives the gain and phase of the simulated (from the Bode plots) and actual responses in all three cases. These values are found during the rate increase portion of the response when the braking algorithm is not in effect. During this portion, the gain and phase parameters of the actual response are similar to those from the Bode plots. However, during the rate decrease portion, the braking algorithm is applied, and the phase lag of the reaction wheel response is reduced to near zero as seen in the below figures. Though this effect does not match the Bode phase plot, the braking algorithm is applied in the simulation as well and drives the phase lag of the simulated response to near zero. Overall, the braking algorithm causes the reaction wheel to respond differently depending on whether it is accelerating away from or towards zero. However, the same

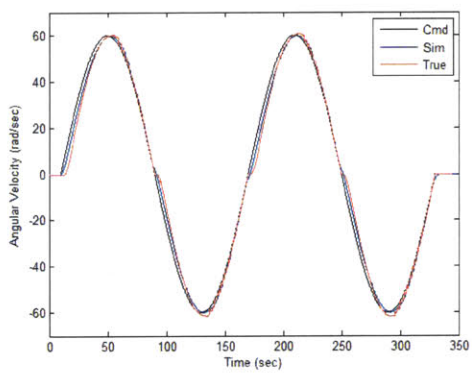
effect is accounted for in the reaction wheel model, which will help the user determine how the two-phase response of the reaction wheels affects the system.



(a) RW Response - 15 Second Period



(b) RW Response - 40 Second Period



(c) RW Response - 160 Second Period

Figure 5-11: Reaction Wheel Response to Sinusoidal Input

Table 5.4: Reaction Wheel Magnitude and Phase Results

Sine Period (sec)	Freq (rad/sec)	Model Mag (dB)	Actual Mag (dB)	Model Phase (deg)	Actual Phase (deg)
15	0.4189	20.0	19.76	-42.00	-51.47
40	0.1571	22.1	22.04	-18.50	-22.24
160	0.0393	22.5	22.57	-4.81	-5.87

5.3 Air Bearing Disturbance Characterization

The air bearing testbed provides a tool to simulate aspects of the space environment to include reduced friction and zero torque due to gravity. However, the air bearing is located within a laboratory environment and therefore subject to the disturbances and external effects caused by such an environment. These disturbances and effects must be clearly understood and characterized in order to produce useful results from ADCS test scenarios performed on the air bearing.

5.3.1 Center of Mass Manipulation

In order to reduce torque on the air bearing due to gravity and produce an environment where the air bearing's attitude control actuators can operate as if in a space environment, the air bearing's center of mass must be co-aligned with the center of rotation. The air bearing has several mass manipulation options for coarse and fine CM adjustment. These options are discussed at length in Section 4.1.1. This section will focus on how to use these devices to place the CM as close to the center of rotation as possible.

For coarse CM adjustment, the battery packs can be adjusted in the vertical direction, and trim masses can be added to the air bearing. The battery packs should only be adjusted to compensate for adding or removing large components on the top of the air bearing. They are difficult to adjust, their adjustment results in large CM changes, and they only move the CM in the vertical direction. If smaller components are added to or taken away from the air bearing, small trim masses can be added for medium CM adjustment.

While adjusting the batteries or adding trim masses, the CM should be placed below the center of rotation and slowly moved up to co-alignment with the center of rotation. To ensure the CM is below the center of rotation, the air bearing should be manually rotated to its maximum pitch angle and released with all air bearing electronics turned off. If the air bearing oscillates in a pendulum fashion, the CM is below the center of rotation. If the air bearing remains against the support column regardless of which portion of the safety ring is in contact with the support column, the CM is above the center of rotation. In this case, coarse adjustments with the battery packs should be made to lower the CM until it is just below the center of rotation.

Once the CM is placed below the center of rotation using the battery packs, and slowly moved up using trim masses, fine CM adjustment can be made using the linear slide CM adjusters. As previously discussed in Section 4.1.1, the CM adjusters can only move the CM 0.189 mm in each axis, so the battery packs and trim masses must be used to place the CM within 0.189 mm of the center of rotation. The air bearing's estimation and control algorithm is used to determine how much to move the CM adjusters. The method works by measuring the torque required by the reaction wheels to hold the air bearing in the zero attitude orientation and using this information to figure out how gravity is affecting the system.

Equation 5.4 shows the simplified vector equation of motion that will form the basis of the fine CM adjustment process. From this equation, it is clear that the reaction wheels must accelerate ($\dot{\bar{\Omega}}$) in order to counter the effects of external torque due to gravity ($\bar{T}_{ext\,g}$).

$$\bar{T}_{ext\,g} = DI_{RW}\dot{\bar{\Omega}} \tag{5.4}$$

Equation 5.5 gives the components that make up torque due to gravity [52]. These components are the force due to gravity vector itself ($m_{AB}\bar{g}$) and the position vector from the center of rotation to the CM ($\bar{r}$).

$$\bar{T}_{ext\,g} = \bar{r} \times m_{AB}\bar{g} \tag{5.5}$$

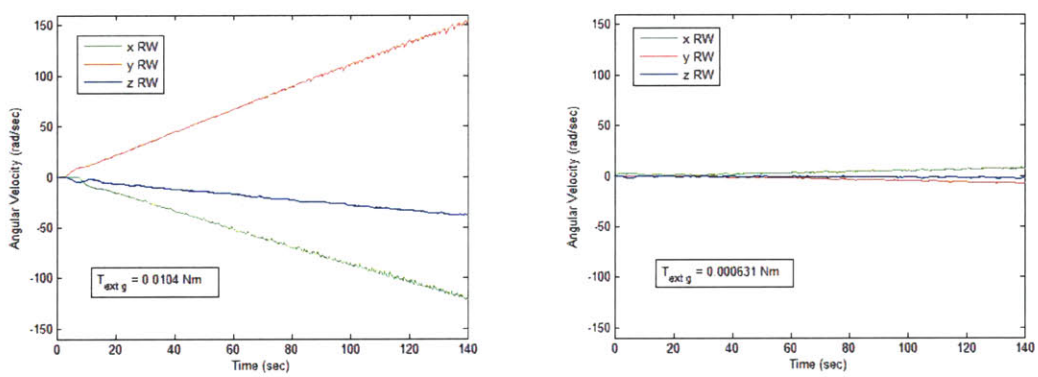
Though it is not possible to determine the complete position vector ($\bar{r}$) by knowing the force due to gravity vector ($m_{AB}\bar{g}$) and the torque due to gravity ($\bar{T}_{ext\,g}$), the projection of the position vector perpendicular to the force vector ($\bar{r}_p$) can be found using Equation 5.6. The division by the force vector magnitude squared is required to correctly convert units of torque and force to distance.

$$\bar{r}_p = \frac{m_{AB}\bar{g} \times \bar{T}_{ext\,g}}{|m_{AB}\bar{g}|^2} \tag{5.6}$$

By substituting Equation 5.4 into Equation 5.6, an equation relating the position vector (perpendicular to the force due to gravity vector) to the angular acceleration of the reaction wheels can be found. Equation 5.7 gives this relationship.

$$\bar{r}_p = \frac{m_{AB}\bar{g} \times DI_{RW}\dot{\bar{\Omega}}}{|m_{AB}\bar{g}|^2} \tag{5.7}$$

By commanding the air bearing to maintain zero attitude orientation and measuring the acceleration of the reaction wheels over time, the distance (perpendicular to gravity) between the center of rotation and the CM can be found using Equation 5.7, and the CM adjusters can be moved to relocate the CM. The following is an example of the CM adjustment process. Figure 5-12(a) shows the angular velocity of the reaction wheels over time required to maintain the air bearing at zero attitude. By applying a linear fit to the rates and measuring the slope of the linear equation, the constant angular acceleration for each reaction wheel is found and used to determine the torque due to gravity ($\bar{T}_{ext\,g}$), the magnitude of which is 0.0102 Nm in the first case.



(a) RW Velocity Before CM Adjustment

(b) RW Velocity After CM Adjustment

Figure 5-12: Reaction Wheel Angular Velocity Before and After CM Adjustment

The CM adjustment process may require several iterations because there is some unknown error in the estimates of the air bearing's mass and the reaction wheels' moments of inertia. Figure 5-12(b) shows the angular velocity of the reaction wheels over time required to maintain the air bearing at zero attitude after two iterations of the CM adjustment process. The figures have the same scale, which clearly shows that much less control torque is required by the reaction wheels to maintain the air bearing at zero attitude. The estimated torque due to gravity after two CM adjustment iterations is 6.31e-4 Nm, which is a reduction of 93.9% of the torque due to gravity before the CM adjustment process.

5.3.2 Compressed Air Vibration

The air bearing's support mechanism introduces oscillation into the system (See Section 4.5 for more details). If the pounds per square inch (PSI) of the pressurized air is too high relative to the air bearing's mass, the compressed air causes a high frequency, low amplitude oscillation in the air bearing's FIR z axis. If the PSI is too low, the air bearing may be damaged due to contact between the hemisphere and the support bowl. Therefore, the PSI should be adjusted to reduce the induced oscillation but avoid contact between the hemisphere and support bowl.

The pressurized air oscillation is characterized during a test of the ExoPlanetSat cubesat optical system, which is mounted on top of the air bearing and directed towards a simulated star mounted on the wall of the laboratory. The air bearing is floating with an air pressure of fifty PSI (normal operating pressure), but the control algorithm is turned off during the test. The air bearing is held at zero attitude and angular rate by a data cable connecting an external computer to the ExoPlanetSat optical system mounted on the air bearing [44].

Figure 5-13 shows the results of the ExoPlanetSat optical system test. The optical system measures attitude error in the plane perpendicular to the position vector from the optical system to the star. The labeled y axis in Figure 5-13 (green) is equivalent to the FIR z axis (vertical axis) in the air bearing FIR frame. The high frequency oscillation in the figure's y axis represents the oscillation due to the compressed air floating mechanism. Based on ExoPlanetSat's optical system's measurements, the oscillation has an amplitude of approximately forty arcseconds. The frequency is not known because it is higher than the data rate of ExoPlanetSat's optical system. The high frequency noise in the figure's x axis (blue) represents oscillation induced by the compressed air in the air bearing FIR y axis (horizontal axis). The amplitude of this oscillation is approximately ten arcseconds. The vertical axis experiences the larger amplitude because it is aligned with the air bearing's support mechanism. The low frequency (approximately 0.05 Hertz), sixty arcsecond amplitude oscillation in the figure's x axis is due to natural rotation of the air bearing. The data cable stabilizes the air bearing but is not rigid enough to remove all rotational motion [44].

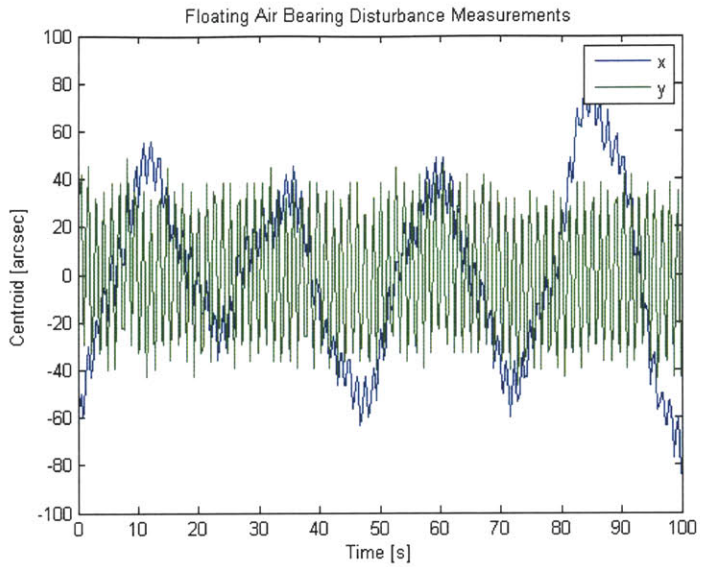


Figure 5-13: Air Bearing Oscillation Due to Compressed Air Support Mechanism [44]

5.3.3 Air Bearing Friction

The air bearing is used to simulate the near frictionless environment of space, but the air bearing is affected by friction due to the laboratory environment. There is some amount of friction due to the compressed air support mechanism as well as the vehicle's interaction with Earth's atmosphere at sea level. These two forces act to dampen the air bearing's angular rate and will eventually return the air bearing to zero angular rate if no control torques are applied.

To characterize the disturbance torque due to friction on the air bearing, the air bearing is made pendulum stable and the vertically mounted reaction wheel is used to induce an angular rate of 16.3 deg/sec about the ABBF z axis. The control law is turned off during this test so that there are no internal torques acting on the air bearing. Because the air bearing is pendulum stable, the air bearing's angular rate remains only in the ABBF z axis and does not precess to another axis. Angular rate measurements are recorded from the IMU at twenty Hertz throughout the test. Figure 5-14 gives the air bearing's angular rate profile over a three hour period. The air bearing's angular rate reduces from 16.3 deg/sec to 2.9 deg/sec after three hours. Because there are no control torques applied during this test, the reduction in angular rate is due to external disturbance torques acting on the system.

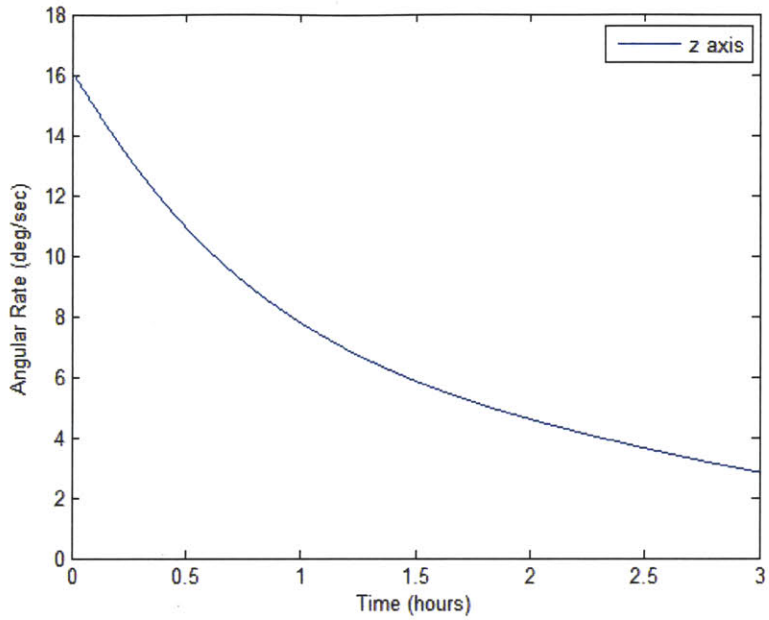


Figure 5-14: Uncontrolled Air Bearing Angular Rate

Torque is found by taking the derivative of the angular rate (angular acceleration) and multiplying it by the air bearing's moment of inertia about the ABBF z axis (I_{zz} = 2.32 kgm^2). Figure 5-15 gives the disturbance torque profile acting on the air bearing during the three hour period. The disturbance torque appears to be a function of angular rate since the torque reduces as the angular rate reduces. At the air bearing's maximum angular rate of 16.3 deg/sec, the magnitude of the disturbance torque is highest at 1.45e-4 Nm. After three hours, the air bearing's angular rate is 2.9 deg/sec and the magnitude of the disturbance torque has reduced to 1.60e-5 Nm.

Based on these results, the disturbance torque due to air bearing friction and atmospheric interaction is one to two orders of magnitude below the disturbance torque due to gravity even with a carefully balanced air bearing. If an ADCS test scenario is required to have extremely small disturbance torques at the cost of reduced degrees of rotational freedom, the air bearing can be made pendulum stable which will allow for only one degree of rotational freedom about the ABBF z axis. A disturbance torque profile similar to Figure 5-15 can be expected in this configuration.

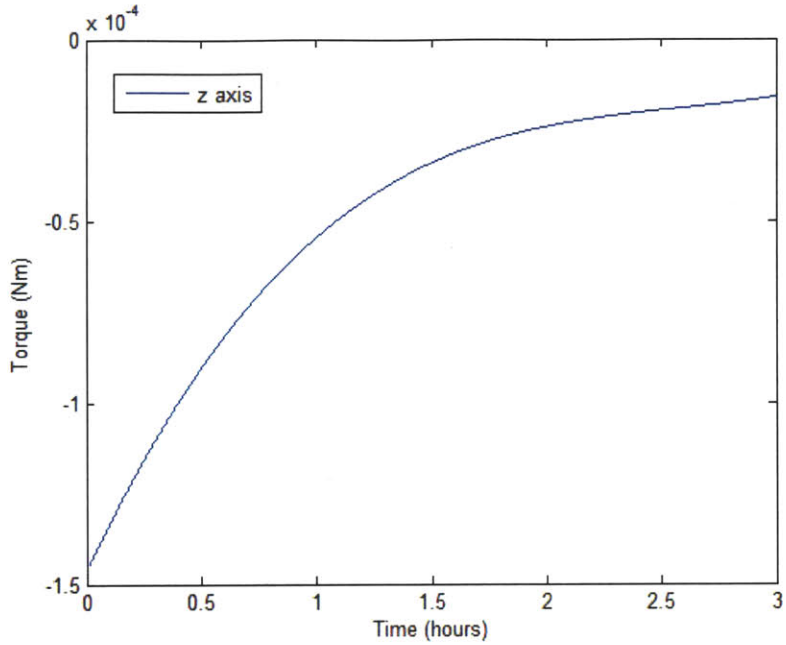


Figure 5-15: Friction Disturbance Torque acting on Air Bearing

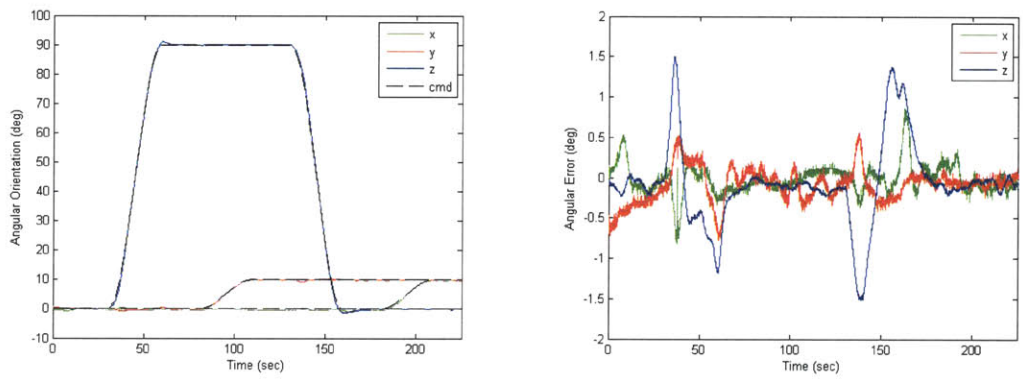
5.4 Integrated Air Bearing and Simulation Characterization

To characterize the behavior of the integrated ADCS testbed, the air bearing and simulation will be commanded to track an attitude profile (first shown and discussed in Figure 3-20 of Section 3.2.4) that requires commanded angular rate in all three ABBF axes. In three separate test cases, the air bearing and simulation will be commanded to track the same attitude profile with three separate sets of initial conditions. In all three cases, the estimation and control algorithm included in the provided software will be used and operated at the default twenty Hertz cycle. In the first test case, the air bearing will begin with zero stored angular momentum. In the second case, the air bearing will begin the test with a nonzero angular momentum vector in the FIR x axis. In the third case, the air bearing will begin the test with the same angular momentum vector as case two; however, the feedforward portion of the air bearing's control law will be disabled.

5.4.1 Case One - Zero Angular Momentum System

In the first test case, the air bearing will begin with zero stored angular momentum. The air bearing itself and all reaction wheels are initially at rest with the ABBF frame aligned

with the FIR frame. Figure 5-16(a) shows the commanded angular orientation profile (black dotted lines) as well as the air bearing's angular orientation as determined by the EKF. The commanded orientation requires a ninety degree rotation about the unconstrained ABBF z axis as well as ten degree rotations about the constrained ABBF x and y axes. Figure 5-16(b) shows the attitude error in degrees between the commanded and EKF estimated angular orientations. Maximum error of approximately 1.5 degrees occurs as the air bearing lags behind and then overshoots the large ABBF z axis commanded rotation.

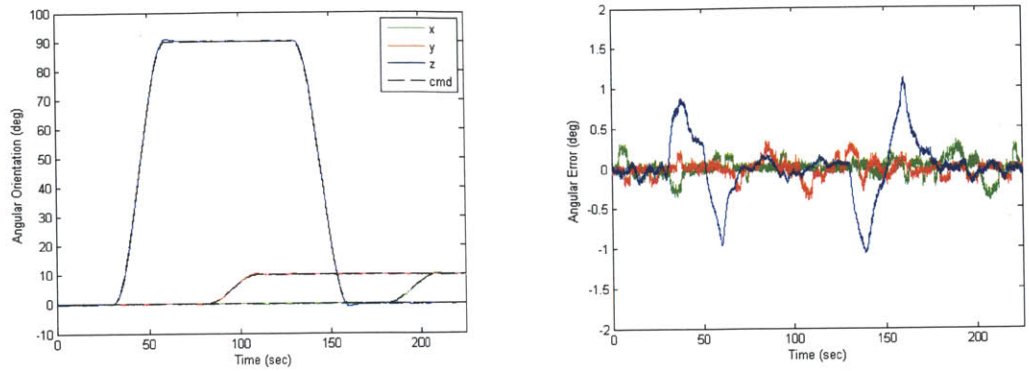


(a) Air Bearing Command and Actual Response

(b) Actual Angular Orientation Error

Figure 5-16: Actual Angular Orientation and Error - Zero Initial Angular Momentum

In order to verify the results of the MATLAB model, the simulation is given the same angular orientation command. Figure 5-17(a) shows the simulated air bearing response to the commanded angular orientation profile, and Figure 5-17(b) shows the simulated error between the commanded and estimated angular orientation. The simulated angular error expects ABBF z axis lag of only about one degree rather than the actual 1.5 degrees shown in Figure 5-16(b). The larger angular error in the actual air bearing is likely due to reaction wheel dead-band as the wheels start from zero angular velocity. Though motor lag is simulated in MATLAB, dead-band is not. Also, the initial angular error in the actual response (Figure 5-16(a)) is due to the air bearing being released in a slightly incorrect orientation. Otherwise, the actual and simulated error have similar high frequency noise characteristics and low frequency air bearing motion characteristics.

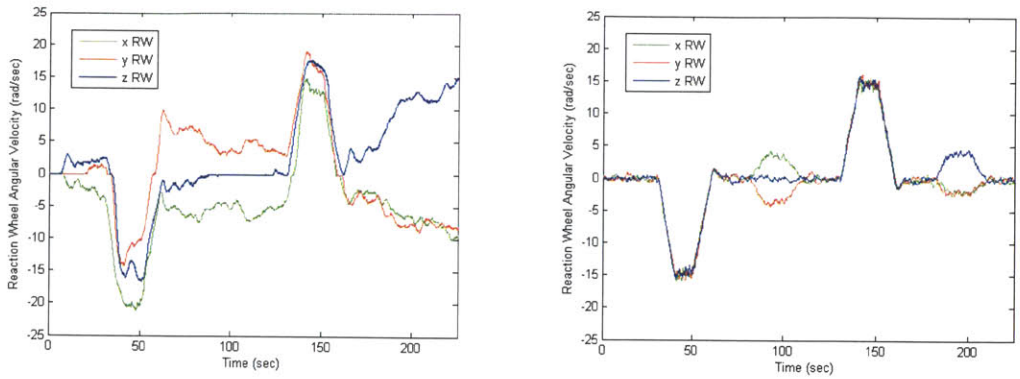


(a) Simulated Air Bearing Command and Simulated Response

(b) Simulated Angular Orientation Error

Figure 5-17: Simulated Angular Orientation and Error - Zero Initial Angular Momentum

Also important to evaluate are the required reaction wheel angular velocities necessary to maneuver the air bearing as commanded. Figure 5-18(a) shows the actual reaction wheel angular velocities as measured by the motor encoders during the maneuver shown in Figure 5-16(a). Figure 5-18(b) shows the simulated reaction wheel angular velocities necessary to perform the same maneuver. These plots are dominated by the reaction wheel velocity changes required to complete the ninety degree ABBF z axis rotation, which requires equal angular velocity changes in the three orthogonal wheels since they are mounted symmetrically with respect to the ABBF z axis. The actual and simulated angular velocity change magnitudes are roughly the same with maximums of fifteen rad/sec for each wheel, which means that the predicted inertial properties of both the air bearing and reaction wheels are close to the true inertial properties. The most significant difference between the two reaction wheel velocity plots are that there are zero external torques in the simulation, which leads to the reaction wheels returning to zero angular velocity at the completion of the test. However, the actual air bearing experiences external torques due to gravity, friction, and atmospheric damping. This torque builds angular momentum in the system, which must be stored by the reaction wheels in order to maintain the air bearing at its commanded angular orientation. Therefore, the reaction wheels do not return to zero, but rather accelerate at some rate, which is a function of the magnitude and direction of the external torques acting on the system.



(a) Actual RW Angular Velocity

(b) Simulated RW Angular Velocity

Figure 5-18: Actual and Simulated Reaction Wheel Angular Velocity - Zero Initial Angular Momentum

5.4.2 Case Two - Non-Zero Angular Momentum System

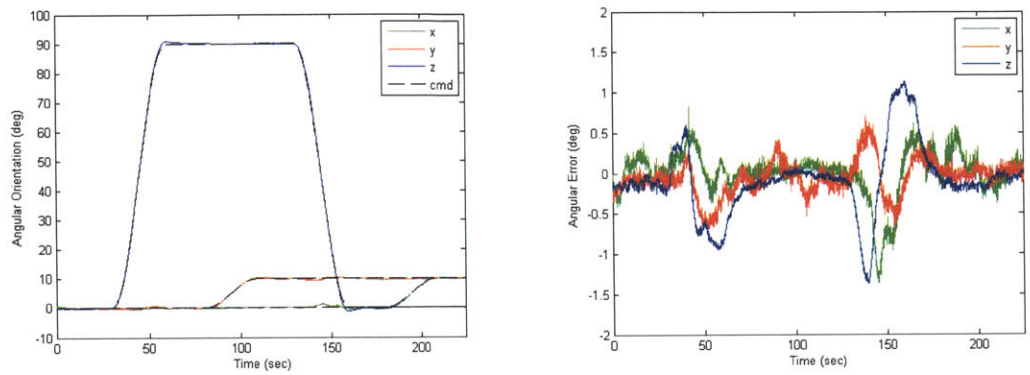
In the second case, the air bearing will be given the same angular orientation commands as case one. However, the air bearing will begin the test with initial reaction wheel speeds given in Equation 5.8. At the beginning of the test, the ABBF frame is aligned with the FIR frame. Therefore, the initial reaction wheel velocities create an angular momentum vector of 0.71 Nms in the FIR x axis as shown in Equation 5.9.

$$\bar{\Omega} = \begin{bmatrix} 40 \\ 40 \\ -80 \end{bmatrix} rad/sec \tag{5.8}$$

$$\bar{H}_{ref} = DI_{RW}\bar{\Omega} = \begin{bmatrix} 0.71 \\ 0 \\ 0 \end{bmatrix} Nms \tag{5.9}$$

Figure 5-19(a) shows the commanded angular orientation profile and the air bearing's response based on the output of the EKF. Figure 5-19(b) shows the angular error between the commanded and EKF estimated angular orientations shown in Figure 5-19(a). Though the air bearing begins the scenario with a large angular momentum vector in the FIR x axis, the air bearing's estimation and control algorithm is capable of estimating the gyroscopic torques caused by the angular momentum vector, and maintaining the air bearing's attitude

to within limits similar to those of the zero angular momentum case. Maximum error of approximately 1.3 degrees occurs as the air bearing begins and completes the ABBF z axis rotation.



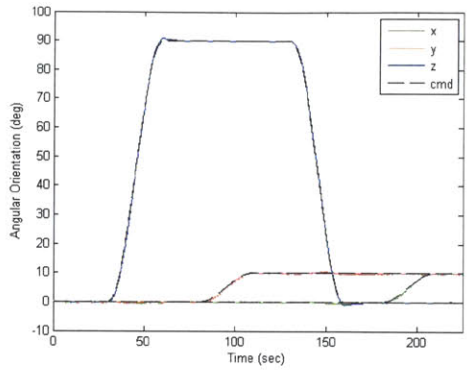
(a) Air Bearing Command and Response

(b) Angular Orientation Error

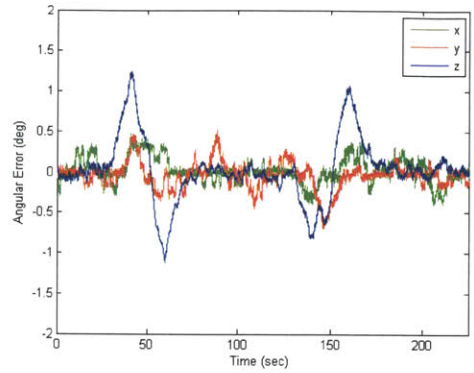
Figure 5-19: Actual Angular Orientation and Error - Nonzero Initial Angular Momentum

The same initial reaction wheel angular rates are applied to the Simulink model, and Figure 5-20(a) shows the simulated air bearing response to the same commanded angular orientation profile. Figure 5-20(b) shows the angular error between the commanded and simulated air bearing orientations. The model predicts maximum angular error of approximately 1.3 degrees in the ABBF z axis as the air bearing lags behind and overshoots the commanded ABBF z axis rotation, which is similar to the actual error given in Figure 5-19(b). Error in the ABBF x and y axes is predicted to have maximums of approximately 0.7 degrees, whereas the actual angular error in the ABBF x axis is approximately 1.3 degrees. The larger ABBF x axis error is likely due to unmodeled external torques.

The most obvious difference between the first and second test cases are the reaction wheel angular velocity plots. Figure 5-21(a) shows the actual angular velocities of the reaction wheels and Figure 5-21(b) shows the simulated angular velocities of the reaction wheels. In both plots, the reaction wheels begin the test at 40, 40, and -80 rad/sec respectively, which creates the large angular momentum vector described above. The effects of gyroscopic torques are clear in the reaction wheel velocity plots because if the torques were not present, the reaction wheels would follow an angular velocity profile similar to those shown in the first case above (Figures 5-18(a) and 5-18(b)) but offset by their initial velocity values.



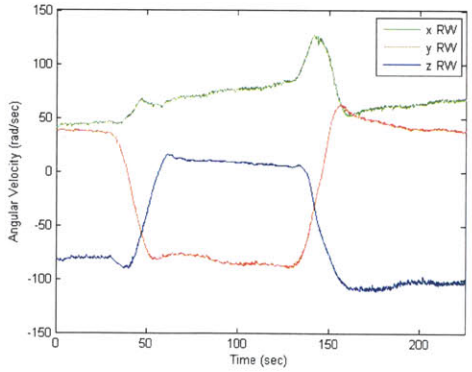
(a) Simulated Air Bearing Command and Simulated Response



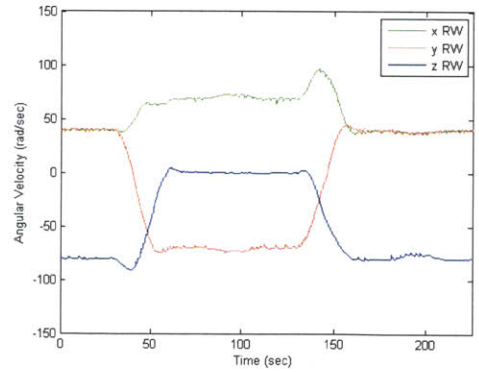
(b) Simulated Angular Orientation Error

Figure 5-20: Simulated Angular Orientation and Error - Nonzero Initial Angular Momentum

However, to maintain air bearing attitude control in the presence of gyroscopic torques, the reaction wheels are required to perform a much different angular velocity profile shown in the below figures. The largest difference between the actual and simulated reaction wheel velocity plots are the presence of external torques on the actual air bearing, which drive the actual reaction wheels to have some unmodeled acceleration seen in Figure 5-21(a) as some slight positive or negative slope to the reaction wheel velocity profile when compared with the simulated profile in Figure 5-21(b).



(a) Actual RW Angular Velocity

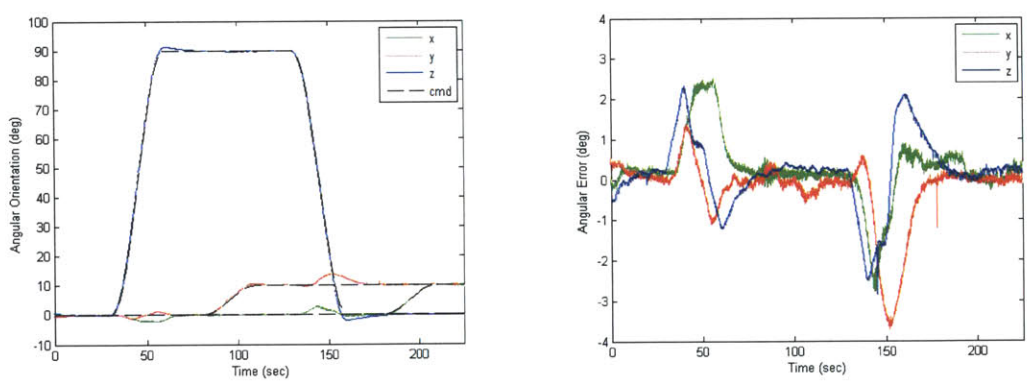


(b) Simulated RW Angular Velocity

Figure 5-21: Actual and Simulated Reaction Wheel Angular Velocity - Nonzero Initial Angular Momentum

5.4.3 Case Three - Nonzero Angular Momentum System - No Feedforward

In the third test case, the reaction wheels are given the same initial angular velocities as case two; 40, 40, and -80 rad/sec respectively, and the air bearing is given the same commanded angular orientation profile. However, in this case, the feedforward term of the control algorithm is disabled. Therefore, rather than predicting the gyroscopic torques caused by the interaction between the air bearing's angular velocity and the angular momentum vector, the LQR controller will have to interpret angular error based on EKF state estimates, and compensate for the error by updating reaction wheel commands. Figure 5-22(a) gives the commanded angular orientation profile and the air bearing's response, and Figure 5-22(b) gives the angular error between the commanded and EKF estimated angular orientations. The additional angular error in all three axes can be seen in both figures. The ABBF z axis lag and overshoot increases from a maximum of approximately 1.3 degrees to 2.5 degrees. The ABBF x and y axis errors increase from a maximum of 1.3 degrees to nearly four degrees.

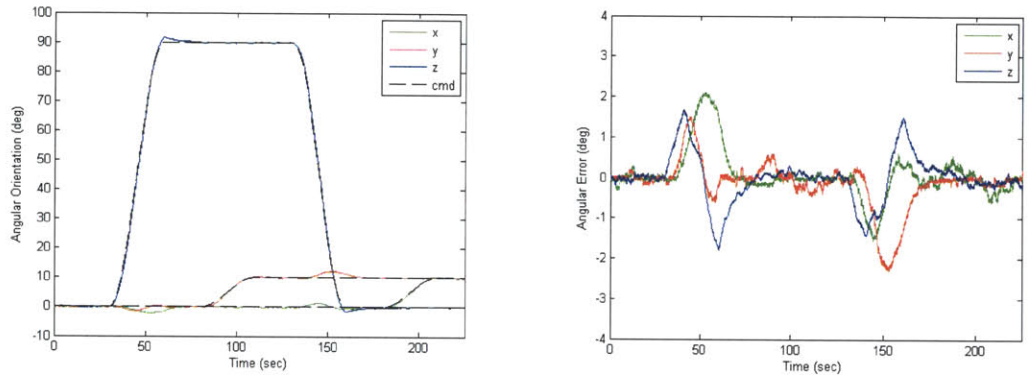


(a) Air Bearing Command and Response

(b) Angular Orientation Error

Figure 5-22: Actual Angular Orientation and Error - Nonzero Initial Angular Momentum, No Feedforward

The same initial conditions and controller changes are applied to the simulation, and Figures 5-23(a) and 5-23(b) give the simulated air bearing response and angular error respectively. Though the simulation predicts that the air bearing's angular error will increase with the disabled feedforward controller, the magnitude of the predicted angular error is less with a maximum of approximately 2.5 degrees. The larger actual angular error is most likely due to unmodeled external torques.



(a) Simulated Air Bearing Command and Simulated Response

(b) Simulated Angular Orientation Error

Figure 5-23: Simulated Angular Orientation and Error - Nonzero Initial Angular Momentum, No Feedforward

5.5 MicroMAS ADCS Scenario

The purpose of the ADCS testbed is to verify and validate ADCS hardware and software algorithms for use on small satellites being developed within the SSL. Now that the air bearing's hardware components have been individually characterized, and the integrated system has been tested, the final step in validating the ADCS testbed is to perform an ADCS scenario test for one of the SSL's satellites.

The Micro-sized Microwave Atmospheric Satellite (MicroMAS) is a three unit cubesat being developed in cooperation between the SSL and Lincoln Laboratory [36]. MicroMAS' mission is to make observations of hurricane dynamics using a microwave spectrometer [36]. In order to complete its mission, MicroMAS will maintain alignment with the Local Vertical, Local Horizontal (LVLH) orbital frame. MicroMAS will have its x axis (long axis) aligned

with the velocity vector and its z axis aligned with the negative nadir vector. The y axis is in the plane tangent to Earth's surface. In order to maintain alignment with this frame, MicroMAS will complete one revolution about its y axis per orbit. Based on MicroMAS' mission requirements, its payload (which consists of one of the three cubesat sections) will spin at one Hertz about the satellite's x axis [36]. Reference Figure 5-24 for clarity on MicroMAS' coordinate system and payload orientation.

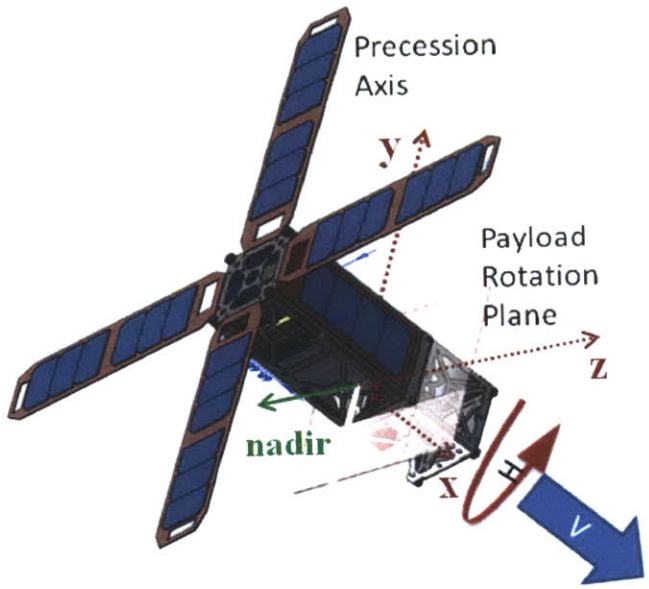


Figure 5-24: MicroMAS with Coordinate System [36]

In order to reduce the effects of gyroscopic torque on the satellite, MicroMAS' x axis reaction wheel will be used to store the momentum induced by spinning the payload at one Hertz. Therefore, the satellite will have zero net angular momentum and should react to control torques just like a satellite with zero angular momentum in all components. The following test will compare the response of three unique systems to the same commanded angular orientation profile. The first system will have zero angular momentum in every component, the second system will have zero net angular momentum with nonzero angular momentum in individual components, and the third system will have a nonzero angular momentum vector. The purpose of the test is to show that a system with net zero angular momentum reacts like a system with zero angular momentum in all components and not like a system with nonzero angular momentum.

To simulate MicroMAS' CONOPs, the air bearing's fourth reaction wheel will be used to mimic MicroMAS' spinning payload. The air bearing's three orthogonal wheels will be used to store the angular momentum of the fourth reaction wheel and create a zero net angular momentum system. Figure 5-25 shows the spin direction of the reaction wheels. MicroMAS will require constant rotation about its y axis (which is perpendicular to the payload spin axis) in order to maintain alignment with the LVLH frame. However, the air bearing is not capable of constant rotation about an axis perpendicular to the fourth reaction wheel's spin axis due to air bearing constraints. Therefore, the air bearing will be commanded to oscillate about the ABBF x axis to stay within its constraints while still rotating the fourth reaction wheel about an axis perpendicular to its spin axis. Figure 5-26 shows the commanded angular orientation profile throughout the test. The ABBF y and z axes will stay at zero attitude while the ABBF x axis oscillates from -15 to +15 degrees.

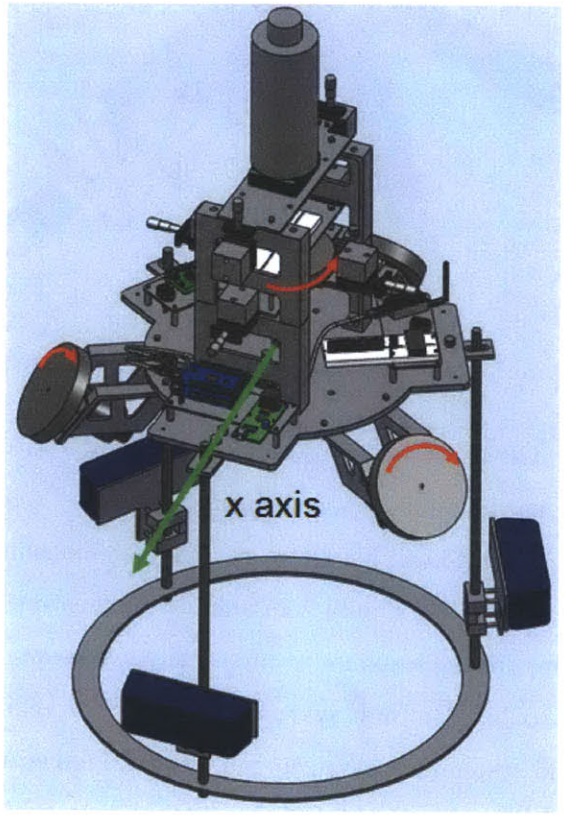


Figure 5-25: MicroMAS CONOPs Test on Air Bearing

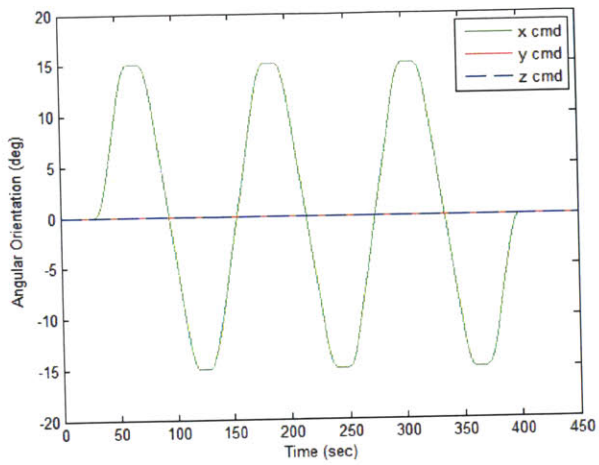
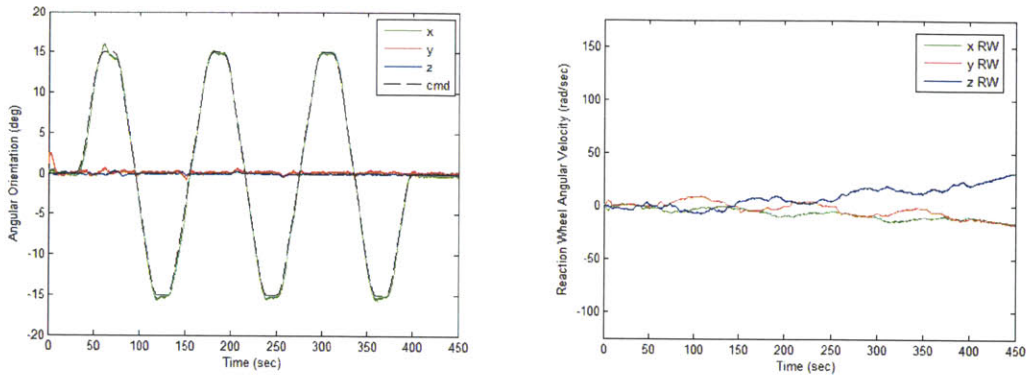


Figure 5-26: Commanded Angular Orientation for MicroMAS Scenario

The first system will follow the same commanded angular orientation shown in Figure 5-26 but will begin with zero angular momentum in all components. The results of this test will be used as a baseline for comparison with the net zero angular momentum system. Figure 5-27(a) shows the commanded and actual angular orientation as estimated by the EKF. The air bearing is able to maintain the commanded angular orientation with maximum error of approximately one degree as it overshoots the first fifteen degree peak. Figure 5-27(b) shows the reaction wheel velocities necessary to maintain the commanded angular orientation. The reaction wheels do not return to zero velocity at the completion of the test due to external torques acting on the system. Though these unknown torques make the overall results difficult to interpret, the same torques should act on the following two scenarios. Therefore, the reaction wheel velocity results for the net zero angular momentum system and the nonzero angular momentum system should still be comparable to the zero angular momentum system.

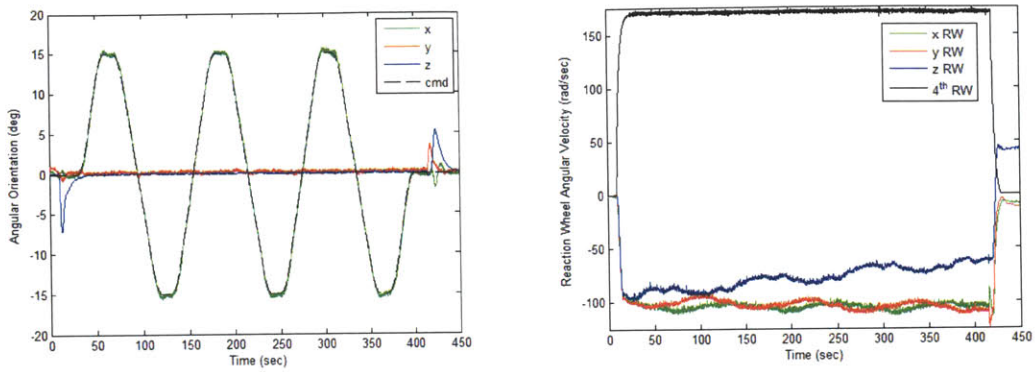


(a) Air Bearing Command and Response

(b) Reaction Wheel Angular Velocity

Figure 5-27: Air Bearing Angular Orientation and Reaction Wheel Angular Velocity - Zero Angular Momentum System

The second system, which simulates MicroMAS' CONOPs begins the test with zero angular momentum in all components. At ten seconds, the fourth reaction wheel is accelerated to 170 rad/sec, which creates an angular momentum vector with a magnitude of 1.23 Nms in the positive FIR z axis. In order to maintain the commanded angular orientation, the three orthogonal reaction wheels accelerate to -98 rad/sec each, which creates an angular momentum vector of equal magnitude in the negative FIR z axis and a net zero angular momentum system. Figure 5-28(a) shows the commanded and actual orientation of the net zero angular momentum system. The angular error at ten seconds is due to the fourth reaction wheel spin up, and the angular error at 420 seconds is due to the fourth reaction wheel returning to zero angular velocity. Figure 5-28(b) shows the reaction wheel velocities throughout the net zero angular momentum test. At ten seconds, the fourth reaction wheel accelerates to 170 rad/sec and the three orthogonal wheels accelerate to -98 rad/sec each. Besides the offset of -98 rad/sec, the three orthogonal reaction wheels respond very similarly to the three reaction wheels in the zero angular momentum test. External torques still act on the system, but the torques are similar and drive the reaction wheels to similar final velocities. At 420 seconds, the fourth reaction wheel returns to zero, and the three orthogonal reaction wheels remove their -98 rad/sec offsets. Once the fourth wheel is stopped, the three orthogonal reaction wheels have velocities almost equal to those of the first test as shown in Figure 5-29, which plots the wheel velocities of both scenario one and two.



(a) Air Bearing Command and Response

(b) Reaction Wheel Angular Velocity

Figure 5-28: Air Bearing Angular Orientation and Reaction Wheel Angular Velocity - Net Zero Angular Momentum System

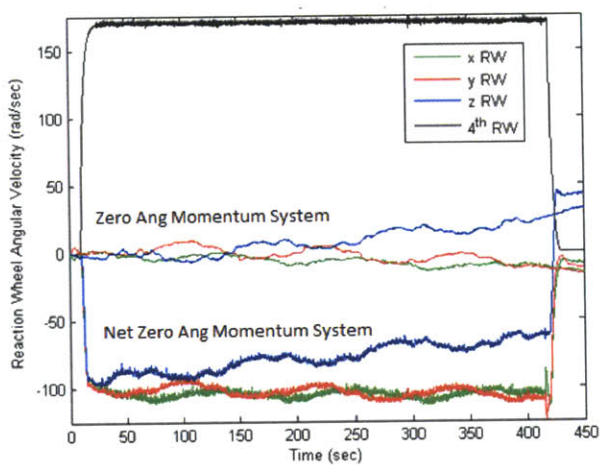
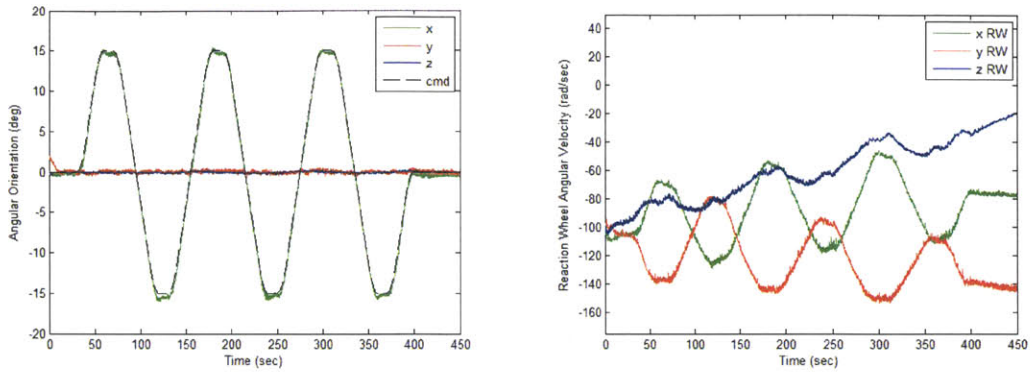


Figure 5-29: Reaction Wheel Angular Velocity - Zero and Net Zero Angular Momentum

The third system will follow the same commanded angular orientation profile given in Figure 5-26, but will begin with an angular momentum vector of 1.23 Nms in the minus FIR z axis. This vector will be created by spinning the three orthogonal reaction wheels to -98 rad/sec each while leaving the fourth reaction wheel at zero velocity. The air bearing is still able to closely track the commanded angular orientation as shown in Figure 5-30(a), but the reaction wheel velocities required to maintain the commanded angular orientation (given in Figure 5-30(b)) are much difference than those in the first two tests. Due to the angular momentum vector, gyroscopic torques have an effect on the system, and the reaction wheels must compensate in order to maintain the air bearing's angular orientation.



(a) Air Bearing Command and Response

(b) Reaction Wheel Angular Velocity

Figure 5-30: Air Bearing Angular Orientation and Reaction Wheel Angular Velocity - Nonzero Angular Momentum System

Overall, the second system simulating MicroMAS' CONOPs performs much like the first system with zero angular momentum. The primary difference is the constant offset of -98 rad/sec per reaction wheel, which is required to store the fourth reaction wheel's angular momentum vector. The gyroscopic torques due to the fourth reaction are canceled by those caused by the equal but opposite angular momentum vector created by the three orthogonal reaction wheels, and the additional control torques necessary to track the air bearing's commanded angular orientation profile are therefore the same as those required by the zero angular momentum system. The third system has a large angular momentum vector, which creates gyroscopic torques that must be compensated for by the reaction wheels. This leads to a much different reaction wheel angular velocity profile than those of the zero and net zero angular momentum systems.

5.6 Testing Summary

To produce useful results for control theory students and satellite development engineers, the ADCS testbed must be tested to characterize and account for the limitations of the system. First, the air bearing's attitude sensors are tested to determine their noise characteristics and observability limitations. Next, the attitude control reaction wheels are tested to measure their response characteristics and torque capabilities. The air bearing is then tested to measure the disturbance torques acting on the system, which are unique to an ADCS simulator operating within a laboratory environment. The primary disturbance torque caused by gravity acting on the air bearing's center of mass can be reduced by carefully aligning the center of mass with the center of rotation. The process for adjusting the center of mass to reduce torque due to gravity is discussed. The integrated ADCS system is then tested in several configurations to create a baseline for expected results in the presence of disturbance torques like gravity, atmospheric drag, and friction. Finally, the air bearing is used to perform a CONOPs test for the SSL's three unit cubesat, MicroMAS in order to prove the functionality of the ADCS testbed as well as provide useful information for the MicroMAS satellite development team.

Chapter 6

Conclusion

The objective of this thesis is to design, develop, and validate an ADCS testbed capable of simulating key characteristics of the space environment in a university laboratory. ADCS systems often require the micro-gravity, reduced friction environment of space to function as designed, and these characteristics are difficult to reproduce in a laboratory. The ADCS testbed designed and built in conjunction with this thesis is capable of simulating micro-gravity and reduced friction using a three degree of freedom rotational air bearing. The ADCS testbed and supporting MATLAB simulation provide university satellite developers with the means to test hardware in the loop ADCS systems just as they will be required to operate on orbit. "Test as you fly" scenarios will help ADCS engineers gain confidence in their designs and provide an increased level of mission assurance for the overall satellite program.

6.1 Thesis Summary

The thesis begins by discussing the basic principles of an attitude determination and control system, which include the subsystem's purpose on a satellite and its importance to a satellite's mission. The current state of the art for satellite ADCS systems is discussed and examples of actual satellites and their corresponding ADCS systems are given. To validate the need for the development of an ADCS testbed, several examples of on-orbit satellite ADCS failures are given. Some failures can be mitigated via software or CONOPs changes while others lead to mission failure. Background research concludes with a discussion of

the current state of ADCS testbed technology and the design requirements for an ADCS testbed within the Space Systems Laboratory.

The thesis then covers the development of a model to simulate the dynamics of the air bearing testbed. The various coordinate systems used by the air bearing are defined, and the air bearing's equations of motion are derived. Development of a MATLAB based simulation of the air bearing is then discussed. The simulation models each major component of the air bearing testbed, which include reaction wheel and air bearing dynamics, attitude estimation using the air bearing's sensors and linearized dynamics equations, and attitude control using feedback and feedforward methods.

Following the simulation development discussion, the thesis covers development of the physical air bearing testbed. The testbed requires all the major subsystems of a generic satellite in order to function independently, and the design and development of each of the subsystems is described. The subsystems include the air bearing's structure, the power system, avionics and communication, attitude determination, attitude control, and ground station operation. Development of the default set of ADCS testbed software is then discussed. The software is designed so that it can be easily modified to meet the requirements of a wide range of ADCS test scenarios. Development of the testbed's external magnetic field generator is discussed along with the generator's operational requirements and capabilities.

Once ADCS testbed development is complete, each component of the air bearing is characterized to ensure the testbed operates as expected. Noise characteristics of the attitude sensors and response characteristics of the attitude control reaction wheels are determined, and the simulation is adjusted to match the physical system. Disturbances acting on the air bearing like torque due to gravity and friction are measured to reduce the error induced by simulating the inertial properties of the space environment in a laboratory. The integrated air bearing is tested to validate its performance throughout several generic ADCS scenarios. Finally, the air bearing is used to perform an ADCS CONOPs scenario for MicroMAS, which is a cubesat being developed within the SSL. The MicroMAS test scenario is used to validate the ADCS testbed's functionality as a useful tool for satellite ADCS system development.

6.2 Future Work

Though the testbed is already an asset to ADCS engineers, the air bearing can be improved to provide expanded functionality for future satellite ADCS developers. The following are just a few examples of testbed updates that would allow for increased testbed capability.

6.2.1 Reaction Wheel Regenerative Motor Controllers

One of the most limiting disturbances associated with the air bearing is the nonlinear braking algorithm required to reduce the angular velocity of the reaction wheels. This is due to the Pololu Trex motor controllers driving voltage across the reaction wheel motor leads rather than current. To mitigate this problem, the SSL recently acquired three Sabertooth dual twelve Amp regenerative motor controllers capable of driving current through the reaction wheel motors to reduce angular velocity when commanded. The Sabertooth motor controllers reverse lead polarity and recharge the batteries to actively reduce reaction wheel angular velocity when commanded. The result is a negative reaction wheel step response which has similar characteristics to the positive step response without the need to apply a nonlinear braking algorithm.

Two Sabertooth dual motor controllers can replace the two Pololu Trex motor controllers relatively easily. The motor controllers have similar physical characteristics, and avionics code updates should be relatively simple. Overall, replacing the Pololu Trex motor controllers with the Sabertooth motor controllers will improve air bearing performance and reduce attitude error due to reaction wheel jitter.

6.2.2 Reaction Wheel Vibration Characterization and Rejection

The air bearing structure is subject to vibration induced by the reaction wheels as they pass through angular velocities that excite fundamental frequencies in the structure. These vibrations cause attitude error that may adversely affect an ADCS test scenario depending on the required attitude control for the given scenario. To avoid this problem, the redundant set of reaction wheels can be used to transfer angular momentum and avoid angular velocities that excite structural fundamental frequencies.

First, the fundamental frequencies must be identified by cycling the reaction wheels from negative angular velocity maximums to positive maximums and carefully measuring

the air bearing's vibration response using sensitive, high frequency accelerometers. After the frequencies are identified, angular momentum transfer algorithms can be written to transfer angular momentum from one of the four reaction wheels to another in order to avoid the reaction wheel angular velocities that cause air bearing vibration. If a wheel must pass through one of the fundamental frequency exciting velocities, the angular momentum transfer algorithm can be used to have the wheel pass through quickly in order to minimize vibration.

6.2.3 Variable EMFG

The current EMFG creates a magnetic field in the air bearing's FIR y axis. The strength of the magnetic field can be manually adjusted from its maximum value in the negative direction to the maximum value in the positive direction. In order to better simulate Earth's magnetic field over the course of an orbit, four additional coils should be aligned with the air bearing's FIR x and z axes. The six total coils could be used to create a magnetic field in any direction with respect to the air bearing. Furthermore, the coils could be controlled via a microprocessor and high power H bridge circuits so that the magnetic field vector could be automatically slewed throughout the ADCS test. The automatic slew could be designed to mimic the motion of the Earth's magnetic field over the course of a satellite's orbit.

6.2.4 Estimation and Control with Integrated SPHERES

The air bearing is designed to physically integrate with a SPHERES satellite, and the ground station is capable of independently communicating with the SPHERES satellite. However, the SPHERES satellite could provide further functionality if it were able to directly communicate with the air bearing's avionics system, which is driven by dual Arduino Megas. The SPHERES satellite has a UART TTL port integrated into its expansion port and the main Arduino has an available UART TTL port. With some software additions, the SPHERES satellite could be used to provide additional attitude sensing directly to the estimation and control algorithm using its global metrology system, and it could provide control torque using its cold gas thrusters.

6.2.5 Automated Center of Mass Adjusters

The air bearing currently uses CM adjusters that require manual operation to change the CM and reduce torque due to gravity. The manual CM adjusters could be replaced by motorized linear slides that could adjust the CM in real time during a test to more precisely align the CM with the center of rotation and significantly reduce the disturbance torque due to gravity. Real time CM adjustment would be most useful in cases where the SPHERES thrusters are used for attitude control. Because the thrusters use an expendable fuel source, the CM of the system will change as the fuel is depleted. Real time automated CM adjusters would be able to estimate the change in the CM due to fuel usage and adjust the CM accordingly throughout the test.

An additional application for automated CM adjusters would be to use them as a type of attitude control device. The CM adjusters could manipulate the CM such that the resulting torque due to gravity would cause controlled maneuvers about rotational axes in the plane perpendicular to the gravity vector.

Appendix A

ADCS Testbed Users Manual

A.1 MATLAB Simulation

The ADCS testbed MATLAB model is named:

air_bearing_sim.mdl

The file should be saved at the following location on the ground station computer:

C:\Users\SSL User\Desktop\Air Bearing Simulation and Init File

If the simulation file is not in the above location, or the file has been modified unexpectedly, a backup of the simulation file is located on the SSL's server at:

\\spacelab\Projects\Air Bearing Testbed\Air Bearing Simulation and Init File

The Simulink simulation was created using MATLAB version R2010b. This version or a more recent one should be used to open and run the simulation.

The MATLAB script files called by the simulation are stored in the same location as the simulation file. The required MATLAB script files are listed below. If any of these files are missing or modified from the ground station computer, they can be retrieved from the SSL server at the same location listed above.

- test_init.m
- linear_eq.m
- skew.m
- mag.m

- sim_plot.m

The simulation can be run by simply opening the Simulink model using MATLAB and clicking the "Start Simulation" block (play button) on the toolbar at the top of the file. The Simulink model calls the test_init.m file before beginning in order to define the required initial conditions. In order to change the simulation's initial conditions, open the test_init.m file and make the desired changes. Initial condition changes include the simulation timestep, reaction wheel initial velocities, air bearing initial angular position, air bearing inertia properties, EKF weighting matrices, and LQR weighting matrices. The desired simulation plots to be displayed once the simulation has completed can also be selected in the test_init.m file.

The Simulation automatically runs the test_init.m and sim_plot.m files. To change the automatically run script files, use the following process.

1. Right click on the "Clock" block in the simulation
2. Select "Block Properties..."
3. Select the "Callbacks" tab
4. To add/remove/change script files that run before the propagation loop begins, select "InitFcn" in the "Callback functions list:" and type the name of the script files (excluding ".m")
5. To add/remove/change script files that run after the propagation loop ends, select "StopFcn" in the "Callback functions list:" and type the name of the script files (excluding ".m")
6. Select "Ok"
7. Ensure that the callback script files are saved in the same location as the simulation file

To change the commanded air bearing angular orientation and rate, double click on the "commanded position and rate" block and change the x, y, and z inputs as desired. The user can change other portions of the simulation as well, though the simulation should be saved under a new file name in order to maintain the original copy.

A.2 Air Bearing Software

A.2.1 Required Arduino IDE Software

The air bearing uses two Arduino Megas as its avionics computer. The Arduinos require an IDE in order to upload software and download data. The Arduino IDE software is installed on the ground station computer and a shortcut to the program is located on the computer's desktop under the name "Arduino IDE". If the program is missing, it can be downloaded from Arduino's website at:

http://www.arduino.cc/en/Main/Software

A.2.2 Opening and Updating Arduino Code for the Air Bearing

The Main and Auxiliary Arduinos each have default software provided as part of this thesis. This software can be updated to meet the needs of an ADCS test scenario, but changes should be saved under a new file name. The original software should not be modified. The Main Arduino's software is named:

mega_main.pde

The Main Arduino's software is located on the ground station computer at the following location:

C:\Users\SSL User\Desktop\Arduino Avionics Software\mega_main

If the mega_main.pde file is not in the above location, or the file has been modified unexpectedly, a backup of the file is located on the SSL's server at:

\\spacelab\Projects\Air Bearing Testbed\Arduino Avionics Software \mega_main

The Main Arduino software files can also be found in Appendix C of this thesis. The mega_main.pde file must be opened with the Arduino IDE software. The default software is a compilation of four code files. The primary file is the mega_main.pde file. The other three supporting files are listed below.

- functions.h
- matrix.h
- init.h

The function.h and matrix.h file should not require regular updating. The init.h file however contains the test scenario initial conditions. The test_init.m file used to define the initial conditions for the Simulink simulation also creates an init.h file containing the same initial conditions. The following process should be used to update the init.h file for a given set of initial conditions in the Arduino software.

1. Open test_init.m from C:\Users\SSL User\Desktop\Air Bearing Simulation and Init File

2. Update test_init.m to reflect the desired initial conditions

3. Run test_init.m, which creates an init.h file and stores the file in the same location as test_init.m

4. Open mega_main.pde from C:\Users\SSL User\Desktop\Arduino Avionics Software \mega_main using the Arduino IDE software

5. If a previous init.h file is already in the software (there should always be an init.h file in place), Select the "init.h" tab at the top of the Arduino IDE. If no init.h file is in place, go to step 8

6. With the "init.h" tab highlighted, click the right arrow block in the top right corner of the Arduino IDE

7. Select "Delete"

8. Select "Sketch" from the menu at the top of the Arduino IDE

9. Select "Add File..." from the drop-down menu

10. Navigate to the location of the updated init.h file, which should be at C:\Users\SSL User\Desktop\Air Bearing Simulation and Init File

11. Select the init.h file and click "Open"

12. The updated init.h file should now be in place. Click the play button in the top left corner of the Arduino IDE to ensure the software compiles correctly.

The commanded air bearing angular orientation and rate must also be updated in the mega_main.pde software to match the commands given in the simulation. To update commanded angular orientation and rate, go to the "Commanded Angular Orientation and Commanded Angular Rate" section of the mega_main.pde software. Angular rate and orientation can be assigned for each axis in this section.

The Auxiliary Arduino's software is named:

mega_aux.pde

The Auxiliary Arduino's software is located on the ground station computer at the following location:

C:\Users\SSL User\Desktop\Arduino Avionics Software\mega_aux

If the mega_aux.pde file is not in the above location, or the file has been modified unexpectedly, a backup of the file is located on the SSL's server at:

\\spacelab\Projects\Air Bearing Testbed\Arduino Avionics Software \mega_aux

The Auxiliary Arduino software file can also be found in Appendix C of this thesis. The mega_aux.pde file must be opened with the Arduino IDE software. Unlike the mega_main.pde software, the mega_aux.pde software does not require regular updating.

A.2.3 Uploading Arduino Code to the Air Bearing

The software for the Main and Auxiliary Arduinos must be uploaded to each Arduino separately. The following list describes the process necessary to upload software from the ground station to the air bearing Arduinos.

1. Open the mega_main.pde or mega_aux.pde using the Arduino IDE software

2. Ensure the TruLink wireless USB receiver is connected to the ground station computer and the green light on the receiver is on

3. On the air bearing, flip the black switch located next to the air bearing's USB receiver to the on position. Do not turn on the black switches located next to the reaction wheel motor controllers

4. The ground station's wireless USB receiver should recognize the air bearing's USB receiver, and a blue indicator light should illuminate on the ground station's receiver.

If this light does not illuminate (occurs once every 8-10 uses), physically remove the ground station USB receiver from its base and replace it. Resetting the receiver should allow the two USB receivers to recognize each other, and the blue indicator light should illuminate.

5. Return to the Arduino IDE and select "Tools" from the menu at the top of the IDE

6. Select "Serial Port" from the drop-down menu

7. The Main Arduinos will be represented by open COM ports. Usually, the Main Arduino is assigned "COM3" on the ground station computer and the Auxiliary Arduino is assigned "COM4". However, the user should check to ensure which Arduino is assigned to which port. Select the desired COM port.

8. Select the "Upload" button at the top of the Arduino IDE. This will begin the upload process as indicated at the bottom of the Arduino IDE

9. Wait until the Arduino IDE indicates that the upload has completed at the bottom of the IDE

In general, software changes must be uploaded to the Main Arduino regularly to reflect initial condition changes, output changes, and commanded angular orientation and rate changes. However, the Auxiliary Arduino software does not require regular updating, and if no changes have been made, the Auxiliary Arduino's software does not need to be uploaded.

A.2.4 Downloading and Processing Air Bearing Data

The first step in downloading data from the air bearing during testing is indicating which data should be downloaded. At the top of the mega_main.pde software is the section labeled "Set Desired Outputs". Variables defining the air bearing's state are available for output at each time step during the test scenario. In this section, the desired outputs can be selected, and the software can be uploaded to the Main Arduino. The following list gives a brief description of each available output variable.

- timekeep - Current test scenario time in seconds

- RealDeltaT - Time step in milliseconds

- deltaT - Time step in number of 4kHz cycles from RTC module
- RW1_ComSpeed - Commanded RWCS x axis reaction wheel speed in rad/sec
- RPS1 - Magnitude of measured RWCS x axis reaction wheel speed in rad/sec
- RW1_Volt - Commanded RWCS x axis reaction wheel voltage from motor controller
- RW2_ComSpeed - Commanded RWCS y axis reaction wheel speed in rad/sec
- RPS2 - Magnitude of measured RWCS y axis reaction wheel speed in rad/sec
- RW2_Volt - Commanded RWCS y axis reaction wheel voltage from motor controller
- RW3_ComSpeed - Commanded RWCS z axis reaction wheel speed in rad/sec
- RPS3 - Magnitude of measured RWCS z axis reaction wheel speed in rad/sec
- RW3_Volt - Commanded RWCS z axis reaction wheel voltage from motor controller
- x_ComPos - Commanded ABBF x axis angular orientation in deg
- y_ComPos - Commanded ABBF y axis angular orientation in deg
- z_ComPos - Commanded ABBF z axis angular orientation in deg
- x_ComRate - Commanded ABBF x axis angular rate in deg/sec
- y_ComRate - Commanded ABBF y axis angular rate in deg/sec
- z_ComRate - Commanded ABBF z axis angular rate in deg/sec
- tx_p - Estimated ABBF x axis angular orientation in deg
- ty_p - Estimated ABBF y axis angular orientation in deg
- tz_p - Estimated ABBF z axis angular orientation in deg
- wx_p - Estimated ABBF x axis angular rate in deg/sec
- wy_p - Estimated ABBF y axis angular rate in deg/sec
- wz_p - Estimated ABBF z axis angular rate in deg/sec
- wx_dot_p - Estimated ABBF x axis angular acceleration in deg/sec^2

- wy_dot_p - Estimated ABBF y axis angular acceleration in deg/sec^2
- wz_dot_p - Estimated ABBF z axis angular acceleration in deg/sec^2
- RPS4 - Magnitude of measured 4th reaction wheel speed in rad/sec
- x_gyro - Raw measurement of IMU's x axis rate gyroscope in deg/sec
- y_gyro - Raw measurement of IMU's y axis rate gyroscope in deg/sec
- z_gyro - Raw measurement of IMU's z axis rate gyroscope in deg/sec
- x_MeasRate - IMU rate gyro measurement in ABBF x axis in deg/sec
- y_MeasRate - IMU rate gyro measurement in ABBF y axis in deg/sec
- z_MeasRate - IMU rate gyro measurement in ABBF z axis in deg/sec
- x_acc - Raw measurement of IMU's x axis linear accelerometer in *g's*
- y_acc - Raw measurement of IMU's y axis linear accelerometer in *g's*
- z_acc - Raw measurement of IMU's z axis linear accelerometer in *g's*
- x_mag - Raw measurement of Magnetometer's x axis magnetic field sensor
- y_mag - Raw measurement of Magnetometer's y axis magnetic field sensor
- z_mag - Raw measurement of Magnetometer's z axis magnetic field sensor
- x_Mag - Magnetometer measurement in ABBF x axis
- y_Mag - Magnetometer measurement in ABBF y axis
- z_Mag - Magnetometer measurement in ABBF z axis
- Wx_est - Estimated angular velocity of RWCS x axis reaction wheel from state space model
- Wy_est - Estimated angular velocity of RWCS y axis reaction wheel from state space model
- Wz_est - Estimated angular velocity of RWCS z axis reaction wheel from state space model

- x_grav - Commanded ABBF x axis gravity vector component
- y_grav - Commanded ABBF y axis gravity vector component
- z_grav - Commanded ABBF z axis gravity vector component
- x_grav_pos - Estimated ABBF x axis angular orientation from gravity vector measurement
- y_grav_pos - Estimated ABBF y axis angular orientation from gravity vector measurement
- z_grav_pos - Estimated ABBF z axis angular orientation from gravity vector measurement
- x_mag_cmd - Commanded ABBF x axis mag field vector component
- y_mag_cmd - Commanded ABBF y axis mag field vector component
- z_mag_cmd - Commanded ABBF z axis mag field vector component
- x_mag_pos - Estimated ABBF x axis angular orientation from magnetic field vector measurement
- y_mag_pos - Estimated ABBF y axis angular orientation from magnetic field vector measurement
- z_mag_pos - Estimated ABBF z axis angular orientation from magnetic field vector measurement

Once the software with the desired outputs is uploaded and the test is ready to begin, select the "Serial Monitor" at the top of the Arduino IDE. The serial monitor displays the data received over the selected COM port. The data will stream to the serial monitor throughout the test. Once the test is complete, the data from the serial monitor can be selected and copied into a text file or Excel file for evaluation using Excel or MATLAB.

A.3 Air Bearing Hardware

A.3.1 Air Bearing SolidWorks Model

The air bearing's SolidWorks files are located on the ground station computer at the following location:

C:\Users\SSL User\Desktop\Air Bearing SolidWorks Files

The SolidWorks assembly file which reflects the air bearing's current configuration is stored in the above location and named:

Air Bearing Assembly with SPHERE

If the SolidWorks files are missing or modified, a backup of the files are located on the SSL's server at:

\\spacelab\Projects\Air Bearing Testbed\Air Bearing SolidWorks Files

The SolidWorks model of the air bearing is used to determine the inertial properties of the air bearing for the simulation as well as the estimation and control algorithm. Changes can be made to the SolidWorks model and the updated inertial properties can be found by clicking on the "Mass Properties" button in the SolidWorks "Evaluate" menu.

A.3.2 Air Bearing Maintenance

Maintaining the air bearing is relatively simple and consists of cleaning the air filters and removing excess water from the air dryer after 8 - 10 hours of air bearing use. To clear the four black air filters; with air pressure applied, slowly twist the knob at the bottom of each filter one at a time to allow some air flow through the filter. Allow air to flow for approximately ten seconds and repeat for each filter.

To remove excess water from the white compressed air dryer, turn off compressed air flow at the SSL laboratory wall source. With the air flow off, twist the knob at the bottom of the first black air filter until all compressed air between the wall source and the filter has been removed. This may take a minute or two. Place a disposable cup below the output of the compressed air dryer and slowly open the value. Some compressed air may still be in the system, and opening the value quickly may eject dirty water around the lab. Once the value is opened and any water is drained, close the value and the air filter's knob, and turn on the compressed air source at the wall.

A.3.3 Charging Air Bearing Batteries

The following process explains how to properly charge the air bearing's three batteries. In general, the batteries should not be removed from the air bearing so that the air bearing's CM is not significantly changed.

1. Disconnect the EMFG leads from the rear of the power source located next to the air bearing

2. Connect the power inputs from the AstroFlight battery charger, which should be located within the SSL

3. Before turning on the power source, turn the course voltage control knob counter clockwise until it stops. This sets the voltage output to zero and ensures the battery charger is not damaged when the power source is turned on

4. Turn on the power source and turn the course voltage control knob clockwise until the battery charger registers just over 12 voltages at the input

5. Turn the "Amps Adjust" knob on the battery charger counter clockwise until it stops. This sets the charging current to zero and protects the batteries while they are being connected to the charger

6. Disconnect one of the batteries at the cable joint directly next each battery pack

7. Using the extension cable, connect the battery pack to the battery charger. The charger should register that the battery is connected

8. Turn the "Amps Adjust" knob clockwise until it registers 6.5 Amps, which is the correct charging current for the air bearing's battery packs

9. The battery charger will sound an alarm when the battery charge process is complete. Disconnect the battery pack and return the "Amps Adjust" knob to zero

10. Repeat steps 5 through 9 for the remaining two battery packs

11. Reconnect the battery packs to their respective air bearing power cables and reconnect the EMFG leads to the power supply

A.3.4 Floating the Air Bearing

The following process explains how to properly float the air bearing.

1. Turn the large black knob on the top of the pressurized air filter assembly clockwise until the gauge reads 50 PSI. Air flow through the support column should be heard at this point

2. Identify the three support rods holding the rotating portion of the air bearing above the support column, and remove the five screws holding the support rods in place. One of the rods has one screw on the bottom and no screw on the top due to its location below the center support truss

3. Grab the air bearing's safety ring directly out from one of the support rods and pull towards yourself and slightly up, which will lift the air bearing up off of the support rod

4. Remove the support rod and **carefully** lower the air bearing down into the support bowl. Do not allow the hemisphere to fall into the bowl.

5. Once the air bearing is resting in the support bowl, continue to rotate the air bearing down and remove the two remaining support rods on the opposite side of the air bearing

The following process explains how to properly stow the air bearing.

1. Place one of the support rods in its correct location rotating the air bearing as necessary to get it out of the way

2. Place the bottom screw in the support rod and tighten the screw so that it is fixed in its position

3. Rotate the air bearing as necessary to start the top screw in the support rod. Leave the screw loose so the air bearing can still be maneuvered

4. Place a second support rod in its correct location rotating the air bearing as necessary to get it out of the way

5. Place the bottom screw in the second support rod and tighten the screw to fix it in place

6. Start the top screw in the second support rod, again leaving it loose to allow some air bearing rotation

7. Grab the air bearing's safety ring directly out from the third support rod position and pull towards yourself to lift the air bearing up enough to place the third support rod in position

8. Place the bottom screw in the third support rod to fix it in place

9. Tighten all five screws attaching the three support rods

10. Turn the large black knob on the top of the pressurized air filter assembly counter clockwise until all air pressure is dissipated and the gauge reads zero PSI

A.3.5 Operating the Air Bearing

The following process explains how to operate the air bearing once a program has been uploaded to the Arduino processors and the air bearing has been floated.

1. The air bearing should be floating, the air bearing's avionics system should be powered on, and the reaction wheel motor controllers should be powered off

2. Open the serial monitor by clicking the serial monitor button on the top of the Arduino IDE at the ground station computer

3. On the air bearing's Main Arduino, press and release the small black reset button to restart the uploaded program and ensure the reaction wheels are not being sent commands

4. Turn on both reaction wheel motor controllers using the two black switches next to the motor controllers

5. Manually place the air bearing in its desired initial angular orientation and press the Main Arduino's black reset button again

6. Hold the air bearing in place until data begins to stream back from the air bearing as observed in the serial monitor on the ground station computer. Streaming data indicates that the air bearing processors have completed initialization and entered the estimation and control loop

7. If the reaction wheels are given zero initial angular velocity, the air bearing processors will enter the est/control loop approximately five seconds after the last depression of the Main Arduino's reset button. If the reaction wheels are given a nonzero initial angular velocity, the user must continue holding the air bearing as the reaction wheels spin up to achieve their initial angular velocities. After approximately fifteen seconds, data will begin streaming in the serial monitor indicating the air bearing has entered the est/control loop, and the air bearing may be released

A.3.6 Air Bearing Center of Mass Adjustment

Course adjustments can be made to the air bearing's CM by moving the battery packs up and down their threaded attachment rods. The battery packs should be placed such that the CM is as close to the center of rotation as possible using the rough adjustment method, but still below the center of rotation. Once the battery packs have been adjusted and any trim masses have been added to get the CM close to but just below the center of rotation, the following process can be used for fine CM adjustment.

Fine adjustments to the air bearing's CM will use the cm_finder.m file in conjunction with the air bearing's attitude estimation and control system. The cm_finder.m file can be found at the below location on the ground station computer:

C:\Users\SSL User\Desktop\Air Bearing Simulation and Init File

The following process explains how to make fine adjustment to the air bearing's center of mass using the cm_finder.m file.

1. Float the air bearing, turn on the air bearing's avionics system, upload a commanded angular orientation profile of zero, and ensure that reaction wheel commanded and measured angular rates are being sent to the ground station. Previous checklists explain how to complete these tasks

2. Allow the air bearing to maintain its commanded zero attitude orientation until one of the reaction wheels approaches saturation. Do not allow a reaction wheel to saturate

3. Store the reaction wheel data in a text file, turn off the air bearing's reaction wheels, and allow the wheels to return to zero angular velocity without slamming the air bearing against its safety ring

4. In MATLAB, ensure that the measured reaction wheel angular velocity magnitudes have the correct signs using the commanded reaction wheel angular velocities

5. Transfer the measured reaction wheel angular velocities to Excel and place a linear best fit line on each of the three reaction wheel angular velocity plots

6. The slope of the best fit line represents each reaction wheel's constant angular acceleration due to gravitational torque acting on the air bearing

7. Enter the three constant angular acceleration coefficients into the three corresponding locations in the cm_finder.m script file

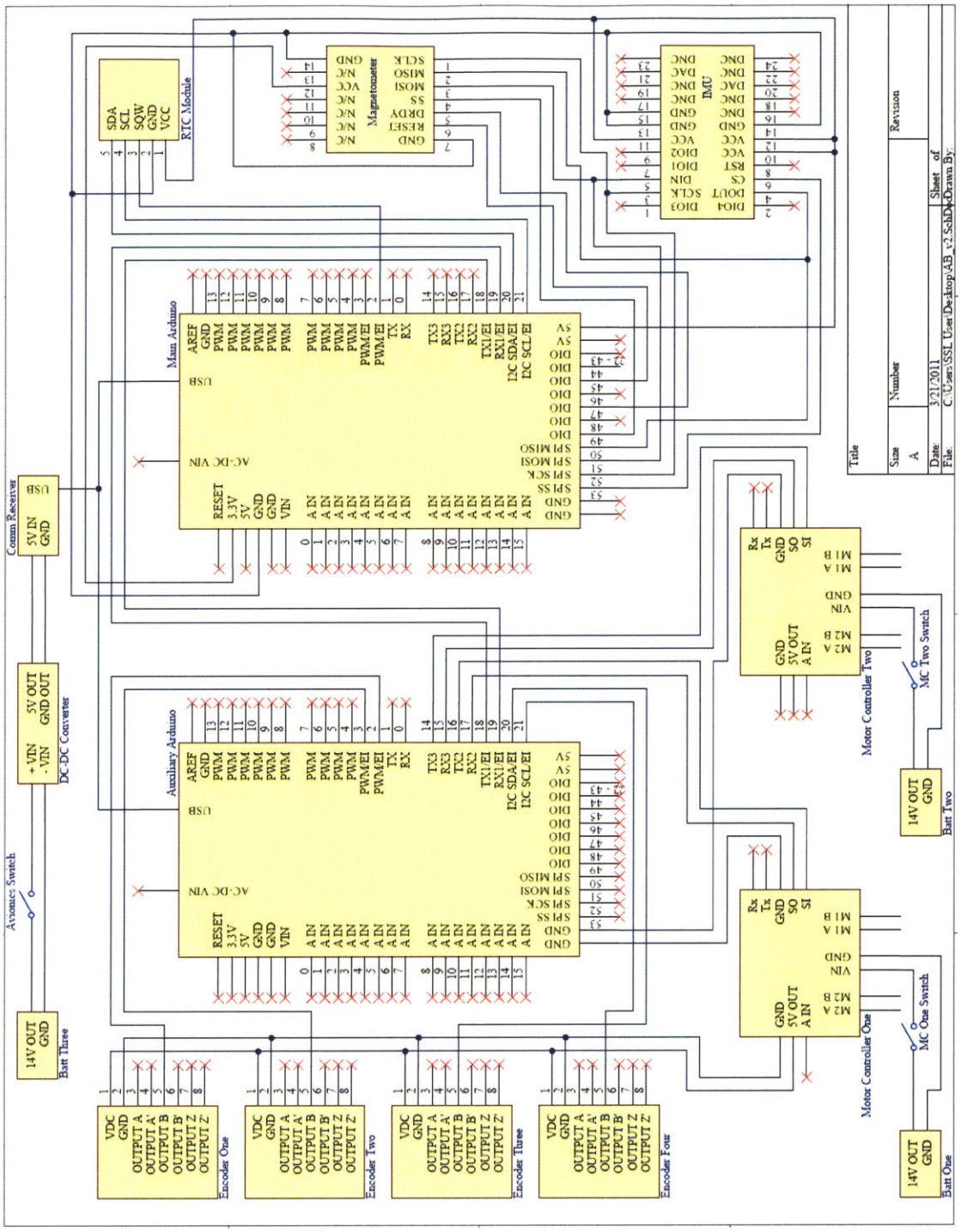
8. Run the script file and record the x_adjust and y_adjust output values

9. For each of the ABBF x and y axis CM adjusters, adjust both CM adjusters in each axis the distance required by the cm_finder.m file. A positive value means that both CM adjusters should be moved the given distance in the positive ABBF axis direction, and visa versa for a negative value. The units of the x_adjust and y_adjust values represent one hundredth of a millimeter, which is the minimum discretized adjustment value on the CM adjusters

10. Repeat the fine CM adjustment process from the beginning until the torque due to gravity is reduced to an acceptable amount

Appendix B

Testbed Wiring Schematic


Main Arduino
Auxiliary Arduino
Magnetometer
IMU
RTC Module
Comm Receiver
DC-DC Converter
Avionics Switch
Motor Controller Two
Motor Controller One
MC Two Switch
MC One Switch
Batt Three
Batt Two
Batt One
Encoder One
Encoder Two
Encoder Three
Encoder Four
Title
Size
Number
Revision
Date: 3/21/2011
File: C:\Users\SSL User\Desktop\AB v2.SchDoc
Sheet of
Drawn By:

Appendix C

Provided Arduino Software

C.1 Provided init.h Code

This code can be copied into a text file, named "init.h", and included in the Arduino's IDE for use on the Main Arduino. However, this file should be created using the test_init.m MATLAB script in Section C.6.

```
// initial values

int timestep =  205;

double Ixx = 4.077;
double Iyy = 4.136;
double Izz = 2.322;

double Ia = 0.00725;

double deriv_gain = 1.50;

double Wx_est_Prev =   0;
double Wy_est_Prev =   0;
double Wz_est_Prev =   0;

double Wx_cmd_ff_Prev =   0;
double Wy_cmd_ff_Prev =   0;
double Wz_cmd_ff_Prev =   0;

double ix_est_Prev =   0;
double iy_est_Prev =   0;
double iz_est_Prev =   0;

double tx_p_Prev = 0.00;
double ty_p_Prev = 0.00;
double tz_p_Prev = 0.00;

double wx_p_Prev = 0.00;
double wy_p_Prev = 0.00;
double wz_p_Prev = 0.00;

double DCM_11_Prev = 1.00000000;
double DCM_12_Prev = 0.00000000;
double DCM_13_Prev = 0.00000000;
double DCM_21_Prev = 0.00000000;
double DCM_22_Prev = 1.00000000;
double DCM_23_Prev = 0.00000000;
double DCM_31_Prev = 0.00000000;
double DCM_32_Prev = 0.00000000;
double DCM_33_Prev = 1.00000000;

double Ad_11 = 1.00000000;
double Ad_12 = 0.00000000;
double Ad_13 = 0.00000000;
double Ad_14 = 0.05004883;
double Ad_15 = 0.00000000;
double Ad_16 = 0.00000000;
```

```
double Ad_21 = 0.00000000;
double Ad_22 = 1.00000000;
double Ad_23 = 0.00000000;
double Ad_24 = 0.00000000;
double Ad_25 = 0.05004883;
double Ad_26 = 0.00000000;
double Ad_31 = 0.00000000;
double Ad_32 = 0.00000000;
double Ad_33 = 1.00000000;
double Ad_34 = 0.00000000;
double Ad_35 = 0.00000000;
double Ad_36 = 0.05004883;
double Ad_41 = 0.00000000;
double Ad_42 = 0.00000000;
double Ad_43 = 0.00000000;
double Ad_44 = 1.00000000;
double Ad_45 = 0.00000000;
double Ad_46 = 0.00000000;
double Ad_51 = 0.00000000;
double Ad_52 = 0.00000000;
double Ad_53 = 0.00000000;
double Ad_54 = 0.00000000;
double Ad_55 = 1.00000000;
double Ad_56 = 0.00000000;
double Ad_61 = 0.00000000;
double Ad_62 = 0.00000000;
double Ad_63 = 0.00000000;
double Ad_64 = 0.00000000;
double Ad_65 = 0.00000000;
double Ad_66 = 1.00000000;

double Bd_11 = -0.00000091;
double Bd_12 = -0.00000091;
double Bd_13 = 0.00000182;
double Bd_21 = 0.00000156;
double Bd_22 = -0.00000156;
double Bd_23 = 0.00000000;
double Bd_31 = -0.00000226;
double Bd_32 = -0.00000226;
double Bd_33 = -0.00000226;
double Bd_41 = -0.00003640;
double Bd_42 = -0.00003640;
double Bd_43 = 0.00007281;
double Bd_51 = 0.00006216;
double Bd_52 = -0.00006216;
double Bd_53 = 0.00000000;
double Bd_61 = -0.00009043;
double Bd_62 = -0.00009043;
double Bd_63 = -0.00009043;

double L11 = 0.00000000;
double L12 = 0.00000000;
double L13 = 0.00000000;
double L14 = 0.00589659;
```

```
double L15 = 0.00000000;
double L16 = 0.00000000;
double L17 = 0.30924972;
double L18 = 0.00000000;
double L19 = 0.00000000;
double L110 = 0.30924972;
double L111 = 0.00000000;
double L112 = 0.00000000;
double L21 = 0.00000000;
double L22 = 0.00000000;
double L23 = 0.00000000;
double L24 = 0.00000000;
double L25 = 0.00589659;
double L26 = 0.00000000;
double L27 = 0.00000000;
double L28 = 0.30924972;
double L29 = 0.00000000;
double L210 = 0.00000000;
double L211 = 0.30924972;
double L212 = 0.00000000;
double L31 = 0.00000000;
double L32 = 0.00000000;
double L33 = 0.00000000;
double L34 = 0.00000000;
double L35 = 0.00000000;
double L36 = 0.00589659;
double L37 = 0.00000000;
double L38 = 0.00000000;
double L39 = 0.30924972;
double L310 = 0.00000000;
double L311 = 0.00000000;
double L312 = 0.30924972;
double L41 = 0.00000000;
double L42 = 0.00000000;
double L43 = 0.00000000;
double L44 = 0.49975695;
double L45 = 0.00000000;
double L46 = 0.00000000;
double L47 = 0.00589659;
double L48 = 0.00000000;
double L49 = 0.00000000;
double L410 = 0.00589659;
double L411 = 0.00000000;
double L412 = 0.00000000;
double L51 = 0.00000000;
double L52 = 0.00000000;
double L53 = 0.00000000;
double L54 = 0.00000000;
double L55 = 0.49975695;
double L56 = 0.00000000;
double L57 = 0.00000000;
double L58 = 0.00589659;
double L59 = 0.00000000;
double L510 = 0.00000000;
```

```
double L511 = 0.00589659;
double L512 = 0.00000000;
double L61 = 0.00000000;
double L62 = 0.00000000;
double L63 = 0.00000000;
double L64 = 0.00000000;
double L65 = 0.00000000;
double L66 = 0.49975695;
double L67 = 0.00000000;
double L68 = 0.00000000;
double L69 = 0.00589659;
double L610 = 0.00000000;
double L611 = 0.00000000;
double L612 = 0.00589659;

double K11 = -87.82774816;
double K12 = 152.19152231;
double K13 = -121.24598643;
double K14 = -342.61780520;
double K15 = 595.11716726;
double K16 = -437.88622843;
double K21 = -87.82774816;
double K22 = -152.19152231;
double K23 = -121.24598643;
double K24 = -342.61780520;
double K25 = -595.11716726;
double K26 = -437.88622843;
double K31 = 175.65549631;
double K32 = -0.00000000;
double K33 = -121.24598643;
double K34 = 685.23561041;
double K35 =  0.00000000;
double K36 = -437.88622843;
```

C.2 Provided matrix.h Code

This code can be copied into a text file, named "matrix.h", and included in the Arduino's IDE for use on the Main Arduino.

```
//*****************************************************************************
// Initialize Required Variables
//*****************************************************************************

double S11,S12,S13,S21,S22,S23,S31,S32,S33;
double c11,c12,c13,c21,c22,c23,c31,c32,c33;
double d11_2,d12_2,d13_2,d21_2,d22_2,d23_2,d31_2,d32_2,d33_2;
double d11_3,d12_3,d13_3,d21_3,d22_3,d23_3,d31_3,d32_3,d33_3;
double d11_4,d12_4,d13_4,d21_4,d22_4,d23_4,d31_4,d32_4,d33_4;
double d11_5,d12_5,d13_5,d21_5,d22_5,d23_5,d31_5,d32_5,d33_5;
double d11_6,d12_6,d13_6,d21_6,d22_6,d23_6,d31_6,d32_6,d33_6;
double d11_7,d12_7,d13_7,d21_7,d22_7,d23_7,d31_7,d32_7,d33_7;
double d11_8,d12_8,d13_8,d21_8,d22_8,d23_8,d31_8,d32_8,d33_8;
double out11,out12,out13,out21,out22,out23,out31,out32,out33;
double g1,g2,g3;
double norm;

//*****************************************************************************
// Required Mathematical Functions
//*****************************************************************************

// skew function
double skew(double first, double second, double third)
{
  S11 = 0;
  S12 = -third;
  S13 = second;
  S21 = third;
  S22 = 0;
  S23 = -first;
  S31 = -second;
  S32 = first;
  S33 = 0;
  return S11,S12,S13,S21,S22,S23,S31,S32,S33;
}

// three by three matrix multiply function
double mat_mult(double a11, double a12, double a13, double a21, double a22,
    double a23, double a31, double a32, double a33, double b11, double b12,
    double b13, double b21, double b22, double b23, double b31, double b32,
    double b33)
{
  c11 = a11*b11+a12*b21+a13*b31;
  c12 = a11*b12+a12*b22+a13*b32;
  c13 = a11*b13+a12*b23+a13*b33;
  c21 = a21*b11+a22*b21+a23*b31;
  c22 = a21*b12+a22*b22+a23*b32;
  c23 = a21*b13+a22*b23+a23*b33;
  c31 = a31*b11+a32*b21+a33*b31;
```

```
  c32 = a31*b12+a32*b22+a33*b32;
  c33 = a31*b13+a32*b23+a33*b33;
  return c11,c12,c13,c21,c22,c23,c31,c32,c33;
}

// three by three matrix exponential function
double expm(double d11_1, double d12_1, double d13_1, double d21_1,
    double d22_1, double d23_1, double d31_1, double d32_1, double d33_1)
{
  mat_mult(d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_2=c11; d12_2=c12; d13_2=c13; d21_2=c21, d22_2=c22;
  d23_2=c23; d31_2=c31, d32_2=c32, d33_2=c33;

  mat_mult(d11_2,d12_2,d13_2,d21_2,d22_2,d23_2,d31_2,d32_2,d33_2,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_3=c11; d12_3=c12; d13_3=c13; d21_3=c21, d22_3=c22;
  d23_3=c23; d31_3=c31, d32_3=c32, d33_3=c33;

  mat_mult(d11_3,d12_3,d13_3,d21_3,d22_3,d23_3,d31_3,d32_3,d33_3,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_4=c11; d12_4=c12; d13_4=c13; d21_4=c21, d22_4=c22;
  d23_4=c23; d31_4=c31, d32_4=c32, d33_4=c33;

  mat_mult(d11_4,d12_4,d13_4,d21_4,d22_4,d23_4,d31_4,d32_4,d33_4,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_5=c11; d12_5=c12; d13_5=c13; d21_5=c21, d22_5=c22;
  d23_5=c23; d31_5=c31, d32_5=c32, d33_5=c33;

  mat_mult(d11_5,d12_5,d13_5,d21_5,d22_5,d23_5,d31_5,d32_5,d33_5,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_6=c11; d12_6=c12; d13_6=c13; d21_6=c21, d22_6=c22;
  d23_6=c23; d31_6=c31, d32_6=c32, d33_6=c33;

  mat_mult(d11_6,d12_6,d13_6,d21_6,d22_6,d23_6,d31_6,d32_6,d33_6,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_7=c11; d12_7=c12; d13_7=c13; d21_7=c21, d22_7=c22;
  d23_7=c23; d31_7=c31, d32_7=c32, d33_7=c33;

  mat_mult(d11_7,d12_7,d13_7,d21_7,d22_7,d23_7,d31_7,d32_7,d33_7,
           d11_1,d12_1,d13_1,d21_1,d22_1,d23_1,d31_1,d32_1,d33_1);

  d11_8=c11; d12_8=c12; d13_8=c13; d21_8=c21, d22_8=c22;
  d23_8=c23; d31_8=c31, d32_8=c32, d33_8=c33;

  out11 = 1 + d11_1 + d11_2/2 + d11_3/6 + d11_4/24 + d11_5/120
            + d11_6/720 + d11_7/5040 + d11_8/40320;
  out12 =     d12_1 + d12_2/2 + d12_3/6 + d12_4/24 + d12_5/120
```

```
            + d12_6/720 + d12_7/5040 + d12_8/40320;
  out13 =     d13_1 + d13_2/2 + d13_3/6 + d13_4/24 + d13_5/120
            + d13_6/720 + d13_7/5040 + d13_8/40320;

  out21 =     d21_1 + d21_2/2 + d21_3/6 + d21_4/24 + d21_5/120
            + d21_6/720 + d21_7/5040 + d21_8/40320;
  out22 = 1 + d22_1 + d22_2/2 + d22_3/6 + d22_4/24 + d22_5/120
            + d22_6/720 + d22_7/5040 + d22_8/40320;
  out23 =     d23_1 + d23_2/2 + d23_3/6 + d23_4/24 + d23_5/120
            + d23_6/720 + d23_7/5040 + d23_8/40320;

  out31 =     d31_1 + d31_2/2 + d31_3/6 + d31_4/24 + d31_5/120
            + d31_6/720 + d31_7/5040 + d31_8/40320;
  out32 =     d32_1 + d32_2/2 + d32_3/6 + d32_4/24 + d32_5/120
            + d32_6/720 + d32_7/5040 + d32_8/40320;
  out33 = 1 + d33_1 + d33_2/2 + d33_3/6 + d33_4/24 + d33_5/120
            + d33_6/720 + d33_7/5040 + d33_8/40320;

  return out11,out12,out13,out21,out22,out23,out31,out32,out33;
}

// three by one vector cross product function
double cross(double e1, double e2, double e3, double f1, double f2, double f3)
{
  g1 = e2*f3-e3*f2;
  g2 = e3*f1-e1*f3;
  g3 = e1*f2-e2*f1;
  return g1,g2,g3;
}

// three by one vector magnitude function
double mag(double h1, double h2, double h3)
{
 norm = sqrt(h1*h1+h2*h2+h3*h3);
 return norm;
}
```

C.3 Provided functions.h Code

This code can be copied into a text file, named "functions.h", and included in the Arduino's IDE for use on the Main Arduino.

```
//*****************************************************************************
// Initialize Required Byte Manipulations and Pins
//*****************************************************************************
#define UBLB16(a,b)  ( (a << 8)|(b) )&0xFFFF;
#define UBLB14(a,b)  (((a << 8)|(b))&0x3000);
#define UBLB12(a,b)  ( (a << 8)|(b) )&0x0FFF;
#define UBLB2(a,b)   ( (a << 8)|(b) )&0x8000;
#define UBLB15(a,b)  ( (a << 8)|(b) )&0x7FFF;

#define RESETPIN 45//reset
#define DATAREADY 47//drdy
#define SLAVESELECTMAG 49//ss
#define DATAIN 50//MISO
#define DATAOUT 51//MOSI
#define SPICLOCK  52//sck
#define SLAVESELECT 53//ss

//*****************************************************************************
// Initialize Required Variables
//*****************************************************************************

// initialize all IMU read variables
byte globe_name=B00111110;  // Read everything
byte supply_b1,supply_b2,x_gyro_b1,x_gyro_b2,y_gyro_b1,y_gyro_b2,z_gyro_b1;
byte z_gyro_b2,x_acc_b1,x_acc_b2,y_acc_b1,y_acc_b2,z_acc_b1,z_acc_b2;
byte x_temp_b1,x_temp_b2,y_temp_b1,y_temp_b2,z_temp_b1,z_temp_b2;
byte aux_b1,aux_b2,clr;

// initialize all magnetometer variables
byte x_data1,x_data2,y_data1,y_data2,z_data1,z_data2;
int sign,mag_abs,mag_val;
int x_mag,y_mag,z_mag;
int x_mag_ref,y_mag_ref,z_mag_ref;
long x_mag_hold=0,y_mag_hold=0,z_mag_hold=0;

double x_pos_gain = 1.298;
double x_neg_gain = 1.088;
double y_pos_gain = 1.000;
double y_neg_gain = 1.060;
double z_bias;

// initialize reaction wheel variables
double RW1_ComSpeed,RW2_ComSpeed,RW3_ComSpeed;
double RW4_ComSpeed = 0;
double RW1_ComSpeed_abs,RW2_ComSpeed_abs,RW3_ComSpeed_abs;
double RW4_ComSpeed_abs = 0;
double ComSpeed1,ComSpeed2,ComSpeed3;
double ComSpeed4 = 0;
long ComDir1,ComDir2,ComDir3;
```

```
long ComDir4 = 0xC8;
long MotorSpeed1 = 0;
long MotorSpeed2 = 0;
long MotorSpeed3 = 0;
long MotorSpeed4 = 0;
long MotorDir1 = 0xC0;
long MotorDir2 = 0xC8;
long MotorDir3 = 0xC0;
long MotorDir4 = 0xC8;

//*****************************************************************************
// Define Required Functions
//*****************************************************************************

// define spi write to IMU function
byte write_IMU(byte first, byte second)
{
  SPCR = B01011101;

  digitalWrite(SLAVESELECT,LOW);
  SPI.transfer(first);
  SPI.transfer(second);
  digitalWrite(SLAVESELECT,HIGH);
}

// define spi global read to IMU function
byte globe_read(byte globe_name)
{
  SPCR = B01011101;

  digitalWrite(SLAVESELECT,LOW);
  clr = SPI.transfer(globe_name);
  clr = SPI.transfer(B00000000);
  supply_b1 = SPI.transfer(B00000000);
  supply_b2 = SPI.transfer(B00000000);
  x_gyro_b1 = SPI.transfer(B00000000);
  x_gyro_b2 = SPI.transfer(B00000000);
  y_gyro_b1 = SPI.transfer(B00000000);
  y_gyro_b2 = SPI.transfer(B00000000);
  z_gyro_b1 = SPI.transfer(B00000000);
  z_gyro_b2 = SPI.transfer(B00000000);
  x_acc_b1 = SPI.transfer(B00000000);
  x_acc_b2 = SPI.transfer(B00000000);
  y_acc_b1 = SPI.transfer(B00000000);
  y_acc_b2 = SPI.transfer(B00000000);
  z_acc_b1 = SPI.transfer(B00000000);
  z_acc_b2 = SPI.transfer(B00000000);
  x_temp_b1 = SPI.transfer(B00000000);
  x_temp_b2 = SPI.transfer(B00000000);
  y_temp_b1 = SPI.transfer(B00000000);
  y_temp_b2 = SPI.transfer(B00000000);
  z_temp_b1 = SPI.transfer(B00000000);
  z_temp_b2 = SPI.transfer(B00000000);
  aux_b1 = SPI.transfer(B00000000);
```

```
  aux_b2 = SPI.transfer(B00000000);
  digitalWrite(SLAVESELECT,HIGH);
  return supply_b1,supply_b2,x_gyro_b1,x_gyro_b2,y_gyro_b1,y_gyro_b2,z_gyro_b1,
         z_gyro_b2,x_acc_b1,x_acc_b2,y_acc_b1,y_acc_b2,z_acc_b1,z_acc_b2,
         x_temp_b1,x_temp_b2,y_temp_b1,y_temp_b2,z_temp_b1,z_temp_b2,
         aux_b1,aux_b2;
}

// define spi write to magnetometer function
byte read_mag()
{
  SPCR = B01010001;

  digitalWrite(SLAVESELECTMAG,LOW);

  digitalWrite(RESETPIN,HIGH);
  digitalWrite(RESETPIN,LOW);

  SPI.transfer(B01000001);
  while(digitalRead(DATAREADY) == LOW)
  {}
  x_data1 = SPI.transfer(B00000000);
  x_data2 = SPI.transfer(B00000000);

  digitalWrite(RESETPIN,HIGH);
  digitalWrite(RESETPIN,LOW);

  SPI.transfer(B01000010);
  while(digitalRead(DATAREADY) == LOW)
  {}
  y_data1 = SPI.transfer(B00000000);
  y_data2 = SPI.transfer(B00000000);

  digitalWrite(RESETPIN,HIGH);
  digitalWrite(RESETPIN,LOW);

  SPI.transfer(B01000011);
  while(digitalRead(DATAREADY) == LOW)
  {}
  z_data1 = SPI.transfer(B00000000);
  z_data2 = SPI.transfer(B00000000);

  digitalWrite(SLAVESELECTMAG, HIGH);

  return x_data1,x_data2,y_data1,y_data2,z_data1,z_data2;
}

// define function to decode raw magnetometer data
int decode_mag(byte first,byte second)
{
  sign = UBLB2(first,second);
  mag_abs = UBLB15(first,second);

  if (sign == 0x8000)
```

```
  {
    mag_val = mag_abs-32768;
  }
  else
  {
    mag_val = mag_abs;
  }

  return mag_val;
}

// define function to initialize magnetic field vector
byte init_mag()
{
   for (int i = 1; i < 101; i++)
   {
    read_mag();

    decode_mag(x_data1,x_data2);
    x_mag_hold = x_mag_hold+mag_val;


    decode_mag(y_data1,y_data2);
    y_mag_hold = y_mag_hold+mag_val;


    decode_mag(z_data1,z_data2);
    z_mag_hold = z_mag_hold+mag_val;

    delay(50);
   }

  x_mag_ref = x_mag_hold/100;
  y_mag_ref = -y_mag_hold/100;

  z_bias = -z_mag_hold/100;
  z_mag_ref = 0;


  return x_mag_ref,y_mag_ref,z_mag_ref;
}

// set reaction wheels to initial commanded speed
double init_rw(double x_init,double y_init,double z_init)
{
    RW1_ComSpeed = x_init;
    RW2_ComSpeed = y_init;
    RW3_ComSpeed = z_init;

    // absolute value of commanded reaction wheel angular rate
    RW1_ComSpeed_abs = abs(x_init);
    RW2_ComSpeed_abs = abs(y_init);
    RW3_ComSpeed_abs = abs(z_init);
```

```
// convert from radians per second to motor controller byte representation
ComSpeed1 = map(RW1_ComSpeed_abs, 0, 194, 0, 127);
ComSpeed2 = map(RW2_ComSpeed_abs, 0, 194, 0, 127);
ComSpeed3 = map(RW3_ComSpeed_abs, 0, 194, 0, 127);

// determine reaction wheel direction command for motor controller
if (RW1_ComSpeed >=0)
{
  ComDir1 = 0xC2;
}
else
{
  ComDir1 = 0xC1;
}

if (RW2_ComSpeed >=0)
{
  ComDir2 = 0xCA;
}
else
{
  ComDir2 = 0xC9;
}

if (RW3_ComSpeed >=0)
{
  ComDir3 = 0xC2;
}
else
{
  ComDir3 = 0xC1;
}

Serial1.write(ComDir1);
Serial1.write(ComSpeed1);

Serial1.write(ComDir2);
Serial1.write(ComSpeed2);

Serial1.write(ComDir3);
Serial1.write(ComSpeed3);

Serial1.write(ComDir4);
Serial1.write(ComSpeed4);

MotorDir1 = ComDir1;
MotorSpeed1 = ComSpeed1;
MotorDir1 = ComDir2;
MotorSpeed1 = ComSpeed2;
MotorDir1 = ComDir3;
MotorSpeed1 = ComSpeed3;
MotorDir1 = ComDir4;
MotorSpeed1 = ComSpeed4;
```

```
    Serial1.flush();

    delay(10000);
}

// print desired variables
// post_flag definition
//     1 - Print variable followed by one tab
//     2 - Print variable followed by two tabs
//     3 - Print variable and go to new line
double serial_out(double output, int post_flag)
{
  if (post_flag == 1)
  {
    Serial.print(output);
    Serial.print("\t");
  }
  if (post_flag == 2)
  {
    Serial.print(output);
    Serial.print("\t");
    Serial.print("\t");
  }
  if (post_flag == 3)
  {
    Serial.println(output);
  }
}

// similar to serial_out, but prints space after var for serial_label function
double serial_post(int post_flag)
{
  if (post_flag == 1)
  {
    Serial.print("\t");
  }
  if (post_flag == 2)
  {
    Serial.print("\t");
    Serial.print("\t");
  }
  if (post_flag == 3)
  {
    Serial.println(" ");
  }
}

// print labels to serial port
double serial_label()
{
  if (timekeep_flag == 1)         {Serial.print("time");
                                   serial_post(timekeep_post);}
  if (RealDeltaT_flag == 1)       {Serial.print("dt");
                                   serial_post(RealDeltaT_post);}
```

```
if (DeltaT_flag == 1)          {Serial.print("dt cnts");
                                 serial_post(DeltaT_post);}
if (RW1_ComSpeed_flag == 1)     {Serial.print("RW1 com");
                                 serial_post(RW1_ComSpeed_post);}
if (RPS1_flag == 1)             {Serial.print("RW1 RPS");
                                 serial_post(RPS1_post);}
if (RW1_Volt_flag == 1)         {Serial.print("RW1 vlts");
                                 serial_post(RW1_Volt_post);}
if (RW2_ComSpeed_flag == 1)     {Serial.print("RW2 com");
                                 serial_post(RW2_ComSpeed_post);}
if (RPS2_flag == 1)             {Serial.print("RW2 RPS");
                                 serial_post(RPS2_post);}
if (RW2_Volt_flag == 1)         {Serial.print("RW2 vlts");
                                 serial_post(RW2_Volt_post);}
if (RW3_ComSpeed_flag == 1)     {Serial.print("RW3 com");
                                 serial_post(RW3_ComSpeed_post);}
if (RPS3_flag == 1)             {Serial.print("RW3 RPS");
                                 serial_post(RPS3_post);}
if (RW3_Volt_flag == 1)         {Serial.print("RW3 vlts");
                                 serial_post(RW3_Volt_post);}
if (x_ComPos_flag == 1)         {Serial.print("xComPos");
                                 serial_post(x_ComPos_post);}
if (y_ComPos_flag == 1)         {Serial.print("yComPos");
                                 serial_post(y_ComPos_post);}
if (z_ComPos_flag == 1)         {Serial.print("zComPos");
                                 serial_post(z_ComPos_post);}
if (x_ComRate_flag == 1)        {Serial.print("xCmRate");
                                 serial_post(x_ComRate_post);}
if (y_ComRate_flag == 1)        {Serial.print("yCmRate");
                                 serial_post(y_ComRate_post);}
if (z_ComRate_flag == 1)        {Serial.print("zCmRate");
                                 serial_post(z_ComRate_post);}
if (tx_p_flag == 1)             {Serial.print("xEstPos");
                                 serial_post(tx_p_post);}
if (ty_p_flag == 1)             {Serial.print("yEstPos");
                                 serial_post(ty_p_post);}
if (tz_p_flag == 1)             {Serial.print("zEstPos");
                                 serial_post(tz_p_post);}
if (wx_p_flag == 1)             {Serial.print("xEsRate");
                                 serial_post(wx_p_post);}
if (wy_p_flag == 1)             {Serial.print("yEsRate");
                                 serial_post(wy_p_post);}
if (wz_p_flag == 1)             {Serial.print("zEsRate");
                                 serial_post(wz_p_post);}
if (wx_dot_p_flag == 1)         {Serial.print("xEstAcc");
                                 serial_post(wx_dot_p_post);}
if (wy_dot_p_flag == 1)         {Serial.print("yEstAcc");
                                 serial_post(wy_dot_p_post);}
if (wz_dot_p_flag == 1)         {Serial.print("zEstAcc");
                                 serial_post(wz_dot_p_post);}
if (RPS4_flag == 1)             {Serial.print("RW4 RPS");
                                 serial_post(RPS4_post);}
if (x_gyro_flag == 1)           {Serial.print("rawIMUx");
                                 serial_post(x_gyro_post);}
```

```
if (y_gyro_flag == 1)          {Serial.print("rawIMUy");
                                serial_post(y_gyro_post);}
if (z_gyro_flag == 1)          {Serial.print("rawIMUz");
                                serial_post(z_gyro_post);}
if (x_MeasRate_flag == 1)      {Serial.print("xRteMea");
                                serial_post(x_MeasRate_post);}
if (y_MeasRate_flag == 1)      {Serial.print("yRteMea");
                                serial_post(y_MeasRate_post);}
if (z_MeasRate_flag == 1)      {Serial.print("zRteMea");
                                serial_post(z_MeasRate_post);}
if (x_MeasPos_flag == 1)       {Serial.print("IMUxPos");
                                serial_post(x_MeasPos_post);}
if (y_MeasPos_flag == 1)       {Serial.print("IMUyPos");
                                serial_post(y_MeasPos_post);}
if (z_MeasPos_flag == 1)       {Serial.print("IMUzPos");
                                serial_post(z_MeasPos_post);}
if (x_acc_flag == 1)           {Serial.print("xGrvMea");
                                serial_post(x_acc_post);}
if (y_acc_flag == 1)           {Serial.print("yGrvMea");
                                serial_post(y_acc_post);}
if (z_acc_flag == 1)           {Serial.print("zGrvMea");
                                serial_post(z_acc_post);}
if (x_mag_flag == 1)           {Serial.print("rawMagx");
                                serial_post(x_mag_post);}
if (y_mag_flag == 1)           {Serial.print("rawMagy");
                                serial_post(y_mag_post);}
if (z_mag_flag == 1)           {Serial.print("rawMagz");
                                serial_post(z_mag_post);}
if (x_Mag_flag == 1)           {Serial.print("xMagMea");
                                serial_post(x_Mag_post);}
if (y_Mag_flag == 1)           {Serial.print("yMagMea");
                                serial_post(y_Mag_post);}
if (z_Mag_flag == 1)           {Serial.print("zMagMea");
                                serial_post(z_Mag_post);}
if (Wx_est_flag == 1)          {Serial.print("RW1SSEs");
                                serial_post(Wx_est_post);}
if (Wy_est_flag == 1)          {Serial.print("RW2SSEs");
                                serial_post(Wy_est_post);}
if (Wz_est_flag == 1)          {Serial.print("RW3SSEs");
                                serial_post(Wz_est_post);}
if (x_grav_flag == 1)          {Serial.print("xGrvCom");
                                serial_post(x_grav_post);}
if (y_grav_flag == 1)          {Serial.print("yGrvCom");
                                serial_post(y_grav_post);}
if (z_grav_flag == 1)          {Serial.print("zGrvCom");
                                serial_post(z_grav_post);}
if (x_grav_pos_flag == 1)      {Serial.print("AccxPos");
                                serial_post(x_grav_pos_post);}
if (y_grav_pos_flag == 1)      {Serial.print("AccyPos");
                                serial_post(y_grav_pos_post);}
if (z_grav_pos_flag == 1)      {Serial.print("AcczPos");
                                serial_post(z_grav_pos_post);}
if (x_mag_cmd_flag == 1)       {Serial.print("xMagCom");
                                serial_post(x_mag_cmd_post);}
```

```
  if (y_mag_cmd_flag == 1)        {Serial.print("yMagCom");
                                   serial_post(y_mag_cmd_post);}
  if (z_mag_cmd_flag == 1)        {Serial.print("zMagCom");
                                   serial_post(z_mag_cmd_post);}
  if (x_mag_pos_flag == 1)        {Serial.print("MagxPos");
                                   serial_post(x_mag_pos_post);}
  if (y_mag_pos_flag == 1)        {Serial.print("MagyPos");
                                   serial_post(y_mag_pos_post);}
  if (z_mag_pos_flag == 1)        {Serial.print("MagzPos");
                                   serial_post(z_mag_pos_post);}
}
```

C.4 Provided mega_main.pde Code

This code can be copied directly into the Arduino's IDE, named "mega_main.pde" and uploaded to the Main Arduino. Be sure to include the three required header files listed above before uploading to the Main Arduino. Remember to update the commanded air bearing angular orientation and rate as necessary.

```
//*****************************************************************************
// Set Desired Outputs
//*****************************************************************************

//  Flag instructions                Post instruction
//     0 - Do Not Print                 1 - Print with one tab after value
//     1 - Print                        2 - Print with two tabs after value
//                                      3 - Print with new line after value

int timekeep_flag     = 1;          int timekeep_post     = 1;
int RealDeltaT_flag   = 1;          int RealDeltaT_post   = 2;
int DeltaT_flag       = 0;         int DeltaT_post       = 2;

int RW1_ComSpeed_flag = 1;          int RW1_ComSpeed_post = 1;
int RPS1_flag         = 1;          int RPS1_post         = 2;
int RW1_Volt_flag     = 0;          int RW1_Volt_post     = 2;

int RW2_ComSpeed_flag = 1;          int RW2_ComSpeed_post = 1;
int RPS2_flag         = 1;          int RPS2_post         = 2;
int RW2_Volt_flag     = 0;          int RW2_Volt_post     = 2;

int RW3_ComSpeed_flag = 1;          int RW3_ComSpeed_post = 1;
int RPS3_flag         = 1;          int RPS3_post         = 2;
int RW3_Volt_flag     = 0;          int RW3_Volt_post     = 2;

int x_ComPos_flag     = 1;          int x_ComPos_post     = 1;
int y_ComPos_flag     = 1;          int y_ComPos_post     = 1;
int z_ComPos_flag     = 1;          int z_ComPos_post     = 2;

int x_ComRate_flag    = 1;          int x_ComRate_post    = 1;
int y_ComRate_flag    = 1;          int y_ComRate_post    = 1;
int z_ComRate_flag    = 1;          int z_ComRate_post    = 2;

int tx_p_flag         = 1;          int tx_p_post         = 1;
int ty_p_flag         = 1;          int ty_p_post         = 1;
int tz_p_flag         = 1;          int tz_p_post         = 2;

int wx_p_flag         = 1;          int wx_p_post         = 1;
int wy_p_flag         = 1;          int wy_p_post         = 1;
int wz_p_flag         = 1;          int wz_p_post         = 3;

int wx_dot_p_flag     = 0;          int wx_dot_p_post     = 1;
int wy_dot_p_flag     = 0;          int wy_dot_p_post     = 1;
int wz_dot_p_flag     = 0;          int wz_dot_p_post     = 2;

int RPS4_flag         = 0;          int RPS4_post         = 2;
```

```
int x_gyro_flag        = 0;        int x_gyro_post        = 1;
int y_gyro_flag        = 0;        int y_gyro_post        = 1;
int z_gyro_flag        = 0;        int z_gyro_post        = 2;

int x_MeasRate_flag    = 0;        int x_MeasRate_post    = 1;
int y_MeasRate_flag    = 0;        int y_MeasRate_post    = 1;
int z_MeasRate_flag    = 0;        int z_MeasRate_post    = 2;

int x_MeasPos_flag     = 0;        int x_MeasPos_post     = 1;
int y_MeasPos_flag     = 0;        int y_MeasPos_post     = 1;
int z_MeasPos_flag     = 0;        int z_MeasPos_post     = 2;

int x_acc_flag         = 0;        int x_acc_post         = 1;
int y_acc_flag         = 0;        int y_acc_post         = 1;
int z_acc_flag         = 0;        int z_acc_post         = 3;

int x_mag_flag         = 0;        int x_mag_post         = 1;
int y_mag_flag         = 0;        int y_mag_post         = 1;
int z_mag_flag         = 0;        int z_mag_post         = 2;

int x_Mag_flag         = 0;        int x_Mag_post         = 1;
int y_Mag_flag         = 0;        int y_Mag_post         = 1;
int z_Mag_flag         = 0;        int z_Mag_post         = 3;

int Wx_est_flag        = 0;        int Wx_est_post        = 1;
int Wy_est_flag        = 0;        int Wy_est_post        = 1;
int Wz_est_flag        = 0;        int Wz_est_post        = 3;

int x_grav_flag        = 0;        int x_grav_post        = 1;
int y_grav_flag        = 0;        int y_grav_post        = 1;
int z_grav_flag        = 0;        int z_grav_post        = 3;

int x_grav_pos_flag    = 0;        int x_grav_pos_post    = 1;
int y_grav_pos_flag    = 0;        int y_grav_pos_post    = 1;
int z_grav_pos_flag    = 0;        int z_grav_pos_post    = 2;

int x_mag_cmd_flag     = 0;        int x_mag_cmd_post     = 1;
int y_mag_cmd_flag     = 0;        int y_mag_cmd_post     = 1;
int z_mag_cmd_flag     = 0;        int z_mag_cmd_post     = 2;

int x_mag_pos_flag     = 0;        int x_mag_pos_post     = 1;
int y_mag_pos_flag     = 0;        int y_mag_pos_post     = 1;
int z_mag_pos_flag     = 0;        int z_mag_pos_post     = 3;

//*****************************************************************************
// Initialize Required Variables and Header Files
//*****************************************************************************
#include <Wire.h>
#include <SPI.h>
#include "matrix.h"
#include "init.h"
#include "functions.h"

int progPin = 9;
```

```
// define gyroscope and accelerometer gains
double gyro_gain = 0.0125;
double acc_gain = 0.00333;

double RAD = 3.141593/180;
double DEG = 180/3.141594;

int x_gyro_sign = 0;
double x_gyro;
int y_gyro_sign = 0;
double y_gyro;
int z_gyro_sign = 0;
double z_gyro;
int x_acc_sign = 0;
double x_acc;
int y_acc_sign = 0;
double y_acc;
int z_acc_sign = 0;
double z_acc;

// initialize time variables
long timercounter = 0;
double freq = 4096;
double timekeep = 0;
double DeltaT = 0;
double RealDeltaT = 0;

byte encode1_b1 = 0;
byte encode1_b2 = 0;
byte encode1_b3 = 0;
byte encode1_b4 = 0;

byte encode2_b1 = 0;
byte encode2_b2 = 0;
byte encode2_b3 = 0;
byte encode2_b4 = 0;

byte encode3_b1 = 0;
byte encode3_b2 = 0;
byte encode3_b3 = 0;
byte encode3_b4 = 0;

byte encode4_b1 = 0;
byte encode4_b2 = 0;
byte encode4_b3 = 0;
byte encode4_b4 = 0;

long Encoder_RW1,Encoder_RW2,Encoder_RW3,Encoder_RW4;
long Encoder_RW1_prev,Encoder_RW2_prev,Encoder_RW3_prev,Encoder_RW4_prev;
double ThresSpeed1,ThresSpeed2,ThresSpeed3;
double ThresSpeed4 = 0;
double RPS1,RPS2,RPS3,RPS4;
double RPM1,RPM2,RPM3,RPM4;
```

```
double x_MeasPos,y_MeasPos,z_MeasPos;
double x_MeasPosPrev,y_MeasPosPrev,z_MeasPosPrev;
double x_MeasRate,y_MeasRate,z_MeasRate;
double Wx_est,Wy_est,Wz_est,ix_est,iy_est,iz_est;
double Wx_dot_est,Wy_dot_est,Wz_dot_est;
double Delta_x_pos,Delta_y_pos,Delta_z_pos;
double Delta_11,Delta_12,Delta_13,Delta_21,Delta_22,Delta_23;
double Delta_31,Delta_32,Delta_33;
double DCM_11,DCM_12,DCM_13,DCM_21,DCM_22,DCM_23,DCM_31,DCM_32,DCM_33;
double d_DCM_11,d_DCM_12,d_DCM_13,d_DCM_21,d_DCM_22,d_DCM_23;
double d_DCM_31,d_DCM_32,d_DCM_33;
double x_grav,y_grav,z_grav;
double x_grav_pos,y_grav_pos,z_grav_pos;
double x_grav_innov,y_grav_innov,z_grav_innov;
double x_mag_cmd,y_mag_cmd,z_mag_cmd;
double x_mag_pos,y_mag_pos,z_mag_pos;
double x_Mag,y_Mag,z_Mag;
double norm_mag_cmd,norm_Mag;
double x_mag_innov,y_mag_innov,z_mag_innov;
double x_pos_innov,y_pos_innov,z_pos_innov;
double x_rate_innov,y_rate_innov,z_rate_innov;
double tx_n,ty_n,tz_n,wx_n,wy_n,wz_n;
double tx_p,ty_p,tz_p,wx_p,wy_p,wz_p;
double wx_dot_p,wy_dot_p,wz_dot_p;
double Wx_cmd,Wy_cmd,Wz_cmd;
double Wx_dot_cmd,Wy_dot_cmd,Wz_dot_cmd;
double Wx_dot_cmd_lqr,Wy_dot_cmd_lqr,Wz_dot_cmd_lqr;
double Wx_dot_cmd_rd,Wy_dot_cmd_rd,Wz_dot_cmd_rd;
double wx_dot_cmd_rd,wy_dot_cmd_rd,wz_dot_cmd_rd;
double Wx_dot_cmd_cl,Wy_dot_cmd_cl,Wz_dot_cmd_cl;
double Wx_cmd_cl,Wy_cmd_cl,Wz_cmd_cl;
double Wx_cmd_cl_Prev = 0;
double Wy_cmd_cl_Prev = 0;
double Wz_cmd_cl_Prev = 0;
double Wx_cmd_ff,Wy_cmd_ff,Wz_cmd_ff;
double Wx_dot_cmd_ff,Wy_dot_cmd_ff,Wz_dot_cmd_ff;
double wx_dot_cmd,wy_dot_cmd,wz_dot_cmd;
double x_ComPos,y_ComPos,z_ComPos;
double x_ComPosPrev,y_ComPosPrev,z_ComPosPrev;
double x_ComRate,y_ComRate,z_ComRate;
double x_ComRate_Prev,y_ComRate_Prev,z_ComRate_Prev;
double x_ComAcc,y_ComAcc,z_ComAcc;
double x_ErrPos,y_ErrPos,z_ErrPos;
double x_ErrRate,y_ErrRate,z_ErrRate;
double x_ErrAcc,y_ErrAcc,z_ErrAcc;
double RW1_Volt,RW2_Volt,RW3_Volt;

double D11 = cos(65.9052*RAD);
double D12 = cos(65.9052*RAD);
double D13 = cos(144.7356*RAD);
double D21 = cos(135*RAD);
double D22 = cos(45*RAD);
double D23 = cos(90*RAD);
double D31 = cos(54.7356*RAD);
```

```
double D32 = cos(54.7356*RAD);
double D33 = cos(54.7356*RAD);

//*****************************************************************************
// Initialize Estimation/Control Loop
//*****************************************************************************
void setup()
{
  Serial.begin(57600);        // port between main arduino and GS computer
  Serial1.begin(115200);      // port between main arduino and aux arduino

  Serial.flush();
  Serial.println("Initializing...");  Serial.println(" ");

  Wire.begin();

  Wire.beginTransmission(0x68); // activate DS1307
  Wire.send(0);                 // where to begin
  Wire.send(0x00);              //seconds
  Wire.send(0x00);              //minutes
  Wire.send(0x80);              //hours (24hr time)
  Wire.send(0x01);              // Day 01-07
  Wire.send(0x01);              // Date 0-31
  Wire.send(0x01);              // month 0-12
  Wire.send(0x10);              // Year 00-99
  Wire.send(0x11);
  Wire.endTransmission();

  pinMode(progPin, OUTPUT);

  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  pinMode(SLAVESELECTMAG,OUTPUT);
  pinMode(DATAREADY, INPUT);
  pinMode(RESETPIN, OUTPUT);

  digitalWrite(SLAVESELECT,HIGH);
  digitalWrite(SLAVESELECTMAG, HIGH);
  digitalWrite(RESETPIN, LOW);

  clr=SPSR;
  clr=SPDR;

  delay(10);
  write_IMU(B10110110,B00000001);    // set internal sample rate - default
  delay(10);
  write_IMU(B10111001,B00000001);    // set measurement range - 75 deg/sec
  delay(10);
  write_IMU(B10111000,B00000100);    // set number of poles - 16

  // determine initial magnetic field vector
  init_mag();
```

```
  // set initial reaction wheel angular velocity
  double rw_init_mag;
  rw_init_mag = sqrt(Wx_cmd_ff_Prev*Wx_cmd_ff_Prev
     +Wy_cmd_ff_Prev*Wy_cmd_ff_Prev+Wz_cmd_ff_Prev*Wz_cmd_ff_Prev);
  if (rw_init_mag > 0)
  {
    init_rw(Wx_cmd_ff_Prev,Wy_cmd_ff_Prev,Wz_cmd_ff_Prev);
  }

  // print label to serial port
  serial_label();

  // initialize interrupts for timing module
  attachInterrupt(0, timer2, RISING);
  interrupts();
}

//*****************************************************************************
// Begin Estimation/Control Loop
//*****************************************************************************
void loop()
{
  if (timercounter >= timestep)
  {
    digitalWrite(progPin, HIGH);

    timekeep = timekeep+timercounter/freq;
    DeltaT =  timercounter;
    timercounter = 0;
    RealDeltaT = DeltaT/freq;

    //*****************************************************************************
    // Write Commanded Reaction Wheel Angular Rate to Motor Controllers
    //*****************************************************************************

    Serial1.write(MotorDir1);
    Serial1.write(MotorSpeed1);

    Serial1.write(MotorDir2);
    Serial1.write(MotorSpeed2);

    Serial1.write(MotorDir3);
    Serial1.write(MotorSpeed3);

    Serial1.write(MotorDir4);
    Serial1.write(MotorSpeed4);

    //*****************************************************************************
    // Read IMU
    //*****************************************************************************

    // read all variables from IMU
    globe_read(globe_name);
```

```
x_gyro_sign = UBLB14(x_gyro_b1,x_gyro_b2);
x_gyro = UBLB12(x_gyro_b1,x_gyro_b2);

if (x_gyro_sign == 0) {
  x_gyro = x_gyro*gyro_gain;
}
else {
  x_gyro = (x_gyro-4096)*gyro_gain;
}

y_gyro_sign = UBLB14(y_gyro_b1,y_gyro_b2);
y_gyro = UBLB12(y_gyro_b1,y_gyro_b2);

if (y_gyro_sign == 0) {
  y_gyro = y_gyro*gyro_gain;
}
else {
  y_gyro = (y_gyro-4096)*gyro_gain;
}

z_gyro_sign = UBLB14(z_gyro_b1,z_gyro_b2);
z_gyro = UBLB12(z_gyro_b1,z_gyro_b2);

if (z_gyro_sign == 0) {
  z_gyro = z_gyro*gyro_gain;
}
else {
  z_gyro = (z_gyro-4096)*gyro_gain;
}

x_acc_sign = UBLB14(x_acc_b1,x_acc_b2);
x_acc = UBLB12(x_acc_b1,x_acc_b2);

if (x_acc_sign == 0) {
  x_acc = x_acc*acc_gain;
}
else {
  x_acc = (x_acc-4096)*acc_gain;
}

y_acc_sign = UBLB14(y_acc_b1,y_acc_b2);
y_acc = UBLB12(y_acc_b1,y_acc_b2);

if (y_acc_sign == 0) {
  y_acc = y_acc*acc_gain;
}
else {
  y_acc = (y_acc-4096)*acc_gain;
}

z_acc_sign = UBLB14(z_acc_b1,z_acc_b2);
z_acc = UBLB12(z_acc_b1,z_acc_b2);
```

```
  if (z_acc_sign == 0) {
    z_acc = z_acc*acc_gain;
  }
  else {
    z_acc = (z_acc-4096)*acc_gain;
  }

  //*************************************************************************
  // Read Magnetometer
  //*************************************************************************

  read_mag();

  decode_mag(x_data1,x_data2);
  x_mag = mag_val;

  decode_mag(y_data1,y_data2);
  y_mag = mag_val;

  decode_mag(z_data1,z_data2);
  z_mag = mag_val;

  //*************************************************************************
  // Commanded Angular Orientation and Commanded Angular Rate
  //*************************************************************************

  // commanded angular rate in radians per second
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = 0;


if (timekeep >= 30)
  {
    x_ComRate = 0;
    y_ComRate = 0;
    z_ComRate = 0.007854*(timekeep-30);
  }

  if (timekeep >= 40)
  {
    x_ComRate = 0;
    y_ComRate = 0;
    z_ComRate = 0.07854;
  }

  if (timekeep >= 50)
  {
    x_ComRate = 0;
    y_ComRate = 0;
    z_ComRate = 0.07854-(0.007854*(timekeep-50));
  }

  if (timekeep >= 60)
```

```
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = 0;
}

if (timekeep >= 80)
{
  x_ComRate = 0;
  y_ComRate = 0.000873*(timekeep-80);
  z_ComRate = 0;
}

if (timekeep >= 90)
{
  x_ComRate = 0;
  y_ComRate = 0.008730;
  z_ComRate = 0;
}

if (timekeep >= 100)
{
  x_ComRate = 0;
  y_ComRate = 0.008730-(0.000873*(timekeep-100));
  z_ComRate = 0;
}

if (timekeep >= 110)
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = 0;
}

if (timekeep >= 130)
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = -0.007854*(timekeep-130);
}

if (timekeep >= 140)
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = -0.07854;
}

if (timekeep >= 150)
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = -0.07854+(0.007854*(timekeep-150));
}
```

```
if (timekeep >= 160)
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = 0;
}

if (timekeep >= 180)
{
  x_ComRate = 0.000873*(timekeep-180);
  y_ComRate = 0;
  z_ComRate = 0;
}

if (timekeep >= 190)
{
  x_ComRate = 0.008730;
  y_ComRate = 0;
  z_ComRate = 0;
}

if (timekeep >= 200)
{
  x_ComRate = 0.008730-(0.000873*(timekeep-200));
  y_ComRate = 0;
  z_ComRate = 0;
}

if (timekeep >= 210)
{
  x_ComRate = 0;
  y_ComRate = 0;
  z_ComRate = 0;
}

// commanded angular orientation in radians
x_ComPos = x_ComPosPrev + x_ComRate*RealDeltaT;
y_ComPos = y_ComPosPrev + y_ComRate*RealDeltaT;
z_ComPos = z_ComPosPrev + z_ComRate*RealDeltaT;

x_ComPosPrev = x_ComPos;
y_ComPosPrev = y_ComPos;
z_ComPosPrev = z_ComPos;

//*************************************************************************
// Extended Kalman Filter
//*************************************************************************

// measured angular rate from IMU in radians per second
x_gyro = (x_gyro-0.0486)*RAD;
y_gyro = (y_gyro+0.1246)*RAD;
z_gyro = (z_gyro-0.1312)*RAD;
```

```
x_MeasRate = -x_gyro;
y_MeasRate = -y_gyro;
z_MeasRate = z_gyro;

// measured ang orientation from integrated IMU ang rate meas in rad
x_MeasPos = x_MeasPosPrev+x_MeasRate*RealDeltaT;
y_MeasPos = y_MeasPosPrev+y_MeasRate*RealDeltaT;
z_MeasPos = z_MeasPosPrev+z_MeasRate*RealDeltaT;

x_MeasPosPrev = x_MeasPos;
y_MeasPosPrev = y_MeasPos;
z_MeasPosPrev = z_MeasPos;

// measured linear acceleration (gravity vector) from IMU in g's
x_acc = -x_acc;
y_acc = -y_acc;
z_acc = z_acc;

// measured magnetic field vector from magnetometer in counts
if (x_mag >= 0)   {x_Mag = x_pos_gain*x_mag;}
else              {x_Mag = x_neg_gain*x_mag;}
if (y_mag >= 0)   {y_Mag = -y_neg_gain*y_mag;}
else              {y_Mag = -y_pos_gain*y_mag;}
z_Mag = -z_mag-z_bias;


// reaction wheel state space model
Wx_est = 0.9775*Wx_est_Prev+0.0211*ix_est_Prev+0.3003*RW1_Volt;
Wy_est = 0.9775*Wy_est_Prev+0.0211*iy_est_Prev+0.3003*RW2_Volt;
Wz_est = 0.9775*Wz_est_Prev+0.0211*iz_est_Prev+0.3003*RW3_Volt;

ix_est = -0.0457*Wx_est_Prev-0.001*ix_est_Prev+0.6445*RW1_Volt;
iy_est = -0.0457*Wy_est_Prev-0.001*iy_est_Prev+0.6445*RW2_Volt;
iz_est = -0.0457*Wz_est_Prev-0.001*iz_est_Prev+0.6445*RW3_Volt;

Wx_dot_est = (Wx_est-Wx_est_Prev)/RealDeltaT;
Wy_dot_est = (Wy_est-Wy_est_Prev)/RealDeltaT;
Wz_dot_est = (Wz_est-Wz_est_Prev)/RealDeltaT;

Wx_est_Prev = Wx_est;
Wy_est_Prev = Wy_est;
Wz_est_Prev = Wz_est;

ix_est_Prev = ix_est;
iy_est_Prev = iy_est;
iz_est_Prev = iz_est;

// air bearing state propagation
tx_n = Ad_11*tx_p_Prev +Ad_12*ty_p_Prev +Ad_13*tz_p_Prev+Ad_14*wx_p_Prev
      +Ad_15*wy_p_Prev+Ad_16*wz_p_Prev
      +Bd_11*Wx_dot_est+Bd_12*Wy_dot_est+Bd_13*Wz_dot_est;
ty_n = Ad_21*tx_p_Prev +Ad_22*ty_p_Prev +Ad_23*tz_p_Prev+Ad_24*wx_p_Prev
      +Ad_25*wy_p_Prev+Ad_26*wz_p_Prev
      +Bd_21*Wx_dot_est+Bd_22*Wy_dot_est+Bd_23*Wz_dot_est;
```

```
tz_n = Ad_31*tx_p_Prev +Ad_32*ty_p_Prev +Ad_33*tz_p_Prev+Ad_34*wx_p_Prev
      +Ad_35*wy_p_Prev+Ad_36*wz_p_Prev
      +Bd_31*Wx_dot_est+Bd_32*Wy_dot_est+Bd_33*Wz_dot_est;
wx_n = Ad_41*tx_p_Prev +Ad_42*ty_p_Prev +Ad_43*tz_p_Prev+Ad_44*wx_p_Prev
      +Ad_45*wy_p_Prev+Ad_46*wz_p_Prev
      +Bd_41*Wx_dot_est+Bd_42*Wy_dot_est+Bd_43*Wz_dot_est;
wy_n = Ad_51*tx_p_Prev +Ad_52*ty_p_Prev +Ad_53*tz_p_Prev+Ad_54*wx_p_Prev
      +Ad_55*wy_p_Prev+Ad_56*wz_p_Prev
      +Bd_51*Wx_dot_est+Bd_52*Wy_dot_est+Bd_53*Wz_dot_est;
wz_n = Ad_61*tx_p_Prev +Ad_62*ty_p_Prev +Ad_63*tz_p_Prev+Ad_64*wx_p_Prev
      +Ad_65*wy_p_Prev+Ad_66*wz_p_Prev
      +Bd_61*Wx_dot_est+Bd_62*Wy_dot_est+Bd_63*Wz_dot_est;

// DCM from reference frame to commanded air bearing angular orientation
Delta_x_pos = x_ComRate*RealDeltaT;
Delta_y_pos = y_ComRate*RealDeltaT;
Delta_z_pos = z_ComRate*RealDeltaT;

skew(Delta_x_pos,Delta_y_pos,Delta_z_pos);

Delta_11 = -S11; Delta_12 = -S12; Delta_13 = -S13;
Delta_21 = -S21; Delta_22 = -S22; Delta_23 = -S23;
Delta_31 = -S31; Delta_32 = -S32; Delta_33 = -S33;

expm(Delta_11,Delta_12,Delta_13,Delta_21,Delta_22,Delta_23,Delta_31,
     Delta_32,Delta_33);

d_DCM_11 = out11; d_DCM_12 = out12; d_DCM_13 = out13;
d_DCM_21 = out21; d_DCM_22 = out22; d_DCM_23 = out23;
d_DCM_31 = out31; d_DCM_32 = out32; d_DCM_33 = out33;

mat_mult(d_DCM_11,d_DCM_12,d_DCM_13,d_DCM_21,d_DCM_22,d_DCM_23,d_DCM_31,
         d_DCM_32,d_DCM_33,DCM_11_Prev,DCM_12_Prev,DCM_13_Prev,DCM_21_Prev,
         DCM_22_Prev,DCM_23_Prev,DCM_31_Prev,DCM_32_Prev,DCM_33_Prev);

DCM_11 = c11; DCM_12 = c12; DCM_13 = c13;
DCM_21 = c21; DCM_22 = c22; DCM_23 = c23;
DCM_31 = c31; DCM_32 = c32; DCM_33 = c33;

DCM_11_Prev = DCM_11;  DCM_12_Prev = DCM_12;  DCM_13_Prev = DCM_13;
DCM_21_Prev = DCM_21;  DCM_22_Prev = DCM_22;  DCM_23_Prev = DCM_23;
DCM_31_Prev = DCM_31;  DCM_32_Prev = DCM_32;  DCM_33_Prev = DCM_33;

// innovation from rate measurement;
x_pos_innov = x_MeasPos-tx_n;
y_pos_innov = y_MeasPos-ty_n;
z_pos_innov = z_MeasPos-tz_n;

x_rate_innov = x_MeasRate-wx_n;
y_rate_innov = y_MeasRate-wy_n;
z_rate_innov = z_MeasRate-wz_n;

// innovation from gravity vector measurement
x_grav = DCM_13*-1;
```

```
y_grav = DCM_23*-1;
z_grav = DCM_33*-1;

cross(x_acc,y_acc,z_acc,x_grav,y_grav,z_grav);

x_grav_pos = x_ComPos+g1;
y_grav_pos = y_ComPos+g2;
z_grav_pos = z_ComPos+g3;

x_grav_innov = x_grav_pos-tx_n;
y_grav_innov = y_grav_pos-ty_n;
z_grav_innov = z_grav_pos-tz_n;

// innovation from magnetic field vector measurement
x_mag_cmd = DCM_11*x_mag_ref+DCM_12*y_mag_ref+DCM_13*z_mag_ref;
y_mag_cmd = DCM_21*x_mag_ref+DCM_22*y_mag_ref+DCM_23*z_mag_ref;
z_mag_cmd = DCM_31*x_mag_ref+DCM_32*y_mag_ref+DCM_33*z_mag_ref;

mag(x_mag_cmd,y_mag_cmd,z_mag_cmd);
norm_mag_cmd = norm;

mag(x_Mag,y_Mag,z_Mag);
norm_Mag = norm;

cross(x_Mag,y_Mag,z_Mag,x_mag_cmd,y_mag_cmd,z_mag_cmd);

x_mag_pos = x_ComPos+(g1/(norm_mag_cmd*norm_Mag));
y_mag_pos = y_ComPos+(g2/(norm_mag_cmd*norm_Mag));
z_mag_pos = z_ComPos+(g3/(norm_mag_cmd*norm_Mag));

x_mag_innov = x_mag_pos-tx_n;
y_mag_innov = y_mag_pos-ty_n;
z_mag_innov = z_mag_pos-tz_n;

// air bearing state update;
tx_p = tx_n + L11*x_pos_innov  + L12*y_pos_innov  + L13*z_pos_innov
            + L14*x_rate_innov + L15*y_rate_innov + L16*z_rate_innov
            + L17*x_grav_innov + L18*y_grav_innov + L19*z_grav_innov
            + L110*x_mag_innov + L111*y_mag_innov + L112*z_mag_innov;
ty_p = ty_n + L21*x_pos_innov  + L22*y_pos_innov  + L23*z_pos_innov
            + L24*x_rate_innov + L25*y_rate_innov + L26*z_rate_innov
            + L27*x_grav_innov + L28*y_grav_innov + L29*z_grav_innov
            + L210*x_mag_innov + L211*y_mag_innov + L212*z_mag_innov;
tz_p = tz_n + L31*x_pos_innov  + L32*y_pos_innov  + L33*z_pos_innov
            + L34*x_rate_innov + L35*y_rate_innov + L36*z_rate_innov
            + L37*x_grav_innov + L38*y_grav_innov + L39*z_grav_innov
            + L310*x_mag_innov + L311*y_mag_innov + L312*z_mag_innov;
wx_p = wx_n + L41*x_pos_innov  + L42*y_pos_innov  + L43*z_pos_innov
            + L44*x_rate_innov + L45*y_rate_innov + L46*z_rate_innov
            + L47*x_grav_innov + L48*y_grav_innov + L49*z_grav_innov
            + L410*x_mag_innov + L411*y_mag_innov + L412*z_mag_innov;
wy_p = wy_n + L51*x_pos_innov  + L52*y_pos_innov  + L53*z_pos_innov
            + L54*x_rate_innov + L55*y_rate_innov + L56*z_rate_innov
            + L57*x_grav_innov + L58*y_grav_innov + L59*z_grav_innov
```

```
            + L510*x_mag_innov + L511*y_mag_innov + L512*z_mag_innov;
wz_p = wz_n + L61*x_pos_innov  + L62*y_pos_innov  + L63*z_pos_innov
            + L64*x_rate_innov + L65*y_rate_innov + L66*z_rate_innov
            + L67*x_grav_innov + L68*y_grav_innov + L69*z_grav_innov
            + L610*x_mag_innov + L611*y_mag_innov + L612*z_mag_innov;

wx_dot_p = (wx_p-wx_p_Prev)/RealDeltaT;
wy_dot_p = (wy_p-wy_p_Prev)/RealDeltaT;
wz_dot_p = (wz_p-wz_p_Prev)/RealDeltaT;

tx_p_Prev=tx_p; ty_p_Prev=ty_p; tz_p_Prev=tz_p;
wx_p_Prev=wx_p; wy_p_Prev=wy_p; wz_p_Prev=wz_p;

//*************************************************************************
// Control Law
//*************************************************************************

// error in angular orientation
x_ErrPos = x_ComPos-tx_p;
y_ErrPos = y_ComPos-ty_p;
z_ErrPos = z_ComPos-tz_p;

// error in angular rate
x_ErrRate = x_ComRate-wx_p;
y_ErrRate = y_ComRate-wy_p;
z_ErrRate = z_ComRate-wz_p;

// error in angular acceleration
x_ErrAcc = 0-wx_dot_p;
y_ErrAcc = 0-wy_dot_p;
z_ErrAcc = 0-wz_dot_p;

// commanded reaction wheel angular acceleration
Wx_dot_cmd_lqr = K11*x_ErrPos + K12*y_ErrPos + K13*z_ErrPos
               + K14*x_ErrRate + K15*y_ErrRate + K16*z_ErrRate;
Wy_dot_cmd_lqr = K21*x_ErrPos + K22*y_ErrPos + K23*z_ErrPos
               + K24*x_ErrRate + K25*y_ErrRate + K26*z_ErrRate;
Wz_dot_cmd_lqr = K31*x_ErrPos + K32*y_ErrPos + K33*z_ErrPos
               + K34*x_ErrRate + K35*y_ErrRate + K36*z_ErrRate;

// derivative control for rate damping
wx_dot_cmd_rd = deriv_gain*x_ErrAcc;
wy_dot_cmd_rd = deriv_gain*y_ErrAcc;
wz_dot_cmd_rd = deriv_gain*z_ErrAcc;

Wx_dot_cmd_rd = -(Ixx*D11*wx_dot_cmd_rd+Iyy*D21*wy_dot_cmd_rd
                 +Izz*D31*wz_dot_cmd_rd)/Ia;
Wy_dot_cmd_rd = -(Ixx*D12*wx_dot_cmd_rd+Iyy*D22*wy_dot_cmd_rd
                 +Izz*D32*wz_dot_cmd_rd)/Ia;
Wz_dot_cmd_rd = -(Ixx*D13*wx_dot_cmd_rd+Iyy*D23*wy_dot_cmd_rd
                 +Izz*D33*wz_dot_cmd_rd)/Ia;

// commanded reaction wheel acceleration from control law
Wx_dot_cmd_cl = Wx_dot_cmd_lqr+Wx_dot_cmd_rd;
```

```
Wy_dot_cmd_cl = Wy_dot_cmd_lqr+Wy_dot_cmd_rd;
Wz_dot_cmd_cl = Wz_dot_cmd_lqr+Wz_dot_cmd_rd;

// commanded reaction wheel rate from control law
Wx_cmd_cl = Wx_cmd_cl_Prev + Wx_dot_cmd_cl*RealDeltaT;
Wy_cmd_cl = Wy_cmd_cl_Prev + Wy_dot_cmd_cl*RealDeltaT;
Wz_cmd_cl = Wz_cmd_cl_Prev + Wz_dot_cmd_cl*RealDeltaT;

Wx_cmd_cl_Prev = Wx_cmd_cl;
Wy_cmd_cl_Prev = Wy_cmd_cl;
Wz_cmd_cl_Prev = Wz_cmd_cl;

// feedforward of dynamics due to angular momentum vector
x_ComAcc = (x_ComRate-x_ComRate_Prev)/RealDeltaT;
y_ComAcc = (y_ComRate-y_ComRate_Prev)/RealDeltaT;
z_ComAcc = (z_ComRate-z_ComRate_Prev)/RealDeltaT;

x_ComRate_Prev = x_ComRate;
y_ComRate_Prev = y_ComRate;
z_ComRate_Prev = z_ComRate;

// commanded reaction wheel acceleration from feed forward algorithm
Wx_dot_cmd_ff = D11*(z_ComRate*(D21*Wx_cmd_ff_Prev+D22*Wy_cmd_ff_Prev
   +D23*Wz_cmd_ff_Prev)-y_ComRate*(D31*Wx_cmd_ff_Prev+D32*Wy_cmd_ff_Prev
   +D33*Wz_cmd_ff_Prev)+(Iyy*y_ComRate*z_ComRate-Izz*y_ComRate*z_ComRate
   -Ixx*x_ComAcc)/Ia)+D21*(x_ComRate*(D31*Wx_cmd_ff_Prev+D32*Wy_cmd_ff_Prev
   +D33*Wz_cmd_ff_Prev)-z_ComRate*(D11*Wx_cmd_ff_Prev+D12*Wy_cmd_ff_Prev
   +D13*Wz_cmd_ff_Prev)+(Izz*x_ComRate*z_ComRate-Ixx*x_ComRate*z_ComRate
   -Iyy*y_ComAcc)/Ia)+D31*(y_ComRate*(D11*Wx_cmd_ff_Prev+D12*Wy_cmd_ff_Prev
   +D13*Wz_cmd_ff_Prev)-x_ComRate*(D21*Wx_cmd_ff_Prev+D22*Wy_cmd_ff_Prev
   +D23*Wz_cmd_ff_Prev)+(Ixx*x_ComRate*y_ComRate-Iyy*x_ComRate*y_ComRate
   -Izz*z_ComAcc)/Ia);

Wy_dot_cmd_ff = D12*(z_ComRate*(D21*Wx_cmd_ff_Prev+D22*Wy_cmd_ff_Prev
   +D23*Wz_cmd_ff_Prev)-y_ComRate*(D31*Wx_cmd_ff_Prev+D32*Wy_cmd_ff_Prev
   +D33*Wz_cmd_ff_Prev)+(Iyy*y_ComRate*z_ComRate-Izz*y_ComRate*z_ComRate
   -Ixx*x_ComAcc)/Ia)+D22*(x_ComRate*(D31*Wx_cmd_ff_Prev+D32*Wy_cmd_ff_Prev
   +D33*Wz_cmd_ff_Prev)-z_ComRate*(D11*Wx_cmd_ff_Prev+D12*Wy_cmd_ff_Prev
   +D13*Wz_cmd_ff_Prev)+(Izz*x_ComRate*z_ComRate-Ixx*x_ComRate*z_ComRate
   -Iyy*y_ComAcc)/Ia)+D32*(y_ComRate*(D11*Wx_cmd_ff_Prev+D12*Wy_cmd_ff_Prev
   +D13*Wz_cmd_ff_Prev)-x_ComRate*(D21*Wx_cmd_ff_Prev+D22*Wy_cmd_ff_Prev
   +D23*Wz_cmd_ff_Prev)+(Ixx*x_ComRate*y_ComRate-Iyy*x_ComRate*y_ComRate
   -Izz*z_ComAcc)/Ia);

Wz_dot_cmd_ff = D13*(z_ComRate*(D21*Wx_cmd_ff_Prev+D22*Wy_cmd_ff_Prev
   +D23*Wz_cmd_ff_Prev)-y_ComRate*(D31*Wx_cmd_ff_Prev+D32*Wy_cmd_ff_Prev
   +D33*Wz_cmd_ff_Prev)+(Iyy*y_ComRate*z_ComRate-Izz*y_ComRate*z_ComRate
   -Ixx*x_ComAcc)/Ia)+D23*(x_ComRate*(D31*Wx_cmd_ff_Prev+D32*Wy_cmd_ff_Prev
   +D33*Wz_cmd_ff_Prev)-z_ComRate*(D11*Wx_cmd_ff_Prev+D12*Wy_cmd_ff_Prev
   +D13*Wz_cmd_ff_Prev)+(Izz*x_ComRate*z_ComRate-Ixx*x_ComRate*z_ComRate
   -Iyy*y_ComAcc)/Ia)+D33*(y_ComRate*(D11*Wx_cmd_ff_Prev+D12*Wy_cmd_ff_Prev
   +D13*Wz_cmd_ff_Prev)-x_ComRate*(D21*Wx_cmd_ff_Prev+D22*Wy_cmd_ff_Prev
   +D23*Wz_cmd_ff_Prev)+(Ixx*x_ComRate*y_ComRate-Iyy*x_ComRate*y_ComRate
   -Izz*z_ComAcc)/Ia);
```

```
// commanded reaction wheel rate from feed forward algorithm
Wx_cmd_ff = Wx_cmd_ff_Prev + Wx_dot_cmd_ff*RealDeltaT;
Wy_cmd_ff = Wy_cmd_ff_Prev + Wy_dot_cmd_ff*RealDeltaT;
Wz_cmd_ff = Wz_cmd_ff_Prev + Wz_dot_cmd_ff*RealDeltaT;

Wx_cmd_ff_Prev = Wx_cmd_ff;
Wy_cmd_ff_Prev = Wy_cmd_ff;
Wz_cmd_ff_Prev = Wz_cmd_ff;

// combined commanded reaction wheel rate from control law
// and feed forward algorithm
Wx_cmd = Wx_cmd_cl+Wx_cmd_ff;
Wy_cmd = Wy_cmd_cl+Wy_cmd_ff;
Wz_cmd = Wz_cmd_cl+Wz_cmd_ff;

//*************************************************************************
// Determine Reaction Wheel Angular Rate from Motor Encoders
//*************************************************************************

Encoder_RW1 = encode1_b1;
Encoder_RW1 = (Encoder_RW1 << 8) | encode1_b2;
Encoder_RW1 = (Encoder_RW1 << 8) | encode1_b3;
Encoder_RW1 = (Encoder_RW1 << 8) | encode1_b4;

Encoder_RW2 = encode2_b1;
Encoder_RW2 = (Encoder_RW2 << 8) | encode2_b2;
Encoder_RW2 = (Encoder_RW2 << 8) | encode2_b3;
Encoder_RW2 = (Encoder_RW2 << 8) | encode2_b4;

Encoder_RW3 = encode3_b1;
Encoder_RW3 = (Encoder_RW3 << 8) | encode3_b2;
Encoder_RW3 = (Encoder_RW3 << 8) | encode3_b3;
Encoder_RW3 = (Encoder_RW3 << 8) | encode3_b4;

Encoder_RW4 = encode4_b1;
Encoder_RW4 = (Encoder_RW4 << 8) | encode4_b2;
Encoder_RW4 = (Encoder_RW4 << 8) | encode4_b3;
Encoder_RW4 = (Encoder_RW4 << 8) | encode4_b4;

RPM1 = ((Encoder_RW1-Encoder_RW1_prev)/1000.0)/RealDeltaT*60.0;
RPM2 = ((Encoder_RW2-Encoder_RW2_prev)/1000.0)/RealDeltaT*60.0;
RPM3 = ((Encoder_RW3-Encoder_RW3_prev)/1000.0)/RealDeltaT*60.0;
RPM4 = ((Encoder_RW4-Encoder_RW4_prev)/1000.0)/RealDeltaT*60.0;

Encoder_RW1_prev = Encoder_RW1;
Encoder_RW2_prev = Encoder_RW2;
Encoder_RW3_prev = Encoder_RW3;
Encoder_RW4_prev = Encoder_RW4;

RPS1 = RPM1*6.2832/60;
RPS2 = RPM2*6.2832/60;
RPS3 = RPM3*6.2832/60;
RPS4 = RPM4*6.2832/60;
```

```
//*************************************************************************
// Reaction Wheel Motor Controller
//*************************************************************************

// limit commanded reaction wheel angular rate
RW1_ComSpeed = constrain(Wx_cmd, -194, 194);
RW2_ComSpeed = constrain(Wy_cmd, -194, 194);
RW3_ComSpeed = constrain(Wz_cmd, -194, 194);

// convert from radians per second to motor controller voltage
RW1_Volt = RW1_ComSpeed/13.379;
RW2_Volt = RW2_ComSpeed/13.379;
RW3_Volt = RW3_ComSpeed/13.379;

// absolute value of commanded reaction wheel angular rate
RW1_ComSpeed_abs = abs(RW1_ComSpeed);
RW2_ComSpeed_abs = abs(RW2_ComSpeed);
RW3_ComSpeed_abs = abs(RW3_ComSpeed);

// convert from radians per second to motor controller byte representation
ComSpeed1 = map(RW1_ComSpeed_abs, 0, 194, 0, 127);
ComSpeed2 = map(RW2_ComSpeed_abs, 0, 194, 0, 127);
ComSpeed3 = map(RW3_ComSpeed_abs, 0, 194, 0, 127);

// determine reaction wheel direction command for motor controller
if (RW1_ComSpeed >=0)
{
  ComDir1 = 0xC2;
}
else
{
  ComDir1 = 0xC1;
}

if (RW2_ComSpeed >=0)
{
  ComDir2 = 0xCA;
}
else
{
  ComDir2 = 0xC9;
}

if (RW3_ComSpeed >=0)
{
  ComDir3 = 0xC2;
}
else
{
  ComDir3 = 0xC1;
}

// brake speed algorithm for motor controller
```

```
ThresSpeed1 = 1.05*RW1_ComSpeed_abs;
ThresSpeed2 = 1.05*RW2_ComSpeed_abs;
ThresSpeed3 = 1.05*RW3_ComSpeed_abs;

if (timekeep < 0.5)
{
  ThresSpeed1 = 500000;
  ThresSpeed2 = 500000;
  ThresSpeed3 = 500000;
}

if (RPS1 > ThresSpeed1)
{
  MotorDir1 = 0xC0;
  MotorSpeed1 = 127;
  RW1_Volt = 0;
}
else
{
  MotorDir1 = ComDir1;
  MotorSpeed1 = ComSpeed1;
}

if (RPS2 > ThresSpeed2)
{
  MotorDir2 = 0xC8;
  MotorSpeed2 = 127;
  RW2_Volt = 0;
}
else
{
  MotorDir2 = ComDir2;
  MotorSpeed2 = ComSpeed2;
}

if (RPS3 > ThresSpeed3)
{
  MotorDir3 = 0xC0;
  MotorSpeed3 = 127;
  RW3_Volt = 0;
}
else
{
  MotorDir3 = ComDir3;
  MotorSpeed3 = ComSpeed3;
}

//*************************************************************************
// Print Results
//*************************************************************************

if (timekeep_flag == 1)     {serial_out(timekeep,
                             timekeep_post);}
if (RealDeltaT_flag == 1)   {serial_out(RealDeltaT*1000,
```

```
                                RealDeltaT_post);}
if (DeltaT_flag == 1)          {serial_out(DeltaT,
                                DeltaT_post);}
if (RW1_ComSpeed_flag == 1) {serial_out(RW1_ComSpeed,
                                RW1_ComSpeed_post);}
if (RPS1_flag == 1)            {serial_out(RPS1,
                                RPS1_post);}
if (RW1_Volt_flag == 1)        {serial_out(RW1_Volt_flag,
                                RW1_Volt_post);}
if (RW2_ComSpeed_flag == 1) {serial_out(RW2_ComSpeed,
                                RW2_ComSpeed_post);}
if (RPS2_flag == 1)            {serial_out(RPS2,
                                RPS2_post);}
if (RW2_Volt_flag == 1)        {serial_out(RW2_Volt_flag,
                                RW2_Volt_post);}
if (RW3_ComSpeed_flag == 1) {serial_out(RW3_ComSpeed,
                                RW3_ComSpeed_post);}
if (RPS3_flag == 1)            {serial_out(RPS3,
                                RPS3_post);}
if (RW3_Volt_flag == 1)        {serial_out(RW3_Volt_flag,
                                RW3_Volt_post);}
if (x_ComPos_flag == 1)        {serial_out(x_ComPos*DEG,
                                x_ComPos_post);}
if (y_ComPos_flag == 1)        {serial_out(y_ComPos*DEG,
                                y_ComPos_post);}
if (z_ComPos_flag == 1)        {serial_out(z_ComPos*DEG,
                                z_ComPos_post);}
if (x_ComRate_flag == 1)       {serial_out(x_ComRate*DEG,
                                x_ComRate_post);}
if (y_ComRate_flag == 1)       {serial_out(y_ComRate*DEG,
                                y_ComRate_post);}
if (z_ComRate_flag == 1)       {serial_out(z_ComRate*DEG,
                                z_ComRate_post);}
if (tx_p_flag == 1)            {serial_out(tx_p*DEG,
                                tx_p_post);}
if (ty_p_flag == 1)            {serial_out(ty_p*DEG,
                                ty_p_post);}
if (tz_p_flag == 1)            {serial_out(tz_p*DEG,
                                tz_p_post);}
if (wx_p_flag == 1)            {serial_out(wx_p*DEG,
                                wx_p_post);}
if (wy_p_flag == 1)            {serial_out(wy_p*DEG,
                                wy_p_post);}
if (wz_p_flag == 1)            {serial_out(wz_p*DEG,
                                wz_p_post);}
if (wx_dot_p_flag == 1)        {serial_out(wx_dot_p*DEG,
                                wx_dot_p_post);}
if (wy_dot_p_flag == 1)        {serial_out(wy_dot_p*DEG,
                                wy_dot_p_post);}
if (wz_dot_p_flag == 1)        {serial_out(wz_dot_p*DEG,
                                wz_dot_p_post);}
if (RPS4_flag == 1)            {serial_out(RPS4,
                                RPS4_post);}
if (x_gyro_flag == 1)          {serial_out(x_gyro*DEG,
```

```
                                x_gyro_post);}
if (y_gyro_flag == 1)          {serial_out(y_gyro*DEG,
                                y_gyro_post);}
if (z_gyro_flag == 1)          {serial_out(z_gyro*DEG,
                                z_gyro_post);}
if (x_MeasPos_flag == 1)       {serial_out(x_MeasPos*DEG,
                                x_MeasPos_post);}
if (y_MeasPos_flag == 1)       {serial_out(y_MeasPos*DEG,
                                y_MeasPos_post);}
if (z_MeasPos_flag == 1)       {serial_out(z_MeasPos*DEG,
                                z_MeasPos_post);}
if (x_MeasRate_flag == 1)      {serial_out(x_MeasRate*DEG,
                                x_MeasRate_post);}
if (y_MeasRate_flag == 1)      {serial_out(y_MeasRate*DEG,
                                y_MeasRate_post);}
if (z_MeasRate_flag == 1)      {serial_out(z_MeasRate*DEG,
                                z_MeasRate_post);}
if (x_acc_flag == 1)           {serial_out(x_acc,
                                x_acc_post);}
if (y_acc_flag == 1)           {serial_out(y_acc,
                                y_acc_post);}
if (z_acc_flag == 1)           {serial_out(z_acc,
                                z_acc_post);}
if (x_mag_flag == 1)           {serial_out(x_mag,
                                x_mag_post);}
if (y_mag_flag == 1)           {serial_out(y_mag,
                                y_mag_post);}
if (z_mag_flag == 1)           {serial_out(z_mag,
                                z_mag_post);}
if (x_Mag_flag == 1)           {serial_out(x_Mag,
                                x_Mag_post);}
if (y_Mag_flag == 1)           {serial_out(y_Mag,
                                y_Mag_post);}
if (z_Mag_flag == 1)           {serial_out(z_Mag,
                                z_Mag_post);}
if (Wx_est_flag == 1)          {serial_out(Wx_est,
                                Wx_est_post);}
if (Wy_est_flag == 1)          {serial_out(Wy_est,
                                Wy_est_post);}
if (Wz_est_flag == 1)          {serial_out(Wz_est,
                                Wz_est_post);}
if (x_grav_flag == 1)          {serial_out(x_grav,
                                x_grav_post);}
if (y_grav_flag == 1)          {serial_out(y_grav,
                                y_grav_post);}
if (z_grav_flag == 1)          {serial_out(z_grav,
                                z_grav_post);}
if (x_grav_pos_flag == 1)      {serial_out(x_grav_pos*DEG,
                                x_grav_pos_post);}
if (y_grav_pos_flag == 1)      {serial_out(y_grav_pos*DEG,
                                y_grav_pos_post);}
if (z_grav_pos_flag == 1)      {serial_out(z_grav_pos*DEG,
                                z_grav_pos_post);}
if (x_mag_cmd_flag == 1)       {serial_out(x_mag_cmd,
```

```
                                x_mag_cmd_post);}
    if (y_mag_cmd_flag == 1)    {serial_out(y_mag_cmd,
                                y_mag_cmd_post);}
    if (z_mag_cmd_flag == 1)    {serial_out(z_mag_cmd,
                                z_mag_cmd_post);}
    if (x_mag_pos_flag == 1)    {serial_out(x_mag_pos*DEG,
                                x_mag_pos_post);}
    if (y_mag_pos_flag == 1)    {serial_out(y_mag_pos*DEG,
                                y_mag_pos_post);}
    if (z_mag_pos_flag == 1)    {serial_out(z_mag_pos*DEG,
                                z_mag_pos_post);}

    //*************************************************************************
    // Read Encoder Information from Auxiliary Arduino
    //*************************************************************************
     if (Serial1.available() > 15);
    {
     encode1_b1 = Serial1.read();
     encode1_b2 = Serial1.read();
     encode1_b3 = Serial1.read();
     encode1_b4 = Serial1.read();

     encode2_b1 = Serial1.read();
     encode2_b2 = Serial1.read();
     encode2_b3 = Serial1.read();
     encode2_b4 = Serial1.read();

     encode3_b1 = Serial1.read();
     encode3_b2 = Serial1.read();
     encode3_b3 = Serial1.read();
     encode3_b4 = Serial1.read();

     encode4_b1 = Serial1.read();
     encode4_b2 = Serial1.read();
     encode4_b3 = Serial1.read();
     encode4_b4 = Serial1.read();
    }

    Serial1.flush();

    digitalWrite(progPin, LOW);
  }
}

void timer2()
{
  timercounter = timercounter+1;
}
```

C.5 Provided mega_aux.pde Code

This code can be copied directly into the Arduino's IDE, named "mega_aux.pde and uploaded to the Auxiliary Arduino.

```
//*****************************************************************************
// Initialize Required Variables and Interrupt Pins
//*****************************************************************************

const int DigitalIn2 = 2;
const int DigitalIn3 = 3;
const int DigitalIn20 = 20;
const int DigitalIn21 = 21;
int progPin = 43;
long EncoderCounter2 = 0;
long EncoderCounter3 = 0;
long EncoderCounter20 = 0;
long EncoderCounter21 = 0;
byte encode2_b1 = 0;
byte encode2_b2 = 0;
byte encode2_b3 = 0;
byte encode2_b4 = 0;
byte encode3_b1 = 0;
byte encode3_b2 = 0;
byte encode3_b3 = 0;
byte encode3_b4 = 0;
byte encode20_b1 = 0;
byte encode20_b2 = 0;
byte encode20_b3 = 0;
byte encode20_b4 = 0;
byte encode21_b1 = 0;
byte encode21_b2 = 0;
byte encode21_b3 = 0;
byte encode21_b4 = 0;
byte byte1_rw1 = 0;
byte byte2_rw1 = 0;
byte byte1_rw2 = 0;
byte byte2_rw2 = 0;
byte byte1_rw3 = 0;
byte byte2_rw3 = 0;
byte byte1_rw4 = 0;
byte byte2_rw4 = 0;
byte clr = 0;

//*****************************************************************************
// Initialize Auxiliary Loop
//*****************************************************************************
void setup()
{
  Serial.begin(57600);        // port between aux arduino and GS computer
  Serial1.begin(115200);      // port between aux and main arduino
  Serial2.begin(19200);       // port between arduino and MC for RW1 and RW2
  Serial3.begin(19200);       // port between arduino and MC for RW3 and RW4
```

```
  pinMode(progPin, OUTPUT);

  attachInterrupt(0, encoder2, RISING);
  attachInterrupt(1, encoder3, RISING);
  attachInterrupt(3, encoder20, RISING);
  attachInterrupt(2, encoder21, RISING);

  interrupts();
}

//*****************************************************************************
// Begin Auxiliary Loop
//*****************************************************************************
void loop()
{
 if (Serial1.available() > 7)
 {
   digitalWrite(progPin, HIGH);

   //*****************************************************************************
   // Write Motor Encoder Values to Main Arduino
   //*****************************************************************************
   noInterrupts();
   encode2_b1 = (EncoderCounter2&0xFF000000) >> 24;
   encode2_b2 = (EncoderCounter2&0x00FF0000) >> 16;
   encode2_b3 = (EncoderCounter2&0x0000FF00) >> 8;
   encode2_b4 = EncoderCounter2&0x000000FF;

   encode3_b1 = (EncoderCounter3&0xFF000000) >> 24;
   encode3_b2 = (EncoderCounter3&0x00FF0000) >> 16;
   encode3_b3 = (EncoderCounter3&0x0000FF00) >> 8;
   encode3_b4 = EncoderCounter3&0x000000FF;

   encode20_b1 = (EncoderCounter20&0xFF000000) >> 24;
   encode20_b2 = (EncoderCounter20&0x00FF0000) >> 16;
   encode20_b3 = (EncoderCounter20&0x0000FF00) >> 8;
   encode20_b4 = EncoderCounter20&0x000000FF;

   encode21_b1 = (EncoderCounter21&0xFF000000) >> 24;
   encode21_b2 = (EncoderCounter21&0x00FF0000) >> 16;
   encode21_b3 = (EncoderCounter21&0x0000FF00) >> 8;
   encode21_b4 = EncoderCounter21&0x000000FF;
   interrupts();

   Serial1.write(encode2_b1);
   Serial1.write(encode2_b2);
   Serial1.write(encode2_b3);
   Serial1.write(encode2_b4);

   Serial1.write(encode3_b1);
   Serial1.write(encode3_b2);
   Serial1.write(encode3_b3);
   Serial1.write(encode3_b4);
```

```
Serial1.write(encode20_b1);
Serial1.write(encode20_b2);
Serial1.write(encode20_b3);
Serial1.write(encode20_b4);

Serial1.write(encode21_b1);
Serial1.write(encode21_b2);
Serial1.write(encode21_b3);
Serial1.write(encode21_b4);

//*****************************************************************************
// Read Reaction Wheel Commands From Main Arduino
//*****************************************************************************
byte1_rw1 = Serial1.read();
byte2_rw1 = Serial1.read();

byte1_rw2 = Serial1.read();
byte2_rw2 = Serial1.read();

byte1_rw3 = Serial1.read();
byte2_rw3 = Serial1.read();

byte1_rw4 = Serial1.read();
byte2_rw4 = Serial1.read();

//*****************************************************************************
// Write Reaction Wheel Commands to Motor Controllers
//*****************************************************************************
Serial2.write(byte1_rw1);
Serial2.write(byte2_rw1);

Serial2.write(byte1_rw2);
Serial2.write(byte2_rw2);

Serial3.write(byte1_rw3);
Serial3.write(byte2_rw3);

Serial3.write(byte1_rw4);
Serial3.write(byte2_rw4);

Serial.flush();
Serial1.flush();
Serial2.flush();
Serial3.flush();

byte1_rw1 = clr;
byte2_rw1 = clr;

byte1_rw2 = clr;
byte2_rw2 = clr;

byte1_rw3 = clr;
byte2_rw3 = clr;
```

```
    byte1_rw4 = clr;
    byte2_rw4 = clr;

    digitalWrite(progPin, LOW);
  }
}

//*****************************************************************************
// Define Motor Encoder Interrupt Functions
//*****************************************************************************

// hardware interrupt function, captures encoder signal from RW1
void encoder2()
{
  EncoderCounter2 = EncoderCounter2 + 1;
}

// hardware interrupt function, captures encoder signal from RW2
void encoder3()
{
  EncoderCounter3 = EncoderCounter3 + 1;
}

// hardware interrupt function, captures encoder signal from RW3
void encoder20()
{
  EncoderCounter20 = EncoderCounter20 + 1;
}

// hardware interrupt function, captures encoder signal from RW4
void encoder21()
{
  EncoderCounter21 = EncoderCounter21 + 1;
}
```

C.6 Provided test_init.m MATLAB Script

This code can be copied directly into a blank MATLAB Script file, named "test_init.m" and used to provide initial conditions to the ADCS testbed simulation and create the init.h file necessary to provide initial conditions to the Arduino code for use on the physical air bearing.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% test_init.m
%
% Purpose:  Initialize constants for ADCS Simulink Simulation and create
%           init.h code file for Main Arduino on ADCS Testbed
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear workspace, clear command window, close figures
% clear all
% close all
% clc

%% Initial Conditions for Simulation and Physical Testbed

% Set time step
% timestep divided by 4096 equals delta T in seconds
% 205/4096 = 0.05 seconds
timestep = 205;

% timestep in seconds, must correspond to timestep from above
dt_rw = 0.05;

% Set initial reaction wheel angular velocity in radians per second
RW1_init = 0;
RW2_init = 0;
RW3_init = 0;

% Set initial air bearing angular orientation in radians
x_pos_init = 0;
y_pos_init = 0;
z_pos_init = 0;

% Set initial air bearing angular velocity in radians per second
x_rate_init = 0;
y_rate_init = 0;
z_rate_init = 0;

% Air Bearing inertia tensor
Ixx = 4.0774;
Iyy = 4.1360;
Izz = 2.3216;

Iab = [Ixx 0   0;
       0   Iyy 0;
       0   0   Izz];
```

```
% Reaction wheel inertia tensor
Ia = 0.00725;

Irw = [Ia 0  0;
       0  Ia 0;
       0  0  Ia];

% DCM from reaction wheel frame to body frame
DCM_BODY_wrt_RW = [ 0.408248    0.408248   -0.816496;
                   -0.7071      0.7071      0;
                    0.5774      0.5774      0.5774];

% Linearized State Space Model of Air Bearing
[Ad_ab Bd_ab] = linear_eq(Ixx,Iyy,Izz,Ia,timestep/4096);

%% Initial Conditions unique to Arduino Code

% Initial direction cosine matrix
DCM_init = expm(-skew([x_pos_init;y_pos_init;z_pos_init]));

%% Initial Conditions unique to Simulation

% Set high frequency time step to simulate continous environment
dt = 0.005;

% Magnetic field in reference frame for simulation
mag_ref = [0;1400;0];

% State Space Model for Reaction Wheels
A_rw = [-0.0246  0       0       9.6552  0       0;
         0      -0.0246  0       0       9.6552  0;
         0       0      -0.0246  0       0       9.6552;
       -20.9440  0       0    -448.3776  0       0;
         0     -20.9440  0       0    -448.3776  0;
         0       0     -20.9440  0       0    -448.3776];

B_rw = [0        0        0;
        0        0        0;
        0        0        0;
      294.9853   0        0;
        0      294.9853   0;
        0        0      294.9853];

C_rw = [1 0 0 0 0 0;
        0 1 0 0 0 0;
        0 0 1 0 0 0];

D_rw = [0 0 0;
        0 0 0;
        0 0 0];

sysc_rw = ss(A_rw,B_rw,C_rw,D_rw);
sysd_rw = c2d(sysc_rw,dt_rw);
```

```
Ad_rw = sysd_rw.a;
Bd_rw = sysd_rw.b;


%% Simulation outputs to be plotted after run is complete
% 1 = Plot,  0 = Do Not Plot

x_pos_flag = 0;
y_pos_flag = 0;
z_pos_flag = 0;

x_rate_flag = 0;
y_rate_flag = 0;
z_rate_flag = 0;

RW1_flag = 0;
RW2_flag = 0;
RW3_flag = 0;

ss_x_pos_flag = 0;
ss_y_pos_flag = 0;
ss_z_pos_flag = 0;

%% Kalman Filter constants, Linear Quadratic Estimator

Q_lqe = 0.05*eye(6,6);

R_lqe = 0.1*        [1 0 0 0 0 0 0 0 0 0 0 0;
                     0 1 0 0 0 0 0 0 0 0 0 0;
                     0 0 1 0 0 0 0 0 0 0 0 0;
                     0 0 0 1 0 0 0 0 0 0 0 0;
                     0 0 0 0 1 0 0 0 0 0 0 0;
                     0 0 0 0 0 1 0 0 0 0 0 0;
                     0 0 0 0 0 0 1 0 0 0 0 0;
                     0 0 0 0 0 0 0 1 0 0 0 0;
                     0 0 0 0 0 0 0 0 1 0 0 0;
                     0 0 0 0 0 0 0 0 0 1 0 0;
                     0 0 0 0 0 0 0 0 0 0 1 0;
                     0 0 0 0 0 0 0 0 0 0 0 1];

C_lqe = [0 0 0 0 0 0;
         0 0 0 0 0 0;
         0 0 0 0 0 0;
         0 0 0 1 0 0;
         0 0 0 0 1 0;
         0 0 0 0 0 1;
         1 0 0 0 0 0;
         0 1 0 0 0 0;
         0 0 1 0 0 0;
         1 0 0 0 0 0;
         0 1 0 0 0 0;
         0 0 1 0 0 0];

Lss = dlqe(Ad_ab,eye(6,6),C_lqe,Q_lqe,R_lqe);
```

```
%% Discrete Linear Quadratic Regulator Constants
Q_lqr = 1*[50000 0 0 0 0 0;
              0 50000 0 0 0 0;
              0 0 50000 0 0 0;
              0 0 0 500000 0 0;
              0 0 0 0 500000 0;
              0 0 0 0 0 500000];

R_lqr = 1* [1 0 0;
            0 1 0;
            0 0 1];

K = dlqr(Ad_ab,Bd_ab,Q_lqr,R_lqr);

%% Derivative Gain

Kd = 1.5;

%% create init.h file for use by Arduino

init = fopen('init.h','w+');

fprintf(init,'// initial values\n\n');

fprintf(init,'int timestep = %4.0f;\n\n',timestep);

fprintf(init,'double Ixx = %4.3f;\n',Ixx);
fprintf(init,'double Iyy = %4.3f;\n',Iyy);
fprintf(init,'double Izz = %4.3f;\n\n',Izz);

fprintf(init,'double Ia = %6.5f;\n\n',Ia);

fprintf(init,'double deriv_gain = %4.2f;\n\n',Kd);

fprintf(init,'double Wx_est_Prev = %3.0f;\n',RW1_init);
fprintf(init,'double Wy_est_Prev = %3.0f;\n',RW2_init);
fprintf(init,'double Wz_est_Prev = %3.0f;\n\n',RW3_init);

fprintf(init,'double Wx_cmd_ff_Prev = %3.0f;\n',RW1_init);
fprintf(init,'double Wy_cmd_ff_Prev = %3.0f;\n',RW2_init);
fprintf(init,'double Wz_cmd_ff_Prev = %3.0f;\n\n',RW3_init);

fprintf(init,'double ix_est_Prev = %3.0f;\n',0);
fprintf(init,'double iy_est_Prev = %3.0f;\n',0);
fprintf(init,'double iz_est_Prev = %3.0f;\n\n',0);

fprintf(init,'double tx_p_Prev = %3.2f;\n',x_pos_init);
fprintf(init,'double ty_p_Prev = %3.2f;\n',y_pos_init);
fprintf(init,'double tz_p_Prev = %3.2f;\n\n',z_pos_init);

fprintf(init,'double wx_p_Prev = %3.2f;\n',x_rate_init);
fprintf(init,'double wy_p_Prev = %3.2f;\n',y_rate_init);
fprintf(init,'double wz_p_Prev = %3.2f;\n\n',z_rate_init);
```

```
fprintf(init,'double DCM_11_Prev = %9.8f;\n',DCM_init(1,1));
fprintf(init,'double DCM_12_Prev = %9.8f;\n',DCM_init(1,2));
fprintf(init,'double DCM_13_Prev = %9.8f;\n',DCM_init(1,3));
fprintf(init,'double DCM_21_Prev = %9.8f;\n',DCM_init(2,1));
fprintf(init,'double DCM_22_Prev = %9.8f;\n',DCM_init(2,2));
fprintf(init,'double DCM_23_Prev = %9.8f;\n',DCM_init(2,3));
fprintf(init,'double DCM_31_Prev = %9.8f;\n',DCM_init(3,1));
fprintf(init,'double DCM_32_Prev = %9.8f;\n',DCM_init(3,2));
fprintf(init,'double DCM_33_Prev = %9.8f;\n\n',DCM_init(3,3));

fprintf(init,'double Ad_11 = %9.8f;\n',Ad_ab(1,1));
fprintf(init,'double Ad_12 = %9.8f;\n',Ad_ab(1,2));
fprintf(init,'double Ad_13 = %9.8f;\n',Ad_ab(1,3));
fprintf(init,'double Ad_14 = %9.8f;\n',Ad_ab(1,4));
fprintf(init,'double Ad_15 = %9.8f;\n',Ad_ab(1,5));
fprintf(init,'double Ad_16 = %9.8f;\n',Ad_ab(1,6));
fprintf(init,'double Ad_21 = %9.8f;\n',Ad_ab(2,1));
fprintf(init,'double Ad_22 = %9.8f;\n',Ad_ab(2,2));
fprintf(init,'double Ad_23 = %9.8f;\n',Ad_ab(2,3));
fprintf(init,'double Ad_24 = %9.8f;\n',Ad_ab(2,4));
fprintf(init,'double Ad_25 = %9.8f;\n',Ad_ab(2,5));
fprintf(init,'double Ad_26 = %9.8f;\n',Ad_ab(2,6));
fprintf(init,'double Ad_31 = %9.8f;\n',Ad_ab(3,1));
fprintf(init,'double Ad_32 = %9.8f;\n',Ad_ab(3,2));
fprintf(init,'double Ad_33 = %9.8f;\n',Ad_ab(3,3));
fprintf(init,'double Ad_34 = %9.8f;\n',Ad_ab(3,4));
fprintf(init,'double Ad_35 = %9.8f;\n',Ad_ab(3,5));
fprintf(init,'double Ad_36 = %9.8f;\n',Ad_ab(3,6));
fprintf(init,'double Ad_41 = %9.8f;\n',Ad_ab(4,1));
fprintf(init,'double Ad_42 = %9.8f;\n',Ad_ab(4,2));
fprintf(init,'double Ad_43 = %9.8f;\n',Ad_ab(4,3));
fprintf(init,'double Ad_44 = %9.8f;\n',Ad_ab(4,4));
fprintf(init,'double Ad_45 = %9.8f;\n',Ad_ab(4,5));
fprintf(init,'double Ad_46 = %9.8f;\n',Ad_ab(4,6));
fprintf(init,'double Ad_51 = %9.8f;\n',Ad_ab(5,1));
fprintf(init,'double Ad_52 = %9.8f;\n',Ad_ab(5,2));
fprintf(init,'double Ad_53 = %9.8f;\n',Ad_ab(5,3));
fprintf(init,'double Ad_54 = %9.8f;\n',Ad_ab(5,4));
fprintf(init,'double Ad_55 = %9.8f;\n',Ad_ab(5,5));
fprintf(init,'double Ad_56 = %9.8f;\n',Ad_ab(5,6));
fprintf(init,'double Ad_61 = %9.8f;\n',Ad_ab(6,1));
fprintf(init,'double Ad_62 = %9.8f;\n',Ad_ab(6,2));
fprintf(init,'double Ad_63 = %9.8f;\n',Ad_ab(6,3));
fprintf(init,'double Ad_64 = %9.8f;\n',Ad_ab(6,4));
fprintf(init,'double Ad_65 = %9.8f;\n',Ad_ab(6,5));
fprintf(init,'double Ad_66 = %9.8f;\n\n',Ad_ab(6,6));

fprintf(init,'double Bd_11 = %9.8f;\n',Bd_ab(1,1));
fprintf(init,'double Bd_12 = %9.8f;\n',Bd_ab(1,2));
fprintf(init,'double Bd_13 = %9.8f;\n',Bd_ab(1,3));
fprintf(init,'double Bd_21 = %9.8f;\n',Bd_ab(2,1));
fprintf(init,'double Bd_22 = %9.8f;\n',Bd_ab(2,2));
fprintf(init,'double Bd_23 = %9.8f;\n',Bd_ab(2,3));
```

```
fprintf(init,'double Bd_31 = %9.8f;\n',Bd_ab(3,1));
fprintf(init,'double Bd_32 = %9.8f;\n',Bd_ab(3,2));
fprintf(init,'double Bd_33 = %9.8f;\n',Bd_ab(3,3));
fprintf(init,'double Bd_41 = %9.8f;\n',Bd_ab(4,1));
fprintf(init,'double Bd_42 = %9.8f;\n',Bd_ab(4,2));
fprintf(init,'double Bd_43 = %9.8f;\n',Bd_ab(4,3));
fprintf(init,'double Bd_51 = %9.8f;\n',Bd_ab(5,1));
fprintf(init,'double Bd_52 = %9.8f;\n',Bd_ab(5,2));
fprintf(init,'double Bd_53 = %9.8f;\n',Bd_ab(5,3));
fprintf(init,'double Bd_61 = %9.8f;\n',Bd_ab(6,1));
fprintf(init,'double Bd_62 = %9.8f;\n',Bd_ab(6,2));
fprintf(init,'double Bd_63 = %9.8f;\n\n',Bd_ab(6,3));

fprintf(init,'double L11 = %9.8f;\n',Lss(1,1));
fprintf(init,'double L12 = %9.8f;\n',Lss(1,2));
fprintf(init,'double L13 = %9.8f;\n',Lss(1,3));
fprintf(init,'double L14 = %9.8f;\n',Lss(1,4));
fprintf(init,'double L15 = %9.8f;\n',Lss(1,5));
fprintf(init,'double L16 = %9.8f;\n',Lss(1,6));
fprintf(init,'double L17 = %9.8f;\n',Lss(1,7));
fprintf(init,'double L18 = %9.8f;\n',Lss(1,8));
fprintf(init,'double L19 = %9.8f;\n',Lss(1,9));
fprintf(init,'double L110 = %9.8f;\n',Lss(1,10));
fprintf(init,'double L111 = %9.8f;\n',Lss(1,11));
fprintf(init,'double L112 = %9.8f;\n',Lss(1,12));
fprintf(init,'double L21 = %9.8f;\n',Lss(2,1));
fprintf(init,'double L22 = %9.8f;\n',Lss(2,2));
fprintf(init,'double L23 = %9.8f;\n',Lss(2,3));
fprintf(init,'double L24 = %9.8f;\n',Lss(2,4));
fprintf(init,'double L25 = %9.8f;\n',Lss(2,5));
fprintf(init,'double L26 = %9.8f;\n',Lss(2,6));
fprintf(init,'double L27 = %9.8f;\n',Lss(2,7));
fprintf(init,'double L28 = %9.8f;\n',Lss(2,8));
fprintf(init,'double L29 = %9.8f;\n',Lss(2,9));
fprintf(init,'double L210 = %9.8f;\n',Lss(2,10));
fprintf(init,'double L211 = %9.8f;\n',Lss(2,11));
fprintf(init,'double L212 = %9.8f;\n',Lss(2,12));
fprintf(init,'double L31 = %9.8f;\n',Lss(3,1));
fprintf(init,'double L32 = %9.8f;\n',Lss(3,2));
fprintf(init,'double L33 = %9.8f;\n',Lss(3,3));
fprintf(init,'double L34 = %9.8f;\n',Lss(3,4));
fprintf(init,'double L35 = %9.8f;\n',Lss(3,5));
fprintf(init,'double L36 = %9.8f;\n',Lss(3,6));
fprintf(init,'double L37 = %9.8f;\n',Lss(3,7));
fprintf(init,'double L38 = %9.8f;\n',Lss(3,8));
fprintf(init,'double L39 = %9.8f;\n',Lss(3,9));
fprintf(init,'double L310 = %9.8f;\n',Lss(3,10));
fprintf(init,'double L311 = %9.8f;\n',Lss(3,11));
fprintf(init,'double L312 = %9.8f;\n',Lss(3,12));
fprintf(init,'double L41 = %9.8f;\n',Lss(4,1));
fprintf(init,'double L42 = %9.8f;\n',Lss(4,2));
fprintf(init,'double L43 = %9.8f;\n',Lss(4,3));
fprintf(init,'double L44 = %9.8f;\n',Lss(4,4));
fprintf(init,'double L45 = %9.8f;\n',Lss(4,5));
```

```
fprintf(init,'double L46 = %9.8f;\n',Lss(4,6));
fprintf(init,'double L47 = %9.8f;\n',Lss(4,7));
fprintf(init,'double L48 = %9.8f;\n',Lss(4,8));
fprintf(init,'double L49 = %9.8f;\n',Lss(4,9));
fprintf(init,'double L410 = %9.8f;\n',Lss(4,10));
fprintf(init,'double L411 = %9.8f;\n',Lss(4,11));
fprintf(init,'double L412 = %9.8f;\n',Lss(4,12));
fprintf(init,'double L51 = %9.8f;\n',Lss(5,1));
fprintf(init,'double L52 = %9.8f;\n',Lss(5,2));
fprintf(init,'double L53 = %9.8f;\n',Lss(5,3));
fprintf(init,'double L54 = %9.8f;\n',Lss(5,4));
fprintf(init,'double L55 = %9.8f;\n',Lss(5,5));
fprintf(init,'double L56 = %9.8f;\n',Lss(5,6));
fprintf(init,'double L57 = %9.8f;\n',Lss(5,7));
fprintf(init,'double L58 = %9.8f;\n',Lss(5,8));
fprintf(init,'double L59 = %9.8f;\n',Lss(5,9));
fprintf(init,'double L510 = %9.8f;\n',Lss(5,10));
fprintf(init,'double L511 = %9.8f;\n',Lss(5,11));
fprintf(init,'double L512 = %9.8f;\n',Lss(5,12));
fprintf(init,'double L61 = %9.8f;\n',Lss(6,1));
fprintf(init,'double L62 = %9.8f;\n',Lss(6,2));
fprintf(init,'double L63 = %9.8f;\n',Lss(6,3));
fprintf(init,'double L64 = %9.8f;\n',Lss(6,4));
fprintf(init,'double L65 = %9.8f;\n',Lss(6,5));
fprintf(init,'double L66 = %9.8f;\n',Lss(6,6));
fprintf(init,'double L67 = %9.8f;\n',Lss(6,7));
fprintf(init,'double L68 = %9.8f;\n',Lss(6,8));
fprintf(init,'double L69 = %9.8f;\n',Lss(6,9));
fprintf(init,'double L610 = %9.8f;\n',Lss(6,10));
fprintf(init,'double L611 = %9.8f;\n',Lss(6,11));
fprintf(init,'double L612 = %9.8f;\n\n',Lss(6,12));

fprintf(init,'double K11 = %11.8f;\n',K(1,1));
fprintf(init,'double K12 = %11.8f;\n',K(1,2));
fprintf(init,'double K13 = %11.8f;\n',K(1,3));
fprintf(init,'double K14 = %11.8f;\n',K(1,4));
fprintf(init,'double K15 = %11.8f;\n',K(1,5));
fprintf(init,'double K16 = %11.8f;\n',K(1,6));
fprintf(init,'double K21 = %11.8f;\n',K(2,1));
fprintf(init,'double K22 = %11.8f;\n',K(2,2));
fprintf(init,'double K23 = %11.8f;\n',K(2,3));
fprintf(init,'double K24 = %11.8f;\n',K(2,4));
fprintf(init,'double K25 = %11.8f;\n',K(2,5));
fprintf(init,'double K26 = %11.8f;\n',K(2,6));
fprintf(init,'double K31 = %11.8f;\n',K(3,1));
fprintf(init,'double K32 = %11.8f;\n',K(3,2));
fprintf(init,'double K33 = %11.8f;\n',K(3,3));
fprintf(init,'double K34 = %11.8f;\n',K(3,4));
fprintf(init,'double K35 = %11.8f;\n',K(3,5));
fprintf(init,'double K36 = %11.8f;\n',K(3,6));

fclose(init);
```

Bibliography

[1] "DANDE: Drag and Atmospheric Neutral Density Explorer". http://csmarts.colorado.edu/mission.htm.

[2] Fast, Affordable, Science and Technology SATellite (FASTSAT) Microsatellite: NASA Facts. Technical report, National Aeronautics and Space Administration, Huntsville, AL.

[3] Organism/Organic Exposure to Orbital Stresses (O/OREOS) Nanosatellite: An Astrobiology Technology Demonstration. Technical report, National Aeronautics and Space Administration, Moffett Field, CA.

[4] TruLink User Guide. Included in TruLink Wireless USB package.

[5] DoD Space Test Program Secondary Payload Planner's Guide For Use On The EELV Secondary Payload Adapter. Technical report, DoD Space Test Program, Kirtland Air Force Base, NM, November 2004.

[6] "Planet Physics: Direction Cosine Matrix". http://planetphysics.org/encyclopedia/DirectionCosineMatrix.html, August 2005.

[7] "Ball Aerospace STPSat-2 Satellite Launches Aboard STP-S26 Mission". http://www.space-travel.com/reports/Ball_Aerospace_STPSat_2_Satellite_Launches_Aboard_STP_S26_Mission_999.html, November 2010.

[8] CASTOR Design Document. Technical report, Massachusetts Institute of Technology, Cambridge, MA, November 2010.

[9] Comtech AeroAstro Astro-200 Fact Sheet. Technical report, Ashburn, VA, 2010.

[10] ExoPlanetSat Design Document. Technical report, Massachusetts Institute of Technology, Cambridge, MA, December 2010.

[11] FASTRAC Press Kit 2010. Technical report, University of Texas at Austin, Austin, TX, 2010.

[12] "FASTSAT Capabilities". `http://dynetics.com/descriptionpage.php?id=FastsatCapabilities\&from=space`, 2010.

[13] "Gunter's Space Page: O/OREOS". `http://space.skyrocket.de/doc_sdat/ooreos.htm`, 2011.

[14] "Gunter's Space Page: STPSat-2". `http://www.skyrocket.de/space/doc_sdat/stpsat-2.htm`, 2011.

[15] "University Nanosat Program (UNP)". `http://www.vs.afrl.af.mil/UNP/index.html`, March 2011.

[16] Ismail H. Altas. Dynamic Model of a Permanent Magnet DC Motor. Technical report, August 2007.

[17] Steven F. Barrett. *Embedded Systems Design with the Atmel AVR Microcontroller.* Synthesis Lectures on Digital Circuits and Systems. Morgan & Claypool, 2010.

[18] Matt Bennett. RAX: The Radio Aurora eXplorer. Technical report, University of Michigan, April 2009.

[19] Sangbum Cho, Jinglai Shen, and N. Harris McClamroch. Mathematical Models for the Triaxial Attitude Control Testbed. Technical report, University of Michigan, Ann Arbor, MI.

[20] Charlie Devivero. Reaction Wheel Engineering Model. Technical report documenting reaction wheel characteristics, September 2010.

[21] Joseph J. Distefano, Allen R. Stubberud, and Ivan J. Williams. *Feedback and Control Systems.* Schaum's Outlines. McGraw-Hill, 1990.

[22] Richard M. Dolphus. Gyroless Attitude Determination. Technical Memorandum, Aerospace Corporation, El Segundo, CA, October 2006.

[23] Jessica A. Eisenstein. *Design and construction of a Helmholtz coil apparatus for nanoparticle heating.* Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2006.

[24] Mary Farmer, Gordon Parker, Lyon B. King, Peter Radecki, and Jeff Katalenich. Nanosatellite Attitude Control System for the Oculus: A Space-Based Imaging Platform for Space Situational Awareness. Technical report, Michigan Technological University, Houghton, MI.

[25] Travis Z. Fullem. *Radiation Detection Using Single Event Upsets in Memory Chips.* Masters Thesis, State University of New York at Binghamton, Vestal, NY, February 2007.

[26] Jeff Ganley and Abbie Stewart. Nanosat-6 User's Guide. User's Guide, Air Force Research Laboratory, Kirtland Air Force Base, NM, January 2009.

[27] J. Greenbaum, S. Stewart, G. Holt, E. Rogstad, R. Zwerneman, T. Campbell IV, and E. G. Lightsey. A Combined Relative Navigation and Single Antenna Attitude Determination Sensor on the FASTRAC Student-Built Nanosatellite Mission. Technical report, University of Texas at Austin, Austin, TX.

[28] Donald T. Greenwood. *Principles of Dynamics.* Prentice Hall, second edition, July 1987.

[29] Christopher D. Hall. When Spacecraft Won't Point. Technical report, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[30] David M. Harland and Ralph D. Lorenz. *Space Systems Failures: Disasters and Rescues of Satellites, Rockets and Space Probes.* Praxis Publishing, Chichester, UK, 2005.

[31] Lou Hernandez. Air Force Space & Missile Systems Center.

[32] Mark Hilstad, Swati Mohan, and Christopher Pong. SPHERES Manual. User's Guide, Massachusetts Institute of Technology, February 2010.

[33] Jonathan P. How. Feedback Control Systems. Course Notes, Massachusetts Institute of Technology, Aeronautical and Astronautical Engineering Department, September 2009.

[34] Jonathan P. How. Stochastic Estimation and Control. Course Notes, Massachusetts Institute of Technology, Aeronautical and Astronautical Engineering Department, March 2010.

[35] Jae Jun Kim and Brij N. Agrawal. Automatic Mass Balancing of Air-Bearing-Based Three-Axis Rotational Spacecraft Simulator. Technical report, Naval Postgraduate School, Monterey, CA, May 2009.

[36] Matthew W. Knutson. ADCS Design for Dual-Spin CubeSat. Presentation to SSL, 14 April 2011.

[37] Gavin C. McCorry. Three-Axis Attitude Control and FalconSat-5. Technical report, United States Air Force Academy, Colorado Springs, CO.

[38] Marian Mitescu and Ioan Susnea. *Microcontrollers in Practice.* Advanced Microelectronics. Springer Berlin Heidelberg, 2005.

[39] Norman S. Nise. *Control Systems Engineering.* John Wiley & Sons, Pomona, CA, fourth edition, 2004.

[40] Shahin T. Nudehi, Umar Farooq, Aria Alasty, and Jimmy Issa. Satellite attitude control using three reaction wheels. Technical report, June 2008.

[41] Michael O'Connor. Interview regarding the mission and design of TERSat, March 2011.

[42] Katsuhiko Ogata. *Modern Control Engineering.* Instrumentation and Controls Series. Prentice Hall, Englewood Cliffs, N.J., 1970.

[43] Mason Peck and Kristopher Young. "Cornell University Space Systems Design Studio CUSat Satellite Project Sponsorship Packet". `http://cusat.cornell.edu/docs/sponsorship_packet.php`.

[44] Christopher Pong. Air Bearing Vibration Analysis using Exoplanet Optical System, March 2011.

[45] J. Prado, G. Bisiacchi, L. Reyes, E. Vicente, F. Contreras, M. Mesinas, and A. Juarez. Three-Axis Air-Bearing Based Platform for Small Satellite Attitude Determination

and Control Simulation. Technical report, Universidad Nacional Autonoma de Mexico, Mexico City, Mexico, 2005.

[46] Mark L. Psiaki, Francois Martel, and Parimal K. Pal. Three-Axis Attitude Determination via Kalman Filtering of Magnetometer Data. Technical report, Cornell University, January 1989.

[47] Alvar Saenz-Otero. *Design Principles for the Development of Space Technology Maturation Laboratories Aboard the International Space Station.* Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, May 2005.

[48] Jana L. Schwartz and Christopher D. Hall. The Distributed Spacecraft Attitude Control System Simulator: Development, Progress, Plans. Technical report, Virginia Polytechnic Institute & State University.

[49] Jana L. Schwartz, Mason A. Peck, and Christopher D. Hall. Historical Review of Air-Bearing Spacecraft Simulators. Technical report, July 2003.

[50] Ryan E. Snider. *Attitude Control of a Satellite Simulator Using Reaction Wheels and a PID Controller.* Masters Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, March 2010.

[51] Craig I. Underwood, Guy Richardson, and Jerome Savignol. In-orbit results from the SNAP-1 nanosatellite and its future potential. Technical report, University of Surrey, Surrey, UK, November 2002.

[52] James R. Wertz and Wiley J. Larson. *Space Mission Analysis and Design.* Space Technology Series. Microcosm Press, Hawthorne, CA, third edition, 1999.

[53] Brady W. Young. *Design and Specification of an Attitude Control System for the DANDE Mission.* Masters Thesis, University of Colorado, Boulder, CO, 2008.

[54] Dennis G. Zill and Michael R. Cullen. *Advanced Engineering Mathematics.* Jones and Bartlett, SudBury, MA, third edition, 2006.