THE DESIGN AND TESTING OF A THREE-DEGREE-OF-FREEDOM SMALL SATELLITE SIMULATOR USING A LINEAR CONTROLLER WITH FEEDBACK LINEARIZATION AND TRAJECTORY GENERATION

by

Marina A. Samuels

A thesis submitted in partial fulfillment of the requirements for the degree
of
MASTER OF SCIENCE
in
Mechanical Engineering

Approved:

Dr. R. Rees Fullmer
Major Professor

Dr. Steven L. Folkman
Committee Member

Dr. David Geller
Committee Member

Dr. Mark R. McLellan
Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah
2014
Abstract

The Design and Testing of a Three-Degree-of-Freedom Small Satellite Simulator Using a Linear Controller with Feedback Linearization and Trajectory Generation

by

Marina A. Samuels, Master of Science
Utah State University, 2014

Major Professor: Dr. R. Rees Fullmer
Department: Mechanical and Aerospace Engineering

A small satellite simulator with attitude determination and control was designed and implemented in hardware. The simulator consists of inertial sensors for attitude determination and a pyramidal four-wheel momentum exchange system as the control actuators. A linearized PV controller with trajectory generation and feedback linearization was implemented, with the focus on controlling yaw. The simulator was tested on a spherical air bearing platform to allow three-degree-of-freedom operation. The simulator software was developed to read measurements from the sensors, apply the control algorithm, and send commands to the actuators. A data processing routine was developed. Electromechanical testing for the system as well as test results are presented.

(127 pages)
Public Abstract

The Design and Testing of a Three-Degree-of-Freedom Small Satellite Simulator Using a Linear Controller with Feedback Linearization and Trajectory Generation

by

Marina A. Samuels, Master of Science
Utah State University, 2014

Major Professor: Dr. R. Rees Fullmer
Department: Mechanical and Aerospace Engineering

A small satellite simulator with attitude determination and control was designed and implemented in hardware. The simulator consists of inertial sensors for attitude determination and a pyramidal four-wheel momentum exchange system as the control actuators. A linearized PV controller with trajectory generation and feedback linearization was implemented, with the focus on controlling yaw. The simulator was tested on a spherical air bearing platform to allow three-degree-of-freedom operation. The simulator software was developed to read measurements from the sensors, apply the control algorithm, and send commands to the actuators. A data processing routine was developed. Electromechanical testing for the system as well as test results are presented.
Acknowledgments

First, I would like to express my gratitude to my major professor, Dr. Rees Fullmer, for allowing me to have this opportunity. He has guided me throughout this process as an advisor and mentor and has provided continual encouragement. He has understood my goals and has motivated me.

Second, I would like to thank Dr. Steven Folkman and Dr. David Geller for their instruction and guidance throughout this process.

I would like to also thank Christine Spall for her constant guidance and genuine interest in me as a student.

I would like to acknowledge those at the Space Dynamics Laboratory who provided funding for this work: Burt Lamborn, Quinn Young, Chad Fish, Steve Wasson, and Brian Bingham.

I would like to acknowledge Ivan Jimenez, Jan Sommer, and Landon Terry for their technical support and contribution in this work.

I also wish to express gratitude to many cheerleaders along the way: Ivan Jimenez, Yari Ventura, Melissa and Lowell Bishop, Jacob Bishop, Tyson Boardman, and Chris Lewis for their support and encouragement.

Finally, and most of all, I would like to thank my parents, Dan and Maria Samuels; my grandparents, Dean and Georgia Samuels; and my sister Elena Samuels for loving me and believing in me.

Marina A. Samuels
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Public Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>ix</td>
</tr>
<tr>
<td>List of Figures</td>
<td>x</td>
</tr>
<tr>
<td>Acronyms</td>
<td>xiii</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Small Satellite Attitude Control</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Approach</td>
<td>3</td>
</tr>
<tr>
<td>1.4.1 Determine the System Model and Attitude Control Algorithm</td>
<td>3</td>
</tr>
<tr>
<td>1.4.2 Design and Build the Reaction Wheel ADCS Satellite Hardware Sim-</td>
<td>3</td>
</tr>
<tr>
<td>ulator</td>
<td></td>
</tr>
<tr>
<td>1.4.3 Design the ADCS Software Structure</td>
<td>3</td>
</tr>
<tr>
<td>1.4.4 Provide Electromechanical Testing</td>
<td>3</td>
</tr>
<tr>
<td>1.4.5 Perform Experimental Tests</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Thesis Overview</td>
<td>4</td>
</tr>
<tr>
<td><strong>2 Background and Literature Review</strong></td>
<td>5</td>
</tr>
<tr>
<td>2.1 2.1 Spacecraft Attitude</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Spacecraft Attitude Control Mechanisms</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1 Inertial Control</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Momentum Exchange Control</td>
<td>7</td>
</tr>
<tr>
<td>2.2.3 Attitude Control in Space</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Commercial Miniature Reaction Wheel Systems</td>
<td>9</td>
</tr>
<tr>
<td>2.4 University Satellite Simulators</td>
<td>9</td>
</tr>
<tr>
<td>2.4.1 Overview of Simulators</td>
<td>9</td>
</tr>
<tr>
<td>2.4.2 Utah State University</td>
<td>10</td>
</tr>
<tr>
<td>2.4.3 Air Force Institute of Technology</td>
<td>10</td>
</tr>
<tr>
<td>2.4.4 Massachusetts Institute of Technology</td>
<td>12</td>
</tr>
<tr>
<td>2.4.5 Virginia Polytechnic Institute and State University</td>
<td>13</td>
</tr>
<tr>
<td>2.4.6 California Polytechnic State University</td>
<td>13</td>
</tr>
<tr>
<td>2.4.7 Naval Postgraduate School</td>
<td>14</td>
</tr>
<tr>
<td>2.4.8 Universidad Nacional Autonoma de Mexico</td>
<td>15</td>
</tr>
</tbody>
</table>
3 Dynamic Modeling and Control Strategy

3.1 Dynamic Modeling

3.1.1 Derivation of Rotation Matrix from Wheel to Body Axes

3.1.2 Reaction Wheel Modeling

3.2 Linearized Control

3.3 Quaternion Trajectory Generation

4 Hardware Design

4.1 Goal of the Design

4.2 Air Bearing System

4.2.1 Fundamentals

4.2.2 Modifications to the Air Bearing Platform

4.3 Attitude Sensing

4.4 Attitude Control

4.4.1 Motor Selection

4.4.2 Wheel Design

4.5 On-board Mini Processor

4.6 Power Supply

4.7 Custom Platform Design

5 Software Development

5.1 Overview of Architecture

5.1.1 Architecture

5.1.2 Overview of Key Files

5.2 User Input

5.3 Attitude Determination

5.3.1 Choosing the IMU Output

5.3.2 IMU to Body Coordinate System

5.3.3 Calculating the Quaternion

5.3.4 IMU Sensors

5.4 Motor Feedback Sensing

5.5 Attitude Control

5.6 Output File

6 Electromechanical Testing

6.1 Motor Speed Controller

6.2 Reaction Wheel Characterization

6.3 Magnetometer

6.4 Gyro

6.5 Filtering Technique

6.5.1 Theory

6.5.2 Implementation
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Flywheel Dimensions</td>
<td>42</td>
</tr>
<tr>
<td>4.2 Flywheel Inertia</td>
<td>43</td>
</tr>
<tr>
<td>5.1 Data Packet Rates</td>
<td>52</td>
</tr>
<tr>
<td>6.1 Wheel 0 Autotuning Gains</td>
<td>57</td>
</tr>
<tr>
<td>6.2 Wheel 1 Autotuning Gains</td>
<td>58</td>
</tr>
<tr>
<td>6.3 Wheel 2 Autotuning Gains</td>
<td>58</td>
</tr>
<tr>
<td>6.4 Wheel 3 Autotuning Gains</td>
<td>58</td>
</tr>
<tr>
<td>6.5 Wheel Model Parameters</td>
<td>59</td>
</tr>
<tr>
<td>6.6 Gyro Bias Correction Values</td>
<td>63</td>
</tr>
<tr>
<td>E.1 User Input Variables</td>
<td>106</td>
</tr>
<tr>
<td>F.1 Output File Column Headings</td>
<td>107</td>
</tr>
<tr>
<td>F.1 Output File Column Headings</td>
<td>108</td>
</tr>
<tr>
<td>I.1 Wheel Speed Step Responses</td>
<td>114</td>
</tr>
<tr>
<td>I.2 Wheel Time Constants</td>
<td>114</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>SSACS Table with Gas Jet and Mechanical Gyro, 1991</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Gas Jet Table, Skipper Flight Control, 1994</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Wheel-Based Simulator, 2004</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Wheels on Axes</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Wheels Off-Axes</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Wheel Dynamics Diagram</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Trapezoidal Trajectory Generation Profile</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Existing Air Bearing Components</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Modified Air Bearing Base</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Modified Air Intake</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Flywheel Model</td>
<td>41</td>
</tr>
<tr>
<td>4.5</td>
<td>BB Processor</td>
<td>43</td>
</tr>
<tr>
<td>4.6</td>
<td>MaxAmps Batteries with connectors</td>
<td>44</td>
</tr>
<tr>
<td>4.7</td>
<td>Printed Table Platform</td>
<td>45</td>
</tr>
<tr>
<td>4.8</td>
<td>Printed Electronics Box</td>
<td>46</td>
</tr>
<tr>
<td>4.9</td>
<td>Simulator on Air Bearing</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>BB Cape</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Power Regulator</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Reaction Wheel Speed Response to 45% Duty Cycle</td>
<td>60</td>
</tr>
<tr>
<td>6.2</td>
<td>Wheel Speed Curves</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Yaw Drift before Compensation</td>
<td>63</td>
</tr>
</tbody>
</table>
6.4 Yaw Drift after Compensation .............................................. 64
6.5 FFT on Gyro Yaw Rate ..................................................... 67
6.7 FFT on Gyro Roll Rate ..................................................... 67
6.6 FFT on Gyro Pitch Rate ..................................................... 68
6.8 Filtered Gyro Yaw Rate .................................................... 69
6.9 Filtered Gyro Pitch Rate .................................................... 70
6.10 Filtered Gyro Roll Rate ................................................... 70
6.11 FFT on Motor Speed Measurements ................................. 71
6.12 Filtered Motor Speed Measurements ................................. 72
6.13 FFT on Motor Current Measurements ............................... 73
6.14 Filtered Motor Current Measurements ............................... 74
7.1 Table Yaw Position ......................................................... 77
7.2 Table Pitch Position ....................................................... 77
7.3 Table Roll Position ........................................................ 78
7.4 Table Position Combined ................................................ 78
7.5 Position Error .............................................................. 79
7.6 Converted Quaternion Error ............................................. 79
7.7 Table Speed Yaw ........................................................... 80
7.8 Table Acceleration Yaw .................................................. 80
7.9 Table Speed Pitch .......................................................... 81
7.10 Table Acceleration Pitch ................................................ 81
7.11 Table Speed Roll .......................................................... 82
7.12 Table Acceleration Roll ................................................ 82
7.13 Yaw Torque Command vs. Measured ............................... 83
7.14 Pitch Torque Command vs. Measured ............................... 83
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.15</td>
<td>Roll Torque Command vs. Measured</td>
<td>84</td>
</tr>
<tr>
<td>7.16</td>
<td>Crossterm Values</td>
<td>84</td>
</tr>
<tr>
<td>7.17</td>
<td>Command vs. Measured Wheel Speeds</td>
<td>86</td>
</tr>
<tr>
<td>7.18</td>
<td>Duty Cycle Commands to Wheels</td>
<td>86</td>
</tr>
<tr>
<td>7.19</td>
<td>Command vs. Measured Wheel Torques</td>
<td>87</td>
</tr>
</tbody>
</table>
Acronyms

ADCS  Attitude Determination and Control System
BB    Beagle Bone
BBB   Beagle Bone Black
BFCS  body-fixed coordinate system
DOF   degree of freedom
ECI   Earth Centered Inertial
PV    Proportional Velocity
SDL   Space Dynamics Laboratory
Chapter 1
Introduction

1.1 Small Satellite Attitude Control

The Cubesat was introduced in 1999 by Stanford University and California Polytechnic State University as a 10 cm-by-10 cm-by-10 cm nanosatellite weighing 1 kg and as an educational tool for university students [1]. It standardized nanosatellite design and thus provided a relatively low cost design to launch experience. The Cubesat has become a standard platform for demonstrating technology, such as docking and reconfiguration maneuvers, remote sensing, and imaging; as well as for making scientific measurements and Earth observations such as earthquake detection and prediction.

Every satellite requires knowledge of its attitude, or its orientation in space with respect to a celestial coordinate reference frame [2]. In addition, attitude control is required for either attitude maneuvering or Earth-pointing stabilized satellites. Attitude control for nanosatellites is especially challenging due to the small size, weight, and power consumption limits. Attitude control is provided by either passive or active mechanisms, or a combination of the two. Magnetic stabilization, which has been commonly used in small satellites, can be either passive or active.

For nanosatellites that require three-axis precise attitude control, such as attitude maneuvering satellites, magnetic control cannot produce the level of torques or accuracies required to point the satellite or on-board instrument. Rather, active control in the form of reaction wheels is necessary for achieving the desired precision pointing capability [3–7]. Much research and development has been dedicated in recent years to answering the need for precise attitude control for nanosatellites. Within the last several years, miniature reaction wheels systems for nanosatellites have become available on the market from such companies as Sinclair Interplanetary, Blue Canyon Technologies, and others [3,8,12]. As miniature
reaction wheels are becoming increasingly developed and available, it is clear that the need
for developing precision pointing capability for nanosatellites is being addressed in the space
industry and that reaction wheels are becoming the state-of-the-art.

1.2 Motivation

The Space Dynamics Laboratory (SDL) is one of the fourteen University Affiliated Re-
search Centers in the United States and has a long history in space related research including
sensors and satellite systems. SDL has developed and launched several small satellite mis-
sions and in recent years has identified precision pointing as a major focus area for future
nanosatellite missions. As several miniature reaction wheel systems have been recently in-
troduced into the market, the SDL is especially interested in using a custom or commercially
available reaction wheel system for precise attitude control in future missions. In addition,
the ground-based test platform may be used for experimental testing of algorithms before
the launch experience, model validation, and implementation of advanced control theory.

1.3 Objectives

The overall objective of this thesis is to support the SDL’s initiative in precision attitude
control through the design of a three degree-of-freedom (DOF) attitude determination and
control system (ADCS) using reaction wheels. This reaction-wheel only satellite simulator
is intended to be a low cost, ground-based testbed for experimental testing and validation.
This objective will be accomplished by doing the following:

- Develop an analytic model of the reaction wheel system and determine the attitude
  control strategy
- Design a 3 DOF ADCS satellite hardware simulator with reaction wheel only control
- Design a comprehensive software structure for implementation of attitude determina-
  tion and control on the simulator
- Provide electromechanical testing of the hardware components and ADCS algorithms
- Provide experimental testing of the pointing controller in the ADCS

1.4 Approach

1.4.1 Determine the System Model and Attitude Control Algorithm

A mathematical model for the reaction wheel system will be presented. The equations for the implementation of a linearized PV angular position controller with added feedback linearization and trapezoidal trajectory generation will be developed.

1.4.2 Design and Build the Reaction Wheel ADCS Satellite Hardware Simulator

A satellite hardware simulator will be designed and built as a low-cost platform consisting of attitude sensing, a credit-card-sized on-board processor, and reaction wheels as the sole control actuators. The platform will operate on a modified 3 DOF air bearing system.

1.4.3 Design the ADCS Software Structure

The software interface with all of the sensors will be integrated into a modular program written in C++ that allows for multiple controller strategies to be easily implemented and tested. The program will be designed to read a user-friendly input file that includes the desired attitude commands, system constants, and control gains; as well as output all relevant test data into a file. Documentation on the test protocol and software structure will be provided.

1.4.4 Provide Electromechanical Testing

The components for attitude sensing and control will be tested for proper function and signal feedback. The reaction wheels will be characterized. Power consumption will be addressed. Issues with sensor accuracy and noise, control actuator limits, and processing capabilities will be identified.
1.4.5 Perform Experimental Tests

The ADCS hardware simulator will be used to do experimental testing of the derived controller algorithm. Gains will be adjusted to explore simulator response. Experimental results will be reported.

1.5 Thesis Overview

Background information and a literature review are provided in Chapter 2. The reaction wheel-simulator model and the ADCS control strategy will be developed in Chapter 3. The design of the ADCS and reaction wheels, along with hardware selection and modification is detailed in Chapter 4. The software structure and attitude determination method for the ADCS will be outlined in Chapter 5. Electromechanical testing including characterization of the reaction wheels and sensor issues will be presented in Chapter 6. Experimental results will be presented in Chapter 7. Chapter 8 will detail conclusions and suggestions for future work.
Chapter 2

Background and Literature Review

2.1 Spacecraft Attitude

The attitude of a spacecraft attitude is its orientation in space with respect to a celestial coordinate reference frame [2]. While multiple coordinate systems are commonly used for the inertial reference frame, the Body Fixed Coordinate System (BFCS) consists of three fixed orthogonal axes on the spacecraft. Attitude determination for a spacecraft generally involves a three dimensional rotation, represented by a rotation matrix \( R \), from the spacecraft’s coordinate system, or the body fixed coordinate system (BFCS), and the Earth Centered Inertial (ECI) reference frame.

A spacecraft’s attitude can be represented in many ways, such as Euler angles, an axis of rotation and an angle, a rotation matrix, and quaternions.

Euler angles describe the attitude as rotations about the \( X \), \( Y \), and \( Z \) axes, which are the BFCS axes in this paper. These rotations are also referred to as roll (\( \phi \)), pitch (\( \theta \)), and yaw (\( \psi \)), respectively.

The angle and axis representation consists of a single axis of rotation, called the eigen-axis of rotation and represented by vector \( e \), about which the spacecraft rotates an angular amount \( \alpha \).

The quaternion representation is often used to describe a spacecraft’s attitude [3]. A quaternion is a four-vector, a hyper complex number, defined as the following:

\[
q = q_1 i + q_2 j + q_3 k + q_4
\]

The scalar component of the quaternion is \( q_4 \), listed in this paper as the final component of the vector, while \( i, j, k \) are traditional unit vectors in three-space. An alternative notation
used in this paper to denote the quaternion q is:

\[ q = < q_1, q_2, q_3, q_4 > \]

Important properties of the quaternion include the following:

1. The norm of the quaternion \( q \) equals 1.

2. \( q^T = -q \)

Quaternions are preferred due to their numerical stability and lack of singularities during a full revolution of rotation. The attitude of the simulator will be represented in quaternions.

A rotation matrix is a 3x3 matrix containing information about the angular rotation of the spacecraft. It may be derived from the Euler angles or quaternion attitude representations, or vice versa.

### 2.2 Spacecraft Attitude Control Mechanisms

Control hardware on board the spacecraft is responsible for creating the torques that will manipulate the spacecraft’s attitude. The four typical mechanisms for producing the necessary torques are earth’s magnetic field, ion or gas reaction thrusters, solar radiation pressure, and momentum exchange devices. The first three mechanisms are classified as inertial controllers while momentum exchange devices are excluded from this group \([5,7]\).

#### 2.2.1 Inertial Control

Inertial controllers are termed such because they change the overall inertial angular momentum of the spacecraft. Magnetic controllers are used as torquerods and are made up of a magnetic core and a coil. When current passes through the coil, or in other words when the coil is energized, the torquerod produces a magnetic moment which then interacts with the Earth’s magnetic field to produce a resulting torque. Magnetic torque can be implemented as passive or active control and can provide smooth and continuous torques. Disadvantages for using magnetic control are that the produced torques are fairly low, the magnetic moments
depend on the Earth’s magnetic field strength in an inverse cube relationship with the distance away from the Earth, and the direction of the torque is dependent on the orientation of the spacecraft relative to the Earth’s magnetic field. Another drawback particular to active magnetic control is that a magnetometer to measure the Earth’s magnetic field is required, but its measurements are affected by the magnetic field produced by the torqrod when it is powered.

Another inertial controller involves using solar pressure on reflecting surfaces of the spacecraft. This setup generally consists of two solar panels, being the primary source of power for the spacecraft, and two attached flaps that are flexible. The torques produced off of the panels and flaps are limited by the geometry and reflexivity properties of the surfaces, the amount of movement of the flaps relative to the rigid spacecraft, and power losses due to changing position of the solar panels. Solar pressure cannot produce high enough torque values and or torques about all three axes necessary for attitude-maneuvering spacecraft.

A final example of an inertial controller is the reaction thruster, which includes cold gas, solid or liquid, and electrical propulsion means. Thrusters have the ability to provide both translational and angular manipulation, but they also tend to be heavier and provide nonlinear, non-smooth torques.

### 2.2.2 Momentum Exchange Control

In contrast with inertial controllers, a momentum exchange device does not change the overall spacecraft inertial angular momentum but rather, uses rotating masses on the spacecraft to produce a torque that acts on the spacecraft. The net angular momentum remains the same and is therefore conserved. Examples of momentum exchange devices are reaction or momentum wheels and control moment gyros (CMGs). While CMGs are capable of producing very high torques, they are typically too heavy and not practical for nanosatellites.

Reaction wheels typically consist of an electrical motor with attached flywheel. The torque produced depends on the speed, acceleration, and moment of inertia of the rotating mass. The purpose of the flywheel is to increase the moment of inertia of the rotating mass
in order to reach a higher torque. The motor outputs a torque to create the angular motion, which in turn exerts an equal and opposite torque on the spacecraft. Reaction wheels are typically capable of producing 0.05-0.2 N-m. Miniature reaction wheels have the capability to produce the necessary torques for attitude-maneuvering nanosatellites while satisfying the design constraints unique to such small satellites.

2.2.3 Attitude Control in Space

Over the last decade, Cubesats in space flight have traditionally employed a combination of passive attitude control and/or active control, including magnetic torque coils and micro reaction wheels, for position control. In a survey of pico- and nanosatellite missions conducted in 2010, it was found that near forty percent used active attitude control, near forty percent used passive control, and twenty percent used no control [13]. Furthermore, only fifteen percent used attitude control to point an instrument, in most cases a camera or radiation detector. According to the survey, magnetic control has been widely used as passive or active control in pico- and nanosatellites, and spin-stabilization and gravity gradient boom are also employed. In contrast, the use of momentum wheels and thrusters for precision pointing has been found to be less commonly implemented.

From the 2010 survey, among the satellites equipped with reaction wheels, the performance in space flight was not satisfactory, though still promising. In 2007, Sinclair noted that precise pointing control in small satellites was a recent milestone, but that the technology for nanosatellites (he noted under 10kg) was only in the very early stages of development [3]. For example, Kayal et al also highlighted the need for miniaturization of reaction wheels for nanosatellites, described custom wheels capable of producing 4x10^-5Nm that were designed by the Technical University Berlin for seven Cubesats launched in late 2007, and characterized that work as their first steps in miniaturization of reaction wheels [14].

Janson et al described the custom 3-axis reaction wheels launched on a picosatellite and candidly reported on the performance issues during space flight and lessons learned [15]. The custom reaction wheel consisted of a miniature brushless DC motor with attached flywheel. Findings included that the Earth’s magnetic field created some drag on the wheel,
eddy currents were induced by the rotating wheel because of the material’s high magnetic properties, and that there was a need for momentum dumping through magnetic control to complement the reaction wheels. These comments are still relevant for current miniature reaction wheel development.

2.3 Commercial Miniature Reaction Wheel Systems

Within the last several years, miniature reaction wheels have finally begun to be available on the market. Sinclair noted in 2007 that there was a total lack of miniature reaction wheel systems in the market at that time. Now there are several to choose from, although not all of them are necessarily low-cost options. Several companies now offer reaction wheel systems for nanosatellites, such as Sinclair Interplanetary (starting in 2007), Blue Canyon Technologies, Moog Bradford, MicroSat Systems Canada Inc, and Astro- und Feinwerktechnik Adlershof GmbH with the University of Berlin. These reaction wheels systems typical come in a three-wheel, triaxial configuration or as a four-wheel pyramidal solution.

Of special note, Sinclair revolutionized the original concept of the reaction wheel by proposing a solution in 2007 that eliminated the wheel but instead maximized the rotor inertia in the DC motor. Many issues, such as vibration and eddy currents were eliminated or reduced through this design.

2.4 University Satellite Simulators

Satellite simulators have long been used to test and verify satellite hardware and software before launch, especially in a university setting. A brief survey of satellite simulators built in the last decade was conducted in order to become familiar with the different combinations of actuators as well as the control theories that have been used. While not a complete list, the following sections illustrate the variations in implementation of small satellite simulators.

2.4.1 Overview of Simulators

Satellite simulators have used both passive and active control, and generally use either
one or a combination of control moment gyros, reactions wheels, fans, thrusters, etc. \[16,33\]
The most common momentum exchange devices are reaction wheels and control moment
gyros (CMGs). While CMGs generally provide larger torques, drawbacks include greater
power consumption, size, and weight \[34\]. Several cases surveyed used only reaction wheels
as well, although various control techniques were implemented in the surveyed testbeds; both
linear and nonlinear controllers \[16,19,26,32,33\]. Because reaction wheels add nonlinear
terms to the system, feedback linearization has been applied to small satellite systems \[18,
20,22,23\].

### 2.4.2 Utah State University

As a brief mention of the history of other small satellite simulators developed at Utah
State University, several pictures are provided of the different experimental platforms: Figure
2.1 of the SSACS table developed by John Jones and Eric Hoffer, Figure 2.2 of another gas
jet table developed for Skipper Flight Control, and Figure 2.3 of a wheel-based simulator
developed by Frank Tebbe and Dr. Rees Fullmer.

### 2.4.3 Air Force Institute of Technology

For example, at the Air Force Institute of Technology (AFIT) several iterations of the
SimSat tabletop simulator have been used to test reaction-wheel type systems controllers
on a spherical air bearing platform as early as 1999. The first generation of SimSat was a
250-lb dumbbell-shaped simulator where two masses were suspended rigidly on either side
of the floating sphere. The on-board processor used was the proprietary dSPACE AutoBox.
Available on the simulator were three reaction wheels aligned parallel with the principal axes
as well as nitrogen cold-gas thrusters for attitude control, plus a gyro for attitude sensing.
In 2005, Smith investigated the accuracy of attitude determination based on only feedback
from the control actuators and a spacecraft model \[35\]. Smith installed a fiber-optic gyro
for comparison with attitude calculated from control actuator telemetry data, used only the
reaction wheels for attitude control, and implemented a PD dual controller. In 2007, Hines
used this simulator to experimentally determine a more accurate estimate of the simulator’s
Fig. 2.1: SSACS Table with Gas Jet and Mechanical Gyro, 1991

Fig. 2.2: Gas Jet Table, Skipper Flight Control, 1994
moment of inertia and relate that to change in fuel mass [36].

SimSat II is the second generation of AFIT’s simulator, designed by Roach, et al, as a tabletop version with six fan thrusters and three reaction wheels for attitude control, an IMU for attitude sensing, and again the dSPACE on-board processor [30]. In 2009, Macfarland implemented near real-time optimal control on the SimSat II, when the reaction wheels were not yet included [37]. In 2010, Snider implemented a linear PID controller using three reaction wheels [33]. In 2011, McChesney added a CMG array and implemented PID control using measured gyro rates for the derivative portion (therefore PIV control) and added feedback linearization [22]. In 2012, Padro implemented a star tracker camera for attitude sensing [38].

2.4.4 Massachusetts Institute of Technology

At the Massachusetts Institute of Technology (MIT), in 2005 Chung et al. used the SPHERES testbed, which simulates formation flight with tethers, cold-gas thrusters, and reaction wheels and allows translational and rotational motion; and uses beacons and ultrasound receivers with a gyroscope for state feedback [23]. The processor was a Texas Instruments C6701 Digital Signal Processor. Chung used only the momentum wheels and
tested a linear controller, a gain-scheduled Linear Quadratic Regulator (LQR), as well as a nonlinear controller, Input-State Feedback Linearization. In 2009, Chung et al reported testing additional nonlinear controllers, still excluding thrusters: partial feedback linearization, linearization via momentum decoupling, and backstepping [39].

Also at MIT, in 2011 Crowell developed an ADCS tabletop simulator that has an IMU with rate sensor, accelerometer, and magnetometer for attitude sensing; four reaction wheels for attitude control; and Arduino processors [26]. Crowell implemented a discrete LQR with both a derivative gain and feedforward gains.

2.4.5 Virginia Polytechnic Institute and State University

At the Virginia Polytechnic Institute and State University (VT), Skelton worked on a tabletop platform called the Whorl-I which has full rotation in yaw and ±5° in pitch and roll and uses PC104 components for the processor. This simulator has three reaction wheels and one CMG as the actuators, and three-axis accelerometers and rate gyros for attitude sensing [18]. Skelton tested three combinations of actuators: using only reaction wheels with a Lyapunov controller with feedback linearization; using only the CMG with a gimbal rate feedforward control law; and finally, using both the CMG and reaction wheel set with a closed loop CMG with momentum wheel feedforward control.

Jana Schwartz in 2004 continued work with the Whorl-I and worked with the upgrade of the hardware to the Whorl-II, which was a dumbbell-type platform [24]. Of particular note, an extended Kalman filter was implemented for attitude determination.

Kowalchuk in 2007 provided simulations representing both platforms with various controllers. Kowalchuk used the Whorl-I platform with only the three reaction wheels to test a Lyapunov based angular rate controller and Modified Rodrigues parameters attitude controller [17].

2.4.6 California Polytechnic State University

At CalPoly, several students made upgrades to and worked with a pyramidal four-reaction-wheel tabletop simulator. Developed as part of Jeffrey Logan’s work in 2008, the
main onboard processor is the Gumstix Linux computer, combined with Robostix expansion
board and microcontroller, which communicate wirelessly with a Pentium 4 Dell computer
as the ground station providing Simulink computation \[40\]. Rate gyros and accelerometers
provide attitude sensing. Seth Silva reported on work in 2008 estimating the simulator mass
properties through a system identification algorithm using least squares estimation \[29\]. Jon
Arthur Bowen in 2009 reported on improved on-board attitude determination using a “simple
orbit propagator, magnetometers with a magnetic field look-up table, Sun sensors with an
analytic Sun direction model, and the TRIAD method to combine vector observations into
attitude information” \[41\].

In 2009, Matthew Downs tested two adaptive control theories on the reaction-wheel-
only system: nonlinear direct model reference adaptive control and adaptive output feedback
control (Downs 2009 \[17\]). In 2010, Ryan Kinnet implemented a closed-loop attitude control
onto the actual simulator: a full state feedback proportional-derivative controller \[16\].

2.4.7 Naval Postgraduate School

The Naval Postgraduate School in Monterey, California has a tabletop rotational small
satellite simulator, the bifocal relay mirror spacecraft test bed, which allows up to an 800kg
load and $\pm 20^\circ$ in pitch and roll; uses a PC104 processor; uses a magnetometer, two incli-
nometers, a two axis sun sensor, and an inertial measurement unit (Crossbow Technology
IMU700) for attitude sensing; and three control moment gyros and automatic mass balanc-
ing as control actuators. Jae Jun Kim et al reported on using adaptive control for automatic
mass balancing of the simulator in 2009 \[42\].

David Meissner worked with another simulator in 2009 \[43\]. This simulator resembles
the block structure of a Cubesat, operates on a spherical air bearing, and uses a Hemholz
cage to create an external magnetic field. The onboard processor is a PC104. On board the
simulator are two single-axis inclinometers, an IMU, a sun sensor, one single-axis reaction
wheel, and torque coils. Meissner implemented a unique adaptive mass-balancing approach
to eliminate the gravitational torques on the simulator without using the momentum ex-
change devices (reaction wheel).
Jason Hall worked with a simulator that is levitated by a combination of planar air bearings and propulsion systems; has a PC 104 on board processor; dual rotating thrusters provide translational motion and a miniature single axis control moment gyro provides attitude control; and has a fiber optic gyro, position sensor, and magnetometer for attitude sensing \cite{20}. Hall tests both Lyapunov based control and perturbed feedback linearization as control strategies.

2.4.8 Universidad Nacional Autonoma de Mexico

At the Universidad Nacional Autonoma de Mexico, in Mexico City, Prado et al developed a tabletop 80-kg satellite simulator in 2005 \cite{31}. Attitude sensing was provided by combined gyro, accelerometer and magnetometer; as well as a sun sensor and four Earth sensors. Attitude control is provided by three orthogonal reaction wheels and three torque coils. MC68HC11microcontrollers were selected. Prado et al. implemented proportional control and also developed manual and automatic center of mass adjustment.

In 2007, Vicente-Vivas et al designed another small satellite simulator \cite{44}. This team chose to incorporate a reaction wheel and six magnetic torque coils (two for each principal axis), an IMU, and a PIC18F4431 microcontroller. A linear PI controller was implemented.

2.4.9 University of Florida

In 2008, Frederik A. Leve reported on a unique testbed developed at the University of Florida \cite{45}. The simulator operates on a pivot instead of an air bearing; uses cameras in the test room to track the motion of the simulator, which then becomes the attitude sensing; and does not have an on board processor. Four reaction wheels in a pyramidal configuration but with gimbaled axes are the actuators. A “non-linear exact model knowledge” controller is implemented, which determinates the gimbal rates of the reaction wheel axes.

2.4.10 Yonsei University

In 2008, Dohee Kim performed experimental tests with hardware developed at Yonsei University in Seoul, Korea \cite{32}. The simulator was a tabletop air bearing platform us-
ing three reaction wheels as actuators, an Attitude Heading Reference System for attitude sensing, and a PC104 as the onboard processor. A PID linear controller was tested.
Chapter 3
Dynamic Modeling and Control Strategy

3.1 Dynamic Modeling

The dynamics of the simulator and reaction wheels will be detailed in this section, followed by the control algorithm and trajectory generation. Several chapters of Sidi were used as a reference in this chapter [4–6].

The angular momentum of a rotating body is denoted \( h \) and is defined as Equation (3.1):

\[
h = I \cdot \omega
\]

where \( I \) is the mass moment of inertia of the body rotating at angular velocity \( \omega \). The applied torsional moment \( M \), on a rigid body equals the time rate of change of its angular momentum.

\[
M = \dot{h}
\]

The following identity defines the rate of the change of a vector, denoted \( a \):

\[
\frac{d}{dt} \left|_I \right. a = \frac{d}{dt} \left|_B \right. a + \omega \times a
\]

When applied to the angular momentum vector \( h \), this identity \(3.3\) becomes the Euler’s moment equation:

\[
m = \dot{h}_I = \dot{h}_B + \omega \times h
\]

Euler’s moment equation defines the rate of change of the momentum vector \( h \), as observed in the fixed (inertial, denoted \( I \)) frame, as equal to the rate of change of \( h \) as observed in
the rotating (body, denoted $B$) frame plus the vector product $\omega \times h$.

The angular velocity and acceleration of a reaction wheel about its own axis of rotation produces an angular torque $\dot{h} \cdot \omega$. This reaction wheel torque then acts on the spacecraft and causes a change in the angular motion of the spacecraft. Momentum is conserved in the combined reaction wheels and spacecraft system; in order to apply a torque on the spacecraft about its rotation axis, an equal and opposite angular momentum must be experienced in the reaction wheels. The spacecraft is denoted by the subscript $s$ while the wheel is denoted by $w$. Recalling Euler’s moment equation, this conservation of angular momentum and moment is written as:

$$M_{s+\omega} = \dot{h}_s + \dot{h}_w = 0$$  \hspace{1cm} (3.5)

$$\dot{h}_s = -\dot{h}_w$$  \hspace{1cm} (3.6)

The reference frame of the simulator is considered the body coordinate system, while the reference frame of the wheels must be rotated into the body reference frame. This rotation matrix will be derived later in this chapter. The angular momentum of the system, as observed in the body reference frame, can be written as the sum of the angular momentum of the simulator without the reaction wheels and the angular momentum of the reaction wheels rotated into the body coordinate system:

$$h_{\text{sys}}^B = h_s^B + h_w^B$$  \hspace{1cm} (3.7)

To calculate the torque on the simulator system, the derivative of the angular momentum with respect to the (Earth) inertial reference will be evaluated:
\begin{align*}
M_{s+w} &= \frac{d}{dt} h_{sys}^B \bigg|_I \\
&= \frac{d}{dt} h_{sys}^B \bigg|_B + \omega \times a \\
&= \frac{d}{dt} \left( I_s \cdot \omega_s + h_w^B \right) \bigg|_B + \omega_s \times \left( I_s \cdot \omega_s + h_w^B \right) \\
M_{s+w} &= I_s \cdot \dot{\omega}_s + \dot{h}_w + \omega_s \times \left( I_s \cdot \omega_s + h_w^B \right) 
\end{align*} 

(3.8)

Since \( M_{s+w} \) equals zero, this equation can be rearranged to describe the effect of the torque on the simulator:

\[-\dot{h}_w = I_s \cdot \dot{\omega}_s + \omega_s \times \left( I_s \cdot \omega_s + h_w^B \right)\]  

(3.9)

\begin{align*}
3.1.1 \text{ Derivation of Rotation Matrix from Wheel to Body Axes} \\
\end{align*}

Reaction wheels can be implemented in different configurations. Placing one reaction wheel on each of the \( X, Y, \) and \( Z \) body axes of the simulator is the simplest implementation, although a fourth wheel at an angle and off-axis is usually added for redundancy in case one of the axis-aligned wheels fails.

An alternative configuration is a symmetrical four-wheel arrangement where all the wheels are inclined at an angle \( \beta \) as shown in Figure 3.1. This pyramidal configuration allows the wheels to apply torques in the \( X, Y, \) and \( Z \) directions. In the design for this

![Fig. 3.1: Wheels on Axes](image_url)
work, detailed in the next chapter, a pyramidal configuration with a 45° inclination was chosen, as shown in Figure 3.2. However, the pyramidal wheel setup was implemented in a different way than usual: instead of placing the wheels on the X and Y planar axes of simulator, which would have been convenient mathematically, all four wheels were arranged 45° off-axis.

The four reaction wheels all produce a torque about their respective axes. When resolved into coordinate components and rotated into the three body frame axes, these resultant torques become the control effort $T$ on the simulator. The following method of determining the resultant torques of a four-wheel system on each of the body frame axes is exemplified in [5], and applied to the unique configuration chosen for this work.

The torques about each of the three body frame axes are represented by $T_{cx}$, $T_{cy}$, and $T_{cz}$ while the torques produced by each wheel about its axis of rotation are represented by
$T_0$, $T_1$, $T_2$, and $T_3$,

\[
T_{cx} = (-T_0 - T_1 + T_2 + T_3) \cos(\alpha) \cos(\beta) \quad (3.11)
\]

\[
T_{cy} = (-T_0 + T_1 - T_2 + T_3) \cos(\alpha) \cos(\beta) \quad (3.12)
\]

\[
T_{cz} = (T_0 + T_1 + T_2 + T_3) \cos(\alpha) \cos(\beta) \quad (3.13)
\]

Expressing this in matrix form, where $\hat{T}_{cx}$, $\hat{T}_{cy}$, and $\hat{T}_{cz}$ are the estimated required torque about the body axes, gives:

\[
\begin{bmatrix}
\hat{T}_{cx} \\
\hat{T}_{cy} \\
\hat{T}_{cz}
\end{bmatrix} =
\begin{bmatrix}
\frac{T_{cx}}{\cos(\alpha) \cos(\beta)} \\
\frac{T_{cy}}{\cos(\alpha) \cos(\beta)} \\
\frac{T_{cz}}{\cos(\alpha) \cos(\beta)}
\end{bmatrix}
\begin{bmatrix}
T_0 \\
T_1 \\
T_2 \\
T_3
\end{bmatrix}
\]

\[
\begin{bmatrix}
\hat{T}_{cx} \\
\hat{T}_{cy} \\
\hat{T}_{cz}
\end{bmatrix} =
\begin{bmatrix}
-1 & -1 & 1 & 1 \\
-1 & 1 & -1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
T_0 \\
T_1 \\
T_2 \\
T_3
\end{bmatrix}
\]

\[
\begin{bmatrix}
\hat{T}_{cx} \\
\hat{T}_{cy} \\
\hat{T}_{cz}
\end{bmatrix} = [A] [T]
\]

An optimizing constraint is then imposed to make the matrix $A$ square and invertible: minimizing the total amount of torque required from the reaction wheels. This constraint is represented by minimizing the norm of the matrix $T$. The norm is represented by the following equation:

\[
H = T_0^2 + T_1^2 + T_2^2 + T_3^2
\]

(3.17)
Define the following functions for use in the Lagrangian method:

\[
g_0 = -T_0 - T_1 + T_2 + T_3 - T_{cx} \quad (3.18)
\]
\[
g_1 = -T_0 + T_1 - T_2 + T_3 - T_{cy} \quad (3.19)
\]
\[
g_2 = T_0 + T_1 + T_2 + T_3 - T_{cz} \quad (3.20)
\]

The Lagrangian is defined as:

\[
L = H + \lambda_0 g_0 + \lambda_1 g_1 + \lambda_2 g_2 \quad (3.21)
\]

In order to minimize the norm \( H \), the following constraint is applied:

\[
\frac{dL}{dT_i} = 0 \quad (3.22)
\]

Applying this constraint \(3.22\) the following equations are derived:

\[
\frac{dL}{dT_0} = 2T_0 - \lambda_0 - \lambda_1 + \lambda_2 = 0 \quad (3.23)
\]
\[
\frac{dL}{dT_1} = 2T_1 - \lambda_0 + \lambda_1 + \lambda_2 = 0 \quad (3.24)
\]
\[
\frac{dL}{dT_2} = 2T_2 + \lambda_0 - \lambda_1 - \lambda_2 = 0 \quad (3.25)
\]
\[
\frac{dL}{dT_3} = 2T_3 + \lambda_0 + \lambda_1 + \lambda_2 = 0 \quad (3.26)
\]

Combining the above equations \(3.23\) with \(3.26\) and \(3.24\) with \(3.25\):

\[
2T_0 + 2T_3 + 2\lambda_2 = 0 \quad (3.27)
\]
\[
2T_1 + 2T_2 + 2\lambda_2 = 0 \quad (3.28)
\]

And then performing a subtraction operation with \(3.27\) and \(3.28\):
\[2(T_0 + T_3 - T_1 - T_2) = 0\]  \hfill (3.29)

\[T_0 + T_3 - T_1 - T_2 = 0\]  \hfill (3.30)

This last equation (3.30) provides a new constraint on the system:

\[\Delta T = T_0 - T_1 - T_2 + T_3 = 0\]  \hfill (3.31)

This allows for the original matrix, Equation (3.15), to incorporate the minimized torque constraint:

\[
\begin{bmatrix}
\hat{T}_{cx} \\
\hat{T}_{cy} \\
\hat{T}_{cz} \\
\end{bmatrix} =
\begin{bmatrix}
-1 & -1 & 1 & 1 \\
-1 & 1 & -1 & 1 \\
1 & 1 & 1 & 1 \\
1 & -1 & -1 & 1 \\
0 & & & &
\end{bmatrix}
\begin{bmatrix}
T_0 \\
T_1 \\
T_2 \\
T_3 \\
\end{bmatrix}
\]  \hfill (3.32)

\[
\begin{bmatrix}
\hat{T}_{cx} \\
\hat{T}_{cy} \\
\hat{T}_{cz} \\
\end{bmatrix} = [A][T]
\]  \hfill (3.33)

Thus,

\[
[A^{-1}] = T_{c3to4} =
\begin{bmatrix}
-0.25 & -0.25 & 0.25 & 0.25 \\
-0.25 & 0.25 & 0.25 & -0.25 \\
0.25 & -0.25 & 0.25 & -0.25 \\
0.25 & 0.25 & 0.25 & 0.25 \\
\end{bmatrix}
\]  \hfill (3.34)
The matrix $A$ is also used and referred to as $T_{c4to3}$.

### 3.1.2 Reaction Wheel Modeling

A reaction wheel typically has an electrical motor that produces an adequate torque to rotate a shaft and added mass (flywheel). The equation for the moment of inertia $I$ of the flywheel is that of a cylinder, or disk, where $m$ is the mass and $r$ is the radius of the flywheel:

$$I = \frac{1}{2} m \cdot r^2$$  \hspace{1cm} (3.36)

The mass of the flywheel is calculated as:

$$m = \rho \cdot \pi \cdot r^2 \cdot h$$  \hspace{1cm} (3.37)

The inertia of the reaction wheel is the sum of the inertia of the shaft, typically listed as a motor specification, and the inertia of the flywheel.

The dynamics of the reaction wheel are illustrated in a free body diagram in Figure 3.3, which is modeled by:

$$T_w = I_w \dot{\omega}_w + b \cdot \omega_w$$  \hspace{1cm} (3.38)

where $T_w$ is the torque produced by the motor, $\omega_w$ is the angular velocity of the flywheel and shaft, $I_w$ is the combined inertia of the flywheel and shaft, and $b$ is the viscous damping.

This second order model (Equation 3.38) was used in the simulator design phase when estimating the torques required from the flywheels, as described in Chapter 4.

In the ADCS program, the controller has to calculate the wheel speed command required
to produce the command torques. Because solving a second order differential equation (3.38) is computationally more intensive, a simplified reaction wheel model (Equation 3.39) replaces it in the simulator code:

\[ T_w = I_w \cdot \dot{\omega}_w \] (3.39)

\[ \omega_w = \frac{1}{I_w} \int T_w \] (3.40)

A simple integration function can be used on the calculated command torques, then divided by the wheel inertia, to calculate the command wheel speeds.

The electric motor can be modeled as a DC motor (see Figure 3.3). The torque \( T_m \) of the motor is proportional to the current \( i \) drawn by the motor, where \( k_T \) is the torque constant typically listed as a motor specification:

\[ T_m = k_T \cdot i \] (3.41)

This becomes a useful relationship when calculating the torque produced by the wheels from the measured current, which can be compared to the simplified wheel model with the measured speed.

### 3.2 Linearized Control

The scope of this work was limited to primarily controlling the yaw motion of the
simulator. While this created some limitation in the control of the pitch and roll, stability
in the tilt axes depended on good balance of the simulator on the air bearing. The simulator-
reaction wheel system uses quaternion and angular rate feedback.

The error quaternion $q_E$ is calculated as follows:

$$q_E = q_s^{-1} q_T$$

(3.42)

$$q_E = \begin{bmatrix}
q_T4 & q_T3 & -q_T2 & q_T1 \\
-q_T3 & q_T4 & q_T3 & q_T2 \\
q_T2 & -q_T3 & q_T4 & q_T3 \\
-q_T1 & -q_T2 & -q_T3 & q_T4
\end{bmatrix} \begin{bmatrix}
-q_S1 \\
-q_S2 \\
-q_S3 \\
q_S4
\end{bmatrix}$$

(3.43)

where $q_s$ is the simulator quaternion and $q_T$ is the target quaternion.

The control law operates on the quaternion error as well as on the angular rates in the
form of a proportional-derivative (PV) controller, using direct velocity feedback. Applying
this control law, the torque required to act on each axis of the simulator is calculated by
Equations 3.44, 3.45, and 3.46

$$T_{cx} = 2K_x \cdot q_{1e} q_{4e} + K_{xd} \cdot p$$

(3.44)

$$T_{cy} = 2K_y \cdot q_{2e} q_{4e} + K_{yd} \cdot q$$

(3.45)

$$T_{cz} = 2K_z \cdot q_{3e} q_{4e} + K_{zd} \cdot r$$

(3.46)

These equations can be summarized as the following:

$$T_{cmd} = 2K_p \cdot q_e q_{4e} + K_v \cdot \omega_{meas}$$

(3.47)

where $\omega_{meas}$ is the measured angular rate 3x1 vector of the simulator.
The body axes angular rates are represented by $p$, $q$, and $r$, and are related to the Euler angles as follows:

$$p = \dot{\phi} - \dot{\psi} \sin(\theta) \quad (3.48)$$

$$q = \dot{\theta} \cos(\phi) + \dot{\psi} \cos(\theta) \sin(\phi) \quad (3.49)$$

$$r = \dot{\psi} \cos(\theta) \cos(\phi) - \dot{\theta} \sin(\phi) \quad (3.50)$$

Since the system dynamics are linearized for small angles in order to apply a linear control algorithm, the approximations for small Euler angles will be used:

$$p \approx \dot{\phi} \quad (3.51)$$

$$q \approx \dot{\theta} \quad (3.52)$$

$$r \approx \dot{\psi} \quad (3.53)$$

In order to take into account the nonlinear dynamics of the system, the nonlinear term in Equation 3.10, also referred to as the cross term, is added into the linear PV controller as a feedback linearization term. Therefore the required torques $T_{\text{cmd}}$ on the body axes of the simulator are calculated in Equation 3.54.

$$T_{\text{cmd}} = K_P \cdot q_e q_4 + K_v \cdot \omega_{\text{meas}} + \omega_{\text{meas}} \times (I_s \cdot \omega_{\text{meas}} + h^B_w) \quad (3.54)$$

where $h^B_w$ is the momentum generated by the reaction wheels.

Note that $h^B_w$ is represented in the body frame, which means that $T_{\text{cmd}}$ from Equation 3.32 must be used to rotate the 4-wheel momentum into momentum in the body axes, as shown in Equation 3.55.

$$h^B_w = I_w T_{\text{cmd}} \omega_{\text{meas, wheels}} \quad (3.55)$$

where $\omega_{\text{meas, wheels}}$ is a $4 \times 1$ vector containing the measured speeds of all the wheels.
In order to reduce the computational complexity of the control law, the crossterm can be expanded using Equation 3.56.

\[
\omega_{\text{meas}} = \begin{bmatrix}
0 & -\omega_{\text{meas},z} & \omega_{\text{meas},y} \\
\omega_{\text{meas},z} & 0 & -\omega_{\text{meas},x} \\
-\omega_{\text{meas},y} & \omega_{\text{meas},x} & 0
\end{bmatrix}
\] (3.56)

The simulator program is currently designed such that a single proportional gain \(K_P\) and velocity gain \(K_V\) each are input into the system. The code arbitrarily assigns a relationship that resembles deadbeat, using an arbitrary natural frequency value \(\omega_n\) and incorporating the simulator’s moment of inertia matrix \(I\), between the input values and the actual matrix gains used in the controller. In this work, only the inertia values on the principal axes of the simulator coordinate system are included, while the products of inertia are not considered. This choice was made because the wiring connections and balancing method added small off-axis masses were not captured in the CAD model used to estimate the simulator’s inertia. As detailed in Chapter 5, the user inputs the scalar \(K_p\) and \(K_v\) gains and the principal inertia values, which are then calculated using the following relationships 3.57, 3.58.

\[
K_P = \begin{bmatrix}
K_P \cdot 2.2 \cdot I_{x,x} \cdot \omega_n^2 & 0 & 0 & 0 \\
0 & K_P \cdot 2.2 \cdot I_{y,y} \cdot \omega_n^2 & 0 & 0 \\
0 & 0 & K_P \cdot 2.2 \cdot I_{z,z} \cdot \omega_n^2 & 0
\end{bmatrix}
\] (3.57)

\[
K_V = \begin{bmatrix}
K_v \cdot 1.9 \cdot I_{x,x} \cdot \omega_n^2 & 0 & 0 \\
0 & K_v \cdot 1.9 \cdot I_{y,y} \cdot \omega_n^2 & 0 \\
0 & 0 & K_v \cdot 1.9 \cdot I_{z,z} \cdot \omega_n^2
\end{bmatrix}
\] (3.58)

Note that because a quaternion is multiplied with the \(K_p\) matrix in Equation 3.54, a fourth column of zeros is appended to \(K_p\) which applies a zero gain value to the fourth and scalar component of the quaternion and also maintains the necessary 1x3 matrix shape.

Because quaternion feedback is used, the pitch and roll are also controlled, but experimentally tuning the gains for these modes was not performed in this work. The proportional
and velocity gain matrices are calculated from one input each, instead of a vector. The input gain can be considered a scaling factor, since the gain matrices basically multiply the associated component of the simulator Inertia matrix by the input gain. The two input gains were only tuned for good control of yaw.

### 3.3 Quaternion Trajectory Generation

Trajectory generation defines the path that the simulator will travel to move from the initial position to the reference position. In contrast with a step response, trajectory generation provides a smoother travel path and typically results in better tracking. The approach taken in this work is trapezoidal trajectory generation, illustrated by Figure 3.4. Trapezoidal trajectory generation is designed to control the velocity of the simulator by dividing the motion into several periods: first accelerating at a constant rate, then holding a constant speed, and finally decelerating to zero velocity by the time the final position is reached. The length of these periods is calculated from two key parameters $A_{\text{max}}$ and $V_{\text{max}}$ that are selected by the user and are related to the maximum velocities and accelerations at which the simulator can travel. In cases where the amount of time to get to maximum speed is too short, a triangular trajectory is followed as a default, where a constant acceleration is continued until the deceleration.

The trajectory generator replaces the input target quaternion with a quaternion related to the desired velocity path. In addition, the trajectory generator also outputs angular velocity and acceleration commands that are incorporated into the control law as feedforward gains. The velocity feedforward term is incorporated into the velocity gain term already included in the PV controller. In this work, the PV controller with feedback linearization and trajectory generation is the following:

$$T_{\text{cmd}} = K_p \cdot q_{E, \text{star}} + K_V \cdot (\omega_{\text{star}} - \omega_{\text{meas}}) + K_{a\text{ff, z}} \alpha_{\text{star}} + w_s \times (I_s \ast \omega_s + h_w^B)$$  \hspace{1cm} (3.59)  

Of particular note, $K_{a\text{ff}}$ is the feedforward gain that is used to drive the simulator to reach
the selected acceleration $A_{\text{max}}$, while $K_V$ is used to drive the velocity of the simulator during the trajectory generation. Because this work focused on yaw control, only the z-value of the gain $K_{\text{aff}}$ was chosen as non-zero.

### 3.3.1 Trajectory Setup

Refer to Figure 3.4 for this section. The time parameters $t_0$, $t_1$, $t_2$, and $t_3$ for the trajectory generation, as well as the angle and axis of rotation of the maneuver, are calculated every time a new position command is sent to the simulator. This is represented by the “setup” of the trajectory generator, followed by the generation of the new quaternion $q_{\text{star}}$, velocity $\omega_{\text{star}}$ and acceleration $\alpha_{\text{star}}$ commands in the trajectory generation.

The first step is that the quaternion error is calculated based on the measured simulator position, the initial quaternion, and the commanded quaternion. The total rotation angle angle required by the maneuver from the initial to commanded quaternion is calculated from the scalar component of the error quaternion:

$$\text{angle} = 2 \arccos (q_{3e})$$  \hspace{2cm} (3.60)

The axis of rotation axis of the maneuver is calculated from the calculated angle of rotation and the error quaternion:

$$\text{axis} = \begin{bmatrix} q_{0e} \\ q_{1e} \\ q_{2e} \end{bmatrix} \frac{1}{\sin \left( \frac{\text{angle}}{2} \right)}$$  \hspace{2cm} (3.61)

The initial time $t_0$ is set as the time when the position command changes. Likewise, $q_0$ is set as the measured quaternion at this time, and is passed onto the trajectory generation. The time at the other period $t_1$, $t_2$, and $t_3$ is determined by assuming that the simulator will travel at the $V_{\text{max}}$ and $A_{\text{max}}$ chosen by the user, and the logic for determining and as given in Algorithm 3.1.
Fig. 3.4: Trapezoidal Trajectory Generation Profile
Algorithm 3.1 Time Period Determination

**Input:** $t_0$, angle, $V_{\text{max}}$, $A_{\text{max}}$

**Output:** $t_1$, $t_2$, $t_3$

1. if angle $> \frac{V_{\text{max}}^2}{A_{\text{max}}}$
2. $t_1 = t_0 + \frac{V_{\text{max}}}{A_{\text{max}}}$
3. $t_2 = t_1 + \text{angle} \cdot \frac{V_{\text{max}}}{A_{\text{max}}}$
4. $t_3 = t_2 + \frac{V_{\text{max}}}{A_{\text{max}}}$
5. else
6. $t_1 = t_0 + \sqrt{\frac{\text{angle}}{A_{\text{max}}}}$
7. $t_2 = t_1$
8. $t_3 = t_2 + \sqrt{\frac{\text{angle}}{A_{\text{max}}}}$
9. end

### 3.3.2 Trajectory Generation

First, the current time $T_{\text{IMU}}$ is compared to the period times from the trajectory setup to decide which period of the trajectory generation the simulator should be executing. Then based on the maximum angular velocity $V_{\text{max}}$ and maximum angular acceleration $A_{\text{max}}$ values chosen, the new angular position, velocity, and acceleration of the simulator are predicted using basic kinematic equations, which are summarized in Algorithm 3.2. These values are then used to calculate the new $q_{\text{star}}$, velocity $\omega_{\text{star}}$ and acceleration $\alpha_{\text{star}}$ commands, as shown in Equations 3.62, 3.63, 3.64, 3.65 and 3.66.
Algorithm 3.2 Kinematics Calculation

**Input:** $T_{IMU}$, $t$, $q_0$, angle, axis

**Output:** $x_0$, $v_0$, $a_0$

1. if $T_{IMU} < t_0$
2. $d = 0$
3. $ddot = 0$
4. $ddot2 = 0$
5. elseif $T_{IMU} < t_1$
6. $dt = T_{IMU} - t_0$
7. $x_0 = 0$
8. $v_0 = 0$
9. $a_0 = A_{max}$
10. elseif $T_{IMU} < t_2$
11. $dt = T_{IMU} - t_1$
12. $x_0 = A_{max} \cdot (t_1 - t_0)^2/2$
13. $v_0 = V_{max}$
14. $a_0 = 0$
15. elseif $T_{IMU} < t_3$
16. $dt = T_{IMU} - t_2$
17. $x_0 = A_{max} \cdot (t_1 - t_0)^2/2$
18. $v_0 = A_{max} \cdot (t_1 - t_0)^2/2 + V_{max} \cdot (t_2 - t_1)$
19. $a_0 = -A_{max}$
20. else
21. $x_0 = angle$
22. $v_0 = 0$
23. $a_0 = 0$
24. end
25. $d = x_0 + v_0 \cdot dt + a_0 \cdot dt^2/2$
26. $ddot = v_0 + a_0 \cdot dt$
27. $ddot2 = a_0$
\[ q = \begin{bmatrix}
\sin(d/2) \cdot \text{axis}_1 \\
\sin(d/2) \cdot \text{axis}_2 \\
\sin(d/2) \cdot \text{axis}_3 \\
\cos(d/2)
\end{bmatrix} \quad (3.62) \]

\[ Q = \begin{bmatrix}
q_{4, \text{meas}} & q_{3, \text{meas}} & -q_{2, \text{meas}} & q_{1, \text{meas}} \\
-q_{3, \text{meas}} & q_{4, \text{meas}} & q_{1, \text{meas}} & q_{2, \text{meas}} \\
q_{2, \text{meas}} & -q_{1, \text{meas}} & q_{4, \text{meas}} & q_{3, \text{meas}} \\
-q_{1, \text{meas}} & -q_{2, \text{meas}} & -q_{3, \text{meas}} & q_{4, \text{meas}}
\end{bmatrix} \quad (3.63) \]

\[ q_{\text{star}} = Q \cdot q_0 \quad (3.64) \]

\[ \omega_{\text{star}} = \ddot{q} \cdot q_0 \quad (3.65) \]

\[ \alpha_{\text{star}} = \dddot{q} \cdot q_0 \quad (3.66) \]

After the combined time periods for the trajectory are completed, the PV controller with feedback linearization operates until the next reference command is given and a new trajectory is calculated to reach it.
Chapter 4
Hardware Design

4.1 Goal of the Design

The small satellite simulator was designed and built as a low-cost, experimental, three DOF testbed consisting of attitude sensing, a credit-card-sized on-board processor, power supplies, and reaction wheels as the control actuators. The design and construction were driven both by the requirements for this work as well as by considerations for future research using this simulator.

4.2 Air Bearing System

In order to test the attitude maneuvering control, a requirement for the simulator was to operate on a three DOF air bearing system. At Utah State University, a custom spherical air bearing was designed and machined in 1997 to test the attitude determination and control for a past spacecraft subsystem. This air bearing was renovated and significantly modified for use as the levitation apparatus for the new tabletop simulator.

4.2.1 Fundamentals

The air bearing basically consists of two brass components, a cup and hemisphere that are separated by a very thin layer of air. The hemisphere is attached on the flat surface to the simulator platform with attached sensors, electronics, and actuators. The cup is attached to a base and air bearing stand that have a pressurized air intake. The cup and hemisphere were not modified for this new generation of simulator.

4.2.2 Modifications to the Air Bearing Platform

The regulator, pressure gauge, trap, 1/2-inch pipe threads, all bolts, and hoses were
all replaced with new and appropriately rated components from the local hardware store, Industrial Tool & Supply. The compressor-to-regulator connection purchased for this new generation simulator was an industrial 100-foot hose; this length enables placing the air compressor near a power source and conveniently outside the test or demonstration area to dampen the sound. A 17-gallon air 130psi max oilless air compressor was selected in consideration of its low maintenance, which is important for passing the equipment on to future students working on the simulator. The amount of input pressure needed to slightly levitate the hemisphere and attached load to therefore allow free rotation depends on their combined weight, the surface area of the cup, and losses through the system. In this case, the operating input pressure was optimally 120psi with a minimum of 110psi for a hemisphere-simulator mass between 15-20 pounds.

Because the simulator requires 360° rotation and ±45° tilt, some redesign of the air bearing platform was required. An anodized aluminum base plate originally provided the connection between the coupled brass cup and hemisphere to the heavy aluminum base stand. This base plate was originally designed with an elbow coupler for the air intake protruding out of the side of the base plate (see Figure 4.1). Unfortunately, this base plate design prohibited free 360° rotation as well as 45° tilt. The design of the simulator with components extending below the platform further necessitated moving the air intake to the underside of the base plate and as close to the aluminum base stand weld as possible. Because the weld plate and brass components would not be modified, the intake hole needed to be channeled at an angle to the surface of the plate to mate with the 1/8-inch brass inlet hole. Careful design was dedicated to this new aluminum base plate which was then manufactured at a local machine shop. The drawing is included in Appendix H. In addition, a series of reducing couplers was selected in order to extend the larger standard ½-inch air coupler further below the plate so that the simulator components could sweep through a larger area under the plate without interference.
4.3 Attitude Sensing

For the work presented in this paper, attitude sensing is provided by the Lord Microstrain 3DM-GX3 inertial measurement unit (IMU), which contains a three-axis accelerometer, gyrometer, and magnetometer. This IMU has a dynamic accuracy of 2 degrees. The 3DM-GX3 has several output options, e.g., quaternion, Euler angles, and angular rates; for this work, Euler angles and rates were chosen as the IMU output, from which a quaternion was calculated for use in the control algorithm. In addition, the 3DM-GX3 has some filtering and bias compensation capabilities. The IMU outputs and filtering will be further discussed in the following chapter.

A requirement for the tester was to include a star camera as an alternative for attitude sensing for another research effort. This was included in the design of the simulator structure.

4.4 Attitude Control

A pyramidal configuration was chosen for the four reaction wheel design. The design of
the reaction wheels involves the selection of commercially off the shelf (COTS) motors and the design of the attached flywheels.

4.4.1 Motor Selection

A preliminary calculation was made to determine the required torque and speed of the motors. The simulator was estimated as a rough cylinder of 0.1m height and 0.1m radius with a mass of 6kg. The moments of inertia in rotation and in tilt are:

\[ I_{\text{rotation}} = \frac{1}{2}mr^2 \]  

\[ = \frac{1}{2}(6)(0.1)^2 = 0.03\, kg - m^2 \]  

\[ I_{\text{tilt}} = \frac{1}{4}mr^2 \]  

\[ = \frac{1}{4}(6)(0.1)^2 = 0.015\, kg - m^2 \]
Considering that the rotation inertia is greater than the tilt inertia, the rotational inertia 0.03 kg-m\(^2\) was used as a conservative estimation of the torque. An aggressive goal for the slew maneuver is to rotate 90 degrees (\(\pi/2\) radians) in a second, which would require an angular acceleration \(\alpha\) or \(\dot{\omega}\) of \(\pi \approx 3.14 \text{ rad/s}^2\), derived from the dynamic equation for angular position \(\theta\) below:

\[
\theta = \frac{1}{2} \alpha t^2
\]

\[
\frac{\pi}{2} = \frac{1}{2} \alpha (1)^2
\]

\[
\alpha = \pi \text{ rad/s}^2
\]

Recalling Equation 3.39 and using \(\alpha = \pi \text{ rad/s}^2\), the amount of torque required to rotate the spacecraft at this rate is calculated below to be about 0.1 Nm:

\[
T = I \cdot \dot{\omega} = (0.03 \text{ kg } \text{m}^2) \cdot (\pi \text{ rad/s}^2) \approx 0.094 \text{ Nm} \approx 0.1 \text{ Nm}
\]

Because the reaction wheels are designed to have an inclination of 45°, the torque about the wheel axis is resolved into vertical and planar components by multiplying by a factor of \(\cos(\alpha) = \cos(45) = 1/\sqrt{2} \approx 0.7071\). Refer to Figure 3.2 and Equations 3.11, 3.12, and 3.13. In other words, in order for an inclined reaction wheel to produce 0.1 Nm about the simulator rotation axis, it would need to produce 0.1/0.707 \(\approx 1.4 \text{ Nm}\) about its own axis. Because there are actually four reaction wheels contributing torque, two at a given time to be conservative, the initial estimate of requiring 0.1 Nm of torque from each reaction wheel is still conservative.

Based on this preliminary estimate, the following DC brushless motor was selected from Maxon Motors: EC 45 flat, 70 Watt, with Hall sensor, Model #397172 (see Appendix A). This motor can provide a nominal torque of 0.128 Nm, nominal current 3.21 A and nominal speed 4860 RPM, while control efforts shorter than 4 seconds can exceed these values. This
motor includes current and speed sensing, which were important for this work. The technical specifications are included in Appendix A. Additionally, the accompanying motor controllers were the ESCON 36/3 EC, Model #414533. The input to the motor controller is a command speed and direction in the form of a pulse width modulation signal (PWM) at a frequency of 53.6 kHz. The controller includes the option for closed loop speed control, which includes a calibration step and was implemented in this work. The technical specifications for the motor controller is included in Appendix B while the full hardware manual is included on the CD accompanying this thesis as “414533_ESCON_36_3_EC_Hardware_Reference_En.”

4.4.2 Wheel Design

The reaction wheel model given in Chapter 3 was used to calculate the torque produced by the motor for a given angular acceleration and speed. Note that the motor-wheel system damping constant \( b = 4.63e - 5 \text{ kg} - \text{m}^2/\text{s} \) was used as an initial guess value. The peak velocity is chosen to be 5000 RPM (\( \approx 524 \text{ rad/s} \)) and the peak acceleration value of 5000 RPM/s (\( \approx 524 \text{ rad/s}^2 \)). The motor shaft inertia is \( I_m = 181 \text{ g-cm}^2 (1.81e - 5 \text{ kg} - \text{m}^2) \).

Applying Equation 3.38, the torque produced by the motor without the wheel for these values is about 0.0337 Nm, which is not sufficient, and indicates that a flywheel is necessary to increase the inertia and torque of the reaction wheel.

Since the motor shaft is relatively short, a flywheel was designed with outer rim extending above and below the radial material, and with a “nub” on the motor side to support a shaft-supported press fit of the flywheel onto the motor shaft. Three set screws were designed to go through the nub to make contact with the motor shaft for extra support. Figure 4.4 shows the flywheel design with labeled dimensions. The values of these dimensions were varied until a desirable low mass flywheel that provided adequate torque was calculated using Equations 4.4 and 4.5. Table 4.1 summarizes the dimensions chosen for the flywheel design, while Table 6.5 summarizes the chosen operating parameters and calculated mass and inertia values.

The mass of the flywheel is calculated by Equation 4.4.
Fig. 4.4: Flywheel Model

\[ m = \rho \pi \left[ (r_1^2 - r_2^2)h_1 + (r_2^2 - r_3^2)h_2 + (r_3^2 - r_4^2)(2h_2) \right] \]  
\hspace{1em} (4.4)

The total moment of inertia of the flywheel is calculated by Equation \ref{eq:4.5}.

\[ I_f = \frac{1}{2} \pi \rho \left[ (r_1^4 - r_2^4)h_1 + (r_2^4 - r_3^4)h_2 + (r_3^4 - r_4^4)(2h_2) \right] + I_m \]  
\hspace{1em} (4.5)

where \( I_m \) is the moment of inertia of the motor shaft, given in the motor specifications.

Choosing peak velocity and acceleration of 5000 RPM \( (524 \text{ rad/s}) \) and 5000 RPM/s \( (524 \text{ rad/s}) \), respectively, and a damping coefficient \( b = 4.63e - 5 \text{ kg} - \text{m}^2/\text{s} \), and applying Equations \ref{eq:4.4} and \ref{eq:4.5}, the inertia of the flywheel is calculated as \( 1.3452e - 4 \text{ kg} - \text{m}^2 \), plus the motor shaft inertia is \( 1.533e - 4 \text{ kg} - \text{m}^2 \). The mass is \( 0.3356 \text{ lbm} \) \( (\approx 0.1522 \text{ kg}) \). The torque produced by the reaction wheel is \( 0.1045 \text{ Nm} \).

In comparison, the CAD model estimated a mass of \( 0.336 \text{ lbm} \) \( (\approx 0.153 \text{ kg}) \), a flywheel less motor shaft inertia of \( 0.462 \text{ lbm-in}^2 \) \( (1.3519e - 4 \text{ kg} - \text{m}^2) \), plus the motor shaft totalling \( 1.5329e - 4 \text{ kg} - \text{m}^2 \). The moments of inertia match between the hand calculation and CAD. According to the reaction wheel torque model, this moment of inertia would result in a torque of \( 0.1045 \text{ Nm} \). The CAD model validated the moment of inertia calculation and indicated
that the resulting torque would meet the design requirement. Four of these flywheels were manufactured. The drawing of the flywheel model is included in Appendix C.

4.5 On-board Mini Processor

Two Beagle Bone (BB) microcomputers were selected to be included on board the simulator. The BB is a low-cost Linux-based credit-card sized computer that connects that can run Ubuntu. On this simulator, one BB is totally dedicated to a different students’ future work with processing star camera images as an alternative method of attitude determination. In this work, only the second BB was used in the ADCS, but both are included in the physical design of the simulator. The BB connects to all the sensors and actuators and serves as the on-board computational platform for attitude determination and control, data handling, and development.

The BB is driven by a 720 MHz ARM-CPU and has 256MB of RAM and a 4GB microSD card, as well as USB and Ethernet ports. The BB provides a large set of low level interfaces to which the simulator components are connected. The BB communicates with the IMU via an RS232-serial connection, which required using a TTL voltage converter. The BB collects receives the 3DM-GX3 IMU and motor sensors measurements, provides the computation for attitude determination and control, sends the control signal as PWMs to the motor controllers, and saves a test data file on-board for later download. A summary of specifications for the BB is included in Appendix D.

Of note, as of late 2013 the Beagle Bone Black (BBB) is the newest version after the BB and provides more RAM and different power management component. A BBB was

<table>
<thead>
<tr>
<th>Dimension</th>
<th>[inches]</th>
<th>[mm]</th>
<th>Dimension</th>
<th>[inches]</th>
<th>[mm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>3.0</td>
<td>76.20</td>
<td>$r_1$</td>
<td>1.5</td>
<td>38.10</td>
</tr>
<tr>
<td>$d_2$</td>
<td>2.25</td>
<td>57.15</td>
<td>$r_2$</td>
<td>1.125</td>
<td>28.575</td>
</tr>
<tr>
<td>$d_3$</td>
<td>0.787</td>
<td>20.00</td>
<td>$r_3$</td>
<td>0.3935</td>
<td>10</td>
</tr>
<tr>
<td>$d_4$</td>
<td>0.157</td>
<td>4</td>
<td>$r_4$</td>
<td>0.0785</td>
<td>2</td>
</tr>
<tr>
<td>$h_1$</td>
<td>0.75</td>
<td>6.35</td>
<td>$h_1$</td>
<td>0.75</td>
<td>6.35</td>
</tr>
<tr>
<td>$h_2$</td>
<td>0.25</td>
<td>2.12</td>
<td>$h_2$</td>
<td>0.25</td>
<td>2.12</td>
</tr>
</tbody>
</table>
Table 4.2: Flywheel Inertia

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material Density $\rho$</td>
<td>2712</td>
<td>kg/m$^3$</td>
</tr>
<tr>
<td>Peak Acceleration $\omega$</td>
<td>524</td>
<td>rad/sec</td>
</tr>
<tr>
<td>Peak Velocity $\alpha$</td>
<td>524</td>
<td>rad/sec$^2$</td>
</tr>
<tr>
<td>Damping Coefficient Estimate $b$</td>
<td>4.63e-5</td>
<td>kg·m$^2$/sec</td>
</tr>
<tr>
<td>Mass $m$</td>
<td>0.1522</td>
<td>kg</td>
</tr>
<tr>
<td>Flywheel Inertia $I_f$</td>
<td>1.352e-4</td>
<td>kg·m$^2$</td>
</tr>
<tr>
<td>Reaction Wheel Total Inertia $I_w$</td>
<td>1.533e-4</td>
<td>kg·m$^2$</td>
</tr>
<tr>
<td>Wheel Torque $T_w$</td>
<td>0.1045</td>
<td>N·m</td>
</tr>
</tbody>
</table>

Fig. 4.5: BB Processor
purchased as a backup microcomputer and is available for future implementation.

4.6 Power Supply

When on-board the simulator, power to the BB is provided by a 7.4V MaxAmps Li-Po battery which goes through a simple voltage regulator to provide the 5V max to the BB and connected sensors. Four 4000mAh 12V MaxAmps batteries (Figure 4.6) were selected to provide power to the reaction wheel motors. The motors were divided into two groups, where two batteries connected in series provided 24V to each of two motors, and the other set of batteries provided power to the motors on the other side of the simulator. The battery dimensions were customized through MaxAmps in order to accommodate the limited space for the batteries on the platform, between the rotating flywheels and not obstructing the star camera field of view. All of the batteries use Deans Ultra 2-pin connectors and can be charged separately on a MaxAmps charger that was purchased.

4.7 Custom Platform Design

In order to accommodate future research with the star camera, all electronics, processors, BB power supplies, IMU, and camera were designed to be contained in a box that can be pulled off the table platform and used independently. The platform structure for the tabletop simulator and accompanying electronics box and mounts for the electronics box, reaction wheels and motor controllers, and motor batteries were designed in Solidedge and then 3-D printed. Landon Terry, an undergraduate student, designed these printed

Fig. 4.6: MaxAmps Batteries with connectors
structures in CAD with design input and modification by me. The platform structure (see Figure 4.7) and the electronics box (see Figure 4.8) were 3-D printed out of ABS plastic in 2-3 days. The 3-D printing provides the option to custom design a new structure for future simulators very quickly, or to accommodate changes in components or organization. Mass balancing was provided by bolts with washers (as weights) placed in strategic locations on the table and was planned for in the printed structure design. The final simulator hardware is shown on the air bearing in Figure 4.9. The CAD model of the simulator with roughly modeled cables but does not include balancing masses or necessary plastic ties. The CAD model estimates the mass moment of inertia of the following as the following in Equation 4.6.

\[
I_s = \begin{bmatrix}
0.02994 & -1.04080e-5 & 9.2379e-5 \\
-1.04080e-5 & 0.0303 & 1.8486e-6 \\
9.2379e-5 & 1.8486e-6 & 0.04668
\end{bmatrix} \text{kg} \cdot \text{m}^2 \tag{4.6}
\]

Fig. 4.7: Printed Table Platform
Fig. 4.8: Printed Electronics Box

Fig. 4.9: Simulator on Air Bearing
Chapter 5

Software Development

5.1 Overview of Architecture

The electrical components and schematic for the BB were selected and organized by a previous student, Jan Somers. Somers installed Ubuntu on the BB, laid out the schematic and selected and soldered components to the BB cape breadboard (Figure 5.1), built the power regulator for supplying 5V to the BB (Figure 5.2). All of these electrical components were used on the simulator, with some troubleshooting and modifications.

Somers also wrote code that laid the foundation for being able to compile the program on the BB, set several sensor parameters, read a selected IMU output, and could send a single duty cycle command to all four motor controllers. While some of the sensor communications code and basic compiling sequence and initial file structure from Somers’ work were built upon for this work, a comprehensive program with combined multiple sensors feedback, processing, controller, multiple outputs, variable user input parameters, and data collection needed to be developed from the ground up. My vision for this program was to make it modular, so that the input, sensor measurements, controller gains or controller

Fig. 5.1: BB Cape
strategy would be independent of each other and allow the user to easily modify the controller or add or modify the sensor inputs. The comprehensive simulator program was developed by myself and Ivan Jimenez, another graduate student. Jimenez set up the periodic thread, described in Section 5.1.1 and primarily worked on reading the measurements from multiple sensors, being able to output commands to the motors, and setting up the output data collection file. The overall structure and requirements of the system, all of the controller files, including trajectory generation, were planned and developed by myself. I developed all of the control algorithms and trajectory generation, manipulating measurements; and also contributed to program inputs, reading sensor measurement. I also developed the Matlab file that processes the test data.

Development by Jimenez and myself was done simultaneously and was synced constantly to the online reservoir GitHub. Code Blocks was the editor used to develop the C++ code, which was compiled and tested on the BB. Documentation for the C++ code is included on a CD accompanying this thesis as the “workspace” folder. Specifically, the necessary source and header program files are located in “workspace/chipsat/src” and “workspace/chipsat/include” folders. The Code Blocks project file “chipsat” located in “workspace/chipsat” folder is linked to the program files and updates them during development.

Customized command sequences were created in order to download updated code, run the code on the simulator with and without a direct Ethernet connection, and upload test data files from the BB. These exist on the BB and can be customized by specific users.
Some documentation on these functions is included in Appendix G.

Finally, a set of post-processing Matlab files was developed and is located with the test data in the “workspace/chipsat/matlab” folder included on the CD. The key file that will read, process, and plot the test data is called “plotData.m”. The user modifies the line of code at the very beginning of the file that specifies which test data file is being read, and then running this file will plot all the relevant test results and checks.

5.1.1 Architecture

The ADCS program is set to run as a periodic thread for a 20ms period, or 50Hz. This number is based on the processor speed and the estimated amount of time for the BB to collect all sensor data and compute the control effort. The response time of the motors and the overall system ideally would be several times greater than the program sampling period, which allows for better attitude monitoring and control. The 20ms period could possibly be reduced through code optimization.

The modular organization of the program files specifically isolates the implementation of the dynamic control law. The attitude control computation is separated from the sensor communications, reading of the reference command and saving of test data. This modularity was required in order to make the attitude control easy to modify and to implement alternative strategies. This structure provides flexibility in controller design, and changing the control law requires only changing one file. As testing attitude control was a major objective for this research, this feature of the software is especially important.

5.1.2 Overview of Key Files

A brief mention of a few key files will help a future student know where to look first to modify the program for their work.

The “main.cpp” file sets up the constructor arguments when running the program: it expects an input text file and optionally the name of an output file (csv) which defaults to “datalog.csv” if not given an argument. The main file also calls functions defined in “controller.cpp”.
The file “controller.cpp” defines most of the functions important to a future user, including: manipulating sensor readings, initializing the IMU setting (including continuous), setting motor inputs, reading the input file and assigning variables, writing to the test file, “fixing” the IMU attitude and rate from the sensors, converting motor analog signals to amps and speeds, quaternion math, integration, filtering, calculating a quaternion from the Euler angles, etc. “Controller.cpp” also calls the “controllaw.cpp” file, which ultimately computes the control effort.

The “controllaw.cpp” file calls the functions that provide attitude determination: rotating the Euler angles and rates into the body coordinate system and returning the quaternion. In future implementations, a function could be called to do a more complex attitude determination than the one described in the section on attitude determination. The filtering function is applied to the gyro rates and motor speeds. The reference command is updated by comparing the current time to the command reference quaternion and time. The trajectory generation function is either called and sets the reference to the trajectory generated one, or is unused (the user decides). Finally, the quaternion error, feedback linearization term, and control torques and corresponding motor speeds are computed. The commands to the motor are sent. Variables and states are updated.

5.2 User Input

The program is designed to take a comprehensive user-friendly input file that first and foremost allows the user to quickly change the command position and/or the controller gains without having to alter or recompile the program. This was essential for making the simulator a functional experimental test platform. The input file also contains system constants related to system dynamics, as well as motor controller settings such as maximum input or output speeds. Appendix E contains a summary of the input parameters.

The reference command is constructed at the end of the input file. A predefined path or trajectory is listed at the end of the text file and always contains at least two lines. The first is an integer indicating the number of references in the trajectory, followed by the total number of seconds to run the program. Each of these lines consists of the number of
event (starts at 1), followed by the number of seconds after program start to change the reference, and followed by four entries representing the quaternion components, with the scalar component listed last.

5.3 Attitude Determination

5.3.1 Choosing the IMU Output

It was required for this simulator to represent the attitude as a quaternion and to be able to use direct angular rate feedback for position control. While the IMU has several options for data outputs, none of them output both angular rates from the gyro and a quaternion. However, a quaternion may be calculated from the orientation matrix calculated from the Euler angles. Using this relationship, two IMU output options could provide the information needed for attitude determination in quaternions and measured angular rates: Acceleration, Angular Rate, and Orientation Matrix, which sends 67 bits of data per packet, and the Euler Angles and Rates, which sends 31 bits of data per packet. The calculation of the quaternion from the Euler angles requires calculating the orientation matrix, which is computationally more intensive than going from the orientation matrix to the quaternion. Based on Equation 5.1 and a baud rate of 115.2k bits/second, the first option takes 4.7 ms per packet, while the second takes 2.1 ms. In comparison to the sampling measurement and controller loop frequency of 20ms, both options take a significant percentage of that periodic thread time period. It was assumed that the BB could execute the more intensive mathematical computation to go from Euler angles to a quaternion in much less time than it would take the IMU to output the greater size data packets. Therefore, several IMU output options were coded for future students, but the Euler Angles and Rates output was selected for this work. A summary of five possible IMU outputs and their corresponding time requirements is provided in Table 5.1.

Jimenez developed a program to test if the IMU data packets were being received properly, called “serialtest.py” in Python.
\[ Time = \frac{(\text{Number of Bytes} \cdot \frac{4 \text{ Bits}}{\text{Byte}} + 1 \text{ stopbit})}{\text{Baud Rate}} \] (5.1)

5.3.2 IMU to Body Coordinate System

The IMU reports the Euler angles as roll about the X axis (\(\phi\)), pitch about the Y (\(\theta\)), and yaw about Z (\(\psi\)), in that order and in units of radians.

The body coordinate system of the simulator was chosen to be X pointing forward in the direction of the camera, Z pointing up to the sky, and Y following the right-hand rule and therefore pointing to the left when looking through the camera. The IMU is mounted “upside down” underneath the other electronics in the printed box, with the ribbon leads channeling toward the back of the simulator (-X) to connect to the mounted BB on the back of the box. The IMU has its own coordinate system, and therefore the IMU data must be rotated into the body coordinate system by a 90° rotation about the Z axis. For the sake of time, a simple experiment was performed to observe the sign of the IMU outputs when rotating about the body coordinate axes. Using the right-hand rule, rotation about a body axis with thumb pointing in the positive direction was considered a positive rotation. According to this convention, sign changes and phase shifts were applied to roll, pitch, and yaw about the body axes if the IMU did not match. The functions “Controller::fixAngles” and “Controller::fixRates” apply the sign changes according to the experimental test.

As a summary, the roll and pitch rates from the IMU needed to be negated to match the body coordinate system; the Euler angle representing yaw needed to be negated; and the Euler angle representing roll needed to be negated and then shifted by positive \(\pi\) if the

<table>
<thead>
<tr>
<th>IMU Output</th>
<th>Bytes</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orientation matrix M</td>
<td>43</td>
<td>3.0</td>
</tr>
<tr>
<td>Euler angles</td>
<td>19</td>
<td>1.3</td>
</tr>
<tr>
<td>Euler angles and angular rates</td>
<td>31</td>
<td>2.1</td>
</tr>
<tr>
<td>Quaternion</td>
<td>23</td>
<td>1.6</td>
</tr>
<tr>
<td>Acceleration, angular rates, and orientation matrix M</td>
<td>67</td>
<td>4.7</td>
</tr>
</tbody>
</table>
angle was already initially positive, or negated and then shifted by negative $\pi$ if the angle was initially negative.

5.3.3 Calculating the Quaternion

The data sheet for the 3DM-GX3 listed the calculations from Euler angles to the orientation matrix to the quaternion. The algorithms listed in the Microstrain data sheet (included on the CD as “MicroStrain - 3DM-GX3-Data-Communications-Protocol.pdf”) for calculating the quaternion assume a convention where the first component of the quaternion $q_1$ is the scalar, which is opposite the convention preferred in this work. The Microstrain algorithms were used to calculate the quaternion from the “fixed” Euler angles in the body coordinate system, and then the quaternion components were rearranged so that the scalar component was in the last position $q_4$. The equations for calculating the quaternion (with the scalar component in the first position) are listed below:

$$M = \begin{bmatrix}
\cos(\psi) \cos(\theta) & \sin(\psi) \cos(\theta) & -\sin(\theta) \\
\cos(\psi) \sin(\theta) \sin(\phi) - \sin(\psi) \cos(\phi) & \sin(\psi) \sin(\theta) \cos(\phi) + \cos(\psi) \cos(\phi) & \cos(\theta) \sin(\phi) \\
\cos(\psi) \sin(\theta) \cos(\phi) + \sin(\psi) \sin(\phi) & \sin(\psi) \sin(\theta) \cos(\phi) + \cos(\psi) \sin(\phi) & \cos(\theta) \cos(\phi)
\end{bmatrix}$$

The orientation of the IMU affects how the quaternion is calculated from the orientation matrix. Four equations or “tests” are calculated using the entries of the orientation matrix, and each test indicates a slightly different calculation for the quaternion, as summarized in Algorithm 5.1. The test that yields the maximum value for a given set of Euler angles is the one that should be used to then calculate the quaternion.

Recalling the relationship $q^T = -q$, it is important to note that by convention the quaternion is represented such that the scalar component is always positive. It was discovered during initial testing of the controller that when calculating the quaternion error and implementing it in the controller equation, it is essential that the scalar component of the quaternion be required to always be positive.
Algorithm 5.1 Calculating the Quaternion from the Orientation Matrix

Input: \( M_{1,1}, M_{2,2}, M_{3,3} \)

Output: \( q \)

1. \[ \text{test} = \begin{bmatrix} test_1 \\ test_2 \\ test_3 \\ test_4 \end{bmatrix} = \begin{bmatrix} M_{1,1} + M_{2,2} + M_{3,3} \\ M_{1,1} - M_{2,2} - M_{3,3} \\ -M_{1,1} + M_{2,2} - M_{3,3} \\ -M_{1,1} - M_{2,2} + M_{3,3} \end{bmatrix} \]

2. \( \max (\text{test}) = \max \{\{test_1, test_2, test_3, test_4\}\} \)

3. if \( \max (\text{test}) = test_1 \)

4. \[ S = 2\sqrt{1 + M_{1,1} + M_{2,2} + M_{3,3}} \]

5. \[ \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} S/4 \\ (M_{3,3} - M_{3,2})/S \\ (M_{1,3} - M_{1,1})/S \\ (M_{1,2} - M_{2,1})/S \end{bmatrix} \]

6. elseif \( \max (\text{test}) = test_2 \)

7. \[ S = 2\sqrt{1 + M_{1,1} - M_{2,2} - M_{3,3}} \]

8. \[ \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} (M_{3,2} - M_{2,3})/S \\ -S/4 \\ -(M_{2,1} - M_{1,2})/S \\ -(M_{1,3} - M_{3,1})/S \end{bmatrix} \]

9. elseif \( \max (\text{test}) = test_2 \)

10. \[ S = 2\sqrt{1 - M_{1,1} + M_{2,2} - M_{3,3}} \]

11. \[ \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} (M_{1,3} - M_{3,1})/S \\ -(M_{2,1} + M_{1,2})/S \\ -S/4 \\ -(M_{3,2} - M_{2,3})/S \end{bmatrix} \]

12. else

13. \[ S = 2\sqrt{1 - M_{1,1} - M_{2,2} + M_{3,3}} \]

14. \[ \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} (M_{2,1} - M_{1,2})/S \\ -(M_{2,1} + M_{1,2})/S \\ -S/4 \\ -(M_{3,2} - M_{2,3})/S \end{bmatrix} \]

15. end
5.3.4 IMU Sensors

While the IMU internally filters its sensor measurements, both the gyro and magnetometer presented issues affecting the attitude determination. Issues related to the magnetometer and gyro noise and bias will be further addressed in Chapter 6.

5.4 Motor Feedback Sensing

Each of the motor controllers outputs two analog signals that measure the current drawn by the motors, which can be related to torque by the torque constant 36.9e-3 Nm/A, as well as the motor speed. These signals are first converted into amps and rad/s, respectively. The conversion from voltage to current in amps is dictated by the settings in the ESCON motor controller, and it was found that the conversion needed to be negated:

\[ m_{\text{MotorScale.vToRads}} = (5997 \text{ RPM})/(4 \text{ V})\pi/30 \approx -157.001 \text{ (rad/s)}/\text{V} \]

\[ m_{\text{MotorScale.vToAmps}} = -9/(3.996 \text{ V}) \approx -2.23225 \text{ A}/\text{V} \]

Both of these signals were filtered with a technique that will be described in Chapter 6 and saved as test data. The filtered motor speeds were used to calculate the feedback linearization term.

5.5 Attitude Control

The “controllaw.cpp” file isolates the implementation of the dynamic control law and computes the motor control effort based on attitude and sensor feedback inputs and outputs motor commands. As noted in the prior section, quaternions must be constrained to have a positive scalar component in order for the quaternion error calculation to be correct.

The quaternion error and feedback linearization term are calculated using the quaternion, filtered gyro rates and motor speeds. The command torque on the spacecraft is calculated using Equation 3.59 and then resolved into four motor torques using Equation 3.35. The required speed commands of the motors are calculated by integrating these torques according to Equation 3.40 using a trapezoidal integration algorithm.

Before the motor command speed is converted to a duty cycle input to the motor controllers, the magnitude of the speed is limited to 5000RPM. This max motor speed can
be changed by the user in the input file variable “mSystem.motorSpeedMax” in units of RPM, which is then calculated in the code as the max speed “maxSpeedCmd” in units of rad/s.

The ESCON controller sets a duty cycle of 10% to be 0 RPM and 90% to be the maximum RPM. The calculation from the command motor speeds to duty cycles is represented by the conversion factor “speed2dc” calculated by:

$$speed2dc = \frac{(DCRANGE \cdot RAD2RPM)/mSystem.motorSpeedMax}{mSystem.motorSpeedMax}$$

where $RAD2RPM = 30/\pi$ and $DCRANGE = 80$, using the 10% to 90% speed range.

5.6 Output File

The output file lists 53 variables and measurements recorded every 20ms. The overall test file for a 2-minute experimental test was found to be roughly 1.5 MB. The write rate accommodates the 53 variables’ worth of data but this would need to be explored in future work. A list of the variables represented in each column is given in Appendix F.
Chapter 6
Electromechanical Testing

6.1 Motor Speed Controller

The ESCON motor selected for this work comes with its own motor controller. The ESCON motor controller is accompanied by software with three controller modes: closed loop speed control, open loop speed control, and current control. Closed loop speed control was used on the simulator. The software includes an autotuning process which requires connecting the motor controller (with connected motor) to a computer running the ESCON software, selecting the controller mode, and allowing the software to send command sets to the motors for calibration. The controller gains are tuned during this process for a basic proportional controller. The major benefit of the autotuning software is that each individual motor response can be characterized with the attached load, the flywheel, already attached to it. The following gain values calculated by the ESCON software for the experimental tests reported in Chapter 7 are reported in the following Tables 6.1, 6.2, 6.3, 6.4.

6.2 Reaction Wheel Characterization

In order to determine a model of the reaction wheel response, each wheel was run through a set of step commands throughout the speed range ($\pm\ 523.1\ rad/s$). The step

<table>
<thead>
<tr>
<th>Tuning Application</th>
<th>Tuning Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Proportional gain</td>
<td>1137</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>54.7 ms</td>
</tr>
<tr>
<td></td>
<td>IxR Factor</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>IxR Time Constant</td>
<td>40 ms</td>
</tr>
<tr>
<td>Current</td>
<td>Proportional Gain</td>
<td>179</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>512 $\mu$sec</td>
</tr>
</tbody>
</table>
### Table 6.2: Wheel 1 Autotuning Gains

<table>
<thead>
<tr>
<th>Tuning Application</th>
<th>Tuning Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Proportional gain</td>
<td>1112</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>52.2 ms</td>
</tr>
<tr>
<td></td>
<td>IxR Factor</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>IxR Time Constant</td>
<td>39 ms</td>
</tr>
<tr>
<td>Current</td>
<td>Proportional Gain</td>
<td>170</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>521 (\mu)sec</td>
</tr>
</tbody>
</table>

### Table 6.3: Wheel 2 Autotuning Gains

<table>
<thead>
<tr>
<th>Tuning Application</th>
<th>Tuning Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Proportional gain</td>
<td>1122</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>53.5 ms</td>
</tr>
<tr>
<td></td>
<td>IxR Factor</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>IxR Time Constant</td>
<td>40 ms</td>
</tr>
<tr>
<td>Current</td>
<td>Proportional Gain</td>
<td>180</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>493 (\mu)sec</td>
</tr>
</tbody>
</table>

### Table 6.4: Wheel 3 Autotuning Gains

<table>
<thead>
<tr>
<th>Tuning Application</th>
<th>Tuning Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Proportional gain</td>
<td>1161</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>56.5 ms</td>
</tr>
<tr>
<td></td>
<td>IxR Factor</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>IxR Time Constant</td>
<td>42 ms</td>
</tr>
<tr>
<td>Current</td>
<td>Proportional Gain</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>Integral Time Constant</td>
<td>513 (\mu)sec</td>
</tr>
</tbody>
</table>
responses consistently show first order responses, as shown in Figure 6.1, the motor response to a step response of 45% duty cycle command \((\approx 2812.5 \text{ RPM or } \approx 294.5243 \text{ rad/s})\). Therefore, the first order system in Equation 6.1 was used as the reaction wheel model.

\[
\omega_{\text{meas}} = \frac{K_w}{(\tau_w s + 1)} \omega_{\text{cmd}}
\]  

(6.1)

where \(\tau_w\) is the time constant and \(K_w\) is the proportional gain constant.

The time constant \(\tau_w\) is the time it takes the wheel to reach 63% of its steady state speed. The proportional gain \(K_w\) is the ratio of the output speed \(\omega_{\text{meas}}\) to the input command speed \(\omega_{\text{cmd}}\), or the slope of the speed output vs. input speed curve. These data were collected and averaged for all of the step responses for each the wheels. It should be noted that the zero speed point was included in the plots but excluded from the average in order to not skew the results for non-zero speeds. The speed curves are shown in Figure 6.2.

The averaged constant parameters \(\tau_w\) and \(K_w\) and corresponding first order system model for each of the wheels is summarized in Table 6.5. The full data set used to determine the constants are included in Appendix I.

### 6.3 Magnetometer

Microstrain provides pre-test calibration software called Magnetometer Iron Calibration that is intended to be run when the IMU is fixed to the simulator. Magnetic field distortion, termed Soft Iron distortion, and magnetic field bias, termed Hard Iron offset, cause ellipsoidal distortion and offset of the magnetometer readings. The iron calibration at least partially

<table>
<thead>
<tr>
<th>Wheel</th>
<th>(\tau_w)</th>
<th>(K_w)</th>
<th>System Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>121.4 ms</td>
<td>0.6939</td>
<td>(\omega_{\text{meas}} = \frac{0.6939}{(0.1214s+1)} \omega_{\text{cmd}})</td>
</tr>
<tr>
<td>1</td>
<td>117.0 ms</td>
<td>0.6938</td>
<td>(\omega_{\text{meas}} = \frac{0.6938}{(0.1170s+1)} \omega_{\text{cmd}})</td>
</tr>
<tr>
<td>2</td>
<td>150.6 ms</td>
<td>0.6912</td>
<td>(\omega_{\text{meas}} = \frac{0.6912}{(0.1506s+1)} \omega_{\text{cmd}})</td>
</tr>
<tr>
<td>3</td>
<td>117.4 ms</td>
<td>0.6926</td>
<td>(\omega_{\text{meas}} = \frac{0.6926}{(0.1174s+1)} \omega_{\text{cmd}})</td>
</tr>
</tbody>
</table>
Fig. 6.1: Reaction Wheel Speed Response to 45% Duty Cycle
Fig. 6.2: Wheel Speed Curves
compensates for distortion and bias.

In this case, the placement of the IMU essentially surrounds it with all the other electronics without any shielding. While troubleshooting the IMU and looking at real-time plots of the magnetometer reading, it became apparent that the magnetometer could not sense the true magnetic north. The IMU is default programmed to use the magnetometer to periodically “fix” its north sensing as well as to “fix” drifting position measurements caused by gyro bias. Because the magnetic north sensing was being completely skewed by magnetic disturbances, even causing it to measure north as any direction throughout the 360° range, a solution needed to be implemented to mitigate the wild sensor readings. It was found that the “Magnetic North Compensation” setting could be turned off, which was implemented in the communications code. While the magnetometer reading will still slowly drift over time, the “true north” is measured as the initial north position when the IMU is powered on. This requires briefly disconnecting the IMU from power before every test. This mitigated the unpredictable magnetic and therefore position readings.

6.4 Gyro

After the magnetometer issue was mediated, it was observed that the gyro measurements have a bias. This was seen by running a stationary test and plotting the Euler angles over time as shown in Figure 6.3. The problem was most significant in yaw, with an average drift rate of 10°/sec.

In order to address the gyro bias, a program was developed by Jimenez to perform a calibration step that is built into the IMU, as well as to save the values in the IMU so that the calibration only has to happen once on a test day, rather than every test. The difference in gyro drift is significant, as shown in Figure 6.4, that the rate is decreased to an order of magnitude less than without the compensation.

For the results presented in the Chapter 7, the bias correction values set after running this program are:
6.5 Filtering Technique

6.5.1 Theory

A first order low-pass filter (LPF) was used to reduce the noise in the motor speed and current measurements as well as in the gyro readings. The general form of a first order LPF $H(s)$ with cutoff frequency $w_c$ is Equation 6.2:

$$H(s) = \frac{1}{w_c s + 1}$$  \hspace{1cm} (6.2)
Rearranged, the first order low pass filter becomes Equation 6.3:

$$H(s) = \frac{w_c}{s + w_c}$$  \hspace{1cm} (6.3)

The continuous filter is converted into discrete time of the form:

$$H(z) = \frac{az + a}{z - b}$$ \hspace{1cm} (6.4)

The discretization $H_d(z)$ of the continuous transfer function $H(s)$ is done using the bilinear transform, or Tustin’s method, where $T_s$ is the sampling frequency:

$$H_d(z) = H(s')$$ \hspace{1cm} (6.5)

where

$$s' = \frac{2}{T_s} \frac{z - 1}{z + 1}$$

Letting $tf = 2/T_s$, Equation 6.5 becomes:
\[ H_d(z) = H(s') = \frac{tf(z-1)}{z+1} \quad (6.6) \]

Applying the bilinear transform 6.6, the discretized first order filter becomes Equation 6.7:

\[ H_d(z) = H(s') = \frac{w_c}{(\frac{tf(z-1)}{z+1} + w_c)} \]

\[ = \frac{w_c(z+1)}{tf(z-1) + w_c(z+1)} \]

\[ = \frac{w_c(z+1)}{(tf + w_c)z + (w_c - tf)} \]

\[ = \frac{w_c(z+1)}{z + \frac{w_c - tf}{tf + w_c}} \]

\[ H_d(z) = \frac{\frac{w_c}{tf + w_c}(z+1)}{z - \frac{tf - w_c}{tf + w_c}} \quad (6.7) \]

Recalling the form 6.4, the coefficients are now the following 6.8:

\[ a = \frac{w_c}{tf + w_c}, \quad b = \frac{tf - w_c}{tf + w_c} \quad (6.8) \]

Expanding the form of a discrete transfer function with input \( U \) and output \( Y \),

\[ H_d(z) = \frac{Y}{U} \quad (6.9) \]

\[ \frac{Y}{U} = \frac{az + a}{z - b} \quad (6.10) \]

\[ Y(z - b) = U(az + a) \]

and applying the filter to discrete measurements \( u(k) \) and where it is assigned that \( z = k \):
\[ Y[k] - bY[k - 1] = aU[k] + aU[k - 1] \]

and isolating the filtered output measurement \( Y[k] \), the discrete filter applied to the discrete measurements becomes Equation 6.11:

\[ Y[k] = bY[k - 1] + aU[k] + aU[k - 1] \]

\[ Y[k] = bY[k - 1] + a(U[k] + U[k - 1]) \quad (6.11) \]

In the simulator code, the filtering function implementing Equation 6.11 for several measured variables appears as:

\[ \text{out} = b(\text{old}_{\text{out}}) + a(\text{in} + \text{old}_{\text{in}}) \quad (6.12) \]

### 6.5.2 Implementation

The filter requires knowing the sampling time and cutoff frequencies of each measurement. The sampling time \( T_s \) of the system is 20ms. The cutoff frequencies for each of the measurements were determined by plotting the frequency response in Matlab using the Fast Fourier Transform (FFT) of collected data. Several preliminary test data sets were analyzed to determine the appropriate cutoff frequencies, and it was determined that a cutoff frequency \( w_c = 1 \text{Hz} \) worked well for all three gyro readings as well as for the motor current and speed measurements. Applied to Equation 6.7, the filter becomes Equation 6.13:

\[ \frac{Y}{U} = \frac{0.05912z + 0.05912}{z - 0.8818} \quad (6.13) \]

The frequency responses for the gyro measurements are presented in Figures 6.5, 6.6.
The results of the filter on the gyro measurements is demonstrated in Figures 6.8, 6.6.
The frequency response and filtered results for the motor speeds are presented in Figures 6.11, 6.12, and the same plots for the current measurements are presented in Figures 6.13, 6.14.

6.6 Power Consumption

It was found that the 11.1V Li-Po battery pack pairs connected in series per each two reaction wheels typically lasted several hours of testing before needing to be recharged. The battery should have 12.6V when fully charged, and is allowed to drop to 11.9V, before needing a recharge. Each of the four batteries typically took between 30 to 45 minutes to recharge. In contrast, it was found that the 7.4V Li-Po battery for the BB power supply only lasted about 30 minutes without problems, from 8.3V to 7.8 to 7.9V, and takes 45+ minutes to recharge. This is why a second 7.4V battery was purchased so that one would always be charging during testing.

The IMU is programmed to continuously send data packets, so if the program fails to receive a data packet for various reasons, a message output to the screen will report it but
the program will continue running. If the program quits at the start when checking the IMU and fails to run, a message reports that the IMU could not be initialized. It was found that two scenarios cause this error: one, if the IMU has been recently disconnected from power, for example to reset the magnetometer, and not enough wait time is provided; and two, when the BB battery begins to drain below 7.7-7.8V (8.3V nominal), which affects the amount of power provided to the IMU. For the first scenario, waiting 5-10 seconds when powered off and again after powering back on solved the problem most of the time. In the second case, using a miniature voltage meter frequently and purchasing a second BB battery to quickly switch out the draining supply after 7.7-7.8V every 30-45 minutes.
Fig. 6.9: Filtered Gyro Pitch Rate

Fig. 6.10: Filtered Gyro Roll Rate
Fig. 6.11: FFT on Motor Speed Measurements
Fig. 6.12: Filtered Motor Speed Measurements
Fig. 6.13: FFT on Motor Current Measurements
Fig. 6.14: Filtered Motor Current Measurements
Chapter 7

Experimental Testing

7.1 Preliminary Tests

The first controller implemented was a simple proportional controller. It was found that for all values of proportional gains tested, there was a steady state error that caused the simulator to respond and oscillate about the setpoint by about $6^\circ$, which eventually grew unstable. While the P controller is not sufficient for attitude control, it was used to troubleshoot other issues with sensor readings and corrections, etc.

7.2 Test Details

The results from a test with a PV linear controller with feedback linearization and trajectory generation are analyzed in the following plots.

The input trajectory was to move to positions $0^\circ$, $-30^\circ$, $-120^\circ$, and $0^\circ$ in yaw.

The Amax and Vmax for the trajectory generator were both 0.5, the feedforward Kff gain was $\begin{bmatrix} 0.0 & 0.0 & 0.1 \end{bmatrix}$.

For $w_n = 0.2$, input $KP = 0.1$, input $KV = 0.01$, the PV gains in the control law were:

$$K_p = \begin{bmatrix} 2.635e-2 & 0 & 0 & 0 \\ 0 & 2.669e-2 & 0 & 0 \\ 0 & 0 & 4.108e-2 & 0 \end{bmatrix}$$

$$K_v = \begin{bmatrix} 2.276e-2 & 0 & 0 & 0 \\ 0 & 2.305e-3 & 0 & 0 \\ 0 & 0 & 3.548e-3 & 0 \end{bmatrix}$$
7.3 Plots

First the measured positions in yaw, pitch, and roll are plotted separately with corresponding input reference and trajectory generated reference (Figures 7.1, 7.2, 7.3).

As a comparison of the rotation about each axes, Figure 7.4 plots rotation about all axes together:

The position error is plotted for all the axes in Figure 7.5 followed by the quaternion error converted into Euler angle errors in Figure 7.6; they should be exactly the same but opposite in sign.

The $K_v$ and $K_{aff}$ gains reflect driving the control effort to push the table to reach higher velocity and acceleration in order to perform the maneuver as quickly as possible. Figures 7.7 and 7.8 show that the table velocity and acceleration in yaw actually matched the trajectory generated command very well, which was expected because it was implemented in the control. The feedforward gain was attempted at higher values, but it was found that the table seemed to still have a slow response but even greater overshoot.

It was expected that the table pitch and roll speeds and accelerations would not necessarily match the trajectory generated command since the corresponding control effort was not implemented for pitch or roll (Figures 7.9, 7.10, 7.11, 7.12).

Finally, the calculated control effort on the table was compared to torque calculated by using Equation 3.39 with filtered, differentiated gyro measurements. The following plots show that the “measured” torque levels for yaw tend are lower than the command control effort, while the measured values for pitch and roll reflect the controller responding to rocking motion that begins to grow during the test (Figures 7.13, 7.14, 7.15).

The feedback linearization term, referred to as the “crossterm,” was added into the controller to account for nonlinear dynamics. The amount of control effort added by the feedback linearization for each axis is plotted in Figure 7.16. It is seen that this term is an order of magnitude smaller than the overall torque, so its inclusion is not necessarily significant.

After the command control torque on the table is calculated, the torque required by the
Fig. 7.1: Table Yaw Position

Fig. 7.2: Table Pitch Position
Fig. 7.3: Table Roll Position

Fig. 7.4: Table Position Combined
Fig. 7.5: Position Error

Fig. 7.6: Converted Quaternion Error
Fig. 7.7: Table Speed Yaw

Fig. 7.8: Table Acceleration Yaw
Fig. 7.9: Table Speed Pitch

Fig. 7.10: Table Acceleration Pitch
Fig. 7.11: Table Speed Roll

![Table Speed Roll Graph](image1)

Fig. 7.12: Table Acceleration Roll

![Table Acceleration Roll Graph](image2)
Fig. 7.13: Yaw Torque Command vs. Measured

Fig. 7.14: Pitch Torque Command vs. Measured
Fig. 7.15: Roll Torque Command vs. Measured

Fig. 7.16: Crossterm Values
wheels is calculated, shown in Figure 7.17 followed by the command wheel speed necessary to achieve that torque, Figure 7.18. It is seen that there is a significant amount of jitter in the wheels, especially at commanded zero speeds; this contributes to tilt disturbance that grows over time. In addition, the actual duty cycle commands sent to the motor are plotted in Figure 7.19.

7.4 Summary

The hardware and software were successfully designed, built, and tested. Sensor issues were addressed, and at least partially mitigated. It was found through experimental testing that a proportional controller was not sufficient or stable, but that the PV controller with feedback linearization and trajectory generation provided fairly smooth rotation in yaw, as well as very quick stops once the reference angle was reached. For smaller angles, the controller appeared to work well, but for very large jumps the overshoot tended to be very large and the simulator was slow to respond outside of the trajectory generation, resulting in oscillatory behavior. It was found that the table itself has a very slow response compared to what was expected.

The trajectory table speed and acceleration were approximated, and the feedforward gain was most effective in driving the simulator faster. Without trajectory generation, the controller is not as effective or stable.

The speed output very closely matched the command wheel speeds. Jitter in the wheels was problematic and partly caused by the conversion of wheel speed to step-wise duty cycle commands. At very low speeds or at zero, the motor jittered and caused rocking in the simulator. Notwithstanding the tilt disturbances, the oscillations were small compared to the yaw motion and did not interfere with smooth yaw rotation.

It was found that good balance is essential for the simulator performance. Because of contamination with age, the air bearing itself does not output an even amount of air pressure from each of the outlet holes in the brass cup. This situation resulted in significant disturbances in the system, which especially drove the tilt rotation. When the balance is not “dead on,” the disturbances become even more of a problem and the simulator may be driven
Fig. 7.17: Command vs. Measured Wheel Speeds

Fig. 7.18: Duty Cycle Commands to Wheels
Fig. 7.19: Command vs. Measured Wheel Torques
unstable. The feedback linearization terms tend to exacerbate this problem because they
grow with disturbances and oscillations. Another contributing factor was that the center of
mass of the hardware was not the same as calculated by the CAD.
Chapter 8
Conclusions and Future Work

While several issues remain with the performance of the simulator, the hardware and software have been designed such that improvements can be easily implemented. The first major improvement needed is the attitude determination. While considerable progress was made in tackling sensor bias and drift from the high-end IMU, these setbacks majorly limit the performance of the ADCS. A comprehensive Kalman filter for attitude determination is recommended for future work.

Future work would include a better estimate of the inertia of the simulator, which would include products of inertia instead of assuming all axis inertia was zero. Also, even better would be a direct measurement of the simulator’s inertia. The CAD model estimates of inertia, but the construction of the axes in the CAD assembly make it difficult to trust. In addition, several components like wiring, which make a significant difference in thr balance of the simulator, could not be included in the CAD model. Furthermore, the process of charging of the batteries required removing them from the simulator, which changed the exact balance of the simulator and so fine balancing with washers and tape was necessary every time a new set of tests was initialized. A more precise and/or consistent method of balancing could improve the simulator inertia estimation and true balance of the simulator.

Future work would necessarily include control of the pitch and roll. By adding inputs to the input file and changing the way the gain matrices are calculated, a vector instead of a scalar could be specified for the input gain values so that the yaw, pitch, and roll could be tuned more easily and independent of each other. The feedfoward gain $K_{aff}$ is already a $3 \times 1$ vector and therefore would need all non-zero components.

The feedback linearization terms grow in tilt motion and drive the simulator toward instability. Better estimation of the simulator inertia as well as including better control
effort on roll and pitch would mitigate this cause of instability.

Future work could include integral control in order to eliminate or reduce the steady state error. The “integrateQ” function in the controller has already been developed and tested in case a future user wants to include it in the control law.

Future work may include using the star camera for attitude determination, which might need a new electronics box to allow the camera to point skyward. Also, an inexpensive might be compared to a higher-end IMU. A Kalman filter must be implemented on this ADCS. A Bluetooth component may be soldered correctly to the BB for possible wireless transfer of position commands.

One possibility for the simulator is to add wireless control capability so that the user can change the reference input in real-time and remotely.

In conclusion, the simulator design and programming, and the initial controller and corresponding response are acceptable. Necessary future work requires better attitude determination, improved physical inertia and time response estimation, and finally, a refined controller to achieve more precise and stable simulator response.
References


Appendices
Appendix A

Motor Datasheet
**EC 45 flat** Ø42.8 mm, brushless, 70 Watt

### Motor Data (provisional)

<table>
<thead>
<tr>
<th>Values at nominal voltage</th>
<th>24</th>
<th>30</th>
<th>38</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal voltage V</td>
<td>234</td>
<td>194</td>
<td>166</td>
<td>148</td>
</tr>
<tr>
<td>No load speed rpm 1</td>
<td>6159</td>
<td>6230</td>
<td>6330</td>
<td>6448</td>
</tr>
<tr>
<td>No load current mA 1</td>
<td>128</td>
<td>112</td>
<td>108</td>
<td>134</td>
</tr>
<tr>
<td>Nominal speed 1 rpm</td>
<td>4860</td>
<td>4993</td>
<td>5080</td>
<td>5040</td>
</tr>
<tr>
<td>Nominal torque (max. continuous torque) mNm 1</td>
<td>3.21</td>
<td>3.36</td>
<td>1.95</td>
<td>0.935</td>
</tr>
<tr>
<td>Nominal current (max. continuous current) A 1</td>
<td>1150</td>
<td>1048</td>
<td>1050</td>
<td>879</td>
</tr>
<tr>
<td>Stall torque mNm 1</td>
<td>39.5</td>
<td>25.8</td>
<td>20.7</td>
<td>6.97</td>
</tr>
<tr>
<td>Starting current A 1</td>
<td>4860</td>
<td>4993</td>
<td>5080</td>
<td>2540</td>
</tr>
<tr>
<td>Max. efficiency % 1</td>
<td>80</td>
<td>84</td>
<td>83</td>
<td>84</td>
</tr>
</tbody>
</table>

### Specifications

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>6.080</th>
<th>1.16</th>
<th>1.74</th>
<th>6.89</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal resistance phase to phase Ω</td>
<td>0.680</td>
<td>1.16</td>
<td>1.74</td>
<td>6.89</td>
</tr>
<tr>
<td>Terminal inductance phase to phase mH</td>
<td>0.463</td>
<td>0.691</td>
<td>0.969</td>
<td>5.85</td>
</tr>
<tr>
<td>Torque constant mNm / A 2</td>
<td>36.9</td>
<td>45.1</td>
<td>53.3</td>
<td>131</td>
</tr>
<tr>
<td>Speed constant rpm / V 2</td>
<td>259</td>
<td>212</td>
<td>179</td>
<td>72.7</td>
</tr>
<tr>
<td>Speed / torque gradient rpm / mNm 2</td>
<td>4.26</td>
<td>5.44</td>
<td>5.85</td>
<td>3.82</td>
</tr>
<tr>
<td>Mechanical time constant ms 2</td>
<td>8.67</td>
<td>10.3</td>
<td>11.1</td>
<td>7.24</td>
</tr>
<tr>
<td>Rotor inertia gcm² 2</td>
<td>181</td>
<td>181</td>
<td>181</td>
<td>181</td>
</tr>
</tbody>
</table>

### Operating Range

<table>
<thead>
<tr>
<th>n [rpm]</th>
<th>10000</th>
<th>9000</th>
<th>7500</th>
<th>6000</th>
<th>5000</th>
<th>4000</th>
</tr>
</thead>
<tbody>
<tr>
<td>[W]</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

- **Continuous operation**
- In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.
- Thermal limit.
- **Short term operation**
- The motor may be briefly overloaded (recurring).
- Assigned power rating

### Other specifications

- Number of pole pairs 8
- Number of phases 3
- Weight of motor 141 g

### Connection

- Pin 1: Hall sensor 1*
- Pin 2: Hall sensor 2*
- Pin 3: VDC, 4.5 – 18 VDC
- Pin 4: Motor winding 3
- Pin 5: Hall sensor 3
- Pin 6: GND
- Pin 7: Motor winding 1
- Pin 8: Motor winding 2

*Internal pull-up (10…15 kΩ) on pin 3

### Wiring diagram for Hall sensors see p. 29

### Cable

- Connection cable Universal, L = 500 mm
- Connection cable to EPOS, L = 500 mm

### Maxon Modular System

**Overview on page 16 - 21**

- Planetary Gearhead 362 mm
- Spur Gearhead 345 mm

### Recommended Electronics:

- ECON 50/5
- DEC 24/3
- DEC Module 50/5
- EPOS2 Module 36/2
- EPOS2 24/2
- EPOS2 24/5
- EPOS2 P 24/5
- EPOS3 70/10 EtherCAT

### Notes

1. Stock program
2. Standard program
3. Special program (on request)
4. Article Numbers
5. Specifications Operating Range
6. Comments
7. Values listed in the table are nominal.
8. Connection
9. Option
10. With Cable and Connector
11. Planetary Gearhead (362 mm)
12. Spur Gearhead (345 mm)
13. With Cable and Connector
14. Ambient temperature -20 … +100°C
15. Recommended Electronics:
16. ECON 50/5
17. DEC 24/3
18. DEC Module 50/5
19. EPOS2 Module 36/2
20. EPOS2 24/2
21. EPOS2 24/5
22. EPOS2 P 24/5
23. EPOS3 70/10 EtherCAT

---

*May 2012 edition / subject to change*
Appendix B

Motor Controller Technical Specifications
## Specifications

### 2.1 Technical Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Electrical Rating</strong></td>
<td></td>
</tr>
<tr>
<td>Nominal operating voltage $V_{cc}$</td>
<td>10…36 VDC</td>
</tr>
<tr>
<td>Absolute operating voltage $V_{cc,min}$ / $V_{cc,max}$</td>
<td>8 VDC / 36 VDC</td>
</tr>
<tr>
<td>Output voltage (max.)</td>
<td>0.98 x $V_{cc}$</td>
</tr>
<tr>
<td>Output current $I_{cont}$ / $I_{max}$ (&lt;4 s)</td>
<td>2.7 A / 9 A</td>
</tr>
<tr>
<td>Pulse Width Modulation frequency</td>
<td>53.6 kHz</td>
</tr>
<tr>
<td>Sampling rate PI current controller</td>
<td>53.6 kHz</td>
</tr>
<tr>
<td>Sampling rate PI speed controller</td>
<td>5.36 kHz</td>
</tr>
<tr>
<td>Max. efficiency</td>
<td>96%</td>
</tr>
<tr>
<td>Max. speed</td>
<td>150'000 rpm (1 pole pair)</td>
</tr>
<tr>
<td>Built-in motor chokes</td>
<td>3 x 47 µH; 2.7 A</td>
</tr>
<tr>
<td><strong>Inputs &amp; Outputs</strong></td>
<td></td>
</tr>
<tr>
<td>Analog Input 1</td>
<td>resolution 12-bit; –10…+10 V; differential</td>
</tr>
<tr>
<td>Analog Input 2</td>
<td>resolution 12-bit; –4…+4 V; referenced to GND</td>
</tr>
<tr>
<td>Analog Output 1</td>
<td>+2.4…+36 VDC (Ri = 38.5 kΩ)</td>
</tr>
<tr>
<td>Analog Output 2</td>
<td>+2.4…+36 VDC (Ri = 38.5 kΩ) / max. 36 VDC (Il &lt;500 mA)</td>
</tr>
<tr>
<td>Digital Input 1</td>
<td>+5 VDC (Il &lt;10 mA)</td>
</tr>
<tr>
<td>Digital Input 2</td>
<td>+5 VDC (Il &lt;30 mA)</td>
</tr>
<tr>
<td>Hall sensor signals</td>
<td>H1, H2, H3</td>
</tr>
<tr>
<td><strong>Voltage Outputs</strong></td>
<td></td>
</tr>
<tr>
<td>Auxiliary output voltage</td>
<td>+5 VDC (Il &lt;10 mA)</td>
</tr>
<tr>
<td>Hall sensor supply voltage</td>
<td>+5 VDC (Il &lt;30 mA)</td>
</tr>
<tr>
<td><strong>Potentiometer</strong></td>
<td>Potentiometer P1 (on board) 210°; linear</td>
</tr>
<tr>
<td><strong>Motor Connections</strong></td>
<td></td>
</tr>
<tr>
<td>EC motor</td>
<td>Motor winding 1, Motor winding 2, Motor winding 3</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td>USB 2.0</td>
</tr>
<tr>
<td><strong>Status Indicators</strong></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>green LED</td>
</tr>
<tr>
<td>Error</td>
<td>red LED</td>
</tr>
<tr>
<td><strong>Physical</strong></td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>approx. 36 g</td>
</tr>
<tr>
<td>Dimensions (L x W x H)</td>
<td>55 x 40 x 19.8 mm</td>
</tr>
<tr>
<td>Mounting holes</td>
<td>for M2.5 screws</td>
</tr>
</tbody>
</table>
Appendix C

Flywheel Drawing
Appendix D

Beagle Bone Specifications
3.0 BeagleBone Features and Specification

This section covers the specifications and features of the BeagleBone and provides a high level description of the major components and interfaces that make up the BeagleBone.

Table 2 provides a list of the BeagleBone’s features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>AM3359</td>
</tr>
<tr>
<td></td>
<td>500MHZ-USB Powered</td>
</tr>
<tr>
<td></td>
<td>720MHZ-DC Powered</td>
</tr>
<tr>
<td>Memory</td>
<td>256MB DDR2 400MHZ (128MB Optional)</td>
</tr>
<tr>
<td>PMIC TPS65217B</td>
<td>Power Regulators</td>
</tr>
<tr>
<td></td>
<td>LiION Single cell battery charger (via expansion*)</td>
</tr>
<tr>
<td></td>
<td>20mA LED Backlight driver, 39V, PWM (via expansion*)</td>
</tr>
<tr>
<td>Debug Support</td>
<td>USB to Serial Adapter</td>
</tr>
<tr>
<td></td>
<td>miniUSB connector</td>
</tr>
<tr>
<td></td>
<td>On Board JTAG via USB</td>
</tr>
<tr>
<td></td>
<td>4 USER LEDs</td>
</tr>
<tr>
<td></td>
<td>Optional 20-pin CTI JTAG</td>
</tr>
<tr>
<td>Power</td>
<td>USB</td>
</tr>
<tr>
<td></td>
<td>5VDC External jack</td>
</tr>
<tr>
<td>PCB</td>
<td>3.4’ x 2.1’</td>
</tr>
<tr>
<td></td>
<td>6 layers</td>
</tr>
<tr>
<td>Indicators</td>
<td>Power</td>
</tr>
<tr>
<td></td>
<td>4-User Controllable LEDs</td>
</tr>
<tr>
<td>HS USB 2.0 Client Port</td>
<td>Access to the USB1 Client mode</td>
</tr>
<tr>
<td>HS USB 2.0 Host Port</td>
<td>USB Type A Socket, 500mA LS/FS/HS</td>
</tr>
<tr>
<td>Ethernet</td>
<td>10/100, RJ45</td>
</tr>
<tr>
<td>SD/MMC Connector</td>
<td>microSD, 3.3V</td>
</tr>
<tr>
<td>User Interface</td>
<td>1-Reset Button</td>
</tr>
<tr>
<td>Overvoltage Protection</td>
<td>Shutdown at 5.6V MAX</td>
</tr>
<tr>
<td></td>
<td>Power 5V, 3.3V, VDD, ADC(1.8V)</td>
</tr>
<tr>
<td></td>
<td>3.3V I/O on all signals</td>
</tr>
<tr>
<td></td>
<td>McASP0, SPI1, I2C, GPIO(65), LCD, GPMC, MMC1, MMC2, 7</td>
</tr>
<tr>
<td></td>
<td>AIN(LAY_MAX), 4 Timers, 3 Serial Ports, CAN0,</td>
</tr>
<tr>
<td></td>
<td>EHRPWM(0,2), XDMA Interrupt, Power button, Battery Charger, LED</td>
</tr>
<tr>
<td></td>
<td>Backlight, Expansion Board ID (Up to 3 can be stacked)</td>
</tr>
<tr>
<td>5V Power</td>
<td>USB or 5.0VDC to 5.2VDC</td>
</tr>
<tr>
<td></td>
<td>See Table 3 for power consumption numbers.</td>
</tr>
<tr>
<td>Weight</td>
<td>1.4 oz (39.68 grams)</td>
</tr>
</tbody>
</table>

*Board will boot to 500MHZ under USB power.

**NOTE:** DUE TO MUXPLEXING ON THE PINS OF THE PROCESSOR, ALL OF THESE EXPANSION SIGNALS CANNOT BE AVAILABLE AT THE SAME TIME.

**NOTE:** The battery configuration is not suitable to power the BeagleBone in its current configuration.

The following sections provide more detail on each feature and are covered under each section of this document.
Appendix E

User Input Variables
<table>
<thead>
<tr>
<th>#</th>
<th>Variable Name</th>
<th>Code Name</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Proportional Feedback Gain Value $K_P$</td>
<td>mPIV.KP</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Integral Feedback Gain Value $K_I$</td>
<td>mPIV.KI</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Velocity Feedback Gain Value $K_V$</td>
<td>mPIV.KV</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>Velocity Feedforward Gain $K_{vff,x}$</td>
<td>mFFGains.Kvff(0)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Velocity Feedforward Gain $K_{vff,y}$</td>
<td>mFFGains.Kvff(1)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Velocity Feedforward Gain $K_{vff,z}$</td>
<td>mFFGains.Kvff(2)</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>Acceleration Feedforward Gain $K_{aff,x}$</td>
<td>mFFGains.KAff(0)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Acceleration Feedforward Gain $K_{aff,y}$</td>
<td>mFFGains.KAff(1)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Acceleration Feedforward Gain $K_{aff,z}$</td>
<td>mFFGains.KAff(2)</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>Damping Coefficient $b$</td>
<td>mSystem.b</td>
<td>kg·m²/sec</td>
</tr>
<tr>
<td></td>
<td>Reaction Wheel Inertia $I_w$</td>
<td>mSystem.Iw</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>System Natural Frequency $w_n$</td>
<td>mSystem.wn</td>
<td>rad/sec</td>
</tr>
<tr>
<td></td>
<td>Maximum Torque $\dot{h}_{\text{max}}$</td>
<td>mSystem.hDotMax</td>
<td>N·m</td>
</tr>
<tr>
<td></td>
<td>Maximum Momentum $h_{\text{max}}$</td>
<td>mSystem.hMax</td>
<td>N·sec</td>
</tr>
<tr>
<td>5</td>
<td>Trajectory Generation Max Acceleration $A_{\text{max}}$</td>
<td>mSystem.Amax</td>
<td>rad/sec²</td>
</tr>
<tr>
<td></td>
<td>Trajectory Generation Max Velocity $V_{\text{max}}$</td>
<td>mSystem.Vmax</td>
<td>rad/sec</td>
</tr>
<tr>
<td>6</td>
<td>Simulator Inertia Component $I_{xx}$</td>
<td>mSystem.Inertia(0)</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>Simulator Inertia Component $I_{xy}$</td>
<td>mSystem.Inertia(1)</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>Simulator Inertia Component $I_{xz}$</td>
<td>mSystem.Inertia(2)</td>
<td>kg·m²</td>
</tr>
<tr>
<td>7</td>
<td>Simulator Inertia Component $I_{yx}$</td>
<td>mSystem.Inertia(3)</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>Simulator Inertia Component $I_{yy}$</td>
<td>mSystem.Inertia(4)</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>Simulator Inertia Component $I_{yz}$</td>
<td>mSystem.Inertia(5)</td>
<td>kg·m²</td>
</tr>
<tr>
<td>8</td>
<td>Simulator Inertia Component $I_{zx}$</td>
<td>mSystem.Inertia(6)</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>Simulator Inertia Component $I_{zy}$</td>
<td>mSystem.Inertia(7)</td>
<td>kg·m²</td>
</tr>
<tr>
<td></td>
<td>Simulator Inertia Component $I_{zz}$</td>
<td>mSystem.Inertia(8)</td>
<td>kg·m²</td>
</tr>
<tr>
<td>9</td>
<td>Max Speed Limit of Reaction Wheel</td>
<td>mSystem.motorSpeedMax</td>
<td>RPM</td>
</tr>
<tr>
<td>10</td>
<td>Conversion from Voltage to Motor Speed</td>
<td>mMotorScale.vToRads</td>
<td>rad/sec-volt</td>
</tr>
<tr>
<td></td>
<td>Conversion from Voltage to Motor Current</td>
<td>mMotorScale.vToAmps</td>
<td>A/V</td>
</tr>
<tr>
<td>11</td>
<td>Total Number of Reference Commands</td>
<td>mTotalRefs</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Total Time to Run Simulator</td>
<td>mStopTime</td>
<td>sec</td>
</tr>
<tr>
<td>12</td>
<td>Time of First Reference Command</td>
<td>mNextReference.time</td>
<td>sec</td>
</tr>
<tr>
<td></td>
<td>Desired First Reference Position $q_1$</td>
<td>mNextReference.q(0)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired First Reference Position $q_2$</td>
<td>mNextReference.q(1)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired First Reference Position $q_3$</td>
<td>mNextReference.q(2)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired First Reference Position $q_4$</td>
<td>mNextReference.q(3)</td>
<td>–</td>
</tr>
<tr>
<td>13</td>
<td>Time of Next Reference Command</td>
<td>mNextReference.time</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired Next Reference Position $q_1$</td>
<td>mNextReference.q(0)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired Next Reference Position $q_2$</td>
<td>mNextReference.q(1)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired Next Reference Position $q_3$</td>
<td>mNextReference.q(2)</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Desired Next Reference Position $q_4$</td>
<td>mNextReference.q(3)</td>
<td>–</td>
</tr>
</tbody>
</table>

Note. # denotes column number.
## Appendix F

### Output File Column Headings

Table F.1: Output File Column Headings

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Code Name</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BB Clock Time</td>
<td>mClock</td>
<td>sec</td>
</tr>
<tr>
<td>2</td>
<td>Measured ( q_1 )</td>
<td>mCurrentQuat(0)</td>
<td>–</td>
</tr>
<tr>
<td>3</td>
<td>Measured ( q_2 )</td>
<td>mCurrentQuat(1)</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>Measured ( q_3 )</td>
<td>mCurrentQuat(2)</td>
<td>–</td>
</tr>
<tr>
<td>5</td>
<td>Measured ( q_4 )</td>
<td>mCurrentQuat(3)</td>
<td>–</td>
</tr>
<tr>
<td>6</td>
<td>Desired Reference Position ( q_1 )</td>
<td>mReference.q(0)</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>Desired Reference Position ( q_2 )</td>
<td>mReference.q(1)</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>Desired Reference Position ( q_3 )</td>
<td>mReference.q(2)</td>
<td>–</td>
</tr>
<tr>
<td>9</td>
<td>Desired Reference Position ( q_4 )</td>
<td>mReference.q(3)</td>
<td>–</td>
</tr>
<tr>
<td>10</td>
<td>Trajectory Generated Reference ( q_1 )</td>
<td>mQuatStar(0)</td>
<td>–</td>
</tr>
<tr>
<td>11</td>
<td>Trajectory Generated Reference ( q_2 )</td>
<td>mQuatStar(1)</td>
<td>–</td>
</tr>
<tr>
<td>12</td>
<td>Trajectory Generated Reference ( q_3 )</td>
<td>mQuatStar(2)</td>
<td>–</td>
</tr>
<tr>
<td>13</td>
<td>Trajectory Generated Reference ( q_4 )</td>
<td>mQuatStar(3)</td>
<td>–</td>
</tr>
<tr>
<td>14</td>
<td>Trajectory Generated Reference ( \omega_1 )</td>
<td>mOmegaStar(0)</td>
<td>(\text{rad/sec})</td>
</tr>
<tr>
<td>15</td>
<td>Trajectory Generated Reference ( \omega_2 )</td>
<td>mOmegaStar(1)</td>
<td>(\text{rad/sec})</td>
</tr>
<tr>
<td>16</td>
<td>Trajectory Generated Reference ( \omega_3 )</td>
<td>mOmegaStar(2)</td>
<td>(\text{rad/sec})</td>
</tr>
<tr>
<td>17</td>
<td>Trajectory Generated Reference ( \alpha_1 )</td>
<td>mAlphaStar(0)</td>
<td>(\text{rad/sec}^2)</td>
</tr>
<tr>
<td>18</td>
<td>Trajectory Generated Reference ( \alpha_2 )</td>
<td>mAlphaStar(1)</td>
<td>(\text{rad/sec}^2)</td>
</tr>
<tr>
<td>19</td>
<td>Trajectory Generated Reference ( \alpha_3 )</td>
<td>mAlphaStar(2)</td>
<td>(\text{rad/sec}^2)</td>
</tr>
<tr>
<td>20</td>
<td>Quaternion Error ( q_1 )</td>
<td>mQuatError(0)</td>
<td>–</td>
</tr>
<tr>
<td>21</td>
<td>Quaternion Error ( q_2 )</td>
<td>mQuatError(1)</td>
<td>–</td>
</tr>
<tr>
<td>22</td>
<td>Quaternion Error ( q_3 )</td>
<td>mQuatError(2)</td>
<td>–</td>
</tr>
<tr>
<td>23</td>
<td>Quaternion Error ( q_4 )</td>
<td>mQuatError(3)</td>
<td>–</td>
</tr>
<tr>
<td>24</td>
<td>Calculated Command ( T_x )</td>
<td>mTc3(0)</td>
<td>(\text{N} \cdot \text{m})</td>
</tr>
<tr>
<td>25</td>
<td>Calculated Command ( T_y )</td>
<td>mTc3(1)</td>
<td>(\text{N} \cdot \text{m})</td>
</tr>
<tr>
<td>26</td>
<td>Calculated Command ( T_z )</td>
<td>mTc3(2)</td>
<td>(\text{N} \cdot \text{m})</td>
</tr>
<tr>
<td>27</td>
<td>Wheel 0 Command Torque ( T_0 )</td>
<td>mTorque(0)</td>
<td>(\text{N} \cdot \text{m})</td>
</tr>
<tr>
<td>28</td>
<td>Wheel 1 Command Torque ( T_1 )</td>
<td>mTorque(1)</td>
<td>(\text{N} \cdot \text{m})</td>
</tr>
</tbody>
</table>

(continued on next page)
Table F.1: Output File Column Headings

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Code Name</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>Wheel 2 Command Torque $T_2$</td>
<td>mTorque(2)</td>
<td>N · m</td>
</tr>
<tr>
<td>30</td>
<td>Wheel 3 Command Torque $T_3$</td>
<td>mTorque(3)</td>
<td>N · m</td>
</tr>
<tr>
<td>31</td>
<td>Wheel 0 Speed Command</td>
<td>mSpeedCmd(0)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>32</td>
<td>Wheel 1 Speed Command</td>
<td>mSpeedCmd(1)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>33</td>
<td>Wheel 2 Speed Command</td>
<td>mSpeedCmd(2)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>34</td>
<td>Wheel 3 Speed Command</td>
<td>mSpeedCmd(3)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>35</td>
<td>Wheel 0 Duty Cycle Command</td>
<td>mDutyC(0)</td>
<td>%</td>
</tr>
<tr>
<td>36</td>
<td>Wheel 1 Duty Cycle Command</td>
<td>mDutyC(1)</td>
<td>%</td>
</tr>
<tr>
<td>37</td>
<td>Wheel 2 Duty Cycle Command</td>
<td>mDutyC(2)</td>
<td>%</td>
</tr>
<tr>
<td>38</td>
<td>Wheel 3 Duty Cycle Command</td>
<td>mDutyC(3)</td>
<td>%</td>
</tr>
<tr>
<td>39</td>
<td>IMU Clock Time</td>
<td>mImuTime</td>
<td>sec</td>
</tr>
<tr>
<td>40</td>
<td>Measured Roll φ</td>
<td>mEuler(0)</td>
<td>rad</td>
</tr>
<tr>
<td>41</td>
<td>Measured Pitch θ</td>
<td>mEuler(1)</td>
<td>rad</td>
</tr>
<tr>
<td>42</td>
<td>Measured Yaw ψ</td>
<td>mEuler(2)</td>
<td>rad</td>
</tr>
<tr>
<td>43</td>
<td>Measured Roll ˙φ</td>
<td>mCurrentRates(0)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>44</td>
<td>Measured Pitch ˙θ</td>
<td>mCurrentRates(1)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>45</td>
<td>Measured Yaw ˙ψ</td>
<td>mCurrentRates(2)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>46</td>
<td>Measured Wheel 0 Speed</td>
<td>mSpeed(0)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>47</td>
<td>Measured Wheel 1 Speed</td>
<td>mSpeed(1)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>48</td>
<td>Measured Wheel 2 Speed</td>
<td>mSpeed(2)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>49</td>
<td>Measured Wheel 3 Speed</td>
<td>mSpeed(3)</td>
<td>rad/sec</td>
</tr>
<tr>
<td>50</td>
<td>Measured Wheel 0 Current</td>
<td>mAmps(0)</td>
<td>amperes</td>
</tr>
<tr>
<td>51</td>
<td>Measured Wheel 1 Current</td>
<td>mAmps(1)</td>
<td>amperes</td>
</tr>
<tr>
<td>52</td>
<td>Measured Wheel 2 Current</td>
<td>mAmps(2)</td>
<td>amperes</td>
</tr>
<tr>
<td>53</td>
<td>Measured Wheel 3 Current</td>
<td>mAmps(3)</td>
<td>amperes</td>
</tr>
</tbody>
</table>
Appendix G

Simulator Instructions

Chipsat for Dummies

UPDATE AND RUN CODE

- Chipsat folder contains all code (controller) - workspace folder will contain laptop copy
- run 'mworkspace' - make sure to reset time every time power on BB - to synchronize laptop
  code to BB: - make sure to update date/time: don’t do this: (sudo ln -s /usr/share/zoneinfo/Etc/GMT-
  7 /etc/localtime) how I set it up, don’t do this again 'sudo date MMDDHHmmYYYY’ -
  'rsync -av workspace/chipsat chipsat’ or use alias 'sync_code’ (after running ‘mworkspace’)
- make sure to be in BB home folder ('cd’ if not sure) - if building everything for the first time:
- get into Chipsat/build folder - 'cmake ..’ - compile: 'make’ - if building just updated code
- get into Chipsat/build folder - compile: 'make’ - if problems building/compiling: - 'make clean’
  - to run program from the build directory: - 'sudo ../bin/chipsat'

TEST IF IMU IS WORKING

- go to chipsat/python - run 'python serialtest.py’ - stop with Ctrl C

COPY FILES FROM BB TO MY LAPTOP - type ‘mworkspace’. This is an alias that
  mounts my computer, so the password to my computer will be requested - 'sudo cp filename
  workspace’. 'workspace’ is an alias used to define the path on my computer to send the file
  to - 'sudo umount workspace’

  IF I NEED TO KILL MWORKSPACE: sudo pkill -9 -f mworkspace

  to look if process is still running: ps ax

RUNNING THE SIMULATOR

Start connected.

Type the command "byobu". This brings up a new shell.
In this shell, try "sudo ping" and it will likely ask you for the ubuntu password (which is "ubuntu"). It is helpful to do this before you disconnect and subsequently can’t type the password if it’s asking for it.

Type: "wait 20; sudo sudo ~/chipsat/bin/chipsat -i ~/chipsat/bin/chipsat/input-1.txt -o ~/chipsat/bin/chipsat/outputfilename.csv"

This waits 20 seconds after you hit Enter before executing the command. Wait a few moments after you hit enter to then do this key sequence:

(function) F6

This opens another window. In this window type the command "exit". Hit Enter.

Now you can physically disconnect the Ethernet cable and when the 20 seconds is up, the program will start running. Take care after removing the Ethernet cable to balance the table and release gently and cause as little disturbance as possible.

In the input file you can set the amount of time to run the program.
Appendix H

Air Bearing Base Drawing
Appendix I

First Order Response Wheel Modeling
Table I.1: Wheel Speed Step Responses

<table>
<thead>
<tr>
<th>$\omega_{\text{cmd}}$ rad/sec</th>
<th>$\omega_{\text{meas0}}$ rad/sec</th>
<th>$\omega_{\text{meas1}}$ rad/sec</th>
<th>$\omega_{\text{meas2}}$ rad/sec</th>
<th>$\omega_{\text{meas3}}$ rad/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>-523.6</td>
<td>-456</td>
<td>-457</td>
<td>-453</td>
<td>-457</td>
</tr>
<tr>
<td>-458.1</td>
<td>-399</td>
<td>-391</td>
<td>-389</td>
<td>-403</td>
</tr>
<tr>
<td>-392.7</td>
<td>-320</td>
<td>-330</td>
<td>-325</td>
<td>-320</td>
</tr>
<tr>
<td>-327.2</td>
<td>-260</td>
<td>-261</td>
<td>-260</td>
<td>-260</td>
</tr>
<tr>
<td>-294.5</td>
<td>-225</td>
<td>-230</td>
<td>-228.1</td>
<td>-230</td>
</tr>
<tr>
<td>-229.1</td>
<td>-163</td>
<td>-163</td>
<td>-162.4</td>
<td>-157</td>
</tr>
<tr>
<td>-163.6</td>
<td>-99</td>
<td>-99</td>
<td>-99.32</td>
<td>-99</td>
</tr>
<tr>
<td>-98.17</td>
<td>-33.23</td>
<td>-33.61</td>
<td>-34.74</td>
<td>-34</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>98.17</td>
<td>28.7</td>
<td>30</td>
<td>29.46</td>
<td>30</td>
</tr>
<tr>
<td>229.1</td>
<td>100</td>
<td>93</td>
<td>93.65</td>
<td>96</td>
</tr>
<tr>
<td>294.5</td>
<td>224</td>
<td>224</td>
<td>222.4</td>
<td>225</td>
</tr>
<tr>
<td>327.2</td>
<td>259</td>
<td>252</td>
<td>254</td>
<td>255</td>
</tr>
<tr>
<td>392.7</td>
<td>324</td>
<td>321</td>
<td>320</td>
<td>318</td>
</tr>
<tr>
<td>458.1</td>
<td>384</td>
<td>389</td>
<td>385</td>
<td>384</td>
</tr>
</tbody>
</table>

Table I.2: Wheel Time Constants

<table>
<thead>
<tr>
<th>$\omega_{\text{cmd}}$ rad/sec</th>
<th>$\tau_{\text{cmd0}}$ ms</th>
<th>$\tau_{\text{cmd1}}$ ms</th>
<th>$\tau_{\text{cmd2}}$ ms</th>
<th>$\tau_{\text{cmd3}}$ ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>-523.6</td>
<td>220</td>
<td>225</td>
<td>249</td>
<td>219</td>
</tr>
<tr>
<td>-458.1</td>
<td>185</td>
<td>180</td>
<td>207</td>
<td>172</td>
</tr>
<tr>
<td>-392.7</td>
<td>148</td>
<td>145</td>
<td>190</td>
<td>156</td>
</tr>
<tr>
<td>-327.2</td>
<td>123</td>
<td>118</td>
<td>153</td>
<td>165</td>
</tr>
<tr>
<td>-294.5</td>
<td>115</td>
<td>106</td>
<td>147</td>
<td>98</td>
</tr>
<tr>
<td>-229.1</td>
<td>77</td>
<td>72</td>
<td>107</td>
<td>75</td>
</tr>
<tr>
<td>-163.6</td>
<td>60</td>
<td>50</td>
<td>92</td>
<td>50</td>
</tr>
<tr>
<td>-98.17</td>
<td>30</td>
<td>30</td>
<td>50</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>98.17</td>
<td>35</td>
<td>33</td>
<td>66</td>
<td>32</td>
</tr>
<tr>
<td>229.1</td>
<td>92</td>
<td>87</td>
<td>119</td>
<td>90</td>
</tr>
<tr>
<td>294.5</td>
<td>115</td>
<td>115</td>
<td>149</td>
<td>105</td>
</tr>
<tr>
<td>327.2</td>
<td>130</td>
<td>125</td>
<td>155</td>
<td>117</td>
</tr>
<tr>
<td>392.7</td>
<td>180</td>
<td>170</td>
<td>207</td>
<td>165</td>
</tr>
<tr>
<td>458.1</td>
<td>190</td>
<td>183</td>
<td>218</td>
<td>174</td>
</tr>
</tbody>
</table>