MODIFICATION OF THE CAL POLY SPACECRAFT SIMULATOR SYSTEM FOR ROBUST CONTROL LAW VERIFICATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

Tomoyuki Kato

June 2014

Committee Membership

Title:	Modification of the Cal Poly Spacecraft Simulator System for Robust Control Law Verification
Author:	Tomoyuki Kato
Date Submitted:	June 2014
Committee Chair:	Eric Mehiel, PhD Associate Professor of Aerospace Engineering
Committee Member:	Charles Birdsong, PhD Professor of Mechanical Engineering
Committee Member:	Kira Abercromby, PhD Associate Professor of Aerospace Engineering
Committee Member:	Jordi Puig-Suari, PhD Professor of Aerospace Engineering

ABSTRACT

Modification of the Cal Poly Spacecraft Simulator System
for Robust Control Law Verification

Tomoyuki Kato

The Cal Poly Spacecraft Dynamics Simulator, also known as the Pyramidal Reaction Wheel Platform (PRWP), is an air-bearing four reaction wheel spacecraft simulator designed to simulate the low-gravity, frictionless condition of the space environment and to test and validate spacecraft attitude control hardware and control laws through real-time motion tests. The PRWP system was modified to the new Mk.III configuration, which adopted the MATLAB xPC kernel for better real-time hardware control. Also the Litton LN-200 IMU was integrated onto the PRWP and replaced the previous attitude sensor. Through the comparison of various control laws through motion tests the Mk.III configuration was tested for robust control law verification capability. Two fixed-gain controllers, full-state feedback (FSFB) and linear quadratic regulator with set-point control(LQRSP), and two adaptive controllers, nonlinear direct model reference adaptive controller (NDMRAC) and the adaptive output feedback (AOF), were each tested in three different cases of varying plant parameters to test controller robustness through real-time motion tests. The first two test cases simulate PRWP inertia tensor variations. The third test case simulates uncertainty of the reaction wheel dynamic by slowing down the response time for one of the four reaction wheels. The Mk.III motion tests were also compared with numerical simulations as well as the older Mk.II motion tests to confirm controller validation capability. The Mk.III test results confirmed certain patterns from the numerical simulations and the Mk.II test results. The test case in which actuator dynamics uncertainty was simulated had the most effect on controller performance, as all four control laws experienced an increase in steady-state error. The Mk.III test results also confirmed that the NDMRAC outperformed the fixed-gain controllers.

Acknowledgments

To Dr. Mehiel, for your guidance and for giving me a chance to work on the most challenging and rewarding project I have ever been a part of.

To Phil Bruner, for investing so much of your time and resource in me.

To my Mom and Grandma, for making my education and my life possible.

Contents

List of Tables

List of Figures

Nomenclature

ω	angular velocity
ω_n	natural frequency
ω_{wl}	wheel speed
$\vec{q}$	quaternion vector
ξ	damping ratio
A	state-space A matrix
B	state-space B matrix
C	state-space C matrix
C_{FSFB}	full-state feedback C gain
D	state-space D matrix
e_y	plant error
F	set-point controller feedforward gain
G	controlled output vector state-space G matrix
g	gravitational acceleration
G_e	NDMRAC and AOF error gain
H	controlled output vector state-space H matrix
I_{est}	solid-model inertia tensor estimate
I_{sysid}	system ID inertia tensor estimate

J	inertia tensor
K	linear quadratic regulator gain
K_{FSFB}	full-state feedback K gain
m	mass
N	set-point controller feedforward gain
q_4	quaternion scalar term
q_e	quaternion error
r	center-of-gravity vector in inertial frame
R_E	inertial to body frame transformation matrix
R_{w_l}	wheel frame to body frame transformation matrix
S_{21}	NDMRAC state gain
S_{22}	NDMRAC input gain
u	plant input
u_m	reference model input
x	plant state vector
x_m	reference model state
y_m	reference model output
y_p	plant output

Chapter 1

Introduction

With the rise of spacecraft technology, the demand for thorough testing of spacecraft systems has grown tremendously. Due to the difficulty and expense of reaching outer space, space systems are often not expected to be recovered. Therefore spacecraft are thoroughly tested before launch in order for developers to ensure that their space systems will function as intended. In the context of spacecraft attitude dynamics, it is crucial on the supplier's side to ensure that their space systems are capable of orienting itself properly such that the onboard payload can function properly.

One method for validating spacecraft attitude control systems is the development of numerical simulations of the spacecraft. To an extent, simulations can offer insight into how well the spacecraft attitude control system can function within its operating environment for its intended mission. However numerical simulations are based on mathematical models and cannot predict the operating environment and the physical space system perfectly. Especially among space systems that have a high pointing-accuracy requirement such as communication satellites and telescopes, the importance of validating attitude control systems before launch cannot be overstated.

An alternative method to spacecraft ground testing would be to recreate the system's operating environment. However the presence of the Earth's gravity makes it difficult to simulate the space environment. Particularly it is important to simulate the micro-gravity nature of the space environment in order to accurately evaluate the effectiveness of the spacecraft attitude control system's capability to orient the space system and to guarantee a specific amount of pointing-accuracy.

An air-bearing spacecraft simulator (ABSS) is a testbed that is capable of simulating a micro-gravity environment. First developed by the United States military[1], the ABSS consists of a structure that holds attitude control hardware and an air compressor system. The structure is

supported on an air-bearing with compressed air flowing between the bearing surfaces. The film of compressed air between the bearing surfaces counteracts the testbed weight and therefore simulates a low-gravity and frictionless environment. The simulation of the space environment allows for the operation of hardware such as reaction wheels and control moment gyroscopes (CMG), and enables the observation of dynamic phenomenon such as the effect of structural vibration and mass property variation on the attitude control system performance. Therefore the ABSS can be utilized to perform real-time attitude control tests and to evaluate control law performance and robustness.

However one condition for the ABSS to perform effectively is to align the ABSS center-of-gravity (CG) with the air bearing center. Otherwise an offset between the CG and the bearing center will introduce an external torque onto the ABSS system due to gravity and therefore necessitate greater controller effort to compensate for the external torque. The ABSS has been proven as a reliable testbed for validating numerous control laws and attitude control hardware such as reaction wheels, CMGs, and thrusters. [2][3][4][5]

This thesis is organized as follows. The rest of Chapter 1 will cover the literature review as well as a brief history of the development of the Cal Poly spacecraft simulator, also known as the pyramidal reaction wheel platform (PRWP), and conclude with a statement of the thesis objective. Chapter 2 will give an overview of the hardware and software system on the current configuration of the Cal Poly Spacecraft Dynamics Simulator as well as a description of the validation process of integrating the LN-200 IMU onboard the PRWP. In chapter 3 the governing equations of spacecraft dynamics and kinematics as well as the various control laws of this project will be described. Chapter 4 will cover the motion test experiments. The results of the motion test experiments with the LN-200 IMU onboard will be discussed in chapter 5. Concluding remarks as well as recommendations for further development of the PRWP will be made in chapter 6.

1.1 Literature Review

This section will discuss relevant concepts of this investigation as well as discuss the results of related publications.

1.1.1 Control Law Validation

To manipulate the behavior of an output of a dynamic system, a control law must be developed. The attitude of a spacecraft is one such case of a dynamic system whose output is of interest and requires a sophisticated controller to ensure a specific output behavior. However the development

of a control law is only a part of the process, as the controller performance must be verified through numerical simulations and real-time hardware-in-loop tests. Due to the ABSS's ability to simulate a micro-gravity environment, the testbed is utilized to verify various control laws applied to spacecraft attitude control. The ABSS experimental results serve as a good comparison with numerical simulation results, and the comparison can either confirm the numerical simulation results or expose certain phenomenon and disturbances the numerical simulation did not account for.

Jung and Tsiotras[6] validated a quaternion-based nonlinear feedback control law through motion tests on an ABSS system. Though the controller was originally developed by Joshi et al[7], the control law was validated through both a numerical simulation as well as a motion test on a cylindrical spacecraft simulator. First the simulator CG was balanced through the combination of a least-squares system identification algorithm and mass balancers, then the feedback control law was tested by simulating an attitude maneuver. Starting at non-zero Euler angles, the spacecraft system was commanded to an all-zero Euler angle. Both the simulation and motion test results consistently show that the feedback controller effectively controlled the simulator attitude.

Romano and Agrawal[8] validated a PD control law for a bifocal relay mirror spacecraft simulator. The simulator emulates a space system designed to redirect a laser beam from a ground source to a different destination. Therefore the proposed PD controller was responsible for both the attitude of the simulator and rejecting laser beam jitter to properly reflect the entire laser beam. The PD controller was tested both in a numerical simulation as well as a hardware-in-loop experiment, and both results were consistent in showing that the simulator system is capable of guaranteeing a pointing accuracy of 0.1 degrees and a angular velocity error of less than 0.02 degrees per second.

Gao et al [9] validated a PD feedback controller on a single reaction wheel table-configuration spacecraft simulator. Although the simulator is constrained to move only about its yaw axis, the simulator successfully executed a yaw rotation of 10 degrees using the PD controller.

1.1.2 Adaptive Control

An inherent problem with fixed-gain control laws is that the gains are dependent on knowing the plant dynamics and parameters. A sudden change in the plant parameter may actually alter the behavior of the system of interest and therefore the plant model may no longer be valid. For example for a space system, the deployment of solar panels and booms will change its mass property. To state another example an aircraft system's CG will change as it consumes fuel and load or unload supplies and munition. Therefore robustness is a quality that is highly desirable for control laws. Robustness

is a controller's capacity to meet specific control performance requirements despite variations in plant model parameters.[10][11] Although fixed-gain controllers can be designed to be robust, such controllers still expect a specific plant model. If there is a sufficient amount of plant parameter variations, fixed-gain controllers may not be robust enough to guarantee a certain output behavior or the system may even become unstable.

Adaptive control is a type of controller that is more capable of dealing with uncertain plant dynamics and disturbances. While most controllers are designed to control the system output for a given system condition, the adaptive controller is a controller that changes itself in response to the changes in plant dynamics. A particular type of adaptive control called model reference adaptive control, which this investigation will focus on, does not rely on knowledge of the plant dynamics. Instead, model reference adaptive controllers rely on a reference model which the controller forces the plant to track. Therefore model reference adaptive controllers tend to be more robust compared to fixed-gain controllers due to dependence on plant output rather than a plant model[12]. Adaptive control itself has many different variations and is a widely-used solution for many different systems today[13][14][15].

Adaptive control has been demonstrated to be a viable solution for spacecraft attitude control and for balancing ABSS systems. Kim and Agrawal [16] applies adaptive control to an autonomous CG balancing system for an air-bearing spacecraft simulator. This was accomplished by ensuring the simulator's total angular momentum tracks a specified trajectory using Lyapunov functions to prove asymptotic stability of the angular momentum. When the simulator angular momentum follows the ideal trajectory, the CG offset torque will be minimized.

Swei [17] develops a bounded adaptive controller to align the spacecraft vector with the sun vector in order to maximize the power collection efficiency of a solar panel. The adaptive controller takes into account the spacecraft's limitation in producing torque, therefore the adaptive controller parameters are bounded as to limit the torque command.

Yoon and Tsiotras [18] develop an adaptive controller that compensates for actuator uncertainties. The authors simulated uncertainties with a variable-speed control moment gyroscope such that its spin, gimbal, and transverse axis had a degree of uncertainty which affects the transformation matrix between the CMG frame and the platform body frame. The authors ran numerical simulations of both a nominal controller and the adaptive controller, and showed that the adaptive controller reduced the Euler-angle error to less than 0.1 degrees, as opposed to the nominal controller which suffered errors of almost 5 degrees.

While adaptive controllers show great potential for application on spacecraft attitude control

systems, it remains to be seen how well adaptive control works in a motion test experiment such as on an ABSS system. Though Kim and Agrawal as previously mentioned have applied adaptive control to the CG balancing system of an ABSS system, it is of great interest to evaluate the capability of adaptive controllers for ABSS motion tests. The application of adaptive controllers to the PRWP will be discussed further in section 1.3.2.

1.2 History of the Cal Poly Spacecraft Dynamics Simulator

This chapter will give a brief history of the evolution of the PRWP, from its creation to its current state of development. The initial design was developed by Mittlestead[19]. Figure 1.1 shows the first configuration of the Cal Poly spacecraft simulator, also referred to as the Mk. I.

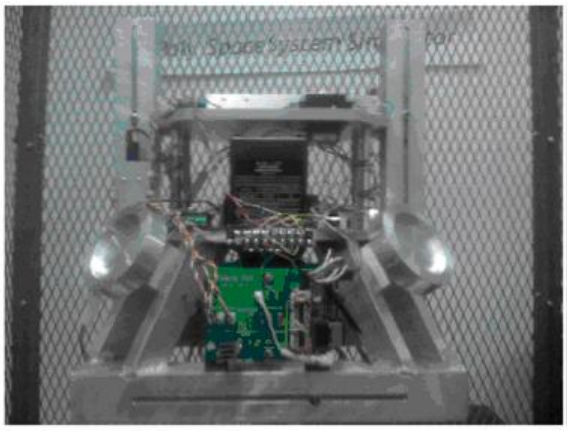

Figure 1.1: Cal Poly Spacecraft Simulator Mk.I.

Although the individual components of the Mk.I were functional, the Mk.I was ultimately shown to be incapable of executing real-time motion tests. The data processing limitation of the Bluetooth link and the onboard CPU prevented the Mk.I from being a viable system for real-time motion tests.

Therefore Kinnett[20] and Downs[21] revised the system hardware and software such that, for the first time, the PRWP was capable of closed-loop attitude control in real-time. The most recent configuration of the PRWP, the Mk. II, is shown in Figure 1.2.

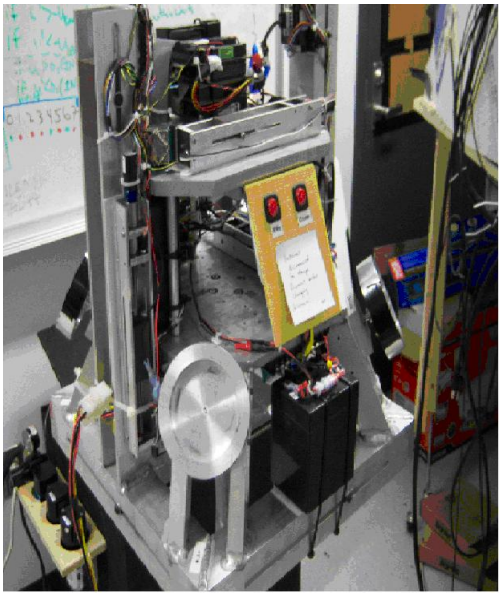

Figure 1.2: Cal Poly Spacecraft Simulator Mk.II.

The Mk.I CPU was replaced with a PC/104 form-factor CPU using the Windows XP operating system and running the MATLAB/Simulink program onboard. Although the Mark II configuration was functional, it was also electrically unreliable. Unfortunately various incidents of electrical failures have caused the failure of some of the hardware and a new revision of the PRWP system was required, which resulted in the Mk. III configuration. The newest PRWP configuration will be described in further detail in Chapter 2.

1.3 Thesis Objectives

This section will discuss the concurrent development on the PRWP as well as the actual thesis objectives of interest.

1.3.1 Concurrent Development

Aside from myself another graduate student, Long Dam, is involved in the spacecraft simulator project. Dam's project will involve the implementation of the system identification algorithm and the fine mass balancers for the purpose of the PRWP inertia estimations and to better balance the platform[22]. Although the hardware and software have been previously developed by Sailes[23] and Silva[24] respectively, the CG correction system have had limited results due to the Mk.II configuration hardware limitations such as the attitude determination sensor bias drift.

1.3.2 System Modification and Controller Validation

The first objective of this project is to integrate both the Litton LN-200 inertial measurement unit (IMU) and the MATLAB xPC kernel onboard the PRWP. Although the Mark II configuration of the PRWP was functional, a combination of onboard hardware failures and the control law verification limitation motivated a modification of the PRWP system. The Mark II configuration of the PRWP utilizes the CRS03 MEMS gyroscope for attitude determination. However the MEMS gyroscopes have a high bias drift, which adds to the difficulty of evaluating control laws for attitude control. Therefore the Litton LN-200 IMU, loaned to California Polytechnic State University by the Northrop Grumman Corporation, will replace the MEMS gyroscopes in order to improve the PRWP's attitude determination capability and in turn improve the PRWP's capability to validate control laws.

However along with the LN-200 IMU, other PRWP system hardware had to be replaced due to the destruction of much of the Mark II hardware due to several incidents of malfunctions. Therefore a decision was made to switch the PRWP CPU operating system from Windows XP to the MATLAB xPC kernel, since the xPC kernel is explicitly designed for real-time hardware tests. The greater computational efficiency of the xPC kernel offered a wider variety of hardware to choose for the PRWP system compared to the Windows XP operating system. The newer PRWP configuration with the xPC kernel system as well as the LN-200 IMU integration process will be explained in further detail in Chapter 2.

The second objective of this project is to verify the PRWP's capability for verifying control laws for spacecraft attitude control systems. Although ABSS systems have been proven to be an effective ground test for control laws applied to spacecraft attitude control, much work have been done on fixed-gain controllers such as PD control laws. Though numerous works as discussed in the literature review have shown adaptive control law to have great potential for application on spacecraft attitude control systems due to its robustness, those works have been done on numerical simulations. However, Downs[21] has tested adaptive control laws onboard the Mk.II configuration of the PRWP and compared adaptive controllers to a fixed-gain controller. Although Downs' comparison was weakened by the high bias drift of the attitude determination sensor, his investigation was able to show that adaptive controllers consistently outperformed the fixed-gain controller. Therefore it is of interest to see how well the new Mk.III configuration of the PRWP, with the LN-200 IMU as the attitude determination sensor, can test the performance of various control laws including adaptive controllers. Similar to Downs', two variations of adaptive controllers will be compared to

a fixed-gain control law by executing motion tests on the PRWP for three separate test cases where plant parameter variations will be simulated. The first two test cases will simulate variations of the PRWP mass property, and the third case will simulate uncertainty in the reaction wheel dynamic. The purpose of the three test cases are to test the control law robustness. The controllers will be evaluated based on how well they can control the simulator attitude amidst uncertainties in the plant model.

Chapter 2

Overview of the PRWP System

This chapter will give a brief description of the PRWP hardware and of the Mk.III configuration. Section 2.1 will explain the motivation for the Mk.III configuration of the PRWP. Section 2.2 will briefly describe the hardware and software modification made to the PRWP. Section 2.3 will describe the LN-200 IMU integration and validation process.

2.1 Motivation for the Mk.III Configuration

Although the Mk.II was a functional configuration of the PRWP, a redesign of the system was motivated by a combination of hardware and software difficulties, and the destruction of much of the Mk.II components. Most of the reaction wheel circuitry were destroyed due to poor power management and poor soldering which led to the shorting of the circuits. Therefore the entire reaction wheel system was replaced. All four motors were replaced with brushless DC motors which have an integrated motion controller device with it. The motion controller is only a fraction of the size of the Mk.II reaction wheel circuitry, and is far more reliable due to it being professionally manufactured.

Also a decision was made to move away from using the Windows XP operating system running MATLAB/Simulink and to adopt the MATLAB xPC kernel because the xPC kernel is specifically designed for real-time hardware control. The Matlab xPC is specified to support real-time simulation for a sample time as fast as $10\mu s$[1]. Although running Simulink on the Windows XP operating system was capable of real-time control on the Mk.II configuration, the configuration was limited to using components using analog or digital I/O connections due to its low data transfer capability. Therefore

[1] http://www.mathworks.com/help/releases/R13sp2/pdf_doc/xpc/xpc_target_gs.pdf

this would present a restriction in the choice of hardware if more parts are to be added in the future. The Mk.III configuration is shown in Figure 2.1.

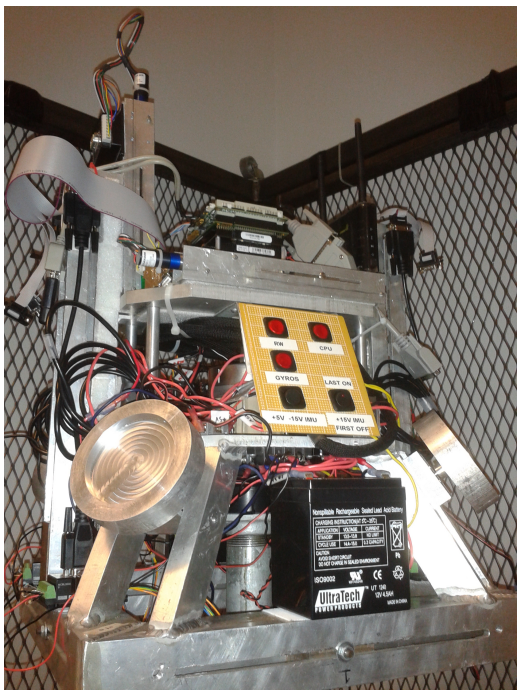

Figure 2.1: Cal Poly Spacecraft Dynamics Simulator Mk.III Configuration.

2.2 Hardware and Software Overview

The schematic of the Mk.III configuration is shown in Figure 2.2.

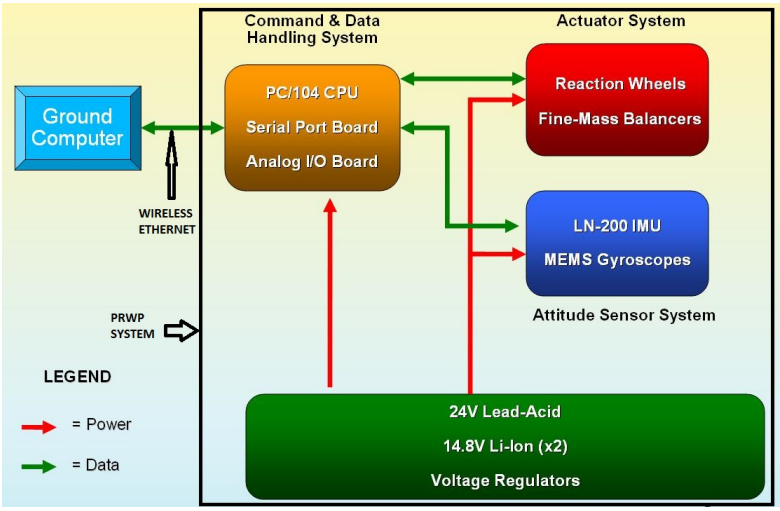

Figure 2.2: PRWP Mk.III System Diagram.

The power subsystem consists of batteries and voltage regulators and they are responsible for providing regulated power to all the other subsystems. The actuator subsystem consists of the reaction wheels and the fine-mass balancers (FMB) which provide attitude control and CG adjustment capability for the PRWP respectively. The attitude sensor subsystem provides rotational velocity measurements for the PRWP. The command and data-handling subsystem is responsible for communicating between the other subsystems and the ground computer.

The motion test experiment involves numerous interaction between the PRWP hardware and software. The motion test process is explained further in the following section.

2.2.1 PRWP Test Process

The execution of the motion tests on the PRWP is made possible by the interaction between the PRWP hardware, software, and the ground computer. The motion test process diagram is shown in Figure 2.3.

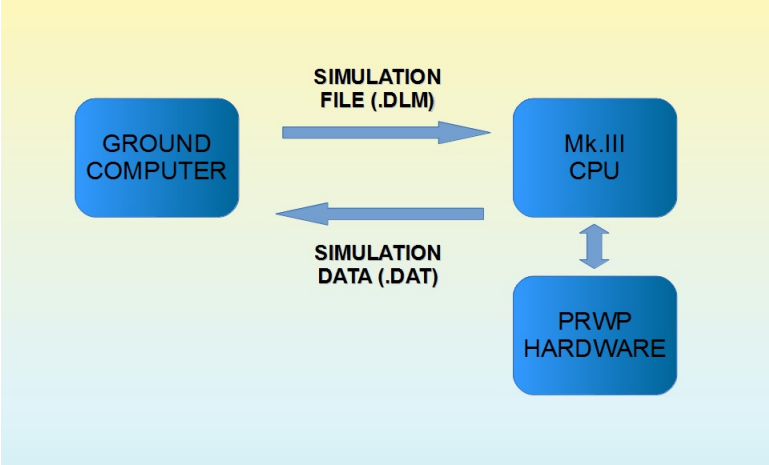

Figure 2.3: PRWP Motion Test Process Diagram.

Unlike the Mk.II configuration, the ground computer serves a greater role in the function of the motion tests. The ground computer is a desktop PC using the Windows XP operating system and running MATLAB/Simulink. The motion test process is as follows,

1. The motion test simulation file is developed in the Simulink environment on the ground computer.

2. The motion test simulation file is then compiled by MATLAB into the .DLM file format which can be used by the MATLAB xPC kernel on the Mk.III CPU.

3. The .DLM motion test file is transferred via wireless Ethernet link from the ground computer to the Mk.III CPU.

4. The MATLAB xPC kernel on the Mk.III interprets the .DLM file

    i. The xPC kernel reads and writes to the Mk.III hardware

    ii. The xPC kernel stores the data from the simulation.

5. Once the motion test is executed, the Mk.III saves simulation data as .DAT files and transfers the .DAT files via wireless Ethernet link back to the ground computer for data processing.

In the Mk. II configuration all the computational process occurred on the PRWP CPU and the only role of the ground computer was to communicate with the PRWP CPU. However the difference with the Mk.III configuration is that some of the computational effort is transferred to the ground computer, and the Mk.III CPU is only responsible for hardware interaction and data storage. Now that the motion test process have been explained, the PRWP CG balancing system will be discussed.

2.2.2 PRWP CG Balancing System

Due to the nature of ABSS systems, it is important to balance the PRWP CG as close to the air-bearing center as possible. Otherwise the PRWP will experience an additional external torque known as the CG offset torque. This effect is problematic because during the motion tests the PRWP control law and reaction wheels will have to expend additional effort to counteract the external torque. This can lead to the reaction wheels reaching its wheel speed saturation limits faster, which will limit the motion range of the PRWP. The PRWP CG balancing system process is illustrated in the diagram in Figure 2.4.

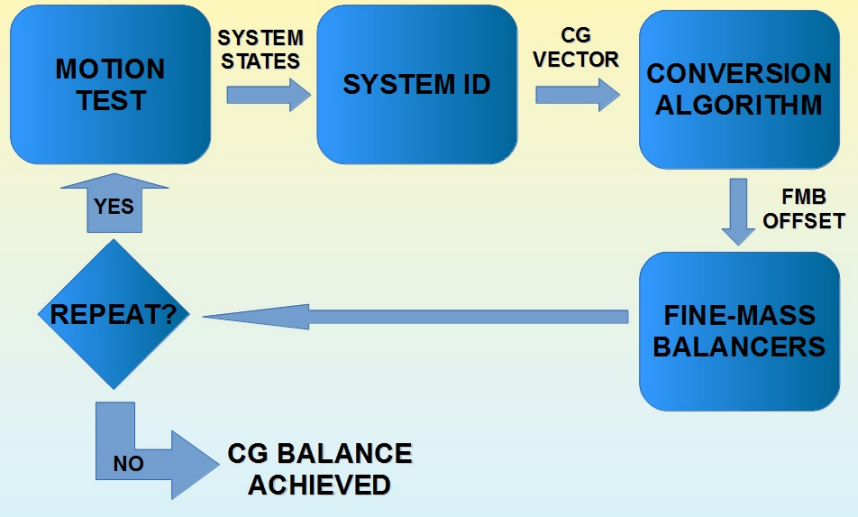

Figure 2.4: PRWP CG Balancing System Diagram.

The PRWP CG balancing process can be further described as follows,

1. The CG vector is determined by the PRWP through a specific motion test.
2. The CG vector is used to determine to calculate the offset distance necessary for the FMBs.
3. The FMBs move its counterweight blocks to shift the CG location.
4. The process is repeated until the operator determines the CG location is acceptable.

Various algorithms in determining and balancing the CG of ABSS systems have been developed by others[16][25]. The method utilized for the determination of the PRWP CG location is called the system identification (ID) algorithm. The role of the system ID algorithm for the operation of the PRWP is further explained in Chapter 3.3. The CG location supplied by the system ID algorithm is then processed by another algorithm to determine the distance that the FMBs need to move their counterweight blocks to adjust the CG. The full detail of the balancing system is beyond the scope of this project and the reader is referred to Dam for a complete explanation of all the algorithm involved in the balancing process[22].

2.3 IMU Integration Requirements

The Litton LN-200 inertial measurement unit (IMU) is a space-rated hardware for the purpose of attitude determination. The device is shown in Figure 2.5.

Figure 2.5: Litton-200 Inertial Measurement Unit.

The unit houses three accelerometers and three fiber-optic gyroscopes (FOG), and has a bias drift rate of less than 3^o per hour. The LN-200 has a rich heritage, being utilized various space systems such as the Mars *Spirit* and *Opportunity* rovers[2]. The particular IMU used for the PRWP simulator was loaned to Cal Poly by the Northrop Grumman Corporation. Phil Iversen[26] developed a test procedure to independently test the LN-200. To install the LN-200 onboard the PRWP, various requirements must be understood and satisfied. The following sections will go into detail about the necessary requirements and the validation process of the integration.

2.3.1 Mechanical Requirement

The IMU was mounted onboard the bottom deck of the PRWP in order to provide conduction cooling from the primary structure and to place the device as close to the CPU as possible. The IMU is bolted down onto the bottom deck of the PRWP with four screws on its case. For the mechanical drawing, refer to the LN-200 manual[3].

2.3.2 Power Requirement

The IMU requires three separate voltage inputs of +5, +15, and -15 volts with a 5 percent tolerance on each line. Also the +15 volts have to be supplied last when activating the LN-200, and disconnected first when deactivating the LN-200. If not properly activated and deactivated, the LN-200 may suffer electrical damage.

Rather than fabricating the circuit, a decision was made to buy board-mount DC/DC converters because the converters are professionally manufactured and therefore more reliable than self-made circuits. The IMU circuit schematic is shown below in Figure 2.6.

[2]http://www.northropgrumman.com/Capabilities/LN200FOG/Documents/ln200s.pdf
[3]Northrop Grumman Corporations proprietary document

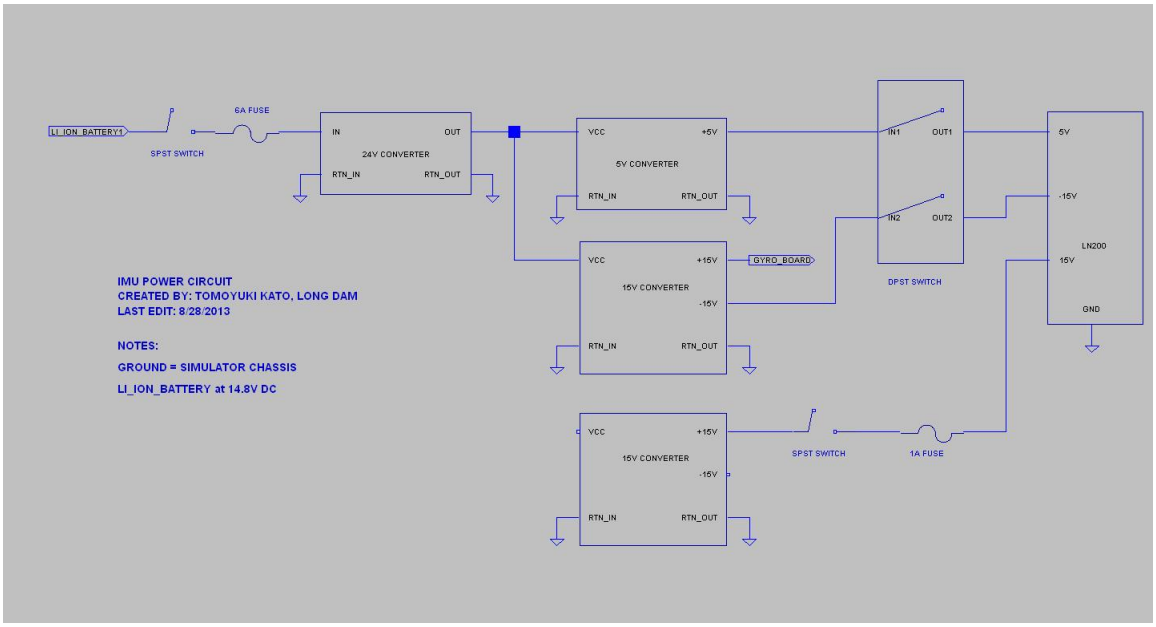

Figure 2.6: IMU Voltage Regulator Circuit.

There are three DC/DC converters: 24V, $\pm 15V$, and a 5V converter as well as a linear +15V regulator. The 24V converter supplies power to the $\pm 15V$ and the 5V converter since they are rated to work best with an input voltage of at least 24V. Although there already exists a 24V battery onboard the PRWP (the lead-acid batteries), a decision was made instead to use a separate battery and boost the input voltage to 24V using the DC/DC converter. The reason is because the lead-acid batteries are specifically for high-current load such as the reaction wheel system and should not be mixed with the low-current circuitry of the voltage regulator circuit. The 5V and the $\pm 15V$ supply power to the +5V and the -15V input for the IMU, and the linear +15V regulator supplies power to the +15V input of the IMU. In order to validate the fulfillment of the IMU power requirement, the IMU's internal voltage data can be accessed. The IMU's internal voltage reading plots are shown in Figure 2.7.

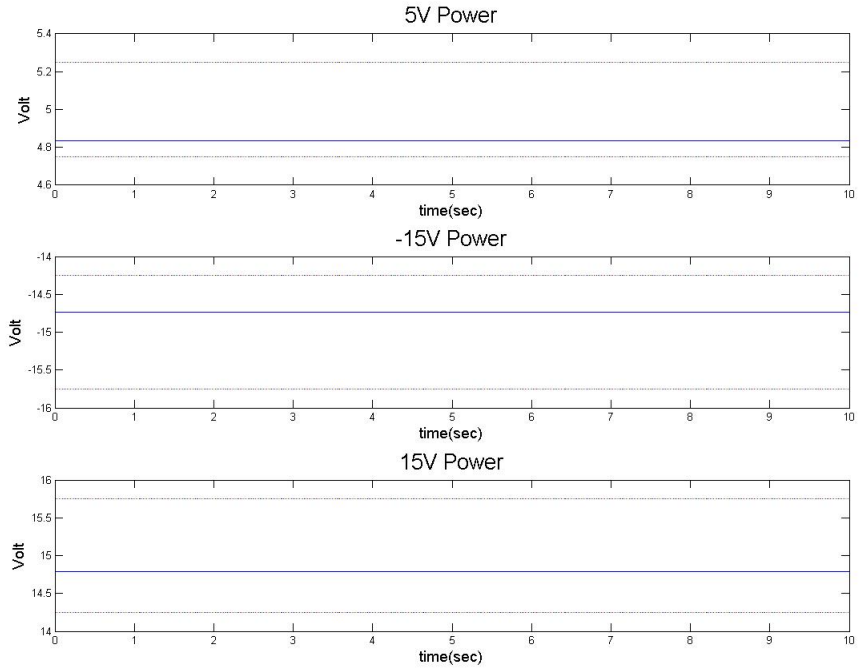

Figure 2.7: IMU Voltage Plot over 60 seconds.

For each of the three plots the dotted lines indicate the ±5% voltage tolerance. The plots clearly show that the IMU voltage falls well within the specification ratings.

2.3.3 Data Requirement

The LN-200 outputs information in the form of a RS-485 signal, and it uses an encoding protocol called synchronous data link control (SDLC). Therefore the Commtech Fastcom ESCC-104 device[4] was chosen to acquire data from the LN-200. The source code was written in the C language, and compiled in Simulink to create an S-function. The source code can be examined in detail in Appendix A of this document.

At a rate of 400Hz, the IMU sends out a multi-word stream, and each word consists of 16 bits. However only three words are required, which are the gyroscope data. The data acquisition software also checks specific bits of the data that corresponds to hardware malfunctions, and will transmit a warning if the IMU detects an anomaly. An example of an IMU test is shown in Figure 2.8.

[4]http://www.commtech-fastcom.com/DataSheets/FastcomESCC104.html

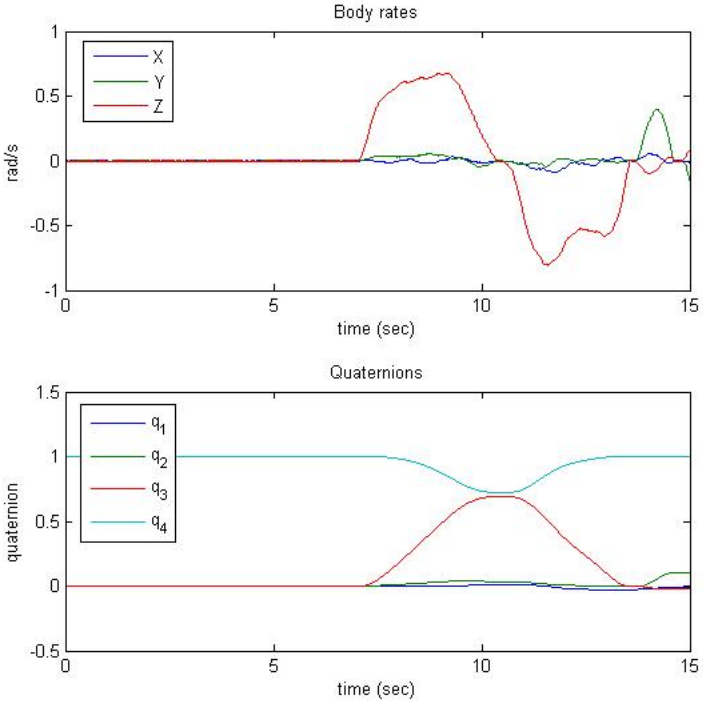

Figure 2.8: IMU Data Plot.

Figure 2.8 shows a 90 degree rotation about the IMU vertical axis. The gyroscope data is converted to angular velocity data, and then the body rate is used to determine the current quaternion rate. Then the quaternion rate is numerically integrated to determine the current quaternion trajectory.

Chapter 3

PRWP Dynamics and Control Law

This section will briefly cover the governing equations of spacecraft attitude dynamics as well as the control laws for attitude control. Section 3.1 will cover the dynamics and kinematics of spacecraft attitude motion. Section 3.2 will describe the various control laws that will be utilized for this project. Section 3.3. will briefly describe the system identification (ID) algorithm and its application for the PRWP.

3.1 Overview of Dynamics and Kinematics

To develop a framework for analyzing the attitude dynamics of a rigid-body spacecraft, we must be aware of the different coordinate systems involved as well as the relationship between the dynamics and kinematics. A free-body diagram of the PRWP is shown in Figure 3.1[21].

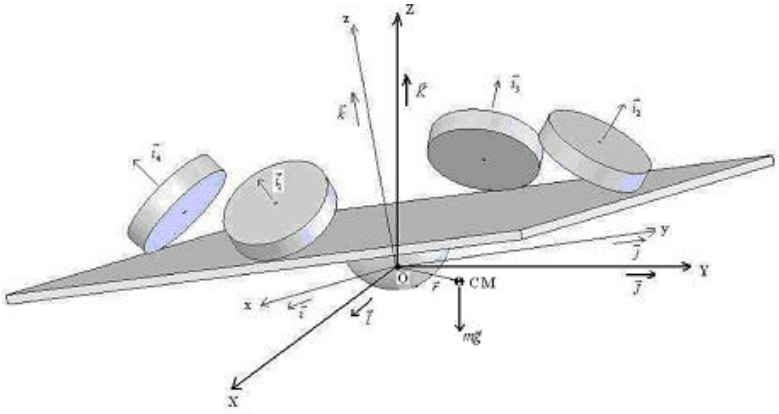

Figure 3.1: PRWP Free Body Diagram.

There are three coordinate frames of interest; the first one is the inertial frame, shown with the axes X, Y, and Z on the above figure. The second frame is the body frame, made up of the axes x, y, and z. The body frame stays attached to the simulator, and the motion of the body frame is measured relative to the inertial frame. The third frame is the reaction wheel frame, shown by the axis i_1 through i_4. When performing analysis on the motion of the PRWP, it is crucial to represent all vectors in the body frame axis. Figure 3.1 also allows us to derive the transformation matrices between the different frames.

Next the dynamics between the simulator body and the reaction wheels must be understood. The following expression is the equation of motion for a rigid-body spacecraft with four reaction wheels:

$$r \times R_E \cdot mg = J\dot{\vec{\omega}} + \vec{\omega} \times J\vec{\omega} + \sum_{l=1}^{4} \left(R_{w_l} \begin{bmatrix} I_{wl} \cdot \dot{\omega}_{wl} \\ 0 \\ 0 \end{bmatrix} + \vec{\omega} \times R_{w_l} \begin{bmatrix} I_{wl} \cdot \omega_{wl} \\ 0 \\ 0 \end{bmatrix} \right) \tag{3.1}$$

Although Eq. 3.1 has been derived by various others[27][28] including Healy[29], it is more generally understood as the Euler's equation of motion. Euler's equation of motion, as derived in various sources[30][31][32], states that for a rigid-body object the net external torque acting on it is expressed as the net rate of change of the angular momentum with respect to time. In the context of a rigid-body spacecraft with reaction wheels, the net change of angular momentum is the difference between the change in angular momentum of both the spacecraft itself and the reaction wheels.

The left term of Eq. 3.1 is an external torque resulting from a center-of-gravity (CG) offset. The right term consists of the torque on the simulator body and the torque from the four reaction-wheels. Therefore CG alignment is important for ABSS in order to minimize external torque on the system which has to be compensated for. Ideally the CG should be aligned with the air-bearing center of rotation, which results in the CG vector being zero, which makes the expression on the left side of Eq. 3.1 zero. When the CG torque term is zero, Eq. 3.1 can be reduced to:

$$J\dot{\vec{\omega}} + \vec{\omega} \times J\vec{\omega} + \sum_{l=1}^{4} \left(R_{w_l} \begin{bmatrix} I_{wl} \cdot \dot{\omega}_{wl} \\ 0 \\ 0 \end{bmatrix} + \vec{\omega} \times R_{w_l} \begin{bmatrix} I_{wl} \cdot \omega_{wl} \\ 0 \\ 0 \end{bmatrix} \right) = 0 \tag{3.2}$$

Eq. 3.2 states that the rate of change of the angular momentum of the simulator body is equal

and opposite to the rate of change of the reaction wheel's angular momentum, assuming no external torque acting on the system. Next the quaternion kinematic equation is introduced:

$$\begin{aligned} \dot{\vec{q}} &= \frac{1}{2}(q_4\vec{\omega} - \vec{\omega} \times \vec{q}) \\ \dot{q_4} &= -\frac{1}{2}\vec{\omega}^T\vec{q} \end{aligned} \tag{3.3}$$

Eq. 3.3 have been derived by several sources for the analysis and simulation of rigid-bodies[33][34][35]. Quaternion kinematics build a framework for the analysis of the PRWP's angular position and velocity. In the context of spacecraft attitude control, quaternions are preferred for spacecraft kinematics because it has advantages over Euler angle kinematics. Unlike Euler angles, quaternions do not suffer from singularities. The expression for Euler angles involves trigonometric terms in the denominator while quaternion kinematics do not deal with denominator terms. Also quaternions tend to be more computationally efficient compared to Euler angles due to the quaternion's lack of trigonometric terms.[28]

Based on the body rate data which is provided by the LN-200 IMU, the quaternion rates can be determined. The quaternion rates are numerically integrated to give the rotational position of the PRWP expressed as quaternions. The combination of the equation of motion in Eq. 3.1 and quaternion kinematics in Eq. 3.3 builds a framework for analyzing the rotational motion and dynamics of the PRWP. However for this project it is insufficient to analyze the dynamics and kinematics of the PRWP, as the purpose of the PRWP is to verify control laws applied to spacecraft attitude control. In the context of control system theory the PRWP is a dynamic system in which the output of interest is its angular position. Therefore control laws must be developed in order to guarantee a specific behavior of the PRWP angular position. The following section will describe the design of the control laws to be tested on the PRWP.

3.2 Control Laws

3.2.1 Full-State Feedback

Full-State Feedback (FSFB) is a control algorithm which generates a plant input by feeding back the plant states. A control-loop diagram is shown in the next page as Figure 3.2.

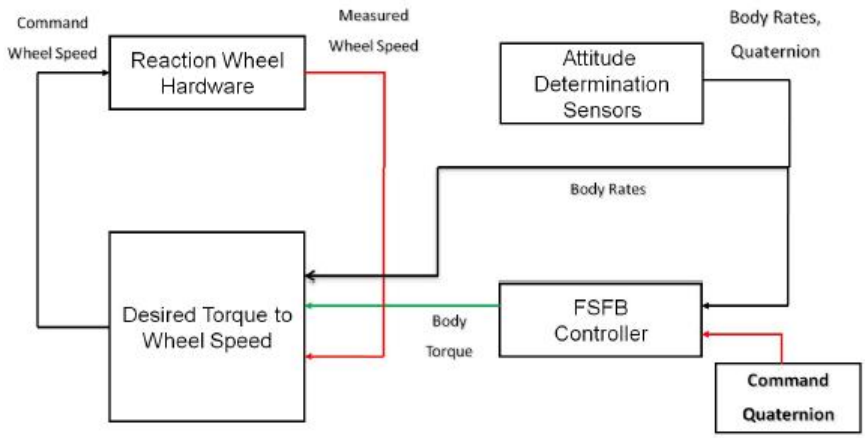

Figure 3.2: Full-State Feedback Control.

The control law assumes that the plant dynamics are well known and that all the states of the system are accessible. While FSFB has the advantage to full system state access, such condition is not always realistic and also the controller is dependent upon the knowledge of the plant model. Wie[28] derived a particular expression of the FSFB such that the quaternion error is included:

$$u = -K_{FSFB} \cdot \vec{q}_e - C_{FSFB} \cdot \vec{\omega} \tag{3.4}$$

The K_{FSFB} and C_{FSFB} terms in Eq. 3.4 are the gain matrices. The FSFB controller in Eq. 3.4 have been proven to be asymptotically stable in Lyapunov's definition for specific K_{FSFB} and C_{FSFB} gains as described by Wie. However Mittlestead[19] derived a different expression of the K_{FSFB} and C_{FSFB} gains:

$$K_{FSFB} = 2\omega_n^2 \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \tag{3.5}$$

$$C_{FSFB} = 2\xi\omega_n \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \tag{3.6}$$

The K_{FSFB} and C_{FSFB} gains are functions of the natural frequency, the damping ratio, and the PRWP inertia tensor. The reason is that the control law assumes the PRWP to be an underdamped

second-order oscillator, and the gains allow the selection of the system poles based on the desired natural frequency and damping ratio. The natural frequency, for an underdamped second-order system, is given as a function of the settling time, settling percentage, and the damping ratio[36].

$$\omega_n = \frac{-\log(percent)}{t_s * \xi} \tag{3.7}$$

In Eq. 3.7 t_s is defined as the settling time, *percent* is defined as the settling percentage, and ξ is defined as the damping ratio. As described in the next chapter, the system settling time and settling percentage will be the criteria used to evaluate controller performance. Therefore Mittlestead chose to derive the K_{FSFB} and C_{FSFB} gains such that the gain selection is dependent upon the desired settling time and percentage.

3.2.2 Linear Quadratic Regulator and Set-Point Control

The Linear Quadratic Regulator (LQR) is a type of a linear optimal controller[37][11]. The feedback controller algorithm is given,

$$u = -Kx \tag{3.8}$$

The LQR gain K in Eq. 3.8 is constructed such that the cost function of the following form is minimized,

$$J = \int_0^t (x^T Q x + u^T R u) dt \tag{3.9}$$

The LQR gain K is solved for by the following expression,

$$K = -R^{-1} B^T P(t) \tag{3.10}$$

The $P(t)$ term is solved through the algebraic Ricatti equation,

$$A^T P(t) + P(t) A - P(t) B R^{-1} B^T P(t) + Q = -\dot{P(t)} \tag{3.11}$$

Starting with the A and B matrices of the state-space representation of the linearized plant and the Q and R gains of our choosing, the $P(t)$ term in the algebraic Ricatti equation is solved to determine the LQR gain K. One condition for determining K is that the plant must be fully controllable and observable. Otherwise a solution for the K gain may not exist.

However for an application in which the reference input to the plant is nonzero, the LQR control law may not be best suited for converging the plant output to the reference input quickly or the plant output may not even converge at all to the nonzero reference input. The set-point controller is designed to make the plant output, either a specific output or all the output, converge to a reference input as fast as possible. Set-point is particularly designed to converge a plant output to nonzero reference input, in other words a set-point (hence the name of the controller). Set-point accomplishes output control by applying two feedforward gains to the reference input. The controller diagram is shown in Figure 3.3[37].

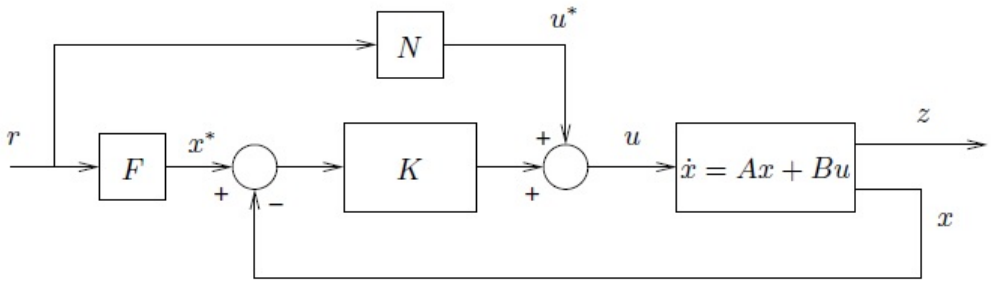

Figure 3.3: LQR Control Combined with Set-Point Control.

In Figure 3.3 the signal labeled z coming out of the plant is the specific plant output to be controlled by the set-point controller. The design of the set-point controller assumes a linearized plant model shown as,

$$\dot{\vec{x}} = A\vec{x} + B\vec{u} \qquad y(t) = C\vec{x} \qquad z(t) = G\vec{x} + H\vec{u} \tag{3.12}$$

In Eq. 3.12 the variable z is known as the controlled output and can either be specific output(s) of $y(t)$ or be equal to $y(t)$. For the PRWP system the state vector is given as,

$$\vec{x} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{3.13}$$

The plant input term $\vec{u}$ is the input moment exerted on the PRWP expressed as,

$$\vec{u} = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix} \tag{3.14}$$

M_1, M_2, and M_3 are the input moment terms in the three body axis. The state vector is a six-element vector consisting of the three-axis body rate and the three-element quaternion vector. Although there are six plant states, the set-point controller will only regulate the three-element quaternion vector. Therefore the $z(t)$ expression becomes,

$$z(t) = y_1 = [C_1\vec{x}] = [C_1]\,\vec{x} + [0]\,\vec{u} \tag{3.15}$$

In Eq. 3.15 y_1 is the particular output of $y(t)$ such that it only contains the quaternion vector terms. Therefore C_1 is the output matrix with dimensions such that when multiplied with the plant states only the three-element quaternion vector remains. The G and H matrices from Eq. 3.12 becomes C_1 and zero respectively. Therefore the expression of the G and H matrices becomes,

$$G = \begin{bmatrix} 0 & 0 & 0 & G_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & G_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & G_3 \end{bmatrix} \qquad H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.16}$$

The terms G_1, G_2, and G_3 are variable terms of the G matrix. They multiply with the desired plant outputs, which is the quaternion vector terms q_1, q_2, and q_3 respectively. Hespanha, by combining the LQR controller and the set-point controller, derives the expression of the control input as follows,

$$u = -K(x - x^*) + u^* \tag{3.17}$$

The terms x^* and u^*, defined as the equilibrium state and equilibrium input respectively, are defined as follows,

$$x^* = FU_r$$
$$u^* = NU_r$$

The F and N feed-forward gains are determined by inverting the P matrix which is defined,

$$P = \begin{bmatrix} A & B \\ G & H \end{bmatrix} \tag{3.18}$$

The A and B matrices are from the state-space representation of the linearized plant. The G and H matrices are from the expression of the controlled output $z(t)$. Once the P matrix is inverted, the F and N gains both have the column dimension equal to the dimension of the desired plant output. The row for the F gain is the dimension of the plant state, and the dimension of the N gain row is the dimension of the reference input. In practice, the set-point controller is adjusted through the G matrix. The G matrix will change the F and N gain, which in turn changes the weighting of the reference input for the entire control law.

3.2.3 Nonlinear Direct Model Reference Adaptive Control

NDMRAC is a form of adaptive control, originally developed by Mehiel[38] and modified by Torres[39]. The property that sets NDMRAC apart from a fixed-gain control law such as FSFB is that the control gains are variable and will adjust according to changes in the system parameters. Also the NDMRAC requires minimal knowledge of the plant model, unlike the FSFB or the LQR controllers. Given the correct initial conditions, the NDMRAC can offer significant improvements in control performance over fixed-gain controllers such as the FSFB. The control diagram for the NDMRAC is shown in Figure 3.4[21].

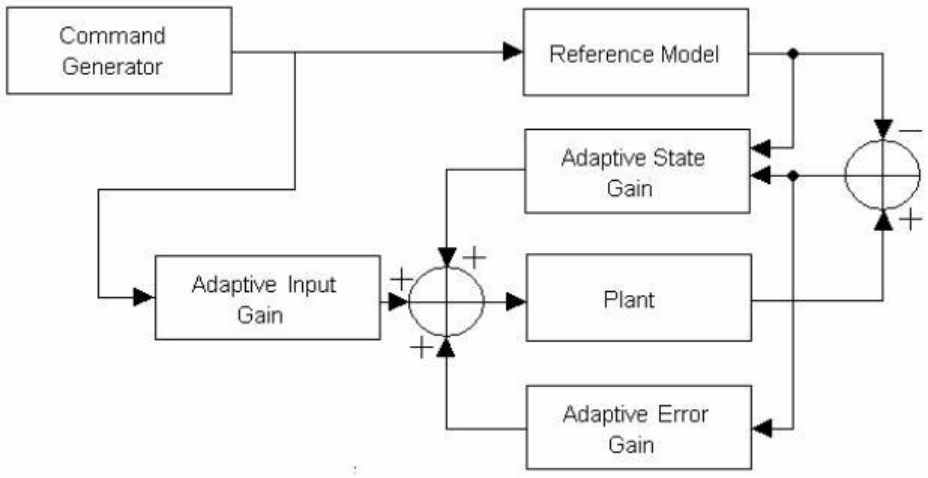

Figure 3.4: Nonlinear Direct Model Reference Adaptive Controller Diagram.

NDMRAC manipulates the plant output such that the plant is forced to track a reference model of equal or lower order. The output error is expressed as:

$$e_y = y_p - y_m \tag{3.19}$$

The output error is the difference between the plant output y_p and the model output y_m. Proven to be asymptotically stable by Lyapunov's definition, the NDMRAC seeks to reduce e_y to zero. NDMRAC does so by taking the model state x_m, model input u_m, and the output error e_y and applying the following control law,

$$u = G_e e_y + S_{21} x_m + S_{22} u_m \tag{3.20}$$

The terms G_e, S_{21}, and S_{22} are the error, input, and state gains respectively. Moreover the adaptive gain rates are defined as:

$$\begin{aligned} \dot{S}_{21} &= -e_y x_m^T H_1 \\ \dot{S}_{22} &= -e_y u_m^T H_2 \\ \dot{G}_e &= -e_y e_y^T H_3 \end{aligned} \tag{3.21}$$

The gain rates are numerically integrated to solve for the adaptive gains according to Eq. 3.20. The H_1, H_2, and H_3 are known as the adaptive parameters. The adaptive parameters must be positive-definite matrices in order for NDMRAC to work. As the control simulation is being executed, the adaptive gains will change depending on the output error and if the controller works correctly, the gains will settle to a certain value.

Currently there are no reliable methods for determining the adaptive gain initial conditions and the adaptive parameters, and therefore several iterations of the control loop must be performed to decide on acceptable values. This process will be explained in further detail in Chapter 4. Although Patel[40] has attempted a method using an optimization command in MATLAB and expressing the adaptive parameters as a cost function to determine the initial gains and adaptive parameters, the results lacked consistency and therefore the method was not reliable.

The reference model for the adaptive control laws is shown in Figure 3.5.

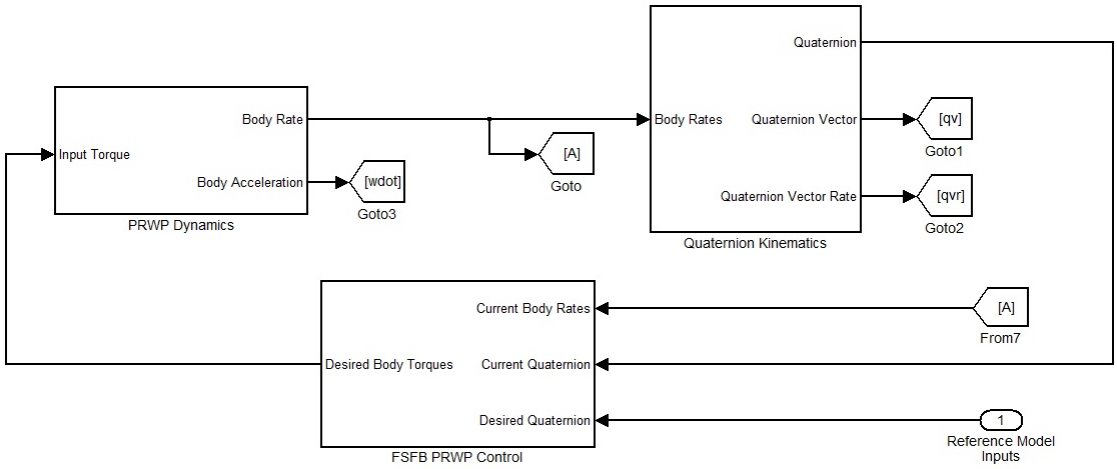

Figure 3.5: Adaptive Controller Reference Model.

The reference model is a purely numerical simulation of the PRWP motion using the FSFB controller. The reference model of Figure 3.5 was chosen because it best represents the ideal trajectory that the plant should follow. The reference model assumes no external torque and that the body torque command from the FSFB controller is exactly the plant input instead of going through a reaction wheel system first. Also the FSFB gains are selected such that the motion test performance requirements are satisfied. The reference model algorithm will be explained further in Chapter 4.

3.2.4 Adaptive Output Feedback

Adaptive Output Feedback (AOF) is a simplified version of the NDMRAC. While the NDMRAC utilizes the output error, model state, and model input for its control input, the AOF only utilizes the error output for its control input. The AOF control law was developed by Patel[40], and the control diagram is shown in Figure 3.6[40].

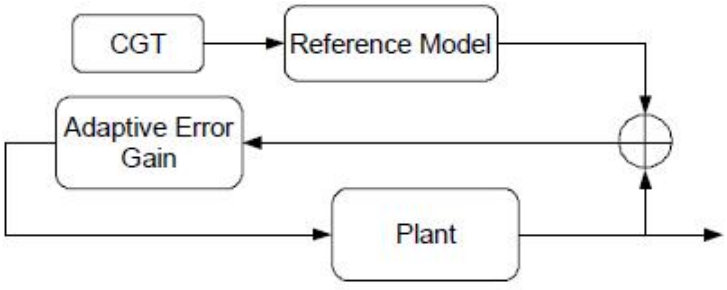

Figure 3.6: Adaptive Output Feedback Diagram.

Similar to the NDMRAC, the FSFB control law in Figure 3.5 is used as the reference model, which the plant is to track. The AOF control law has been shown in simulation to outperform NDMRAC in certain cases, and AOF is also less expensive computationally since there is only one adaptive gain as opposed to three. Although the control algorithm is slightly different than the NDMRAC, the AOF controller has also been proven to be asymptotically stable through Lyapunov's definition.

3.3 System Identification

Although the dynamics, kinematics, and the control laws of the PRWP system are important regarding the analysis of the PRWP motion, there are still two outstanding issues that must be resolved in order to operate the PRWP system. The first issue is that the mass property of the PRWP must be determined. In order to design the controllers, the fixed-gain controllers in particular, the PRWP system must be well understood. A poor estimation of the PRWP inertia tensor can lead to bad controller design which in turn will have poor control performance.

The second issue is that the PRWP CG must be balanced as close as possible in order to minimize the CG offset torque. The CG offset torque will constantly act on the PRWP throughout the motion test, and therefore the control law and the reaction wheel must expend additional effort to compensate for it. This is problematic because the additional torque necessary to counteract the CG offset torque will cause the reaction wheels to reach its wheel speed saturation limit faster which in turn will limit the motion range of the PRWP.

One method to address both issues is the system identification algorithm. The system identification algorithm is a well known method to determining an empirical model of a dynamic system based on its input and output data[41][42][43]. There are many variations of the system ID algorithm, and Healy[29] develops an algorithm specific to the PRWP using the least-squares method of the system ID algorithm. For the rest of this thesis a distinction is made between the terms system ID algorithm and the PRWP system ID algorithm which is derived by Healy. For the sake of brevity the PRWP system ID algorithm will not be discussed in detail and the reader is urged to refer to Healy[29] for a complete derivation of the PRWP system ID algorithm. Through a separate motion test in which the PRWP undergoes oscillatory motions about all three of its body axis, the PRWP system ID algorithm uses the test data to estimate both the PRWP inertia tensor and the CG vector. For more information regarding the test refer to Dam[22].

Although the PRWP inertia can also be determined using solid-modeling software, the solid-modeling program determines mass property information based on user-supplied numerical data as

opposed to experimental data. In the context of this project it is of interest how utilizing different estimates of the PRWP inertia will affect the control law performance. This will be discussed further in the next chapter.

The PRWP system ID algorithm is also crucial in the proper balancing of the PRWP. Complemented with the FMBs, the PRWP system ID algorithm determines both the PRWP CG location and the CG correction necessary to balance the simulator. This completes the derivation of all necessary concepts necessary for the analysis and operation of the PRWP system.

Chapter 4

PRWP Motion Tests

This chapter will describe the motion test experiments that will be performed on the PRWP. The purpose of the motion tests are to determine if the PRWP Mk.III configuration can function as a reliable test apparatus for robust control law verification. Section 4.1 will describe the motion test parameters and maneuver. Section 4.2 will describe the three different motion test cases designed to determine control law robustness. Section 4.3 will discuss the initialization of the control law parameters.

4.1 Motion Test Description

To evaluate the performance of the different control laws in the different parameter cases, control performance requirements have to be identified. The motion test performance requirements are defined as,

1. The settling time, t_s, is set to 20 seconds.

2. The settling percentage is set to 1 percent.

For the PRWP system the control laws are applied to the attitude control motion tests, so the plant output to be controlled is the PRWP's angular position expressed as quaternions. Unlike the Mk. II configuration motion tests, the Mk. III motion tests will follow a different quaternion trajectory. The Mk.II configuration started with an initial quaternion of [0, 0, 0, 1] and moved to [0.145, 0.111, 0.145, 0.972] which is equivalent to an 18 degree rotation in the roll and yaw axis, and a 10 degree rotation in the pitch axis. The Mk.III motion test will maneuver from an initial quaternion of [0, 0, 0, 1] to [0, 0, 0.1305, 0.9914] which is equivalent to a 15 degree counter-clockwise

rotation about the yaw axis. The motion test maneuver had to be adjusted because of the Mk.III's limitations in executing roll and pitch axis maneuvers which will be discussed further in Chapter 5.

The motion test is 40 seconds long, and the PRWP will initiate rotation 5 seconds into the simulation. To satisfy the expected control performance, the PRWP has to reach its destination no later than the 25 second-mark of the motion test with a steady-state error of no more than 1% of the command quaternion destination.

4.2 Motion Test Cases

For the purpose of control law validation it is not enough to confirm the performance of a control law through one test. It is also of interest to determine the robustness of the controller, which is a measure of how well the control law can perform under variations of the plant model. For this project three test cases will be utilized to determine the robustness of each of the four control laws. Therefore there will be twelve tests total performed for this project since each control law will be tested through motion tests in three separate test cases.

The first test case is referred to as the system identification inertia estimate case. The PRWP system ID algorithm is utilized to estimate the inertia tensor and the CG vector, and the system states from separate motion tests are required. Although the PRWP system ID algorithm takes into account the nonlinear effects acting on the PRWP (e.g. measurement noise and external torque), the algorithm is still a model and by nature cannot perfectly determine the exact inertia of the simulator. Therefore to validate the inertia tensor estimation, multiple trials are required to determine the inertia tensor within a specified confidence interval. Through Dam's tests the system ID inertia was determined to be, within a 95% confidence interval,

$$I_{sysid} = \begin{bmatrix} 0.6071 & 0.0266 & 0.0149 \\ 0.0266 & 0.6560 & 0.001 \\ 0.0149 & 0.001 & 0.6376 \end{bmatrix} \tag{4.1}$$

The second test case is referred to as the solid-model inertia estimate case. The PRWP inertia tensor will be estimated using a solid-model of the PRWP. The solid model is shown below in Figure 4.1.

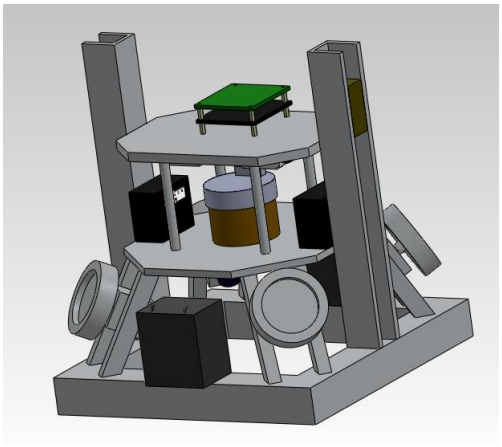

Figure 4.1: Solid Model of the PRWP.

Since the solid-model estimate is not based on physical test data, the solid-model inertia estimate is expected to vary significantly from the system ID estimate. Therefore using both the system ID and the solid-model inertia estimates in the motion tests may reveal how the different control laws perform with varying mass property data. The result of interest from the motion tests is to determine the robustness of the control laws. As defined in Chapter 1, the robustness of a controller is a measure of how well the control law performs under variations of the plant model. A robust control law should be able to satisfy expected control performances even when there are uncertainties in the parameters of the plant model. In the context of this project, it is of interest to determine if the control laws can satisfy control performance even with varying inertia tensor estimates. A control law with low robustness will experience control performance issues (e.g. higher overshoot, longer settling time, higher steady-state error) while a more robust control law will consistently meet expected control performance despite uncertainty in the plant model.

The inertia tensor determined by the SolidWorks[1] program is:

$$I_{est} = \begin{bmatrix} 0.5485 & -0.0002 & -0.0003 \\ -0.0002 & 0.5884 & -0.0005 \\ -0.0003 & -0.0005 & 0.6456 \end{bmatrix} \tag{4.2}$$

The differences between the solid-model and the system ID inertia estimates are the moment of inertia (diagonal terms) and the products of inertia. The solid-model estimate, except for the principal z-axis, has a lower moment of inertia. Also the product of inertia terms are much higher for the system ID estimate compared to the solid-model estimate. While the solid-model of the

[1] www.solidworks.com

PRWP modeled much of the large components on the physical PRWP system, there are still many components that were not modeled such as cables and the three FMBs which may explain why the solid-model estimate is lower in magnitude than the system ID estimate. Also the solid-model makes assumptions about the material property of the various components of the PRWP, which affects the density and mass data. Therefore the mass data may vary from the actual real-world value, and this may also account for discrepancies between the system ID and the solid-model inertia tensor estimates.

The third test case is called the degraded wheel case. The solid-model inertia estimate will again be used, but this time a reaction wheel malfunction will be simulated. To do so, the fourth reaction wheel, which lies on the -Y body axis, will be modeled as an overdamped second-order oscillator. The differential equation and its state-space representation is shown below.

$$\ddot{\Omega} + 2\xi\omega_n\dot{\Omega} + \omega_n\Omega = \omega_n^2 u \tag{4.3}$$

$$A = \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\xi\omega_n \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ \omega_n^2 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad D = [0] \tag{4.4}$$

The damping ratio will be raised to 1.1 and the natural frequency is set at $1\frac{rad}{sec}$. An overdamped system will operate considerably slower than an underdamped system which is how the wheels behave nominally. The purpose of this test case is to determine how well the controller can perform when actuator dynamics uncertainty is introduced.

4.3 Controller Setup

Now that the motion test has been described, the controller setup must be explained. To complete the design of the control laws, the control law algorithm must be designed according to the PRWP plant model. The FSFB controller was set up by selecting the K_{FSFB} and C_{FSFB} gains based on the desired damping ratio and natural frequency, in which the latter is a function of the settling time and settling percentage. The damping ratio and the natural frequency were chosen as,

$$\xi = 0.85$$

$$\omega_n = \frac{-\log(0.01)}{20 * 0.85} = 0.2709\frac{rad}{sec}$$

Therefore the K_{FSFB} and C_{FSFB} gains are,

$$K_{FSFB} = 0.1468I \tag{4.5}$$

$$C_{FSFB} = 0.4605I \tag{4.6}$$

I is the controller's inertia tensor, which could either be the system ID estimate or the solid-model estimate depending on the desired test case.

The LQR gain was determined by linearizing the plant around the desired equilibrium point, and then using the 'lqr' command in MATLAB to compute the LQR gain. The 'lqr' command needs the A and B state-space matrices as well as the positive-definite Q and R matrices. The Q and R matrices are currently set as identity matrices as shown:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The equilibrium point of which the plant was linearized about is,

$$\omega_e = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \qquad \vec{q}_e = \begin{bmatrix} 0 & 0 & 0.1305 \end{bmatrix} \qquad q_{4e} = 0.9914$$

At the equilibrium point the PRWP should be stationary (all zero angular-velocities) and should have rotated 15 degrees from its initial angular position about the z-axis. The state-space A and B matrices of the plant are,

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2}q_{4e} & -\frac{1}{2}q_{3e} & 0 & 0 & 0 \\ \frac{1}{2}q_{4e} & \frac{1}{2}q_{3e} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}q_{4e} & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} \frac{1}{I_{11}} & 0 & 0 \\ 0 & \frac{1}{I_{22}} & 0 \\ 0 & 0 & \frac{1}{I_{33}} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

q_{3e} and q_{4e} are the third vector term and the scalar term of the equilibrium quaternion which the plant is linearized about respectively. I_{11}, I_{22}, and I_{33} are the first, second, and third diagonal terms

of the PRWP inertia tensor respectively. The model state vector is a six-element vector consisting of the body rate and the quaternion vector, and the reference input U_r is a three-element vector consisting of the desired quaternion vector. Regarding the motion test cases, changing the inertia tensor estimate will affect the A and B state-space matrices, which will then affect the value of the LQRSP controller gains.

The set-point controller was designed by combining the A and B state-space matrices with the G and H gains set as:

$$G = \begin{bmatrix} 0 & 0 & 0 & G_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & G_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & G_3 \end{bmatrix} \qquad H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For the PRWP motion tests, G_1 and G_2 are set to 1, and G_3 varies in order to adjust the F and N gains. The G_3 term multiplies with the third quaternion vector term, which is the yaw axis term, and therefore adjusting G_3 would allow for the adjustment of the set-point controller output torque on the yaw axis.

The setup of the NDMRAC and the AOF controllers will now be discussed. Between the motion test cases, only the reference model inertia tensor changes. As mentioned in Chapter 3, the reference model is a numerical simulation of the PRWP using the FSFB controller. The setup is the same as the FSFB setup described earlier in this section. The FSFB controller gains selection is based on the settling time and the steady-state error requirements as well as the desired inertia tensor estimate, and is equivalent to the expressions in Eq. 4.5 and Eq. 4.6. The plant model is nonlinear but assumes no external torque, and the model is the same as Eq. 3.2.

Figure 4.2 shows the reference model quaternion vector output for the system ID inertia estimate.

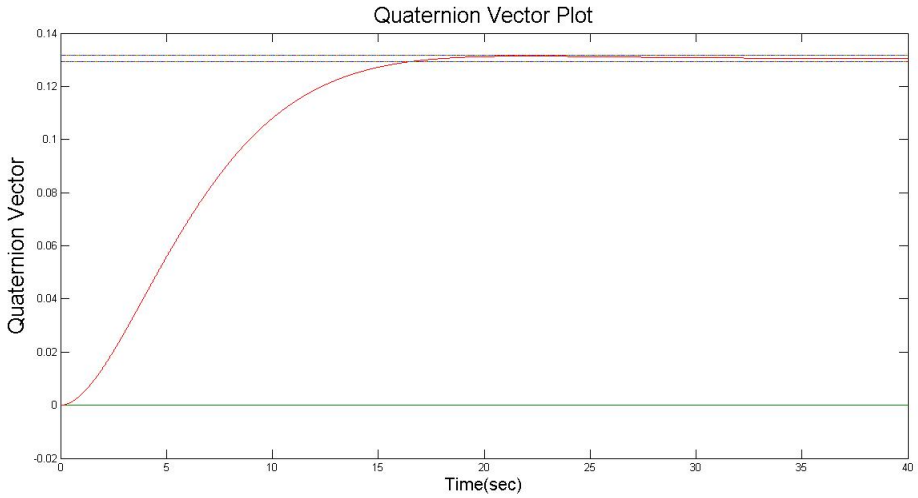

Figure 4.2: Reference Model Quaternion Vector, System ID Inertia Tensor Estimate.

Figure 4.3 shows the reference model quaternion vector output for the solid-model inertia estimate and the degraded wheel case (since the reference model does not model the reaction wheel system).

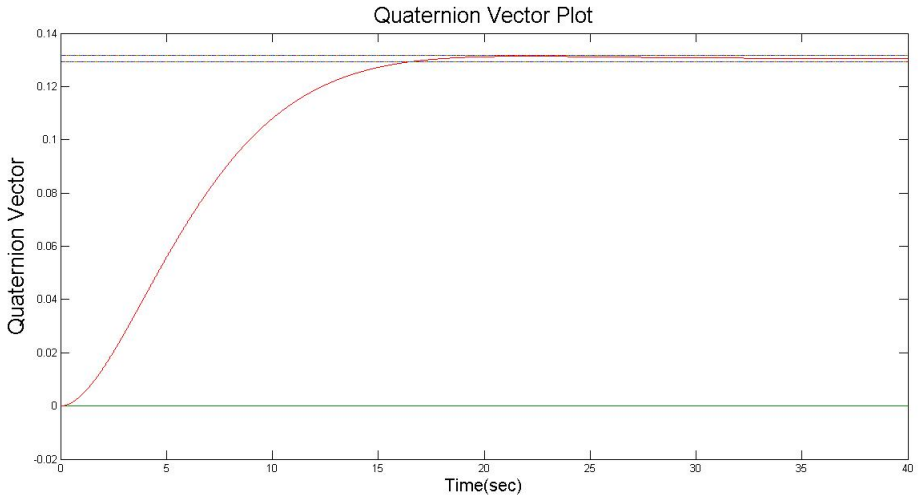

Figure 4.3: Reference Model Quaternion Vector, Solid-Model Inertia Tensor Estimate.

Both reference models show a maneuver of a 15 degree yaw-axis rotation, which is what the motion test will execute. The dotted lines indicate the ±1% steady-state error bound. Both Figures 4.2 and 4.3 are similar in showing that the reference model satisfies the settling time requirement of 20 seconds and the steady-state error requirement of 1% of steady-state value. During the motion tests the adaptive controllers will make the PRWP system track the outputs of Figure 4.2 or Figure 4.3 depending on the test case.

The adaptive gain initial conditions and the adaptive parameters need to be initialized. The initial gains were determined in the numerical simulations by running numerous iterations. The adaptive gains would start with an all-zero initial condition for the first iteration, and then the final value of the adaptive gains after several iterations would then become the initial conditions of the adaptive gains for the motion test. Adaptive control works by making the plant output track the output of a reference model, and this is accomplished by adjusting the adaptive gains as necessary. Although in theory there exists an ideal set of adaptive gains to make the plant track a reference model perfectly, there currently is no known method to reliably determine the ideal gains. Therefore the adaptive gain initial conditions must be determined through several iterations of numerical simulations until the adaptive gains do not experience significant change in magnitude. Then the final values of the adaptive gains from the numerical simulations are used as the adaptive gain initial conditions for the motion tests.

The adaptive parameters are also determined through numerical simulations. So far the process of choosing the adaptive parameters is also iterative, and therefore various combinations are tested in a numerical simulation to see which adaptive paramter matrices yield the best results. Higher adaptive parameters enable the adaptive gains to respond faster to the plant error but it also increases the risk of instability due to higher gain values. Lower adaptive parameters are less likely to suffer instability, but comes at the sacrifice of controller response time. The adaptive parameters used for the NDMRAC are,

1. Adaptive Parameter $H_1 = 10{*}Id_{6\times 12}$
2. Adaptive Parameter $H_2 = 0.1{*}Id_{6\times 4}$
3. Adaptive Parameter $H_3 = 100{*}Id_{6\times 6}$

The variable Id indicates an identity matrix with the dimensions detailed in the subscript terms. The AOF controller adaptive paramter is,

$$H_3 = 500 * Id_{6\times 6}$$

This concludes the explanation for the motion test setup.

Chapter 5

Motion Test Results

This chapter will discuss the results of the PRWP numerical simulations and motion tests. The performance of all 4 control laws (FSFB, LQRSP, NDMRAC, and AOF) will be evaluated for all three cases of the platform parameters as discussed in the previous chapter. Again the control law effectiveness is evaluated based on its ability to meet the settling time and steady-state error requirements for the quaternion vectors. Section 5.1 will summarize the results from the Mk. II motion tests. Section 5.2 will go over the results from the numerical simulations. Section 5.3 will describe the Mk.III motion test results with the LN-200 IMU onboard the PRWP.

5.1 Past Results: PRWP Mk.II Configuration

Downs and Kinnett have developed closed-loop attitude control tests for the PRWP. In this section Downs' motion test results will be reviewed, with all three test case results. Downs ran both the numerical simulations and motion tests for each parameter case, and compared both results. In all the test cases, the simulator was commanded to maintain a home quaternion coordinate of [0 0 0 1] for the first 20 seconds of the simulation, then commanded to rotate to the new coordinate of [0.145 0.111 0.145 0.978], which is equivalent to an 18 degree rotation in the roll and yaw axis, and a 10 degree rotation in the pitch axis. The motion tests were 70 seconds long.

	Controller	Results
Case I	FSFB	Did not settle within 1 percent
	NDMRAC	Settled within 1 percent, under 20 seconds
	AOF	Settled within 1 percent, under 20 seconds
Case II	FSFB	Did not settle within 1 percent
	NDMRAC	Settled within 1 percent, under 20 seconds
	AOF	Settled within 1 percent, under 20 seconds
Case III	FSFB	Did not settle within 1 percent
	NDMRAC	Settled within 1 percent, under 20 seconds, required adaptive parameter reduction
	AOF	Settled within 1 percent, under 20 seconds, required adaptive parameter reduction

Table 5.1: Summary of Mark II Motion Test Data.

The results of the Mk. II motion tests are summarized in Table 5.1. In all 3 parameter cases, the FSFB failed to meet the settling percentage requirement. The cause of the FSFB controller's failure to settle the quaternion vector was most likely a combination of the MEMS gyroscope drift after 70 seconds and the CG offset of the platform. The NDMRAC met the performance requirements for cases I and II, but the quaternion vector went unstable for case III. The AOF's results were similar to the NDMRAC's, as the controller went unstable for the third case. However both adaptive controllers was eventually able to stabilize the quaternion and meet the performance requirement after the adaptive parameter matrix for the error gain was reduced, and the third case tests were repeated.

Unfortunately the Mark II experimental results were insufficient to make a good comparison between the various control laws. Both case I and II performances were nearly identical, and therefore it was not clear if all three control laws performed better with the system ID iertia estimates as opposed to the solid-model inertia estimates. Also the adaptive controllers became unstable for the degraded wheel test case, and a change in adaptive parameter was necessary to stabilize the system. Even though the adaptive control laws were shown to consistenly outperform the FSFB controller, it became clear that a reduction in the attitude determination sensor bias drift was necessary to make stronger comparisons between the controllers under varying plant parameters.

5.2 Numerical Simulation Results

This section will discuss the results of the numerical simulations for all four control laws for each simulation case with the PRWP Mk.III as the plant. The numerical simulations does not take into account any external torque, and therefore assumes that the CG is perfectly balanced.

5.2.1 Numerical Simulations Case I: System ID Estimate

First the quaternion vectors from the system ID estimate case is shown below in Figure 5.1.

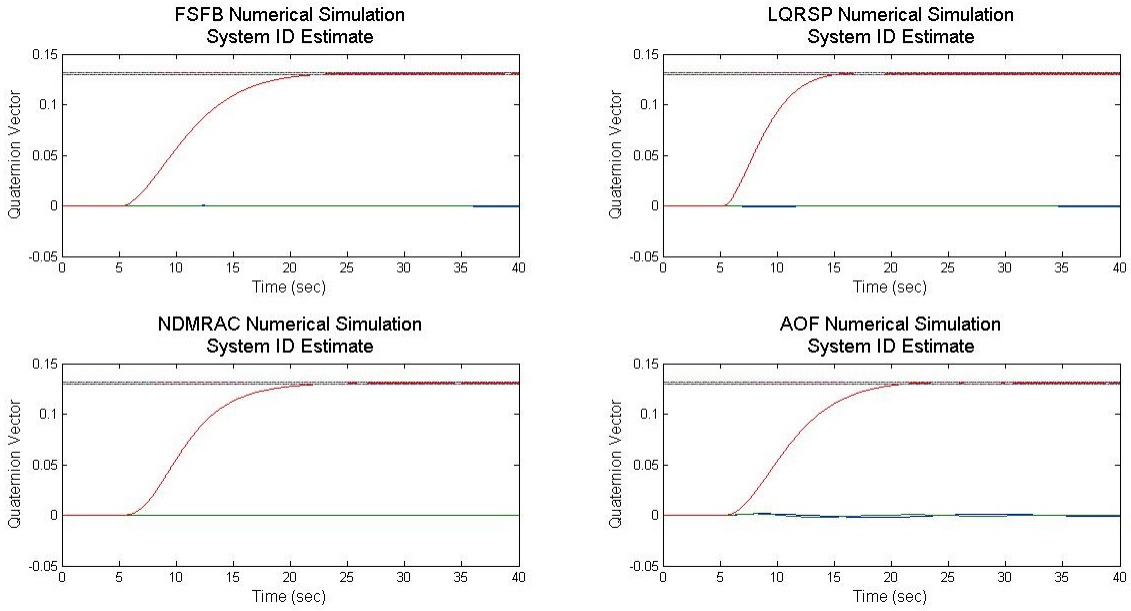

Figure 5.1: Numerical Simulation Quaternion Vector, System ID Estimate Case.

All four controllers met both the settling error and settling-time requirements. All four controllers settle around the 15 to 20 second mark, well before the 25 second mark which is the settling time limit. Shown in Figure 5.2 are the steady-state error plot for each of the four controllers.

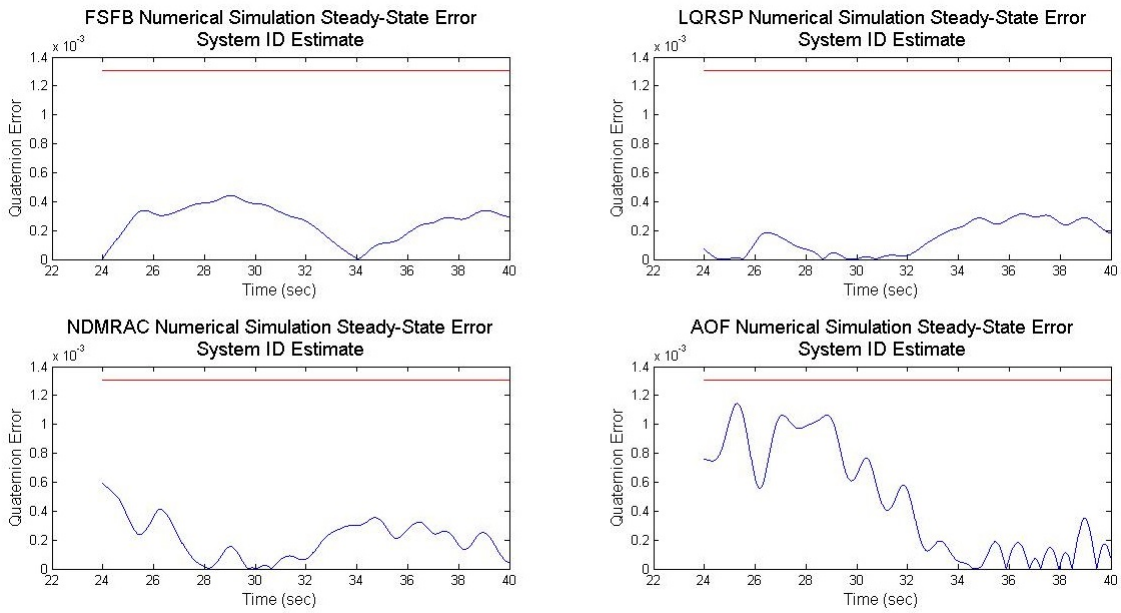

Figure 5.2: Numerical Simulation Steady-State Error, System ID Estimate Case.

The steady-state error is calculated as the absolute value of the difference between the command quaternion and the actual quaternion. The steady-state error plot begins on the 24 second mark, right before the settling-time mark of 25. The blue curve is the steady-state error, and the dashed line is the error bound of 1% of the steady-state quaternion value. If the respective controller met the steady-state error requirement, the steady-state error curve should stay under the steady-state bound line.

The FSFB, LQRSP, and the NDMRAC control laws all perform similarly with a near-zero steady-state error. However the AOF, though it eventually settles close to the command quaternion, experiences a significant amount of overshoot from the 24 to 32 second mark. This is a possible indication that the AOF controller has some difficulty in conforming the plant outputs to track the reference model.

5.2.2 Numerical Simulations Case II: Solid-Model Estimate

The numerical simulation results for the solid-model inertia estimate case will now be discussed. Shown in Figure 5.3 are the quaternion vectors for the four controllers.

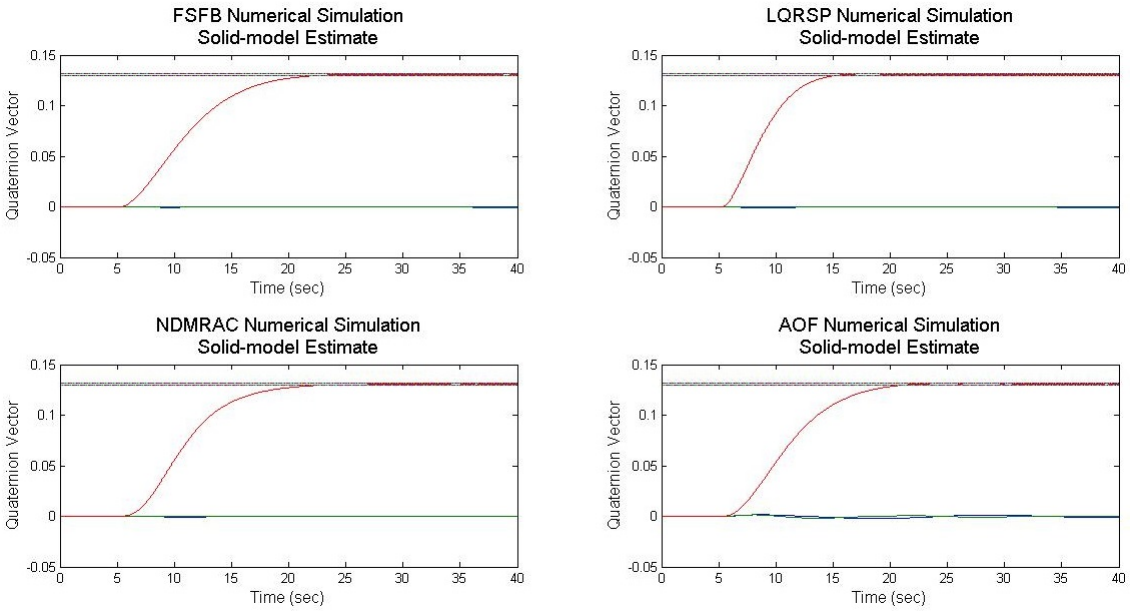

Figure 5.3: Numerical Simulation Quaternion Vector, Solid-Model Estimate Case.

Shown below in Figure 5.4 are the steady-state error plots.

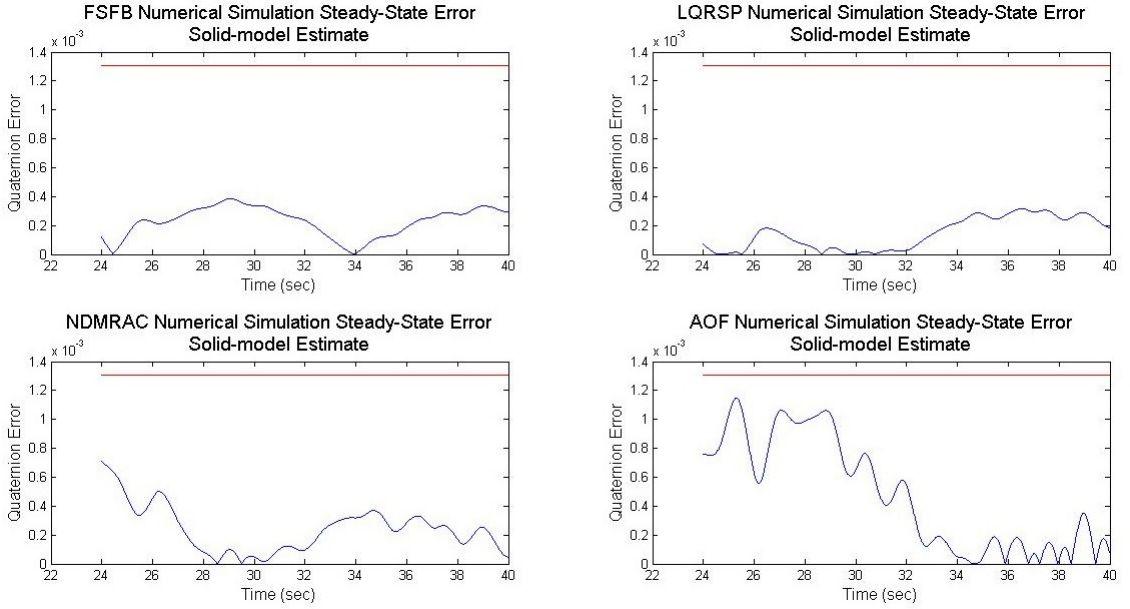

Figure 5.4: Numerical Simulation Steady-State Error, Solid-Model Estimate Case.

Both the quaternion vector and steady-state error data are very similar to the system ID estimate case. Once again the FSFB, LQRSP, and NDMRAC controllers settle the quaternion vectors within the settling time requirement and the steady-state errors are near-zero. Also the AOF again shows

an overshoot between the 24 and 32 second-mark, showing that the AOF controller is experiencing difficulty in making the plant track the reference model. The similarity of the results between the system ID estimate and the solid-model estimate cases indicate that the inertia tensor difference may not have too much effect on the control law performances.

5.2.3 Numerical Simulations Case III: Degraded Wheel

The numerical simulation results for the degraded wheel case will now be discussed. The quaternion vector plots are shown in Figure 5.5.

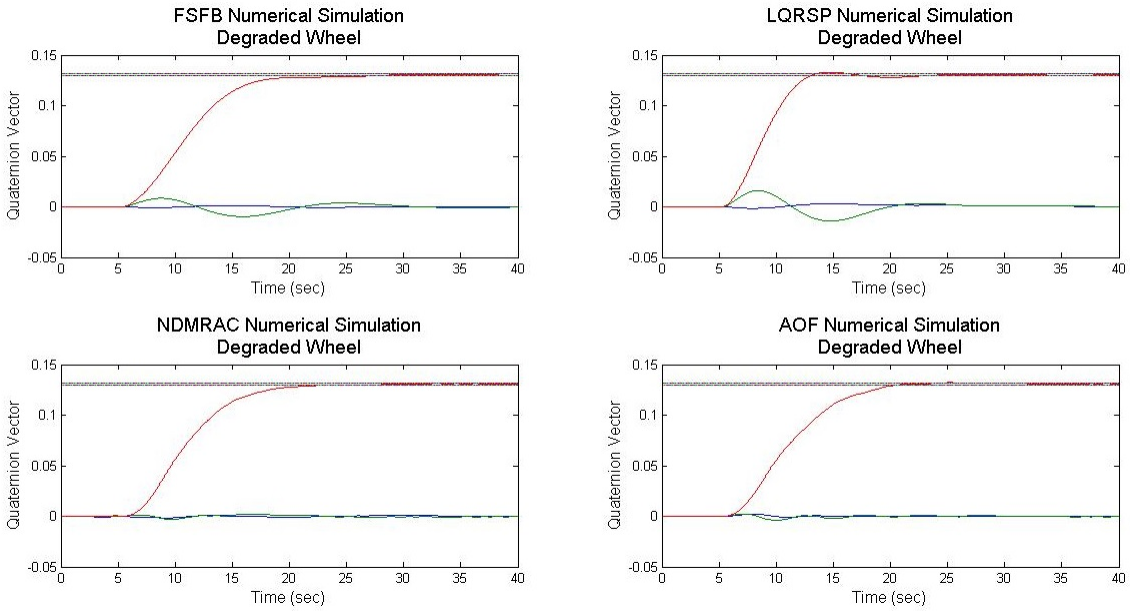

Figure 5.5: Numerical Simulation Quaternion Vector, Degraded Wheel Case.

The steady-state error plots are shown in Figure 5.6.

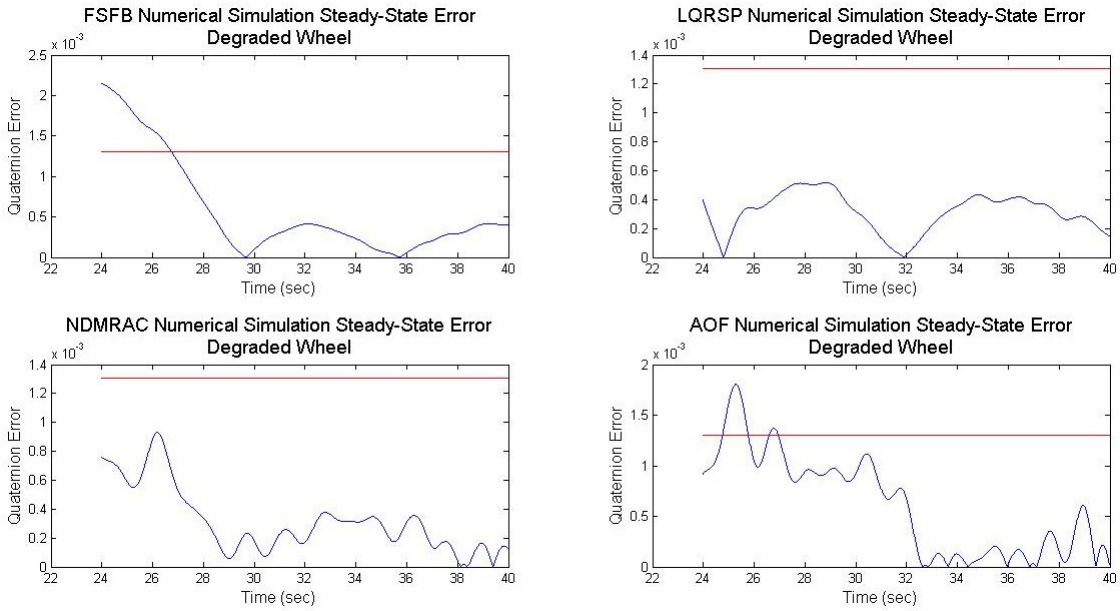

Figure 5.6: Numerical Simulation Steady-State Error, Degraded Wheel Case.

Out of all the test cases the quaternion vectors exhibit the highest amount of oscillations. This is to be expected because the degradation of one reaction wheel means the wheel speed command cannot keep up with the body torque signal from the control laws. However the oscillations in all three axis damps out by the 30 second-mark.

The steady-state errors are also noticeably higher compared to the steady-state errors from the previous two test cases. Although the quaternion vector eventually settles within the error bound, the FSFB controller does not meet the settling-time requirement and takes longer to settle properly. The LQRSP still retains good control performance, but its steady-state error is noticeably higher than the other two test cases. The NDMRAC, despite slightly higher steady-state error, also delivered good control performance. The AOF controller's overshoot worsened such that the quaternion vector was not able to settle by the 25 second-mark. Overall the introduction of actuator degredation, even in a numerical simulation, noticeably worsens control law performance and alters the plant dynamics.

5.3 PRWP Mk.III Motion Test Results

This section will discuss the motion test results from the PRWP Mk. III configuration, with the LN-200 as the attitude determination sensor.

5.3.1 Platform System Limitations

Although the PRWP system is operational and capable of executing motion tests, there are several system hardware limitations that restrict the testing capability of the PRWP. One of the limitations is that the reaction wheel saturation limits are smaller compared to the Mk. II configuration. Since the shafts of the new motors are significantly smaller compared to the previous motor, the reaction wheels cannot be spun as fast without incurring damage to the actuators. Therefore the current reaction wheel saturation limits are smaller, set at 3000 rpm in both directions. Since the wheel saturation limits are smaller, the actuators cannot deliver as much torque as the Mk. II configuration. The lower saturation limits especially affect PRWP motion in the pitch and roll axis. Shown in Figure 5.7 below is a pitch motion by the PRWP using the NDMRAC for the control law.

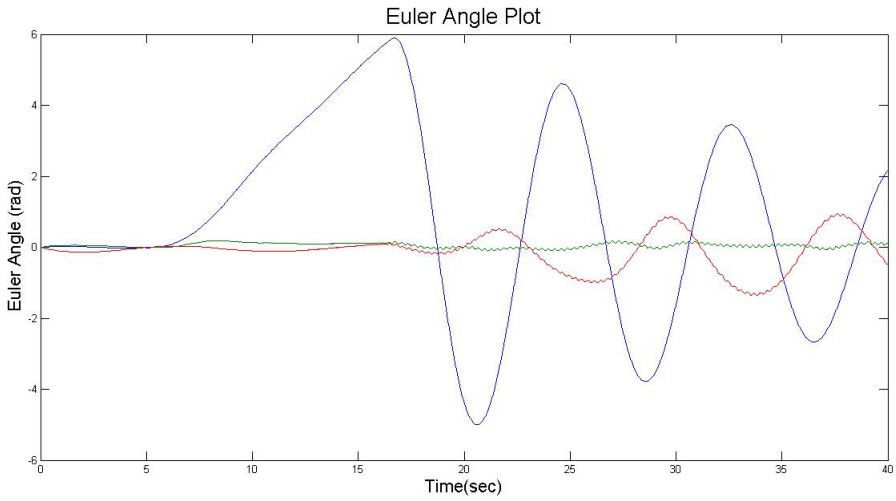

Figure 5.7: PRWP Pitch Motion.

A pitch maneuver of 10 degrees was commanded, but the PRWP could only reach 6 degrees. At the 16 second-mark, the wheels reached its saturation limits and therefore the pitch curve oscillates wildly.

Another unforseen limitation is the presence of a disturbance torque most likely resulting from actuator dynamics uncertainties. Before the motion tests are executed, the reaction wheels are spun

to their initial speeds. It is after the wheels are spun to their bias values that the simulator tends to drift, indicating the presence of a disturbance torque. Particularly the drift tends to be in the clockwise direction around the yaw axis of the simulator, regardless of the wheel spin direction.

To illustrate the effect of the drift effect, the quaternion vector from two separate motion tests with the FSFB controller is shown in Figure 5.8.

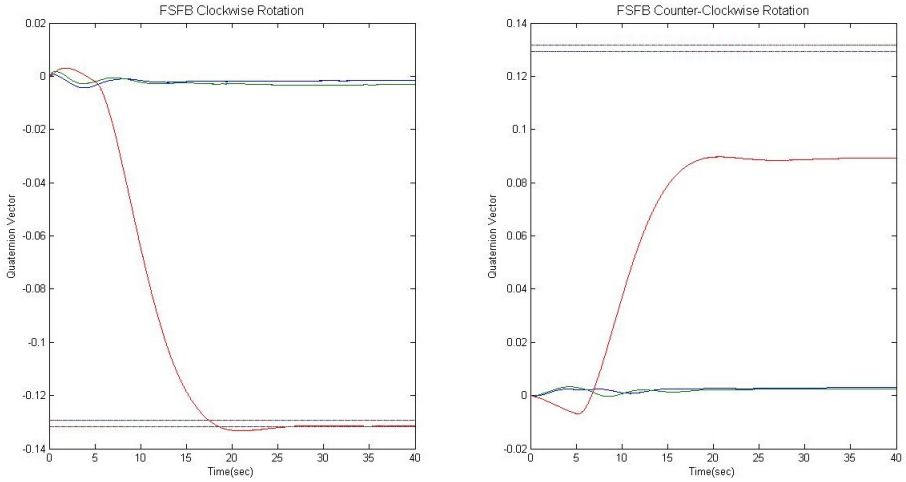

Figure 5.8: Comparison of Counter-Clockwise and Clockwise Rotations.

The left plot is a clockwise yaw rotation of 15 degrees, and the right plot is a counter-clockwise yaw rotation of 15 degrees. Although the settling-time requirement is still not met, the clockwise rotation quaternion vector successfully settles within the error bounds by the end of the simulation while the counter-clockwise rotation fails to do so. These results are consistent with the clockwise drift observed when the reaction wheels are biased. When executing a clockwise yaw rotation, the FSFB does not have to exert as much controller effort to arrive at the command quaternion due to the drift torque. However stopping the simulator and keeping the simulator at the desired quaternion now becomes more difficult since the controller has to resist the drift. When executing a counter-clockwise yaw rotation, the FSFB controller not only has to supply torque to move the platform, but it must also go against the drift torque. This may explain why the quaternion vector remains so far from the command quaternion.

Besides the quaternion vector, other data from the counter-clockwise yaw test indicate the existence of a significant amount of external torque. Shown in Figure 5.9 are the reaction wheel speed, controller output, and the IMU body rate data.

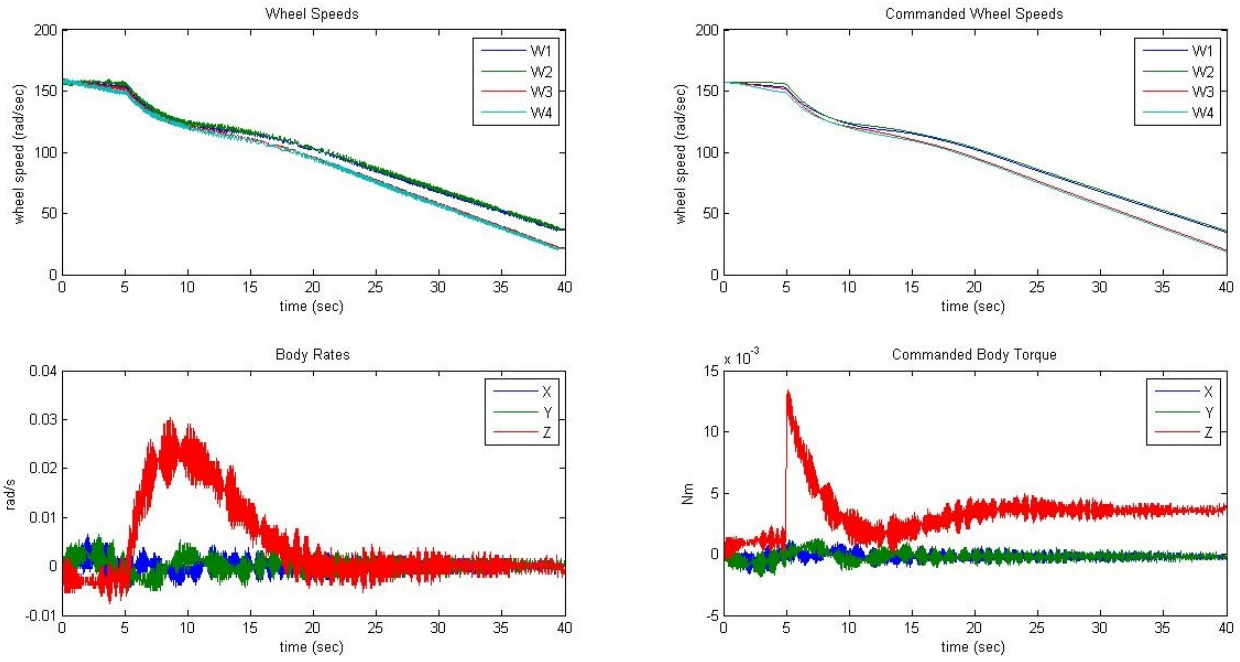

Figure 5.9: Full-State Feedback Test Data, System ID Estimate Case.

Even after the PRWP motion ceased around the 20 second-mark (as indicated by the near-zero body rate curve), the FSFB controller output continues to command a torque and the reaction wheels continue to decelerate. It becomes evident that the drift torque has changed the plant dynamics such that the current FSFB controller cannot compensate for it.

The drift torque must now be taken into account when analyzing the different control laws. The drift torque will necessitate greater controller effort, especially when a counter-clockwise yaw rotation is executed. While it remains unknown the exact cause of the drift torque, it is evident that the reaction wheels are responsible and that further investigations are required to understand the nature of the actuator uncertainty.

5.3.2 Case I: System Identification Inertia Estimate Case

The system ID tests, performed by Dam[22], was used to determine an estimate for the PRWP inertia tensor and CG location. The results for the four controllers are shown in Figure 5.10.

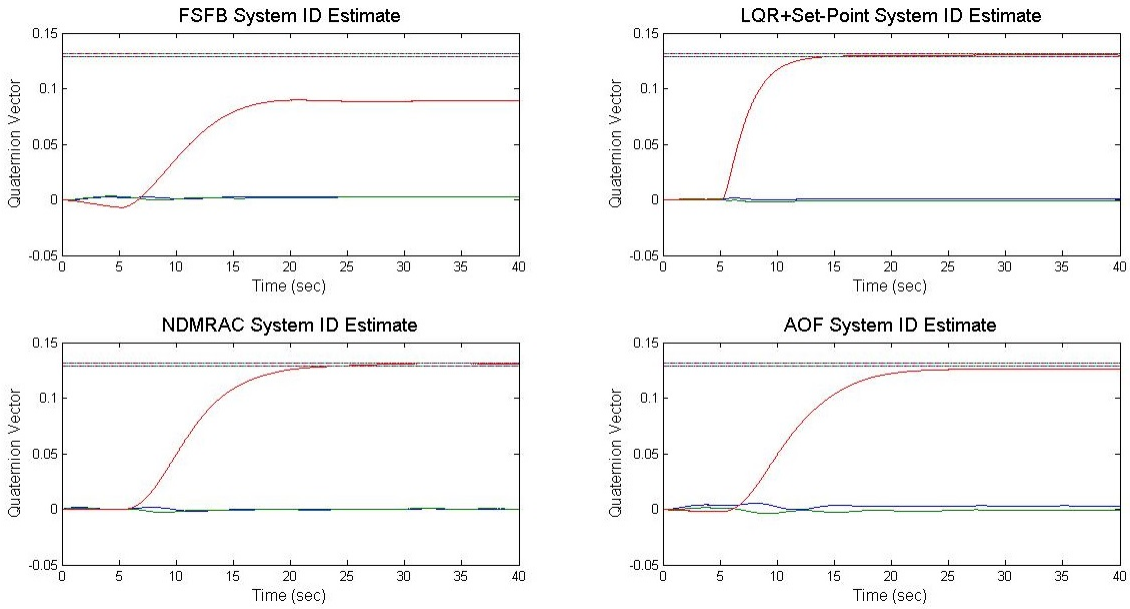

Figure 5.10: System ID Estimate Case Quaternion Vectors.

Both the FSFB and the AOF controllers were not able to settle within the 1% limit, while both the NDMRAC and the LQRSP controllers managed to meet the settling requirements. The steady-state error of the quaternion vectors are also introduced as Figure 5.11 for further comparison between the controllers.

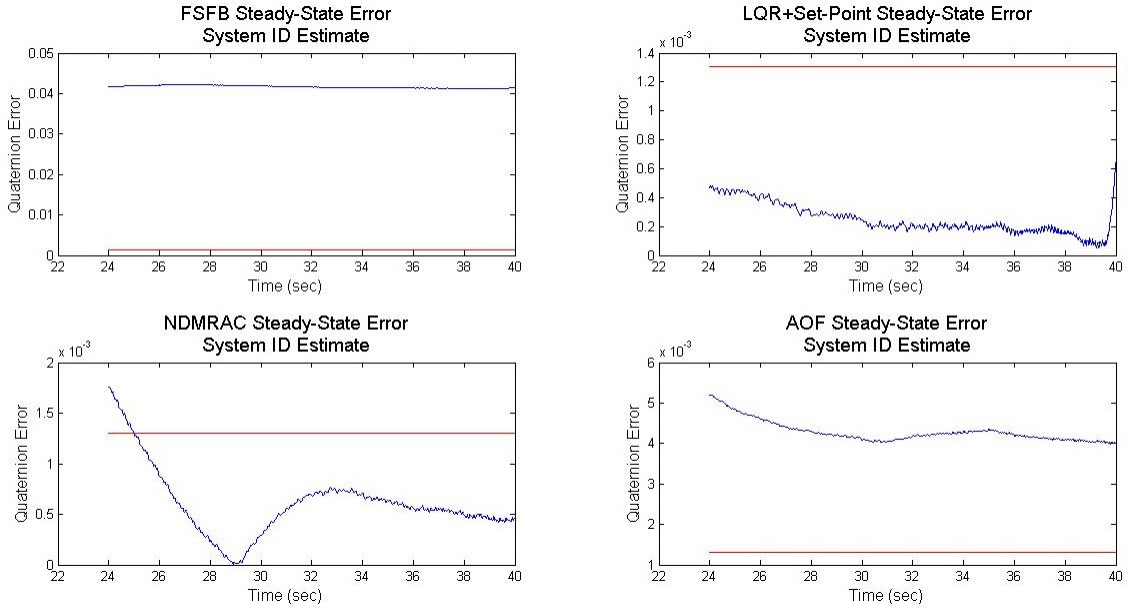

Figure 5.11: Quaternion Steady-State Error, System ID Estimate Case.

The steady-state error plot starts at the 24 second mark, right before the quaternion vector is expected to settle within the error bound, and proceeds to the 40 second mark. The steady-state error is the absolute value of the difference between the commanded and the actual quaternion vector third term. If the control law's steady-state error curve lies under the 1% bound, the steady-state error requirement is satisfied.

Unlike the numerical simulation, the FSFB failed to settle within the steady-state error bound. Again the FSFB's inability to settle is attributed to the drift torque caused by the reaction wheels. To compensate for the settling issue, the FSFB controller was modified such that it can command higher torque values. Specifically the K_{FSFB} and C gains were multiplied by 6, and the damping ratio was reduced to 0.35. The quaternion vector result is shown in Figure 5.12.

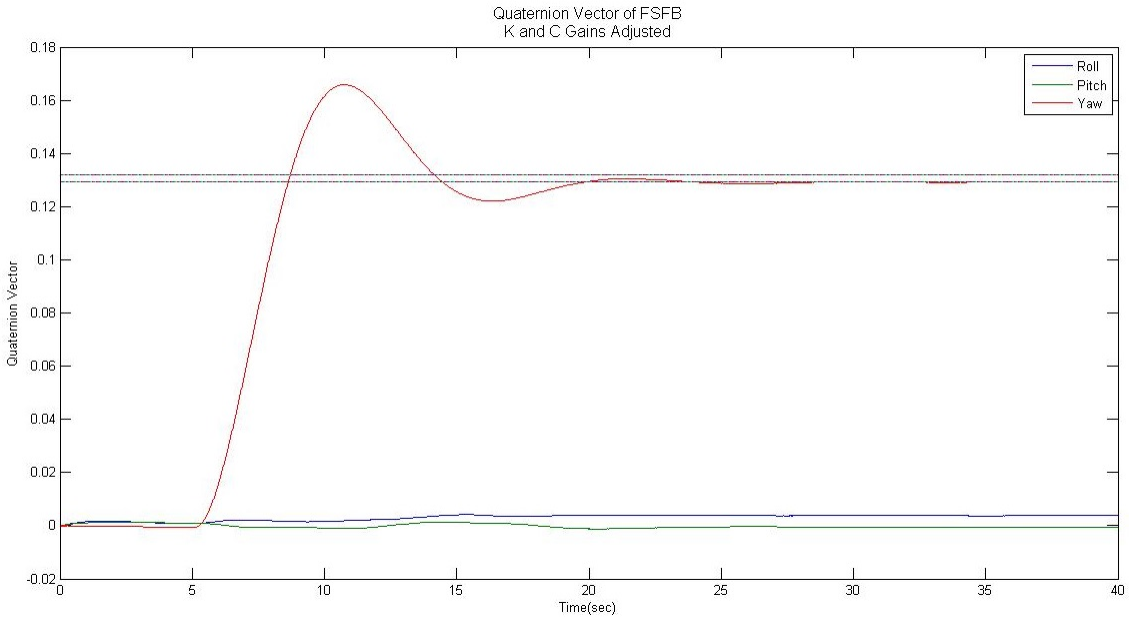

Figure 5.12: Quaternion Vector of Counter-Clockwise Yaw, K and C Gains Adjusted.

At the expense of higher overshoot, the damping ratio was reduced to reduce the C gain and increase the natural frequency of the PRWP which in turn will increase the K gain. The K_{FSFB} and C_{FSFB} gain of the PRWP are equivalent to a PD controller, where the proportional gain places weight on the error signal directly and the derivative gain reduces overshoot at the expense of increasing steady-state error. By placing higher weight on the K_{FSFB} gain and reducing the C_{FSFB} gain, the FSFB now behaves like a PID controller with only a proportional gain. Also the error quaternion, because it is multiplied with the K_{FSFB} gain, now has greater influence in the torque command so that even when the PRWP body rate goes to zero (which nullifies the C gain

term), the FSFB controller will still output a significant amount of body torque.

Until the issue of the drift torque from the reaction wheels can be compensated for, the current K_{FSFB} and C_{FSFB} gains for the FSFB controller are not suited to meet control performance requirements. Although the FSFB control law itself is proven to be asymptotically stable, the K_{FSFB} and C_{FSFB} gains as derived by Mittelstead cannot reliably deliver control performance. For this reason the LQR controller complemented with a set-point controller will serve as the fixed-gain controller for comparison with the adaptive controllers.

The LQRSP controller had the best performance for the system ID estimate case, with the fastest settling time of about 10 seconds and a steady-state error of less than 0.4×10^{-3}. This result was expected since the LQR and set-point controllers were designed for a very specific plant model. The LQR was derived by linearizing the plant model about a 15 degree yaw equilibrium point, and the set-point gains were adjusted until the quaternion vector settled as close as possible to the commanded quaternion value. Even with the presence of external torques, the LQRSP controller meets performance requirements because the set-point controller is specifically designed to converge the PRWP state to a non-zero equilibrium point as fast as possible.

The NDMRAC controller took the full 20 seconds to reach within the steady-state error bound and shows a higher steady-state error compared to the LQRSP. While it is unknown why the NDMRAC experiences such a high steady-state error compared to the two remaining test cases, as will be discussed, the NDMRAC still proves to be asymptotically stable and meet our control performance requirements.

The AOF controller, even though the quaternion vector settled to a constant value, has not settled the quaternion vector within the error bounds. Even though the AOF met settling requirements for the numerical simulations, the difference in plant model due to the external torques may affect the controller such that it is no longer asymptotically stable. One of the condition of asymptotic stability for the adaptive controller is for the plant error to damp to a near-zero value. Although the plant error signal could not be directly obtained, the adaptive gains can confirm the behavior of the plant error since the adaptive gain rates are a function of the plant error. Shown in Figure 5.13 are the adaptive gains for the NDMRAC controller.

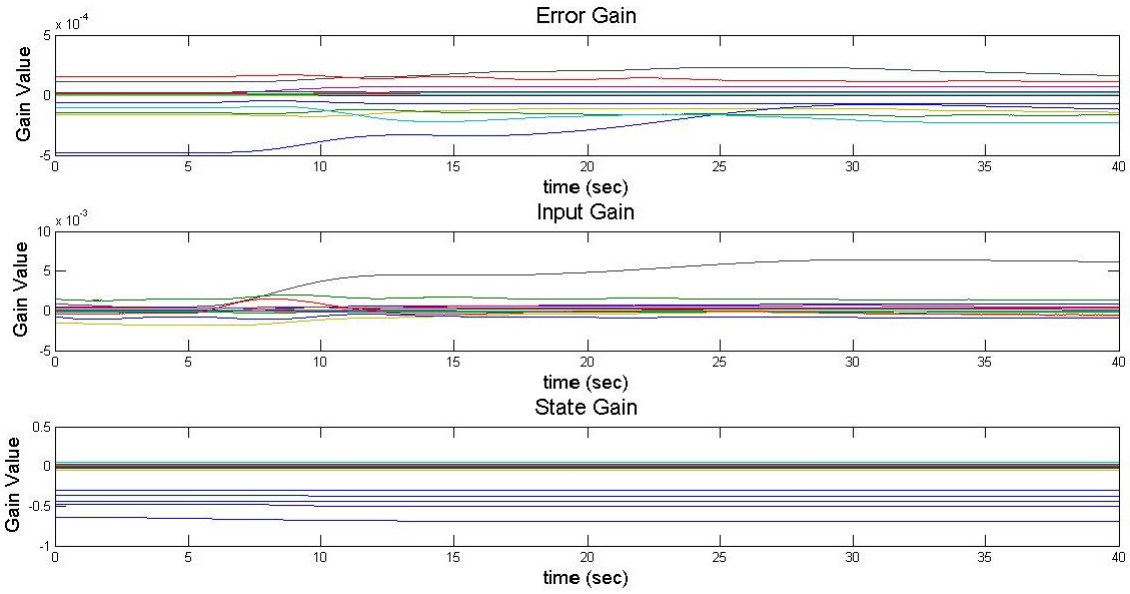

Figure 5.13: NDMRAC Adaptive Gains, System ID Estimate Case.

Shown in Figure 5.14 is the adaptive gain for the AOF controller.

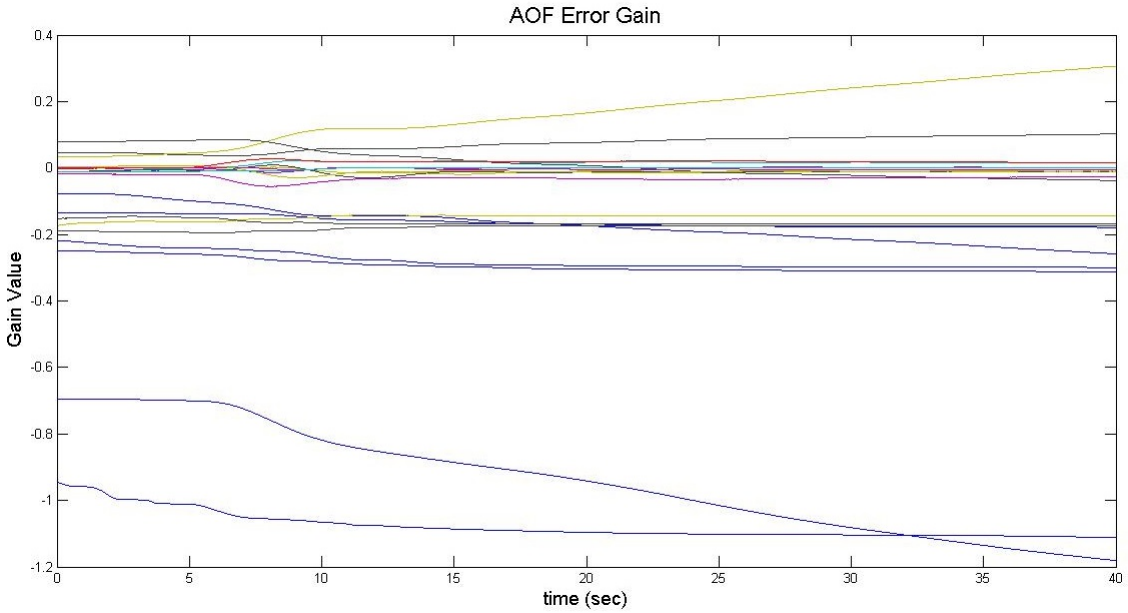

Figure 5.14: AOF Adaptive Error Gain, System ID Estimate Case.

Recalling Eq. 3.21, the adaptive gain rates are a function of the plant error, model input, model states, and adaptive parameters. All 3 adaptive gains for the NDMRAC controller settles to a constant value, indicating that the gain rates reach near-zero values. When the gain rates approach

zero, the plant error also damps out to a near-zero value. This fact is consistent with the quaternion vector steady-state error from the NDMRAC, which shows that the quaternion vector settles within the steady-state error bounds. However the AOF error gain has several terms that do not settle to a constant value, which indicates that the AOF controller has not achieved asymptotic stability. This is consistent with the quaternion vector data for the AOF controller, since the quaternion vector was not able to reach the error bound.

5.3.3 Case II: Solid-Model Estimate Case

The motion test results for the solid-model estimate case will now be discussed. The quaternion vectors for the four controllers are shown below in Figure 5.15.

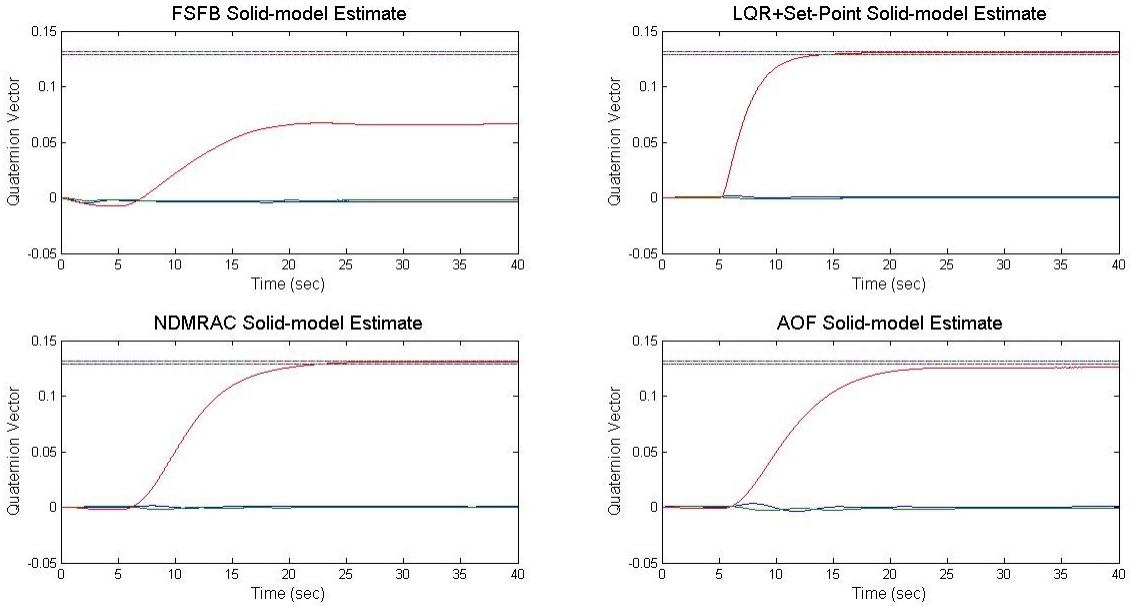

Figure 5.15: Solid-Model Estimate Case Quaternion Vectors.

The quaternion steady-state error plots are shown in Figure 5.16.

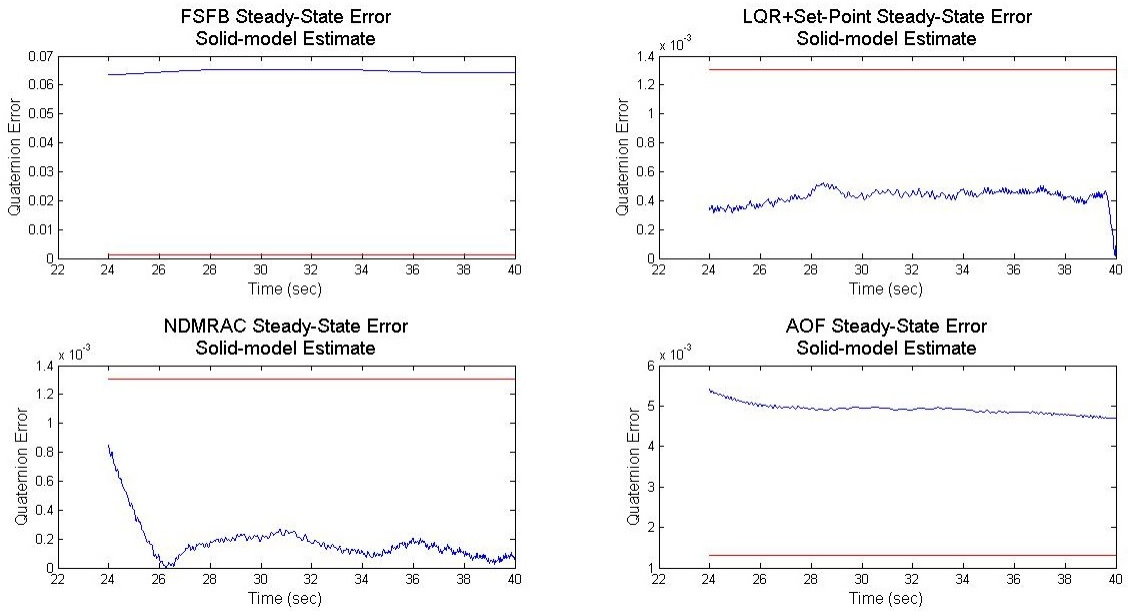

Figure 5.16: Quaternion Steady-State Error, Solid-Model Estimate Case.

The steady-state error shows that the LQRSP and the NDMRAC controllers met the steady-state error and settling time requirements while the FSFB and the AOF controllers did not. Overall the results are similar to the system ID estimate case, with a few differences. One interesting observation to note is that except for the NDMRAC, all the controllers' steady-state error values are higher compared to the steady-state error data from the system ID estimate case. Although it is unknown why only the NDMRAC performance improved compared to the result from the system ID estimate case, the NDMRAC still remains asymptotically stable for both test cases.

To verify asymptotic stability for the adaptive controllers, the respective adaptive gains will now be discussed. Shown in Figure 5.17 are the NDMRAC gains.

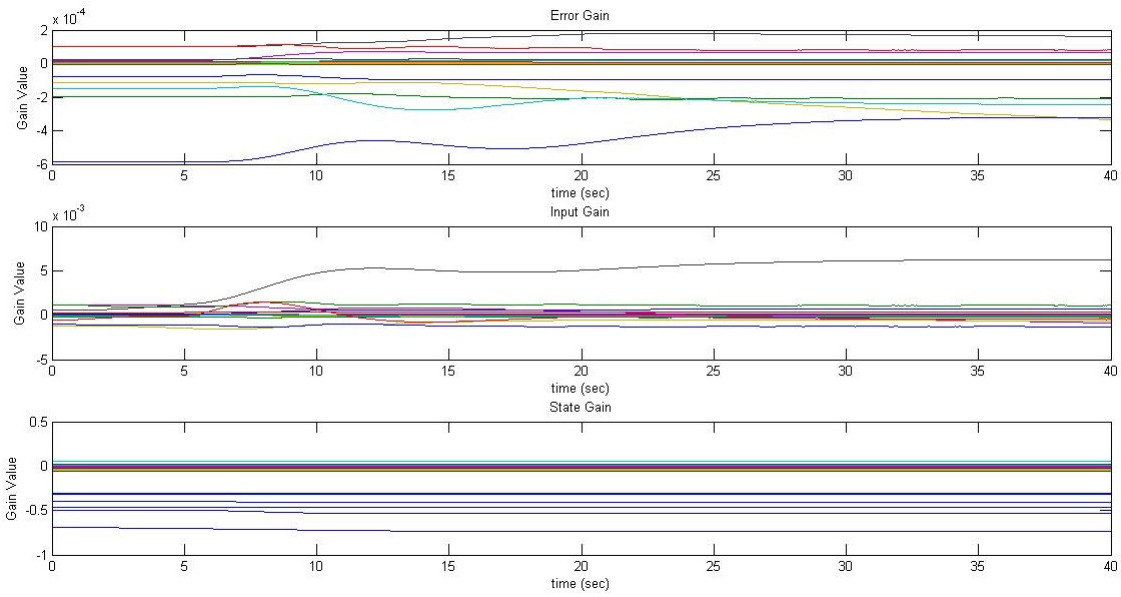

Figure 5.17: NDMRAC Adaptive Gains, Solid-Model Estimate Case.

Shown in Figure 5.18 is the AOF adaptive error gain.

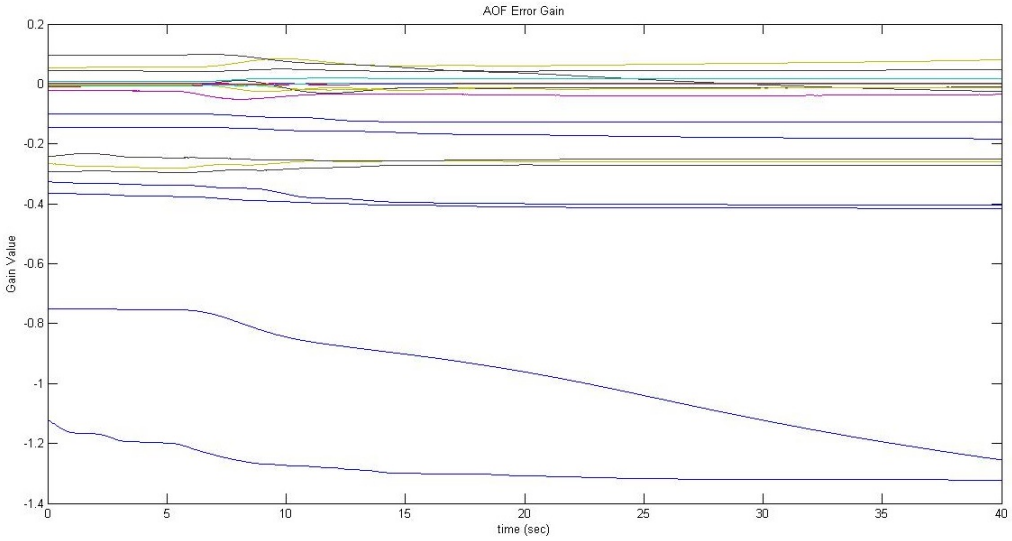

Figure 5.18: AOF Error Gain, Solid-Model Estimate Case.

Again the gain behavior indicates if the adaptive control law achieved asymptotic stability or not because the adaptive gains should settle to a fixed value since the gain rates are dependent on the plant error. All except one term in the AOF error gain settled to a fixed value at the end of the simulation; this fact indicates that the AOF has yet to contain the plant error and therefore may not

have achieved asymptotic stability. On the other hand the NDMRAC adaptive gains all managed to settle to fixed values by the end of the simulation, indicating the achievement of asymptotic stability. The NDMRAC controller may have performed better for this test case due to a better selection of the initial gains compared to the system ID estimate test case.

With the exception of the NDMRAC data, the results from the other controllers suggest that the solid-model inertia tensor estimate may not be as accurate as the system ID inertia estimate due to the higher quaternion vector steady-state errors. However the difference in the inertia tensor estimates does not seem to seriously impact the performance of the controllers. While the quaternion vector curves look quite different between the numerical simulations and the motion tests for the solid-model inertia estimate, both results are consistent in showing that the variation in the inertia tensor did not impact controller performance by much.

5.3.4 Case III: Degraded Wheel Case

The results of the motion test for the degraded wheel case will now be discussed. To verify the degradation of the -Y axis wheel, Figure 5.19 shows the wheel speed plot from the FSFB test.

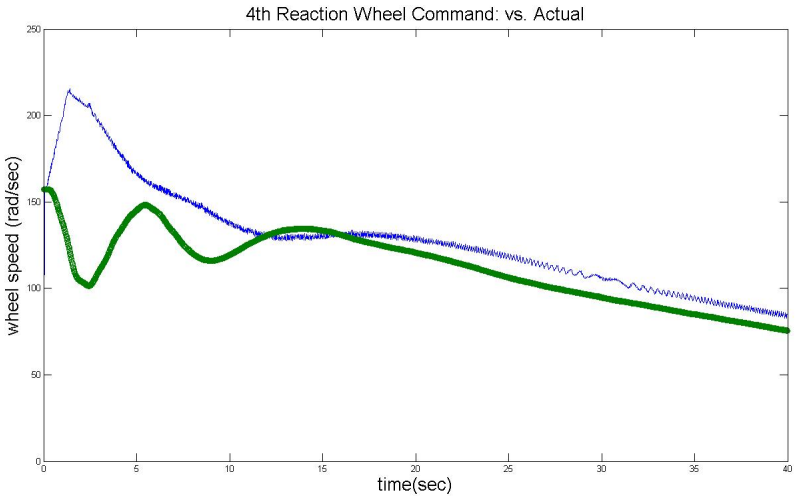

Figure 5.19: -Y Axis Reaction Wheel Plot, Commanded vs. Actual.

The green line is the commanded wheel speed, and the blue line is the actual wheel speed. The commanded wheel speed suffers from greater oscillations than the nominal cases such that the reaction wheel cannot keep up with the commanded speeds. Since the -Y axis wheel is now modeled as an overdamped second-order oscillator, the wheel speed command can no longer keep up with the control law output and therefore will exhibit more oscillations.

In Figure 5.20 the quaternion vectors from the 4 controllers are shown.

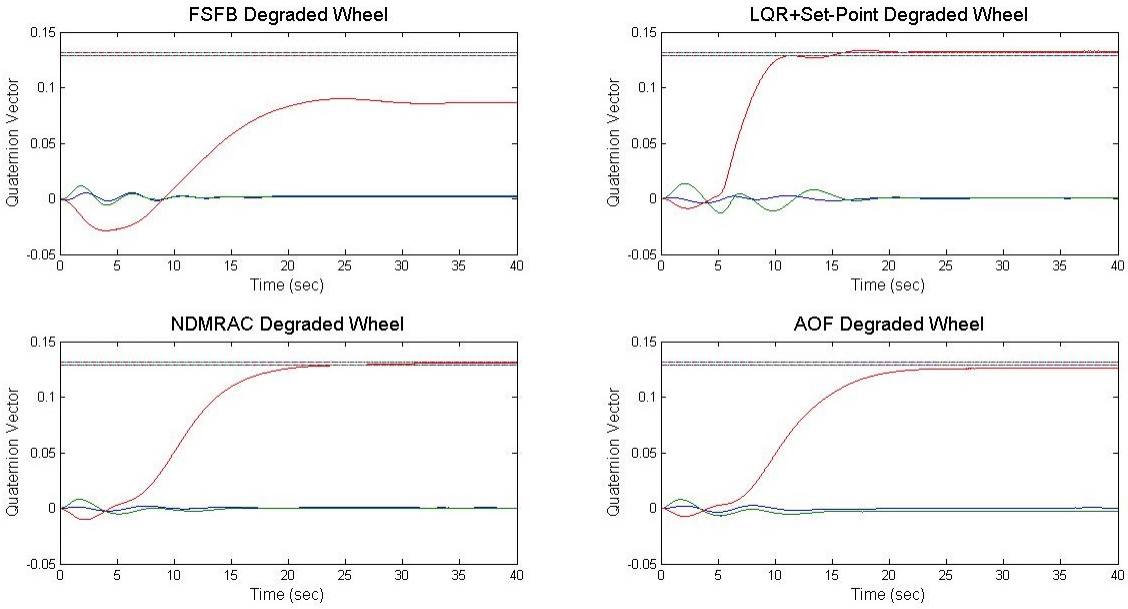

Figure 5.20: Degraded Wheel Case Quaternion Vectors.

Shown in Figure 5.21 are the steady-state error data.

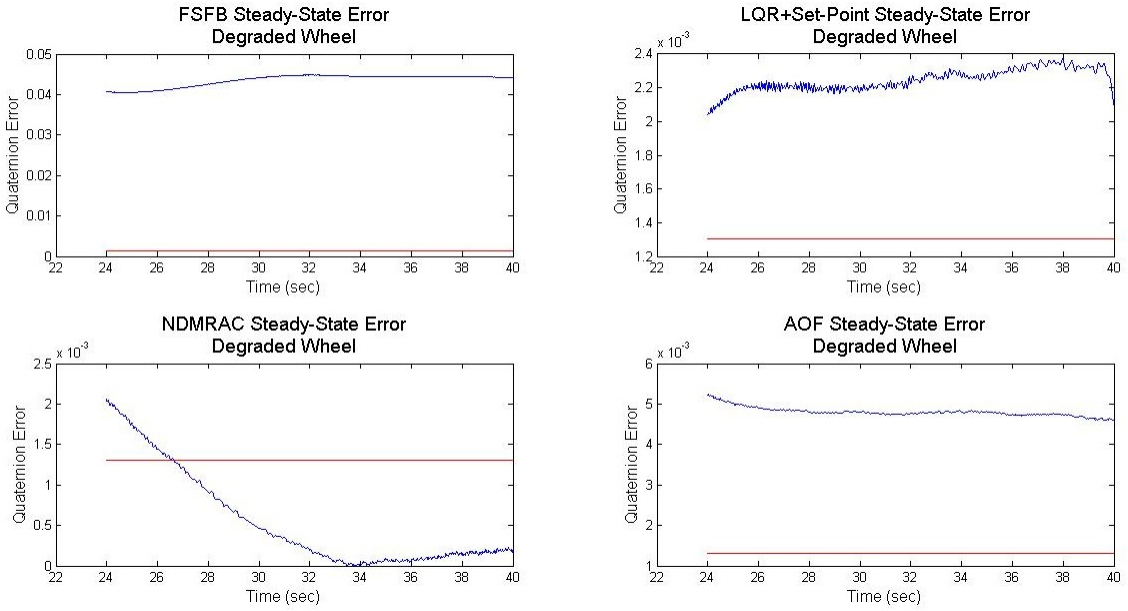

Figure 5.21: Quaternion Steady-State Error, Degraded Wheel Case.

In this test case only the NDMRAC controller managed to make the quaternion vector settle within the error bound. However the NDMRAC fails to meet the settling time requirement of

20 seconds, as the steady-state error does not fall within bounds until after the 26 second mark. At the expense of settling time, the NDMRAC was still able to achieve asymptotic stability and settle the quaternion vector within the error bounds even with actuator uncertainty involved. Also the quaternion vector pitch and roll axis for all controller results experienced greater oscillations compared to the system ID estimate and the solid-model estimate cases. This additional oscillation verifies the presence of actuator uncertainty, since a faulty reaction wheel can negatively affect pitch and roll axis control.

The LQRSP controller, though it settled to a constant steady-state value, was not able to meet the steady-state error bound requirement. Since the LQR and Set-Point controller gains were determined assuming a specific plant model, the introduction of actuator uncertainty made the controller unable to provide control performance. Therefore the LQRSP result verifies that the plant model has significantly changed due to actuator uncertainty.

The NDMRAC adaptive gains are shown in Figure 5.22.

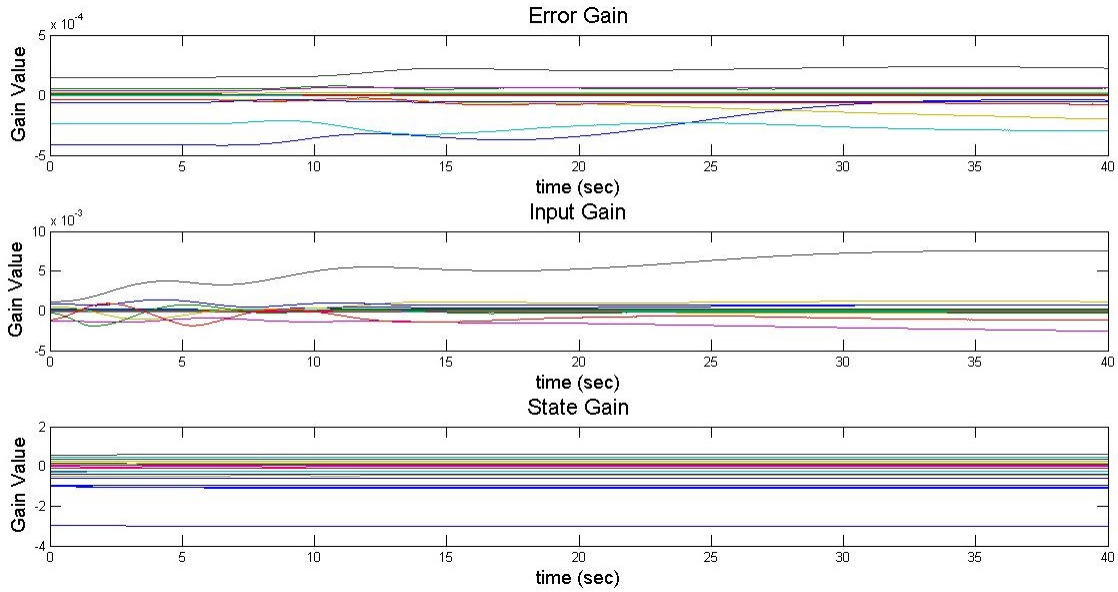

Figure 5.22: NDMRAC Adaptive Gains, Degraded Wheel Case.

Shown in Figure 5.23 is the AOF adaptive gain.

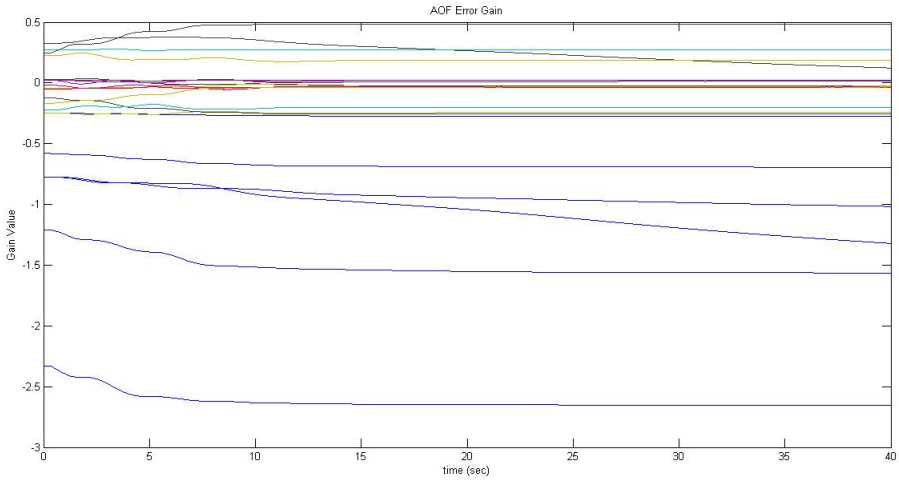

Figure 5.23: AOF Adaptive Error Gain, Degraded Wheel Case.

The NDMRAC gains all settle to constant values by the end of the test, while several terms of the AOF adaptive error gains are not settling to constant values. The gain behaviors are consistent with the quaternion vector data, as the NDMRAC still settles to the command quaternion value while the AOF clearly has too much steady-state error and therefore has a non-trivial amount of plant error remaining.

Of all three test cases, the degraded wheel case had the greatest effect on controller performance. This is in agreement with the numerical simulation results, which also showed that the simulation of actuator performance degradation had the largest impact on controller performance. Unlike the system ID estimate and the solid-model estimate case results, the controllers all showed increasing oscillations and difficulty or inability in meeting performance requirements. However the NDMRAC controller proved to be the most robust of all the controllers, as the fixed-gain controllers (FSFB and LQRSP) couldn't handle significant variations in the plant model without altering the controller algorithm. Although the AOF controller could not settle for all three test cases, the controller all settled to nearly the same equilibrium point. This may indicate that the AOF is stable to a degree but not asymptotically stable. Due to the lack of the state and input gains of the NDMRAC, the AOF may not be able to force the plant to track the reference model properly.

5.3.5 Analysis and Comparison of Motion Test Result

This section will analyze the results from the Mk.III motion tests. In order to confirm the control law validation capability of the Mk.III configuration the motion test results must be compared with

the results from the numerical simulation as well as the Mk.II motion test results produced by Downs[21].

Test Case	Controller	Results
Case I	FSFB	Did not settle within 1%
	LQRSP	Settled within 1%, settled within 20 seconds
	NDMRAC	Settled within 1%, settled within 20 seconds
	AOF	Did not settle within 1%
Case II	FSFB	Did not settle within 1%
	LQRSP	Settled within 1%, settled within 20 seconds
	NDMRAC	Settled within 1%, settled within 20 seconds
	AOF	Did not settle within 1%
Case III	FSFB	Did not settle within 1%
	LQRSP	Did not settle within 1%
	NDMRAC	Settled within 1%, but exceeded 20 seconds settling time
	AOF	Did not settle within 1%

Table 5.2: PRWP Mk.III Motion Test Result Summary.

Table 5.2 summarizes the PRWP Mk.III motion tests. In terms of meeting the control performance requirements, the NDMRAC consistently met performance requirements. Although the NDMRAC was not able to meet the settling-time requirement of 20 seconds for case III, the NDMRAC still showed the best control performance compared to the other three control laws. While the LQRSP controller was able to meet requirements for cases I and II, the controller failed for case III. Both the FSFB and the AOF failed to meet all control performance requirements for all three test cases. In terms of the results between the three test cases, the results between cases I and II were similar and case III showed a general decrease in control performance. The steady-state error for all the controllers were higher and the quaternion vectors experienced more oscillations for the case III test.

Compared to the numerical simulation results the motion test results show that the controllers did not perform as well. In the numerical simulations all four controllers met performance requirements while the motion test results show that only the NDMRAC and the LQRSP met performance requirements for certain test cases. The difference in results indicate a discrepancy between the numerical simulation's plant model and the physical PRWP system. Particularly the external torque

may be responsible for the difference between the results of the numerical simulation and the motion test. While the numerical simulation assumes no external torque acting on the system, the PRWP may have been experiencing a substantial amount of external torque throughout the motion test. A significant amount of external torque will necessitate greater effort from the controller to counteract it or cause the controller to miss its requirements completely. The FSFB and the AOF controllers' inability to meet performance requirements may be explained by the presence of external torque acting on the PRWP. A combination of the drift torque, as mentioned in chapter 5 section 3.1, and the CG offset torque is likely the cause of the FSFB and the AOF controller's inability to settle the PRWP quaternion vector.

In terms of the controller robustness the Mk.III confirmed the trends shown in the numerical simulation. Both the numerical simulation and the motion test results show that both the case I and II tests had similar results. While the inertia estimate between the system ID and the solid-model estimates were different, the variations in inertia tensor estimate did not significantly affect the controller performance. Also both the numerical simulations and the motion tests show that the introduction of actuator dynamic uncertainty in the case III test had the most effect on controller performance. Both the numerical simulation and the motion test results showed the quaternion vectors experiencing increased oscillations and increased steady-state error compared to the case I and II results.

The Mk.III motion test results have various similarities as well as differences compared to the Mk.II motion test results. Both the Mk.II and the Mk.III tests show that the results between cases I and II were similar. Although the two test cases' inertia tensor estimates were different, the estimate difference was not enough to significantly affect the performance of the control laws. Just like the Mk.II degraded wheel case results, the Mk.III degraded wheel case results show that all the quaternion vectors experienced more oscillations. Both the Mk.II and the Mk.III test results agree that the introduction of actuator dynamics uncertainty has had a greater effect on controller performance than using varying inertia tensor estimates of the PRWP. Also the Mk.III test results confirm that the NDMRAC consistently meets performance requirements in all three test cases. While the LQRSP controller was specific to the Mk.III test, the NDMRAC has shown itself to be more robust than both the FSFB and LQRSP fixed-gain controllers. Both the Mk.II and the Mk.III tests show the adaptive control laws outperforming fixed-gain control laws.

However the Mk.III motion test results showed some differences from the Mk.II test results. Although in all three test cases for both the Mk.II and the Mk.III the FSFB control law was not able to meet the 1% steady-state requirement, the Mk.III results show that the steady-state error

for the FSFB controller is much worse than what the Mk.II results show. A possible explanation of the difference is the external torque acting on the PRWP Mk.III as well as the higher bias drift that the Mk.II attitude determination sensor suffers from. As explained in section 5.3.1, the external torque acting on the PRWP system will necessitate greater effort from the control law or result in greater steady-state error if the controller is not robust enough to compensate for it. In the case of the PRWP Mk.III configuration there is good evidence to believe that the drift torque, an external torque acting in the clockwise direction about the yaw axis, exists and that it is affecting the performance of the PRWP control laws. While it is unknown if the Mk.II configuration of the PRWP experienced an equal amount of external torque compared to the Mk.III configuration, the external torque seems to have had a significant effect on the Mk.III test results.

In the case of the Mk.II configuration a high sensor bias drift can also affect motion test results. Because the sensor bias measurement is constantly drifting, the Mk.II is using a flawed measurement of the PRWP attitude and may not complete attitude maneuvers correctly. In the Mk.II motion tests the FSFB, although it still did not settle to the desired quaternion value, had a far lower steady-state error compared to the FSFB controller from the Mk.III test results. The Mk.II configuration's higher sensor bias drift may account for the discrepancy, meaning that the Mk.II sensor showed an attitude rotation that may have been higher than how much the PRWP physically rotated.

Even the AOF controller, which had consistently met performance requirements on the Mk.II tests, failed to meet performance requirements on the Mk.III tests likely due to external torque acting on the PRWP. The Mk.III results may have exposed a limitation of the AOF controller that was not made apparent in the Mk.II motion tests. Even though the AOF controller was proven to be asymptotically stable by Lyapunov's definition, the stability proof also assumes that the nonlinear portion of the plant model is bounded. The external torque acting on the PRWP Mk.III may have violated such assumption and therefore caused the AOF controller to be unable to meet the performance requirements. While the stability proof and its assumptions are the same for the NDMRAC, the NDMRAC outperformed the AOF controller possibly due to the difference in its control input algorithm. Unlike the AOF the NDMRAC utilizes the plant error, the model input, and the model state as part of its control input. Therefore the NDMRAC is able to produce greater control effort compared to the AOF controller.

The Mk.III results confirm both with the numerical simulations and the Mk.II test results that uncertainties in the actuator dynamics affect controller performance more than variations of the PRWP inertia tensor estimates. This confirmation shows that the PRWP Mk.III configuration is capable of testing the robustness of control laws for attitude control. The Mk.III results showed some

differences of the performance of the individual controllers such as the FSFB and AOF controllers. However the Mk.III motion test results still agree with the numerical simulations and the Mk.II motion test results that the NDMRAC performance exceeds that of fixed-gain controllers. Overall the Mk.III configuration has shown itself to be capable of testing control law performance and robustness despite interference from external torque acting on the system.

Chapter 6

Concluding Remarks

6.1 Conclusion

Four different control laws were tested through motion tests on the Mk.III configuration of the PRWP. To test the robustness of the controllers, three different cases of the plant parameters were investigated.

The integration of the LN-200 IMU significantly reduced the sensor bias drift compared to the MEMS gyroscope, and the Mk.III motion tests revealed controller capabilities and limitations previously not seen from the Mk.II motion test results. One important result from the motion tests is that the PRWP experiences a significant amount of external torque besides the CG torque. Named the drift torque, the drift torque is most likely caused by the spinning of the reaction wheels. Only when the wheels are spun, does the simulator drift clockwise. This is problematic because the drift torque necessitates higher effort from the controllers to compensate for it, if the controller can compensate for it at all.

The motion tests also revealed that the FSFB controller, as it is currently derived, is not capable of counteracting the external torque on the system. Only after manipulating the gains by increasing both feedback gains and reducing the system damping ratio did the quaternion vector converge to its commanded value. The FSFB controller's inability to settle is likely due to the external torque, since the numerical simulation showed that the FSFB should be able to control the simulator. Therefore the LQRSP controller was used as the fixed-gain controller to be compared with the adaptive controllers. The LQRSP controllers met performance requirements for the system ID and the solid-model estimate cases, but failed on the degraded wheel case.

The NDMRAC turned out to have the best control performance of all the controllers, as the NDMRAC settled the quaternion vector within the ±1% consistenly for all three cases. However the NDMRAC failed the settling time requirement on the degraded wheel case, as it took slightly longer for the quaternion vector to settle properly. The AOF controller was not able to meet the settling error requirement for all three test cases, though the steady-state error remained consistent for all three test cases. This result may indicate that the AOF is only asymptotically stable when the external torque acting on the PRWP is under a certain amount. While the numerical simulations showed that the AOF is stable assuming no external torque acting on the PRWP, the CG torque combined with the drift torque may have been too much for the AOF controller to compensate for.

Unfortunately the Mk.III test results also revealed several limitations of the Mk.III testing capability. Due to the external torque acting on the system as well as the lower reaction wheel speed saturation limits the PRWP Mk.III's motion range is limited only to yaw-axis rotations. Despite attempts for pitch and roll-axis motions the PRWP cannot rotate more than 6 degrees before the reaction wheels encounter speed saturation limits.

Despite the Mk.III's limitations as well as several differences in test results, the Mk.III motion test results tend to agree with the results from the numerical simulations as well as the Mk.II motion test results. The Mk.III confirms that the introduction of actuator dynamics uncertainty has a greater effect on the controllers than using different estimates of the PRWP inertia tensor. The Mk.III tests also confirm that the NDMRAC tends to be more robust than fixed-gain controllers.

6.2 Further Development for the PRWP

The drift torque, which is likely caused by uncertainty in the reaction wheels, must be understood and compensated for for further work. The drift torque significantly increases controller effort requirements and changes the plant model, and therefore it must either be minimized or accounted for in the future plant model. The cause is most likely a poor balancing of the reaction wheels, as the reaction wheel noticeably vibrate while spinning. The proper mating of the reaction wheel and the motor shaft may correct the wheel imbalance and thereby reduce the drift torque.

Also the motion of the PRWP is currently limited to yaw axis rotations, as a sustained roll or pitch rotation of over 6 degrees cannot be performed with the current reaction wheels. To give the simulator a greater range of rotational maneuvers, a new set of reaction wheels and motors that have higher saturation limits and higher torque capabilities are necessary to deliver the high amount of torque necessary for pitch and roll axis motions. Another alternative would be to integrate

another attitude control device such as cold-gas thrusters or control moment gyroscopes. Using a combination of reaction wheels and thrusters and/or CMGs can reduce required effort from the reaction wheels which can reduce the likelihood of reaction wheel saturations.

Although the bias drift of the LN-200 IMU is significantly lower than that of the Mk.II configurations' MEMS gyroscopes, the IMU drift is not negligible and will affect the validity of the Mk.III configuration's attitude determination. Therefore it would be beneficial to add a second attitude determination device onto the PRWP Mk.III such as a horizon sensor, star tracker system, or an optical system such as laser guidance. A secondary attitude determination can be used to correct the IMU drift as well as determine the absolute inertial position of the PRWP. Currently the PRWP Mk.III does not have a method for obtaining its position in an inertial frame, and therefore the addition of a second attitude determination device would help in improving the Mk.III configuration's attitude determination capability.

Bibliography

[1] Schwartz, J., Peck, M., and Hall, C. *Historical Review of Air-Bearing Spacecraft Simulators*, Journal of Guidance, Control, and Dynamics, Vol.26 No.4, 2003.

[2] Jian, X., Gang, B., Yang, Q., Jun, Li. *Design and Development of a 5-DOF Air-Bearing Spacecraft Simulator*, International Asia Conference of Informatics in Control, Automation and Robotics, pp.126 - 130, 2009.

[3] Li, Y., Gao, Y. *Study on Attitude Control for Three Degrees of Freedom Air-Bearing Spacecraft Simulator*, International Conference on Mechanic Automation and Control Engineering, pp.408-411, 2010.

[4] Kim, B., Velenis E., Kriengsiri, P., Tsiotras, P., *A Spacecraft Simulator for Research and Education*, AAS/AIAA Astrodynamics Specialists Conference, 2001.

[5] Cho, S., Shen, J., McClamroch, N.H., Bernstein, D.S. *Equations of Motion for the Triaxial Attitude Control Testbed*, 40th IEEE Conference on Decision and Control, 2001.

[6] Jung, D., and Tsiotras, P., *A 3-DoF Experimental Test-Bed for Integrated Attitude Dynamics and Control Research*, AIAA Guidance, Navigation, and Control Conference and Exhibit, Austin, Texas, 2003, AIAA paper 2003-5331.

[7] Joshi, S.M., Kelkar, A.G., and Wen, J.T.Y., *Robust Attitude Stabilization of Spacecraft Using Nonlinear Quaternion Feedback*, IEEE Transactions on Automatic Control, Vol.4, No.10, 1995, pp. 1800 - 1803.

[8] Romano, M., and Agrawal, B.N., *Acquisition, Tracking and Pointing Control of the Bifocal Relay Mirror Spacecraft*, Acta Astronautica, Vol.53, 2003, pp. 509 - 519.

[9] Gao, Y., Wu, J., Hu, M., and Zheng, W. *Design and Development of Hardware-in-Loop Simulation of Spacecraft Attitude Control System Based on Wireless Ad Hoc Networking*, International

Conference on Industrial Control and Electronics Engineering, Xi'an, People's Republic of China, 2012.

[10] Stengel, R.F., and Ray, L.R. *Stochastic Robustness of Linear Time-Invariant Systems*, IEEE Transactions on Automatic Control, Vol.36, No. 1, January 1991.

[11] Brogan, W.L. *Modern Control Theory*, 3rd ed., Prentice Hall, Upper Saddle River, NJ, 1991.

[12] Astrom, K. and Wittenmark, B. *Adaptive Control*, Dover Publications Inc., Mineola, NY, 2008.

[13] Mahyuddin, M.N., Arshad, M.R., Mohamed, Z. *Simulation of Direct Model Reference Adaptive Control on a Coupled-Tank System using Nonlinear Plant Model*, International Conference on Control, Instrumentation and Mechatronics Engineering, 2007.

[14] Shin, Y., Calise, A.J., Johnson, M.D. *Adaptive Control of Advanced Fighter Aircraft in Nonlinear Flight Regimes*, Journal of Guidance, Control, and Dynamics, Vol.31, No.5, 2008.

[15] Sozer, Y., Kaufman, H., Torrey, D.A. *Direct Model Reference Adaptive Control of Permanent Magnet Brushless DC Motors*, IEEE International Conference on Control Applications, pp.633-638, 1997.

[16] Kim, J.J., Agrawal, B.N., *Automatic Mass Balancing of Air-Bearing-Based Three-Axis Rotational Spacecraft Simulator*, Journal of Guidance, Control, and Dynamics, Vol.32, No.3, May-June 2009.

[17] Swei, S.S. *Spacecraft Attitude Control Design with Adaptive Bounded Controllers*, AIAA Guidance, Navigation, and Control Conference, Boston, MA, 2013.

[18] Yoon, H., and Tsiotras, P. *Adaptive Spacecraft Attitude Tracking Control with Actuator Uncertainties*, The Journal of the Astronautical Sciences, Vol.56, No.2, April-June 2008, pg.251-268

[19] Mittlestead, Carson O. *Results on the Development of a Four-Wheel Pyramidal Reaction Wheel Platform*, Master's Thesis, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2006.

[20] Kinnett, Ryan. *System Integration and Control of a Low-Cost Spacecraft Attitude Dynamics Simulator*, Master's Thesis, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2010.

[21] Downs, Matt. *Adaptive Control Applied to the Cal Poly Spacecraft Attitude Dynamics Simulator*, Master's Thesis, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2009.

[22] Dam, L., *Applied Mass Properties Identification Method to the Cal Poly Spacecraft Simulator*, Master's Thesis, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2013.

[23] Saile, Christopher A. *Center of Mass Fine Tuning System for the Cal Poly Spacecraft Simulator*, Senior Project, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2007.

[24] Silva, Seth F. *Applied System Identification for a Four Wheel Reaction Wheel Platform*, Master's Thesis, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2008.

[25] Peck, M.A., and Cavender, A.R. *An Airbearing-Based Testbed for Momentum-Controlled Systems and Spacecraft Line of Sight*, 13th AAS Spaceflight Mechanics Winter Meeting, 2003.

[26] Iversen, Phil. *LN-200*, Senior Project, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2005.

[27] Sidi, Marcel J. *Spacecraft Dynamics and Controls: A Practical Engineering Approach*, Cambridge University Press 40 West 20th Street, New York, NY, 2007.

[28] Wie, Bong. *Space Vehicle Dynamics and Control*, 2^{nd} ed., AIAA Education Series, AIAA, Reston, VA., 2008.

[29] Healy, Patrick. *Mass Property System Identification of a Spacecraft Simulator*, Master's Thesis, Aerospace Engineering Dept., California Polytechnic State University, San Luis Obispo, CA, 2006.

[30] Landau, L.D., and Lifshitz, E.M. *Mechanics*, 2nd. ed., Pergamon Press, London, England, UK, 1969.

[31] Meriam, J.L., and Kraige, L.G. *Engineering Mechanics: Dynamics*, 6th ed., John Wiley and Sons, Hoboken, NJ, 2007.

[32] Goldstein, H., Poole, C., and Safko, J. *Classical Mechanics*, 3rd ed., Addison Wesley, Boston, MA, 2001.

[33] Robinson, A.C. *On the Use of Quaternions in Simulation of Rigid-Body Motion*, WADC Technical Report 58-17, December 1958.

[34] Harding, C.F. *Solution of Euler Gyrodynamics*, Journal of Applied Mechanics, Vol.31, Ser. E, No. 2, 1964.

[35] Mortenson, R.E. *Comment on Solution of Euler Gyrodynamics*, Journal of Applied Mechanics, Vol.32, Ser. E, No. 1, 1965.

[36] Nise, N. *Control Systems Engineering*, 6th ed., John Wiley & Sons, Hoboken, NJ, 2010.

[37] Hespanha, Joao. *Linear System Theory*, Princeton University Press, Princeton, New Jersey 2009.

[38] Mehiel, Eric A. *On Direct Model Reference Adaptive Controller Design for Flexible Space Structures*, Ph.D. Dissertation, Aerospace Engineering Sciences, University of Colorado, Boulder, Colorado, 2003.

[39] Torres, Simon. *Nonlinear Adaptive Control for Spacecraft Rigid Body Equations of Motion*, Master's Thesis, Aerospace Engineering Department, California Polytechnic State University, San Luis Obispo, CA, 2006.

[40] Patel, Rushabh C. *Adaptive Output Feedback Control as Applied to the Rigid Body Equations of Motion*, Master's Thesis, Aerospace Engineering Department, California Polytechnic State University, San Luis Obispo, CA, 2007.

[41] Ljung, L. *System Identification: Theory for the User*, Englewood Cliffs, NJ, Prentice-Hall, 1987.

[42] Sinha, N.K., and Kuszta, B. *Modeling and Identification of Dynamic Systems*, Van Nostrand Reinhold Co, New York, NY, 1983.

[43] Juang, J. *Applied System Identification*, Prentice Hall, Englewood Cliffs, NJ, 1994.

Appendix A

Appendix: Software

This appendix section outlines the software developed for the Mk.III configuration of the PRWP. Section A.1 will go over the software developed to operate the Litton LN-200 inertial measurement unit (IMU). Section A.2 will go over the software developed to operate the reaction wheel system. Section B will go over motion test setup m-file scripts.

A.1 LN-200 IMU Code

The data-acquisition code for the LN-200 IMU consists of components written in both the C language as well as Simulink. The source code is written in C, and is compiled and executed in the Simulink environment. Subsection A.1.1 will cover the Simulink files. Subsection A.1.2 contains the source codes written in the C language.

A.1.1 Simulink Block

Shown in Figure A.1 is the first level of the Simulink block for the LN-200 IMU data-acquisition code.

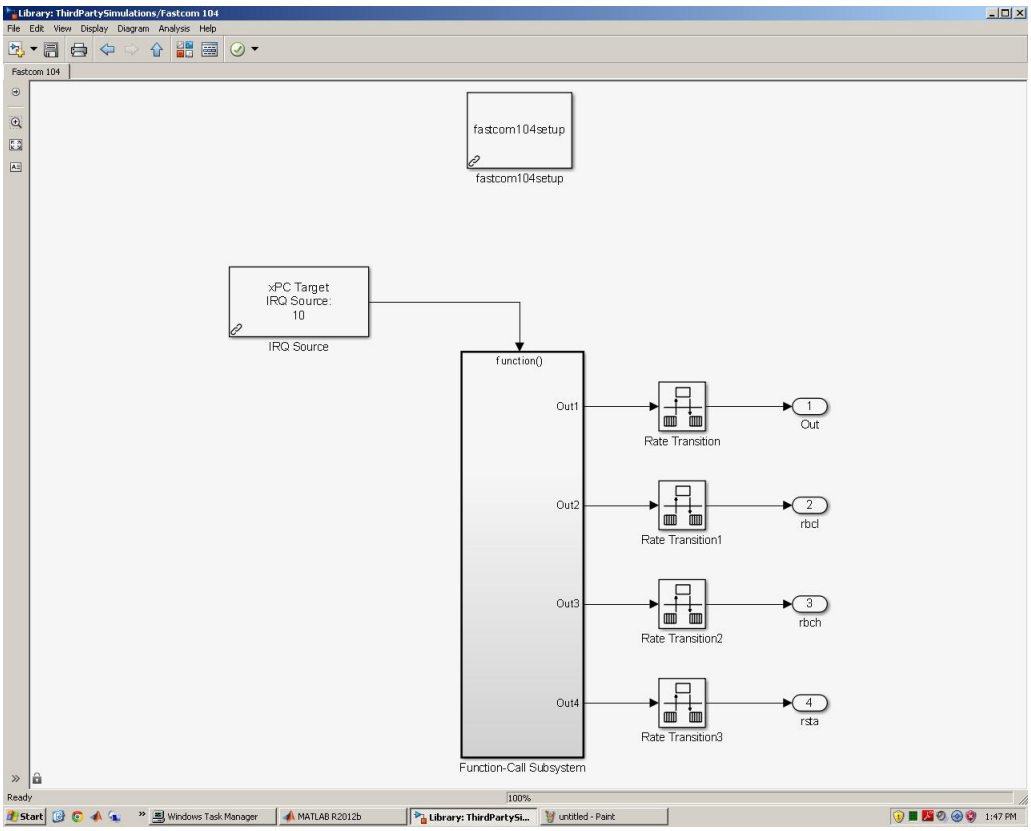

Figure A.1: LN-200 Data Acquisition Code, Simulink Diagram Level 1.

Figure A.2 shows the second level of the Simulink block of the LN-200 IMU data-acquisition code.

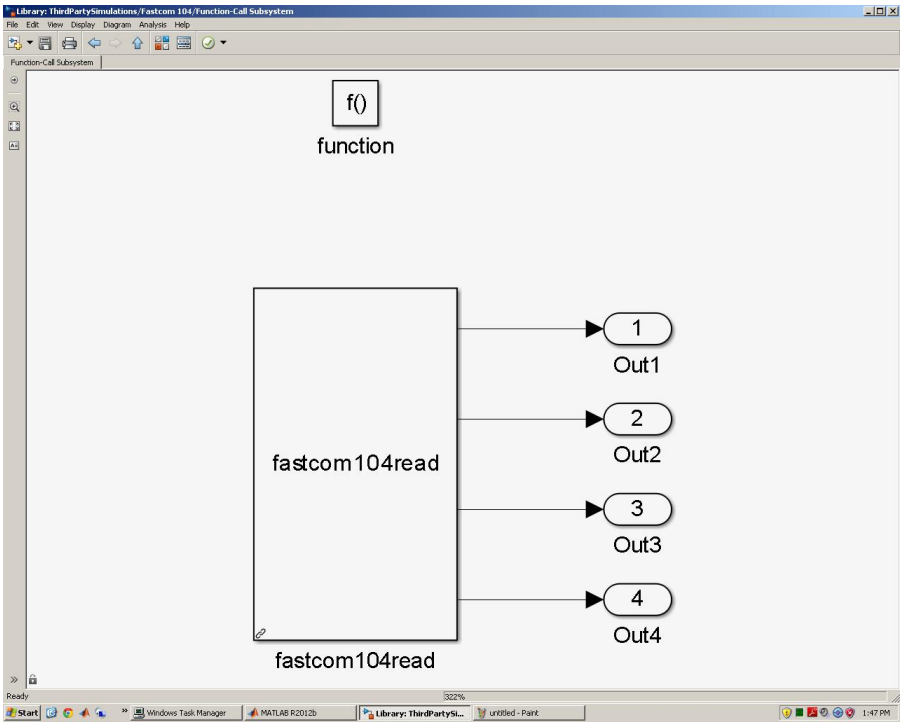

Figure A.2: LN-200 Data Acquisition Code, Simulink Diagram Level 2.

The code reads the IMU data through the Fastcom ESCC-104 board and converts the raw data to body rate and quaternion data. The LN-200 Simulink block consists of three custom-developed sub-blocks:

1. fastcom104setup
2. fastcom104read
3. fastcom104hook

The block 'fastcom104setup' is responsible for initializing the Fastcom ESCC-104 board. Specifically the code sets up the board chip registers such that it will operate in the correct configuration to read the IMU's data.

The block 'fastcom104read' is responsible for extracting the IMU's raw data from the Fastcom board buffer and transfer the raw data to Simulink to be processed.

The block 'fastcom104hook' is responsible for the proper handling of interrupt signals from the Fastcom board. Specifically the code recognizes the Fastcom 104 board's interrupt signals that ensure proper timing of data-acquisition by the CPU.

A.1.2 C Language Code

'fastcom104read.c' source code:

```
/*
Filename: fastcom104read.c
Created by: Tomoyuki Kato, Long Dam
Last edit: 11/6/2013

Description: This code will access the Fastcom ESCC104 data.
*/

#define          S_FUNCTION_LEVEL        2
#undef           S_FUNCTION_NAME
#define          S_FUNCTION_NAME         fastcom104read

#include         <stddef.h>
#include         <stdlib.h>

#include         "simstruc.h"

#ifdef           MATLAB_MEX_FILE
#include         "mex.h"
#endif

#ifndef          MATLAB_MEX_FILE
#include         <windows.h>
#include         "xpctarget.h"
#include         "fastcom104.h"

#endif

int framecount = 0;
int framelost  = 0;

short int data[128];             // For temporary storage of FIFO data

#define NO_I_WORKS               (1)
#define NO_R_WORKS               (0)
static char_T msg[256];
#define REG_I_IND                                   (0)

#define NUMBER_OF_ARGS           (1)
#define ADDRESS_ARG              ssGetSFcnParam(S,0)

/* S-Function */

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n%d arguments are expected\n",
                NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if( !ssSetNumOutputPorts(S, 4) )return;
    ssSetOutputPortWidth(S, 0, 33);
    ssSetOutputPortWidth(S, 1, 1);
    ssSetOutputPortWidth(S, 2, 1);
    ssSetOutputPortWidth(S, 3, 1);
    ssSetOutputPortDataType(S, 0, SS_INT32);
    ssSetOutputPortDataType(S, 1, SS_INT32);
    ssSetOutputPortDataType(S, 2, SS_INT32);
    ssSetOutputPortDataType(S, 3, SS_INT32);

    if( !ssSetNumInputPorts(S, 0) )return;

    ssSetNumSampleTimes(S, 1);

    ssSetSimStateCompliance( S, HAS_NO_SIM_STATE );

    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

}

static void mdlInitializeSampleTimes(SimStruct *S)
{
        ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
        ssSetOffsetTime(S, 0, 0);
}

#define MDL_START
static void mdlStart(SimStruct *S)
{
}

static void mdlOutputs(SimStruct *S, int_T tid)
```

```
{
        #ifndef MATLAB_MEX_FILE
            int sab                         = (uint16_T)mxGetPr(ADDRESS_ARG)[0];            //
                SAB82532 Registers
            int_T *OPtr                     = ssGetOutputPortSignal(S,0);                   // Pointer
               to the output signal
            int_T *rbcl         = ssGetOutputPortSignal(S,1);
            int_T *rbch         = ssGetOutputPortSignal(S,2);
            int_T *rsta         = ssGetOutputPortSignal(S,3);
            int count                       = 0;
                                // Counter
            int c;
                                                        // Temp. var.
            int i;
            int j;

            rbcl[0] = xpcInpB((uint16_T)(sab + XBCL));
            rbch[0] = xpcInpB((uint16_T)(sab + XBCH));

                    while (count < (rbcl[0]-1)) // start FIFO read (32bytes deep + count)
                    {
                                    count++;
                                    c = xpcInpB((uint16_T)(sab + DATA));
                                    OPtr[count] = c;
                    }

//          rbcl[0] = xpcInpB((uint16_T)(sab + XBCL));
//          rbch[0] = xpcInpB((uint16_T)(sab + XBCH));
            rsta[0] = xpcInpB((uint16_T)(sab + RSTA));

            OPtr[0] = count;


                    xpcOutpB((unsigned short)(sab + CMDR),(uint8_T)(CMDRRMC));              // Receive
                           message complete

                    /* Convert data word content to double */
//                  for (i = 0; i < 13; i++)
//                  {
//                          OPtr[i] = (data[2*i] + (data[(2*i)+1] << 8));
//                  }
//                  OPtr[13] = 0;
//                  OPtr[14] = 0;
//                  OPtr[15] = 0;

        #endif
}

static void mdlTerminate(SimStruct *S)
{
}

#ifdef MATLAB_MEX_FILE  /* Is this file being compiled as a MEX-file? */
#include "simulink.c"   /* Mex glue */
#else
#include "cg_sfun.h"    /* Code generation glue */
#endif
```

'fastcomm104setup.c' source code:

```
/*
Filename: fastcomm104setup.c
Created by: Tomoyuki Kato, Long Dam
Last edit: 11/6/2013

Description: This code will initialize the Fastcom ESCC-PCI device for operation.
*/

#define         S_FUNCTION_LEVEL        2
#undef          S_FUNCTION_NAME
#define         S_FUNCTION_NAME         fastcom104setup

#include        <stddef.h>
#include        <stdlib.h>

#include        "simstruc.h"

#ifdef          MATLAB_MEX_FILE
#include        "mex.h"
#endif

#ifndef         MATLAB_MEX_FILE
#include        <windows.h>
#include        "xpctarget.h"
#include        "fastcom104.h"
#endif

#define NO_I_WORKS              (1)
#define NO_R_WORKS              (0)
static char_T msg[256];
#define REG_I_IND                                       (0)

#define NUMBER_OF_ARGS          (1)
#define ADDRESS_ARG             ssGetSFcnParam(S,0)

/* S-Function */
```

```
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n%d arguments are expected\n",
            NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if( !ssSetNumOutputPorts(S, 0) )return;

    if( !ssSetNumInputPorts(S, 0) )return;

    ssSetNumSampleTimes(S, 1);

    ssSetSimStateCompliance( S, HAS_NO_SIM_STATE );

    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, SS_OPTION_DISALLOW_CONSTANT_SAMPLE_TIME | SS_OPTION_EXCEPTION_FREE_CODE );
} // End mdlInitializeSizes

static void mdlInitializeSampleTimes(SimStruct *S)
{
        ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
        ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);
} // End mdlInitializeSampleTimes

#define MDL_START
static void mdlStart(SimStruct *S)
{
  #ifndef MATLAB_MEX_FILE

            uint16_T base;

            base = (uint16_T)mxGetPr(ADDRESS_ARG)[0];            // The base address of the Fastcom
                 104 board.

        /* Initialize SAB 82532 Registers*/

//              xpcOutpB((uint16_T)(base + CCR1),(uint8_T)(0x10)); // CCR1
                xpcOutpB((uint16_T)(base + CCR0),(uint8_T)(0x80)); // CCR0 (Setting HDLC mode)
        xpcOutpB((uint16_T)(base + CCR1),(uint8_T)(0x10)); // CCR1
                xpcOutpB((uint16_T)(base + CCR2),(uint8_T)(0x18)); // CCR2
                xpcOutpB((uint16_T)(base + BRG),(uint8_T)(0x00)); // BGR
        xpcOutpB((uint16_T)(base + PRE),(uint8_T)(0x00)); // Preamble (PRE)
        xpcOutpB((uint16_T)(base + MODE),(uint8_T)(0x88)); // Mode (MODE) (Transparent mode 0,
             receiver active)
        xpcOutpB((uint16_T)(base + TIMR),(uint8_T)(0x1f)); // Timer (Timer)
//          xpcOutpB((uint16_T)(base + XAD1),(uint8_T)(0x00)); // Transmit address (XAD1)
//          xpcOutpB((uint16_T)(base + XAD2),(uint8_T)(0x00)); // Transmit address (XAD2)
//          xpcOutpB((uint16_T)(base + RAH1),(uint8_T)(0x00)); // Receive address high (RAH1) (
     Transparent mode 0, so address doesn't matter)
//          xpcOutpB((uint16_T)(base + RAH2),(uint8_T)(0x00)); // Receive address high (RAH2) (
     Transparent mode 0, so address doesn't matter)
//          xpcOutpB((uint16_T)(base + RAL1),(uint8_T)(0x00)); // Receive address low (RAL1) (
     Transparent mode 0, so address doesn't matter)
//          xpcOutpB((uint16_T)(base + RAL2),(uint8_T)(0x00)); // Receive address low (RAL2) (
     Transparent mode 0, so address doesn't matter)
          xpcOutpB((uint16_T)(base + XAD1),(uint8_T)(0x00)); // Transmit address (XAD1)
          xpcOutpB((uint16_T)(base + XAD2),(uint8_T)(0x00)); // Transmit address (XAD2)
          xpcOutpB((uint16_T)(base + RAH1),(uint8_T)(0x00)); // Receive address high (RAH1) (
               Transparent mode 0, so address doesn't matter)
          xpcOutpB((uint16_T)(base + RAH2),(uint8_T)(0x00)); // Receive address high (RAH2) (
               Transparent mode 0, so address doesn't matter)
          xpcOutpB((uint16_T)(base + RAL1),(uint8_T)(0x00)); // Receive address low (RAL1) (
               Transparent mode 0, so address doesn't matter)
          xpcOutpB((uint16_T)(base + RAL2),(uint8_T)(0x00)); // Receive address low (RAL2) (
               Transparent mode 0, so address doesn't matter)
          xpcOutpB((uint16_T)(base + XBCL),(uint8_T)(0x00)); // Transmit byte count low (XBCL)
          xpcOutpB((uint16_T)(base + XBCH),(uint8_T)(0x00)); // Transmit byte count high (XBCH)
//          xpcOutpB((uint16_T)(base + CCR3),(uint8_T)(0x04)); // CCR3 (Activate CRC for receiver)
          xpcOutpB((uint16_T)(base + CCR3),(uint8_T)(0x00)); // CCR3 (Activate CRC for receiver)
          xpcOutpB((uint16_T)(base + RLCR),(uint8_T)(0x00)); // RLCR
          xpcOutpB((uint16_T)(base + IVA),(uint8_T)(0x00)); // Interrupt vector address (IVA)
//          xpcOutpB((uint16_T)(base + IPC),(uint8_T)(0x83)); // Interrupt port configuration (IPC)
       (Masked interrupts still visible, INT pin push-pull drain output)
          xpcOutpB((uint16_T)(base + IPC),(uint8_T)(0x03)); // Interrupt port configuration (IPC) (
               Masked interrupts still visible, INT pin push-pull drain output)
          xpcOutpB((uint16_T)(base + IMR0),(uint8_T)(0x00)); // Interrupt mask register (IMR0)
          xpcOutpB((uint16_T)(base + IMR1),(uint8_T)(0xff)); // Interrupt mask register (IMR1)
          xpcOutpB((uint16_T)(base + PVR),(uint8_T)(0x00)); // Port value register (PVR)
          xpcOutpB((uint16_T)(base + PIM),(uint8_T)(0xff)); // Port interrupt mask (PIM)
          xpcOutpB((uint16_T)(base + PCR),(uint8_T)(0xe0)); // Port configuration register (PCR)
//              xpcOutpB((uint16_T)(base + CCR0),(uint8_T)(0x80)); // CCR0 (Power Up)

//              xpcOutpB((uint16_T)(base + CMDR),(uint8_T)(0x41)); // Reset the TFIFO and the
     RFIFO to initialize interrupts

     #endif
} // End mdlStart

static void mdlOutputs(SimStruct *S, int_T tid)
{
#ifndef MATLAB_MEX_FILE

#endif
```

```
} // End mdlOutputs

static void mdlTerminate(SimStruct *S)
{
#ifndef MATLAB_MEX_FILE
//      uint16_T base           = (uint16_T)(0x340);
//
//      xpcOutpB((uint16_T)(base + IMR0),(uint8_T)(0xff)); // Mask all interrupts
//      xpcOutpB((uint16_T)(base + CCR0),(uint8_T)(0x00)); // Power down Fastcom ESCC-PCI*/
#endif
} // End mdlTerminate

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

'xpcfastcom104hook.c' source code:

```
/*
Filename: xpcfastcom104hook.c
Created by: Tomoyuki Kato, Long Dam
Last edit: 11/6/2013

Description: Contains xPC target hook functions designed to service interrupts from the Fastcomm
     ESCC-104 board.
*/

#ifndef __XPCFASTCOM104HOOK_C__
#define __XPCFASTCOM104HOOK_C__

#include <conio.h>
#include <windows.h>
#include "xpctarget.h"

/* Function Prototypes*/

int  __cdecl xpcfastcom104prehook(xpcPCIDevice *pci);
void __cdecl xpcfastcom104start(xpcPCIDevice *pci);
void __cdecl xpcfastcom104stop(xpcPCIDevice *pci);

/* Function Definitions */

// Prehook Function
    ------------------------------------------------------------------------------------------------------------------

int _cdecl xpcfastcom104prehook(xpcPCIDevice *pci)
{
        uint16_T sab  = (uint16_T)(pci->BaseAddress[0]);                    // SAB82532 Registers
        uint8_T gis;                                                                                    //
              Variable for SAB chip global interrupt status register
        uint8_T isr0;

        /* Read the SAB82532 interrupt register */

        gis             = xpcInpB((uint16_T)(sab + 24));        // Read the global status
             interrupt register.
//      printf(("%x\n"),gis); // Debug

        // The interrupts will automatically reset upon read of the ISR.

        if (gis != 0)
        {
                isr0 = xpcInpB((uint16_T)(sab + 26));
//              printf("%x\n",isr0);

                if ((isr0 & 0x01) == 0x01)
                {
                        //printf("%x\n",0x01);
                        return XPC_RUN_ISR;
                }

                if ((isr0 & 0x80) == 0x80)
                {
                        //printf("%x\n",0x80);
                        return XPC_RUN_ISR;
                }
//              if ((isr0 & 0x40) == 0x40)
//              {
//                      //printf("%x\n",0x40);
//                      return XPC_RUN_ISR;
//              }
                return XPC_RUN_ISR;
        }

//      printf("%u\n",456);
        return XPC_DROP_ISR;
}

// Start Function
    ------------------------------------------------------------------------------------------------------------------
```

```
void _cdecl xpcfastcom104start(xpcPCIDevice *pci)
{
        uint16_T sab    = (uint16_T)(pci->BaseAddress[0]);  // SAB82532 Registers

        xpcOutpB((uint16_T)(sab),(uint8_T)(0x41)); // Reset the TFIFO and the RFIFO to initialize
             interrupts
        return;
}

// Stop Function
-----------------------------------------------------------------------------------------------------------------------

void _cdecl xpcfastcom104stop(xpcPCIDevice *pci)
{
        uint16_T sab = (uint16_T)(pci->BaseAddress[0]);
        //printf(("%u\n"),33333);
        xpcOutpB((uint16_T)(sab + 26),(uint8_T)(0xff)); // Mask all interrupts
        xpcOutpB((uint16_T)(sab + 12),(uint8_T)(0x00)); // Power down Fastcom ESCC-PCI
        return;
}

#endif
```

A.2 Reaction Wheel Code

The software for the reaction wheels is responsible for both commanding the motors and reading the motor speed data. The source code is written in the C language and is compiled and executed in the Simulink environment. Subsection A.2.1 contains the Simulink block diagrams and subsection A.2.2 contains the source code.

The code operates the Diamond Systems Emerald MM-8P board which communicates with the reaction wheel motors. The reaction wheel code is nearly identical to that of the Emerald MM-8 board code that was pre-developed by Mathworks available with the purchase of the MATLAB xPC system, with a few differences. The only change made with the MM-8P board code is the block 'sersetupemeraldmm8p', which is responsible for initializing the chip registers for each of the 8 serial ports. The source code of the block 'sersetupemeraldmm8p' is shown in subsection A.2.2.

A.2.1 Simulink Block

Shown in Figure A.3 is the Simulink block for the reaction wheel driver code.

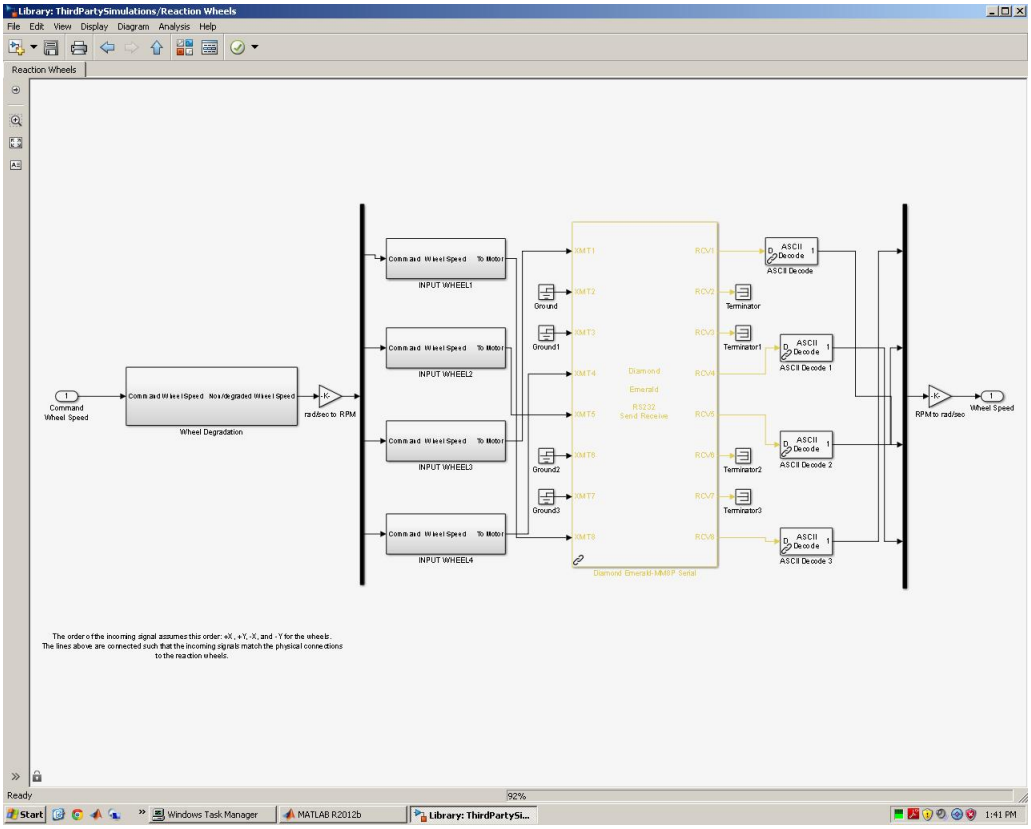

Figure A.3: Reaction Wheel Driver Code, Simulink Diagram.

Figure A.4 shows the first level of the reaction wheel driver code.

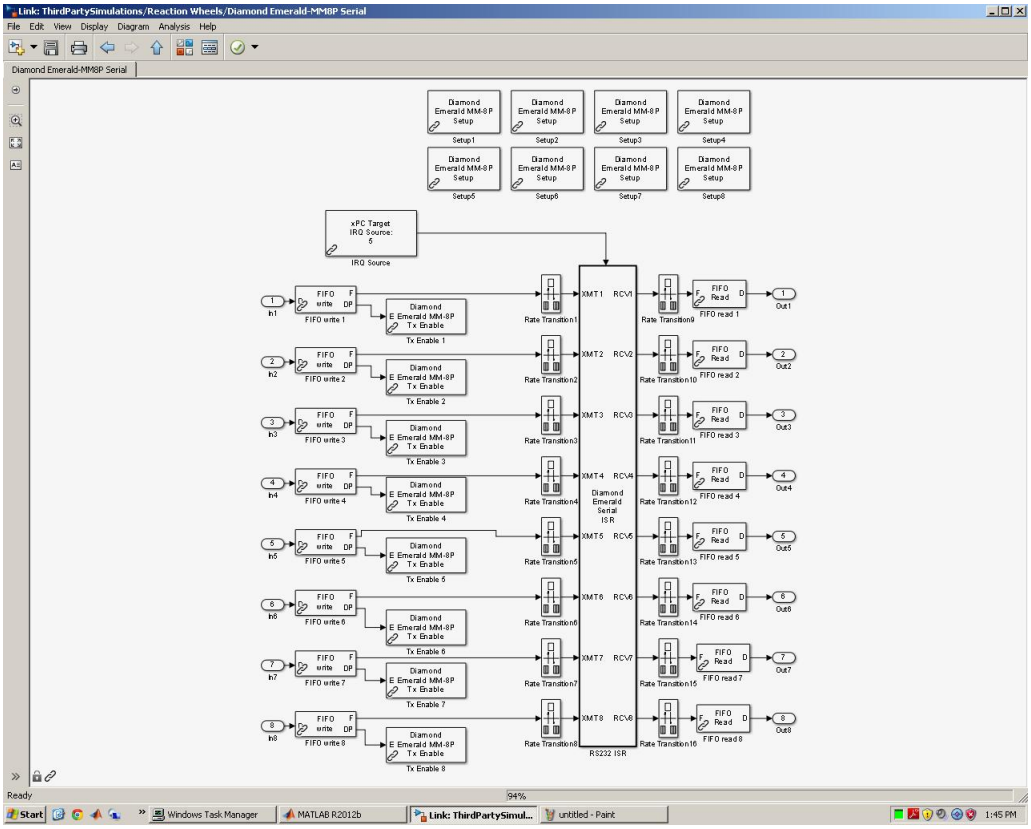

Figure A.4: Reaction Wheel Driver Code, Simulink Diagram, First Level.

Figure A.5 shows the second level of the reaction wheel driver code.

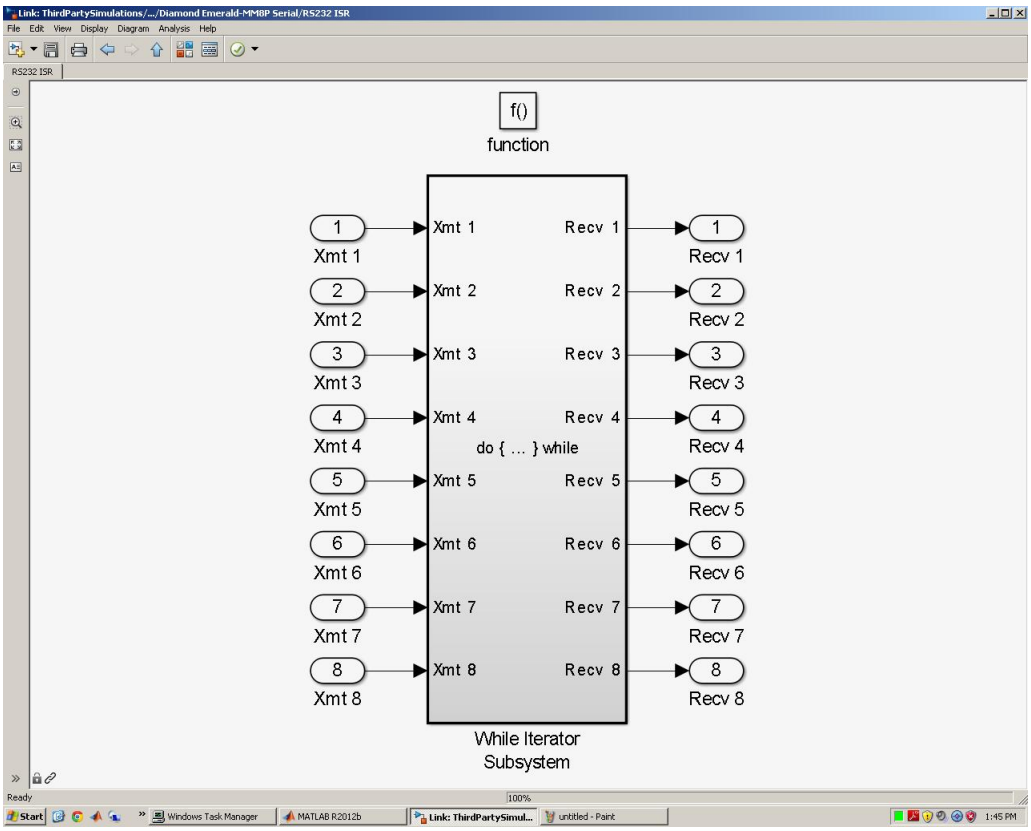

Figure A.5: Reaction Wheel Driver Code, Simulink Diagram, Second Level.

Figure A.6 shows the third level of the reaction wheel driver code.

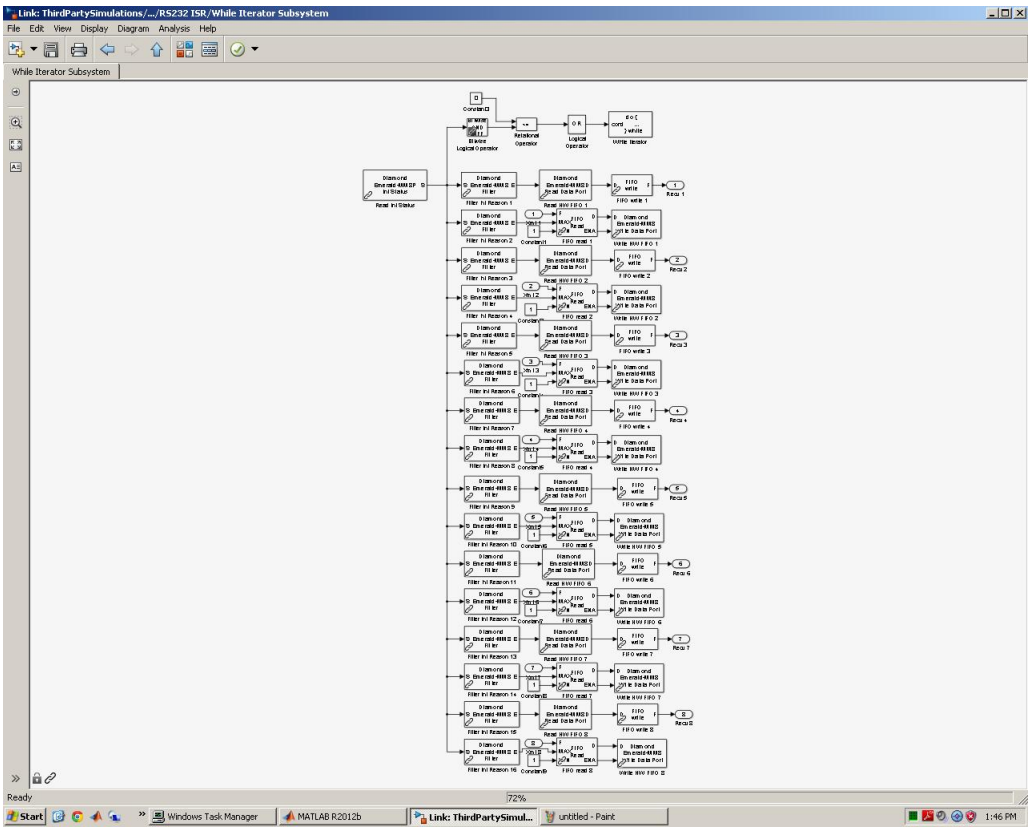

Figure A.6: Reaction Wheel Driver Code, Simulink Diagram, Third Level.

A.2.2 C Language Code

```
/*
Filename: sersetupemeraldmm8p.c
Created by: Tomoyuki Kato
Last edit: 4/11/2013

Description: Setup block for the Diamond Systems Emerald MM-8P board.
*/

#define          S_FUNCTION_LEVEL          2
#undef           S_FUNCTION_NAME
#define          S_FUNCTION_NAME           sersetupemeraldmm8p

#define     DEBUG_PRINTFS    0

#include         <stddef.h>
#include         <stdlib.h>

#include         "simstruc.h"

#ifdef           MATLAB_MEX_FILE
#include         "mex.h"
#endif

#ifndef          MATLAB_MEX_FILE
#include         <windows.h>
#include         "xpcimports.h"
#include         "serialdefines.h"

// Emerald MM8P UART drivers have 230400 max baud rate
static int32_T basedivisors[14] =
{
          1,    // 115200
          2,    // 57600
          3,    // 38400
          6,    // 19200
         12,    // 9600
         24,    // 4800
         48,    // 2400
         96,    // 1200
        192,    // 600
        384,    // 300
```

```
};

#endif

/* Input Arguments */
#define NUMBER_OF_ARGS          (8)
#define ADDR_ARG                ssGetSFcnParam(S,0)
#define IRQ_ARG                 ssGetSFcnParam(S,1)
#define BAUD_ARG                ssGetSFcnParam(S,2)
#define WIDTH_ARG               ssGetSFcnParam(S,3)
#define NSTOP_ARG               ssGetSFcnParam(S,4)
#define PARITY_ARG              ssGetSFcnParam(S,5)
#define CTSMODE_ARG             ssGetSFcnParam(S,6)
#define RLEVEL_ARG              ssGetSFcnParam(S,7)

#define NO_I_WORKS              (2)
#define BASE_I_IND              (0)
#define FIRST_I_IND             (1)

#define NO_R_WORKS              (0)

// Definitions for the FCR TXFIFO trigger levels
#define FCRTX8  0
#define FCRTX16 0x10
#define FCRTX32 0x20
#define FCRTX56 0x30

static char_T msg[256];

static void mdlInitializeSizes(SimStruct *S)
{
    int i;

    ssSetNumSFcnParams(S, NUMBER_OF_ARGS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        sprintf(msg,"Wrong number of input arguments passed.\n%d arguments are expected\n",
            NUMBER_OF_ARGS);
        ssSetErrorStatus(S,msg);
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if( !ssSetNumOutputPorts(S, 0) )return;

    if( !ssSetNumInputPorts(S, 0) )return;

    ssSetNumSampleTimes(S, 1);

    ssSetSimStateCompliance( S, HAS_NO_SIM_STATE );

    ssSetNumRWork(S, NO_R_WORKS);
    ssSetNumIWork(S, NO_I_WORKS);
    ssSetNumPWork(S, 0);

    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    for( i = 0 ; i < NUMBER_OF_ARGS ; i++ )
    {
        ssSetSFcnParamTunable(S,i,0);  /* None of the parameters are tunable */
    }

    ssSetOptions(S, SS_OPTION_DISALLOW_CONSTANT_SAMPLE_TIME | SS_OPTION_EXCEPTION_FREE_CODE );
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);
}

#define MDL_START
static void mdlStart(SimStruct *S)
{

#ifndef MATLAB_MEX_FILE

    int_T addr;
    int port;
    int bauddiv;
    int i;
    int lcrtemp;
    int configbase;
    int irq;

    // Save the base address for mdlterminate
    addr = (int_T)mxGetPr(ADDR_ARG)[0];
    irq = (int)mxGetPr(IRQ_ARG)[0];
//      printf("addr = 0x%x, addr>>3 = 0x%x, irq = 0x%x\n", addr, addr>>3, irq ); // DEBUG

    // Derive the configuration base address and port number from base.
    // This assumes that addr is (configbase + (port+1)*8) and the last port
    // overlaps the next configbase address.
    configbase = (addr - 1) & ~0x3f;
    port = (((addr - 1) >> 3) & 7);                                 // 0 based port number, range
        [0,7]

    xpcOutpB( (uint16_T)(configbase), port );                   // set address pointer to this port
        UART address
    xpcOutpB( (uint16_T)(configbase+1), addr>>3 );              // Set this port address to addr.
    xpcOutpB( (uint16_T)(configbase), port+8 );                 // set address pointer to this port
        IRQ number
    xpcOutpB( (uint16_T)(configbase+1), irq );                  // Set this port irq number to irq.
    xpcOutpB( (uint16_T)(configbase), 0x80 );                   // Enable the port
```

```
//      printf("%x\n",xpcInpB((uint16_T)(configbase + 2))); // DEBUG

    // Debug readback
//      {
//       int testaddr, testirq;
//       xpcOutpB( (uint16_T)(configbase), port );
//       testaddr = (xpcInpB( (uint16_T)(configbase+1) ) & 0xff) << 3;
//       xpcOutpB( (uint16_T)(configbase), port+8 );
//       testirq  = xpcInpB( (uint16_T)(configbase+1) ) & 0xff;
//       printf("Readback: addr = 0x%x, irq = %x\n", testaddr, testirq );
//      }
    // End debug readback

    ssSetIWorkValue(S, BASE_I_IND, addr);                          // Save the port address
    bauddiv = basedivisors[ (int)(mxGetPr(BAUD_ARG)[0] - 1) ];  // Select baud rate divisor
//      printf("%d\n",bauddiv); // DEBUG

    // Set the DLAB bit so we can get to the baud rate divisor
    // and the options register
    lcrtemp = xpcInpB( (uint16_T)(addr + LCR) );
    xpcOutpB( (uint16_T)(addr + LCR), (uint8_T)(lcrtemp | LCRDLAB) );

    // Set the baud rate divisor, assumes 1x multiplier
    xpcOutpB( (uint16_T)(addr + DLSB), (uint8_T)(bauddiv & 0xff) );
    xpcOutpB( (uint16_T)(addr + DMSB), (uint8_T)((bauddiv >> 8) & 0xff) );
//      printf("DLSB: %x\n DMSB: %x\n",xpcInpB((uint16_T)(addr+DLSB)),xpcInpB((uint16_T)(addr+DMSB)
    )); // DEBUG
    // Clear the DLAB bit
    xpcOutpB( (ushort_T)(addr + LCR), (uint8_T)lcrtemp );

    // Construct the contents of the LCR register.
    lcrtemp = 0;

    // Determine the word length
    switch( (int)(mxGetPr(WIDTH_ARG)[0]) )
    {
      case 1:
        i = LCR5BIT;
        break;
      case 2:
        i = LCR6BIT;
        break;
      case 3:
        i = LCR7BIT;
        break;
      case 4:
        i = LCR8BIT;
        break;
    }
    lcrtemp |= i;

    if( (int)(mxGetPr(NSTOP_ARG)[0]) == 2 )  // If 2 stop bits are chosen,
        lcrtemp |= LCRSTOP;

    // Parity selection involves 3 bits in the LCR
    i = 0;
    switch( (int)(mxGetPr(PARITY_ARG)[0] ) )
    {
      case 1:   // no parity
        i = 0;
        break;
      case 2:   // even parity
        i = LCRPARITY | LCREVEN;
        break;
      case 3:   // odd parity
        i = LCRPARITY;
        break;
      case 4:   // mark parity
        i = LCRPARITY | LCRSTICK;
        break;
      case 5:   // space parity
        i = LCRPARITY | LCREVEN | LCRSTICK;
        break;
    }
    lcrtemp |= i;

    xpcOutpB( (uint16_T)(addr + LCR), (uint8_T)lcrtemp );

    // Set the RFIFO trigger level.
    switch( (int)(mxGetPr(RLEVEL_ARG)[0] ) )
    {
        case 1:  // interrupt at 1 byte
          i = FCRONE;
          break;
        case 2:  // interrupt at quarter full (4 or 16)
          i = FCRQUARTER;
          break;
        case 3:  // interrupt at half full   (8 or 32)
          i = FCRHALF;
          break;
        case 4:  // interrupt at almost full (14 or 56)
          i = FCRFULL;
          break;
    }

    i |= FCREBL | FCRRCLR | FCRTCLR;                // Enable and clear both FIFOs, set TX FIFO
        trigger to 8 characters

    // Set the FCR
    xpcOutpB( (uint16_T)(addr + FCR), (uint8_T)i );          // Load configuration onto FCR

    i = MCROUT2;  // On the baseboard UARTS, OUT2 is an extra interrupt mask!
    if( (int)(mxGetPr(CTSMODE_ARG)[0]) == 1 )
    {
        i |= MCRAFE | MCRRTS;  // Auto RTS/CTS mode
    }
```

```
        // Defaults to DTR, RTS, OUT1, OUT2 all high, bits = 1 sets outputs high
        xpcOutpB( (uint16_T)(addr + MCR), (uint8_T)i );

        // clear power modes,
        // but wait for mdlOutputs to Enable receive interrupts
        xpcOutpB( (ushort_T)(addr + IER), 0 );
        ssSetIWorkValue(S, FIRST_I_IND, 1);

        // Transmit interrupt isn't enabled yet, but is delayed until there
        // are characters available to be sent.
    #endif

    }

    static void mdlOutputs(SimStruct *S, int_T tid)
    {

    #ifndef MATLAB_MEX_FILE
        int addr = ssGetIWorkValue( S, BASE_I_IND );
        int first = ssGetIWorkValue( S, FIRST_I_IND );
        int i;

        if( first == 1 )
        {
            // Since TX enable block may have executed first, read ier and modify.
            int ier = xpcInpB( (ushort_T)(addr + IER) );
            // Flush the hardware fifo on startup.
            i = 0;
            while( i++ < 65 && xpcInpB( (unsigned short)(addr + LSR) ) & LSRDR )
            {
                // Read and discard the data.
                xpcInpB( (unsigned short)(addr + DATA) );
            }

            // Now enable the interrupt.
            xpcOutpB( (ushort_T)(addr + IER), ier | IERRCV);
            ssSetIWorkValue( S, FIRST_I_IND, 0 );
        }

    #endif

    }

    static void mdlTerminate(SimStruct *S)
    {
    #ifndef MATLAB_MEX_FILE
        int addr = ssGetIWorkValue(S, BASE_I_IND);
        //int uarttype = ssGetIWorkValue(S, TYPE_I_IND);

        if( addr == 0 )  // This UART is not configured
            return;

        // Disable interrupts.
        xpcOutpB( (ushort_T)(addr + IER), 0 );

            // Flush the transmit and receive fifos so the next start will be clean
            xpcOutpB( (ushort_T)(addr + FCR), FCREBL | FCRRCLR | FCRTCLR );
    #endif
    }

    #ifdef MATLAB_MEX_FILE  /* Is this file being compiled as a MEX-file? */
    #include "simulink.c"   /* Mex glue */
    #else
    #include "cg_sfun.h"    /* Code generation glue */
    #endif
```

A.3 Motion Test Scripts

This section contains MATLAB scripts which initialize the Simulink motion test files. There are four scripts:

- rxnwhl_calibration_setup.m
- fsfbsetup.m
- ndmracsetup.m
- aofsetup.m

```
% Filename: rxnwhl_calibration_setup.m
% Created by: Tomoyuki Kato, Long Dam
% Last edit: 12/14/2013
%
% Description: This m-file script will initialize the initial wheel speed
% values for the Simulink file 'RXNWHL_CALIBRATION.mdl'.

%% Variables

% Desired test case (1 = motion test, 2 = sine wave test)
test = 2;

% Motion test initial speeds
whl0 = 1500;
% whl01 = 1500;

% Sinewave test initial speeds
% sine = [10; -100; -25; -10]*60/2/pi;
sine = 500*ones(1,4);

%% Get parameters

id  = tg.getparamid('Parameter Command/Initial Speed','Value');
id1 = tg.getparamid('Parameter Command1/Initial Speed','Value');
id2 = tg.getparamid('Parameter Command2/Initial Speed','Value');
id3 = tg.getparamid('Parameter Command3/Initial Speed','Value');


%% Set parameters

if test == 1
    tg.setparam(id,whl0);
    tg.setparam(id1,whl0);
    tg.setparam(id2,whl0);
    tg.setparam(id3,whl0);
elseif test == 2
    tg.setparam(id,sine(1));
    tg.setparam(id1,sine(2));
    tg.setparam(id2,sine(3));
    tg.setparam(id3,sine(4));
else
    disp('Do nothing');
end

disp(tg.getparam(id))
disp(tg.getparam(id1))
disp(tg.getparam(id2))
disp(tg.getparam(id3))
```

Full-State Feedback Test Setup

```
% Filename: fsfbsetup.m
% Created by: Tomoyuki Kato
% Last edit: 12/5/2013
%
% Description: This m-file script will initialize the properties of all the
% blocks in the Simulink file 'FSFB_SIM_IMU.mdl'.

%% Variables

testcase = 1;               % 1: system ID 2: Solidworks 3: degraded wheel
cont = 2;                   % 1: FSFB 2: LQR + set-point 3. Optimal feedback + set-point

if cont == 1
    disp('Full-State Feedback Control\n');
elseif cont == 2
    disp('LQR + Set-Point Control\n');
else
    disp('Optimal Feedback + Set-Point Control\n');
end

% Sample rates
imudt   = 1/400;            % IMU sample rate (sec)
dt      = 1/50;             % Simulation sample rate (sec)

% Initial conditions
q0      = [0 0 0 1]';                           % Quaternion initial conditions
w0      = [0 0 0]';                             % Body rate initial conditions (rad/sec)
whl0vect = [1500 1500 1500 1500]*(2*pi/60);  % Initial wheel speed vector (rad/sec)

% Reaction Wheel Performance
zetarw  = 1.1;          % Damping ratio of degraded wheel
wnrw    = 1;            % Natural frequency of degraded wheel (rad/sec)

% Mass Property
Jsysid = [.6071     0.0266     0.0149;
          0.0266    .6560      0.001;
          0.0149     0.001     .6376]; % System ID estimate
Jest = [.5485    -.0002   -.0003;
        -.0002   .5884    -.0005;
        -.0003   -.0005    .6456]; % Solidworks estimate
% Jsysid = [.6071+0.035421     0.0266      0.0149;
%           0.0266    .6560+0.031053       0.001;
```

```
%           0.0149      0.001      .6376+0.027591];  % System ID upper bound
% Jsysid = [.6071-0.035421      0.0266      0.0149;
%           0.0266     .6560-0.031053       0.001;
%           0.0149      0.001      .6376-0.027591];   % System ID Lower bound

% Continuous-time state-space model of 4th wheel
AWc = [0 1;-wnrw^2,-2*zetarw*wnrw];
BWc = [0;wnrw^2];
CWc = [1,0];
DWc = [0];
sysC = ss(AWc,BWc,CWc,DWc);

% Convert to discrete-time state-space model
sysD = c2d(sysC,dt);
A4 = sysD.a;
B4 = sysD.b;
C4 = sysD.c;
D4 = sysD.d;

% Wheel disable command time
whldi = 39.5;

%% Set-point gains

% State-space matrices of the linearized spacecraft EOM
a = -((Jest(3,3)-Jest(2,2))/Jest(1,1));
b = -((Jest(1,1)-Jest(3,3))/Jest(2,2));
c = -((Jest(1,1)-Jest(2,2))/Jest(3,3));
hp = .5;
hm = -.5;
eul = [0 0 -pi/12];
qd = e2q(eul);

% Linearize spacecraft EOM about a 15 deg. yaw rotation
A = zeros(6,6);
    A(4,1) = 0.5*qd(4);
    A(4,2) = -0.5*qd(3);
    A(5,1) = 0.5*qd(4);
    A(5,2) = 0.5*qd(3);
    A(6,3) = 0.5*qd(4);
%     A(7,3) = -0.5*qd(3);
B = zeros(6,3);
    B(1,1) = 1/Jest(1,1);
    B(2,2) = 1/Jest(2,2);
    B(3,3) = 1/Jest(3,3);
C = eye(6);
D = zeros(6,3);
% G and H matrices to make input dimension equal to output dimension
G = zeros(3,6);
    G(1,4) = 1;
    G(2,5) = 1;
    G(3,6) = 1.1;
H = zeros(3,3);

% Form a square matrix of state-space matrices and invert
P = [A B;G H];
dim = size(P);
if dim(1) == dim(2)
    P1 = inv(P);
else
    P1 = pinv(P);
end
[row,col] = size(B);

ob = rank(obsv(A,C));                   % Check for observability
wo = rank(ctrb(A,B));                   % Check for controllability
ev = [-1 -0.7 -1.1 -0.1 -0.5 -0.8]';    % Case 1 and 2
% ev = [-.5 -1 -1.5 -2 -2.5 -3]';   % Case 3
kopt = place(A,B,ev);                   % Optimal feedback-gain

F = P1(1:6,end-5:end);                  % Set-point feedforward gain F
N = P1(7:end,end-5:end);                % Set-point feedforward gain N
feed = N + kopt*F;                      % Total feedforward term
feedfwd = feed*[0;0;0;0;0;.1305];

%% LQR
Q = eye(6);
Q(1,1) = Q(1,1)*.05;
Q(2,2) = Q(2,2)*.05;
Q(3,3) = Q(3,3)*.05;
Q(4,4) = Q(4,4)*.001;
Q(5,5) = Q(5,5)*.001;
Q(6,6) = Q(6,6)*.05;
R = eye(3);
klqr = lqr(A,B,Q,R);

%% Get parameters

imublock = 'IMU LN-200';          % Directory to 'IMU LN-200' subsystem
rxnwhl = 'Reaction Wheels';       % Directory to 'Reaction Wheels' subsystem
motalg = 'Motion Algorithm';      % Directory to 'Motion Algorithm' subsystem

whl0id = tg.getparamid('Motion Algorithm/Discrete-Time Integrator','InitialCondition');
jid = tg.getparamid('Motion Algorithm/Inertia','Value');
% rwid = tg.getparamid('Motion Algorithm/Manual Switch','Value');
kopid = tg.getparamid('Motion Algorithm/optgain','Value');
feedid = tg.getparamid('Motion Algorithm/feedfwd','Value');
lqrid = tg.getparamid('Motion Algorithm/lqr','Value');
contid = tg.getparamid('Motion Algorithm/controller','Value');

%% Set parameters

% Set up initial wheel speeds

tg.setparam(whl0id,whl0vect)          % Set initial wheel speed
tg.setparam(kopid,kopt);              % Set optimal feedback gain
tg.setparam(feedid,feed);             % Set feedforward gain
```

```
tg.setparam(lqrid,klqr);              % Set LQR gain

% Set up inertia tensor and choose nominal or degraded reaction wheel
if testcase == 1
    tg.setparam(jid,Jsysid);
    tg.setparam(contid,cont);
    disp('Case 1: System ID Estimate');
else
    tg.setparam(jid,Jest);
    tg.setparam(contid,cont);
    if testcase == 3
%         tg.setparam(rwid,0);
        disp('Case 3: Degraded Wheel');
    else
%         tg.setparam(rwid,1);
        disp('Case 2: Solidworks Estimate');
    end
end
```

NDMRAC Test Setup

```
% Filename: ndmracsetup.m
% Created by: Tomoyuki Kato
% Last edit: 5/22/2013
%
% Description: This m-file script will initialize the properties of all the
% blocks in the Simulink file 'AOF_SIM_IMU.mdl'.

%% Variables

testcase = 1;               % 1: perfect inertia 2: estimated 3: degraded wheel

file = input('Which gains would you like to load?\n','s');
load(file)
Ge0 = Ge0NDMRAC;
H1 = 0.1*eye(12);
H2 = 0.1*eye(4);
H3 = 100*eye(6);

imudt   = 1/400;            % IMU sample rate (sec)
dt      = 1/50;             % Simulation sample rate (sec)
q0      = [0 0 0 1]';       % Quaternion initial conditions
w0      = [0 0 0]';         % Body rate initial conditions (rad/sec)
whl0    = 100*2*pi/60;     % Initial wheel speed (rad/sec)
whl0vect = whl0*ones(1,4);  % Initial wheel speed vector (rad/sec)

zetarw  = 1.1;          % Damping ratio of degraded wheel
wnrw    = 1;            % Natural frequency of degraded wheel (rad/sec)

% Continuous-time state-space model of 4th wheel
AWc = [0 1;-wnrw^2,-2*zetarw*wnrw];
BWc = [0;wnrw^2];
CWc = [1,0];
DWc = [0];
sysC = ss(AWc,BWc,CWc,DWc);

% Convert to discrete-time state-space model
sysD = c2d(sysC,dt);
A4 = sysD.a;
B4 = sysD.b;
C4 = sysD.c;
D4 = sysD.d;

% Wheel disable command time
whldi = 39.5;

% Mass Property
Jsysid = [.6071     0.0266     0.0149;
          0.0266    .6560      0.001;
          0.0149     0.001     .6376]; % System ID estimate
Jest = [.5485    -.0002   -.0003;
        -.0002   .5884    -.0005;
        -.0003   -.0005    .6456]; % Solidworks estimate

%% Get parameters

errid = tg.getparamid('Motion Algorithm/NDMRAC/Adaptive Error Gain/gain_init','Value');
h3id = tg.getparamid('Motion Algorithm/NDMRAC/Adaptive Error Gain/H3','Value');
inpid = tg.getparamid('Motion Algorithm/NDMRAC/Adaptive Input Gain/gain_init','Value');
h2id = tg.getparamid('Motion Algorithm/NDMRAC/Adaptive Input Gain/H2','Value');
stateid = tg.getparamid('Motion Algorithm/NDMRAC/Adaptive State Gain/gain_init','Value');
h1id = tg.getparamid('Motion Algorithm/NDMRAC/Adaptive State Gain/H1','Value');
rwid = tg.getparamid('Reaction Wheels/Wheel Degradation/Manual Switch','CurrentSetting');
jid = tg.getparamid('Motion Algorithm/Inertia','Value');
rw0id = tg.getparamid('Motion Algorithm/Discrete-Time Integrator','InitialCondition');

%% Set parameters

tg.setparam(errid,Ge0);
tg.setparam(inpid,S220);
tg.setparam(stateid,S210);
tg.setparam(h1id,H1);
tg.setparam(h2id,H2);
tg.setparam(h3id,H3);
tg.setparam(rw0id,whl0vect);

if testcase == 1
```

```
    tg.setparam(rwid,1);
    tg.setparam(jid,Jsysid);
    disp('Case 1: System ID Estimate');
else
    tg.setparam(jid,Jest);
    if testcase == 2
        tg.setparam(rwid,1);
        disp('Case 2: Solidworks Estimate');
    else
        tg.setparam(rwid,0);
        disp('Case 3: Degraded Wheel');
    end
end
```

AOF Test Setup

```
% Filename: aofsetup.m
% Created by: Tomoyuki Kato
% Last edit: 11/14/2013
%
% Description: This m-file script will initialize the properties of all the
% blocks in the Simulink file 'AOF_SIM_IMU.mdl'.

%% Variables

testcase = 3;               % 1: system ID 2: Solidworks 3: degraded wheel

% load('AOFDATA_20130524.mat');
% Ge0 = reshape(out1(end,:),[6,6]);
% Ge0 = zeros(6,6);
file = input('Which gains would you like to load?\n','s');
load(file);
Ge0 = Ge0AOF;
H3 = 500*eye(6);

imudt   = 1/400;            % IMU sample rate (sec)
dt      = 1/50;             % Simulation sample rate (sec)
q0      = [0 0 0 1]';       % Quaternion initial conditions
w0      = [0 0 0]';         % Body rate initial conditions (rad/sec)
whl0    = 1500*2*pi/60;     % Initial wheel speed (rad/sec)
whl0vect = whl0*ones(1,4);  % Initial wheel speed vector (rad/sec)

zetarw  = 1.1;          % Damping ratio of degraded wheel
wnrw    = 1;            % Natural frequency of degraded wheel (rad/sec)

% Mass Property
Jsysid = [.6071     0.0266     0.0149;
          0.0266    .6560      0.001;
          0.0149     0.001     .6376]; % System ID estimate
Jest = [.5485    -.0002   -.0003;
        -.0002   .5884    -.0005;
        -.0003   -.0005    .6456]; % Solidworks estimate

% Continuous-time state-space model of 4th wheel
AWc = [0 1;-wnrw^2,-2*zetarw*wnrw];
BWc = [0;wnrw^2];
CWc = [1,0];
DWc = [0];
sysC = ss(AWc,BWc,CWc,DWc);

% Convert to discrete-time state-space model
sysD = c2d(sysC,dt);
A4 = sysD.a;
B4 = sysD.b;
C4 = sysD.c;
D4 = sysD.d;

% Wheel disable command time
whldi = 39.5;

%% Get parameters

ge0id = tg.getparamid('Motion Algorithm/AOF/Adaptive Error Gain/gain_init','Value');
h3id = tg.getparamid('Motion Algorithm/AOF/Adaptive Error Gain/H3','Value');
rwid = tg.getparamid('Reaction Wheels/Wheel Degradation/Manual Switch','CurrentSetting');
jid = tg.getparamid('Motion Algorithm/Inertia','Value');

%% Set parameters

tg.setparam(ge0id,Ge0);     % Set initial error gain
tg.setparam(h3id,H3);       % Set adaptive parameter

if testcase == 1
    tg.setparam(rwid,1);
    tg.setparam(jid,Jsysid);
    disp('Case 1: System ID Estimate');
else
    tg.setparam(jid,Jest);
    if testcase == 2
        tg.setparam(rwid,1);
        disp('Case 2: Solidworks Estimate');
    else
        tg.setparam(rwid,0);
        disp('Case 3: Degraded Wheel');
    end
end
```

Appendix B

Appendix: Motion Test Data

This appendix section contains relevant data from the PRWP motion test experiments. Each plot set contains four plots: the top-left is the actual wheel speed, the top-right is the command wheel speed, the bottom-left is the IMU body rate, and the bottom-right is the command body torque.

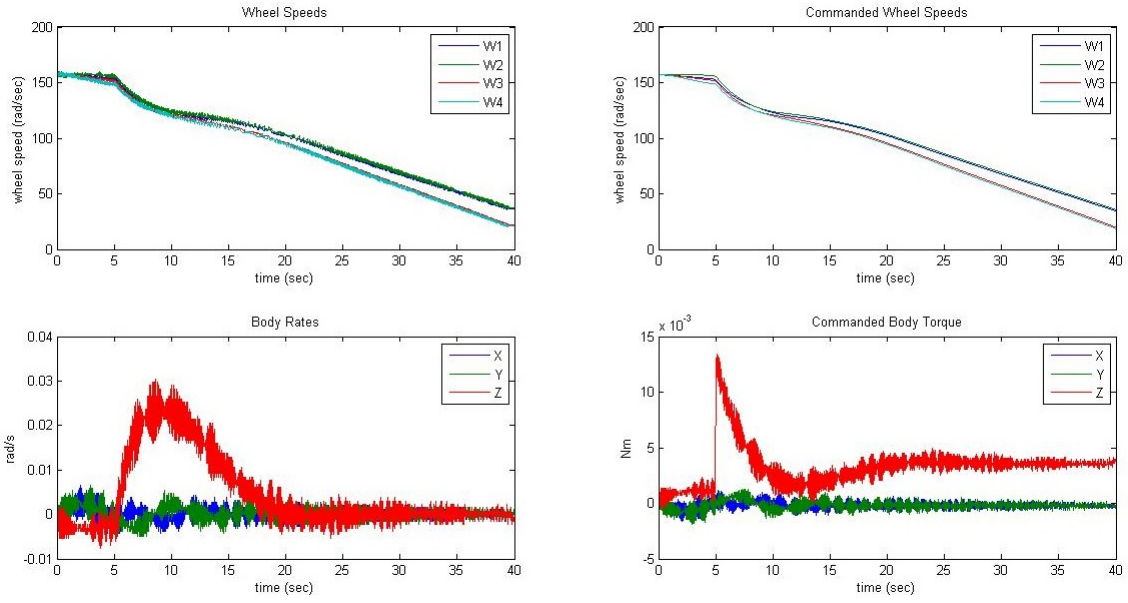

Figure B.1: Motion Test Data, FSFB, System ID Estimate Case.

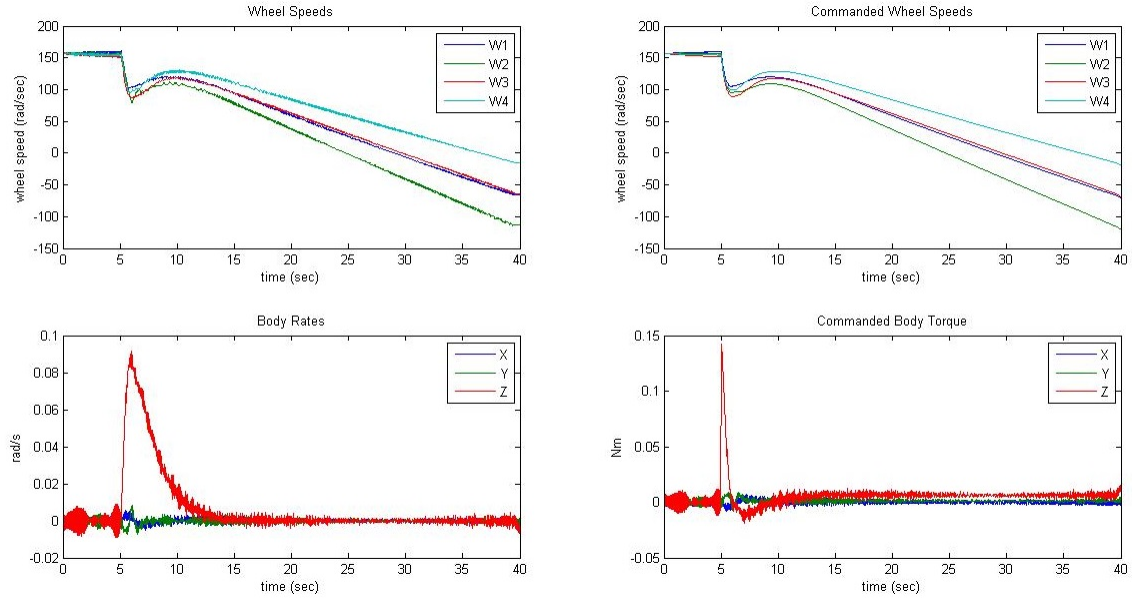

Figure B.2: Motion Test Data, LQRSP, System ID Estimate Case.

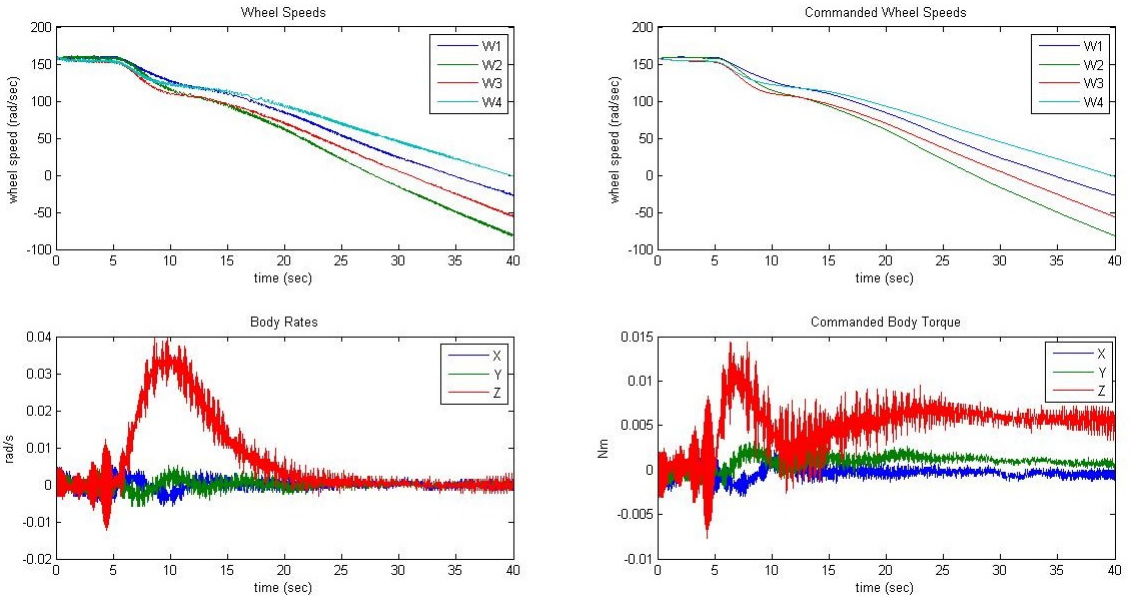

Figure B.3: Motion Test Data, NDMRAC, System ID Estimate Case.

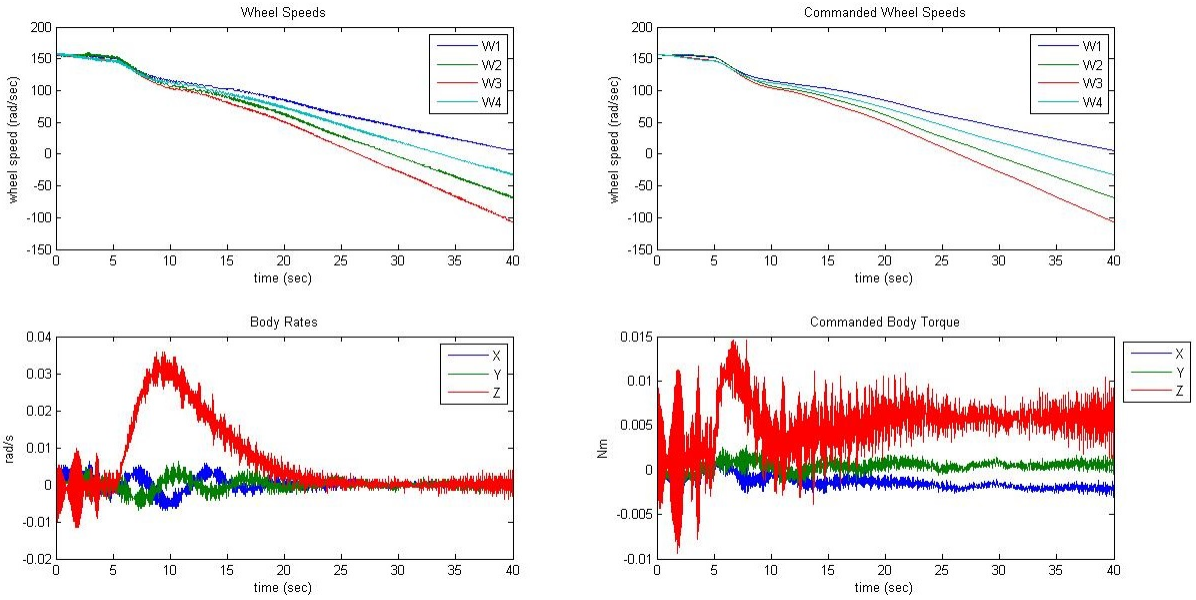

Figure B.4: Motion Test Data, AOF, System ID Estimate Case.

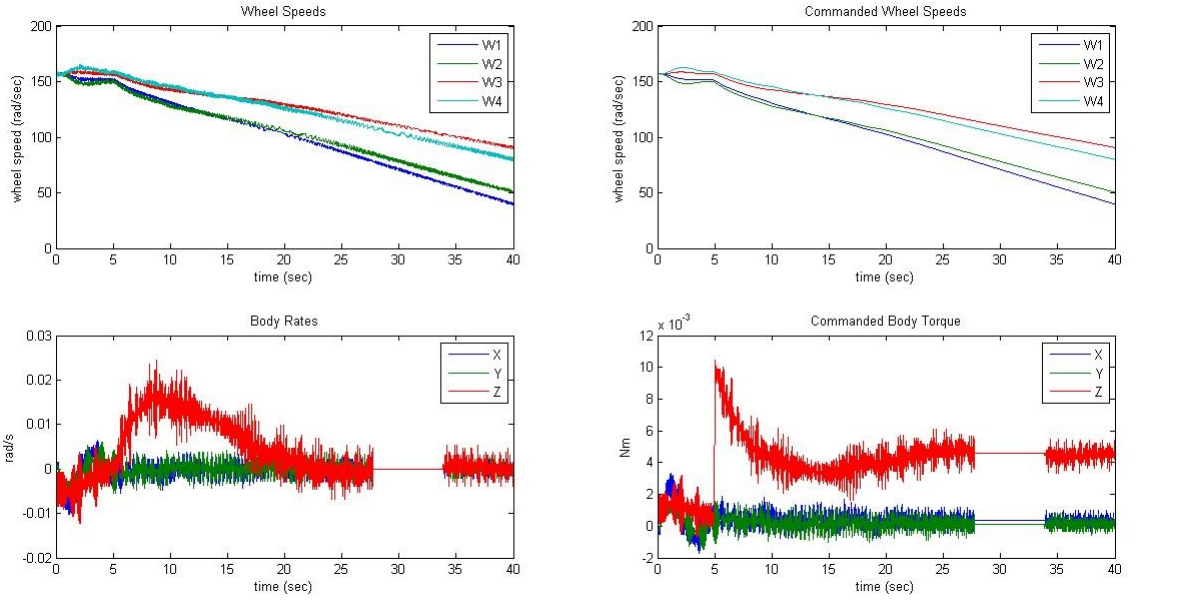

Figure B.5: Motion Test Data, FSFB, Solid-Model Estimate Case.

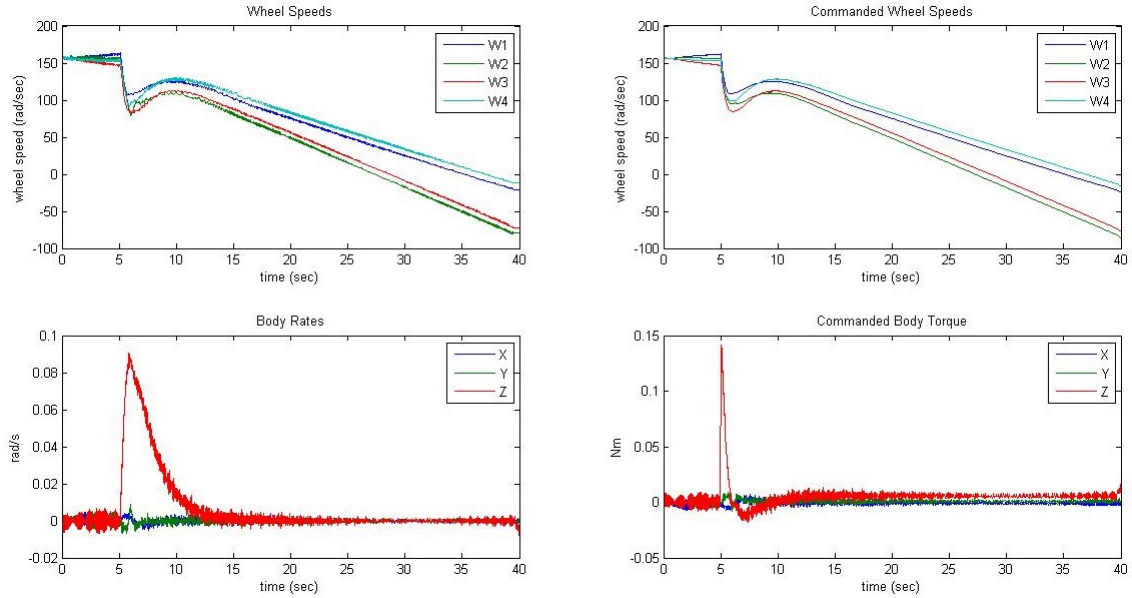

Figure B.6: Motion Test Data, LQRSP, Solid-Model Estimate Case.

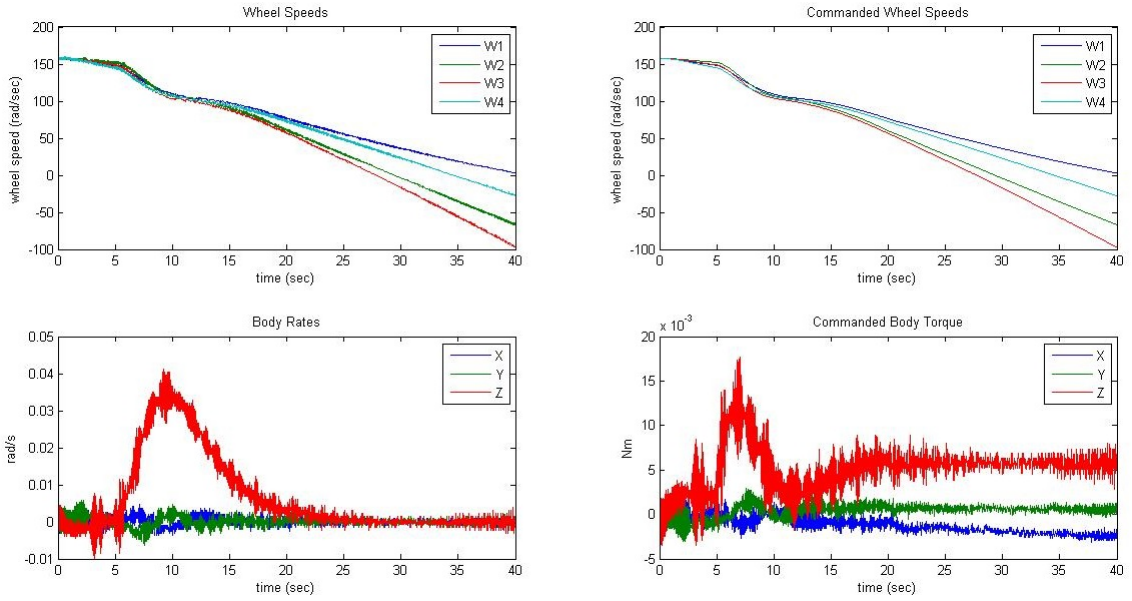

Figure B.7: Motion Test Data, NDMRAC, Solid-Model Estimate Case.

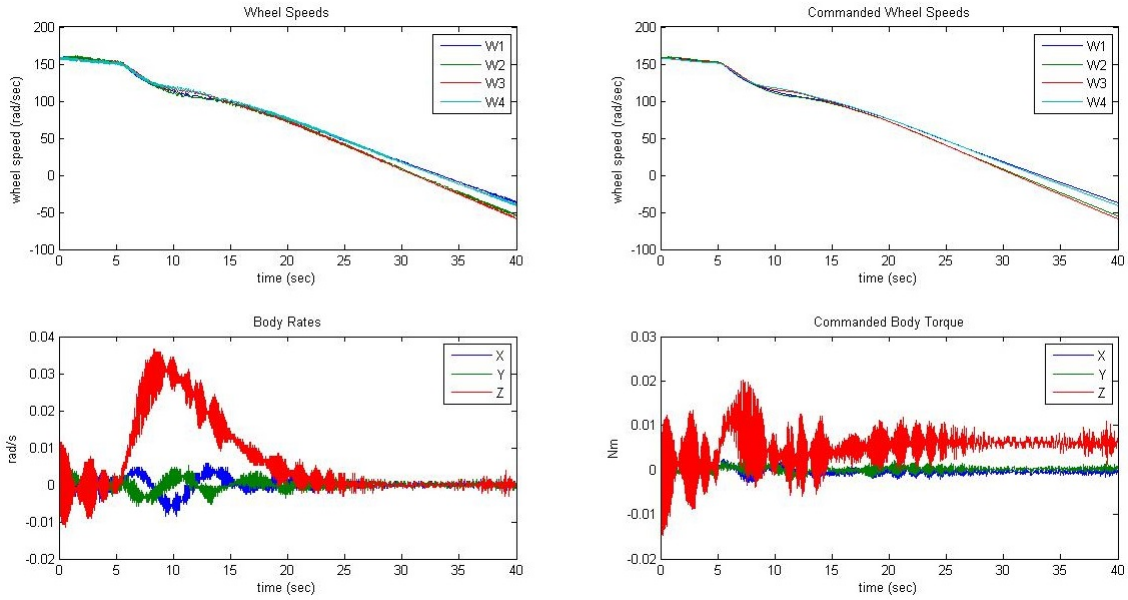

Figure B.8: Motion Test Data, AOF, Solid-Model Estimate Case.

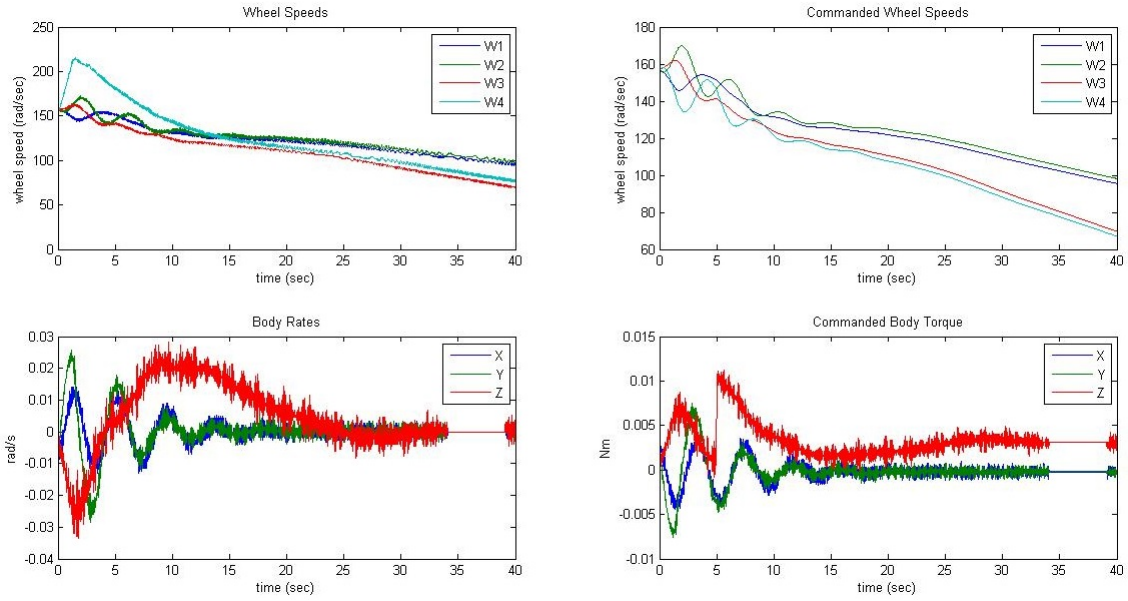

Figure B.9: Motion Test Data, FSFB, Degraded Wheel Case.

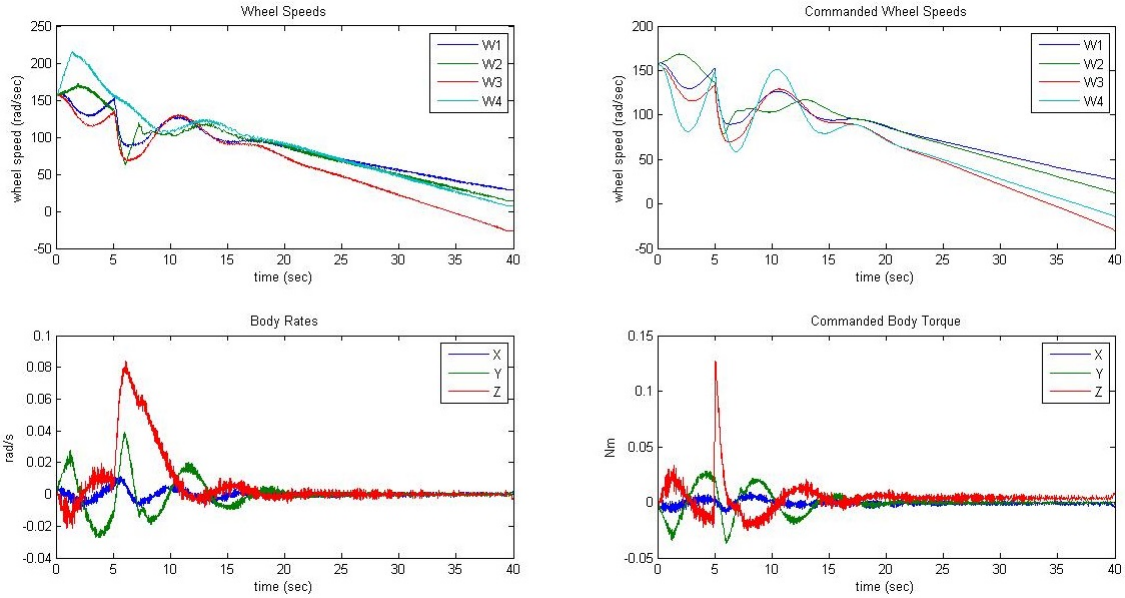

Figure B.10: Motion Test Data, LQRSP, Degraded Wheel Case.

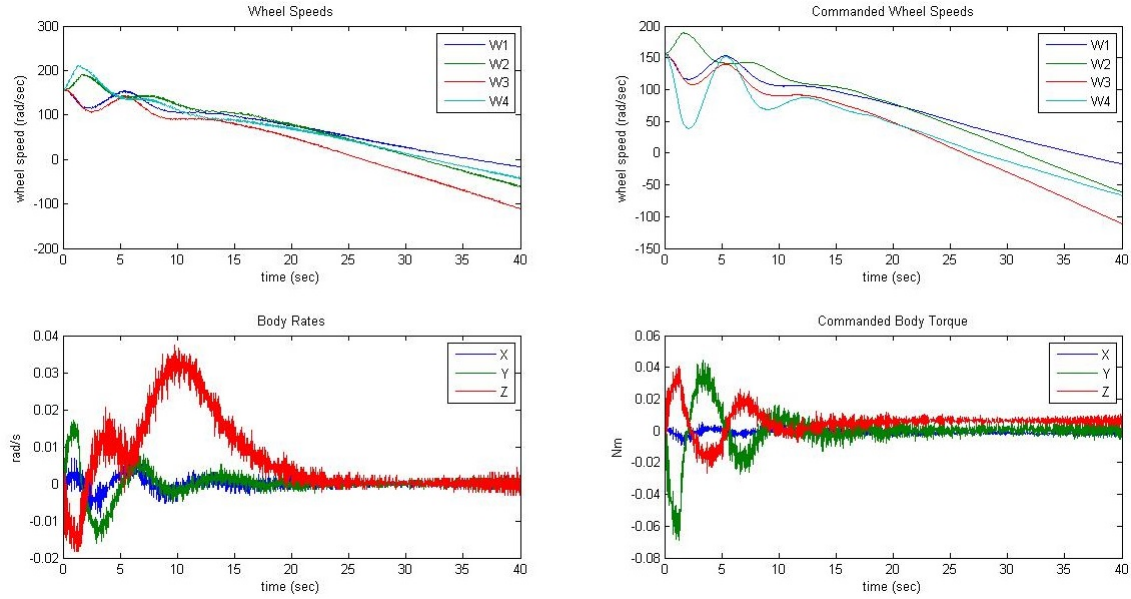

Figure B.11: Motion Test Data, NDMRAC, Degraded Wheel Case.

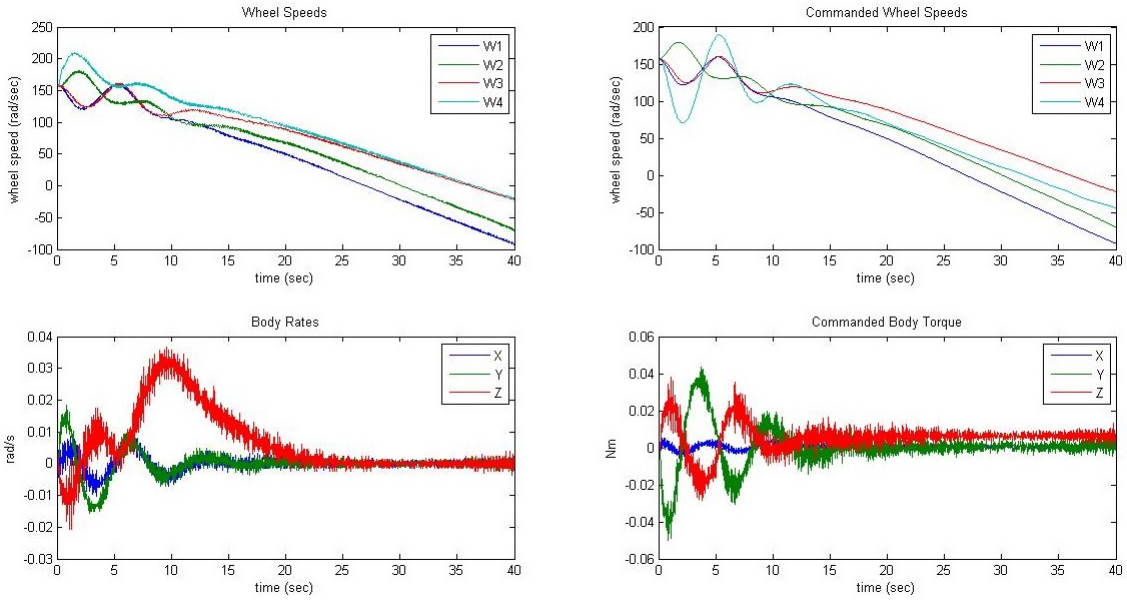

Figure B.12: Motion Test Data, AOF, Degraded Wheel Case.