APPLIED MASS PROPERTIES IDENTIFICATION METHOD TO THE CAL POLY'S SPACECRAFT SIMULATOR

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

Long H. Dam

March 2014

COMMITTEE MEMBERSHIP

Title:	Applied Mass Properties Identification Method to the Cal Poly's Spacecraft Simulator
AUTHOR:	Long H. Dam
DATE SUBMITTED:	March 2014
COMMITTEE CHAIR:	Eric Mehiel, Ph.D. Associate Professor of Aerospace Engineering
COMMITTEE MEMBER:	Kira Abercromby, Ph.D. Assistant Professor of Aerospace Engineering
COMMITTEE MEMBER:	Jordi Puig-Suari, Ph.D. Professor of Aerospace Engineering
COMMITTEE MEMBER:	Helen Yu, Ph.D. Professor of Electrical Engineering

Abstract

Applied Mass Properties Identification Method to the Cal Poly's Spacecraft Simulator

by

Long H. Dam

The Cal Poly Spacecraft Simulator is currently being developed for future testing and verifying theoretical control applications. This paper details the effort to balance the platform and remove undesired external torque from the system using the System Identification technique developed by Patrick Healy. Since the relationship between the input and output of the system is linear, the least square method is proposed to identify the mass properties and location of center of mass of the system. The tests use four sine wave generators that are out of phase with different amplitudes as the inputs to excite various structural modes of the system. The outputs, angular rates of the platform, are measured by the newly implemented LN-200 Inertial Measurement Unit that helps reducing the measurement noise. Two test cases of 90^0 yaw rotations with the identified inertia were performed and validated against the computer simulation model; and the result shows that the test cases trajectories followed closely with the computer simulation model.

Keywords: System Identification, Spacecraft Mass Properties, Spacecraft Simulation and Control.

Acknowledgments

I would like to thank:

My thesis advisor, Dr. Mehiel,

for his support and guidance throughout the project;

and

My mother, and family

for their unwavering support, and patience.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Building and launching a complex space vehicle often require high cost for designing, extensive testing, and verification process due to the spacecraft's operating environment. Since retrieving and servicing spacecraft on-orbit is still not an affordable option for commercial space products, therefore understanding the system, and verifying designs at ground level will assure the mission's success. With the development of system engineering, the space industry aimed to reduce mishaps through testing components and assembly level of spacecraft. The Jet Propulsion Laboratory (JPL) created complex simulators to test their control applications and flight software. In order to test and verify control laws, many academic institutions including Cal Poly use a low cost air bearing simulator to replicate the dynamic of a spacecraft on ground. Many applications of dynamics of spacecraft such as fuel sloshing, structural damping, or momentum exchange devices can be studied and tested before launching.

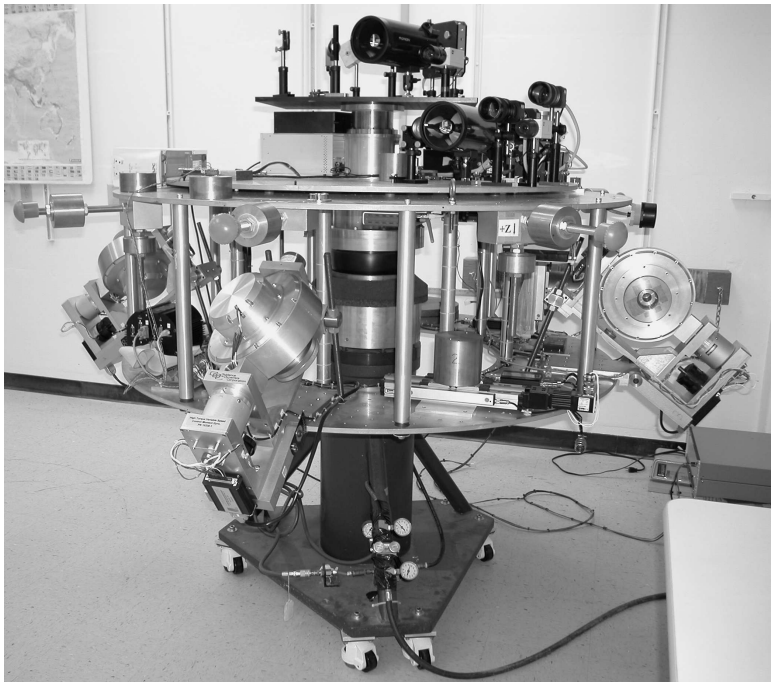


Figure 1.1: Naval Postgraduate School Spacecraft Simulator [1]

1.1 Cal Poly Spacecraft Dynamics Simulator

Cal Poly is one of many universities that utilize an air bearing platform for spacecraft simulation and control. The list of these universities and research institutions includes: NASAs Jet Propulsion Laboratory, Naval Postgraduate School (NPS) in Monterey, Massachusetts Institute of Technology, Virginia Tech University, University of Michigan, Stanford University, and others. The two main types of motion simulators are translational and rotational, and these models can have varied degrees of freedom based on their applications. The Cal Poly Spacecraft Dynamics Simulator is a three-axis spacecraft simulator which was built to demonstrate and verify control laws. Virginia Tech has two different simulators: a typical air bearing support tabletop platform, and a dumbbell platform for separate maneuvers. While the tabletop platform is limited to only $\pm 5^0$ around x and y axes, the dumbbell platform makes full rotation about two

axes possible [2]. The Cal Poly spacecraft simulator presides over a spherical air bearing support that make a full rotation around one axis (normally defined as yaw-axis) possible, with limited motion around the other two axes, $\pm 30^0$ about roll and pitch. An air bearing support uses a spherical bearing with compressed air to create a boundary layer between the male and female sides. This thin air layer eliminates friction and allows the attached platform to rotate freely under the influence of some external torques. Torque can be created by momentum exchange devices, also called actuators, such as reaction wheels (RWs) or control moment gyroscopes (CMGs) to rotate the platform. Naval Postgraduate School's simulator has control moment gyros (CMGs) [1], while Cal Poly's utilizes a four reaction wheels (RWs) system. NPS's simulator requires high torque for its application, so the CMGs would be a suitable fit. Virginia Tech tabletop simulator uses a combination of cold gas thruster and reaction wheels for its tabletop platform. The reason for this is that thrusters can quickly change the attitude of the platform when reaction wheels cannot compensate for the torque required due to the tabletop's mass moment of inertia distribution. The Cal Poly's spacecraft simulator is a smaller platform with two separate layers for equipments, thus using reaction wheels is adequate for simulation. When dealing with spacecraft simulation at ground level, there is also gravity force acting on the platform. A challenge using this type of ground equipment is the uncertainty of external torque due to gravity acting on the simulator as comparing to negligible gravitational force in space. When the center of mass is not right on top of the center of rotation, this offset causes undesirable effects when simulating control laws, therefore the mass properties of the simulator should be well understood in order to remove the additional torque due to gravity offset. The main navigation sensors for these simulators include accelerometers, micro-electromechanical

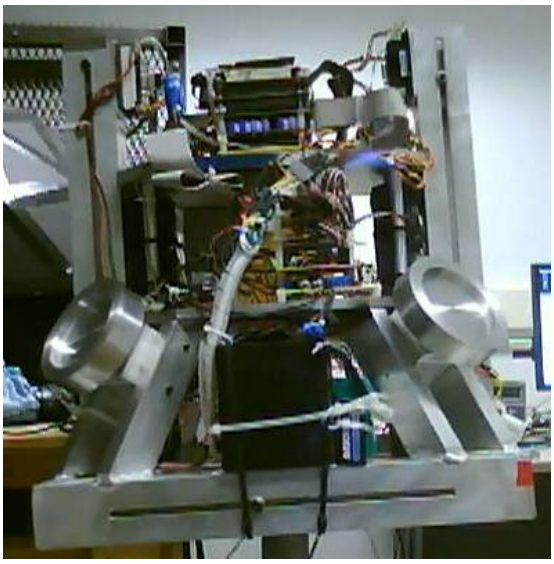


Figure 1.2: Previous Configuration of Cal Poly Spacecraft Simulator

gyroscopes (MEMS Gyros), laser-ring gyroscopes, optical sensors such as star tracker, and/or GPS. The accuracy of these sensors depends on their application and can vary from 0.01^0 to 1^0. A need for an inertial measurement system which will provide the absolute attitude for the platform is crucial to complement the relative measurements. The absolute inertial system will not be demonstrated in this thesis for another graduate student is developing a process with absolute inertial measurement.

1.2 Spacecraft Dynamics and Control

The first step of building spacecraft is understanding its behavior in the operating environment, and then modeling appropriate system dynamics to achieve certain requirement. For controlling spacecraft, instrument's pointing accuracy is often the main requirement driven by design and customer's need. The growing of live television programs, commercially, means satellite operators demand

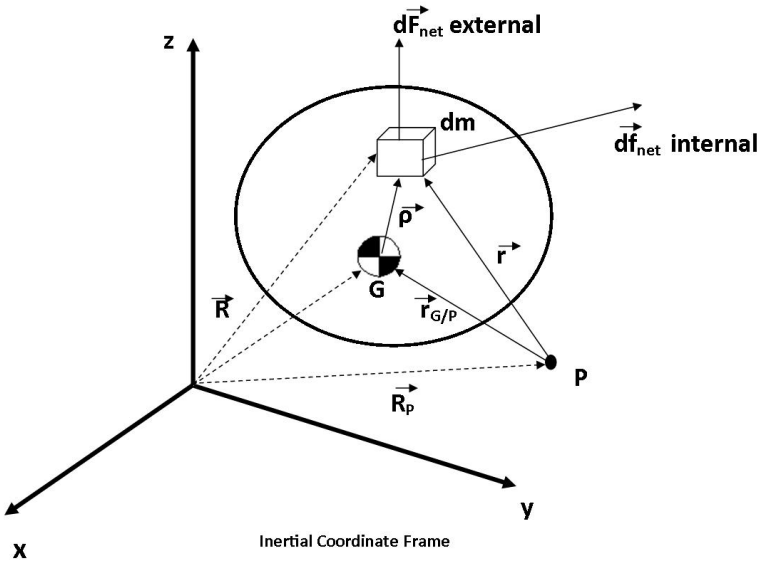


Figure 1.3: Rigid body dynamics deals with translation and rotational motion

better coverage and accuracy. In addition, many scientific and space exploration missions also have strict pointing requirements for precise observation of stars and galaxies. In general, modeling spacecraft often scales down to the simple form of rigid body. Rigid body dynamics concern two types of motions: the translational forces that deal with motion of center of mass, and rotational or angular motion of the rigid body about the center of mass. Translational motion of spacecraft is often studied under orbital mechanics while rotational motion is one of the main interest of control subjects.

Controlling spacecraft is not limited to rotate the vehicle around its center of mass but also including structure identification and sensor noise rejection or filters. Different spacecrafts and operating environments use variety of methods for attitude control. Large spacecraft such as the International Space Station (ISS) controls its attitude using CMGs as actuators while smaller class of satellite might use reaction wheels or thrusters. Actuators such as CMGs and RWs

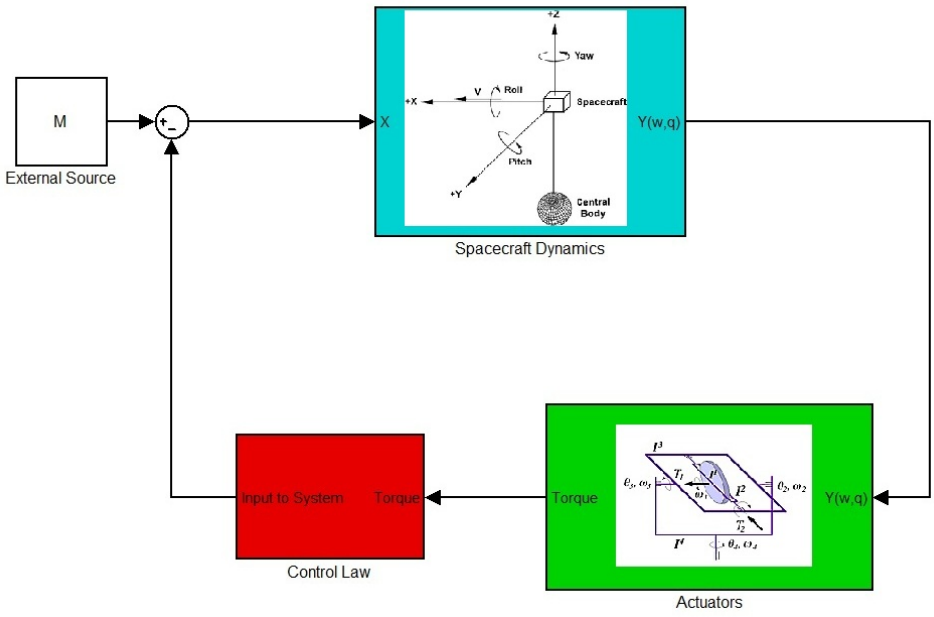


Figure 1.4: A basic control loop for spacecraft in block form

are used to store momentum, hence the wheels need desaturation from external sources. These sources could come from operating environment such as using magnetic torque or gravity-gradient torque in low-Earth orbit, and thrusters in other environment. In the simple form of rigid body, the equations will produce a relationship between torque and momentum with product of angular rates and inertia. The equations of motion will be further derived in chapter 3.

For any shape of rigid body, there exists a set of axes, principal axes, for which the products of inertia are all zeo. The corresponding moment of inertia is called principal moment of inertia. The moment of inertia, often denoted as J, should be well understood to control spacecraft. Building platform and testing moment of inertia generally requires a spin or shake table in industry. This method requires component to be designed and secured on the spacecraft for various sweep signals. Even when the moment of inertia can be identified with

ground tests, spaceflight operation must sometimes take into account of inertia variation. The space shuttle and other commercial resupply ships, for example, dock with the ISS on regular basis. This process often changes the mass properties of the orbiting outpost. For this reason, in-flight identification were developed and tested to ensure smooth operation and control. A demonstrated example using the NASA model of the ISS was presented by Bergmann and Dzielski [4]. The fundamental approach to identify unknown parameters is based on Newton-Euler equations of motion; one can convert the equations into a regression form and then apply the least-square or other filter techniques to solve for the unknown mass, position vector of the mass center, and inertia tensor of the spacecraft in-flight.

Since the Cal Poly spacecraft simulator already has a set of four reaction wheels mounted on the platform, an 'in-flight' scenario mass properties identification is accessible. Combining with the output from the inertial measurement unit LN-200, the wheels input can also be recorded from the motor controller. With the input and output known, a least-square method for mass properties identification can be derived from the equations of motion. The derivation of this method will be discussed in section 3.4. Another advantage to implement this system identification method with the reaction wheels is this method eliminates the need for a shake/spin table, and the whole platform will not be moved or reseated on the air-bearing support. This method allows the operator to quickly identify the mass properties if new components were implemented on the platforms.

1.3 Mass Properties/System Identification

In aerospace engineering applications, the initial structure designs often start with 3D Computer Aided Design (CAD). There are many CAD programs that offer the ability to input the components' materials and calculate the mass properties for those structures. Since all components, especially small parts, and wire hardness cannot be included in CAD model, it is necessary to seek the best estimated values through a system identification method. System Identification is a process to estimate a set of unknown parameters based on the relationship between the input signals and observable outputs of the system. In other context, system identification can be used to obtain a plant's transfer function of a control problem. The later process involving fitting a regression model for the transfer function under some assumption such as linear time-invariant. The difference between these two methods is the former assumes or understands the dynamics of the plant quite well, while the later only knows the inputs and certain observable outputs. For an air bearing platform such Cal Poly spacecraft simulator, the identification process is an algorithm to estimate the platform's mass moment of inertia and the center of mass location based on the kinematics and dynamics of a rotating body under an applied torque. This process involves intentionally applying external torque with the reaction wheels to disturb the platform and measures the corresponding angular rates using an inertial measurement unit, LN-200. Then using the relationship of rotational body and inertia, the mass properties of the platform can be uncoupled from the torque products. There exists many methods for spacecraft mass properties identification in literature. Ma, Dang, and Pham [5] applied momentum technique to compute spacecraft's inertia using robotic arm to change the spacecraft's velocity. Other papers proposed

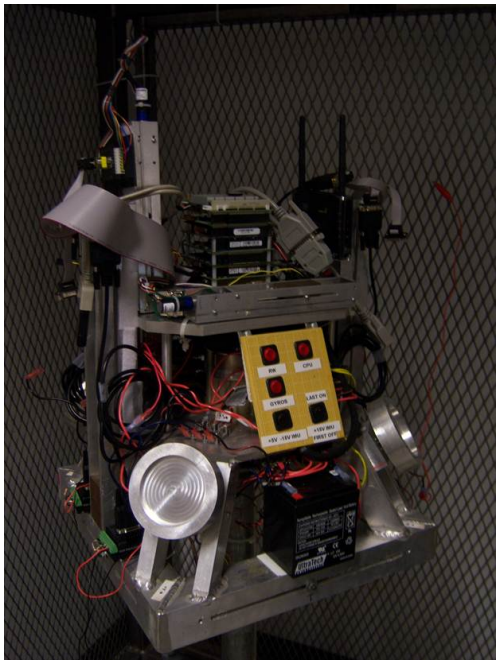


Figure 1.5: Current Configuration of Cal Poly Spacecraft Simulator

and applied an approach using Newton-Euler equations of motion to estimate the unknown inertia [4]-[7]. The equations of motion are converted into a regression form and applied the least-square techniques or other filtering methods. To deal with multi-input/multi-output (MIMO) system, Cooper and Write employed the eigensystem realization for in-orbit inertia identification of spacecraft [8]. Other technique includes Observer/Kalman filter identification algorithm which formulate entirely in time domain [9]. This process identifies the unknown open-loop plant, controller gain matrices, and observer gain using Markov parameters (impulse response functions) [10]. More Markov parameters discussion and formulation can be found in [12]-[13].

Understanding the mass properties and center of mass location is important to spacecraft simulation. Tom Kato, a graduate student at Cal Poly, is currently comparing and validating theoretical results for three different controllers on the spacecraft simulator. One simulation case requires a near perfect inertia which

Summary of modal parameter estimation algorithms

	Domain		Matrix polynomial order			Coefficients	
Algorithm	Time	Freq	Zero	Low	High	Scalar	Matrix
CEA	●				●	●	
LSCE	●				●	●	
PTD	●				●		$N_i \times N_i$
ITD	●			●			$N_o \times N_o$
MRITD	●			●			$N_o \times N_o$
ERA	●			●			$N_o \times N_o$
PFD		●		●			$N_o \times N_o$
SFD		●		●			$N_o \times N_o$
MRFD		●		●			$N_o \times N_o$
RFP		●			●	●	Both
OP		●			●	●	Both
CMIF		●	●				$N_o \times N_i$

CEA: Complex exponential

LSCE: Least squares complex exponential

PTD: Polyreference time domain

ITD: Ibrahim time domain

MRITD: Multi-reference time domain

ERA: Eigensystem realization algorithm

PDF: Poly-reference frequency domain

SFD: Simultaneous frequency domain

MRFD: Multi-reference frequency domain

RFP: Rational fraction polynomial

OP: Orthogonal polynomial

CMIF: Complex mode indication function

Figure 1.6: Summary of Identification Methods for Modal Parameters [14]

can be achieved using the system identification to complement estimated values from CAD modeling such as SolidWorks. It is also important to position the center of mass as close as possible to the center of rotation as the external torque from gravity will cause wheel's speed saturation and prohibit some maneuvers on the platform. This master thesis applied the system identification method derived by Cal Poly graduate student Patrick Healy [15] to the spacecraft simulator to obtain the mass properties of the platform. Previously, Seth Silva [16] attempted to demonstrate the system identification on the platform. However, his test results were inconclusive due to low data sampling rate on bluetooth server, high noise level in platform's angular rates measurements, and inherent error from taking discrete derivative of noisy angular rates.

1.4 Thesis Objectives

There are two primary objectives in this theis. The first objective is to integrate new hardware, Litton LN-200 Inertial Measurement Unit, on the platform. The scope of this objective is to improve the noise measurement from the previous set of navigation sensors (MEMS Gyros), and apply Hardware-in-the-Loop testing for real time simulation with new software Mathworks' xPC Target.

The second objective of this thesis to apply the mass properties identification method derived by Patrick Healy [15]. Generally, many of the aforementioned techniques, [4]-[7], involve least-square regression method for system identification. In his thesis, Healy [15] also used the regression model and least square estimation for the system identification. Healy [15] shows perfect mass properties can be obtained without measurement noise and approximately 5.5 % error when a noise level similar to the LN-200 is present in numerical simulation. The

new test applies a set of four sine waves with different amplitudes and out of phase to excite various structural modes on the platform. This test allows a large of amount of torque to be applied to the platform in a short amount of time, in other word a high momentum input to overcome the initial gravity acting on the platform for pitch and roll axes. The outputs from this test will be measured by the LN-200 with reduced noise, and thus should improve the result of the estimated inertia. Finally, a validation case with the Full-State Feedback controller will be discussed for the identified inertia.

Chapter 2

Hardware and Software

2.1 Real Time xPC Target

In simulation and testing, the data rate is important to analysis, particularly in the experiment with sine wave generators. This experiment excites different structural modes of the platform around the body axes. Higher data rate will enable majority, if not all, of the modes to be sampled, and thus produce better result for system identification. Previous configuration with Gumtix/Robotix

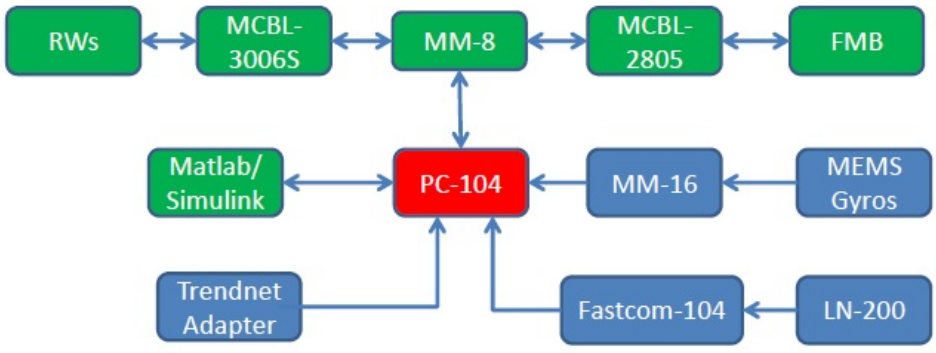


Figure 2.1: Simulator's Command and Data Handling

through a Bluetooth server was only able to sample at 1 Hz, so this limitation was one of the main contributors to the errors for system identification performed by Silva. Since then Matt Down and Ryan Kinnet improved the data rate by implementing a remote desktop session on a PC/104 form factor computer on the platform. Although Matlab/Simulink is a great application for object oriented programming and visual modeling, running it on a PC/104 computer with limited processing and RAM is not desirable due the extensive GUI of Simulink. In addition, Simulink inherently is not designed to handle multiple serial connections with Window API. Therefore, a new method to operate the platform was reexamined.

Mathworks offer a complete real-time hardware in-the-loop simulation environment that would be suitable for the spacecraft simulator. xPC is a kernel that interface directly with the PC/104's interrupt channels and I/O resources, and this option eliminates the need to: install Simulink software on target computer, modify existing software configuration, or access the hard disk on the target computer via a bluetooth server. The xPC kernel can be booted using a flash drive with the boot files. This will conveniently remove the need for a portable SATA hardrive which was required for Window XP since all the data from xPC can be saved on the NAND flash disk on the PC/104 target computer. This new software will also allow the reaction wheels and LN-200 to execute real-time tasks simultaneously. Switching to xPC, however, will require appropriate data acquisition boards and develop new hardware drivers for those boards which will be discussed in the next section.

When the target computer is first booted, the BIOS searches for an executable image from the USB drive. It is important to disable any power saving features in the BIOS as specified in the xPC target instruction manual since it could po-

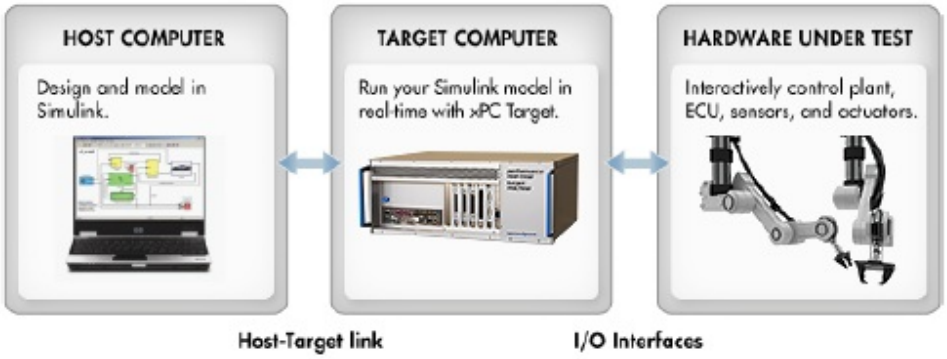


Figure 2.2: Hardware and software flow of xPC target [17]

tentially degrade the real-time performance. The Plug-and-Play (PnP) must also disable so the BIOS can set up the plugged-in PCI card. Additional features that must be disabled include: PCI board with class code 0xff, and hyper-threading. Once the kernel is loaded, the resources on the CPU motherboard are addressed entirely through I/O addresses. For example, the interrupt channel of the Fastcom/104 board for the IMU data acquisition is set at 10 with the base address of 0x340. Any register addresses specified in the IMU's instruction manual will be offset from the base address. For debugging purposes, user can connect a display screen to the PC/104 target computer to confirm the host-target connection after the kernel starts running. Another way to check without a display screen is to use the scope block in Matlab GUI on the host computer or the command 'xpctargetspy' in the command window. In general, a user can customize any xPC supported hardware by writing their own source code and compile it in Matlab. Then the interface block between Simulink (on host computer) and the target computer's resources can be created for each compiled code by calling it under the Simulink's mask with appropriate masked variables.

Figure 2.3 shows a typical connection between the host and target computer. The host computer can be any computer that operates Matlab/Simulink envi-

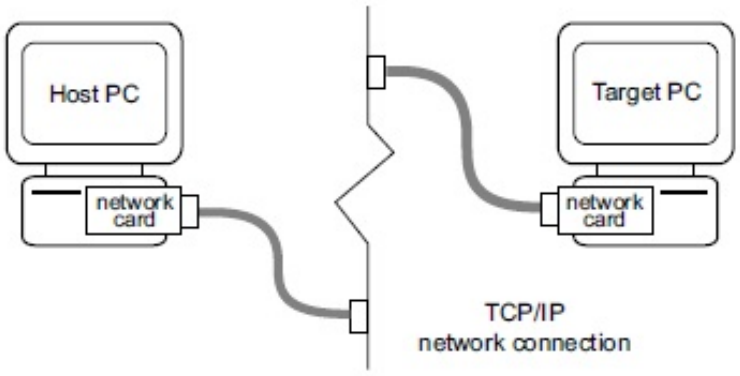


Figure 2.3: A typical configuration of xPC target for the target and host computer [17]

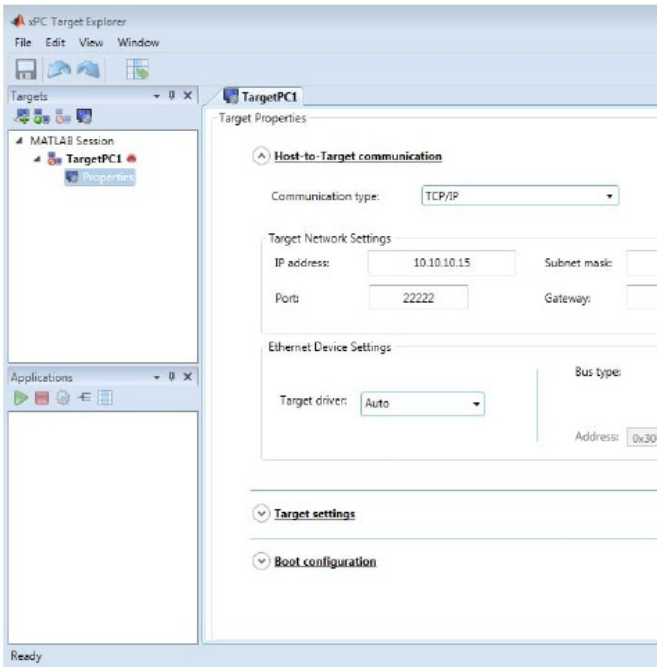


Figure 2.4: xPC GUI setup for target computer [17]

ronment while the target computer can be a small compact form like PC/104 with DOS available. There are two type of connections between the host and target in xPC: serial and network. For serial connection, both computers will be connected through a null modem link cable RS-232 upt to 5 meters long with a transfer rate between 1200 and 115200 baud. Network connection can be setup through LAN, direct connection through a crossover Ethernet cable or wireless server. Typically, the network communication often has advantages over serial communication due to: longer distances between host and computer, less wire/-cables for multiple target computers, and higher data throughput since Ethernet can transfer data up to 100 Mbit/second compared to 115200 baud in RS-232 connection.

The setup example for a host-target through the network is shown in figure 2.4. The port is always set to 22222 while the IP address is depending on the server address allocation. The boot configuration is set to USB-drive. Once the target is connected, user can set up Simulink model and perform real-time hardware simulation with a host computer.

2.2 Litton LN-200 Inertial Measurement Unit

Measurement noise is an important factor in experimental data analysis. Due to nature of electrical signals, random white noise in voltage measurement cannot be avoided. In the previous configuration, three micro-electromechanical gyroscopes (MEMS gyros), shown in figure 2.5, were used to measure the angular rates of the platform. The MEMS gyros' data often have high noises and drift rate at compared to the Northrop Grumman LN-200 inertial measurement unit, IMU. Figures 2.6 and 2.7 details the difference in measurement noises and drift

Figure 2.5: MEMS Gyroscope CRS03

rates between the two navigation measurement devices. Figure 2.6 shows the drift for one MEMs gyroscope. Assuming the three MEMs gyroscopes are similar, the drift rate shown could be considered similar. For Euler angles (roll, pitch, yaw) measurement over 60 seconds at static position, the drift rates for MEMs gyroscopes exceed 5^0 while the IMU's gyroscope only drifts approximately 0.12^0 over the same period. This shows that the IMU is desirable for long period simulation especially when the both angular rates and acceleration are required for the system identification method. Therefore, it is important to implement this device onto the platform to improve the data's fidelity.

Another important factor when receiving the data is the rate. Higher data rate also increase the fidelity of the data receive. This gives the user options to down-sample to appropriate multiple integer for simulation purpose. LN-200's data rate by manufacturer's default setting is 400 Hz, with 26 Bytes per package of data. However, Simulink's simulation rate is set to 50 Hz, so the sampled data by xPC target will be down-sample. Down-sampling will not, in this case, affect the data since the down-sample is a multiple integer of the IMU's sampling rate

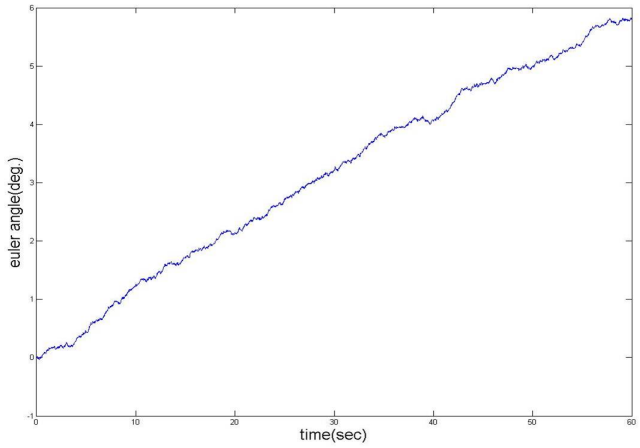


Figure 2.6: Single MEMs Gyro Drift

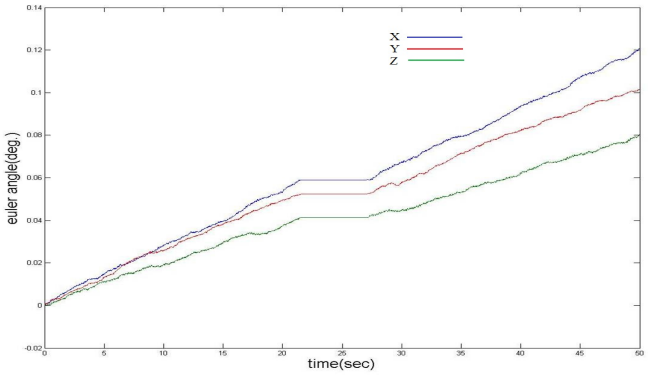


Figure 2.7: IMU Drift

which means this case does not require interpolation. LN-200 also uses IBM's High-Level Data Link Control (HDLC) for data formatting and non-return-to-zero (NRZ) encoding.

During the IMU's implementing process, two drivers were written for xPC target with two data acquisition (DAQ) cards: Fastcom ESCC-PCI and Fastcom ESCC-104 (ESCC stands for Enhanced Serial Communication Controller). The difference between those cards is the pin connection. Fastcom-PCI is designed for PCI card, Fastcom-104 is for PC/104 form factor. Both cards utilizes the Siemens 82532 (SAB 82532) ESCC with both RS-422/RS-485 communication protocols. The driver for Fastcom-PCI was written first to demonstrate the capability of xPC target since it was already used in the lab's computer from previous IMU's demonstration by Phil Iversen. With this card, the user could debug the error (either coding or power issue) of the IMU since Fastcom-PCI can be easily plugged back into the lab computer. The Fastcom-104 was purchased later to reduce the stress on the PC/104 stack since the PCI version requires an addition PCI to 104 adapter,which made the Fastcom-PCI suspended off the side of the PC/104 stack.

The driver for the Fastcom-104 contains four codes: the header file, a setup file, the read file, and a hook function file. The header file 'fastcom104.h' contains the register offset addresses, initializing values, and command messages to the registers. The most important value is the 'receive message complete, CMDR-RMC 0x80' as this value will let the controller reset the interrupt status until the next available and valid IMU's data frame is received. The 'fastcom104setup.c' file initializes all SAB 82532's registers according to HDLC setting. It also powers up the Fastcom-104 card after the initialized stage. An important step after this sequence is to flush the previous data out of the transmit First In First Out

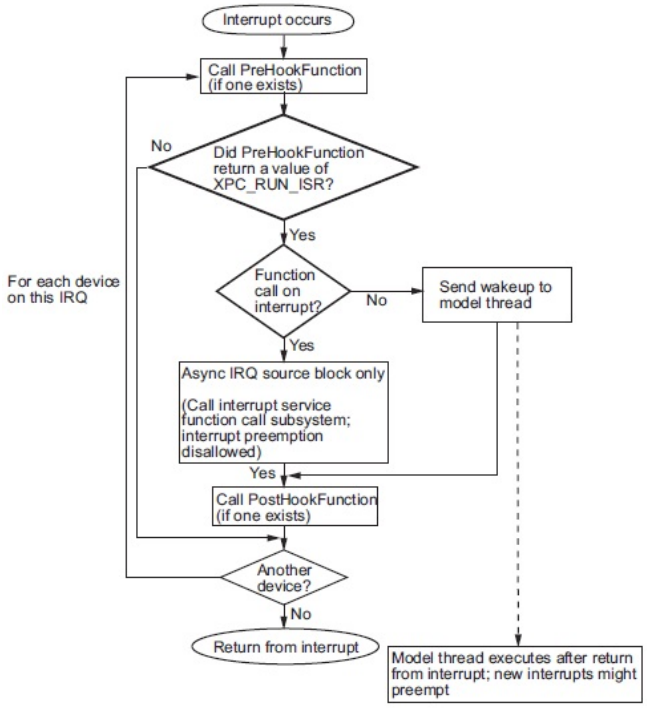


Figure 2.8: IMU driver flow chart for xPC target [17]

(TFIFO) and receive FIFO (RFIFO) registers. The DAQ card is now ready to receive the data from IMU at the first interrupt. The 'fastcom104read.c' file performs FIFO read from the SAB 82532 register when a RFIFO interrupt is available. User should note that data from xPC target is given in binary values; therefore, negative number should be converted using the 2-complement method. For this code's first version, the raw data will be converted to double per Matlab/Simulink standard, while Simulink is running, with customized conversion block. Figure 2.8 shows the interrupt handling process when writing custom driver for xPC target. The 'xpcfastcom104hook.c' file is designed to service the interrupt of the Fastcom-104 as specified in the xPC target's instruction manual and figure 2.8.

2.3 Reaction Wheel Motors and Controllers

In previous configuration, a set of four brushed motors were used to impart torque on the platform. These motors required a Pulse Width Modulation (PWM) circuit, a Voltage to Analog data acquisition board, Polulu motor driver, and a direction sensing circuit to produce the desired speeds. This setup complicated wires management since the communication between the wheels and PC/104 must go through all the aforementioned boards.

The new configuration calls for a more robust system that can be easily programmed and removed, a plug and play scenario, if there was a need to simulate a test with just three reaction wheels. This new configuration implemented a set brushless DC motors from Faulhaber and four MCBL3006S motor controllers. These motors carry Hall sensors that can accurately detect the motors' speeds. In addition, the motor's settling time and acceleration can be programmed with the built-in Proportional Integral (PI) controller, and a serial RS-232 cable provides the communication between the PC/104 and motor controller which could be easily removed. The default manufacturer's setting for the proportional and integral terms are 7 and 40 while the current setting for this simulator's motors are 90 and 3, respectively. For many motors, acceleration or deceleration limits plays a role in real time simulation as opposed to instantaneous response in theoretical simulation. A large acceleration in short period of time causes phase lag in the actual response of the motor seen in figure 2.9. The appropriate acceleration and deceleration limits for the motors were set to 50 rad/sec^2. Figure 2.10 shows the tracking of the commanded and actual wheels' speed. The actual responses follow closely, if not right on top, of the commanded values.

Due to limited serial ports on the PC/104 board, a Diamond MM-8 board

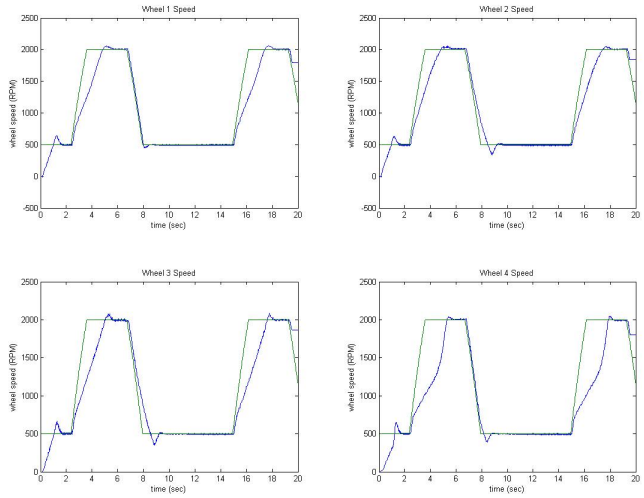


Figure 2.9: Wheels' speed profile with high acceleration

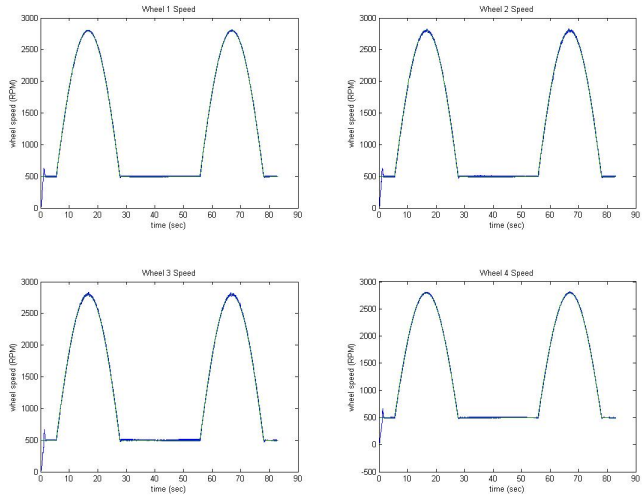


Figure 2.10: Wheels' speed profile with acceleration limit of 50 rad/sec^2

is used to accommodate all 4 reaction wheel motors and three fine mass balance motors. Much of the driver were rewritten to suit xPC target software. Similar to the IMU's driver, this driver services the interrupt upon available data from read or write FIFO, and flush the hardware's data at start-up. Additional information will be explained in the laboratory's manual.

2.4 Data Acquisition Boards

With the newly chosen operating environment in xPC target, appropriate hardware are required to be compatible with this software. LN-200 requires high data acquisition with RS-485 protocol. Due to the LN-200 communication protocol, data formatting, and encoding, special board is needed to achieve data transfer. Phil Iversen demonstrated the LN-200 data acquisition with the Fastcom ESCC-PCI synchronous communication adapters in the lab computers using customized driver written in C language for Simulink. For xPC target kernel, the project started with the same Fastcom ESCC-PCI card using a PC/104 to PCI adapter to demonstrate and verify data acquisition from LN-200. Since the driver works for xPC with the PCI card, the team then purchased and rewrote the driver for the Fastcom ESCC-104 to stack on the PC/104 module in order to minimize spaces taken by the Fastcom ESCC-PCI. Figure 2.11 shows the fully assembled PC/104 stack while figure 2.12 shows the Fastcom ESCC-PCI and the Fastcom ESCC-104.

The PC/104 stack also contains two data acquisition boards for the motors and MEMs gyroscopes. The MEMs gyroscopes are used as the backup navigation sensor for the LN-200 IMU. Diamond MM-16 is a 16-bit analog I/O module that provides 16 single-ended or 8 differential inputs to the embedded computer. The

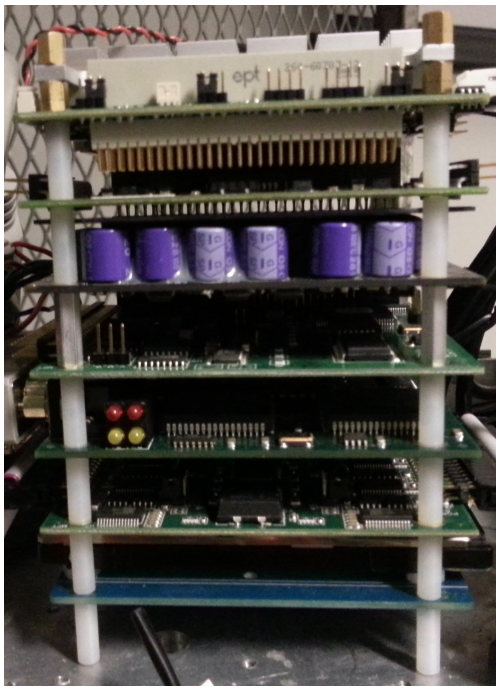


Figure 2.11: Simulator's PC/104 Stack

Figure 2.12: Fastcom ESCC-PCI and Fastcom ESCC-104 D.A.Q cards

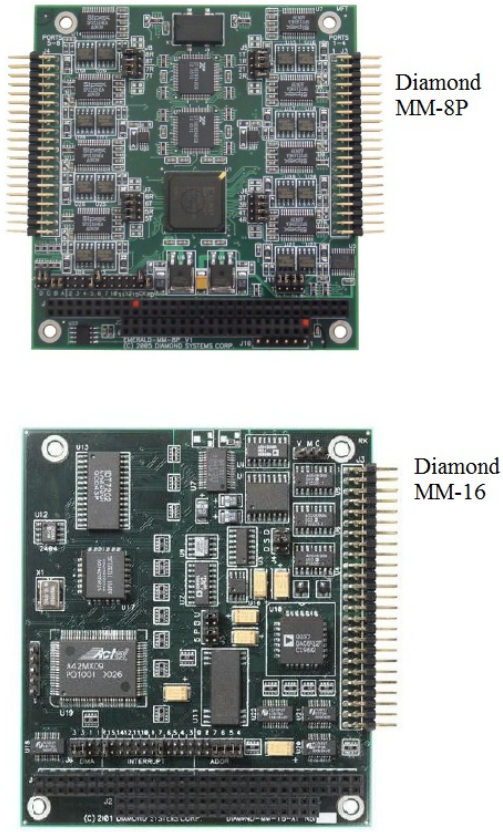


Figure 2.13: Diamond MM-8P and MM-16 D.A.Q cards

board also includes 8 additional digital inputs and outputs port that are rarely used for the simulator. With the embedded PC/104 computer carrying only 2 dedicated RS-232 ports, the team purchased the Diamond MM-8P to accommodate the ports requirements: 4 ports for the reaction wheels and 3 for the fine mass balance. Since the Diamond MM-8P has 8 ports, the remaining port and the PC/104 computer's ports will serve as backup to the current configuration or expansion options for future equipments. Diamond MM-8P is an 8-channel software configurable protocol serial port PC/104 module. It is software configurable for RS-232, RS-422, or RS-485 with 64-byte FIFO and up to 115200 baud in standard configuration. Figure 2.13 shows the Diamond MM-8P and MM-16 cards used on the platform.

2.5 Mass Balance System

As additional components integrated onto the spacecraft simulator, it is crucial to adjust the center of mass location and balance the platform. Therefore, a mass balance system is needed for the platform. This system consists of two types: coarse mass balance and fine tuning system (FTS).

Coarse mass balance consists of six identical rectangular steel blocks two on the vertical channels, and four along the base of the platforms. Each coarse mass balance is approximately .77kg and made of 1022 steel with a pair of bolts connected to the sides. The movement for these masses can be done by sliding along the tracks on the vertical and base channels and tightened by a hex nut. These coarse masses are manually tuned to balance the platform reasonably, since it is impossible to correctly adjust the center of mass by hand. Further tuning is accomplished with the fine mass balance system.

The fine tuning system consists of three steel blocks that are controlled by high precision motors. Each of these blocks is 0.195 kg and they are connected to the motor by a lead screw. The lead screw has a total length of 18.5 cm with a threaded length of 16 cm. The lead screw has a 1 mm lead which mean one revolution of the lead screw will result in a 1 mm linear movement for each steel block. The fine mass balance set utilizes the Faulhaber 1628T brushless DC servomotor with torque of 2.5 mNm and the Faulhabers MCBL 2805S motor controller. In addition, the motors also carry a gearhead with 246:1 reduction ratio. To get one revolution on the motor shaft, one can apply the position control mode on the MCBL 2805s with a command of 1000. Combining the position control mode with the gear head output, a very precise position control for the mass can be obtained: 246000 position control for 1 mm on the lead

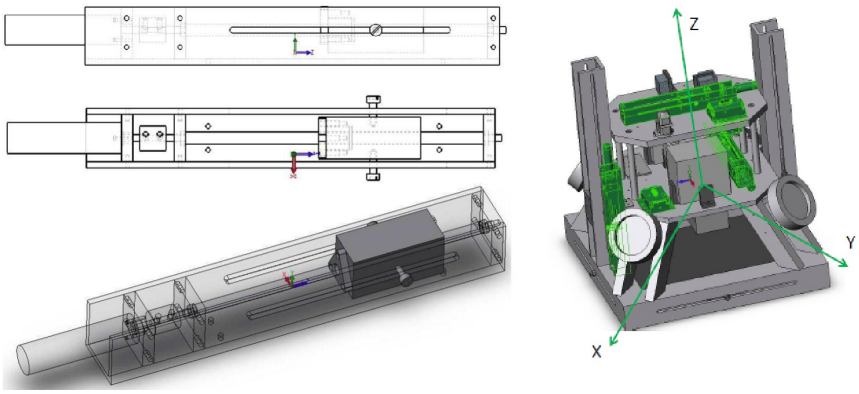


Figure 2.14: Fine Mass Balance Model from Solidwork

screw. In other words, the center of mass of the platform can be precisely moved in micrometer range.

Each steel block is held on an aluminum U-channel by a set of screw nut on each side, and the length of the track that this mass can moved on in the U-channel is approximately 10 cm. If the FTS are placed parallel to the body axes, then the maximum direct center of mass compensation on each axis can be easily calculated using the ratio of the fine balance mass to the platforms total mass which will be described on the next chapter.

Chapter 3

Rigid Body Dynamics and Quaternion Kinematics

In numerical simulation for SDS platform, it is important to understand and model the dynamics of a rotating body, the torque from reaction wheels, and any additional torque acting on the platform. Figure 3.1 shows the schematic of the platform with X, Y, and Z represents the inertial coordinate reference frame with unit vectors $\vec{I}, \vec{J}, \vec{K}$, and x, y, z are the body axes with the origin at the center of rotation O. The vector $\vec{r}$ represents the distance from the center of rotation (CR) to center of mass (CM) of the platform.

Figure B.2 in Appendix A shows the alignment of the wheel along the body x and y axes. Each wheel is inclined by an angle β of approximately 28.3^o [23], and their coordinate transformations from wheel's frames to body frame are shown in appendix B.3.

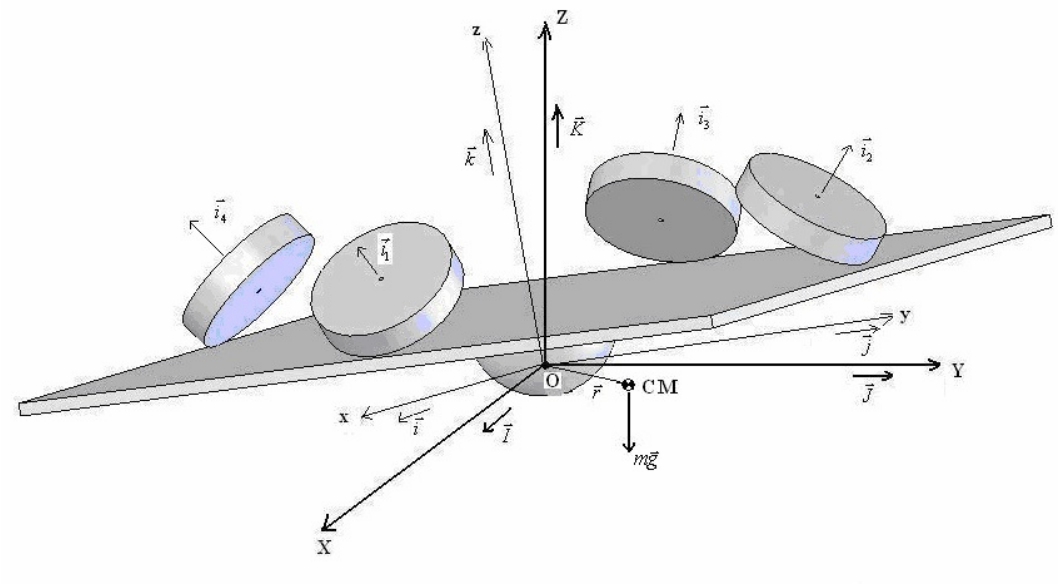


Figure 3.1: Reaction Wheel Platform Schematic

3.1 Equations of Motion

In Healy's thesis, he assumed the platform is a rigid body, and began with Euler's momentum equation to describe the platform's rotational motion. With M as the external torque to system; J as the platforms mass moment of inertia; ω as the platforms angular rates; R as the transformation matrix that brings each wheel torque into platforms torque; I as wheels inertia, and ω_{Wi} as the speed for each wheels, the general equations of motion for a pyramidal reaction wheels platform can be described as:

$$\vec{M} = J\dot{\vec{\omega}} + \vec{\omega} \times J\vec{\omega} + \sum_{i=1}^{4}(R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix} + \vec{\omega} \times R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\omega_{Wi} \\ 0 \\ 0 \end{bmatrix}) \qquad (3.1)$$

Equation (3.1) simply states that the external torque is equal to the rate of

change of platform's and wheels' angular momentum. The external moment to the platform for an air bearing simulator is the torque generated when the center of mass (CM) is not aligned with center of rotation (CR). The offset between CM and CR produces a moment since gravity will be acting at center of mass. The equation for the external moment by center of mass offset is given as

$$\vec{M} = \vec{r} \times R_E.m\vec{g} \tag{3.2}$$

where r is offset between CM and CR, R_E is the Euler rotation matrix which is computed by integrating the platforms angular rates, m is the platforms total mass, and g is the gravity vector.

$$J\dot{\vec{\omega}} + \vec{\omega} \times J\vec{\omega} - \vec{r} \times R_E.m\vec{g} + \sum_{i=1}^{4} (R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix} + \vec{\omega} \times R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\omega_{Wi} \\ 0 \\ 0 \end{bmatrix}) = 0 \tag{3.3}$$

Combining the equations (3.1) and (3.2), equation (3.3) describes the motion of the platform under the influence of gravity. In control application, it is important to note that when a motion is settled i.e. angular rates are zero, the first two terms of on the left side of equation (3.3) will become 0 which leaves the wheel torques equal to gravity torque. This means the controller will try to command the reaction wheels to compensate for the external torque caused by gravity. After a period of simulation, the wheels will reach saturation level without momentum unloading; therefore, it is crucial to remove the offset between center of mass and center of rotation. The first three terms of equation 3.3 are used as the plant's dynamics while the last two terms are defined as control inputs to the system. The control inputs are the reaction wheels' commanded speed,

and the output of the system will be platform's angular rates measured by the LN-200 IMU. Since the states of the platform can be measured, the numerical simulation will use the Full State Feedback controller.

3.2 Quaternion Kinematics and Control Input

In spacecraft simulation and computer 3D graphic modeling, Euler angle kinematic is not preferred due to singularities when integrating Euler angle's rates. Any two (or more) successive rotations about three orthogonal unit vectors in Euclidean space can be represented by a single eigenaxis rotation about the eigenvectors with unity eigenvalues [24]. This leads to the introduction of quaternion. Quaternion kinematics is defined by Wie [3] as:

$$\begin{aligned} \dot{\vec{q}} &= \tfrac{1}{2}(q_4\vec{\omega} - \vec{\omega} \times \vec{q}) \\ \dot{q}_4 &= -\tfrac{1}{2}\vec{\omega}^T\vec{q} \end{aligned} \tag{3.4}$$

where the quaternion is $q = [q_1, q_2, q_3, q_4]^T = [\vec{q}, q_4]^T$ with $\vec{q}$ is the vector part and q_4 is the scale parts. $\vec{q}$ represents the direction (eigenvectors) that a body needs to rotation to achieve a rotation, and q_4 is defined by as $\cos(\theta/2)$. The half angle means that it takes $\theta = 4\pi$ to come back the original position. Quaternion is constrained by the following relationship: $q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$. The quaternion error, defined below, will be used to compute the command body torque at each

time step for the Full State Feedback (FSFB) controller.

$$\begin{bmatrix} \vec{q}_e \\ q_{4e} \end{bmatrix} = \begin{bmatrix} q_{4c} & q_{3c} & -q_{2c} & -q_{1c} \\ -q_{3c} & q_{4c} & q_{1c} & -q_{2c} \\ q_{2c} & -q_{1c} & q_{4c} & -q_{3c} \\ q_{1c} & q_{2c} & q_{3c} & q_{4c} \end{bmatrix} \begin{bmatrix} \vec{q}_a \\ q_{4a} \end{bmatrix} \tag{3.5}$$

where q_c is the commanded quaternion and q_a is the actual quaternion. More details about quaternion kinematics derivation can be found in Mittelsteadt's Master Thesis [23]. It is important to note that equations (3.4) is nonlinear; therefore Mittlesteadt linearized the equations around the quaternion's origin $q = [0, 0, 0, 1]^T$. Then the linearized quaternion kinematics equations can be represented with a second order differential mass-damping system of the quaternion error as:

$$\frac{d^2\vec{q}_e}{dt^2} + CJ^{-1}\frac{d\vec{q}_e}{dt} + \frac{1}{2}KJ^{-1}\vec{q}_e = 0 \tag{3.6}$$

with C and K are the gains for the differential equation. K and C are related to the platform's damping, ζ, and the natural frequency, ω_n, by:

$$\begin{aligned} K = 2J\omega_n^2 &= \begin{bmatrix} 2J_{xx}\omega_n^2 & 0 & 0 \\ 0 & 2J_{yy}\omega_n^2 & 0 \\ 0 & 0 & 2J_{zz}\omega_n^2 \end{bmatrix} \\ C = 2J\zeta\omega_n &= \begin{bmatrix} 2J_{xx}\zeta\omega_n & 0 & 0 \\ 0 & 2J_{yy}\zeta\omega_n & 0 \\ 0 & 0 & 2J_{zz}\zeta\omega_n \end{bmatrix} \end{aligned} \tag{3.7}$$

From classical control, the damping, ζ, and natural frequency of the system ω_n, can be solved by selecting corresponding settling time, and percentage steady state error. Combining the states equations 3.3- 3.6 , the command input torque of the system is defined as:

$$u = \vec{T_c} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = -K\vec{q_e} - C\vec{\omega} \tag{3.8}$$

Mittelsteadt [23] shows in his thesis that the controller above is asymptotically stable by the defining a Lyapunov function:

$$V(x) = \frac{J_{xx}\omega_x^2}{2k_1} + \frac{J_{yy}\omega_y^2}{2k_2} + \frac{J_{zz}\omega_z^2}{2k_3} + q_1^2 + q_2^2 + q_3^2 + (q_4 - 1)^2 \tag{3.9}$$

where $K = [k_1, k_2, k_3]^T$ is the gain of the FSFB given above. It can be shown that $V(x) \geq 0$, $\dot{V}(x) \leq 0$, and $\ddot{V}(x)$ is **bounded**. Therefore, by Lyapunov 2nd Stability Theorem, the controller is stable with any gain $K \quad and \quad C \geq 0$ given in equation (3.7).

3.3 Reaction Wheels Torque Distribution

With the control law for numerical simulation defined, this section will show how the wheel torques can be calculation using the commanded body torque.

Another way to write equation (3.8) is:

$$u = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = -\sum_{i=1}^{4} (R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix} + \vec{\omega} \times R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\omega_{Wi} \\ 0 \\ 0 \end{bmatrix}) \tag{3.10}$$

Then the command body torque and wheel's gyroscopic torque can be combined as:

$$\begin{bmatrix} \hat{T}_x \\ \hat{T}_y \\ \hat{T}_z \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} + \sum_{i=1}^{4} (R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix}) = -\sum_{i=1}^{4} (\vec{\omega} \times R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\omega_{Wi} \\ 0 \\ 0 \end{bmatrix}) \tag{3.11}$$

Now, the right hand side of equation 3.11 can be analyzed further. Sidi [24] define the rate of angular momentum of the wheel as:

$$\vec{T} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} I_{11}^{W1}.\dot{\omega}_{W1} \\ I_{11}^{W2}.\dot{\omega}_{W2} \\ I_{11}^{W3}.\dot{\omega}_{W3} \\ I_{11}^{W4}.\dot{\omega}_{W4} \end{bmatrix} \tag{3.12}$$

Since each wheel is inclined at angle $\beta = 28.3^o$, the Hamiltonian of the torque and distribution of wheel torques for each body axis is given by Sidi:

$$H = \sum_{i=1}^{4} (T_i^2)$$

$$\begin{aligned} T_{cx} &= T_1 \cos\beta - T_3 \cos\beta \\ T_{cy} &= T_2 \cos\beta - T_4 \cos\beta \\ T_{cz} &= T_1 \sin\beta + T_2 \sin\beta + T_3 \sin\beta + T_4 \sin\beta \end{aligned} \tag{3.13}$$

Then the relationship between the wheel torques and $[\hat{T}_x, \hat{T}_y, \hat{T}_z]^T$ is given as:

$$\begin{bmatrix} \hat{T}_{cx} \\ \hat{T}_{cy} \\ \hat{T}_{cz} \end{bmatrix} = \begin{bmatrix} T_{cx}/\cos\beta \\ T_{cy}/\cos\beta \\ T_{cz}/\sin\beta \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{3.14}$$

Equation (3.14) cannot be inverted to find the wheel torques since the matrix is not squared. Sidi introduces Lagrangian method to minimize the norm of $[T_1, T_2, T_3, T_4]^T$ and apply pseudo-inverse to obtain the following equation for the wheel torques:

$$\begin{bmatrix} \hat{T}_{cx} \\ \hat{T}_{cy} \\ \hat{T}_{cz} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{3.15}$$

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 1 & \frac{1}{2} & -\frac{1}{2} \\ -1 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & -1 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} \hat{T}_{cx} \\ \hat{T}_{cy} \\ \hat{T}_{cz} \\ 0 \end{bmatrix} \tag{3.16}$$

Combined the results from sections (3.2), the wheel torque can be found by calculating the body torque with desired damping and settling time and converted to wheel frame using equation (3.16). The simple flow diagram of the simulation is shown in figure 3.2.

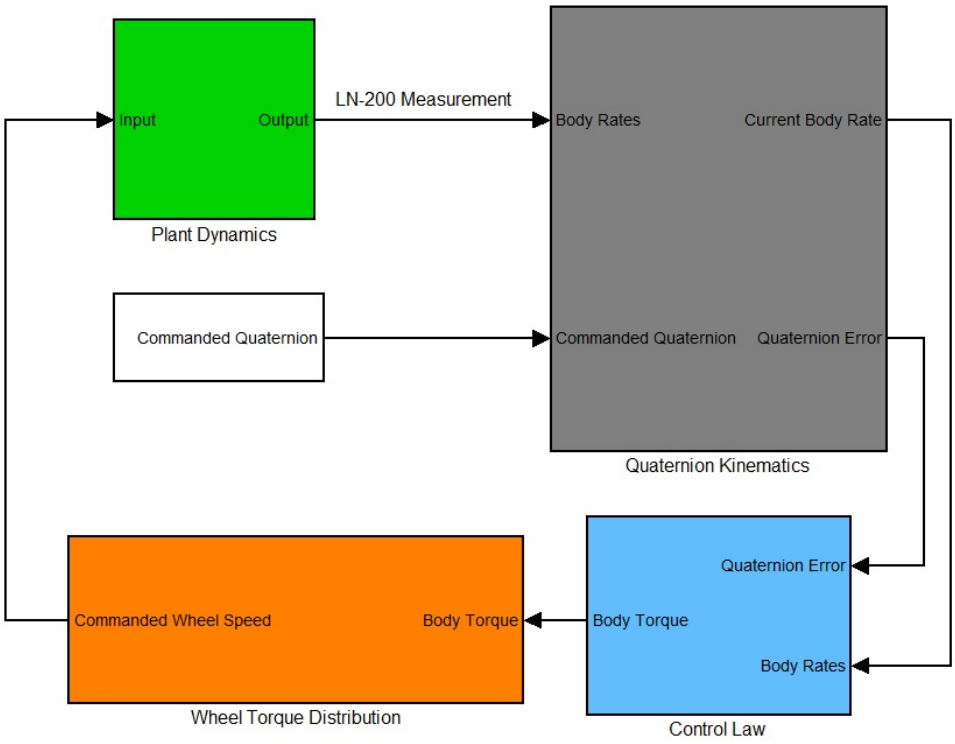


Figure 3.2: Simulation Flow Diagram

3.4 Mass Properties Identification

System identification is the process of developing or improving a mathematical representation of a physical system using experimental data [9]. For an air bearing simulator, the moment of inertia and the location of the center of mass must be well-known in order to eliminate undesired torques due to gravity. As shown in section (3.2), the control law for FSFB requires accurate inertia tensor to computes the K and C gains. Though 3D modeling programs are great for initial design, the models can often be found to be incorrect as much as 10% [9]. The inaccuracies includes, but not restricted to: lack of component and subcomponent parts and wire hardness modeling, non-uniform material properties, and mismodelling. There are several techniques for system identification such as identification using observer/Kalman filter, classical recursive least-square filter, or frequency-domain/State-space system identification. Since the simulator already use reaction wheels as actuators, the wheel speeds can be used as the inputs to the system and the LN-200 will provide the outputs of the system with body rates measurement. Using the equations of motion shown in previous section, Healy manipulated the equation to fit a least-square regression model. The general least-square model is:

$$Ax - T = 0 \tag{3.17}$$

Equation (3.18) defines the mass inertia tensor components and the torque from gravity when the center of mass is not aligned with center of rotation.

$$x = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} & J_{yy} & J_{yz} & J_{zz} & mgr_x & mgr_y & mgr_z \end{bmatrix}^T \tag{3.18}$$

Then each term of the least-square estimation (LSE) can be defined as:

$$Ax = J\dot{\vec{\omega}} + \vec{\omega} \times J\vec{\omega} - \vec{r} \times R_E.m\vec{g} \tag{3.19}$$

and

$$T = \sum_{i=1}^{4} (R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix} + \vec{\omega} \times R_{Wi} \begin{bmatrix} I_{11}^{Wi}.\omega_{Wi} \\ 0 \\ 0 \end{bmatrix}) \tag{3.20}$$

The solution to the least-square estimation can be found using pseudo-inverse

$$x = ((A^T A)^{-1} A^T) T \tag{3.21}$$

Note that at each time step, the A and T matrices will be concatenated with the all data from previous time steps. Then A and T can be defined as $A = \begin{bmatrix} A_1 & A_2 & ... & A_k \end{bmatrix}^T$ and $T = \begin{bmatrix} T_1 & T_2 & ... & T_k \end{bmatrix}^T$ with A_k and T_k is data at the k^{th} time step. This setup improves the correlation for the least-square estimation using all the available data points. Sine the right hand side of equation (3.19) consists of all cross products and dot product, the inertia can be uncoupled from the dynamics:

$$\mathrm{A} = \begin{bmatrix} \dot{\omega}_x & (\dot{\omega}_y - \omega_x\omega_z) & (\dot{\omega}_z + \omega_x\omega_y) & -\omega_y\omega_z & (\omega_y^2 - \omega_z^2) & \omega_y\omega_z & 0 & -\cos\theta_1\cos\theta_2 & \sin\theta_1\cos\theta_2 \\ \omega_x\omega_z & (\dot{\omega}_x + \omega_y\omega_z) & (\omega_z^2 - \omega_x^2) & \dot{\omega}_y & (\dot{\omega}_z - \omega_x\omega_y) & -\omega_x\omega_z & \cos\theta_1\cos\theta_2 & 0 & \sin\theta_2 \\ -\omega_x\omega_y & (\omega_x^2 - \omega_y^2) & (\dot{\omega}_x - \omega_y\omega_z) & \omega_x\omega_y & (\dot{\omega}_y + \omega_x\omega_z) & \dot{\omega}_z & -\sin\theta_1\cos\theta_2 & -\sin\theta_2 & 0 \end{bmatrix} \tag{3.22}$$

The input torque is given as:

$$T = \begin{bmatrix} \cos\beta & 0 & -\cos\beta & 0 \\ 0 & \cos\beta & 0 & -\cos\beta \\ \sin\beta & \sin\beta & \sin\beta & \sin\beta \end{bmatrix} \begin{bmatrix} I_{11}^{W1}.\dot{\omega}_{W1} \\ I_{11}^{W2}.\dot{\omega}_{W2} \\ I_{11}^{W3}.\dot{\omega}_{W3} \\ I_{11}^{W4}.\dot{\omega}_{W4} \end{bmatrix} \tag{3.23}$$

$$+ \begin{bmatrix} \omega_y\sin\beta & \omega_y\sin\beta - \omega_z\cos\beta & \omega_y\sin\beta & \omega_y\sin\beta + \omega_z\cos\beta \\ \omega_z\cos\beta - \omega_x\sin\beta & -\omega_x\sin\beta & -\omega_x\sin\beta - \omega_z\cos\beta & -\omega_x\sin\beta \\ -\omega_y\cos\beta & \omega_x\cos\beta & \omega_y\cos\beta & -\omega_x\cos\beta \end{bmatrix} \begin{bmatrix} I_{11}^{W1}.\omega_{W1} \\ I_{11}^{W2}.\omega_{W2} \\ I_{11}^{W3}.\omega_{W3} \\ I_{11}^{W4}.\omega_{W4} \end{bmatrix}$$

When solving the least-square estimation, it is necessary to use the actual wheel speeds measured by the Hall sensor on the motor instead of the commanded wheel speeds. The reason is that the body rates measured by LN-200 is the

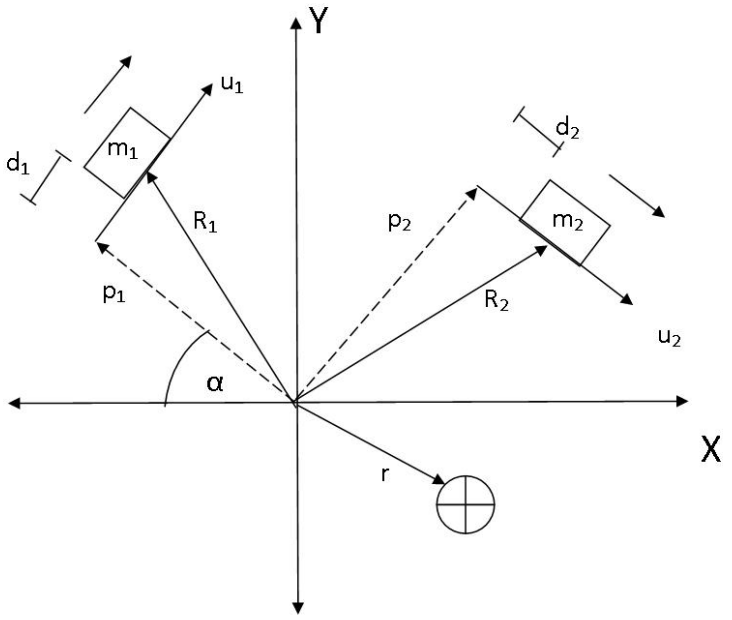


Figure 3.3: Fine mass balance and body frame coordinate illustration

response from the actual wheel speeds combined with other dynamics, and since the wheel controller, MCBL3006S, also has its own proportional integral (PI) control, the wheel speed does not immediately settle to the commanded values. Equation 3.23 allows the user to recompute the input torques using the actual wheels' speeds and angular acceleration.

3.5 Fine Mass Balance Control

Once the CM offset has been identified, we need to derive the relationship between the offset and the distance that the fine mass balance needs to move. This thesis uses the direct mass compensation technique derived by Jae-Jun Kim and Brij N. Agrawal [1] at the Naval Postgraduate school in Monterey CA. Figure 3.3 shows the FMB with respect to the body frame coordinate.

In practice, the mass balance system is often aligned with the axes of the platform's body frame; this however, is not a requirement as in the case of the Cal Poly spacecraft simulator due to the dimension constraints of the platform. If $\vec{u}_1, \vec{u}_2$, and $\vec{u}_3$ are the units vector of the FMBs, then a coordinate transform to body frame is required when the FMBs are not aligned with the body frame's axes. The location for each FMB in body frame is $\vec{R}_i = \vec{\rho}_i + d_i\vec{u}_i$. It follows that the center of mass vector $\vec{r}$ is defined as:

$$\vec{r} = \frac{1}{m}[(m - m_{FMB})R_0 + \sum_{i=1}^{3} m_i R_i] \tag{3.24}$$

where m is the total mass of the platform, $\mathrm{m_{FMB}}$ is mass of FMB, $\mathrm{R_0}$ is the center of mass of the platform when FMB are removed. When the FMB move by a distance Δd, the new location of center of mass is:

$$\vec{r}' = \frac{1}{m}[(m - m_{FMB})R_0 + \sum_{i=1}^{3} m_i(\vec{\rho}_i + (d_i + \Delta d_i)\vec{u}_i)] \tag{3.25}$$

The change in center of mass offset by the movement of the FMB can be found by the difference from equation (3.25) and (3.24).

$$\Delta\vec{r} = \vec{r}' - \vec{r} = \frac{1}{m}\sum_{i=1}^{3} m_i \Delta d_i \vec{u}_i \tag{3.26}$$

If $\hat{r}$ is the center of mass solution from the least-square estimation, then Δr needs to be equal to $-\hat{r}$ in order to remove the offset. To find the distance that each FMB needs to move for an identified offset, equation (3.26) is inverted and

substituting $\Delta r = -\hat{r}$ to obtain:

$$\Delta\vec{d} = \begin{bmatrix} \Delta d_1 \\ \Delta d_2 \\ \Delta d_3 \end{bmatrix} = - \begin{bmatrix} m_1\vec{u}_1 & m_2\vec{u}_2 & m_3\vec{u}_3 \end{bmatrix}^{-1} m\vec{\hat{r}} \tag{3.27}$$

Chapter 4

Results

This chapter details the result from System Identification (System I.D.) for the Simulator's mass properties using the experiment data with the IMU integrated onto the platform. In addition, a validation test for the inertia matrix will also be presented using the Full State feedback control law stated from the previous chapter. The newly identified inertia is used in the Full State Feedback controller experiment test on the simulator and is, later, compared to the computer/numerical simulation model of the platform. Additional controller uncertainty and gravity torque effects will also be discussed.

4.1 Sine-waves Test

In his thesis, Healy [15] demonstrated the computer simulation for the control law and System Identification results based on the estimated values from Solidworks. Healy [15] tested four cases in total: one case to validate System I.D. when there is no noise in angular rate's measurements, and three cases where there is noise in measurement with different center of mass locations. Healy [15]

	Trial 1			Trial 2			Trial 3		
	Simulated Values	Identified Values	Percent Error (%)	Simulated Values	Identified Values	Percent Error (%)	Simulated Values	Identified Values	Percent Error (%)
I_{xx} (kgm^2)	0.65	0.6499	0.0154	0.65	0.65002	0.00308	0.65	0.6498	0.0308
I_{xy} (kgm^2)	0.00045	0.00046	2.22	0.00045	0.00045	0	0.00045	0.00046	2.22
I_{xz} (kgm^2)	-0.00055	-0.00054	1.818	-0.0006	-0.0005	1.818	-0.0006	-0.0005	5.45
I_{yy} (kgm^2)	0.6	0.60005	0.00833	0.6	0.59982	0.03	0.6	0.60002	0.00333
I_{yz} (kgm^2)	0.0015	0.001509	0.6	0.0015	0.00156	3.67	0.0015	0.0015	0.1333
I_{zz} (kgm^2)	0.56	0.55983	0.0304	0.56	0.55999	0.001786	0.56	0.55999	0.001786
r_x (cm)	-0.0165	-0.0165	0	-1	-1	0	0.1	0.1	0
r_y (cm)	0.04	0.04	0	0.05	0.05	0	2	2	0
r_z (cm)	-0.5656	-0.5656	0	-0.5	-0.5	0	-0.5	-0.5	0

Figure 4.1: Healy's results for three cases with noise [15]

showed that when there is no noise in measurement, all the mass properties and center of mass location were properly identified using System I.D. least square algorithm presented in previous chapter. In the present of noise, approximately 0.05 deg/s, the center of mass location were properly identified with no error while the off-diagonal term of the inertia matrix has the highest error at 5.45% as shown in figure 4.1. In summary from Healy's numerical simulation, the mass properties can be identified correctly when the center of mass offset is small, typically in the millimeter range. This is due to the facts that the reaction wheels will eventually reach saturated level, thus cannot compensate for gravity torque, and large gravity torque will also affect controller's stability.

For the actual experiment, a simple motion around any axes will not be sufficient for structural excitation, so a new test is developed aiming to excite all structural modes of the simulator. The simulation diagram for this test is presented in figure 4.2. The test uses three sets of sine-wave generators in Simulink to create certain sine-wave combination for the body rates and to sweep through various structural frequency. Since there is a rotation limit on the x and y axes, the test was carefully iterated to prevent the platform from hitting the air bearing support. The commanded wheel speeds are then be computed using the spacecraft's torque equation, and then feed into motor controllers through xPC target.

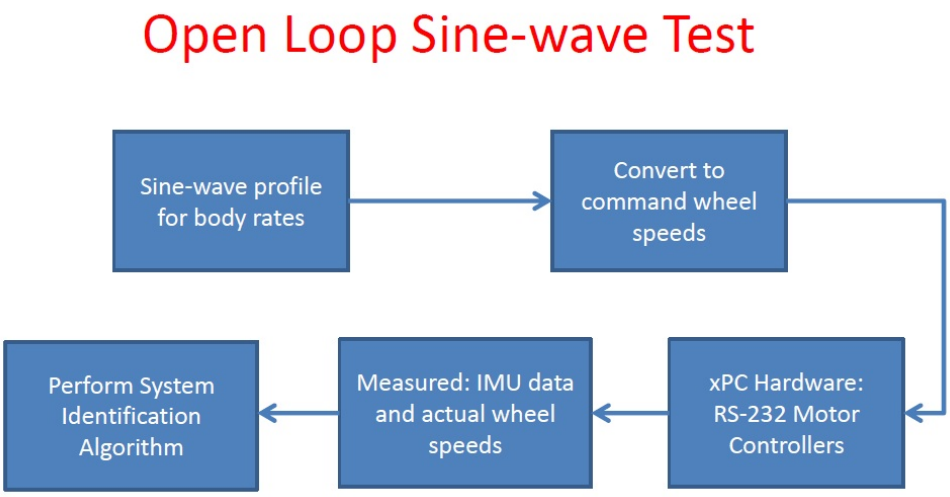


Figure 4.2: Open Loop Sine-wave Test Diagram

The System I.D algorithm is performed using actual wheel speeds measured by Hall sensor on the motor controllers and the LN-200's measured body rates. Another advantage of this test compared to a FSFB test is the ability to impart a large torque on the system in the roll and pitch (X and Y) axes since the sudden change in wheel speeds creates a large momentum and allows the platform to overcome the initial gravity torque.

Figure 4.3 shows the actual wheel speeds and the platform's body rates for a 180 seconds test with the sine-wave generator. The wheel speeds can be seen to reach saturation limit set by the user. The reason for this limit was explained in section 2.3. Thus, the body rates reacted to the wheel speeds change instead of the intended smooth sine-wave profile. This result, however, created a random oscillatory motions for the platform which help identifying most of the structural modes. The random motions of the platform do not hinder System I.D algorithm since the algorithm only relates the actual inputs (measured actual wheel speeds) and the outputs (LN-200 body rates measurements) in the open-loop, and linear

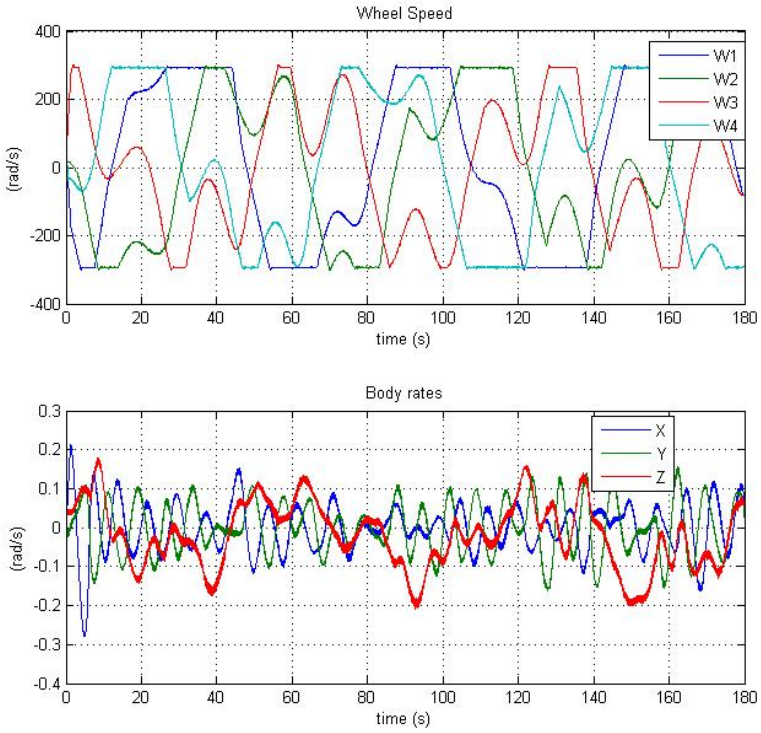


Figure 4.3: Open loop sine-wave test body rates and wheel speeds

system.

To ensure the accuracy of the data, the sine-wave test is repeated numerous times with the same configuration i.e there are no changes in components or wire placement on the platform. In all, 20 tests were performed with System I.D algorithm for statistical purposes, and the demonstrated result of 5 runs for the mass properties is shown in table 4.1. All the main diagonal terms are, at least, one magnitude larger than the off-diagonal terms. This result is consistent with the inertia product predicted with Solidworks, at least in term of magnitude. The values in table 4.1 should be more reliable than Solidworks' estimation with the data from sine-wave test as this test measures the output of the system based on the current platform's configuration. It should be noted that the center of

mass offset is in the millimeter range for the z-axis. The negative values for r_z indicate that the center of mass is below the center of rotation which will make the platform acting like a pendulum. This is the behavior expected from the visual inspection as the platform does not roll toward its sides. One should start with center of mass below center of rotation when starting to experiment in order to balance the X and Y directions with the coarse mass balance. Then, the center of mass could be moving up or down with the two vertical coarse mass balance. It should also be noted that the envelope that the fine mass balance can operate is in sub-millimeter range. Recall from equation 3.27 that if the fine mass balance is placed parallel to one axis, then the maximum distance to move the platform is proportional to the mass ratio of the fine mass balance's steel block and the platform. Since the ratio is 0.0063 (to be exact .195kg/30.91kg) and the maximum operable track length in the FMB is 100 mm, the maximum center of mass offset compensation is 0.63 mm if the FMB is parallel to the body axis, and the steel block would have to start at the end of the lead screw. Since one of the FMB is placed vertically on the platform this ratio can be conveniently used if both x and y-axes are balanced. In table 4.1, both r_x and r_y are, precisely, at the center of rotation, and r_z is also sharply below the center of rotation considering the magnitude is millimeter. Many attempts to move the center of mass up (in the z-direction) proved to be more difficult as the platform will begin to roll to its side like an inverted pendulum. At this point, it was difficult to determine if the x and y-axes are balanced since any small external forces will knock the platform to its sides. Through some visual observations and system ID results, the range for r_z provided in table 4.1 is a good starting point to operate in the future with the fine mass balance. Since the current steel block on the fine mass balance weighs significantly less than the coarse balance mass, future student

Table 4.1: Samples mass properties identified from Sine-wave data

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Jxx(kgm^2)	**0.6225**	**0.601**	**0.6215**	**0.6214**	**0.6042**
Jxy(kgm^2)	**0.0279**	**0.0284**	**0.022**	**0.0256**	**0.0242**
Jxz(kgm^2)	**0.0146**	**0.0118**	**0.012**	**0.0187**	**0.0134**
Jyy(kgm^2)	**0.6662**	**0.6429**	**0.6544**	**0.6571**	**0.6543**
Jyz(kgm^2)	**0.0038**	**0.0005**	**0.0104**	**-0.0012**	**0.0031**
Jzz(kgm^2)	**0.6462**	**0.6428**	**0.6355**	**0.6416**	**0.6471**
rx(mm)	**-0.0278**	**-0.0304**	**-0.0289**	**-0.0273**	**-0.0284**
ry(mm)	**-0.0127**	**-0.0126**	**-0.0003**	**-0.0064**	**-0.0025**
rz(mm)	**-1.6069**	**-1.5569**	**-1.5811**	**-1.597**	**-1.5767**

Table 4.2: Results for 20 runs with Sine-wave test

	20 runs average	3σ
Jxx(kgm^2)	**0.6071**	**0.0354**
Jxy(kgm^2)	**0.0266**	**0.0095**
Jxz(kgm^2)	**0.0149**	**0.0233**
Jyy(kgm^2)	**0.6560**	**0.0311**
Jyz(kgm^2)	**0.0010**	**0.0172**
Jzz(kgm^2)	**0.6376**	**0.0276**
rx(mm)	**-0.0207**	**0.0327**
ry(mm)	**-0.0112**	**0.0172**
rz(mm)	**-1.581**	**0.0707**

should implement motors on the coarse masses and remove the fine mass balance system. This allows a precise control of center of mass location with greater range since the coarse masses weigh more than the steel block on the fine mass balance.

The average and standard deviation were computed and the 3σ error bars are plotted for each test result on the figure 4.4 and table 4.2. Appendix A shows the plotted errors bars the off-diagonal terms. The off-diagonal terms, presented above, are particularly small. This feature matches well with the intended design as the platform is setup to operate with the main diagonal terms in the control laws. Since these off-diagonal terms are small, any slight changes in magnitude

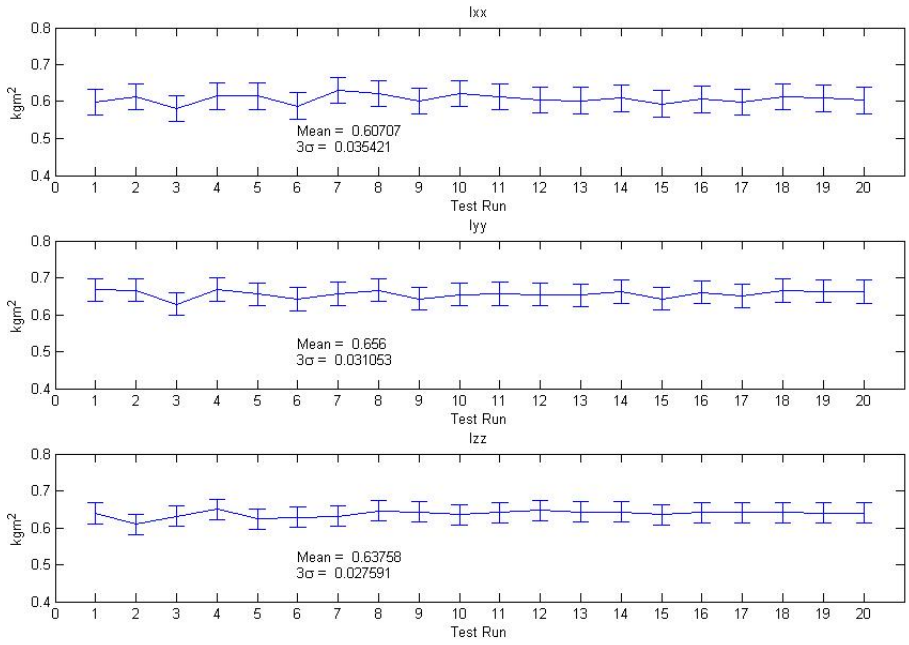


Figure 4.4: Inertia Result from System I.D. for main diagonal terms.

will become large error as seen in the standard deviation in 4.2. In all, the values for inertia matrix were identified with the least square method and this result provides a good starting values for the inertia matrix until the LN-200 is fixed and the System I.D. is setup for autonomous operation. The next chapter will describe the convergence of the mass properties system identification method.

4.2 System Identification Convergence

For the System I.D. experiment, the test was run at 50 Hz over 180 seconds which results in 9000 data points for the measured parameters. Recall that the System I.D. method was set up such that the mass properties can be solved at each time step using the data from previous step. Then the convergence for identified mass properties using least square method can be examined by solving the least square problem at each time step and plotting the result. Figure 4.5 shows the convergence for the main diagonal terms while figure 4.6 shows the zoom-in feature of the previous figure. And appendix A shows the convergence for the off-diagonal terms.

In figure 4.6, the main diagonal J_{xx} and J_{yy} converged well before 10 seconds while J_{zz} took approximately 80 seconds to settle. In the first few seconds, the parameters overshot, then slowly converge to the final values. This result indicates that the simulation time of 180 seconds might be adequate for System I.D test. The reason for J_{zz} slow convergence rate is that it took the platform longer to rotate around the z-axis (full 360^0) as opppose to only $\pm 30^0$ on x and y axes. In Healy's numerical simluation, the off-diagonal term did not converge after 150 seconds due to the magnitude of these terms being small. Similarly, J_{yz} of the platform's inertia also did not converge after 180 seconds while J_{xy}

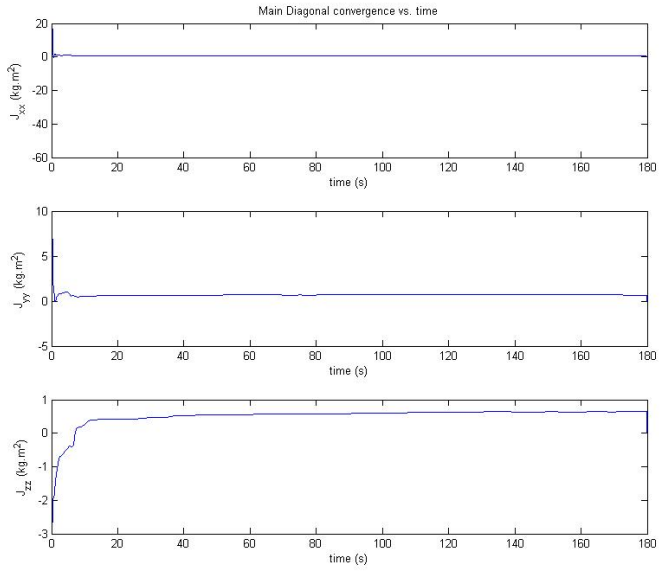


Figure 4.5: Convergence from System I.D. for main diagonal terms.

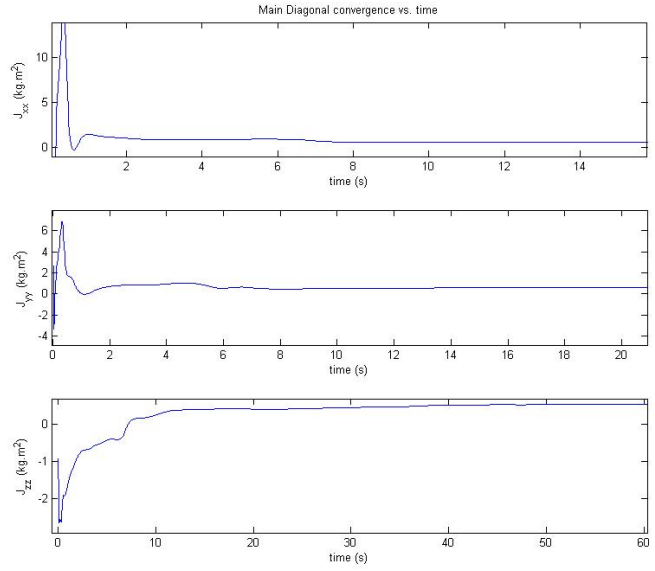


Figure 4.6: Zoom-in Convergence from System I.D. for main diagonal terms.

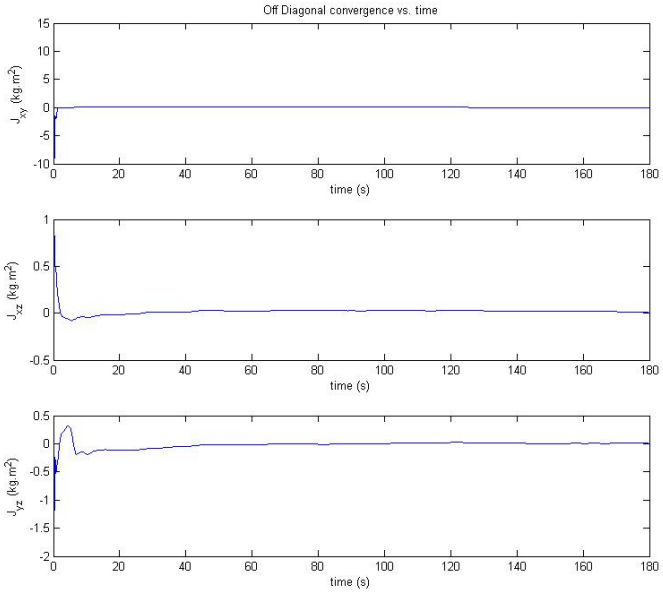


Figure 4.7: Convergence from System I.D. for off-diagonal terms.

and J_{xz} appears to converge after 150 seconds. Since these terms are relatively small compared to the main diagonal terms (at least 20 times), they are more susceptible to large error due to smaller magnitude.

Perhaps, the most important parameters of mass properties are the center of mass locations in order to correctly balance the platform. Figure 4.9 shows the zoom-in for center of mass convergence. The center of mass locations converges within 5 seconds. This same convergence rate was also seen in Healy's numerical simulation. Both tests suggested that the center of mass locations can be quickly uncoupled from the torque, and converge faster than other mass properties parameters.

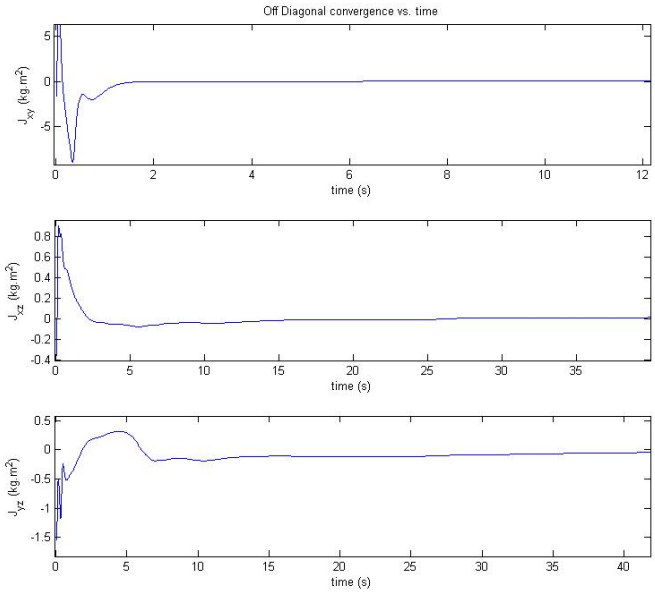


Figure 4.8: Zoom-in Convergence from System I.D. for off-diagonal terms.

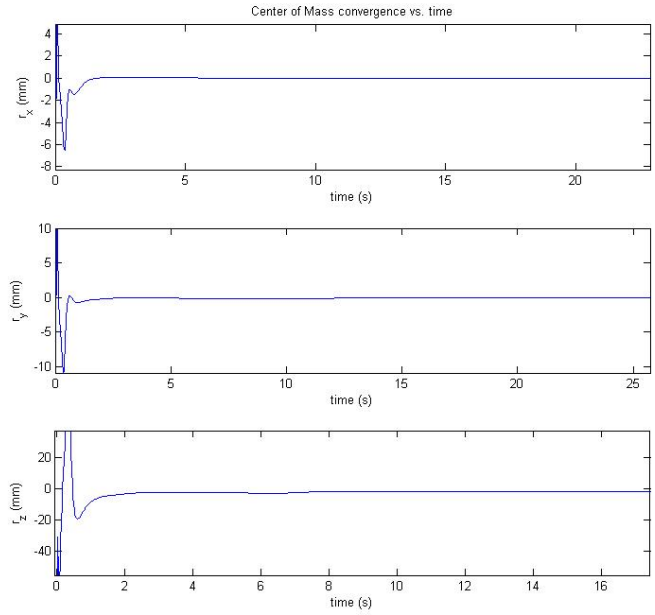


Figure 4.9: Zoom-in Convergence from System I.D. for center of mass locations.

4.3 Validation Case for System I.D. result

As mentioned from previous chapter, Healy's computer simulation observed some errors for System I.D. in the present of sensor's noise [15]. A computer model was easier to validate the result than and actual experiment since the inertia was an assumed value used to calculate the spacecraft's torque equation. The System I.D. result obtained in computer model can be compared directly to the assumed inertia values. However, many challenges arose for System I.D validation in experiment. Comparing to numerical simulation, experiment tests in general are more difficult to set up since there are more variables to consider when performing a test. One of the common issue is the sampling rate for each devices on the platform. LN-200's default manufacturer's setting is 200 Hz while the reaction wheels sampling rate is varied with baud rates and commands. An issue arose in the previous experiment test by Silva [16] was the Bluetooth communication could only sample the data at 1Hz. Any other sampling rate would cause the server to crash, thus this low sampling rate affected the fidelity of the data. The current configuration of the platform allows the test to be run at higher rate with xPC target in real time. Since xPC is a separate environment from Simulink and does not require extensive memory and processing power, as it does not have any operating systems, the test can be executed by the PC-104 and the data are uploaded to the host computer after the test completes. The identified inertia and center of mass location obtained in a System I.D. test need to be validated with the FSFB controller described in section 3.2. A numerical simulation of the FSFB controller was run with the new identified inertia, and then compared to an actual motion test. Two test cases were chosen for the motion test with FSFB controller: 90^0 yaw counter-clockwise (postive rotation around Z-axis, looking down on the X-Y plan) and 90^0 yaw clockwise. At first, only a counter-clockwise

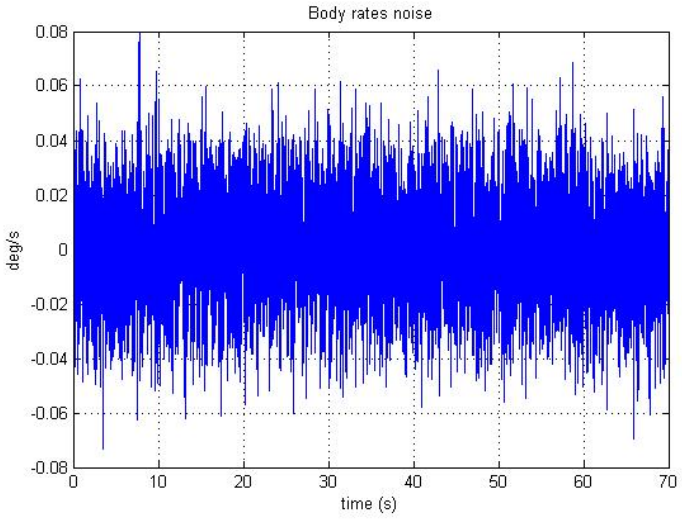


Figure 4.10: White noise (RMS) added to angular rates measurements

direction test was selected; however, additional effects of air-bearing support simulator were detected, therefore a clockwise direction test was performed to observe some of these effects which will be describe later.

To simulate a realistic experiment, a noise should be added to the angular rates measurement. Figure 4.10 shows the white noise level, approximately 0.05 deg/s (rms), that is added to the measurement. This noise level should be similar to the noise from the LN-200 angular rates measurements.

For System I.D algorithm validation test, the platform is commanded to a 90^0 yaw motion (around z-body axis) since the pitch and roll axes are limited to $\pm 30^0$ for an air bearing support system. The equivalent quaternion command is $q = \begin{bmatrix} 0 & 0 & 0.7071 & 0.7071 \end{bmatrix}$. Settling time of 20 seconds, damping ratio of 0.8, and percent steady state error of 2% are chosen to calculate the K and C gains for the control law in equation 3.7. In this experiment, the whole system will be set to 50 Hz sampling rate; however, the LN-200 is still running at 200 Hz internally, and the data will be then reduced to 50 Hz by Simulink's rate transition block.

This is because Simulink requires all the blocks at the highest level to match sampling rate. xPC target allows components to operate either asynchronously and synchronously at different rates to make sure high fidelity data such as those from the LN-200 can be captured. It is important to note that down-sampling, especially at multiple integer of sampling rate, will keep the fidelity of the data. The simulation is setup for 40 seconds: the platform stays at the quaternion origin during the first 5 seconds, then execute a 90^0 yaw from 5 to 40 seconds. Each wheel is biased at 150 rad/s at the start of the simulation as shown in figure 4.11. When biasing the wheel, the platform was help at rest position. From 5 to 40 seconds, the wheel speeds vary indicating the momentum change on the platform. In both cases, each wheel began to diverge, instead of staying on top of each other, which indicates that there is an external torque acting on the platform. The controller then compensated for this external torque which made the wheel starting to diverge in order to stay at the commanded attitude. This external torque could come from several sources: angle β (wheel inclination) might not be at 28.3^0, uneven air flow in the air-bearing, vibration from holding aluminum block of the wheels, and/or air flow in the room. A closer look at the differences between the wheel speeds at the end of both test cases support the possibility of angle β is not at 28.3^0. In addition, visual inspection indicated that the platform tends to drift clockwise around Z-axis when the wheels are biased, regardless of wheels' direction. This observation further support the possibility of faulty spacing (90^0 apart around z-axis) and/or inclination of the reaction wheels.

In all, 4 test cases were performed for each direction in order to validate the System I.D. results: numerical simulation with the average values of the mass properties terms, platform test with average values $\bar{x}$, $\bar{x} + 3\sigma$ values, and $\bar{x} - 3\sigma$ values of the mass properties terms. The errors were added the mean state

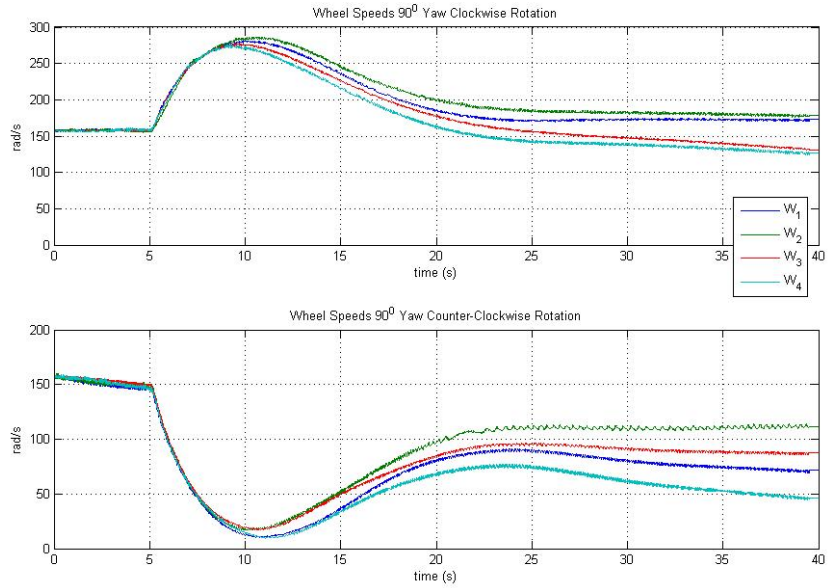


Figure 4.11: Actual wheel speeds for both counter-clockwise and clockwise rotation

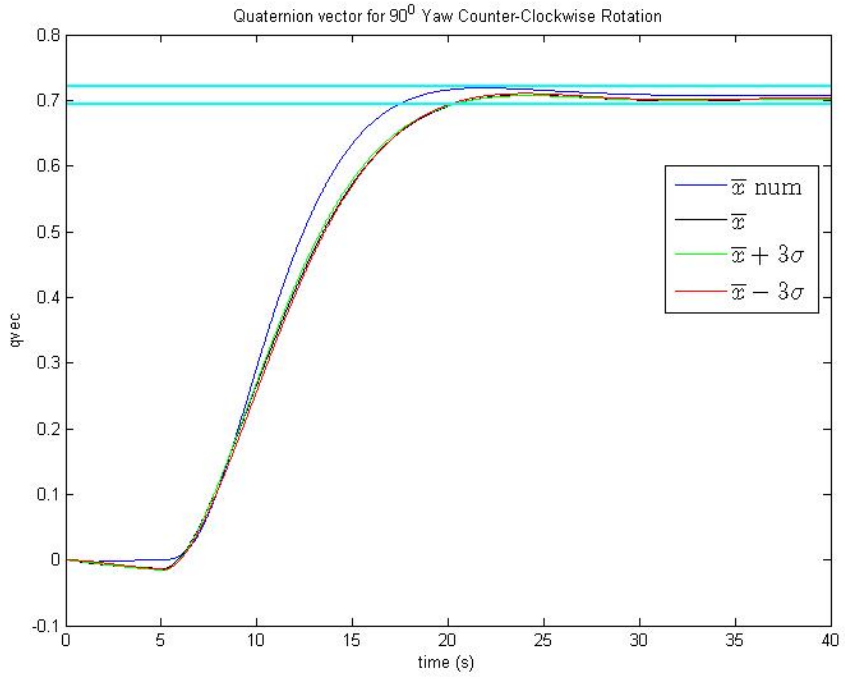


Figure 4.12: Quaternion vector for 90^0 yaw in counter-clockwise rotation

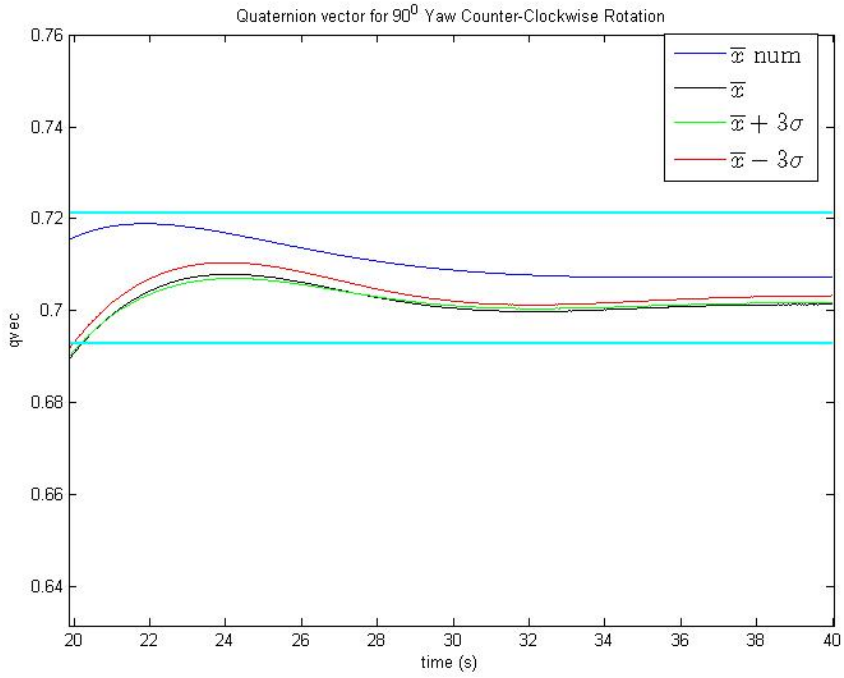


Figure 4.13: Zoom-in quaternion vector for 90^0 yaw in counter-clockwise rotation

values instead of adding to the experimental result after the test. Therefore, one should not expect the test result of the inertia's mean values to be always bounded between $\bar{x} \pm 3\sigma$ test cases. The results for quaternion and wheel speeds were then plotted together on the same plots for comparison. Figure 4.12 shows the quaternion vector of 4 test cases for counter-clockwise 90^0 yaw test while figure 4.13 zooms in the last 20 seconds of the test. These 2 figures show that FSFB tests on the platform followed the numerical simulation, blue line, at the beginning (from 5 to 10 seconds). The platform tests began to lag behind the numerical simulation, and then settled (although still within the specified 2% steady state error) below the numerical simulation. This result combined with the wheel speeds shown in figure 4.11 support the existence of the external torque described previously. The platform test with average values is in between the $\bar{x} + 3\sigma$ and $\bar{x} - 3\sigma$ values of the mass properties terms. Another reason for the

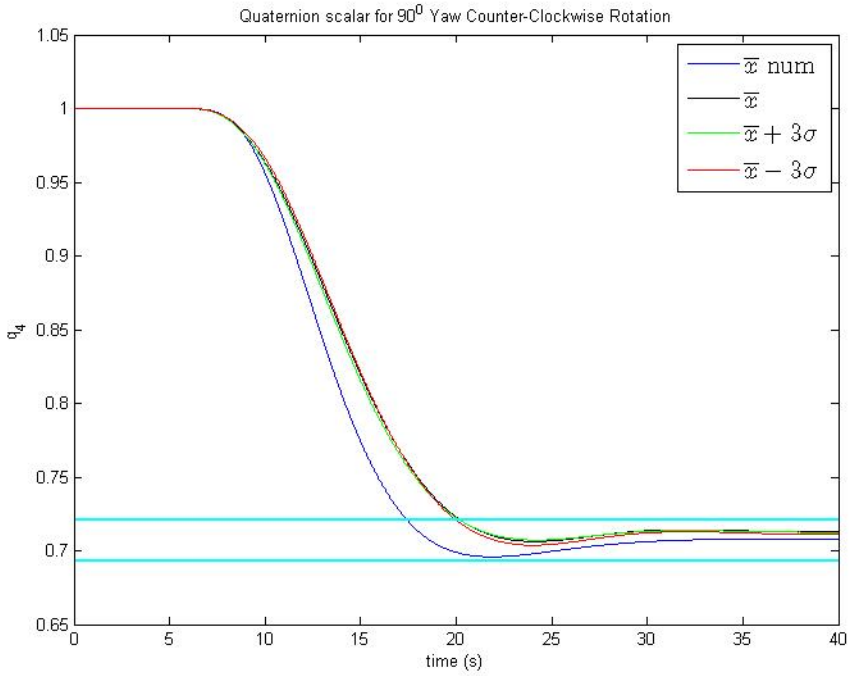


Figure 4.14: Quaternion scalar for 90^0 yaw in counter-clockwise rotation

lag between numerical simulation and platform tests is the wheel model. The numerical simulation assumed a second order differential model for the wheels while the actual motor controller's model is undetermined. In addition, there is an acceleration limit set for the wheel as described in section 2.3. Figure 4.12 and 4.13 show the quaternion scalar and zoom-in version of the same plot. In these plots, all cases settled perfectly before 25 seconds within 2% steady state error. An important note is that if the test was to be run for longer period, the wheels will eventually reach saturation limit since there exists an external torque mentioned previously. At that point, the controller will not be able to maintain attitude control.

Similarly, figure 4.16 and 4.17 show the quaternion vector for the 90^0 yaw clockwise rotation while figure 4.18 and 4.19 show the quaternion scalar. For

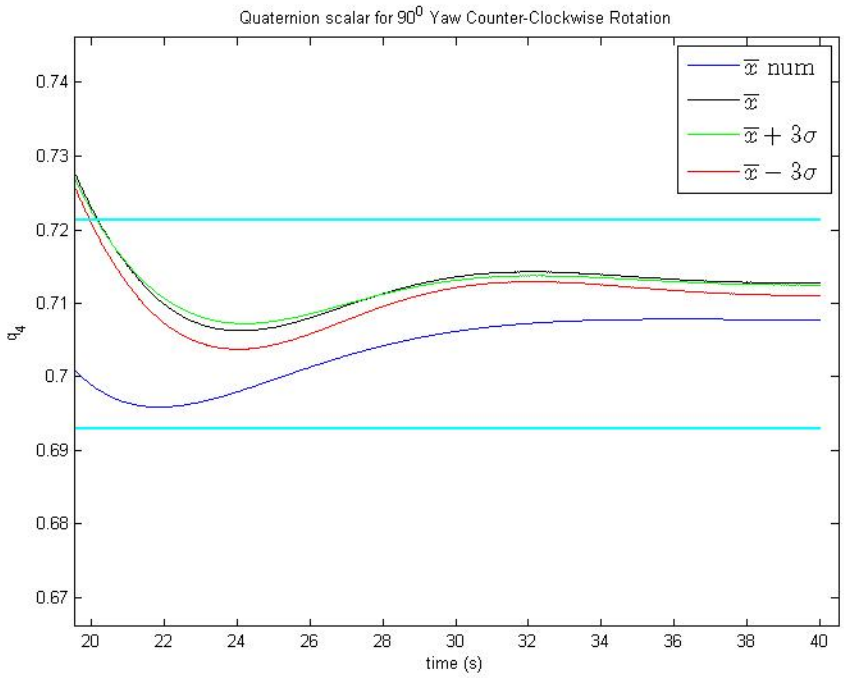


Figure 4.15: Zoom-in quaternion scalar for 90^0 yaw in counter-clockwise rotation

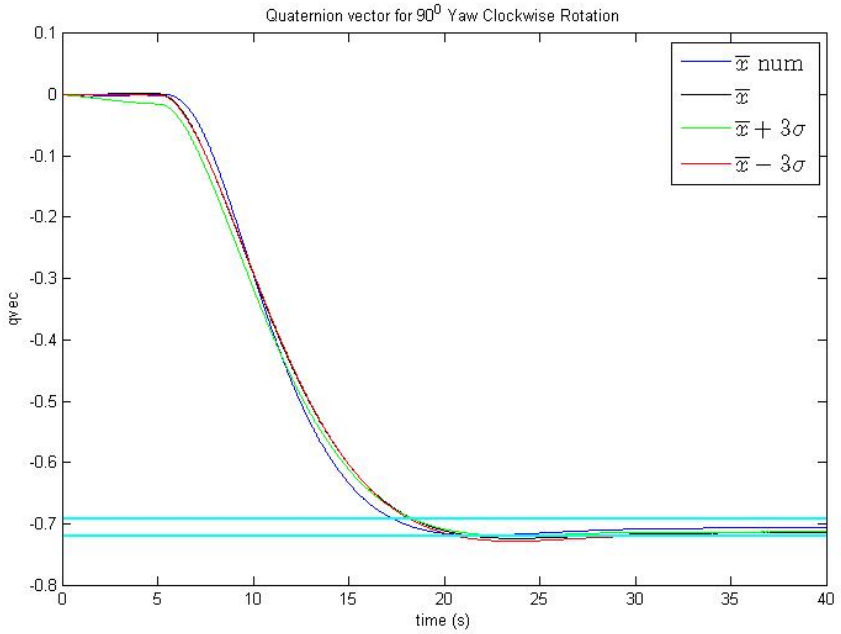


Figure 4.16: Quaternion vector for 90^0 yaw in clockwise rotation

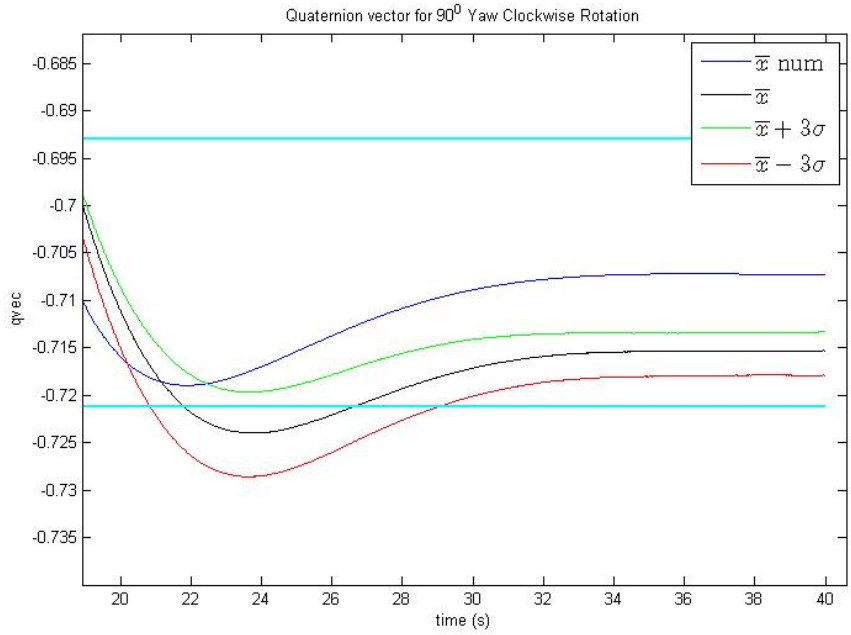


Figure 4.17: Zoom-in quaternion vector for 90^0 yaw in clockwise rotation

this rotation, only the $\bar{x} + 3\sigma$ case settled within the 20 seconds settling time. It can be seen that all three test cases with statistical result of mass properties also created the error bound for the platform's test. Similarly, only the $\bar{x} + 3\sigma$ test case settled for the quaternion scalar in figure 4.19. This result suggested that the actual mass properties may be closer to $\bar{x} + 3\sigma$ than the average values of the mass properties terms.

One intriguing result of the experiment test is that the gravity torque can be observed when the motion of the platform has stopped or settled to a commanded attitude. In figure 4.20 between 30 and 40 seconds, the gravity torque can be seen which indicating that there is still a center of mass offset from center of rotation. Further evidence of this torque can be seen in figure 4.11 of the reaction wheels' speeds. Since there is a gravity torque, the control law commanded two of the four wheels to a higher speed to compensate for this torque. In this case, if the

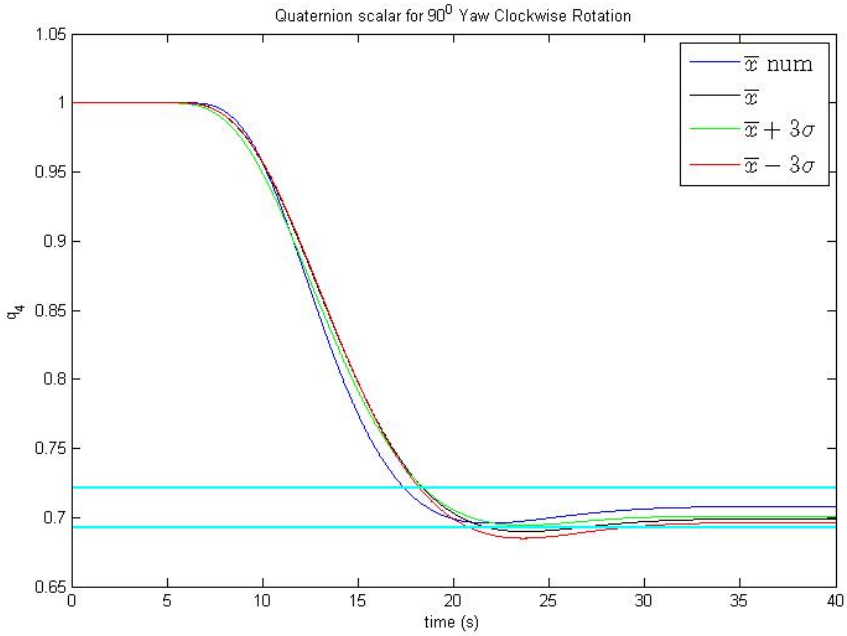


Figure 4.18: Quaternion scalar for 90^0 yaw in clockwise rotation

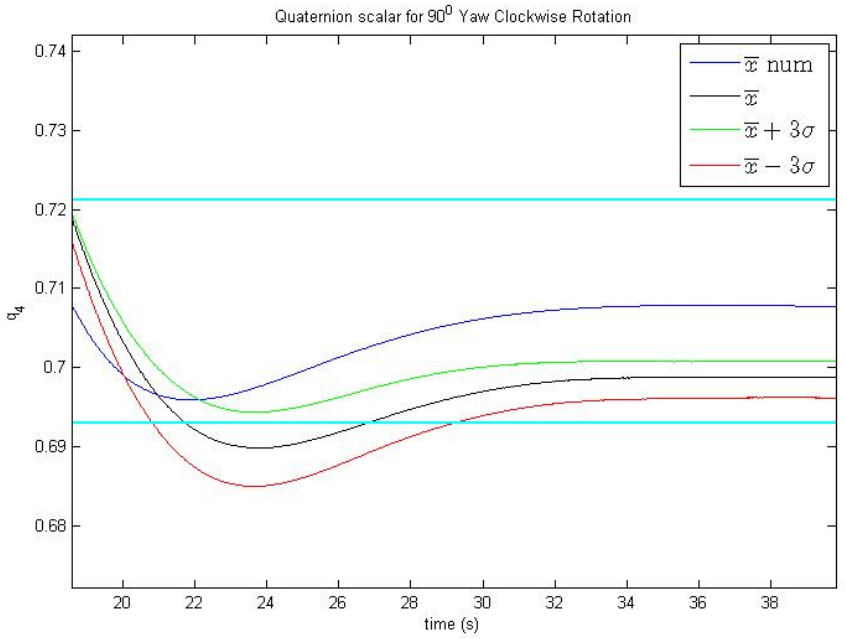


Figure 4.19: Zoom-in quaternion scalar for 90^0 yaw in clockwise rotation

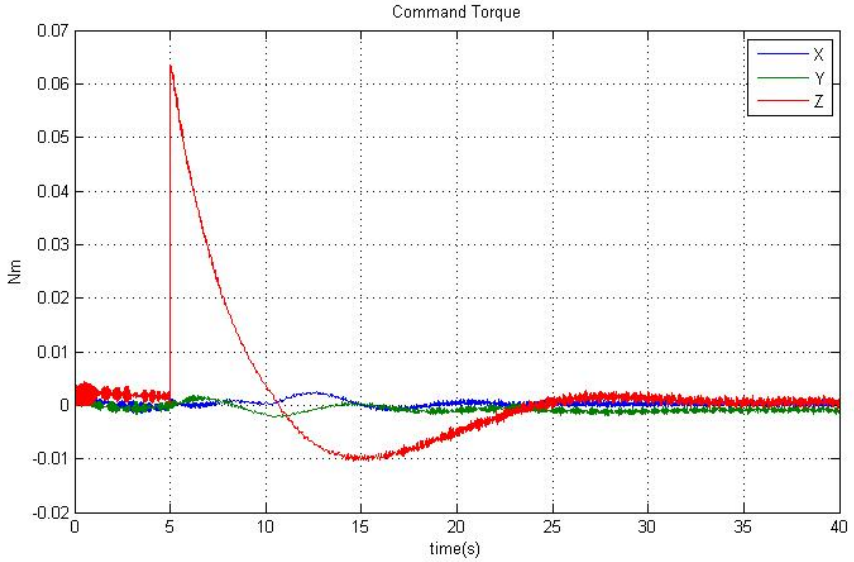


Figure 4.20: Command torque for a 90^0 yaw rotation

test was to be run for a longer period, the wheel speeds will eventually reach saturated level. When there is no gravity torque, the speeds of four wheels are to be nearly identical as seen in numerical simulation.

Using the system ID algorithm, the mass properties are identified and presented in table 4.2. When performing FSFB around the roll or pitch axis (X or Y), the wheels saturated quickly due the center of mass offset. The reason is that the platfom is now a pendulum and it will try to restore its position around roll and pitch. A yaw test does not experience this behavior since it does not have a restoring motion of the pendulum. Therefore, a test with FSFB around the roll and pitch requires a large torque in in those directions.

Chapter 5

Conclusion

Several pre-set goals for the Spacecraft Simulator projects were achieved throughout the course of this thesis. Implementation of the IMU LN-200 was a big step to ensure angular rate measurement quality for upcoming projects. In addition, System I.D. method was also performed to retrieve the mass moment of inertia and center of mass' location. Although the center of mass' location was identified, the goal to reduce the offset in the z-axis within 1 millimeter was not met. The reason is due to the mass of the fine balance block is significantly smaller comparing to the mass of the platform. Suggestion for improvement to meet this goal can be found in the next chapter.

In addition to implementing the IMU LN-200, the Cal Poly Spacecraft Simulator takes advantage of xPC Target software from Mathworks for minimal memory usage without having to use the target's Matlab on Windows operating system like previous configuration. xPC Target offers solution for small, and compact PC/104 computer form factor with easily customized drivers. Additionally, it also eliminates the need to load a remote desktop session as the host computer can easily scopes and monitor the target with Matlab GUI on the host computer.

The four reaction wheels motor have also been upgraded from analog to serial communication for ease to operate and maintenance. Any plug-and-play scenarios with the four wheels could easily be implemented by removing the RS-232 cable and modifying the Simulink's blocks. With xPC implemented, the LN-200 IMU is now also operational on the platform providing the angular rates measurement for any experiments. Combining both hardware and software improvement, one can achieve the goal of testing a control law for spacecraft application efficiently and repeatedly. Comparing to the last project on the Simulator by Kinnet [25], the platform can now operate for longer time (minimum 5-10 minutes continuously as opposed to 1-2 minutes, previously).

After hardware and software implementation, the sine-wave test and Full State Feedback controller test were also implemented, the results were satisfying. Gravity torque can be observed in the commanded torque and is consistent with the defined torque equation for the platform. The present of gravity torque indicates there is an offset from center of mass to the center of rotation, thus it is important to identify the offset and inertia products for the platform. The mass properties identification which utilizes least square technique is then performed on the data to identify the inertia and center of mass offset. The sine-wave test was repeated many times to show the consistency, and validated new inertia values with the FSFB control law.

Chapter 6

Recommendations for future works

The first priority after the completion of this thesis is to investigate the uneven torque seen in the FSFB experiments. This step will improve the result of all the control experiments in the future. The investigation should focus on the reaction wheels' inclination angel β. Since all the wheels are inclined at an angle from the z-axis, there is always a torque about the z-axis direction when the wheels are biased (presumably at the same speed). When the inclination angle are not the same for all the wheels, then there will be an uneven torque causing the platform to drift away from stationary position. The layout of the four reaction wheels around the z-axis should also be investigated. In this work, the author assumed that the wheels are spaced at 90^0 apart around the z-axis based on previous works' assumption. This assumption might not be true since the wheels' spacing has never been verified in any of the previous theses.

Perhaps, the next important work in the future for the platform is to change

to coarse mass balance system into a fine mass balance by implementing a set of motors on the current coarse masses. The reason for this work is that the coarse masses have more mass that then fine masses, roughly seven times higher. When the coarse masses can be precisely controlled by the motor, the initially balancing effort for the platform will also become easier than hand-balancing. In addition, the extra mass will further reduce the center of mass offset to below one millimeter.

The next step is to set up an autonomous mass balance experiment through an iterative process. Future student can easily include a new component on the platform and run an autonomous program to reduce the center of mass offset within the allowable envelope of the fine mass balance. This process, however, requires a very stable wi-fi connection between the host and the target computers. The next improvement on the platform is to implement an absolute navigation sensor to independently verify the LN-200 data. Currently, the FSFB test cannot be cross-referenced for absolute accuracy whether it has settled exactly at 2% or better, or worse. An optical sensor system such as star tracker or a circular ring of L.E.D with webcam could be used for this purpose. The later would involve using Gram-Schmidt orthogonalization and least square regressions method - described in [26]- with some modification.

Although it is not a pressing need at the time of developing this thesis, future students could include motor capable of higher torque than the current motors in experiment. This allows the platform to freely operate around the roll and pitch axes since it is currently limited. Other higher torque application such as control moment gyros, CMG, could also be used, although this might be difficult to manufacture and integrate on the current platform. The last recommendation is to actually measure the inertia of the platform. A shake table could easily achieve

this goal, or using hanging technique such as those in aircraft manufacturing. A independently verified inertia would finally provide a good - or 'true' in sense of the best data - inertia for the perfect inertia FSFB test case when verifying with other control laws.

Bibliography

[1] Kim, Jae-Jun., and Agrawal, Brij N. *System Identification and Automatic Mass Balancing of Ground-Based Three-Axis Spacecraft Simulator*. AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado, Aug. 2006.

[2] Schwartz, J., and Hall, C. *System Identification of a Spherical Air-bearing Spacecraft Simulator*. AAS Paper 04-122, 8-12 Feb. 2004.

[3] Wie, Bong. *Space Vehicle Dynamics and Control*. 2nd ed., Reston, VA, AIAA, 2008.

[4] Bergmann, E. V., and Dzielski, J. *Spacecraft Mass Property Identification with Torque-Generating Control*. Journal of Guidance, Control, and Dynamics, Vol. 13, No. 1, 1990, pp. 99-103.

[5] Ma, O. , Dang, H., and Pham, K. *On-Orbit Identification of Inertia Properties of Spacecraft Using a Robotic Arm*. Journal of Guidance, Control, and Dynamics, Vol. 31, No. 6, 2008, pp. 1761-1771.

[6] Tanygin, S., and Williams, T., *Mass Property Estimation Using Coasting Maneuvering*. Journal of Guidance, Control, and Dynamics, Vol. 20, No. 4, 1997, pp. 625632

[7] Wilson, E., Lages, C., and Mah, R., *On-Line, Gyro-Based, Mass-Property Identification for Thruster-Controlled Spacecraft Using Recursive Least Squares*. 45th Midwest Symposium on Circuits and Systems, Vol. 2, Inst. of Electrical and Electronics Engineers, New York, Aug. 2002, pp. 334337

[8] Cooper, J. E., and Wright, J. R., *Spacecraft In-orbit Identification Using Eigensystem Realization Methods*. Journal of Guidance, Control, and Dynamics, Vol. 13, No. 1, 1990, pp. 99-103.

[9] Juang, Jer-Nan. *Applied System Identification*. Eaglewood Cliffs, NJ: Prentice-Hall, 1987.

[10] Juang, J., and Phan, M., *Identification of System, Observer, and Controller from Closed-Loop*. Journal of Guidance, Control, and Dynamics, Vol. 17, No. 1, 1994, pp. 91-96.

[11] Majii, M., Juang, J., and Jukins, J. L., *Observer/Kalman-Filter Time-Varying System Identification*. Journal of Guidance, Control, and Dynamics, Vol. 33, No. 3, May-June 2010.

[12] Su, T., and Juang, J., *Structure System Identification and Synthesis*. Journal of Guidance, Control, and Dynamics, Vol. 17, No. 5, 1994, pp. 1087-1095.

[13] Juang, J., and Pappa, R. S., *Effects of Noise on Modal Parameters Identified by the Eigensystem Realization Algorithm*. Journal of Guidance, Control, and Dynamics, Vol. 9, No. 3, 1986, pp. 294-303.

[14] Allemang, R. J., and Brown, D. L., *A Unified Matrix Polynomial Approach to Model Identification*. Journal of Sound and Vibration, Vol. 211, No. 3, 1998, pp. 301-322.

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

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

[17] Mathworks^{TM}. *xPC Target Getting Started*. Rev. 5.3, September, 2008.

[18] Huang, J., Lee, H. C., Schoen, M. P., and Hsiao, M. H., *State-Space System Identification from Closed-Loop Frequency Response Data*. Journal of Guidance, Control, and Dynamics, Vol. 19, No. 6, Nov-Dec 1996.

[19] Lee, J. H., and Kim, J., *Identification of Damping Matrices From Measured Frequency Response Functions*. Journal of Sound and Vibration, Vol. 240, No. 3, 2001, pp. 545-565.

[20] Miller, D. N., de Callafon, R. A., and Brennet, M. J., *Covariance-Based Realization Algorithm for the Identification of Aeroelastic Dynamics*. Journal of Guidance, Control, and Dynamics, Vol. 35, No. 4, July-August 2012.

[21] Grauer, J., and Morelli, E., *Method for Real-Time Frequency Response and Uncertainty Estimation*. Journal of Guidance, Control, and Dynamics, Vol. 37, No. 1, January-February 2014.

[22] Schwartz, J.L., Peck, M.A., and Hall, C.D. *Historical Survey of Air-Bearing Spacecraft Simulators*. Journal of Guidance, Control, and Dynamics, vol. 26, pp. 513522, July-August 2003.

[23] Mittlesteadt, Carson, *Results on the Development of a Four-Wheel Pyramidal Reaction Wheel Platform*. Master's Thesis, Dept. Aerospace Engineering, Cal Poly State University, San Luis Obispo, CA, 2006.

[24] Sidi, Marcel. *Spacecraft Dynamics and Control: A Practical Engineering Approach*. New York, NY: Cambridge University Press, 2000.

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

[26] Cowen, Carl. *A Project on Circles in Space*. Purdue University, 1995.

Appendix A

Additional Result for Mass Properties Identification

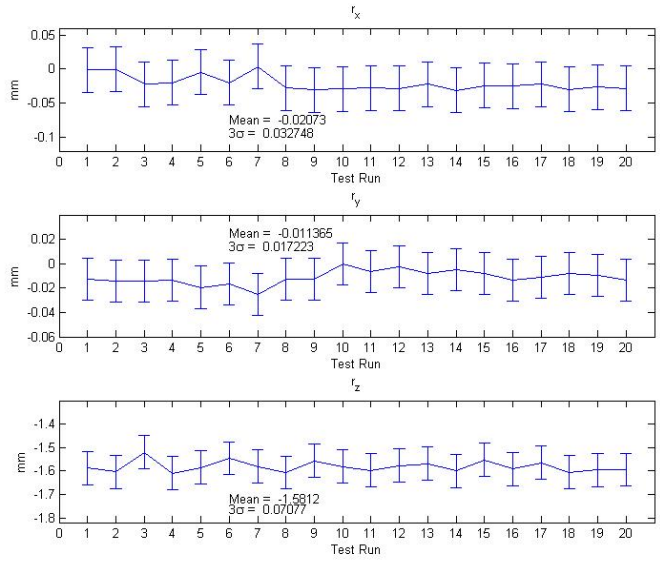


Figure A.1: System I.D results for center of mass.

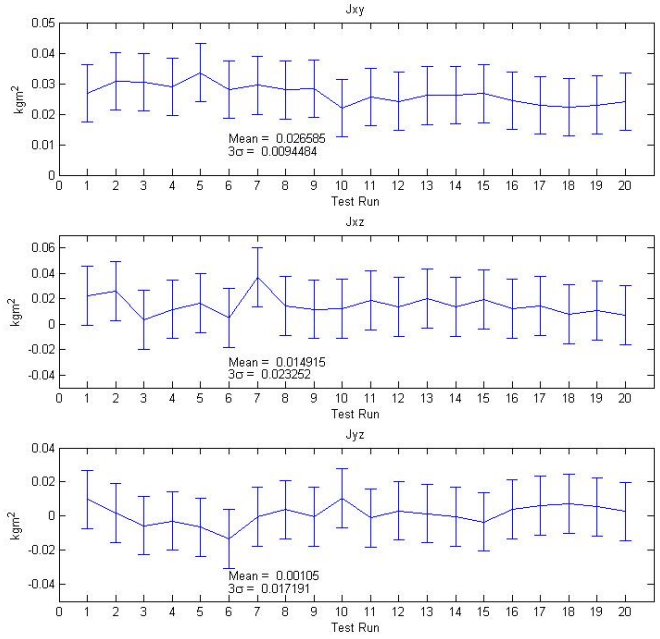


Figure A.2: System I.D results for off-diagonal terms.

Appendix B

Equations

B.1 Rigid Body Kinematics

$$\vec{R}_B = \vec{R}_A + \vec{R}_{B/A} \tag{B.1}$$

$$\begin{aligned} \tfrac{d}{dt}\vec{R}_B &= \tfrac{d}{dt}\vec{R}_A + \tfrac{d}{dt}\vec{R}_{B/A} \\ \dot{\vec{R}}_B &= \dot{\vec{R}}_A + \vec{\omega} \times \vec{R}_{B/A} \end{aligned} \tag{B.2}$$

$$\begin{aligned} \ddot{\vec{R}}_B &= \ddot{\vec{R}}_A + \tfrac{d}{dt}(\vec{\omega} \times \vec{R}_{B/A}) \\ \ddot{\vec{R}}_B &= \ddot{\vec{R}}_A + \tfrac{d}{dt}\vec{\omega} \times \vec{R}_{B/A} + \vec{\omega} \times \tfrac{d}{dt}\vec{R}_{B/A} \\ \ddot{\vec{R}}_B &= \ddot{\vec{R}}_A + \alpha \times \vec{R}_{B/A} + \vec{\omega} \times \vec{\omega} \times \vec{R}_{B/A} \end{aligned} \tag{B.3}$$

B.2 Rigid Body Dynamics

$$d\vec{M}_p = \vec{r} \times (d\vec{F}_{net} + d\vec{f}_{net}) \tag{B.4}$$

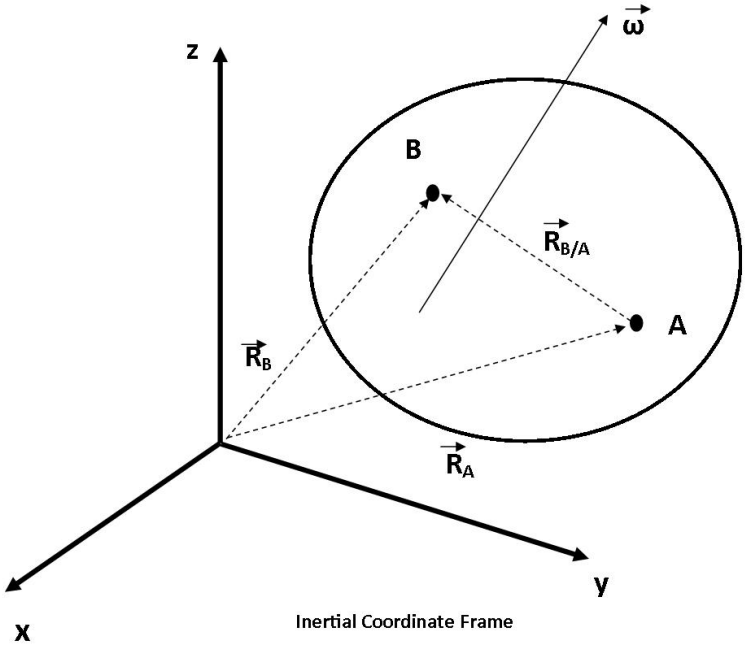


Figure B.1: Rigid Body Kinematics Schematic

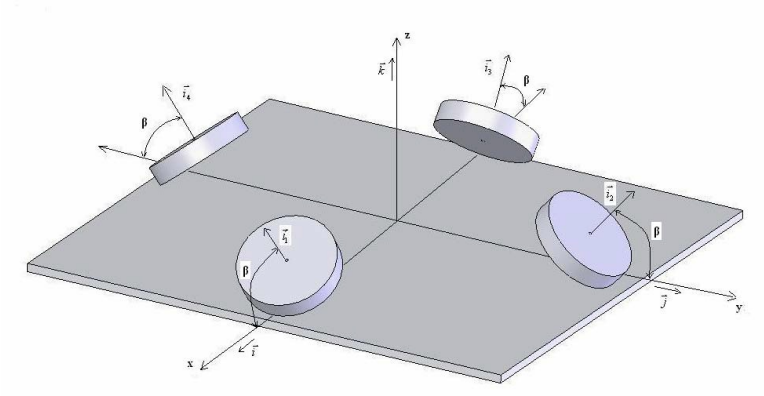


Figure B.2: Reaction Wheel Reference Frame Schematic

$$\begin{aligned}
\sum \vec{F} &= m\vec{a} \\
(d\vec{F}_{net} + d\vec{f}_{net}) &= dm\ddot{\vec{R}} \\
d\vec{M}_p &= \vec{r} \times dm\ddot{\vec{R}} \\
\vec{M}_p &= \int (\vec{r} \times \ddot{\vec{R}}) dm
\end{aligned} \tag{B.5}$$

Note:

$$\begin{aligned}
\tfrac{d}{dt}\vec{r} \times \ddot{\vec{R}} &= \dot{\vec{r}} \times \dot{\vec{R}} + \vec{r} \times \ddot{\vec{R}} \\
\vec{r} \times \ddot{\vec{R}} &= \tfrac{d}{dt}(\vec{r} \times \dot{\vec{R}}) - \dot{\vec{r}} \times \dot{\vec{R}} \\
\vec{M}_p &= \int [\tfrac{d}{dt}(\vec{r} \times \dot{\vec{R}}) + \dot{\vec{R}}_p \times \dot{\vec{R}}] dm \\
\vec{M}_p &= \tfrac{d}{dt} \int (\vec{r} \times \dot{\vec{R}}) dm + \dot{\vec{R}}_p \times \int \dot{\vec{R}} dm
\end{aligned} \tag{B.6}$$

$$\begin{aligned}
\vec{M}_p &= \dot{\vec{H}}_p + \dot{\vec{R}}_p \times m\dot{\vec{R}}_G \\
\vec{M}_p &= \dot{\vec{H}}_p + \vec{V}_p \times m\dot{\vec{R}}_G \\
\vec{M}_p &= \dot{\vec{H}}_p
\end{aligned} \tag{B.7}$$

B.3 Wheel Coordinate Transformation and Euler Matrix

$$
\begin{aligned}
R_{W1} &= \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \\
R_{W2} &= \begin{bmatrix} 0 & 1 & 0 \\ \cos\beta & 0 & -\sin\beta \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \\
R_{W3} &= \begin{bmatrix} -\cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \\
R_{W4} &= \begin{bmatrix} 0 & 1 & 0 \\ -\cos\beta & 0 & \sin\beta \\ \sin\beta & 0 & \cos\beta \end{bmatrix}
\end{aligned}
\tag{B.8}
$$

$$
\begin{aligned}
I^{Wi}\vec{\omega}_{Wi} &= \begin{bmatrix} I_{11}^{Wi} & 0 & 0 \\ 0 & I_{22}^{Wi} & 0 \\ 0 & 0 & I_{33}^{Wi} \end{bmatrix} \begin{bmatrix} \omega_{Wi} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} I_{11}^{Wi}.\omega_{Wi} \\ 0 \\ 0 \end{bmatrix} \\
I^{Wi}\dot{\vec{\omega}}_{Wi} &= \begin{bmatrix} I_{11}^{Wi} & 0 & 0 \\ 0 & I_{22}^{Wi} & 0 \\ 0 & 0 & I_{33}^{Wi} \end{bmatrix} \begin{bmatrix} \dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} I_{11}^{Wi}.\dot{\omega}_{Wi} \\ 0 \\ 0 \end{bmatrix}
\end{aligned}
\tag{B.9}
$$

$$R_E = \begin{bmatrix} \cos\theta_2 \cos\theta_3 & \cos\theta_2 \sin\theta_3 & -\sin\theta_2 \\ \sin\theta_1 \sin\theta_2 \cos\theta_3 - \cos\theta_1 \sin\theta_3 & \sin\theta_1 \sin\theta_2 \sin\theta_3 + \cos\theta_2 \cos\theta_3 & \sin\theta_1 \cos\theta_2 \\ \cos\theta_1 \sin\theta_2 \cos\theta_3 + \sin\theta_1 \sin\theta_3 & \cos\theta_1 \sin\theta_2 \sin\theta_3 - \sin\theta_1 \cos\theta_3 & \cos\theta_1 \cos\theta_2 \end{bmatrix} \tag{B.10}$$

Appendix C

System Schematics

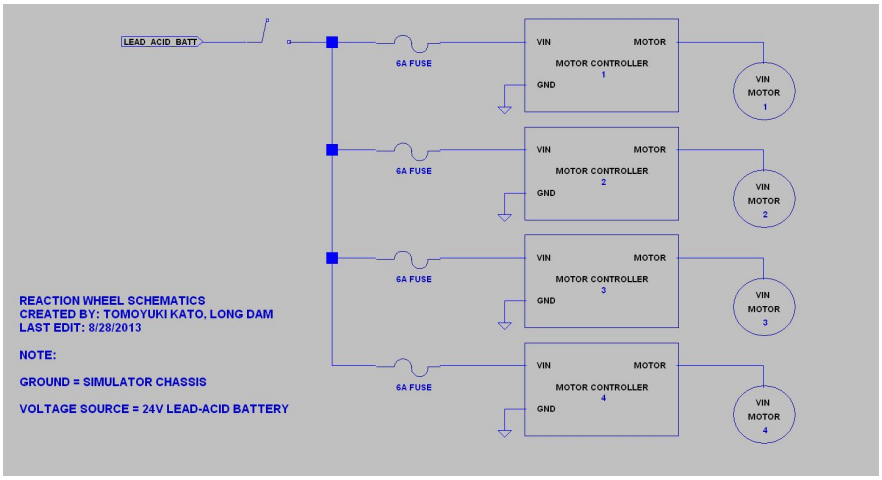


Figure C.1: Reaction Wheel Circuit Schematic

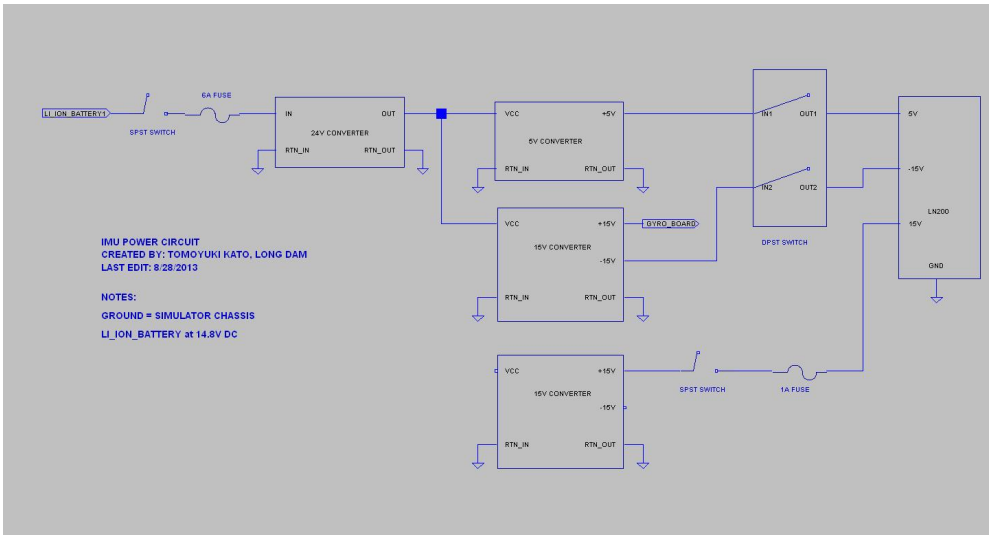


Figure C.2: IMU Circuit Schematic

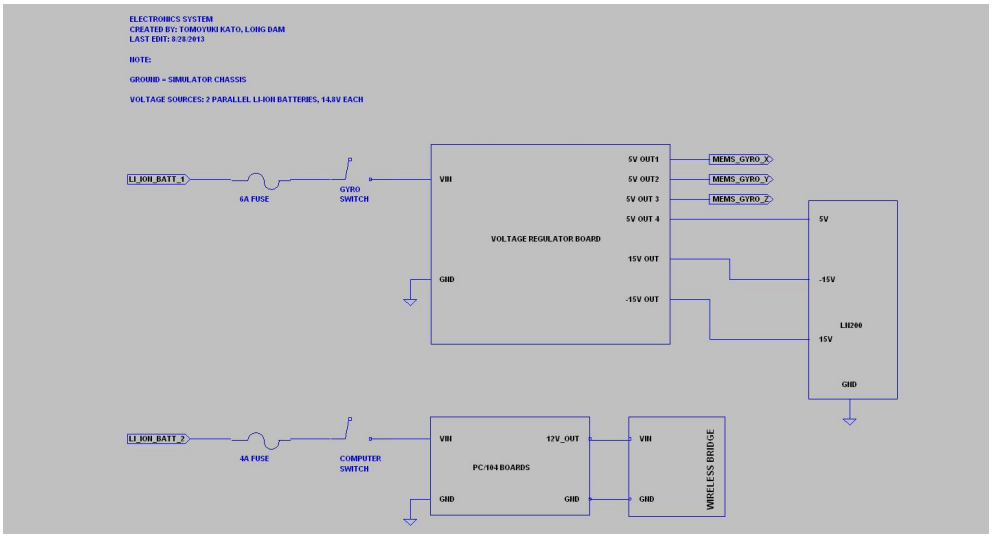


Figure C.3: Electronics Circuit Schematic

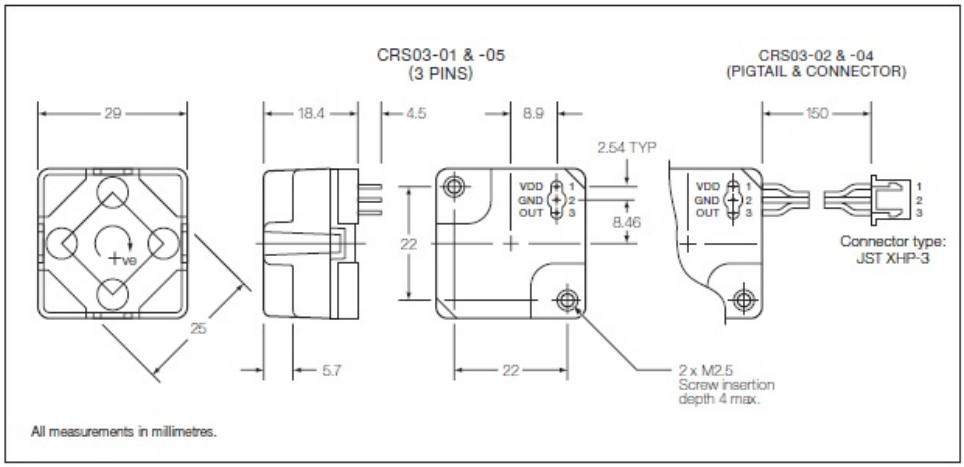


Figure C.4: MEMS Gyroscope Circuit Schematic

Appendix D

Code and Simulation Block

D.1 Source Codes

```
/*
Filename: xpcfastcom104hook.c
Created by: Tomoyuki Kato, Long Dam
Last edit: 4/17/2013

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(("%u\n"),11111); // 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

/*
Filename: fastcomm104setup.c
Created by: Tomoyuki Kato, Long Dam
Last edit: 5/1/2013

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)
                (0x90)); // CCR1
            xpcOutpB((uint16_T)(base + CCR0),(uint8_T)
                (0x00)); // CCR0 (Setting HDLC mode)
            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 + 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 + 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 + 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

/*
Filename: fastcom104read.c
Created by: Tomoyuki Kato, Long Dam
Last edit: 5/1/2013

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, 1) )return;
    ssSetOutputPortWidth(S, 0, 33);
    ssSetOutputPortDataType(S, 0, 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 count                   = 0;

                  // Counter
            int c;

                                                // Temp. var
               .
            int i;
            int j;


                 while (count < 32) // start FIFO read (32
                    bytes deep + count)
```

```
                {
                                count++;
                                c = xpcInpB((uint16_T)(sab
                                    + DATA));
                                OPtr[count] = c;
                }

        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

function [x,rx,ry,rz] = systemIDalgorithm(BodyRates,
    BodyRatesDot,WheelSpeeds,WheelSpeedsDot,Thetas,time,Iw,
    m,beta)
% This function takes the body rates and wheel rates
    generated by the simulation, breaks
% them apart into usable arrays, calculates the A and T
    matrices, and
% solves for the inertia matrix and center of gravity
    location.
% time is a tx1 array of the simulation time.
% BodyRates is a tx3 array of the body rates [wx,wy,wz]
% WheelSpeeds is a tx4 array of the wheel speeds [wW1,wW2,
    wW3,wW4]
% m is the total platform mass in kg
% beta is the inclination angle of the reaction wheels in
    degrees
```

```
% Get the x,y,z, rotation rates and accels from the input
omegax = BodyRates(:,1);
omegay = BodyRates(:,2);
omegaz = BodyRates(:,3);
omegaxdot = BodyRatesDot(:,1);
omegaydot = BodyRatesDot(:,2);
omegazdot = BodyRatesDot(:,3);

% Get the wheel rates from the structure. Use the Kalman
    Filtered Rates
omegaW1 = WheelSpeeds(:,1);
omegaW2 = WheelSpeeds(:,2);
omegaW3 = WheelSpeeds(:,3);
omegaW4 = WheelSpeeds(:,4);
omegaW1dot = WheelSpeedsDot(:,1);
omegaW2dot = WheelSpeedsDot(:,2);
omegaW3dot = WheelSpeedsDot(:,3);
omegaW4dot = WheelSpeedsDot(:,4);
% Get the x,y,z angular accelerations and wheel
    accelerations by
% differentiating.
%{
for i = 1:length(time)-1
        omegaxdot(i) = (omegax(i+1)-omegax(i))/(time(i+1)-
            time(i));
```

```
        omegaydot(i) = (omegay(i+1)-omegay(i))/(time(i+1)-
            time(i));
        omegazdot(i) = (omegaz(i+1)-omegaz(i))/(time(i+1)-
            time(i));
        omegaW1dot(i) = (omegaW1(i+1)-omegaW1(i))/(time(i
            +1)-time(i));
        omegaW2dot(i) = (omegaW2(i+1)-omegaW2(i))/(time(i
            +1)-time(i));
        omegaW3dot(i) = (omegaW3(i+1)-omegaW3(i))/(time(i
            +1)-time(i));
        omegaW4dot(i) = (omegaW4(i+1)-omegaW4(i))/(time(i
            +1)-time(i));
end
%}
% Get theta values by integrating the body rates. Theta1
    is a rotation about the x
% axis, theta2 is a rotation about the y axis.
%{
Thetas = [0;0;0];
for i=1:length(time)-1
ThetasDot(:,i) = 1/cos(Thetas(2,i))*[cos(Thetas(2,i)),sin(
    Thetas(1,i))*sin(Thetas(2,i)),cos(Thetas(1,i))*sin(
    Thetas(2,i));0,cos(Thetas(1,i))*cos(Thetas(2,i)),-sin(
    Thetas(1,i))*cos(Thetas(2,i));0,sin(Thetas(1,i)),cos(
    Thetas(1,i))]*[omegax(i);omegay(i);omegaz(i)];
Thetas(:,i+1) = Thetas(:,i)+ThetasDot(:,i)*(time(i+1)-time
```

```
    (i));
end
%}
theta1 = Thetas(:,1);
theta2 = Thetas(:,2);

% Now, calculate the A and T matrices at each time step.

A = Agenerator(theta1(1),theta2(1),omegax(1),omegay(1),
    omegaz(1),omegaxdot(1),omegaydot(1),omegazdot(1));
T = Tgenerator(beta,Iw,omegax(1),omegay(1),omegaz(1),
    omegaW1(1),omegaW2(1),omegaW3(1),omegaW4(1),omegaW1dot
    (1),omegaW2dot(1),omegaW3dot(1),omegaW4dot(1));
for i = 2:length(time)-1
A = [A;Agenerator(theta1(i),theta2(i),omegax(i),omegay(i),
    omegaz(i),omegaxdot(i),omegaydot(i),omegazdot(i))];
T = [T;Tgenerator(beta,Iw,omegax(i),omegay(i),omegaz(i),
    omegaW1(i),omegaW2(i),omegaW3(i),omegaW4(i),omegaW1dot(
    i),omegaW2dot(i),omegaW3dot(i),omegaW4dot(i))];
end
% Use the psuedo-inverse to calculate x = [Ixx,Ixy,Ixz,Iyy
    ,Iyz,Izz,mgrx,
% mgry, mgrz]'

x = (((A'*A)^-1)*A')*T;
```

```
g = 9.81; % m/s^2
rx = x(7,:)./(m*g);
ry = x(8,:)./(m*g);
rz = x(9,:)./(m*g);
end

function T = Tgenerator(beta,Iw,omegax,omegay,omegaz,
    omega1,omega2,omega3,omega4,omega1dot,omega2dot,
    omega3dot,omega4dot)
% This function generates the T matrix for a given time
    step.
Sbeta = sin(beta);
Cbeta = cos(beta);
T = -[Cbeta, 0, -Cbeta, 0; 0, Cbeta, 0, -Cbeta; Sbeta,
    Sbeta, Sbeta, Sbeta]*[Iw*omega1dot;Iw*omega2dot;Iw*
    omega3dot;Iw*omega4dot] - [omegay*Sbeta, (omegay*Sbeta-
    omegaz*Cbeta), omegay*Sbeta, (omegay*Sbeta+omegaz*Cbeta
    ); (omegaz*Cbeta-omegax*Sbeta), -omegax*Sbeta, -(omegax
    *Sbeta+omegaz*Cbeta), -omegax*Sbeta; -omegay*Cbeta,
    omegax*Cbeta, omegay*Cbeta, -omegax*Cbeta]*[Iw*omega1;
    Iw*omega2;Iw*omega3;Iw*omega4];
end

function A = Agenerator(theta1,theta2,omegax,omegay,omegaz
    ,omegaxdot,omegaydot,omegazdot)
% This function generates the A matrix for a given time
```

```
    step.

A = [omegaxdot, (omegaydot-omegax*omegaz),(omegazdot+
    omegax*omegay), -omegay*omegaz, (omegay^2-omegaz^2),
    omegay*omegaz, 0, -cos(theta1)*cos(theta2), sin(theta1)
    *cos(theta2);
          omegax*omegaz, (omegaxdot+omegay*omegaz), (omegaz
              ^2-omegax^2), omegaydot, (omegazdot-omegax*
              omegay), -omegax*omegaz, cos(theta1)*cos(theta2
              ), 0, sin(theta2);
         -omegax*omegay, (omegax^2-omegay^2), (omegaxdot-
              omegay*omegaz), omegax*omegay, (omegaydot+
              omegax*omegaz), omegazdot, -sin(theta1)*cos(
              theta2), -sin(theta2), 0];
end

% SYS ID Script
% Written by: Long Dam
% Last Edited: 8/7/13
% This file loads a simulation data file and plots the
    commanded wheel
% speed and body rates from the LN-200. The file then
    executes the system
% ID algorithm with butterworth filter.
clear all; close all; clc;
load('SINEWAVE_20130610.mat');
m = 68/2.2;
```

```
Iw = .00053552; % Wheel inertia, kg-m^2
Beta = 28.3*pi/180; % Wheel inclination angle, rad
fsfb = out;

time = t;
whl          = fsfb(:,1:4);
whlcmd       = fsfb(:,5:8);
quat         = fsfb(:,9:12);
body         = fsfb(:,13:15);
E            = fsfb(:,16:18);

figure(1)
subplot(2,1,1)
plot(time,whlcmd)
xlabel('time (s)')
ylabel('(rad/s)')
legend('W1','W2','W3','W4')
title('Commanded Wheel Speed')
grid on
subplot(2,1,2)
plot(time,whl)
xlabel('time (s)')
ylabel('(rad/s)')
legend('W1','W2','W3','W4')
title('Wheel Speed')
grid on
```

```
figure(2)
plot(time,body)
title('Body rates')
xlabel('time (s)')
ylabel('(rad/s)')
legend('X','Y','Z')
grid on
% subplot(3,1,3)
% plot(time,E*180/pi)
% xlabel('time (s)')
% ylabel('(deg)')
% legend('Roll','Pitch','Yaw')
%%

[B,A]=butter(8,1/25,'low');
bxfilt = filtfilt(B,A,body(:,1));
byfilt = filtfilt(B,A,body(:,2));
bzfilt = filtfilt(B,A,body(:,3));
W1filt = filtfilt(B,A,whl(:,1));
W2filt = filtfilt(B,A,whl(:,2));
W3filt = filtfilt(B,A,whl(:,3));
W4filt = filtfilt(B,A,whl(:,4));

for i = 1:length(time)-1
```

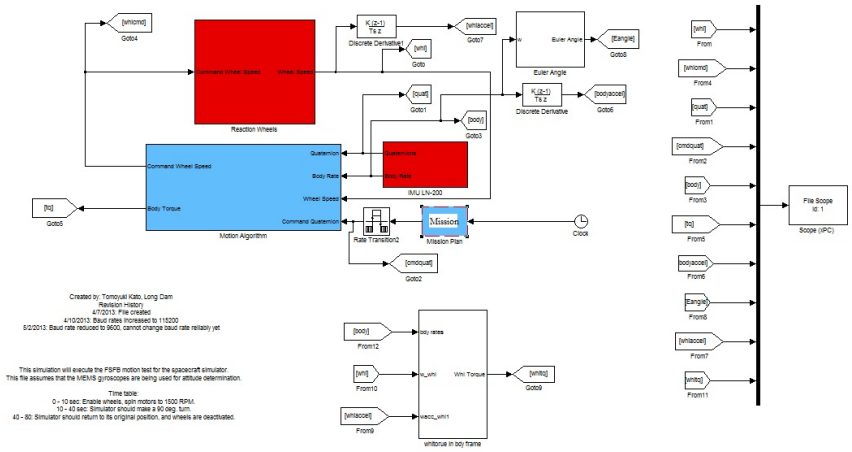
```
        bxfiltdot(i,1) = (bxfilt(i+1)-bxfilt(i))/(time(i
            +1)-time(i));
        byfiltdot(i,1) = (byfilt(i+1)-byfilt(i))/(time(i
            +1)-time(i));
        bzfiltdot(i,1) = (bzfilt(i+1)-bzfilt(i))/(time(i
            +1)-time(i));
        W1filtdot(i,1) = (W1filt(i+1)-W1filt(i))/(time(i
            +1)-time(i));
        W2filtdot(i,1) = (W2filt(i+1)-W2filt(i))/(time(i
            +1)-time(i));
        W3filtdot(i,1) = (W3filt(i+1)-W3filt(i))/(time(i
            +1)-time(i));
        W4filtdot(i,1) = (W4filt(i+1)-W4filt(i))/(time(i
            +1)-time(i));
    end
    bxfiltdot(length(time),1) = bxfiltdot(end,1);
    byfiltdot(length(time),1) = byfiltdot(end,1);
    bzfiltdot(length(time),1) = bzfiltdot(end,1);
    W1filtdot(length(time),1) = W1filtdot(end,1);
    W2filtdot(length(time),1) = W2filtdot(end,1);
    W3filtdot(length(time),1) = W3filtdot(end,1);
    W4filtdot(length(time),1) = W4filtdot(end,1);
    % Calculate Euler Angles
    Thetas = [0,0,0];
    for i=1:length(time)-1
    ThetasDot(:,i) = 1/cos(Thetas(i,2))*[cos(Thetas(i,2)),sin(
```

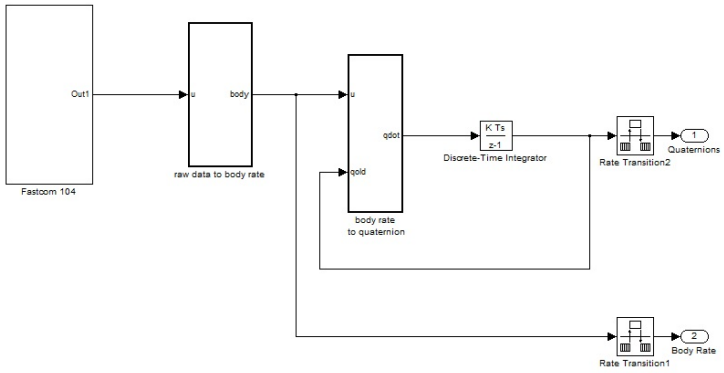
```
    Thetas(i,1))*sin(Thetas(i,2)),cos(Thetas(i,1))*sin(
    Thetas(i,2));0,cos(Thetas(i,1))*cos(Thetas(i,2)),-sin(
    Thetas(i,1))*cos(Thetas(i,1));0,sin(Thetas(i,1)),cos(
    Thetas(i,1))]*[bxfilt(i);byfilt(i);bzfilt(i)];
Thetas(i+1,:) = Thetas(i,:)+ThetasDot(:,i)'*(time(i+1)-
    time(i));
end
% dx = zeros(length(whl),4);
% for i =1:4
%     dx(2:end,i) = whl(2:end,i)-whl(1:end-1,i);
% end
% dt = 1/50;
% whlacc = dx/dt;

%%
st = length(time);
time1 = time(1:st);
WheelSpeeds1 = [W1filt(1:st),W2filt(1:st),W3filt(1:st),
    W4filt(1:st)];
WheelSpeedsDot1 = [W1filtdot(1:st),W2filtdot(1:st),
    W3filtdot(1:st),W4filtdot(1:st)];
BodyRates1 = [bxfilt(1:st),byfilt(1:st),bzfilt(1:st)];
BodyRatesDot1 = [bxfiltdot(1:st),byfiltdot(1:st),bzfiltdot
    (1:st)];
Thetas1 = Thetas(1:st,:);
[x,rx,ry,rz] = systemIDalgorithm(BodyRates1,BodyRatesDot1,
```

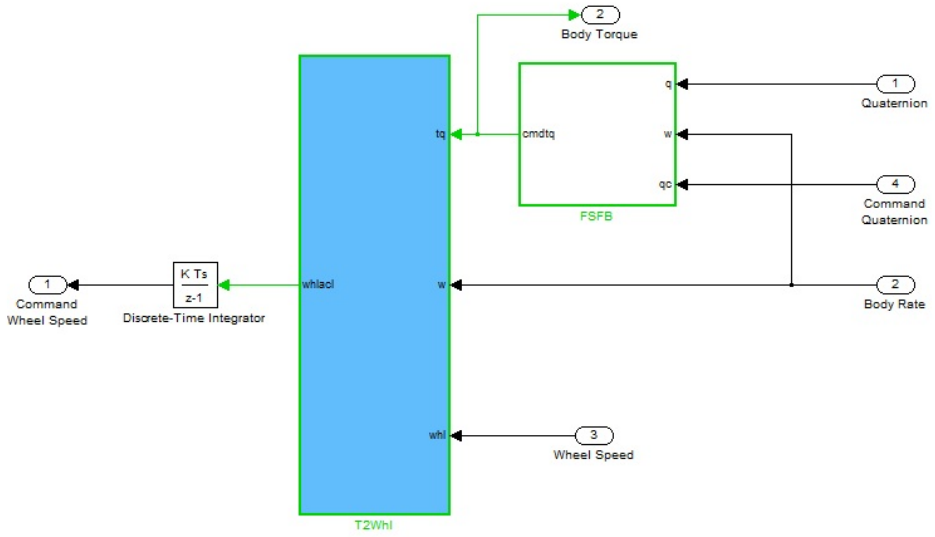
```
    WheelSpeeds1,WheelSpeedsDot1,Thetas1,time1,Iw,m,Beta);

x
r = -[rx; ry; rz]*1000
%%
% Beta = 57;
% Iw = 5.3552e-4;
% A = zeros(length(t)*3,9);
% Tbar = zeros(length(t)*3,1);
RR = [cos(Beta)  0          -cos(Beta)   0;
      0          cos(Beta)    0          -cos(Beta);
      sin(Beta)  sin(Beta)    sin(Beta)   sin(Beta)];
% Torque = Tbar;
% TW = zeros(length(t),3);
wxdot = BodyRatesDot1(:,1);
wydot = BodyRatesDot1(:,2);
wzdot = BodyRatesDot1(:,3);

wx = BodyRates1(:,1);
wy = BodyRates1(:,2);
wz = BodyRates1(:,3);

wheel =WheelSpeeds1;
whlacc = WheelSpeedsDot1;

E1 = Thetas(:,1);
```

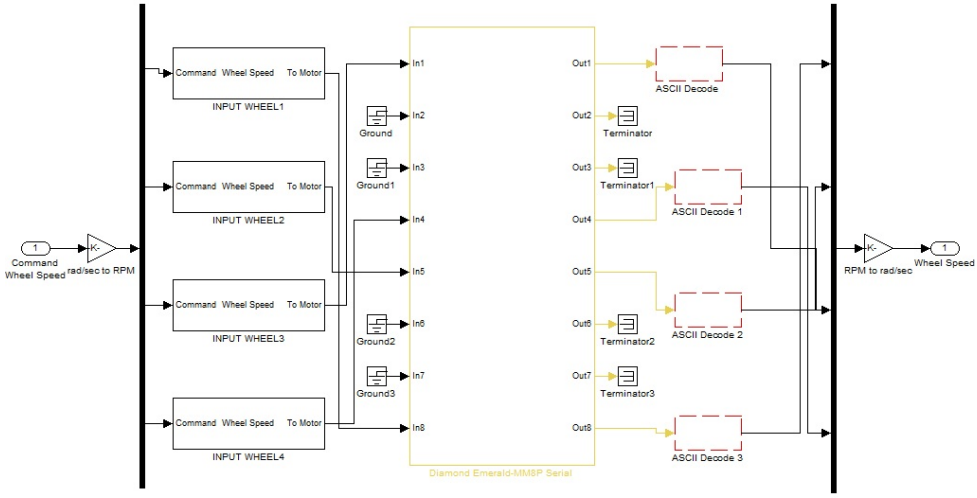
```
E2 = Thetas(:,2);
for i = 1:length(t)-1
    A((3*i-2):(3*i),:) = [wxdot(i) (wydot(i)-wx(i)*wz(i))
        (wzdot(i)+wx(i)*wy(i)) -wy(i)*wz(i) (wy(i)^2-wz(i)
        ^2) wy(i)*wz(i) 0 -cos(E1(i))*cos(E2(i)) sin(E1(i))
        *cos(E2(i));
      wx(i)*wz(i) (wxdot(i)+wy(i)*wz(i)) (wz(i)^2-wx(i)^2)
        wydot(i) (wzdot(i)-wx(i)*wy(i)) -wx(i)*wz(i) cos(
        E1(i))*cos(E2(i)) 0 sin(E2(i));
-wx(i)*wy(i) (wx(i)^2-wy(i)^2) (wxdot(i)-wy(i)*wz(i)) wx(i
    )*wy(i) (wydot(i)+wx(i)*wz(i)) wzdot(i) -sin(E1(i))*cos
    (E2(i)) -sin(E2(i)) 0];

RM = [wy(i)*sin(Beta) (wy(i)*sin(Beta)-wz(i)*cos(Beta))
      wy(i)*sin(Beta) (wy(i)*sin(Beta)+wz(i)*cos(Beta));
     (wz(i)*cos(Beta)-wx(i)*sin(Beta)) -wx(i)*sin(Beta)
        -(wx(i)*sin(Beta)+wz(i)*cos(Beta)) -wx(i)*sin(Beta
        );
    -wy(i)*cos(Beta) wx(i)*cos(Beta) wy(i)
        *cos(Beta) -wx(i)*cos(Beta)];
Torque(3*i-2:3*i,1) = -(RR*Iw*whlacc(i,:)'+RM*Iw*wheel(i
    ,:)');
end
x1 = ((A'*A)\A')*Torque
r1 = x(7:9)/-9.81/m*1000
```

D.2 Simulation Blocks

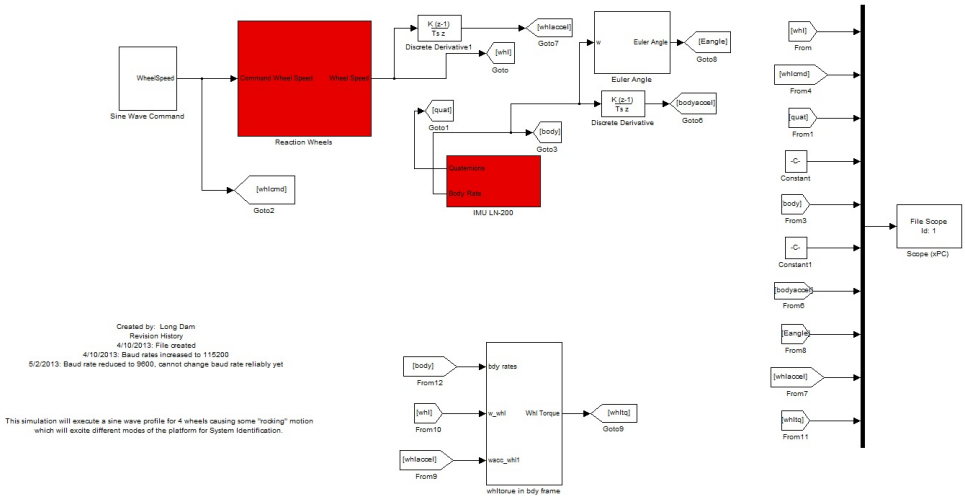








The order of the incoming signal assumes this order: +X, +Y, -X, and -Y for the wheels.
lines above are connected such that the incoming signals match the physical connections
to the reaction wheels.


WheelSpeed
Sine Wave Command
Command Wheel Speed
Wheel Speed
Reaction Wheels
[whlcmd]
Goto2
K (z-1) Ts z
Discrete Derivative1
[whl]
Goto
[whlacel]
Goto7
[quat]
Goto1
[body]
Goto3
Quaternions
Body Rate
IMU LN-200
w
Euler Angle
Euler Angle
[Eangle]
Goto8
K (z-1) Ts z
Discrete Derivative
[bodyacel]
Goto6
Created by: Long Dam
Revision History
4/10/2013: File created
4/10/2013: Baud rates increased to 115200
5/2/2013: Baud rate reduced to 9600, cannot change baud rate reliably yet
This simulation will execute a sine wave profile for 4 wheels causing some "rocking" motion which will excite different modes of the platform for System Identification.
[body]
From12
[whl]
From10
[whlacel]
From9
bdy rates
w_whl
Whl Torque
vacc_whl1
whltorue in bdy frame
[whltq]
Goto9
[whl]
From
[whlcmd]
From4
[quat]
From1
Constant
[body]
From3
Constant1
[bodyacel]
From6
[Eangle]
From8
[whlacel]
From7
[whltq]
From11
File Scope
Id: 1
Scope (xPC)