Evaluation of Assembly Simulators Used in Closed Loop Attitude Control System Testing

by

Jason Christopher Bunn

S.B., Aeronautics and Astronautics
Massachusetts Institute of Technology, 1996

SUBMITTED TO THE DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Master of Science in Aeronautics and Astronautics
at the
Massachusetts Institute of Technology

June, 1997

Signature of Author: [illegible]
Department of Aeronautics and Astronautics
April 30,1997

Certified by: [illegible]
Keyur C. Patel
Validation Group Supervisor, Avionic Systems Engineering
Jet Propulsion Laboratory

Certified by: [illegible]
Steven R. Hall
Associate Professor of Aeronautics and Astronautics
Thesis Advisor

Accepted by: [illegible]
Jaime Peraire
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Office

Evaluation of Assembly Simulators Used in Closed Loop Attitude Control System Testing

by

Jason Christopher Bunn

Submitted to the Department of Aeronautics and Astronautics

on May 5,1997 in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Aeronautics and Astronautics

Abstract

The Cassini spacecraft's Attitude and Articulation Control Subsystem has been tested extensively at the Jet Propulsion Laboratory in Pasadena, California. Three of the subsystem's assemblies have been tested using assembly simulators in place of actual hardware. These simulators have been designed and tested to ensure as much commonality with the hardware as possible. Several early design choices, the most crucial of which concerned the interface between the assembly simulators and the flight hardware, have influenced the degree to which simulator performance emulated flight hardware performance. However, these difficulties were overcome and all testing requirements were satisfied.

The use of simulators has resulted in testing ability otherwise impossible due to the small number of flight components constructed. Through this experience, several key lessons were learned, chief among them being clear definition of expectations and the importance of defining simulation interfaces as nearly identical to the flight equipment interfaces as possible. Assembly simulators, properly developed, should prove a valuable alternative to physical hardware for testing future flight projects.

Thesis Advisor: Professor Steven R. Hall
Title: Associate Professor of Aeronautics and Astronautics

Company Supervisor: Keyur C. Patel
Title: Validation Group Supervisor, Avionic Systems Engineering, JPL

Acknowledgments

First, I would like to thank the Engineering Internship Program office at MIT for making the opportunity to conduct research at the Jet Propulsion Laboratory possible. Both William Ramsey and Laura Robinson have been a source of great support.

I would also like to thank the Co-Op office at the Jet Propulsion Laboratory for giving me the opportunity to participate in the testing of the Cassini spacecraft.

For their support and assistance, I would like to thank the Validation Group of the Avionic Systems Engineering Section at JPL. I can honestly say I am proud to be a former Cassini AACS dynamicist, even if it is one in a long line. I would like to especially thank Richard Haga, Mario Mora, and James Roberts of JPL, as well as Roy Okuno of Boeing North American and Andrew Engelmann of the University of Colorado.

Finally, I wish to thank my parents, Anthony and Patricia Bunn, for their constant support and love over the years.

Table of Contents

List of Figures

Acronym List

AACS	Attitude and Articulation Control Subsystem
ACC	Accelerometer
AFC	AACS Flight Computer
ALF	Accelerated Load Format
ATLO	Assembly, Test and Launch Operations
BAIL	Backdoor ALF Injection Loader
BIT	Built In Test
BPLVD	BiPropellant Latch Valve Driver
CATS	Cassini AACS Test Station
CDS	Command and Data Subsystem
DARTS	Dynamics Algorithms for Real Time Simulation
EFC	Engineering Flight Computer
EGA	Engine Gimbal Actuator
EGECU	Engine Gimbal Electronics Control Unit
EGED	Engine Gimbal Electronics Driver
GMT	Greenwich Mean Time
GUPI	Gyro Signal Processing Intergrated Circuit
FSDS	Flight Software Development System
HELVD	Helium Latch Valve Driver
HRG	Hemisperical Resonating Gyroscope
IRU	Inertial Reference Unit
ITL	Integration and Test Laboratory

JPL	Jet Propulsion Laboratory
LVDT	Linear Variable Differential Transformer
MEVD	Main Engine Valve Driver
MPD	MonoPropellant Driver
PCU	Power Conversion Unit
PIU	Pixel Input Unit
PMS	Propulsion Module Subsystem
PSM	Power Supply Module
POU	Pixel Output Unit
PPS	Power and Pyrotechnic Subsystem
RTIOU	Remote Terminal Input Output Unit
RWA	Reaction Wheel Assembly
RWM	Reaction Wheel Motor
RWX	Reaction Wheel Electronics
SEHW	Support Equipment Hardware
SESW	Support Equipment Software
SEM	Sensor Electronics Module
SHARC	Space HRG and Accelerometer Readout Controller
SSH	Sun Sensor Head
SSE	Sun Sensor Electronics
SSPS	Solid State Power Switch
SRU	Stellar Reference Unit
VDECU	Valve Drive Electronics Control Unit
VDEX	Valve Drive Electronics Extension

Chapter 1

Introduction

Hardware-in-the-loop testing of attitude control systems has become an important part of a spacecraft's development cycle. With flight testing unavailable, the ability to simulate in real time the operation of a control system is critical to understanding the behavior of the system and to debugging software prior to launch. The growing interest in the development of such closed-loop simulators is evidenced by the recent Nineteenth Annual AAS Rocky Mountain Guidance and Control Conference in Breckenridge, Colorado, 7-11 February 1996, in which a new session was added to the agenda, "Applications of Testbeds to Spacecraft Guidance and Control" [2].

Spacecraft attitude control system testing typically involves both software and hardware testbeds. Software testbeds use software models of the spacecraft to test attitude control, fault protection, navigation, and other algorithms to be used by the flight software. The hardware testbed is used to test both the electrical integration of the various components and the flight software performance in a closed-loop environment. For the Mars Pathfinder flight project, 80% of electrical interfaces and 85% of functional tests were performed at the Flight Software Testbed for Pathfinder (FST/P), which housed both hardware and software testbeds [1]. The NASA Submillimeter Wave Astronomy Satellite (SWAS) was also tested in two separate testbeds. A software testbed was used to test the flight algorithms; a hardware testbed was used to test the subsystem in a closed-loop environment [10].

The California Institute of Technology's Jet Propulsion Laboratory (JPL) is currently in the final stages of testing the Cassini spacecraft. In October, 1997, a Titan IV/Centaur launch vehicle will lift the Cassini spacecraft towards Saturn, beginning the last in a series of grand tour missions that have included the Voyager probes and the Galileo spacecraft. Cassini's mission is to deliver the Huygens Titan probe to the surface of Titan and to perform an orbital analysis of the Saturnian system. The spacecraft has been developed and tested primarily by JPL for NASA's Office of Space Science.

The attitude and articulation control subsystem (AACS) for Cassini has been extensively tested at JPL through a comprehensive closed-loop testing plan. As in the two previously mentioned flight projects, there are two types of testbeds for the AACS. The first, the Flight Software Development System (FSDS), is a software testbed that is used to test the flight software algorithms. The AACS is also tested through real-time, hardware-in-the-loop simulation. This subsystem has a number of inputs to allow the support equipment to simulate the outside environment of the spacecraft during the mission. Examples of these inputs include biases for the accelerometer and gyroscopes, simulated starfields, and simulated sun sensor inputs. Interfaces external to the AACS are also simulated. These include the Command and Data Subsystem (CDS), the Power and Pyrotechnic Subsystem (PPS), and the Propulsion Module Subsystem (PMS). The combination of support equipment inputs and external interface simulators allow the hardware to experience flightlike conditions (with the exception of environmental conditions) and make closed-loop testing of the hardware possible.

There is one critical difference between the previously mentioned projects and the Cassini mission to Saturn. In both the Mars Pathfinder mission and the SWAS mission, the hardware-in-the-loop testbeds consisted of the entire subsystem. The Galileo closed-loop simulation that is still active at JPL uses a complete hardware set as well. This allows engineers on the ground to use a complete subsystem to investigate on-orbit AACS anomalies. For Cassini, however, the hardware testbed does not include a complete set of AACS hardware. During the busiest testing period, three hardware testbeds of varying fidelity run simultaneously at JPL for AACS testing. For most of the assemblies that make up the subsystem, there exists sufficient flight hardware to accommodate these laboratories. But for three assemblies, this is not the case. Due to the high cost of these assemblies and the quicker development time of simulators, the reaction wheel assembly, the inertial reference unit, and the engine gimbal actuators are replaced by assembly simulators during closed-loop testing. These simulators, which are primarily software simulations, play a pivotal role in AACS testing. During pre-flight testing, the simulators must act sufficiently like the flight units to permit testing of software functionality and mission sequences. After launch, these simulators are even more important, as they become the only mechanism by which testing of sequences or anomalies can occur. The importance of investigating and documenting the fidelity of these simulators is the motivation for this thesis.

This thesis evaluates the effectiveness of these simulators during testing of the AACS at JPL. With all of the laboratories relying on these simulators after launch, it is very important to understand the consequences of this approach. As the trend to drive down development costs in space missions continues, there is reason to expect that future missions may have to rely on assembly simulators in an ever increasing capacity. Evaluation of these assemblies of the Cassini spacecraft is a step toward developing a knowledge base that could be used when considering different testing options in the future.

This thesis begins with a description of the closed-loop testbed at JPL and how the laboratory environment works. This is followed by an analysis of the testing of the engine gimbal actuators, the inertial reference unit, and the reaction wheel assembly. Finally, we conclude with an evaluation of the testbed as a whole as well as lessons learned during closed-loop simulation.

This study was conducted at the Jet Propulsion Laboratory in conjunction with the Engineering Internship Program. The simulators described were developed over a period of several years as part of the development of the Cassini spacecraft. In particular, the engine gimbal actuator simulator was developed by Rick Graves and Edward Kopf. The inertial reference unit simulator was developed by Leticia Montanez and John Mayer. The reaction wheel simulator was developed by the author. In addition, several engineers were responsible for the development of the dynamic models described herein. Those responsible include James Roberts, Karl Pendergast, Paul Enright, Roy Okuno, and the author. The author was also extensively involved with the validation of the dynamic models as well as substantial upgrades to them.

Chapter 2

Closed-loop Attitude Control Testing

2.1 AACS

The attitude and articulation control subsystem is responsible for attitude determination and control during all phases of the mission. The components of the subsystem, shown in Figure 2.1, are described briefly below.

- AACS Flight Computer (AFC). The flight computer is responsible for acting on commands from the command and data subsystem (CDS) concerning guidance, navigation and control. The system is dual-redundant, and interfaces with the AACS databus, the CDS databus, and the power and pyrotechnic subsystem (PPS). A direct access port is used to probe the AFC memory or load flight software. When the actual command subsystem is not present, the AFC receives commands from a software simulation of the command and data subsystem. The AFC also interfaces with the Stellar Reference Unit through a dedicated databus used to gather pixel information during star identification.
- Accelerometer (ACC). The accelerometer is used to determine changes in velocity along the spacecraft Z axis. The accelerometer is a non-redundant component that interfaces with the power and pyrotechnic subsystem and the AACS databus. The accelerometer has a direct access port that allows for simulation of Z axis acceleration during testing.
- Backdoor ALF Injection Loader (BAIL). This assembly is a fault protection device that is used to provide a level of redundancy in the event that the command and data sub-

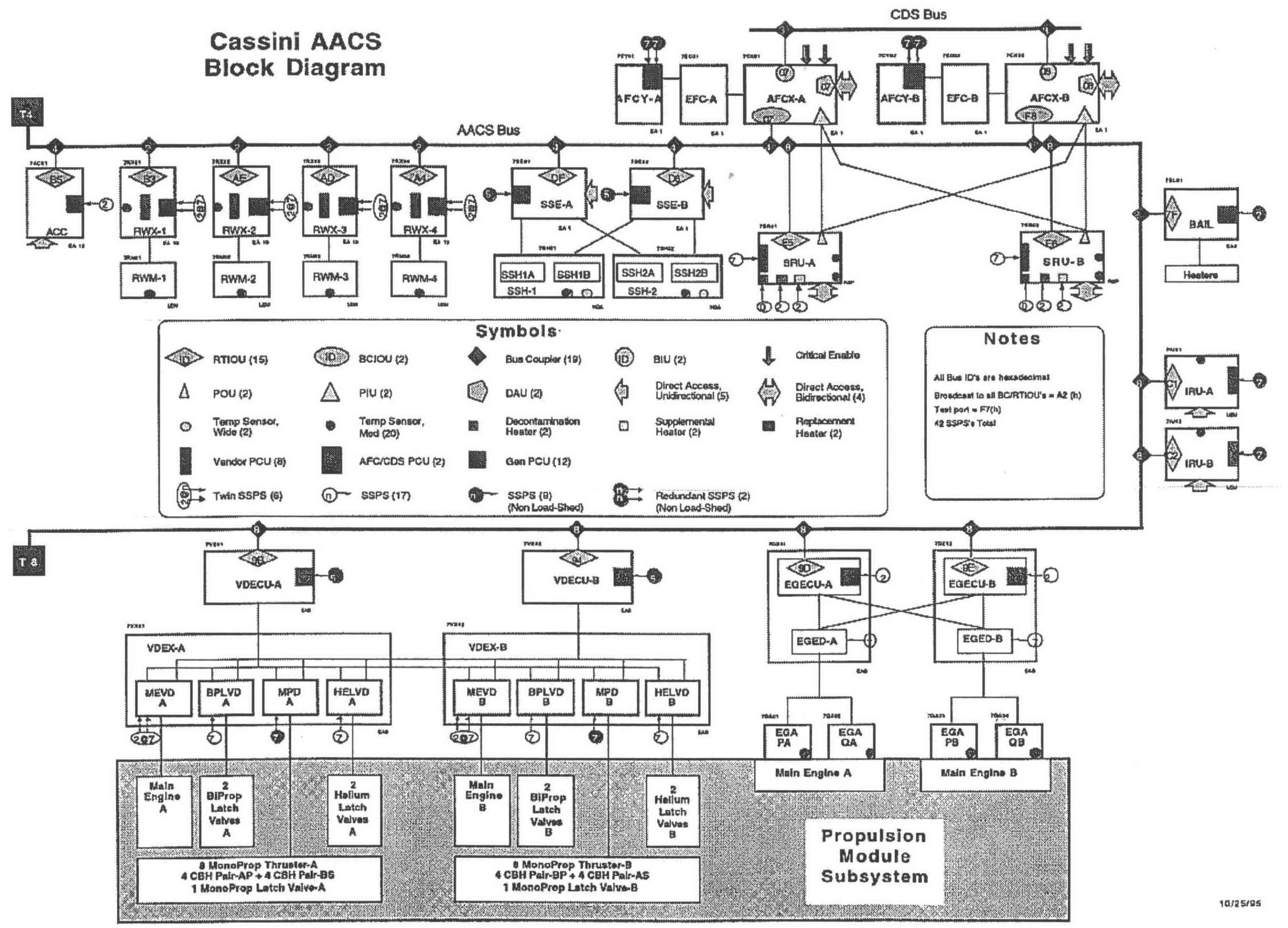


Figure 2-1 Attitude and Articulation Control Subsystem Block Diagram

system has difficulty loading the flight computer with its software. The BAIL contains accelerated load format (ALF) data blocks that can load the AFC in the event of problems with the nominal loading procedure.

- Engine Gimbal Electronics/Actuators (EGE/EGA). The engine gimbal actuators articulate the gimbals attached to the dual main engines of the spacecraft. The actuators are controlled by the engine gimbal electronics, which interface with the AACS databus.
- Inertial Reference Unit (IRU): The dual-redundant inertial reference unit uses a set of four hemispherical resonating gyroscopes (HRG) to provide inertial rate information to the flight computer for use in attitude control. A direct access port is available for injection of simulated spacecraft angular rates.
- Reaction Wheel Assembly (RWA): There are four reaction wheels on the spacecraft. Three of these are arranged for use as actuators during attitude control. The fourth wheel is redundant, and can be positioned to replace any of the three primary wheels in the event of a failure.
- Stellar Reference Unit (SRU): The dual-redundant stellar reference unit is a sensor used to determine position and attitude during the cruise portion of Cassini's mission. It utilizes a charged coupled device (CCD) camera to detect bright bodies in its field of view and this data is passed to the AFC via a pixel interface unit (PIU) for processing. During testing, the SRU receives picture information from an Image Emulation Unit through a direct access port.
- Sun Sensor Assembly (SSA): The dual-redundant sun sensor assembly is used to detect the position of the sun during attitude determination. Sun sensor heads are placed on the high gain antenna and generate voltages proportional to the amount of light that hits the heads.
- Valve Drive Electronics (VDE): The valve drive electronics are used to interface with the Propulsion Module Subsystem to open and close the various thrusters used for attitude control and as well as the main engine valves.

2.2 AACS Closed-loop Testbeds

2.2.1 ITL

The primary lab for integration and testing at the subsystem level is the Integration and

Test Laboratory (ITL). The purpose of the ITL is to perform hardware integrations of the subsystem and test functional sequences of the Cassini mission. All flight hardware, as well as engineering models or flight spares, are tested in the laboratory to ensure proper electrical configuration when the assembly is integrated with the AACS databus, the Power and Pyrotechnic Subsystem (PPS) and the Propulsion Module Subsystem (PMS). For this purpose, there are hardware simulators of these subsystems in the laboratory that allow testing of the interfaces to these subsystems.

For testing functional sequences, the ITL has an extensive set of support equipment hardware and software. The hardware and software work together to simulate the external interfaces to the AACS that permit closed-loop testing. Support equipment hardware consists of a series of computers and additional equipment which interface with the users of the system as well as the hardware. This equipment includes an Inertial Sensors Controller that permits biasing of the accelerometer and inertial reference unit; the assembly simulator hardware for the engine gimbal actuators, the inertial reference unit, and the reaction wheel assembly; electronics to generate bias voltages to send to the sun sensor assembly; star field data to send to the stellar reference unit; and equipment to interface with the AACS Flight Computer. The hardware also includes the simulators for the subsystems that interface with the AACS [4].

Support equipment software is an extensive network of computer programs working to simulate the outside environment of the spacecraft. The programs fall into two large groups, real time and non-real time. Real time programs consist of the assembly simulator software and subsystem simulation software, as well as software to monitor different assemblies and report on their status. The non-real time software includes tasks such as user console interfacing and generation of displays.

The software also uses the concept of a "blackboard" to provide for data visibility across the simulation. The software runs on a set of five processors, called chasses. These processors must work in a synchronized fashion and timing is very critical. Therefore, the processors use shared memory to facilitate data transfer. All of the chasses use the same set of memory to read and write variables. This means that all of the processors know what the state of the system is at any given time. This is analogous to how a common blackboard is used so that all those in a room have a consistent data set [6].

Finally, the dynamics of the simulation are propagated in real time via a computer program called DARTS (Dynamic Algorithms for Real Time Simulation). This program, developed by the Jet Propulsion Laboratory, computes the time rate of change of the state of the Cassini spacecraft as a dynamic system [5]. The program accepts any number of actuators, sensors, and flexible modes, making closed-loop dynamics testing of various Cassini configurations possible.

The ITL permits limited system mode testing. The Command and Data Subsystem is usually simulated in the laboratory, but for some testing, the actual CDS is used. The AACS Flight Computer takes its commands from the actual command computer, permitting testing of this interface in the laboratory before assembly, test, and launch operations begin.

2.2.2 Cassini AACS Test Station

There also exists the Cassini AACS Test Station (CATS) for development of flight and support software. CATS is similar to the ITL in that actual hardware is present in most cases, support equipment hardware processors are used to interface with the hardware and users, and closed-loop dynamics testing is possible with the use of DARTS. The major difference between CATS and ITL is their purpose. The integration and test laboratory is used to test electrical interfaces of the flight hardware, and thus all hardware that is flight certified is first tested in that laboratory before delivery to the Spacecraft Assembly Facility. The ITL is also the formal test and validation environment for the subsystem. CATS, on the other hand, is used to test the software of the system. The hardware is present, but electrical breadboards are primarily used in CATS. The functionality of these breadboards is identical to the flight units in most cases, but shielding, grounding and electrical interfaces may be different.

2.2.3 Assembly, Test, and Launch Operations

Assembly, Test, and Launch Operations (ATLO) for Cassini began in late 1995 and will continue through launch of Cassini in October, 1997. Testing primarily occurs at the

Spacecraft Assembly Facility at JPL. At this location, the flight hardware is integrated together for the final time. Also, many of the simulators are not present. The Power and Pyrotechnic Subsystem hardware is there, and thus all power comes from this subsystem. There are no assembly simulators once all the hardware is integrated. The Command and Data Subsystem is also present. However, the Propulsion module subsystem simulation is still running in ATLO due to the danger to personnel of testing main engine firings and reaction control thrusters. The Propulsion Module is tested separately for the majority of the testing period. During most testing in ATLO, the DARTS simulation is still present and permits closed-loop testing. The exception is during environmental testing when the support equipment is disconnected and no closed-loop testing is conducted.

Throughout the testing plan, the engine gimbal actuators, reaction wheel assemblies, and inertial reference units are simulated in both the integration and test laboratory and in the Cassini AACS test station. The real hardware is present for interface checkout in the ITL and during ATLO testing. The next sections go into detail concerning the three assembly simulators and evaluates their usage in the testing of Cassini's AACS.

Chapter 3

Engine Gimbal Actuators

3.1 Description

The purpose of the engine gimbal actuators is to rotate the main engines of the propulsion module subsystem about their gimbal axes in response to commands by the AACS flight computer. The main engines are rotated such that the thrust vector passes through the center of mass of the spacecraft, as well as in the desired inertial direction. The spacecraft sensors are used in conjunction with the EGAs to determine this direction as well as to determine when the required velocity change has been achieved.

The signal flow for the EGAs is shown in Figure 3.1. The actuators act on extension commands provided by the AACS flight computer through the engine gimbal electronics. The extension is controlled by comparing the actual positions of the actuators to the desired position and correcting the position by altering the motor voltage. The actual positions are determined by the feedback signal from a Linear Variable Differential Transformer (LVDT) that is attached to each actuator. The electronics that control the position consist of a control unit and a driver. The driver accepts voltage commands from the control unit and generates excitation signals for both the motor and the LVDT of the actuator. The control unit accepts commanded extensions via the AACS databus and the LVDT feedback signal. The control unit generates a digital representation of the LVDT signal for the databus as well as the voltage commands for the driver.

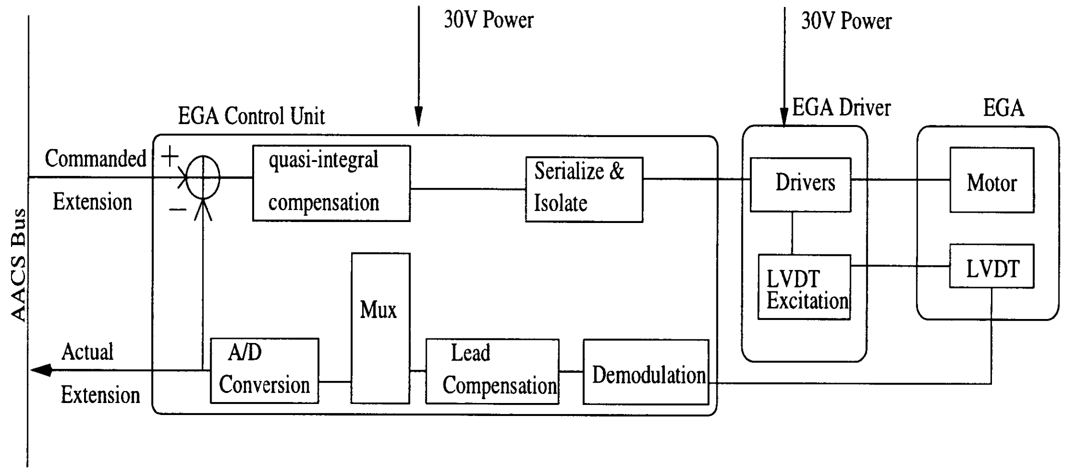


Figure 3-1 Engine Gimbal Actuator Block Diagram

3.2 Cassini Laboratory Configuration

The Engine Gimbal Actuators are simulated in the ITL and CATS through a combination of a hardware simulator and a software dynamics interface. The purpose of the hardware is to simulate the engine gimbal actuators and generate a feedback signal that represents the LVDT signal for the real actuators. The dynamics software accepts the commanded extensions and the LVDT extension information and then computes the main engine thrust vector direction for use in the DARTS simulation.

3.2.1 EGA Hardware Simulator

Description

The EGA hardware simulator was designed to simulate the LVDT feedback information that is supplied by the real EGAs in response to the LVDT excitation signal and the motor drive input. A block diagram is shown in Figure 3.2.

To simulate the LVDT, the EGA hardware simulator consists of a motor simulator and the LVDT simulator. The motor simulator accepts the EGE drive signal and passes it through an optical isolator. The signal is scaled such that a position signal is generated that is proportional to the commanded position of the actuator. This signal is then multiplied by the LVDT excitation signal, also received from the EGE. The multiplication retains the sign of the motor drive signal. The resulting signal simulates the LVDT feedback signal.

Validation Test

When the engine gimbal actuators are integrated into the subsystem in the ITL, they are integrated per a hardware integration procedure. This procedure exercises the actuators and ensures that they perform adequately. Since the simulators and the real actuators are integrated using the same procedure, the data can be compared to evaluate the simulators.

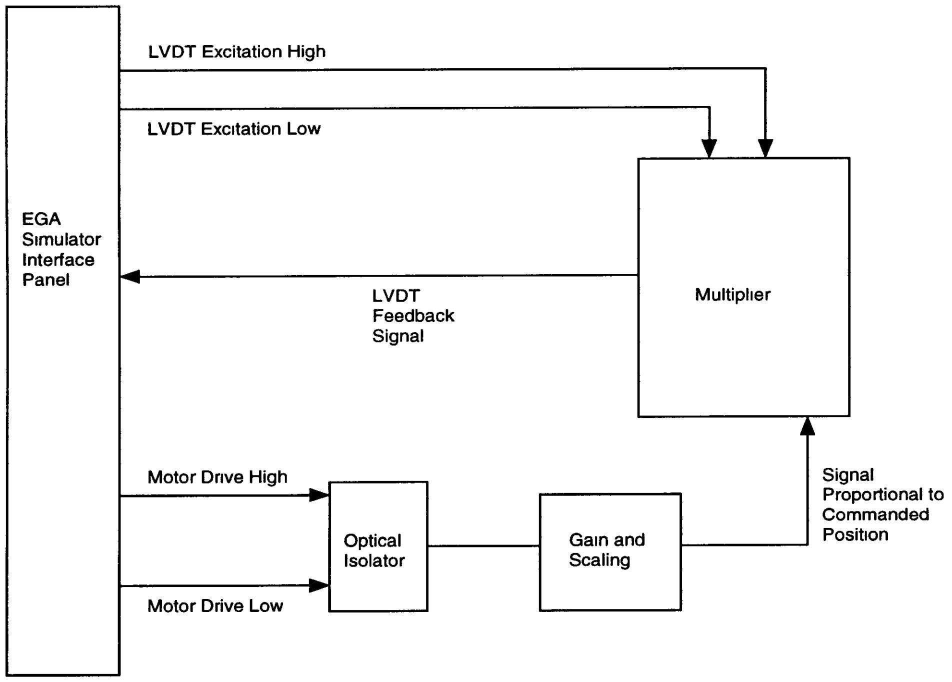


Figure 3-2 Engine Gimbal Actuator Hardware Simulator

The actuators (or simulators) were exercised through a series of extensions during the integration procedure. The results of this procedure were that the simulators perform to within specified tolerances and track the hardware very well. These results are summarized below:

- Average absolute deviation from the commanded position for simulation serial number 005 was 0.033 mm.

- Average absolute deviation from the commanded position for simulation serial number 010 was 0.022 mm.
- Cassini AACS requirement: 0.1 mm deviation.

Even though the AACS requirement was met on average, the maximum error did deviate from the AACS requirement for both of the simulators. This deviation was deemed acceptable for testing, however. Cassini fault protection will activate if the deviation is greater than 0.27 mm for two consecutive readings of the EGA (readings occur every 125 ms). This behavior was not observed when testing either the EGAs or the EGA simulators and thus the EGA simulators were accepted for testing. Due to the variance of the EGA positions, the power measurements of the EGA simulators also had more variance than with the flight equipment, but this was also acceptable [7].

Differences and Problems

One problem that occurred during testing was that a fault protection error in flight software was not found during integration testing. The error was discovered when the fault protection autonomously powered down the engine gimbal actuator driver during integration of the flight Propulsion Module Subsystem on the spacecraft. The investigation concluded that the EGA simulators were not designed to simulate an EGA under actual flight loading conditions. Therefore, the first time the flight software interfaced with an EGA loaded onto a gimbal and a Main Engine was during Assembly, Test and Launch Operations (ATLO). This brought out one of the important lessons learned through AACS testing. If the requirements are not stated clearly at the outset and thought through in their entirety, unforeseen events may occur. The result of this testing in ALTO was a decision to use flight spare EGAs attached to a load fixture for future testing. Thus the EGA simulators will not be used for post launch ITL analysis activities.

The other difference between the EGA simulators and the actual Engine Gimbal Actuators is that the real actuators to not drift from their last commanded position when the engine gimbal electronics are not holding them there. The simulators, on the other hand, have a tendency to drift away from their last commanded position without the engine gimbal electronics holding them there. This difference does not impact most testing, however, since most sequences require the actuators to be held in a particular position.

3.2.2 EGA Dynamics Model

Description

To properly represent the motion of the EGAs in the closed-loop dynamics simulation, the support equipment software must ensure that the main engine thrust vector direction is consistent with the EGA extensions. To accomplish this, a software model of the EGA accepts information from the EGA (or EGA simulator) and computes the main engine thrust vector. The relationship between the EGA (or simulator) and the EGA model is shown in Figure 3.3 and is described below.

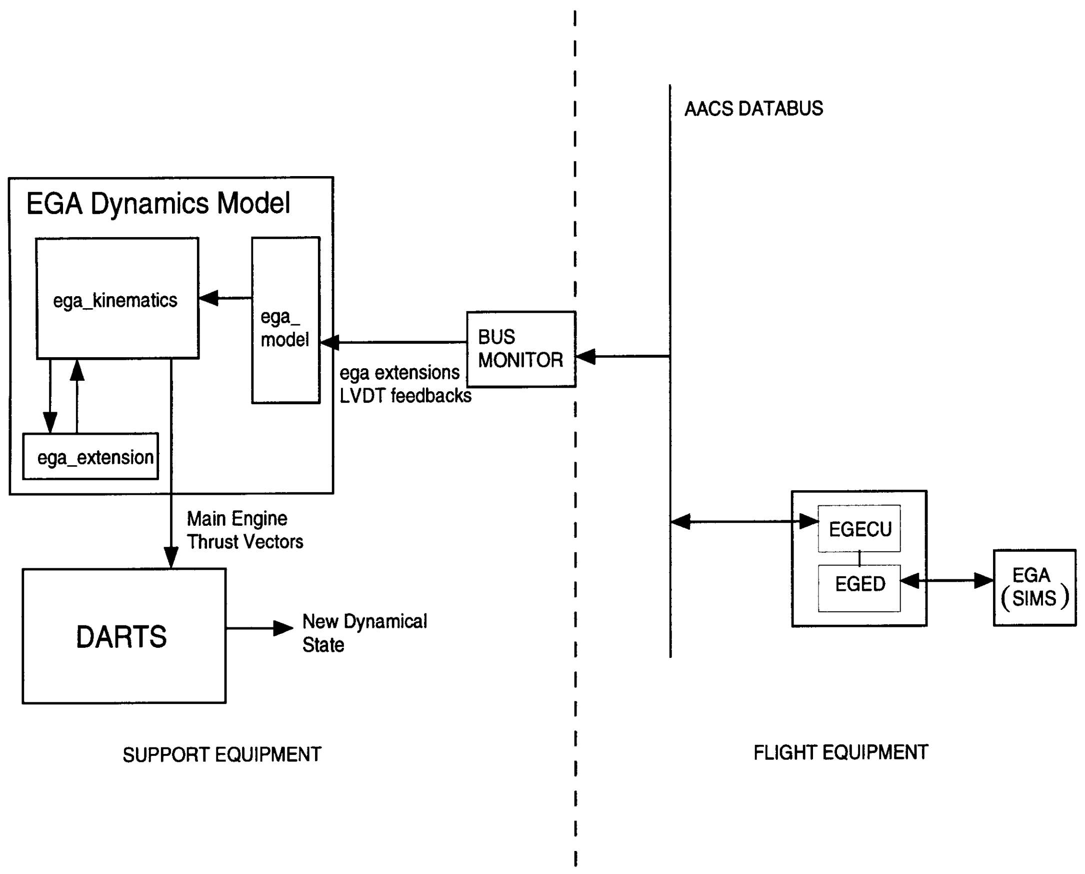


Figure 3-3 Engine Gimbal Actuator Dynamics Model

The model consists of three modules: ega_model, ega_kinematics, and ega_extension. The first module, ega_model, accepts the LVDT signal and commanded extension information and computes the new extensions of the actuators. This is done without regard to dynamics; the extension is simply set to the current commanded extension. The only exception is if the step required to update the position is too high. In this case, the model updates the extension with a series of smaller steps. Once the extensions are computed, ega_kinematics is called. This module calculates the thrust vector by first computing the gimbal angles (accomplished by the module ega_extension) and then transforming the angle information into the main engine thrust vector direction. Once the thrust vector direction is known, this information is used during the next iteration of the dynamics simulation.

Validation Testing and Results

The EGA model was validated by comparing resulting engine gimbal angles to analytical predictions. The EGAs were commanded to several different positions and the computed main engine thrust vector was recorded. Given this thrust vector, a solution for the EGA position was derived and compared to the commanded position. The data was consistent and the model was validated in this isolated case. Figure 3.4 shows the predicted and actual values for the EGA extensions in response to several commands to stroke the gimbal actuators.

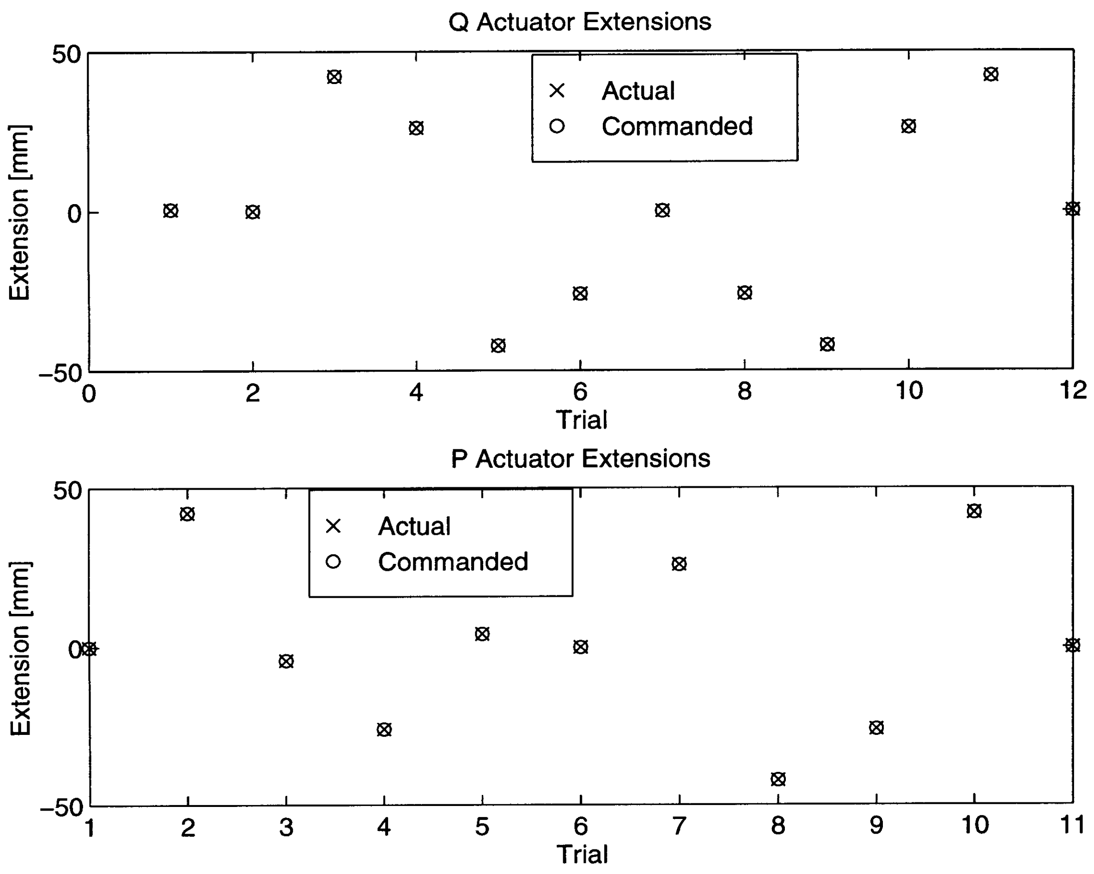


Figure 3-4 Engine Gimbal Actuator Validation Test Results

3.2.3 Closed-loop Simulation

The two primary test activities of the laboratory that have validated the described EGA simulation have been the Main Engine Trajectory Correction Maneuver (TCM) Testing and Fault Injection Testing. The purpose of the Main Engine TCM is to test the hardware and software under a realistic set of circumstances where the main engine is used to alter the path of the spacecraft. Thrust Vector Control (TVC) is performed by the flight software and it is critical that the dynamic model of the actuators alter the thrust vector as commanded by the flight software. As the following data in Figures 3.4 and 3.5 show, the

EGA simulators performed well in a closed-loop environment. The X and Y components

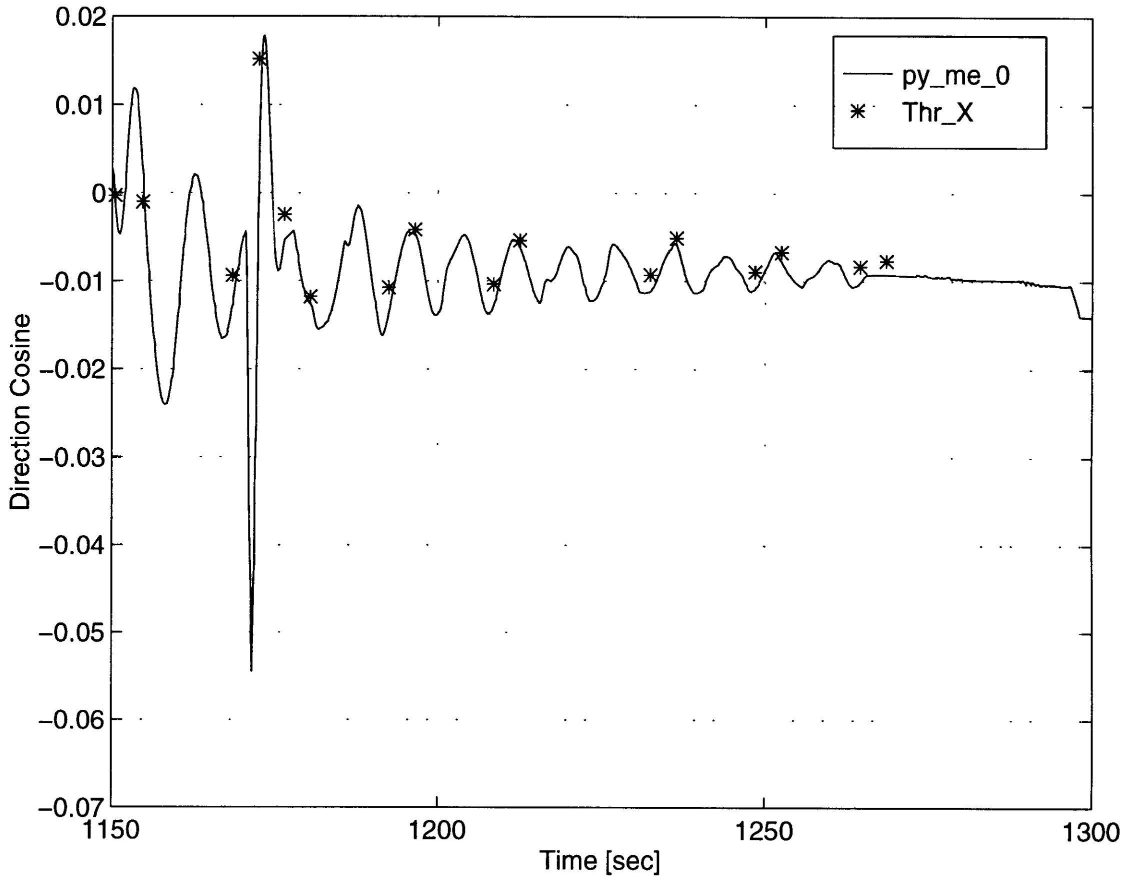


Figure 3-5 Simulated and Flight Software Thrust Vectors in the X direction

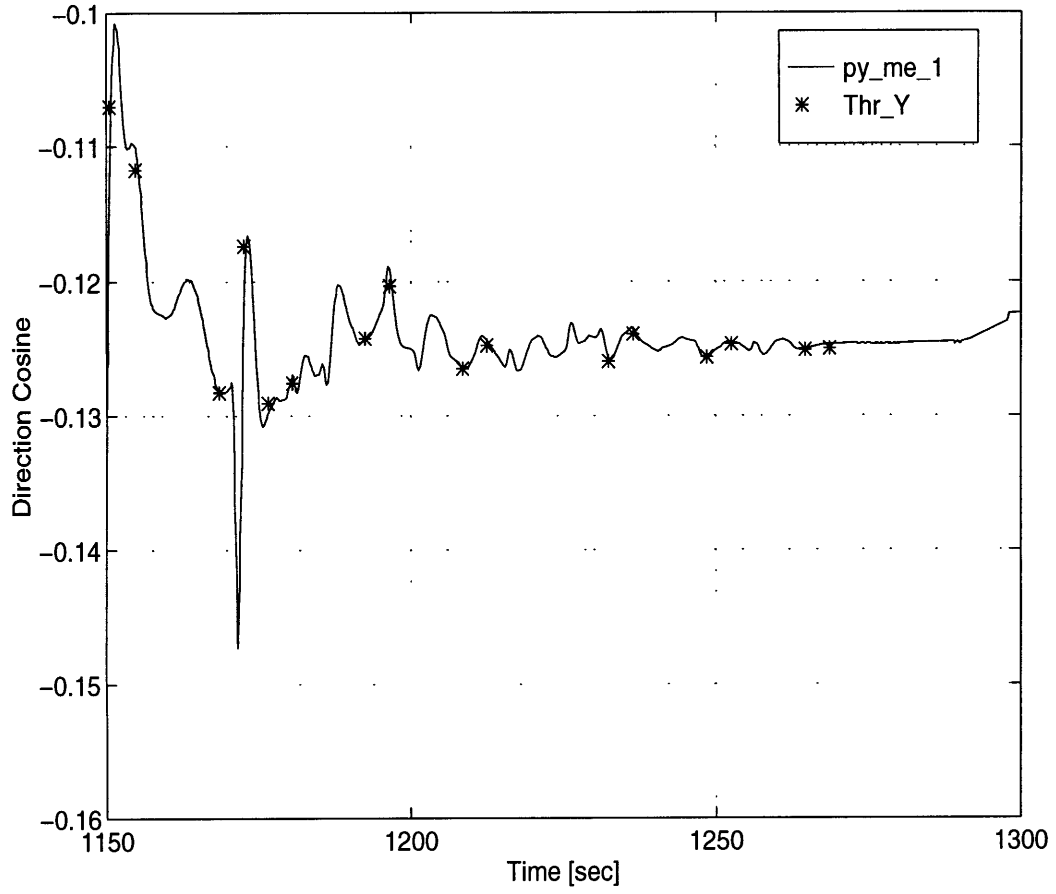


Figure 3-6 Simulated and Fight Software Thrust Vectors in the Y direction

of the thrust vectors of the simulation(py_me_0 and py_me_1) and what flight software believed to be the thrust vector (Thr_X and Thr_Y) matched very well. Because the Z axis direction cosine is very close to one, the flight software does not record Z axis data and thus a Z component comparison is not available.

Another closed-loop aspect of the EGA simulation is the task of injecting faults into the simulation to test the fault protection responses. The EGA simulator was designed to interface with the support equipment exactly like the real hardware. Thus, it was not possible to simulate any faults that involved hardware failure. The software dynamics model could have been altered to simulate many faults, but since there is no way to alter the EGA hardware simulator there was no need. If the model was to simulate a stall, for example, it could easily be disabled from the rest of the simulation. This would not be possible for the hardware, however, and the result would be flight software receiving an external disturbance with no indication that a stalled EGA is the cause.

3.3 Evaluation

The EGA simulation met the requirement for testing of the engine gimbal actuators. Specifically, the EGA simulator did not cause fault protection to activate unexpectedly during testing in either CATS or ITL. This facilitated testing of the flight software, as well as the Main Engine Trajectory Correction Maneuver. Due to the nature of the hardware simulator, testing of the fault protection capabilities was not performed. When the flight actuators were integrated with the AACS and the flight propulsion subsystem, fault protection did activate unexpectedly. But since the EGA simulators were never meant to simulate a loaded EGA, the conclusion is that the EGA simulator did perform adequately during AACS testing.

Chapter 4

Inertial Reference Unit

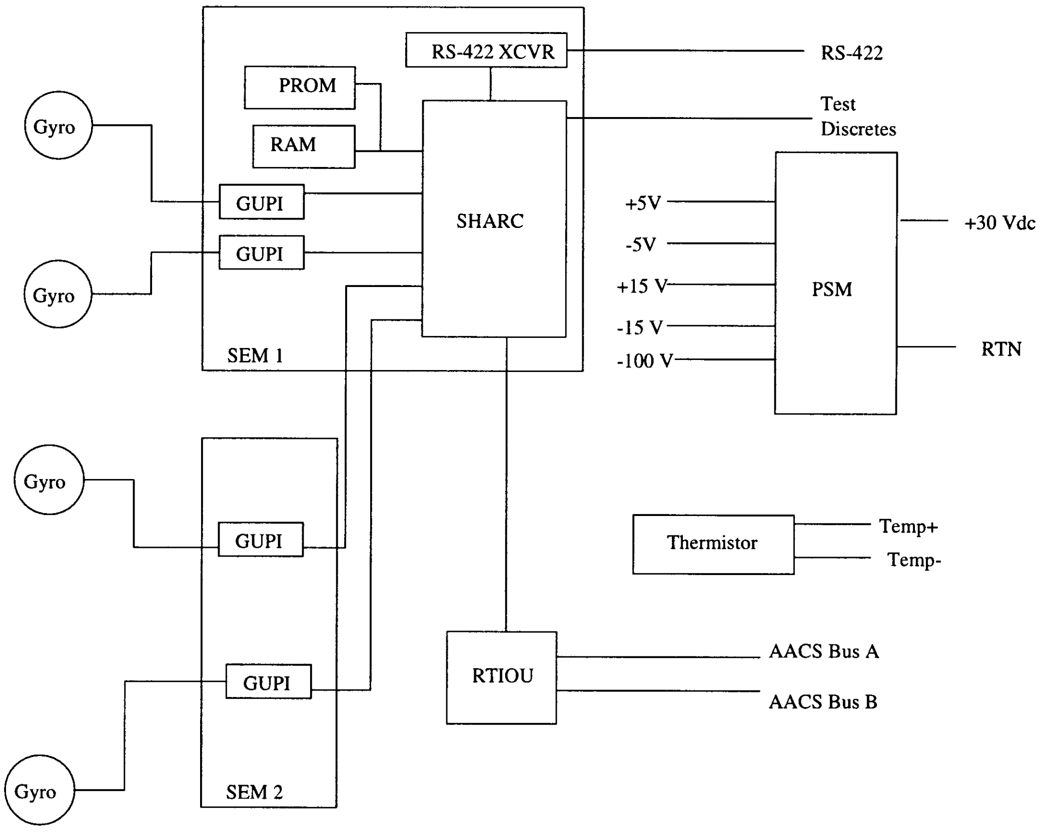
4.1 Description

The Inertial Reference Unit (IRU) is a dual-redundant assembly that is used to detect the inertial angular velocity of the spacecraft. Each IRU contains four hemispherical resonating gyroscopes and processing electronics. The IRU contains its own processing circuitry that interfaces with the AACS databus RTIOU and the gyroscopes. A block diagram of the IRU is shown in Figure 4.1 [8].

The gyroscopes each run at a frequency of approximately 2000 Hz. The gyroscope interface unit obtains a rate estimate from each gyroscope every cycle. The main processor has software that runs at 100 Hz. Each time the software loop is executed, the rate measurements since the last time the software was executed are integrated and the angle is added to the data that will be passed to the flight computer. The flight computer can read the accumulated angle or the IRU status from the inertial reference unit. The flight computer can also write data to the IRU. This data, for example, would be new software to the main processor in the event of an IRU reset.

4.2 Cassini Laboratory Configuration

As in the case with the EGA simulation, there are two parts to the IRU simulation-- an assembly simulator and a dynamics model. The assembly simulator is software and hardware that represents the IRU when the actual hardware is unavailable in the laboratory.



SEM	Sensor Electronics Module	Circuit Card
SHARC	Space HRG and Accelerometer Readout Controller	Main processor IC (located on SEM)
GUPI	Gyro Unit Processor Interface	Gyro signal interface IC (located on SEM)
RTIOU	Remote Terminal Input Output Unit	External bus interface IC (located on SEM)
PSM	Power Supply Module	Power converter circuit card

Figure 4-1 Inertial Reference Unit Block Diagram

The dynamics model is used to generated biases for the IRU by converting the spacecraft angular rates to rates in the gyroscope sensing axes which are used to bias either the real gyroscopes or the simulator.

4.2.1 IRU Assembly Simulator

Description

The assembly simulator has the function of accepting biases from the dynamics interface and converting these biases into data for the flight software to interpret. This simulator consists of three parts. The first is a remote terminal input output unit (RTIOU). This unit is a remote terminal on the Cassini AACS databus and facilitates communication with the AACS flight computer. The second component is an interface card that communicates with the RTIOU. This card is a series of static RAM registers that the simulator can write to. This allows the software to act as the actual hardware by interfacing with the RTIOU similar to the actual hardware. The third component is the actual software for the simulator. The software supports five functions of the IRU: power on initialization of the IRU output data, IRU Built in Test (BIT), IRU SHARC software download, IRU "soft reset," and normal operation of the IRU. The data flow through these five functions is shown in Figure 4.2

Upon receipt of a power on command, the simulator sets all output information to zero and sets a timer to begin the BIT. The BIT simulation is a one second hold that simulates the time for the SHARC built in test. Once this hold is complete, the IRU simulator reports this information via the RTIOU to the flight software and sets a flag indicating readiness to begin the software download. The software download function is simulated by verifying receipt of the data sent by the flight software and then indicating a valid checksum and a valid load. The soft reset is another one second hold for the simulator. After this one second delay, the software simulation reports a good status message back to the flight software.

Finally, there is the actual operation of the IRU. The software checks for a soft reset and if none occurs, the software proceeds to calculate the outputs of the gyroscopes. Since the IRU simulator receives the rates the gyroscopes sense in the gyroscope coordinate system, the simulator simply has to convert these rates into a change in angle and properly format this information for the remote terminal on the databus. This is accomplished by multiplying the rate received by the IRU cycle time and adding this angle to the last angle computed to determine the accumulated angle. Once this is accomplished, the angle is converted to a form compatible with the RTIOU and the data is passed to the flight software.

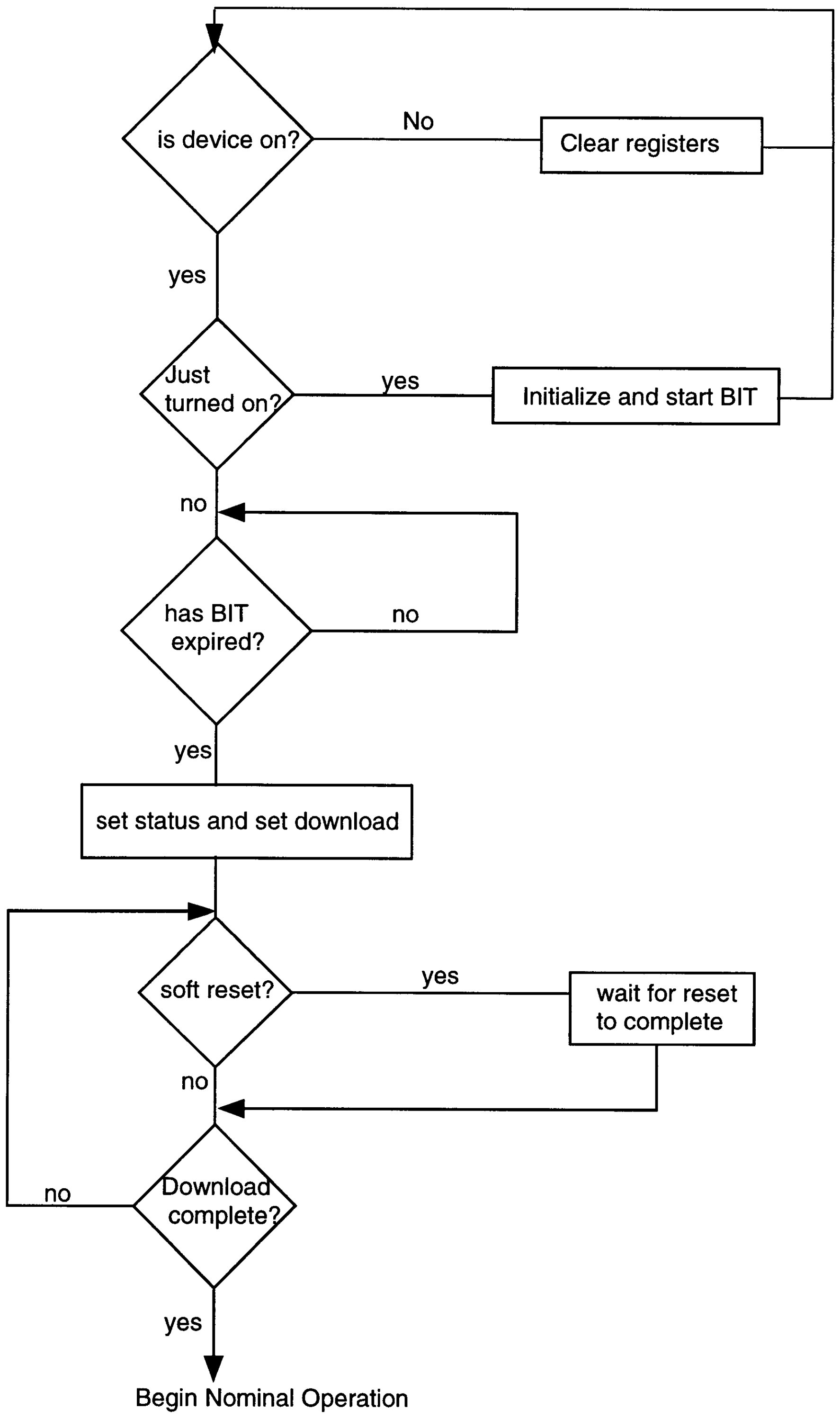


Figure 4-2 IRU Assembly Simulator Flow Diagram

Validation Testing and Results

This simulator was integrated into the ITL and CATS with the same integration procedure that was used for the actual flight hardware. Thus, we have a common test to compare the simulator and the hardware. During the integration procedure, the gyroscopes are biased with a set of support equipment commands. Figures 4.3 and 4.4 show the results of these commands.

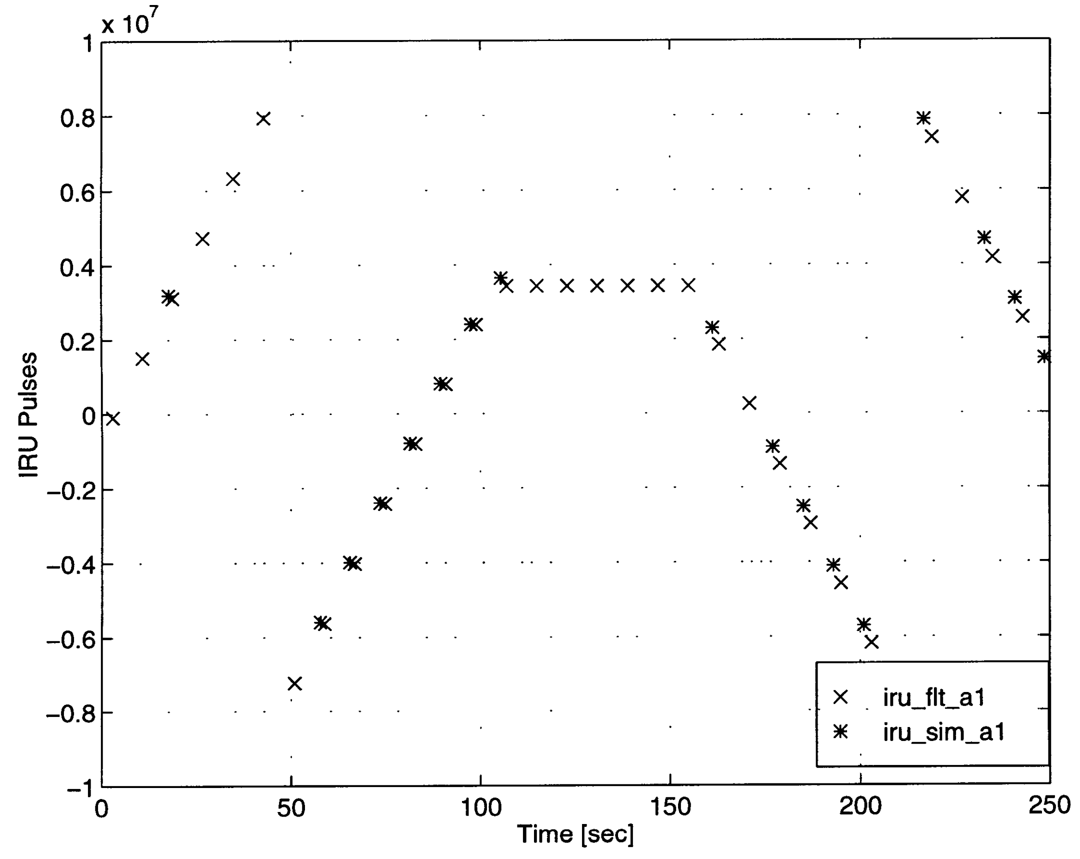


Figure 4-3 IRU Simulator and Hardware Comparison

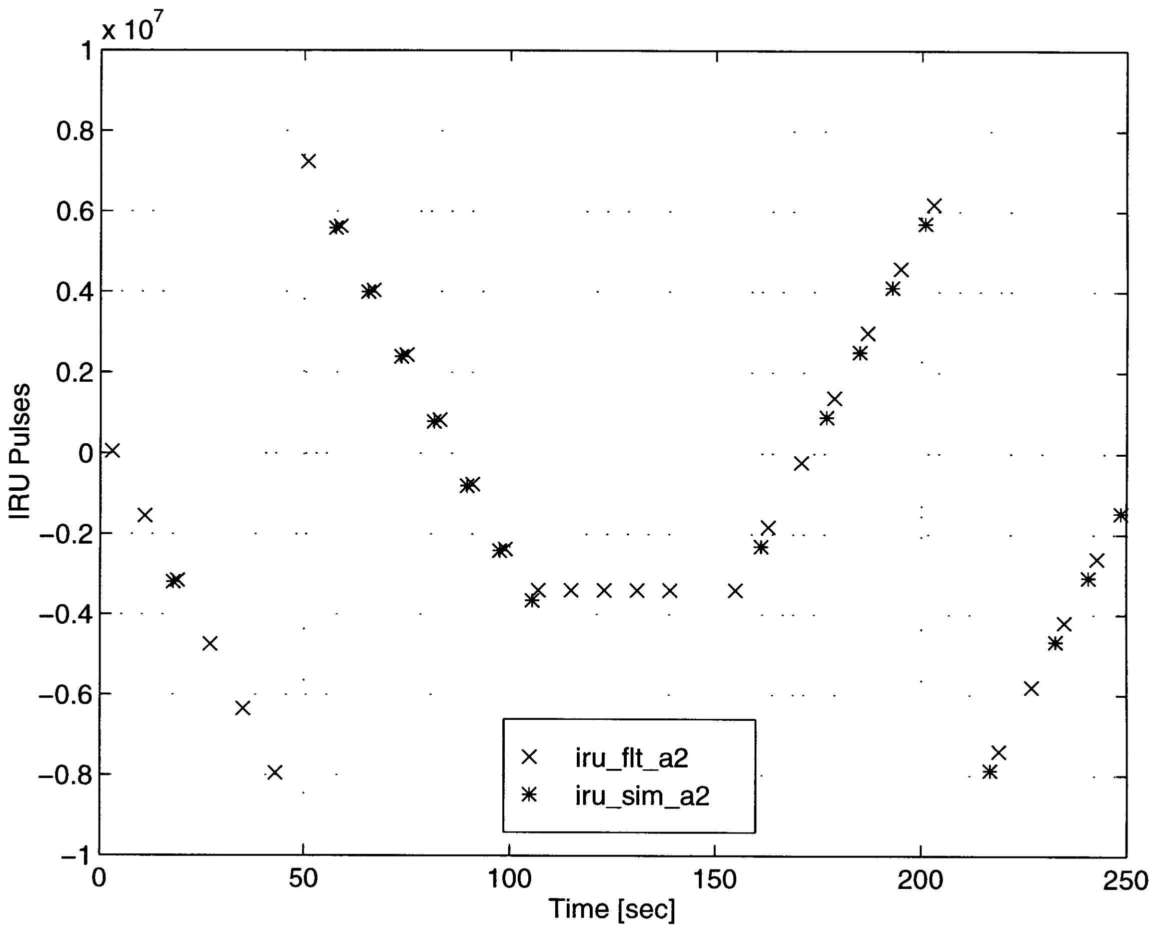


Figure 4-4 IRU Simulator and Hardware Comparison

As the data show, the IRU simulators matched the performance of the flight equipment well.

Differences and Problems

Integration revealed some problems with the simulator that were corrected to increase the fidelity of the software. The first problem was with the processing cycle. The original version of the software integrated at the same speed as the dynamic simulation, 16 Hz. This was not consistent with the actual hardware, which reads data from all four gyroscopes at a speed of 100 Hz. With the simulator running at 16 Hz instead of 100 Hz, unrealistically large changes in angles could be reported by the simulator. It would be more consistent with the hardware if the software ran at a speed of 400 Hz and processed one gyroscope at a time. Then, when all four gyroscopes were processed, this data could be made available to the flight software through the RTIOU. This configuration change was implemented and performed well.

A second problem was with the noise of the IRU. The original model did not have any simulation of gyroscope noise and once again a change would improve fidelity. The final solution was for the model to cycle through a large set of experimental data representing realistic numbers for gyroscope noise. These numbers would be added to the output of the gyroscope and would simulate noise coming from the IRU gyroscopes.

A final problem was with the timing of the simulator. In reality, the IRU has a double buffer that is used to prevent the flight software from reading partially updated data. One of these buffers always has a complete packet of information for the flight software. However, the assembly simulator card does not support double buffering. Thus, flight software was reading the information out of the IRU simulator, but the information was not completely updated. To correct this problem, the assembly simulator card was modified such that when flight software was reading the information from the card, a signal was sent to the IRU assembly simulator. The software was changed to verify this signal was not active before a read was attempted. If this signal was sensed, output was delayed until the read was concluded. This correction worked well, resulting in valid information sent to the flight software.

Our conclusion is that once coding errors are corrected, the simulator does act sufficiently like the hardware to permit testing. However, there are several aspects of the hardware that are not simulated. The actual computation of the SHARC is not simulated. For example, the Built In Test is not actually performed. Instead, a timer simulates the delay the BIT would cause in the processing cycle. Similarly, the checksum during the download of the new SHARC software is not performed. Again, a timer is used to simulate the delay of the download and checksum. Simulation of the actual SHARC software was not an objective in the IRU simulation. The basic philosophy was that the inputs and outputs of the IRU simulator would be as identical as possible to the inputs and outputs of the hardware. Thus, the simulator can report that the data has been received or that a BIT has been performed, but the BIT or data download may not have actually happened in the simulation software. This black box concept has important consequences when fault injection is considered.

One of the advantages of using simulators is that the test analyst has the ability to simulate faults without damaging actual hardware. However, with the IRU simulator, any failures of the SHARC or other processing electronics cannot be simulated. The simulators were designed such that the laboratory environment only has a command path to the simulator if it has a command path to the actual device. Therefore, since one cannot command a BIT failure with the actual hardware, for example, it was decided that one would not be able to with the simulators. Any fault injection would have to be possible with either real hardware or simulators and thus involve changing existing inputs into the devices. For example, it was possible to simulate a failed gyroscope by sending a large bias into one of the gyroscopes. The SHARC or IRU simulator would see this large rate on one of the gyroscopes and declare that gyroscope's data invalid. But failures internal to the IRU would have to be simulated in another testbed.

4.2.2 Gyroscope Dynamics Model

The gyroscope model within the dynamics simulation computes the biases that are sent to the IRU assembly simulator or the IRU hardware. First, the locations of the gyroscopes and the transformations of the angular rates of the spacecraft from the spacecraft coordinate system to the gyroscope sensing axes are determined and loaded into the model. Then the model accepts the spacecraft angular rates from the DARTS simulation and converts these to rates that the gyroscopes sense. Then this information is passed to the IRU model described above. The interaction between the model and the simulation is shown in Figure 4.5

The gyroscope dynamics model was validated as a part of the closed-loop simulation described below.

4.2.3 Closed-loop Simulation

The inertial reference unit simulator was used extensively in all laboratories during the testing of the Cassini AACS. During testing, the reported spacecraft rate and the simulated rate were examined to determine how effectively the simulated angular rates of the dynamics simulation were reported to the flight software. This is shown in Figures 4.6 through 4.8. The variables b_ang_rt_x, b_ang_rt_y and b_ang_rt_z represent the angular rates being simulated and the variables X_rate, Y_rate, and Z_rate are what flight software is reporting the angular rate to be after converting data from the IRU simulator. Both sets of rates tracked extremely well, and we concluded that the IRU simulator performed well in the closed-loop environment.

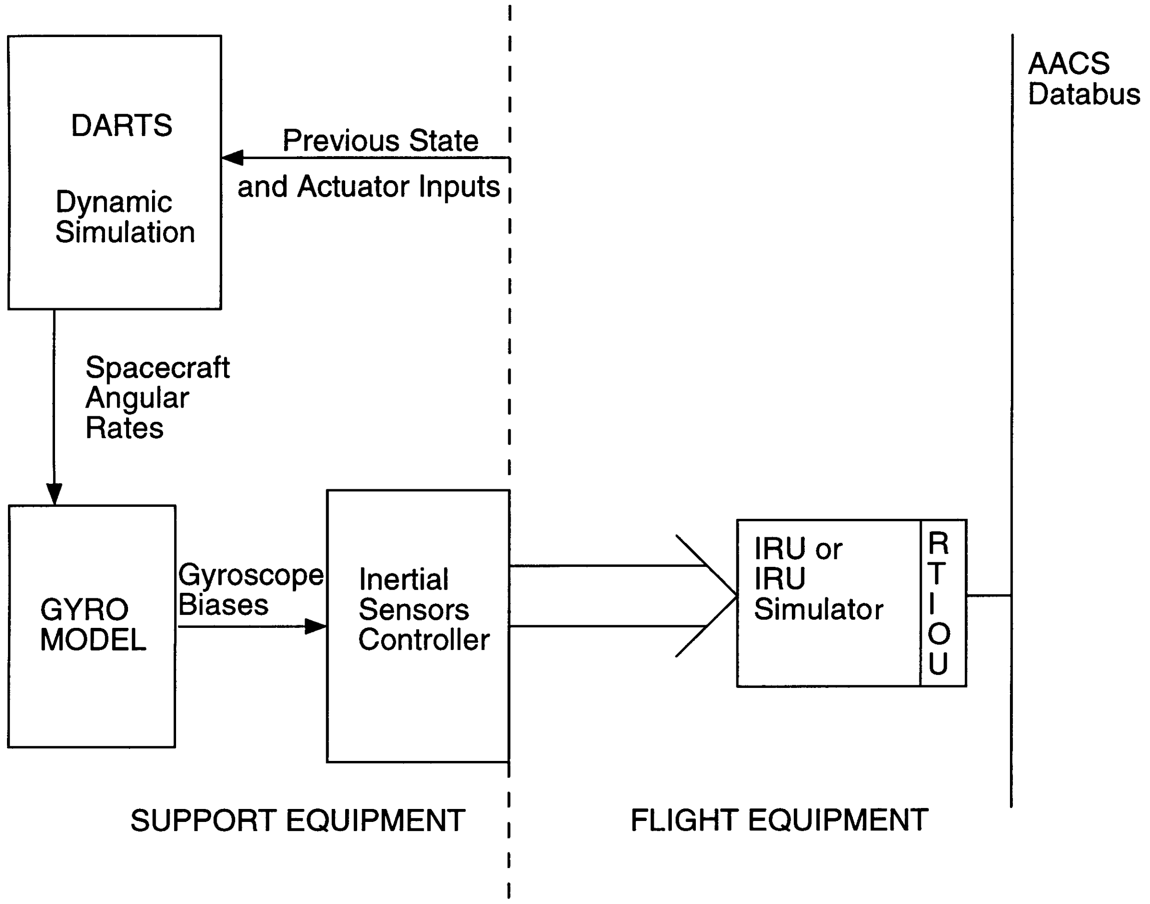


Figure 4-5 Inertial Reference Unit Dynamics Model

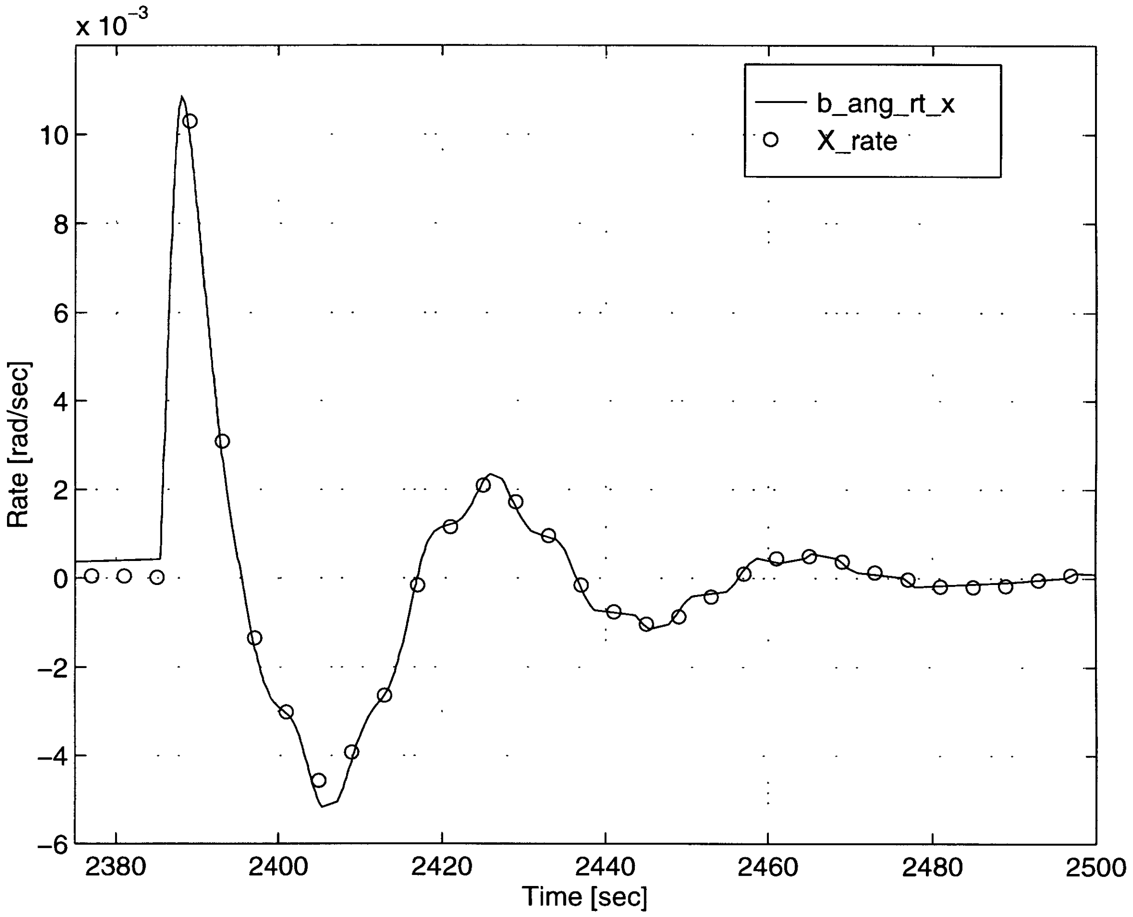


Figure 4-6 Simulated and Flight Software Angular Rates, X Axis

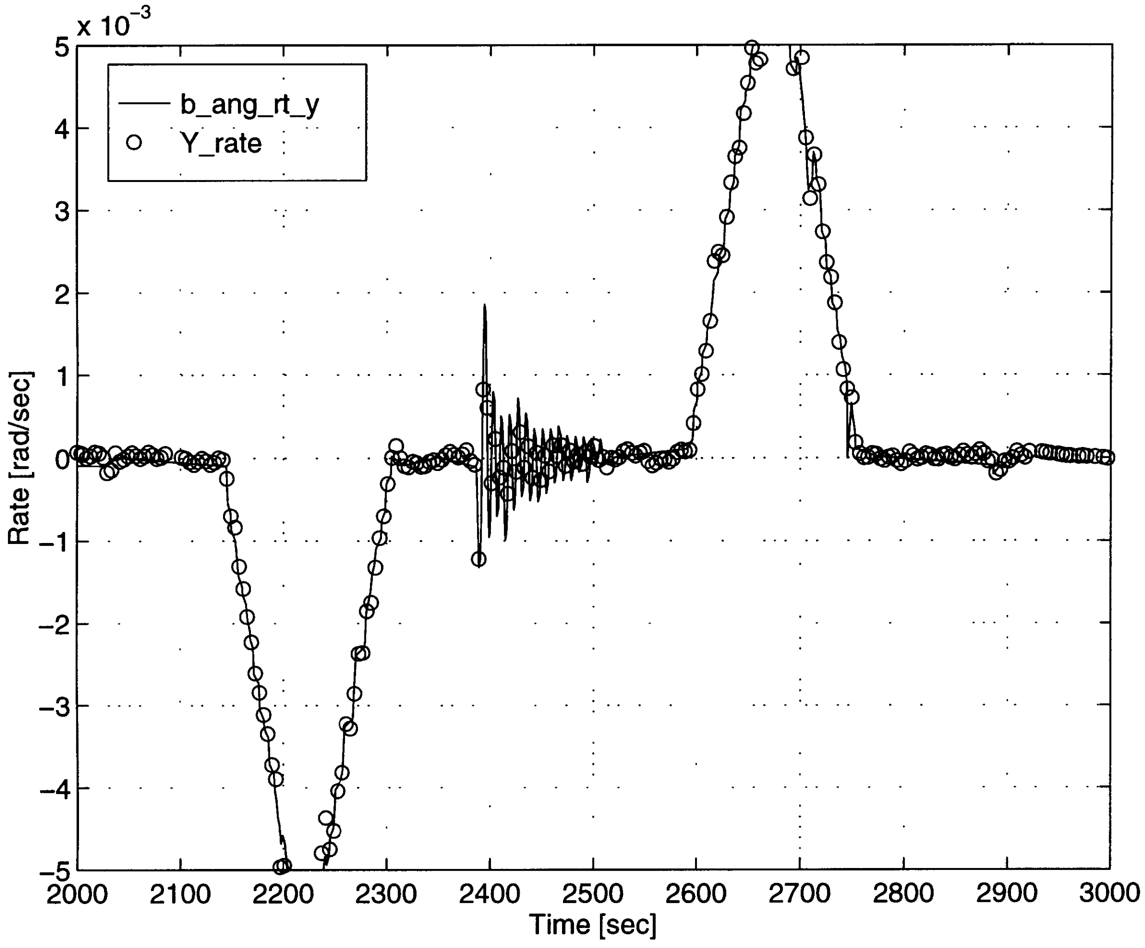


Figure 4-7 Simulated and Flight Software Angular Rates, Y Axis

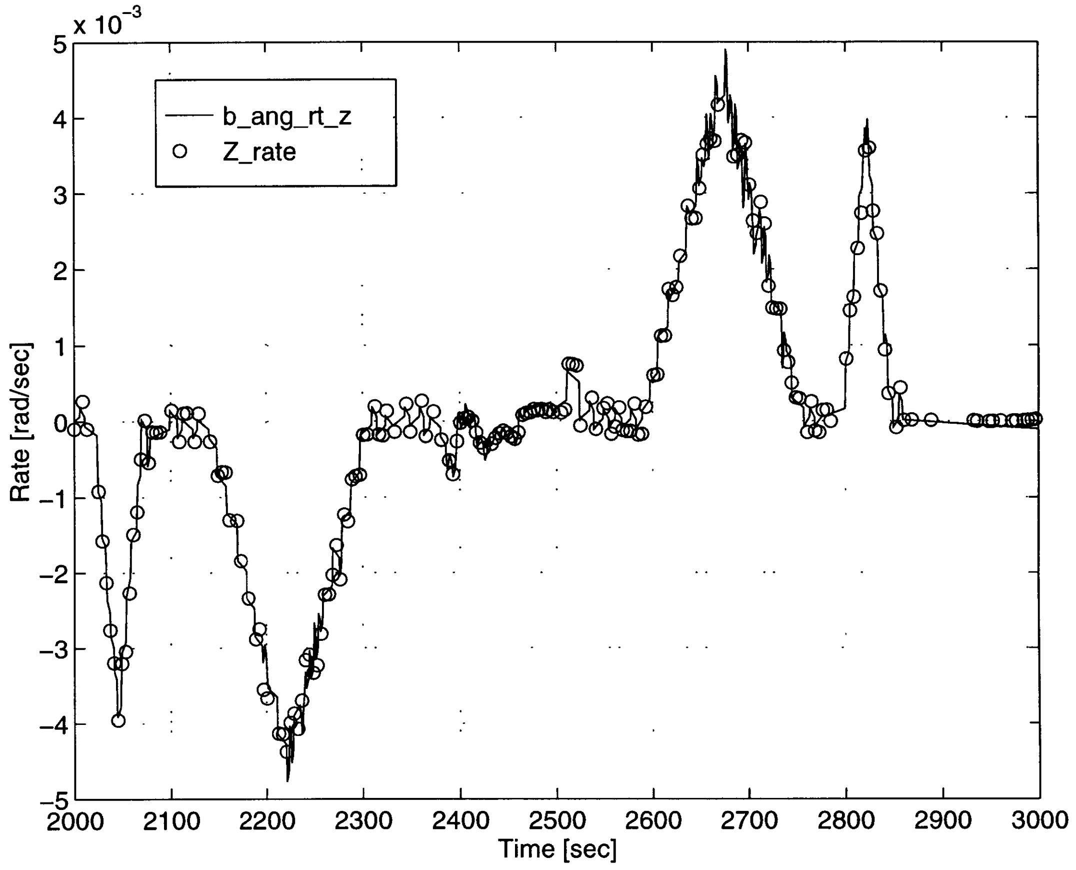


Figure 4-8 Simulated and Flight Software Angular Rates, Z Axis

4.3 Evaluation

The Inertial Reference Unit simulator performed well in all three laboratories and has proven a valuable tool when IRU hardware was unavailable. The integration and test of the simulator revealed that the simulator had several shortcomings, but once these were corrected the simulator had a high degree of fidelity when compared to the actual hardware, resulting in excellent closed-loop performance during flight sequences. Even though fault injection testing was limited, the IRU assembly simulator represented the actual hardware adequately in the laboratory environment.

Chapter 5

Reaction Wheel Assembly

5.1 Description

The Cassini spacecraft has two types of attitude control actuators. The first is the Propulsion Module Subsystem consisting of the main engines and 16 reaction control thrusters. The second type of actuator is the reaction wheel assembly. There are four reaction wheels on the spacecraft, three that are the primary reaction wheels, and a fourth that is a backup reaction wheel. The purpose of the reaction wheels is to store angular momentum of the spacecraft, as well as to provide attitude control.

The reaction wheels receive commands from the flight software over the AACS databus. There are seven commands that are accepted by the reaction wheels. These are summarized in Table 5.1 [8].

Table 5.1: Commands and Responses for the Reaction Wheel Assembly

Command	Response
Read Delta Angle	Return the accumulated angle count, (including over (under) flow indicator) and reset counter to zero.
Read Torque	Return the current command torque setting for the RWA
Read RWA Current	Return the current value of total RWA electrical current.

Command	Response
Read Motor Current	Return the current value of RWA motor electrical current
Read Status	Return the current operational status data of the RWA
Set Torque	Hold the reaction torque output at the value specified in the command
Reset	Set delta angle pulse counter to zero before resumption of counting, and set torque command to zero. This response shall be automatically executed at the time power to the RWA assembly is commanded on.

The reaction wheels use a brushless DC motor to spin the wheels and a Hall effect tachometer to sense the motion of the wheel. Twenty four magnets are attached to the reaction wheel and Hall devices sense the motion of these magnets past a sensor. This sensor counts each 1/24th of a revolution of the reaction wheel and, along with a timer, forms a tachometer to measure reaction wheel speed.

5.2 Cassini Laboratory Configuration

5.2.1 RWA Assembly Simulator

Description

As in the case of the IRU, the RWA assembly simulator consists of a remote terminal input output unit (RTIOU) for the RWA, software for the simulation, and an interface card that permits communication between the two. The reaction wheel assembly simulation software simulates the dynamics of the reaction wheel for the purpose of calculating outputs to communicate with the AFC via the RTIOU. The inputs to the program are a torque command from the AFC and power on/off commands. The program's outputs are tachometer counts, wheel power, and a torque command wrap around. The program data flow consists of reading the power and torque commands, propagating a three dimensional state vector consisting of the wheel position, rate and a time dependent frictional term, and then computing the tachometer and power outputs for the AFC.

When the loop starts, the model reads a RAM register to determine if the AFC has commanded the wheel to power on. If the wheel is off, the power state is set to zero and the state continues to propagate. In that case, the model would simulate frictional spin down of the reaction wheel. If the wheel power state is set to one, the model performs two more reads of the RAM registers to read the torque enable command and the 2's complement torque command.

The second step of the loop is to propagate the state of the system. This is performed by first computing the derivative of the state, then performing numerical integration to determine the actual state. To determine the derivative of the state, the program first computes the total torque that will be applied to the wheel. This torque computation, shown in Figure 5.1, consists of four elements. First, there is the commanded torque. This is

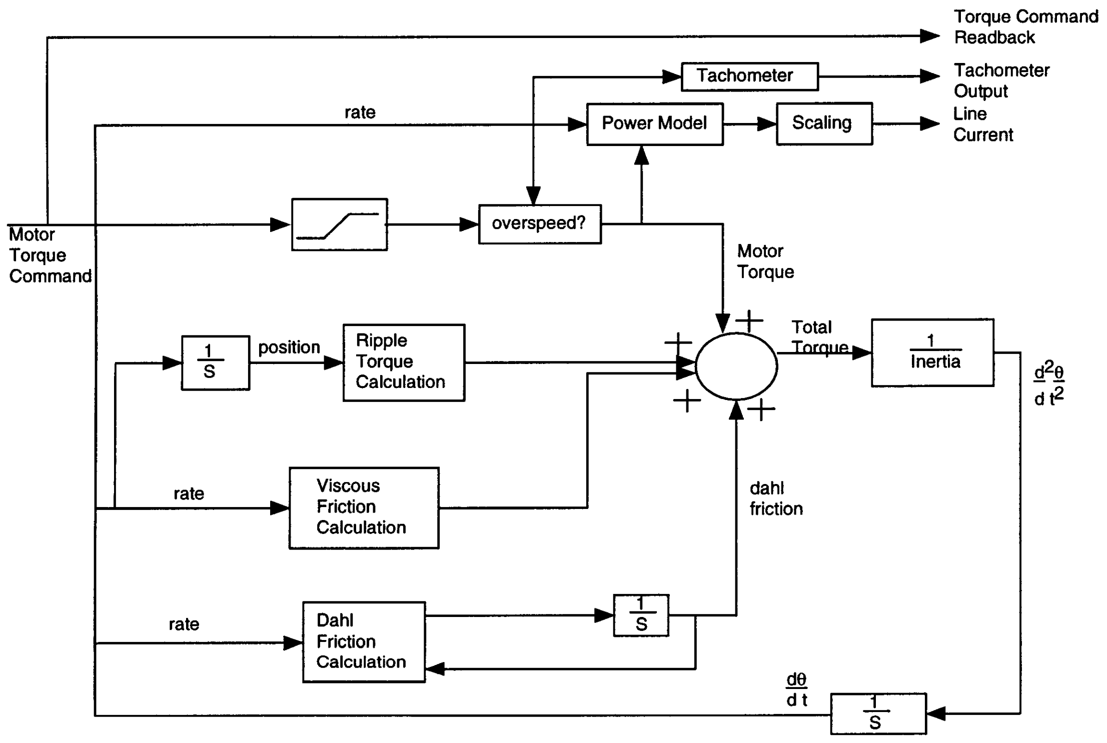


Figure 5-1 Reaction Wheel Assembly Simulator Torque Computation

converted from two's complement to an equivalent value in Newton Meter-seconds and compared against a maximum motor torque. If the command exceeds the maximum torque, the motor torque is set to the maximum. There is also an overspeed flag that is set if the wheel is spinning too fast. If it is spinning too fast, the motor torque is set to zero. The second element of the torque command is a term to account for commutation ripple. This torque is due to the brushless DC motor and is sinusoidal in nature due to the switching of the DC current windings. The third component of the torque is the viscous friction which opposes wheel velocity and is determined by multiplying the wheel rate by a constant. Finally, there is bearing friction to consider. This torque is called Dahl friction based on the bearing model developed by P.R. Dahl [3]. This model computes the time derivative of the frictional torque which is integrated to determine the torque to apply. These four components are summed to determine the torque that will be applied to the wheel.

Once the torque is known, the state derivative is calculated. This is accomplished by setting the derivative of the position to the rate and the derivative of the rate to the torque divided by the wheel inertia. The derivative of the Dahl friction term is calculated via Dahl's model. These values are then used in a 4th order Runge-Kutta numerical integration algorithm to determine the state of the system. The time step used in this routine in 0.0625 seconds.

Finally, the output is computed. This is performed by first calculating the power consumed by the wheel based on the vendor's power model and converting the power into a current. Next, the tachometer data is computed. This simulation runs every 62.5 ms, but the tachometer data is read by the flight software every 125 ms. Therefore, the tachometer output should reflect what the real tachometer would read after 125 ms. This is accomplished by multiplying the current rate of the wheel by 125 ms. This gives an estimate of the position of the wheel after 125 ms. Then, this is multiplied by a scale factor representing the quantization of the tachometer. This is then the output of the tachometer. Since the output is an integer, the fractional value computed by this calculation is saved and added to the next read. In this manner, no tachometer counts are lost. Finally, the current, tachometer counts, and the torque command are sent to the RAM registers for transmission through the RTIOU back to the flight computer.

Validation Testing and Results

The reaction wheel assembly simulator was unit tested by performing the reaction wheel simulation integration procedure in all three testing environments. This procedure contains power off and power on tests to verify electrical interfaces between the AFC and the RWA simulators. The power off section is performed with break out boxes in the loop to protect the hardware in the event of a incorrect connection. The power on tests verify that the flight software can command the wheels and that the reaction wheel simulation responds as expected. These tests were performed in both the ITL and in CATS and no significant problems were found that would inhibit testing using the simulators.

A second set of tests was also performed. This set duplicated tests that were performed on the flight equipment during stand alone testing. These tests were performed in CATS to allow for comparison of simulator response and that of the actual hardware. The results of this test are shown in Figures 5.2 and 5.3. Figure 5.4 shows the error between the two rate plots.

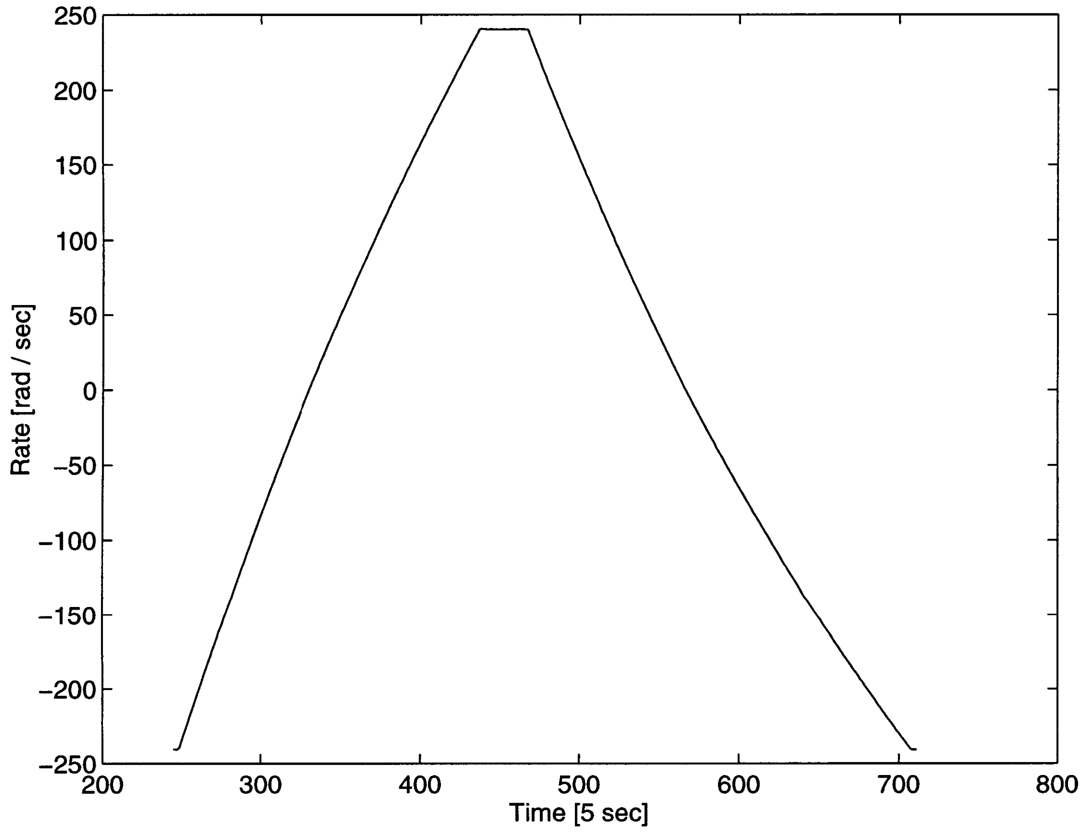


Figure 5-2 Reaction Wheel Rates During Comparison Test

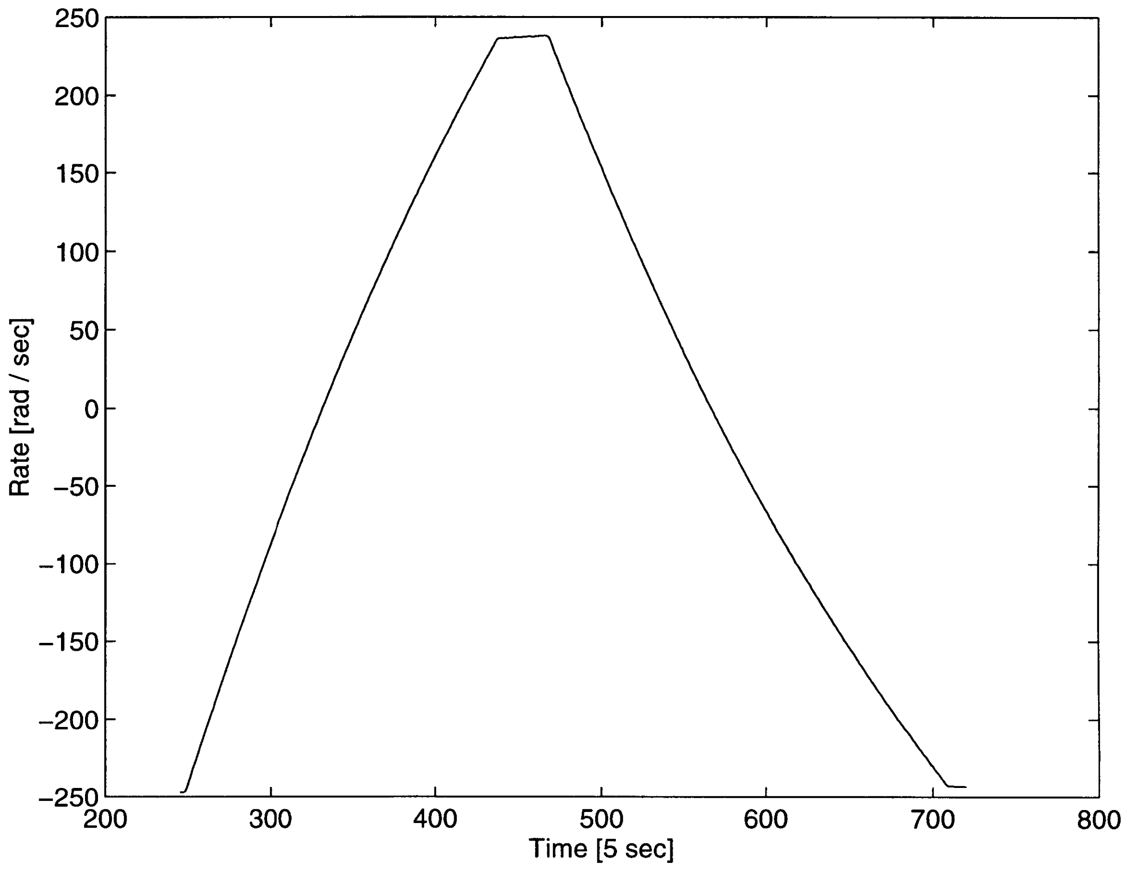


Figure 5-3 Reaction Wheel Simulator Rates During Comparison Test

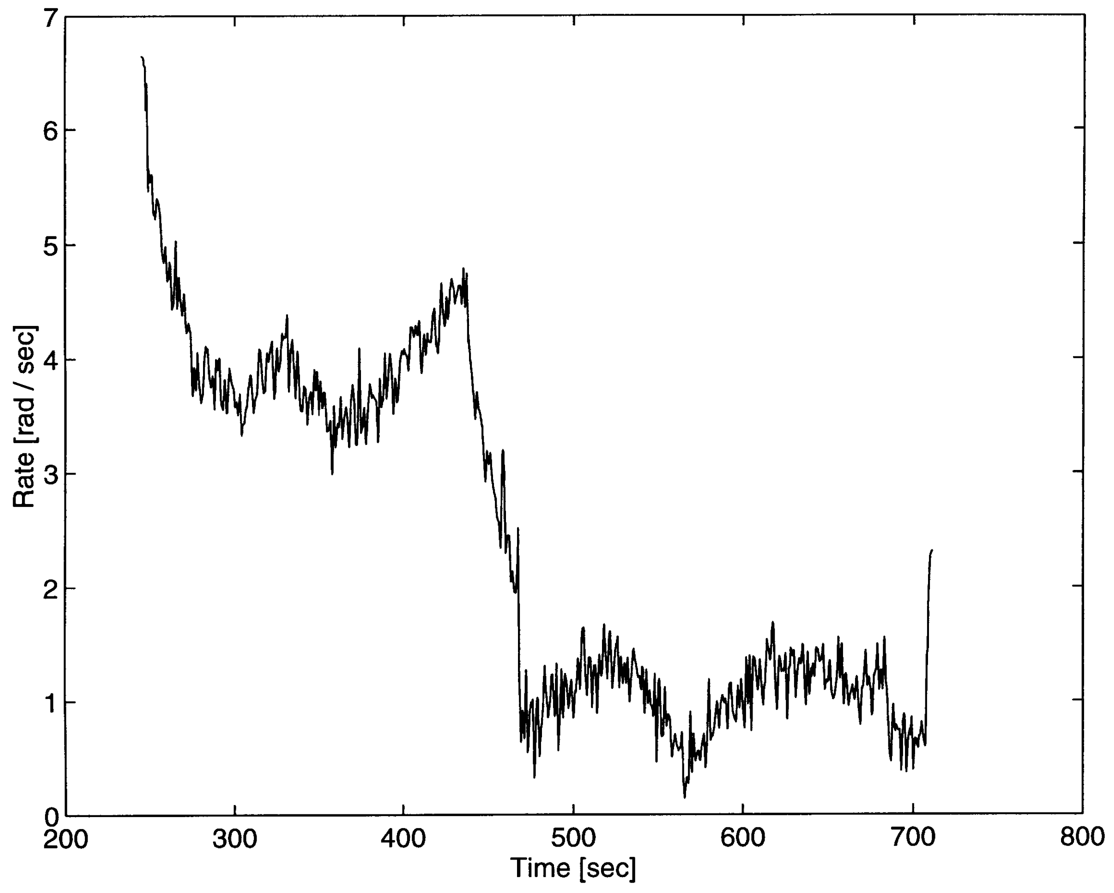


Figure 5-4 Error Between Reaction Wheel Simulator and Hardware

As is shown in the plots, particularly in the error plot, the simulator was able to track the hardware quite well. The initial error is a result of not being able to exactly reproduce the earlier test. The hardware was tested in a stand alone environment, while the simulator was tested with flight software active. As a result, the simulator test was subject to flight software constraints on allowable wheel torques and rates. However, during wheel rate changes on the order of 500 radians per second, the error was less than 10 radians per second. In addition, this error did not change appreciably when the wheels were accelerating.

There are a number of important differences between the simulation and the real reaction wheels. The first difference is the multiplexer that exists on the real wheel. This allows the flight software to receive different measurements of reaction wheel current and voltage. However, flight software is not designed to use any information other than the line current for the reaction wheels. Thus, the simulator only computes line current.

A second difference points to an important issue that was seen earlier with the IRU simulator. Neither of these simulators has any knowledge as to when the flight software is going to read the RTIOU output. This means that information must be kept current at all times at the RTIOU output. A consequence of this is seen in the way the output of the RWA was computed. The RWA routine runs at 16 Hz, but the flight software reads the RTIOU of the RWA at 8 Hz. Thus, it would be desirable to run the output routine at 8 Hz as well. The problem is that if this were attempted, the simulator and the flight software would diverge from each other resulting in incorrect output. "Old" information would be kept at the RTIOU registers for too long and incorrect information would be relayed to flight software. This is why the output routine runs at 16 Hz and integrates the tachometer output as if it were running at 8 Hz. This way, the simulator and the flight software will not diverge far enough to cause problems. In addition, the reaction wheel simulator will use the signal from the assembly simulator card indicating flight software is reading the assembly simulator card to prevent data corruption (For more information on this assembly simulator card signal, see discussion in Chapter 4, page 40).

A third difference lies in the fact that the static RAM that allows communication between the software and the remote terminal uses the same registers for reading and writing. That is, the memory location for reading information from a register is the same that is used to write information to that register. This is a problem with register 5 of the RWA. This register is simultaneously the lower byte of the tachometer data from the RWA simulator and the load mux command to the RWA. This means that there is a chance, if the timing is unfortunate, for the RTIOU to read what it believes is the lower byte of tachometer output, but is actually the old load mux control command from that last cycle. This is a rare occurrence due to the faster processing time of the RWA simulator, but if the processing takes excessively long or if flight software is quicker than usual, bad data could reach the AFC. With the real RWA, read and write registers are separate.

One of the most difficult features of the reaction wheels to simulate was the response of the reaction wheel to a reset of its remote terminal on the AACS databus. A reset of the remote terminal input/output unit (RTIOU) can occur in one of three ways: the RTIOU can be commanded to reset, reset as a result of a power on, or the remote terminal can reset due to a timeout. A timeout occurs when the RTIOU does not receive a command for 250 ms.

An RTIOU reset results in the reaction wheel setting its torque command to zero until further instructions are received from flight software. This prevents the reaction wheel from acting on an incorrect or obsolete torque command and spinning out of control. Furthermore, the reaction wheel hardware manager inside flight software expects to read a zero torque value back from the reaction wheel and will not command the wheels until this zero is received.

The problem with simulating this reset response is threefold. First, the static RAM card that allows the reaction wheel simulation software and the RTIOU to communicate does not include a channel to relay the RTIOU reset information to the reaction wheel assembly simulator. Therefore, the software has no direct knowledge of the RTIOU reset. Secondly, the static RAM uses the same memory locations as read and write registers. This means that if an RTIOU reset was detected falsely, the software could overwrite torque commands by incorrectly resetting the torque register to zero. The final difficulty is that the flight software and the reaction wheel simulator have no knowledge of each others timing. A reset can occur anytime during the simulator's processing cycle and the software must correctly reset the torque register to zero when a reset occurs.

The first attempt to solve this problem is depicted in Figure 5.5 The software nominally runs twice as fast as the flight software, so the simulator should expect a new torque command every other cycle. The first action is to read a new torque command and overwrite the RAM memory location with a zero. Then the simulation continues using the new torque command. It also copies this command to a temporary buffer. The next cycle, the software expects a zero (which it wrote), since no new update has occurred. If this is true, the software acts on the previous torque command that was buffered and propagates the state again. Finally, the register is read a third time. If the zero is still present, the software assumes that a reset has occurred, since it did not receive a new update. The simulation then acts on the zero until a new update is received, when the cycle begins again.

This method did not work well. The software overrode the good torque commands with zeros incorrectly, invaliating the simulation. This was caused by the fact that flight software timing is not exact and synchronizing the simulator and flight software was too difficult in a real time environment. This can be seen by examining the databus transactions between the reaction wheel and the flight software. The data is from testing on GMT day 96-264 in CATS.

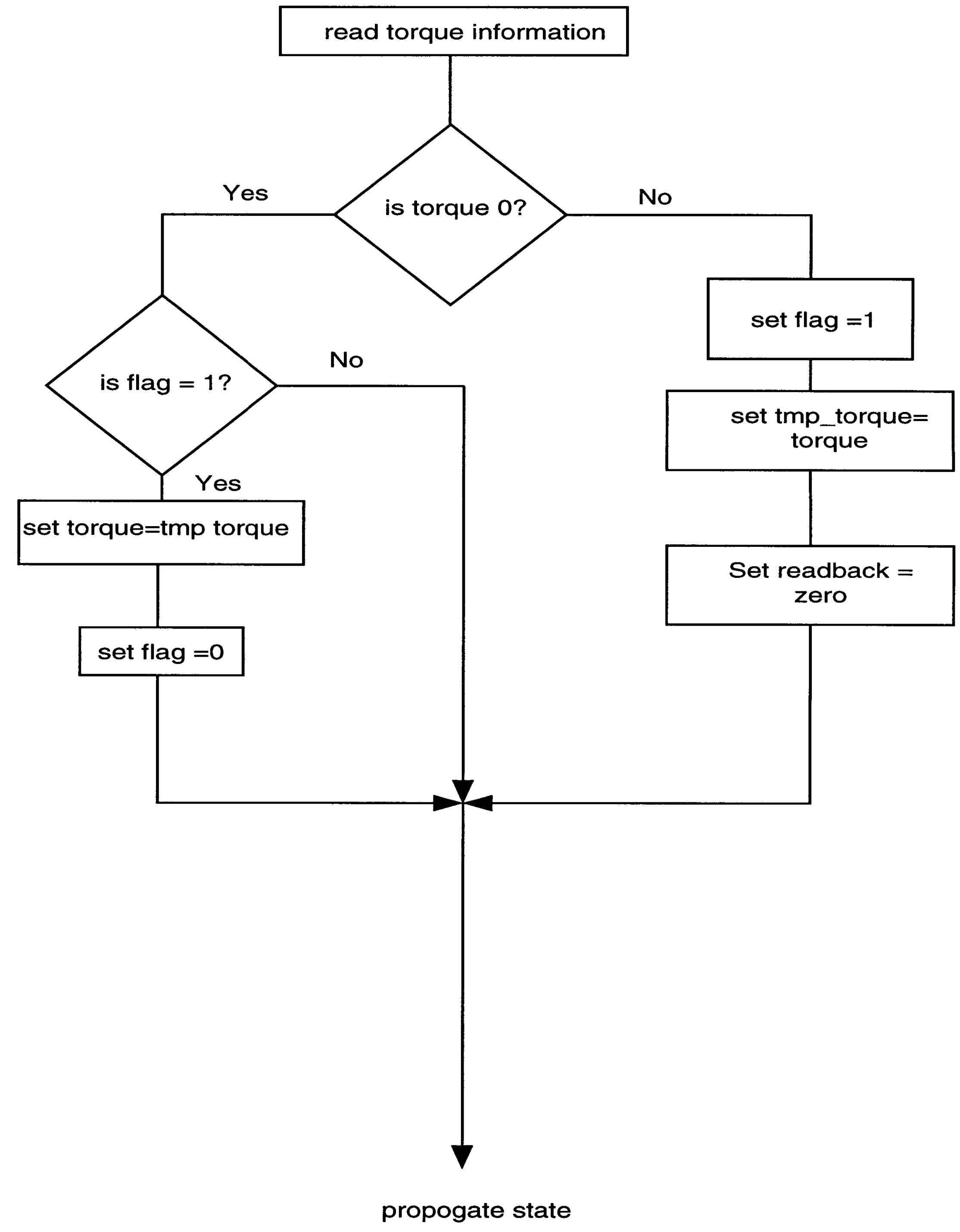


Figure 5-5 First Attempt in Solving Remote Terminal Reset Anomaly

Table 5.2: Databus Transactions During RTIOU Reset Response Testing

Time	Source	Torque Command	Reply
18:30:52.153	Flight Computer	0x65	
18:30:52.235	Reaction Wheel		0x65
18:30:52.278	Flight Computer	0x65	
18:30:52.360	Reaction Wheel		0x65
18:30:52.403	Flight Computer	0x66	
18:30:52.548	Reaction Wheel		0x00
18:30:52.590	Flight Computer	0x00	
18:30:52.673	Reaction Wheel		0x00
18:30:52.715	Flight Computer	0x66	
18:30:52.860	Reaction Wheel		0x00
18:30:52.903	Flight Computer	0x00	
18:30:52.985	Reaction Wheel		0x00
18:30:53.028	Flight Computer	0x65	
18:30:53.110	Reaction Wheel		0x65
18:30:53.153	Flight Computer	0x66	

The first packet writes a 0x65 to the RWA torque register. This corresponds to a torque command of about 0.14 Nm). This is acted upon and the RWA reports back a 0x65. This happens again with no incident. But at 18:30:52.403, the flight software commands a torque of 0x66, but the RWA is delayed in responding until 18:30:52.548. Since the flight software commands a reading from the reaction wheels about 4 ms before the data from the reaction wheel appears, this corresponds to a delay between flight software commanding a torque and reading the reaction wheel of over 140 ms. This delay results in the simulator reporting a zero as a read back, since the software believed an RTIOU reset had occurred. Flight software responds by requesting a zero and verifying it reads back a zero. This happens at 18:30:52.673. Flight software then commands a torque of 0x66 to resume processing. But again a delay occurs, returning the commanded torque to zero. The simulator/flight software combination break this pattern at 18:30:53.028, resuming nominal operation.

Based on this information, it was deemed essential to support as many as three simulation iterations between flight software commands as well as to minimize the writing of zeros to the torque command. A new method was devised to met these additional requirements.

This new method is shown in Figure 5.6. This method uses the torque enable flag, which is set to zero by the flight software each time a torque command is sent. The torque enable command is monitored at the beginning of each iteration. If the torque enable command indicates new data has been received, the simulation accepts this data and resets the torque enable command. The simulation then acts on this new data until 3 cycles without an update are complete. Then the simulation software sets the torque command to zero and acts on it as well. Thus the torque is set to zero only if no updates are received for three cycles. This method was much more successful, as shown in Figure 5.7. The reaction wheel rates closely follow the ideal rates computed by flight software.

This modeling revealed an important consideration in real time simulation. It is critical that the simulated wheels have the same access to information that the hardware does. The reaction wheel simulator does not have a reset line, nor does it have separate input and output registers. This lack of information made the RTIOU reset response a much more difficult item to model. This is an area for improvement for Cassini's reaction wheel simulator. The presence of a torque enable command was fortunate, allowing the software to model the reset response, but the software response is different from the real hardware because the software and the hardware have different information available to them.

A final concern is the impact on fault injection. It has been decided that no fault injection would be conducted in the ITL or CATS with the RWA simulators. The same decision that impacted the fault injection of the inertial reference unit led to a similar decision with the reaction wheels. Since no real command path exists to bias the tachometer of a real wheel, no attempt will be made to do the same with the simulator. In addition, the timing issues discussed above make it impossible to guarantee that a particular flight software command will generate a particular response, so injecting faults (such as an incorrect torque readback during a flight software cycle) is not possible.

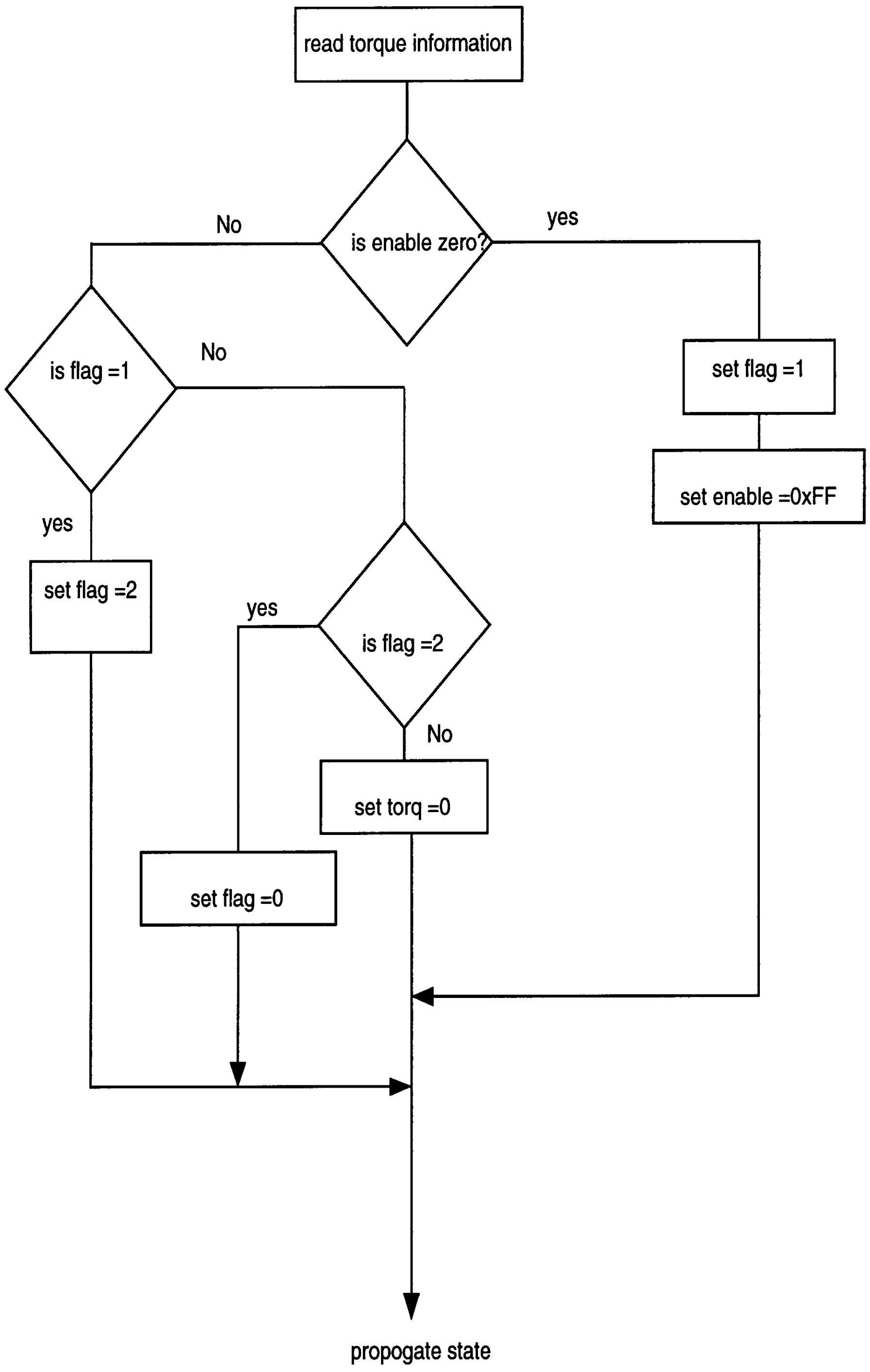


Figure 5-6 Second Attempt to Solve Remote Terminal Reset Anomaly

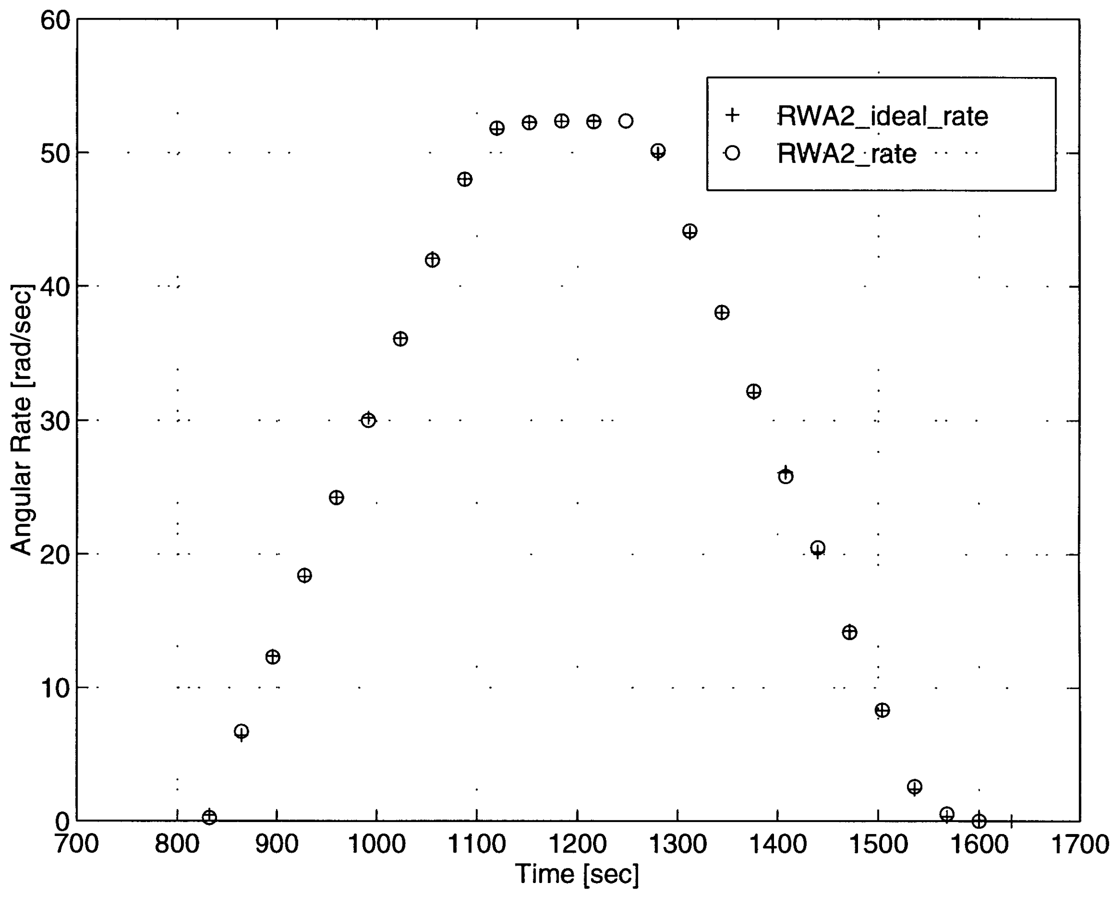


Figure 5-7 Successful Restest of Remote Terminal Reset Anomaly

5.2.2 RWA Dynamics Model

Description

The reaction wheel dynamics model consists of a C program that is designed to simulate the reaction wheels in order to generate torques to apply to the spacecraft dynamics model. The model receives three pieces of information from the assembly simulator described in Section 5.2.1 above: the commanded torque, which is read back from the assembly simulator; the tachometer reading from the assembly simulator; and the time of the torque command read back. This information is used to drive the dynamics model as shown in Figure 5.8. The first step is a call to DARTS to compute the angular rate of the modeled wheels. Once the rates are known, the model computes a rate error. First, the time since the last calculation is computed from the timing information. Second, a rate sample is calculated using the tachometer output and the elapsed time since the last calculation. This rate sample is put through a low pass filter that combines the rate sample with a previous rate estimate to obtain a new rate estimate. The purpose of this step is to limit the amount the rate estimate can jump in any one iteration. This rate estimate is then used with the rates of the DARTS wheels to obtain the rate error.

Once the rate error is known, the model computes the torque to be applied to the simulated wheels inside DARTS. First, a correction torque is computed. This correction torque is a function of an estimate of the drag torque and the rate error. The rate error is also used to compute the next value of the drag torque estimate. Finally, the torque applied is computed by subtracting the correction torque from the commanded torque.

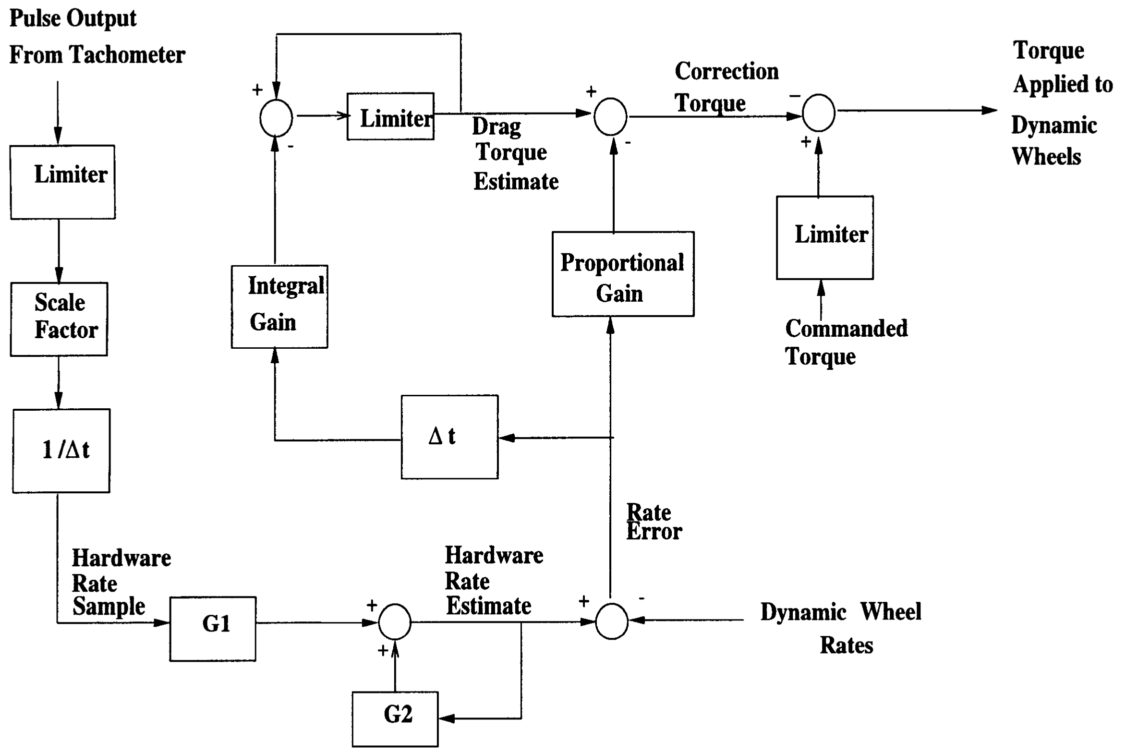


Figure 5-8 Reaction Wheel Dynamics Model Controller Block Diagram

Validation Testing and Results

To validate the model, conservation of angular momentum was invoked to develop predictions for the simulated spacecraft rates as follows:

$$H = I\omega \tag{5.1}$$

$$I_X\omega_X = I_{rw}\omega_{rw}l_X \tag{5.2}$$

$$\omega_X = \frac{I_{rw}\omega_{rw}l_X}{I_X} \tag{5.3}$$

Where

I_X = Principle Moment of Inertia, X axis, Spacecraft

ω_X = Angular Velocity about X axis, Spacecraft

I_{RW} = Moment of Inertia, Reaction Wheel

ω_{RW} = Angular Velocity, Reaction Wheel

l_X = Direction Cosine transform from reaction wheel to Spacecraft

Therefore, the effect of a reaction wheel's velocity on the spacecraft could be predicted, knowing the moments of inertia of the spacecraft and reaction wheel, as well as the transform from reaction wheel to spacecraft. The results of the testing was that after several coding errors were discovered and fixed, the model did match analytical predictions to within 10^{-4} radians/second. Model validation uncovered one particular feature of the real time simulation that almost prevented the dynamical wheels from functioning before it was fixed. The problem was with the corruption of data coming to the simulator. The simulator assumes that the data it receives are valid and relies on this information to track the reaction wheels on the spacecraft side of the simulation (real or simulator). This means, for example, that if the timetags used to determine the new speed of the spacecraft wheels are corrupted and the change in time is falsely computed to be extremely small, the model will compute an unusually large rate estimate. In response, the model will attempt to torque the dynamics wheel to match this incorrect rate estimate. This resulted in several problems that had to be corrected. For example, Figures 5.9 and 5.10 show data from GMT day 96-233 and 96-235 where the reaction wheels spun out of control and later recovered. Examination of the model software revealed that this second order behavior would be possible if the software received a small change in time estimate (used in the computation of the rate sample and the drag torque estimate). This would result in a torque sent to the dynamics wheels that would be unrealistically large based on the unrealistic speed and drag torque computed, causing the rates would jump as shown. When the next update was reasonable, however, the rates would decrease and the wheels would gradually be brought under control. The response is second order, due to the proportional plus integral controller used in the dynamics model. To correct this, a limiter was added to the drag torque estimate (shown in Figure 5-8). Filters were also added to the incoming tachometer and torque data, as well as software to limit the change in time estimate to within 25 ms of the expected value. Initially, the necessity of the limiter on the change in time estimate was not discovered and when the time was not limited (and only the drag torque, commanded torque and tachometer filters were activated), the error became a first order response, shown in Figure 5.11. This is because the drag torque filter limited the integral controller, but not the proportional controller. The proportional controller could still be in error if a large, incorrect rate sample was computed. Once the change in time computation was limited, the model did behave as expected with no anomalous error responses.

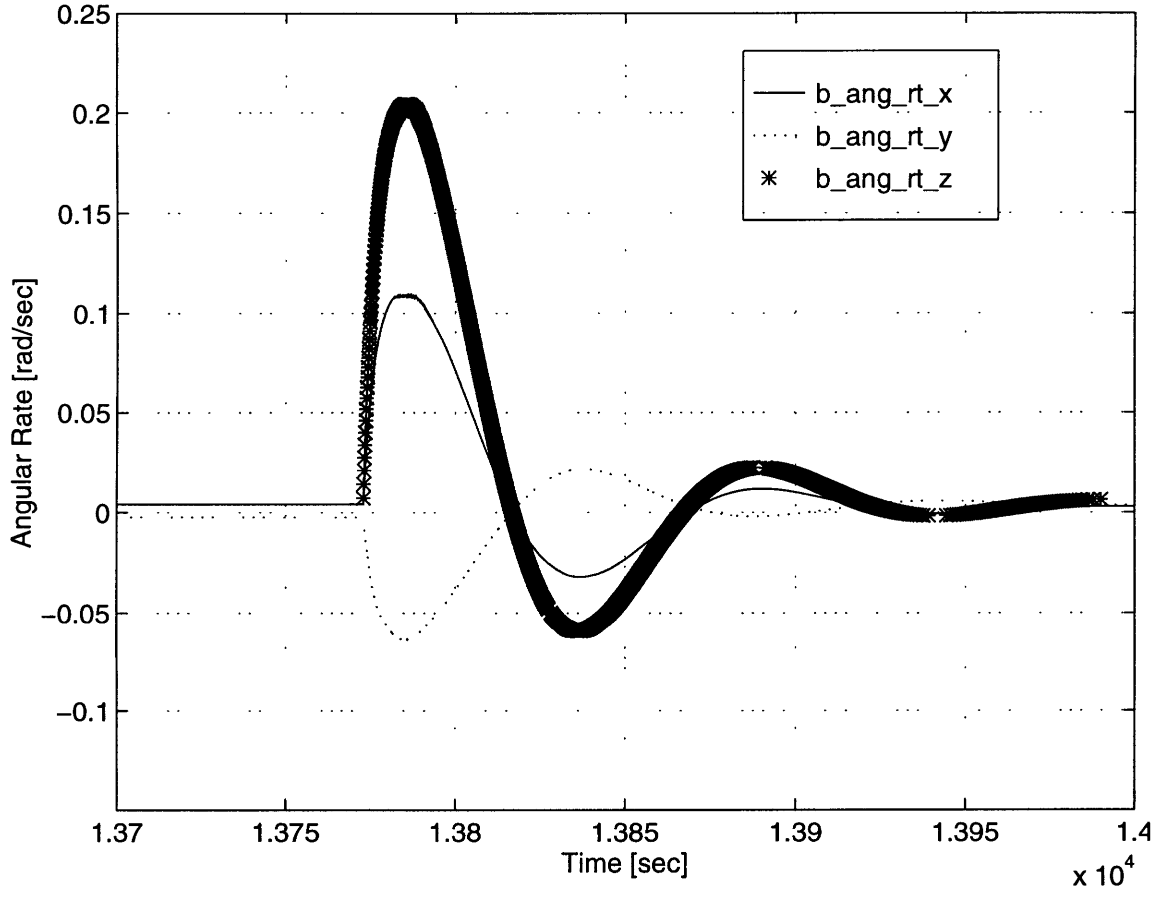


Figure 5-9 Reaction Wheel Dynamics Model Anomaly- 96-233

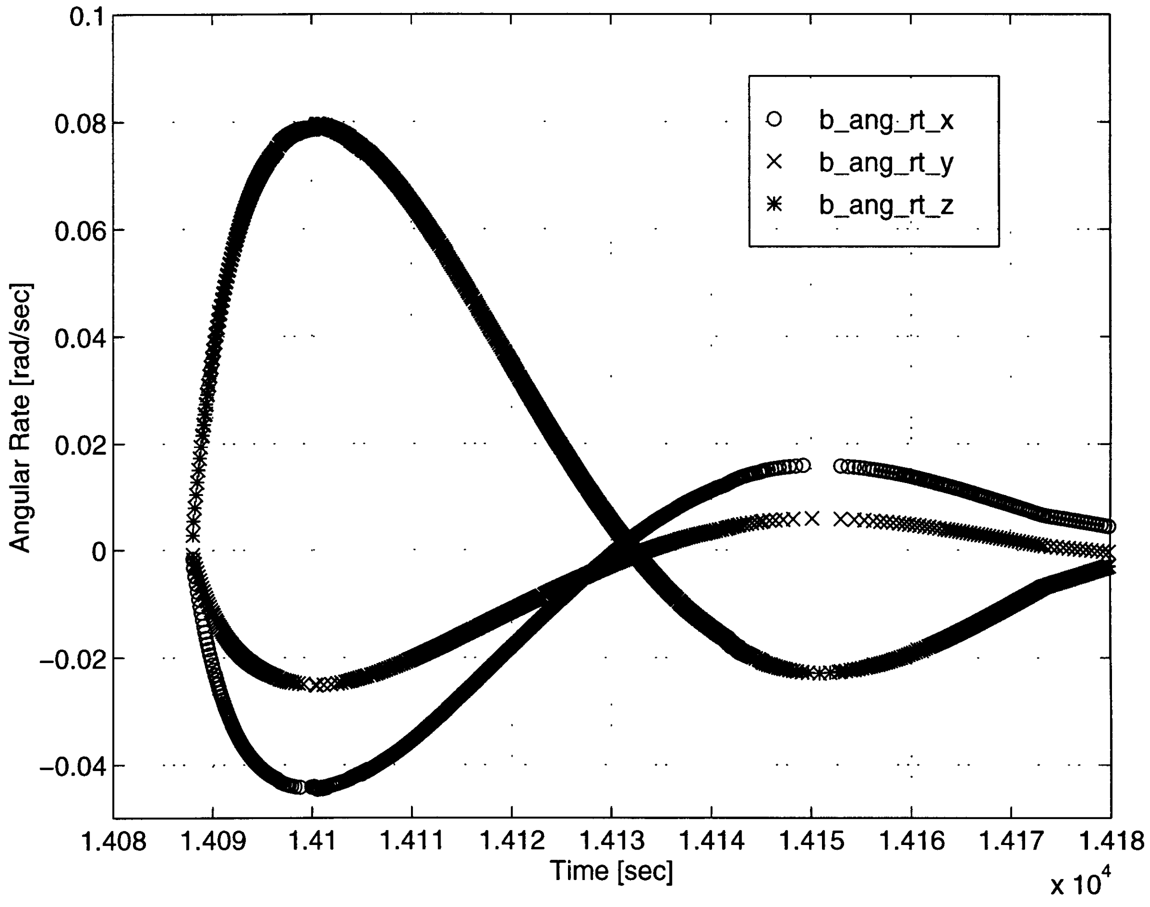


Figure 5-10 Reaction Wheel Dynamics Model Anomaly- 96-235

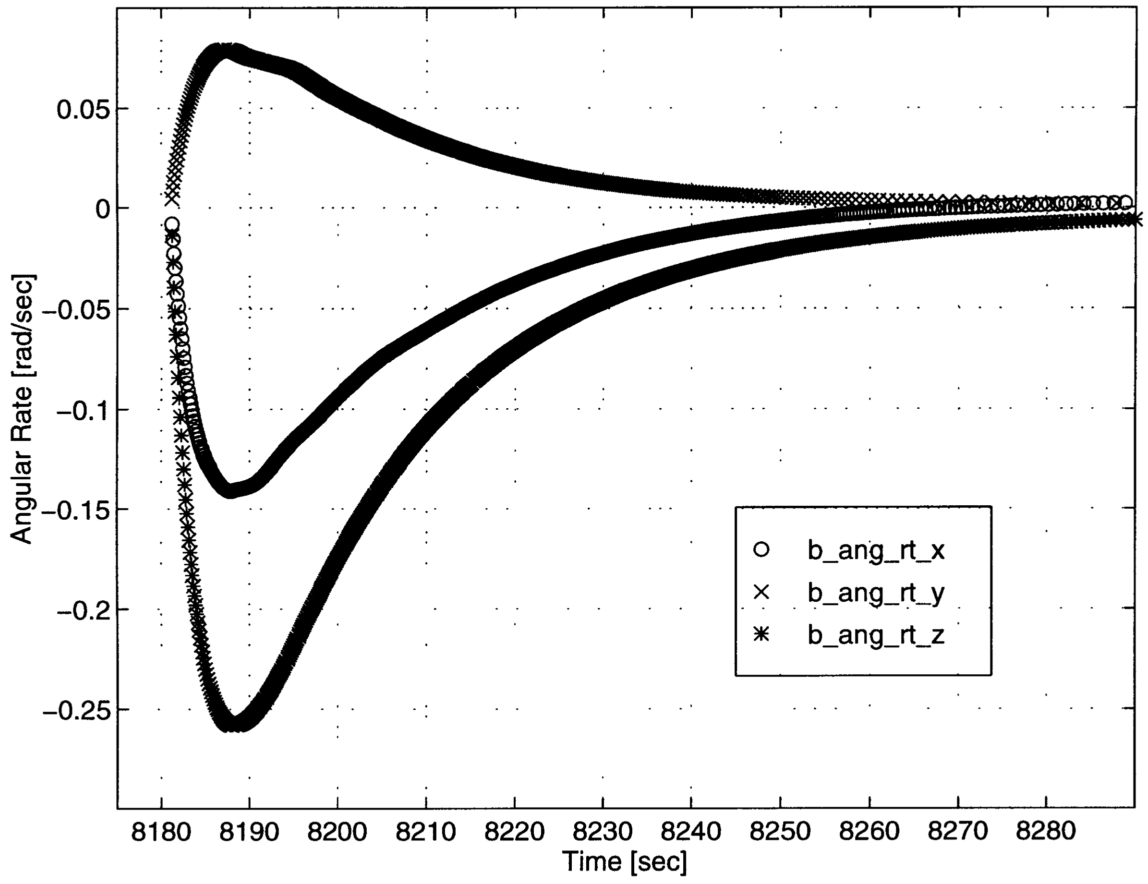


Figure 5-11 Reaction Wheel Dynamics Model First Order Anomaly

5.2.3 Closed-loop Simulation

A closed-loop flow diagram is shown in Figure 5.12. This configuration was used to

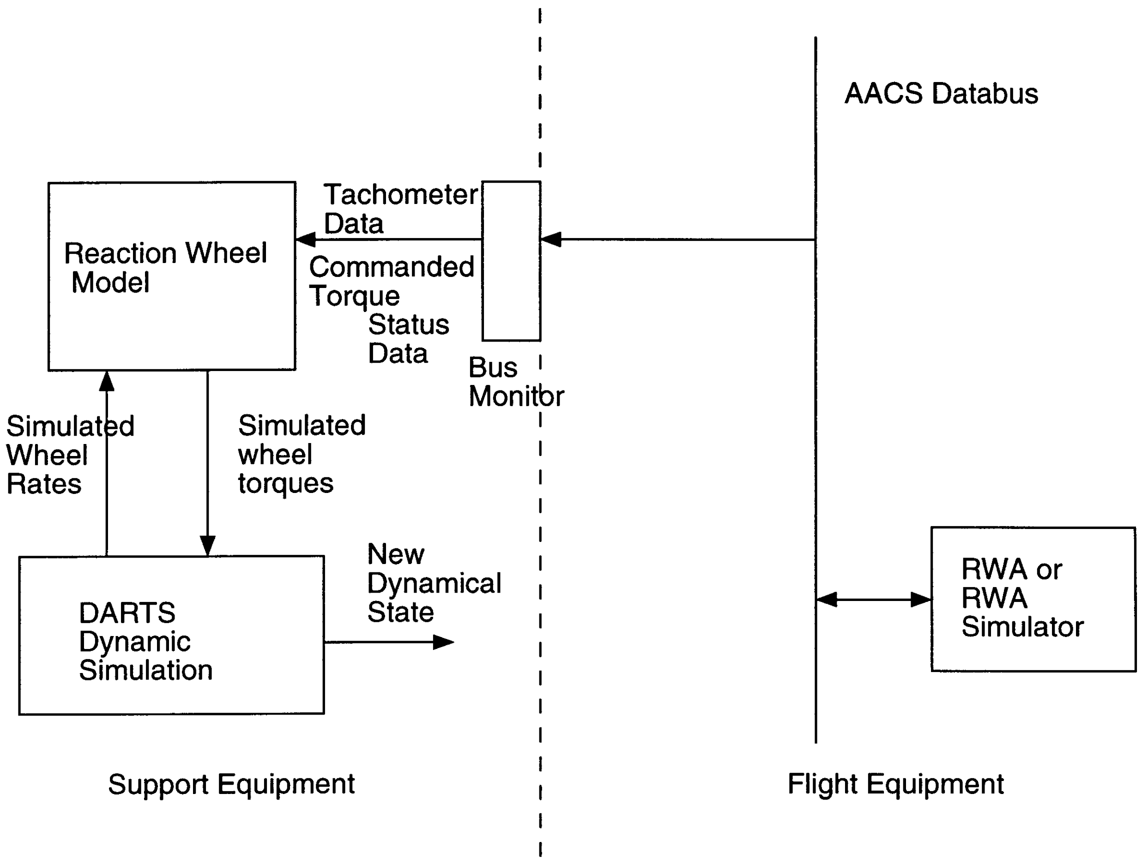


Figure 5-12 Reaction Wheel Simulation Closed-Loop Flow Diagram

test the AACS during the operational modes (OPM) sequence testing. The OPM sequences test the reaction wheels during a series of precision pointing maneuvers simulating science data gathering operations when Cassini performs its orbital tour. The desired profile of the spacecraft angular rate is shown in Figure 5.13. The data was generated during flight software testing on the Flight Software Development System (FSDS). Figure 5.14 shows the results from the testing in CATS on GMT day 96-237. The spacecraft is under reaction wheel control for the second part of the sequence. This section can be identified by the reduction in noise of the rate plots. The CATS test matches very closely with the FSDS test, verifying the closed-loop performance of the RWA simulator and the dynamics model.

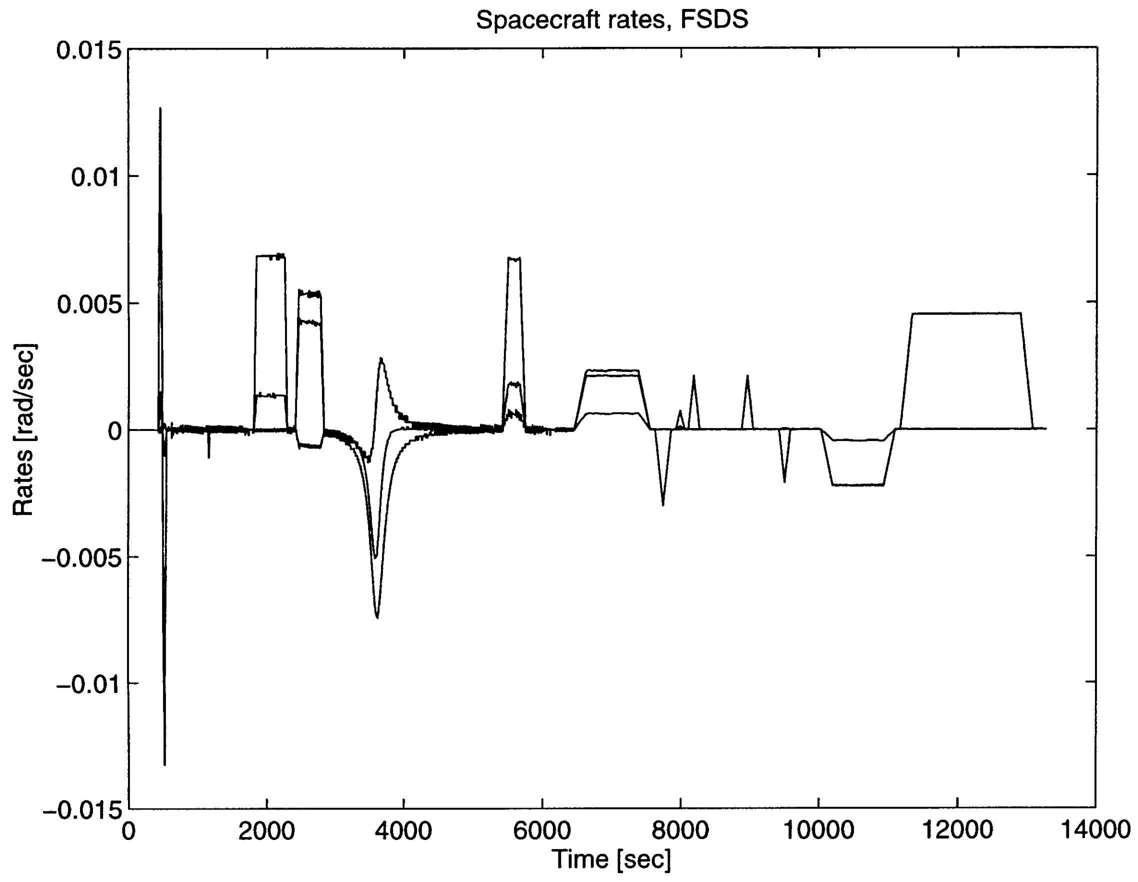


Figure 5-13 RWA OPM Testing- Flight Software Development System

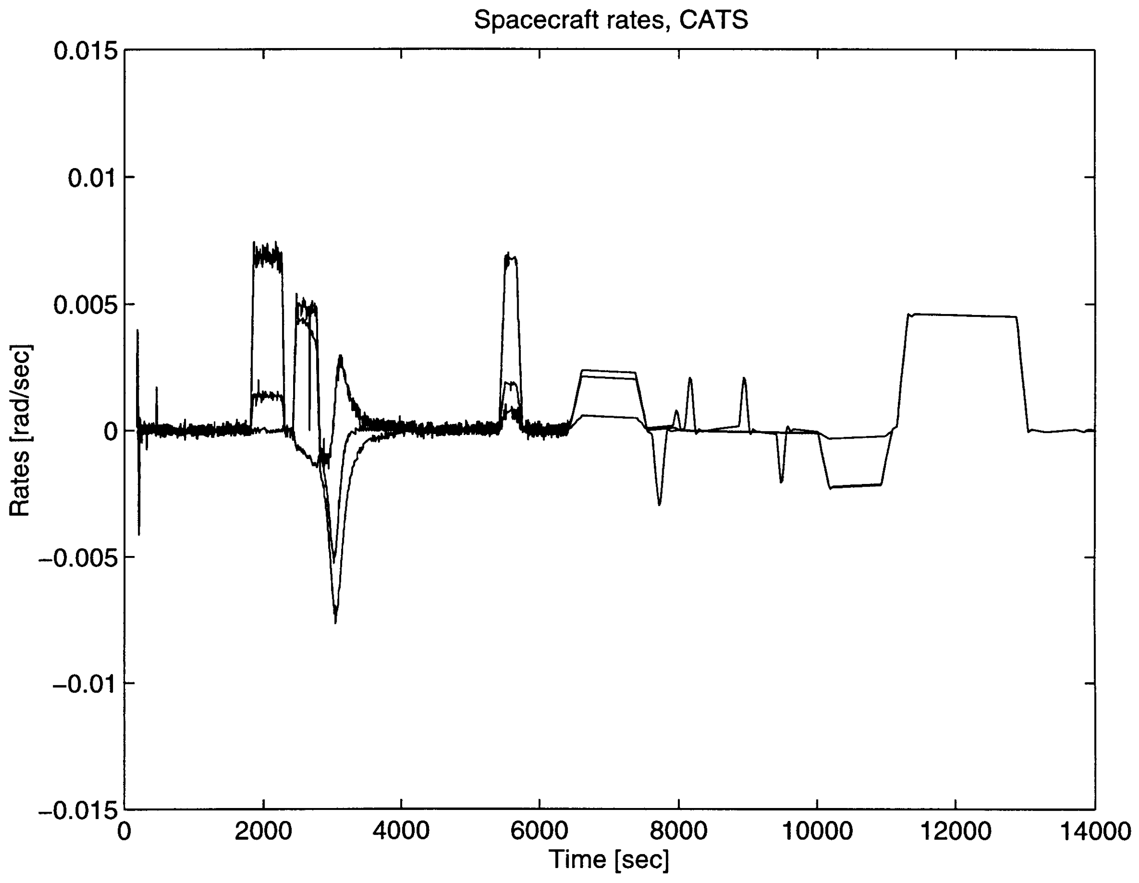


Figure 5-14 RWA OPM Testing- CATS

An important lesson that was learned during the RWA closed-loop testing is the value of simulators when software is being tested. As the complexity of software grows, the chance that something will operate incorrectly grows as well. This is evident from several flight software tests when the software reaction wheel manager was acting incorrectly, sending incorrect torque commands to the reaction wheels. If these commands were sent to the real hardware, the hardware may have been damaged. Thus, the simulators played a critical role in closed-loop software testing as well.

5.3 Evaluation

The reaction wheel simulator is the most complicated software simulator due to its internal dynamics and the importance of timing associated with it. Both functionally and in performance, the RWA simulator closely matched the real hardware. The laboratories relied heavily on the simulators during testing and they performed their functions well. Validation activities for the reaction wheels revealed the problem of data corruption within the simulation. This problem was surmounted by implementing filters on incoming reaction wheel data. The fact that the flight and support equipment software do not talk to one another resulted in a difficult implementation of the reaction wheel RTIOU reset response as well. However, even with these problems, the RWA simulator permitted effective testing of the Attitude and Articulation Control subsystem.

Chapter 6

Conclusion

This thesis has investigated the use of simulators during the testing of Cassini's Attitude and Articulation Control Subsystem. The use of these simulators has improved testing of the subsystem in all three laboratories at JPL.

The Engine Gimbal Actuator (EGA) simulator is a hardware simulator designed to correctly generate a LVDT feedback signal in response to the flight computer commands. The simulators performed as expected during integration and testing at the subsystem level, permitting flight software testing of main engine firings without requiring actual actuators. The only failure of the EGA simulation test plan occurred when subsystem fault protection powered down the actual EGA hardware during integration of the Propulsion Module Subsystem on the spacecraft. This occurred because the EGA simulators were never designed to simulate an EGA loaded with a main engine, an oversight that led to the fault protection event during assembly. As a result of this occurrence, engine gimbal actuator spares attached to a load fixture will be used in the laboratory for post launch testing activities.

The Inertial Reference Unit (IRU) simulator is primarily a software simulator to take the place of the unit for software testing. This simulator performed well, permitting a vast amount of software testing to be accomplished. Problems encountered during testing were related to timing of the simulation. The software that represented the inertial reference unit during testing does not have the same interface with the rest of the databus that the real hardware does. As a result, additional time and effort were spent trying to circumvent a problem that would never exist with the actual hardware. Once these problems were corrected, the simulator was successful in replacing the flight hardware during closed-loop testing.

The final assembly studied was the Reaction Wheel Assembly (RWA). This simulator was the most complicated of the three simulators studied since the simulator had to maintain and propagate the dynamics of the reaction wheels. This was accomplished by propagating a three element state vector (wheel position, wheel velocity, and a time dependent frictional term) using a fourth order Runge Kutta integration routine. The simulator represented all four Cassini reaction wheels for the majority of subsystem testing. The method that the simulator interfaced with the rest of the subsystem was very similar to the IRU simulator in structure leading to the same problems with timing that the IRU simulator had. For the reaction wheels, this manifested itself as the inability of the simulator to correctly respond to a commanded reset of the remote terminal on the databus. A substantial amount of debugging lead to a solution that makes the simulator respond internally quite different from the hardware. Externally, however, the response to a reset of the remote terminal is identical. The reaction wheel simulator performed well during testing of the software, successfully in replacing the flight hardware.

There were also several lessons that have been learned during the testing period. The first lesson is the importance of clear requirements and objectives in simulation. For example, the fact that the engine gimbal actuator simulators were never meant to simulate a loaded EGA led to the activation of fault protection during Assembly, Test and Launch Operations. It was the misunderstanding of this fact that led to the fault protection event. An example of clear requirement understanding is the Inertial Reference Unit (IRU) processor simulator. It was clear from the beginning that the support equipment software would not simulate the actual built in test or software download. Since this was specified, testing of these functions was handled elsewhere. This clear expectation led to effective testing.

A second lesson is that there is a trade-off when deciding on additional command paths in simulators. The Cassini project decided not to add any additional command paths. The advantage of this is that the simulators act just like the hardware in terms of how it interfaces with the support or flight equipment. The disadvantage is that the ability of the tester to inject faults into the system is compromised. The simulators could not be forced to report false information. For example, the only way an IRU simulator could report bad data is if the dynamics simulation reported incorrect data to the IRU. Thus the designer of a simulator must decide between increased ability to inject faults and a simulator that is more like the actual flight equipment.

A third lesson is more of a warning as to the dangers of developing simulators with the sole objective of matching the flight hardware. At JPL during Cassini's testing, the official requirement for the simulators was to conform to the specifications of the Attitude and Articulation Control Subsystem Interface Control Document (ICD) [6]. However, in the absence of clear guidelines in the ICD, the simulators were designed to conform to the hardware. This invites problems due to the lack of an independent check of the component. If the simulator was designed to match incorrect hardware, then problems with the hardware may never be caught. There were many instances where the simulation designers did not have accurate or up to date information in the ICD from which to develop simulators. The requirements from the ICD were also constantly changing. This situation underscored the importance of requirement statement and verification with both the hardware and the simulators.

A final lesson that applies not only to these simulators but to the laboratory environment as a whole is that in real time simulation, timing is everything. This was demonstrated with the Reaction Wheel Assembly(RWA) remote terminal input output unit (RTIOU) reset response simulation as well as the data transfers between the aspects of the simulators. Loss of data between the RWA simulator and model caused testing failures and the biggest problem with the support equipment software as a whole was data loss and corruption. Real time simulation is a great asset and an indispensable tool, but care must be taken to preserve data integrity to ensure consistent and reliable testing of subsystem hardware and software.

References

[1] Basilio, Ralph D. Flight System Testbed Verification of the Mars Pathfinder Attitude Control System. AAS Paper 96-011. in Guidance and Control 1996- Proceedings of the Annual Rocky Mountain Guidance and Control Conference. Advances in the Astronautical Sciences, vol. 92.

[2] Culp, Robert D. and Martin L. Odefey, eds. Guidance and Control 1996- Proceedings of the Annual Rocky Mountain Guidance and Control Conference. Advances in the Astronautical Sciences, vol. 92. San Diego: Univelt, Inc., 1996.

[3] Dahl, P. R. Measurement of Solid State Friction Parameters of Ball Bearings. NASA Document ID 77N33528. 10 March 1977.

[4] Graves, Rick D. Cassini AACS Support Equipment Hardware Design Requirements and Description Document. JPL EM 343-1318, Rev. D. 15 September 1996.

[5] Jain, Abhinandan and Guy K. Man. Real-time dynamics simulation of the Cassini spacecraft using DARTS. Part 1; Functional capabilities and the spatial algebra algorithm. Proceedings of the Fifth NASA(NSF) DOD Workshop on Aerospace Computational Control. 15 February 1993.

[6] Montanez, Leticia M., custodian. AACS Cassini Support Equipment Software Design Requirements and Description Document. JPL IOM 3413-96-122. 12 April 1996.

[7] Rittmuller, Philip A. ITL Testing of the Engine Gimbal Actuators. JPL IOM 3410-95-224 CAS. 15 June 1995.

[8] Walker, W. John, ed. Cassini AACS Interface Control Document. JPL Document D-12463. Cassini Project Document PD 699-113. Issue #2, 16 June 1995.

[9] Wong, Dr. Edward C., ed. Project Cassini Control Analysis Book. JPL Document D-9638. Cassini Project Document PD 699-410. Update #2, 31 January 1994.

[10] Zimmelman, Darrel, et al. The Attitude Control System Test-Bed for SWAS and Future SMEX Missions. AAS Paper 96-013. in Guidance and Control 1996- Proceedings of the Annual Rocky Mountain Guidance and Control Conference. Advances in the Astronautical Sciences, vol. 92.