An Integrated Evolutionary Model Approach to Small Satellite Engineering

by

Joseph B. Robinson

B.S., United States Air Force Academy (2008)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .....
Department of Aeronautics and Astronautics
May 14, 2010

Certified by .....
Col John Keesee, USAF, Ret.
Senior Lecturer of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Dave Miller
Professor of Aeronautics and Astronautics
Thesis Supervisor

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

An Integrated Evolutionary Model Approach to Small Satellite Engineering

by

Joseph B. Robinson

Submitted to the Department of Aeronautics and Astronautics
on May 14, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

A deficiency exists in the use of detailed integrated modeling in the design, fabrication, and operations of small satellites (<180kg). This need led to the design of the Integrated Evolutionary Model (IEM) approach to small satellite engineering. The IEM approach uses integrated modeling throughout the satellite design life cycle to support the verification and validation of the satellite capabilities and limitations. This thesis describes the IEM approach and introduces its products: a detailed command-based integrated simulation (SIM), telemetry and data analyzer (TDA), and a mission assurance tool (MAT). Recently developed integrated systems models for the MOTV and CASTOR satellites are described and critiqued against the IEM approach, and a new simulation framework is developed for CASTOR utilizing the IEM framework. The development and use of a TDA for the FalconSAT-3 satellite is also described. The thesis presents several FalconSAT-3 operational events where the TDA was paramount in identifying and characterizing unexpected on-orbit behaviors during spacecraft and experiment operations. The utility of the IEM products is demonstrated through the successful use of the FalconSAT-3 TDA in these events.

Thesis Supervisor: Col John Keesee, USAF, Ret.
Title: Senior Lecturer of Aeronautics and Astronautics

Thesis Supervisor: Dave Miller
Title: Professor of Aeronautics and Astronautics

Acknowledgments

This thesis could not have been written without the advice and support of many individuals. Col (Ret.) John Keesee advised and supported me through the entire process, and allowed me the flexibility to focus on topics that I had interest in. Professor David Miller was vital to pushing the use of integrated modeling and planting a seed in my mind for further research. The faculty and professors from all the interesting classes I have taken here were essential in forming some of my conclusions and I am humbled to have learned from the best.

I would also like to thank the Air Force Space and Missile Systems Center for their support of the SEA program at MIT. Without their funding, none of this work would have been possible. Col David Swanson's initial feedback of my research thoughts was vital in pushing more towards what I believe is a much more useful product. Thanks to Dr. Kyle Yang for being a very effective liaison between SSL and SMC.

A large debt is owed to the USAFA Department of Astronautics and the FalconSAT program personnel for their continued support for the five years that I have worked with them. While I would have slept much more if the FalconSAT-3 satellite would have never been built or launched, I would not have the experience and knowledge that I now have. Thanks to Col Tim Lawrence and Professor Bill Saylor for keeping the program on track and allowing me a great amount of flexibility with FalconSAT-3. Thanks to the FalconSAT-3 operations team, particularly Cadet Ben Shoptaugh, for performing the tests and supplying the data that I often desired. Lastly, thanks to Col (Ret.) Jack Anthony, who always provided colorful emails and was a supportive of all that I did.

Thanks to all of the SSL students and staff. It is great to work with such a knowledgeable group of people that can quickly tell me when I'm going down the wrong path or find solutions to problems that I faced. Jake Katz and Jaime Ramirez were vital in the support of all FS-3 analysis that was done at MIT. Good luck to the SEA class of 2011 in George Sondecker, Corey Crowell, and Matt McCormack, and thank you for doing the day to day tasks in the last semester so I could focus on writing. To John Richmond, for pushing me the whole way and making sure I felt the pressure.

And last but certainly not least, I'd like to thank my family. Without them I know that I would not have had the opportunities that I have had. Mom and Dad, I greatly appreciate your unwavering support and pride in all that I do.

Contents

List of Figures

List of Tables

List of Acronyms

ADCS	Attitude Determination and Control System (aka ACS)
AF	Air Force
ATB	Avionics Test Bed
BCR	Battery Charge Regulator
BSEKF	Backwards-Smoothing Extended Kalman Filter
C_n	Mission Constraint Element ($1 \leq n \leq 4$)
CAD	Computer Aided Design
CASTOR	Cathode Anode Satellite Thruster for Orbital Repositioning
CCSDS	Consultative Committee for Space Data Systems
CDR	Critical Design Review
CMD	Command
CONOPS	Concept of Operations
COTS	Commercial off the Shelf
ΔV	Difference in velocity
DCM	Direction Cosine Matrix
ECI	Earth Centered Inertial
EP	Electric Propulsion
EPS	Electrical Power System
FM	Flight Model
FOM	Figure(s) of Merit
FRR	Flight Readiness Review
FS-3	FalconSAT-3
FS-5	FalconSAT-5
GEO	Geosynchronous Orbit
GNC	Guidance, Navigation, and Control System
GPS	Global Positioning System
GTO	Geotransfer Orbit
GUI	Graphical User Interface
GUIDE	GUI Development Environment
IEM	Integrated Evolutionary Model
IGRF	International Geomagnetic Reference Field
INCOSE	International Council on Systems Engineering
ISS	International Space Station

LEO	Low Earth Orbit
LimWOD	Limited Whole Orbit Data
LTAS	Low Thrust Attitude System
LVLH	Local Vertical, Local Horizontal
MA	Mission Assurance
MAT	Mission Assurance Tool
MIT	Massachusetts Institute of Technology
NORAD	North American Aerospace Defense Command
NASA	National Aeronautics and Space Administration
N^2	N-Squared
OMG	Object Management Group
OOP	Object Oriented Programming
OPCAT	Object-Process Case Tool
OPM	Object-Process Methodology
P_n	Primary Element ($1 \leq n \leq 4$)
PDR	Preliminary Design Review
PPM	Parts Per Million
OPM	Object Process Methodology
QM	Qualifications Model
s_n	Supporting Element ($1 \leq n \leq 4$)
S_n	Secondary Element ($1 \leq n \leq 4$)
SAT	Satellite
SCL	Satellite Command Level
SDR	Systems Design Review
SEA	Space Engineering Academy
SGP4	Simplified General Perturbations 4
SIM	Satellite Integrated Model
SMAD	Space Mission Analysis and Design [21]
SMC	Space and Missile Systems Center
SRR	Systems Requirement Review
SSL	Space Systems Laboratory (MIT)
SSRC	Space Systems Research Laboratory (USAFA)
SPL	Space Propulsion Laboratory (MIT)
SVN	Subversion
SysML	System Modeling Language
TDA	Telemetry Data Analyzer
TLE	Two Line Element
TLM	Telemetry
UML	Unified Modeling Language
USAFA	United States Air Force Academy
UTC	Coordinated Universal Time
WBS	Work Breakdown Structure
WOD	Whole Orbit Data

Chapter 1

Introduction

1.1 Problem Statement

Many modeling tools and techniques are available to support the development of space systems. However, modeling tools are subject to constraints and limitations and require extra resources. This resource burden causes many university-built satellites and small satellites to minimize or even forgo modeling, which can lead to design oversight and improper assumptions. If used effectively, an integrated evolutionary model approach can be used for small satellite systems to assist mission assurance efforts, enhance operational performance and flexibility, and indirectly free resources.

1.2 Motivation

Integrated modeling can assist engineers, customers, and stakeholders in all phases of the space system design sequence by communicating the design and interfaces, simulating the mission applications, and illustrating design traceability. By increasing the fidelity of the model through an evolutionary approach, these models can turn into documentation, training, and mission operations tools. There are a few specific topics that this thesis will discuss in greater depth to demonstrate the application of the integrated modeling. The motivation for these topics and a description of how the integrated evolutionary modeling concept can be applied is described in Sections 1.2.1 to 1.2.5.

1.2.1 University Satellites

The term "university satellite" can have many connotations. In this thesis, a university satellite refers to a stand-alone small satellite that is designed and built by a student team. The University Nanosat Program (UNP), an Air Force Office of Scientific Research (AFOSR) funded competition through the Air Force Research Lab (AFRL), provides excellent examples of university satellites. Every other year approximately ten universities enter UNP, with the goal of winning the two year competition and being sponsored by the Air Force for a launch. Small satellites can be referred to as a microsat, nanosat, or picosat, with the mass and the volume providing the distinction between categories. These terms also have many connotations that disagree between different sources; in this thesis, a university satellite will be assumed to have a mass less than 180 kg. Note that these definitions do not limit the application of this thesis on larger satellites or satellites that do not fit all criteria of a "university satellite;" they are merely meant to assign a name for a class of satellites that will be described and referenced throughout the thesis.

Integrated modeling can provide multiple benefits to a university satellite. These satellites and their programs are unique in that their product is not only hardware, but also personnel. A desired output for any university satellite program is knowledgeable engineers with real knowledge of a system through hands-on experience on a real project. However the student-worker environment inherently causes remarkably high turnover to the project team. If the project encompasses many semester or years, it is necessary to have in place a continuation and training program for new students. This requirement can force a great deal of documentation and extra work that will hold back students from focusing on the satellite design or operations. Integrated modeling can be used throughout the system's life cycle to quickly explain the system and its properties. This is very useful for university students, but it can also be useful for presentations or reviews to demonstrate the concept, functionality, or operational results of the satellite.

The question that will be addressed regarding integrated modeling and university satellites in this thesis is, "What is the correct role and effort that integrated modeling should have throughout the design and operational life of a university satellite?"

1.2.2 Mission Assurance

Mission assurance is defined as "the disciplined application of general systems engineering, quality, and management principles towards the goal of achieving mission success, and, toward this goal, provide confidence in its achievement" [36]. Mission assurance actions should inherently take place throughout the life cycle of a mission; however, the mission success cannot be determined until the operations phase of the mission. For this reason, an ideal end goal of an integrated evolutionary model is to produce a mission assurance tool to be used on-orbit. As each project and mission has unique parameters and metrics to measure mission success, this final mission assurance tool can come in many different forms. The thesis will explore the development of a mission assurance tool through a project currently in the final design stage.

The question that this thesis will address is, "How can integrated modeling validate mission assurance requirements without requiring additional commitment of time or manpower from the design team?"

1.2.3 Space Engineering Academy

The Space Engineering Academy (SEA) is a program currently in development by the Air Force Space and Missile Systems Center (AF SMC). The pilot program initiated in fall 2008 at the Massachusetts Institute of Technology (MIT). The mission of the SEA is to provide a comprehensive foundation for junior Air Force space officers through a hands-on satellite design, build, test, and operate experience; developing leaders with enhanced programmatic decision making skills to ensure the superiority of Air Force space systems.

The desire to create the SEA program stems from a recurring problem throughout SMC: Air Force junior officers do not have the experience to effectively manage and procure space systems. Only 10% of officers entering SMC have an aerospace background, and of those, even less have actual flight hardware program experience. SEA is meant to provide SMC and the Air Force with an accelerated way for junior officers to gain flight hardware experience that incorporates "best practices" with low programmatic risk. In the context of an actual satellite, the participants of the SEA program will conceive, design, build, integrate, test,

and operate flight hardware.

This thesis will explore the possibility of using integrated modeling in the SEA program to enhance the education of future Air Force officers and their understanding of space systems as a whole. The thesis will describe how integrated modeling is intended to be used in the SEA two-year project life cycle. The phases of this cycle as well as a proposed timeline are shown below in Figure 1-1.

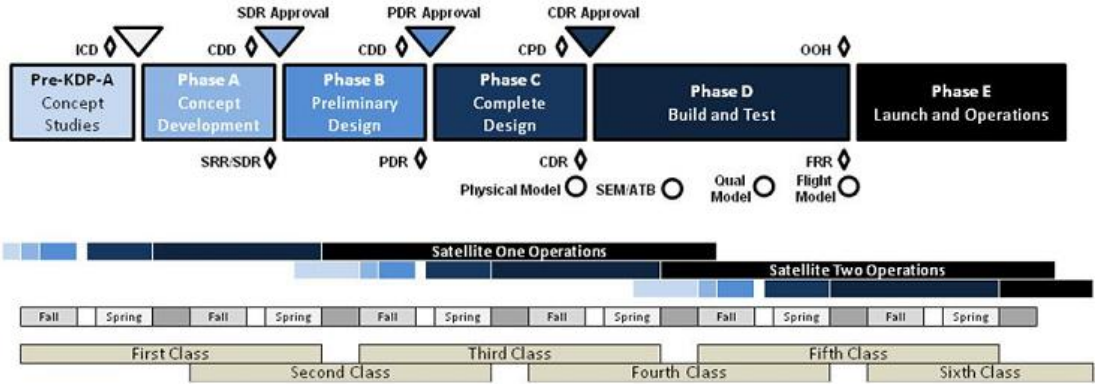


Figure 1-1: Space Engineering Academy Timeline

Therefore the question that this thesis will answer is, "How can integrated modeling be implemented into a structured satellite design and build curriculum such as SEA?"

1.2.4 CASTOR

A large source of motivation for this thesis comes from the CASTOR satellite. CASTOR, which stands for Cathode Anode Satellite Thruster for Orbital Reposition, is currently being designed and built by undergraduate and graduate students in MIT's Space Systems Laboratory (SSL). The mission of CASTOR is to validate the performance and application of Diverging Cusped Field Thruster (DCFT) technology. This mission will be achieved by taking on-orbit state data to compare the degradation experienced by the DCFT to that of similar technologies. The DCFT is a low-thrust electric propulsion (EP) system that has been built and is being finalized by the MIT Space Propulsion Laboratory (SPL).

The CASTOR mission concept is to utilize the DCFT to perform maneuvers once on-orbit and track with time how effective the thruster is at performing these maneuvers. However, this is not a trivial task for a low-power, low-thrust electric propulsion system.

To get a complete picture of the thruster performance, the following conditions must be accounted for:

- Power available to thruster
- Flow rate to thruster
- Thruster steering angle
- Attitude knowledge error
- Center of Gravity (CG) and Center of Mass (CM) offset
- Orbital perturbations and disturbances
- Duty cycle and thrust schedule

To simplify this problem, significant design and operational decisions can be made to make assumptions that will allow for easier analysis of the on-orbit data. An example is continuously firing the thruster in the velocity direction at a constant flow rate and power setting in a circular orbit with no disturbances. This maneuver is described in many books such as *An Introduction to the Mathematics and Methods of Astrodynamics* [3] by Richard Battin or *Fundamentals of Astrodynamics and Applications* [41] by David Vallado. An analytical solution exists for this maneuver, as well as others for constant thrust, accurate pointing cases. However, a design of such a system would be very costly. In a small satellite that has mass, volume, power, and budget constraints, this approach is simply not an option. It should be noted that direct measurement of the thruster's output performance is not possible, and therefore state-based trajectory analysis using position and velocity data will be the primary source of performance analysis.

If the problem cannot be simplified by a robust and predictable design, the alternative way to solve it is to utilize vehicle state data and an understanding of the space environment to predict the expected satellite behavior and compare it to the actual behavior. This must be done by sending the on-orbit telemetry of interest to the ground and running it through a tool that accounts for the different conditions to compare against the actual state. The motivation of using integrated modeling in the CASTOR program is to develop this analysis tool for on-orbit thruster characterization. Additionally, the integrated modeling will provide CASTOR operators with a model that not only can analyze the past data, but

also predict the future state data. This capability then can compare different expected and actual behavior, and potentially identify anomalous behavior prior to a large vehicle incident such as a component or system failure. This thesis will use CASTOR as an example to look at how an experimental electric thruster can be performance tested on a low-cost spacecraft with loose pointing and thrusting requirements with proper command based modeling and simulation techniques.

The question that this thesis will answer regarding the CASTOR satellite is, "How will integrated modeling be used to ensure that the CASTOR satellite will meet its mission objectives after launch?"

1.2.5 FalconSAT

FalconSAT is the satellite program at the United States Air Force Academy (USAFA). Run through USAFA's Space System's Research Center (SSRC), the FalconSAT program flies Air Force payloads on satellites designed, built, and operated by USAFA cadets, who are all undergraduate students [34]. This program started in the mid 1990's and has produced four flight satellites to date, with the fifth satellite, FalconSAT-5 (FS-5), due for launch in the Summer of 2010.

The FalconSAT program is being analyzed in this thesis due to a large amount of data available from FalconSAT-3 (FS-3), as well as the similarities between CASTOR and FS-5. Still operational, FS-3 has provided cadets, faculty, and contractors at USAFA with many lessons throughout its lifetime. Many of these lessons have come from FS-3 anomalies that the operations and engineering teams identified and resolved after launch. FS-5's design contains a propulsion system containing both cold-gas and electric propulsion units, and therefore there is a desire to analyze the trajectory of the satellite. As FS-5 will launch before CASTOR, the techniques and tools utilized for CASTOR may be able to be used on FS-5 for an additional source of verification.

The question that is being approached in this thesis is, "How can the lessons learned through FalconSAT-3's lifetime be applied to future satellite designs such as CASTOR and FalconSAT-5 so that the on-orbit anomalies can be prevented or resolved more quickly?"

1.2.6 Thesis questions

To summarize the different areas of motivation for this thesis, the following is a list of the main topics and questions that this thesis will attempt to answer.

- **University Satellites** — What is the correct role and effort that integrated modeling should have throughout the design and operational life of a university satellite?
- **Mission Assurance** — How can integrated modeling validate mission assurance requirements without requiring additional commitment of time or manpower from the design team?
- **Space Engineering Academy** — How can integrated modeling be implemented into a structured satellite design and build curriculum such as SEA?
- **CASTOR** — How will integrated modeling be used to ensure that the CASTOR satellite will meet its mission objectives after launch?
- **FalconSAT** — How can the lessons learned through FalconSAT-3's lifetime be applied to future satellite designs such as CASTOR and FalconSAT-5 so that the on-orbit anomalies can be prevented or resolved more quickly?

1.3 Approach

The thesis will begin by reviewing previous work in Chapter 2. The difference between "modeling" and "simulation" in regards to the paper will be addressed at the onset, along with their role in several systems engineering handbooks from NASA, INCOSE, and particularly the DoD and the Air Force Space and Missile Systems Center (AF SMC). The background research will also include a discussion of many different modeling architectures, both general systems modeling approaches (such as UML or OPM) and space specific modeling approaches (dynamic simulations, satellite tool-kits). This section will conclude by identifying a model-based design and operations gap for small satellites.

Chapter 3 will discuss the Integrated Evolutionary Model (IEM) design approach. This will begin by defining the different elements in the IEM design (SIM, TDA, MAT) and the general IEM approach. The chapter will then describe how the model should be used in all phases of the satellite design, from concept through operations. Examples are given or referenced for each of the five design phases, and will include ExoplanetSat, CASTOR, and FalconSAT-3.

The IEM elements previously defined are implemented in Chapter 4. The SIM element design is implemented through additions of a command-based, event driven capabilities in the CASTOR satellite integrated model. FalconSAT-3 is used to illustrate the operational development of a TDA tool, and a trajectory analysis tool design is described for the CASTOR MAT.

The next chapter, Chapter 5, will analyze and discuss the results of the implemented model described in Chapter 4. It will begin with describing the results of the CASTOR and FalconSAT-3 models, and then look at general guidelines and lessons that can be applied to small satellite and university satellite design processes as a whole.

The final chapter concludes the thesis and analyzes areas of improvement and future work than can be pursued regarding this topic and the work outlined throughout the thesis.

Chapter 2

Background

2.1 Modeling, Simulation, and Integrated Modeling

The DOD System Engineering Fundamentals Guide defines a model as "a physical, mathematical, or logical representation of a system entity, phenomenon, or process," and a simulation as "the implementation of a model over time" [37]. To expand on these definitions, an integrated model is simply defined as a collection of models that interface with each other through dependencies and relationships. The field of systems engineering includes many different high-level modeling architectures, frameworks, and methodologies, such as the Unified Modeling Language (UML) and Object-Process Methodology (OPM). Software platforms such as SysML (Systems Modeling Language) and OPCAT (Object-Process CASE Tool) utilize respective UML and OPM methodologies, and can be used to represent, and therefore model, any system [10]. These modeling methodologies and tools have many uses throughout all disciplines, including aerospace engineering and satellite operations. However, this thesis will focus on more specific applications of modeling and simulation that can be used in greater depth in the satellite engineering process.

2.2 Satellite Engineering Process

The satellite engineering process and general systems engineering process is described by many different agencies and organizations. Examples of these processes can be found in

documents such as the AF SMC Systems Engineering Primer and Handbook (SEP&H) [1], NSS 03-01 [2], NASA's System Engineering Handbook [25], and the INCOSE Systems Engineering Handbook [17]. These processes include programmatic and technical specifications, standards, and methodologies.

2.2.1 Satellite Engineering Life Cycle

Prior to discussing modeling in the satellite engineering process, the general life cycle of a satellite must be understood. The life cycle is specified by most organizations as lettered design phases. Figure 2-1 shows the DoD Space Systems Acquisition Timeline as specified in NSS 03-01, the DoD Space System Acquisition Process document.

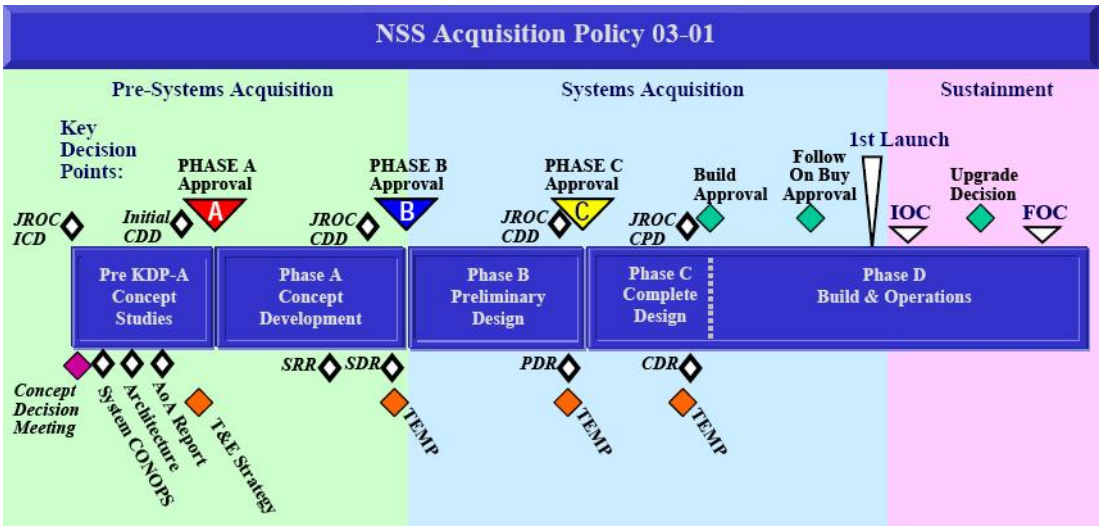


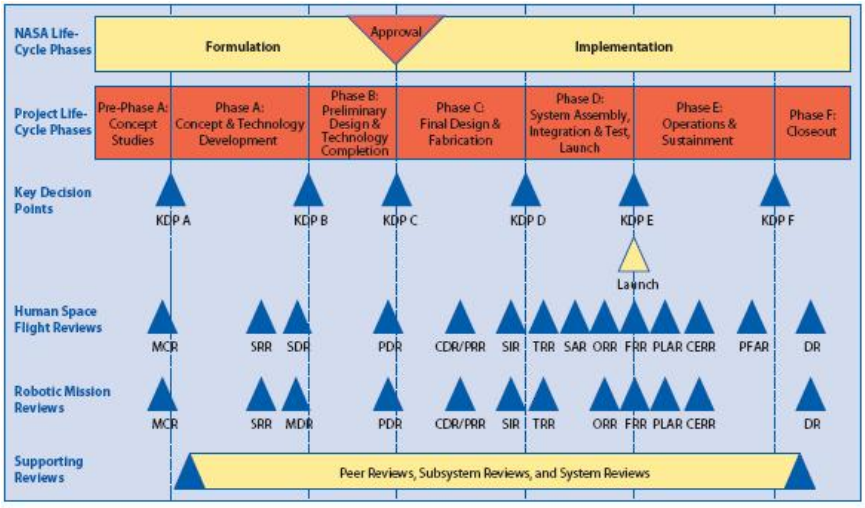
Figure 2-1: NSS 03-01 Acquisitions Phases [2]

This figure identifies five distinct phases in the spacecraft life cycle. These phases are summarized below:

- **Pre-Phase A: Concept Studies** — Identification of mission objectives, architecture and key systems and technologies. KDP-A in NSS 03-01 stands for Key Decision Point-A, which is the approval point to move to Phase A.
- **Phase A: Concept Development** — Define system requirements and perform trade studies to identify desired design architecture. Phase A includes two reviews, the Systems Requirement Review (SRR) and Systems Design Review (SDR).
- **Phase B: Preliminary Design** — Increased details in design, analyze and reduce system risk, and test critical technologies. Phase B is concluded by a Preliminary Design Review (PDR).

- **Phase C: Complete Design** — Finalize any design choices and have a fully verified design and hardware selected. Test components and assemblies to increase design confidence and lower risk. Phase C concludes with a Critical Design Review (CDR).
- **Phase D: Build and Operations Phase** — Manufacture, assemble, and integrate all components, sub-assemblies, and assemblies into complete system. Perform functional and environmental tests on both ground and space segments to give full confidence in success of mission. This phase also includes all launch activities and then transitions to mission operations.

The NSS 03-01 document is just one example of the space system acquisitions life cycle. Other agencies such as NASA have similar project time lines, although they differ in the number of reviews, number of phases, and the nomenclature. The NASA timeline is shown in Figure 2-2.



CDR	Critical Design Review	PLAR	Post-Launch Assessment Review
CERR	Critical Events Readiness Review	PRR	Production Readiness Review
DR	Decommissioning Review	P/SDR	Program/System Definition Review
FRR	Flight Readiness Review	P/SRR	Program/System Requirements Review
KDP	Key Decision Point	PSR	Program Status Review
MCR	Mission Concept Review	SAR	System Acceptance Review
MDR	Mission Definition Review	SDR	System Definition Review
ORR	Operational Readiness Review	SIR	System Integration Review
PDR	Preliminary Design Review	SRR	System Requirements Review
PFAR	Post-Flight Assessment Review	TRR	Test Readiness Review
PIR	Program Implementation Review		

Figure 2-2: NASA Project Life Cycle [25]

This thesis will use a timeline that is very similar to both the DoD and NASA examples. It is more general as it does not specify approval points or important documentation, and it only includes the four reviews. This general design phase is shown in Figure 2-3. This figure take the majority of the phase names from NSS 03-01, but like the NASA timeline, it distinguishes between build and operations phases with a Phase E. The fourth review shown

is the Flight Readiness Review (FRR). The SRR and SDR reviews are shown together to culminate Phase A, however it is likely that the SRR would occur prior to SDR. It should be noted that these phases were also picked to represent the SEA design described in Section 1.2.3.

Figure 2-3: Generic Space System Life Cycle

The design life cycle of a space system can also be referred to as the conceive (Pre-A and Phase A), design (Phases B and C), implement and test (Phase D), and operate (Phase E) process. This process, also known as CDIO, is an educational framework created in the MIT Department of Aeronautics and Astronautics, that has spread to over 50 universities in more than 25 countries [11].

2.2.2 Modeling in the Satellite Engineering Process

A large amount of work has been done previously in modeling in the satellite engineering process. This work can be sorted into the following categories, which follow the space systems life cycle:

- System architecture (Phases Pre-A to A)
- Preliminary design (Phases A to B)
- Detailed design (Phase C to D)
- Operational analysis (Phase E)

Systems architecture studies occur at the very beginning of the system life cycle. In many cases, these are feasibility studies to perform large scale concept trades in selecting a mission architecture for complicated constellation or fractionated designs. Example in this class are Jilla's master's thesis on analyzing the trade space for a separated spacecraft interferometry mission [18], de Weck's master's thesis on modeling and simulation for the next generation space telescope design [8], and Cohan's master's thesis on integrated modeling in the control architecture of lightweight space telescopes [4]. These examples are all highly technical

papers that analyze the optimal architecture to meet a specific objective, and all utilize integrated modeling in the process.

Significant work has also been accomplished in the modeling of satellite preliminary designs. Pannebecker's thesis compares and contrasts different options for preliminary design modeling tools [29]. Pannebecker also develops a spreadsheet tool that can be used for preliminary spacecraft design. All of the tools described involve integrated modeling, as it is important during the preliminary design to show the effects of different subsystems on each other. COTS software platforms such as STK (AGI), FreeFlyer (a.i. solutions), and Satellite Control Toolbox (Princeton Satellite Systems) support conceptual and preliminary designs and come built with common functions for all spacecraft, such as orbital propagators and coverage analysis.

However, when the satellite moves into more detailed designs, the preliminary models are typically replaced with detailed subsystem modeling tools. Examples of these modeling tools are structural Computer Aided Design (CAD) and analysis tools, thermal design tools, attitude models, and payload models. These models are used to perform the complex analysis that is expensive or unrealistic to test on the ground. No standard commercial tool exists that meets the detailed design requirements for all subsystems and demonstrates the integrated performance of the design. Creation of an integrated performance model for a complete system is not standard for small satellites due to the large resource cost for a seemingly minimal benefit.

Finally, operational analysis tools are used to analyze satellite data. While certain extensions of this data can be compared to COTS tools, such as the orbital position or attitude, the majority of the telemetry is specific to the satellite's payload data and supporting subsystem health (bus) sensors. Operational analysis products are typically internally-created programs or scripts developed during later design phases or even after launch. Only recently have university satellites become a reality, and the operational analysis methods for these spacecraft are largely undefined.

The subject of model-based design is gaining momentum in the systems engineering field. The Object Management Group (OMG), the same organization whose work led to the adoption of UML, outlined a general Model Based Architecture in 2003 [15]. Noriyasu,

et al apply model based design principles to the satellite engineering process [27]. Noriyasu's work on the Quasi-Zenith satellite system identifies many of the same principles outlined in the modeling design explored in this thesis. These principles include not only a model-driven satellite design, but also a command-based simulation as an operational tool. The increasing use of model-based design in the satellite life cycle has also led to NASA's Standard for Models and Simulations (NASA-STD-7009) [24]. This standard describes the requirements for the use of modeling and simulation in a system design with an emphasis of how credible the model and simulation results must be the function that they serve.

2.3 Background Summary

Modeling, simulation, and integrated modeling have been used in all stages of the space system design process. However, the majority of the integrated modeling tools are only available in preliminary design phases, as the detail required in complete designs is specific to a given spacecraft. With university satellites starting to perform complete missions, a gap is identified in the development of complete integrated performance models and operational analysis tools. This thesis attempts to move towards filling the gap by providing a model-based approach to small satellite design and operations. This approach will utilize systems engineering and mission assurance principles as well as applicable modeling and simulation standards such as NASA-STD-7009 in the creation of the approach.

Chapter 3

Integrated Evolutionary Model Approach

3.1 IEM Framework

This chapter describes the design of an integrated evolutionary model approach for small satellite development. The general approach that will be used is shown below in Figure 3-1. There are four main blocks in this diagram: Satellite Integrated Model (SIM), Satellite (SAT), Telemetry and Data Analyzer (TDA), and Mission Assurance Tool (MAT).

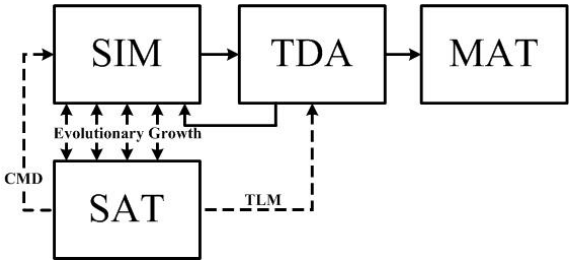


Figure 3-1: Integrated Evolutionary Model Framework

The chapter will first define these blocks, and then describe how the integrated model will change through the satellite life cycle as given in Section 2.2.1, Figure 2-3. All acronyms can be found on page 17. Examples are given for many of the different phases of design (A-E) as well as for each integrated model element (SIM, TDA, and MAT).

Throughout the chapter, certain tasks are written to state that they "should" occur

while others are written that they "will" or "must" occur in a certain structure or method. This is done as this chapter describes a baseline design process and it is expected that certain elements described may be modified for a specific project. However, to fully track the progress of the IEM design through all phases, all tasks or decisions that "should" occur will be assumed to be executed as written.

3.1.1 Satellite Integrated Model (SIM)

The SIM represents the integrated model that is developed through the entire design process to represent the system. In the evolutionary model approach, the SIM is created at the beginning of a program. During the conceptual phase, the SIM can be used to develop specific requirements and Concepts of Operations (CONOPS), compare major design decisions, model new systems and analyze their effects on the mission, and to assist in the feasibility and visualization of a new idea. As the mission is defined and requirements are identified, the SIM can be tailored to specifically look at the design options being analyzed. Unknown systems or components can be parameterized to allow for more specific subsystem and component trades. Near the end of the design process, the SIM can be used to demonstrate the feasibility of the design through integrated simulation or similar feasibility studies. As hardware is acquired and tested, the SIM can be updated to tune the model prior to operation of the satellite. By launch, the SIM would be complete and ready to be used to predict the satellite's behaviors and/or performance.

The SIM interacts with two other blocks: the satellite (SAT) and the Telemetry and Data Analyzer (TDA). The interaction between SIM and SAT is shown by four two-way arrows in Figure 3-1. These arrows represent all phases of the satellite design life cycle other than operations (Phases A-D). During these phases, the SIM would be used to define what would ultimately become a satellite Flight Model (FM). Additionally changes in the satellite design, which can occur often during a program due to programmatic or technical considerations, will need to be fed back into the SIM. The SIM primarily interfaces with the TDA by providing its output to the TDA to be used as comparison with the SAT. A feedback loop from the TDA to the SIM represents the operational level of the design process. In this loop, the SIM would be updated to tune simulation and satellite parameters, assess the

feasibility of operational changes, account for new information (e.g. loss of hardware device or new control software), or fix any inaccuracies.

Additionally, a command input segment exists in the SIM, displayed in Figure 3-1 by the dashed line between the SAT and the SIM. This represents using the actual commands executed by the satellite as inputs to the SIM. This allows the integrated model analysis elements to have a complete comparison between the simulation output and the actual satellite telemetry. Therefore the SIM can produce two outputs: expected future behavior of the satellite (*a priori*) and the post-processed event-based (*a posteriori*) behavior. Both outputs are useful and can be fed into the TDA. The *a priori* results can be compared to the *a posteriori* output to see if the satellite's major events and tasks lines up to the predicted behavior. Furthermore, the *a posteriori* generated output is useful to effectively compare the simulation and satellite data. Without the synchronization of commands and common baseline, it becomes difficult for the TDA to clearly find any anomalous data.

In this chapter the SIM will be referred to as the "integrated model" in the early phases. Once the other IEM model elements are created in later stages, "integrated model" will refer to the SIM, TDA, and MAT.

3.1.2 Satellite (SAT)

The satellite (SAT) block in Figure 3-1 represents both the satellite from its early concept as only an idea to its final on-orbit flight configuration. The block could be represented by the generic satellite phases or CDIO description from Section 2.2.1. All design reviews, prototypes, and engineering models are represented by this as well. The dashed line from the SAT block to the TDA block represents the telemetry that comes from the satellite. Although this could represent test data or early functional testing, this primarily represents the space to ground segment telemetry from the on-orbit satellite. As previously described, the dashed line that connects the SAT to the SIM represents the commands that the satellite has executed. The frequency and number of commands sent will depend on the scope of the mission as well as the importance in simulation on the ground. For the IEM architecture to succeed, the following requirements are placed on the satellite:

- The satellite must send time-tagged commands that it executes
- The satellite must downlink time-tagged telemetry to include bus health, system and subsystem performance, and payload data at a minimum

The four top arrows in the SAT block represent the conceive, design (preliminary and complete), and implement stages of the design process, and the right dashed arrow represents the operate stage. The diagram does not show any feedback to the satellite once it is in the operational stage, as it is assumed that one cannot change the existing FM hardware once on-orbit. The exception to this is the case of having redundant hardware systems, often referred to as "A-side" and "B-side" components. Most small satellites and university-class satellites do not have these redundant systems in place. Therefore, while possible to model using an IEM, the on-orbit transition from different hardware elements is not heavily discussed in this thesis. The most common way to change the FM when on-orbit is by uploading new software, although this is not necessarily a direct feature in the SIM or TDA. The SIM will likely be used to test new software or algorithms for the FM, so there is an indirect relationship with the SIM updating the SAT once on orbit.

3.1.3 Telemetry and Data Analyzer (TDA)

The TDA block represents the juxtaposition of the satellite telemetry data with the integrated model output. A feedback arrow runs back to the SIM block representing possible updates in the integrated model based on TDA results. No feedback arrow returns to the satellite as the TDA does not represent the command and control interface with the satellite. The TDA outputs the result from comparing the measured and processed data with the integrated model data and feeds into the Mission Assurance Tool (MAT). The type and format of this data is dependent on each mission, and will be discussed in greater detail with possibilities and examples later in the thesis. In general, the TDA will analyze the bus health and functionality of the satellite, leaving the analysis of the payloads and major mission performance characteristics to the MAT. Any post-processing or filtering of telemetry is done in the TDA in this framework. One common example is filtering attitude sensor data to determine the attitude position and angular rates of the satellite.

It should be noted that the TDA is different than an operational telemetry display

screen. Often during real-time satellite contacts, the current telemetry is streaming live and being displayed on an operator's screen. This is referred to simply as a real-time telemetry output, or a status of health output. The TDA compares and measures simulation data with satellite telemetry, whether in real-time or post processed.

Although the TDA and SIM blocks are separated in this diagram, they may be integrated in one program in reality. This decision will depend on the satellite operations and analysis plan for the given mission. The blocks are distinguished in the diagram to illustrate the difference in their function. The SIM is meant to predict the state as well as to run the commands from the SAT to give an event-based prediction, and the TDA is meant to take either prediction and compare it to the actual data. The function of comparing and analyzing data is purely done by the TDA.

3.1.4 Mission Assurance Tool (MAT)

The MAT block represents the final step of analysis of the fully processed data. This block focuses on measuring the high-level mission of the satellite system. This tool is used to further process any data to provide to a customer or to track the progress and success of the mission. As with the TDA, the MAT may be represented in the same tool as the SIM; however, its function is unique enough to list it separately. Certain missions may have mission objectives that are very straightforward and easy to measure, which lead to a grays area in the importance of the MAT. One such fictional mission is to downlink one thousand images in three months at a minimum frequency of five pictures per day. This mission is easy to measure, so it is not clear why an additional tool is necessary. The team may decide that tracking the total number of images and the number of images downlinked per day is sufficient to meeting its requirements, and the MAT will simply track these values. However, the intended use for the MAT is to not only track the measurable criteria, but to analyze the payload and mission performance to get a better look into the interaction of the different systems. For example, the TDA may output that the actual link margin is becoming increasingly lower than the expected, leading to the conclusion that a problem may occur that would force a lower data rate, which in turn will decrease the number of images that can be downlinked per day. Although the TDA identified this problem, the

mission impact is identified through the MAT design and execution.

The MAT is most useful for small satellites that have a mission or payloads that cannot be fully predicted without taking into consideration the dynamic nature of a multitude of integrated subsystems. The CASTOR design is a clear example of this case. The mission objective is to characterize the thruster's on-orbit performance. Without considering the operating commands, timing, pointing direction, and power levels, the thruster performance could not be fully analyzed. With a significant increase in cost, CASTOR could be designed to have perfect pointing, operate the thruster continuously and at a constant power level, making the thrust characterization problem much easier; the implementation of the integrated model and MAT bypass this cost increase and allow for a much lower satellite complexity and cost.

3.1.5 General IEM Design Approach

The depth and detail of each part of the the IEM framework will depend greatly on the mission, as well as the design phase. The next sections will outline the baseline requirements for using the integrated model throughout the design phases described in Section 2.2.1, Figure 2-3. Examples will be given for each phase for each IEM element (SIM, TDA, and MAT). However, there are certain elements of the IEM design approach that are consistent through each of the phases. Three such elements are IEM deliverables timing, documentation, and configuration and change control.

IEM Deliverables Timing

It is important for the IEM elements (SIM, TDA, MAT) to be delivered in time to benefit the satellite design. The simulation framework and structure is most important to design and complete earliest, due to its inherent nature with all other elements and models. Changes are anticipated to occur after this baseline model is created and are discussed in the configuration and change control section. While important to be completed in time for analysis, the integrated model must also be understood early in the different phases to ensure the design team will properly utilize the different functions. Therefore a base set of documentation must be available.

Documentation

Documentation of the IEM design and its products is necessary throughout each phase. At a minimum, documentation must exist that gives instructions for utilizing the different tools, so that different teams can perform analysis with the model. The following process for documentation is recommended, and it creates two sources of documentation: a user's manual and a design document.

The user's manual will describe how to run the integrated model and its different features. This includes explaining the types of analysis possible, example outputs, and limitations of the model. The user's manual is written for individuals who will execute the IEM products without understanding the programming structure or logic involved. This is most important in the operations phase, as those using the integrated model may know little about the development of the model. The user's manual can also serve a training purpose to get new individuals familiar with the satellite by analyzing outputs from various simulated operating conditions. The user's manual must be produced by the time the integrated model is delivered.

The integrated model design document is written to give a much deeper background and understanding of the integrated modeling framework. This document will grow through the different phases as the integrated model evolves. The design document can be outlined as follows:

1. Overview
    - Satellite overview
    - Model objectives and purpose
2. Model Organization
    - Model structure (e.g. software interactions, functions used, objects and methods)
    - Information flow (e.g. N^2 diagram)
3. Model Design
    - Operational logic
    - Individual model functions
        - Reasoning/justification and assumptions
        - Model verification and validation results
        - Shortcomings and updates required

The introductory section will include an overview of the satellite and state how the model will be used in the satellite design and operations.

The model organization section will describe the flow of information through the integrated model. This includes the different inputs, outputs, variables, parameters, functions, and objects that are contained in the model. In this section, it is acceptable to treat most functions and objects as black boxes, meaning that the internal behavior of the specific function is not described. A common organization structure, the N^2 diagram, is shown on page 48. This is done as there may be different authors of specific model elements, and the interface with other elements must be set to ensure proper functionality upon integration.

The majority of the design document will describe the design and logic of specific model elements. This is meant to provide a greater level of understanding and reasoning than basic code comments, although ideally the code will be commented with similar notes. This logic can be represented by algorithms that specify all formulas, conditional statements, and assigning of local and global variables. Vallado's astrodynamics book gives an excellent example of how different functions are represented by algorithms [41]. This is an excellent example as the code can be found in many different programming languages for free online [40]. This section will also describe the motivation and reasoning for any user-created function or statement, which has great utility in programs with high turnover. This section should also discuss the verification and validation of the model and simulation elements. Prior to making any critical design decisions based on a model, it is important to have full confidence in the model. Model elements may be implemented prior to being verified or validated, so it is important to understand their effect on the system. The effect on the user and the limitations for which the model is valid will be specified in the user's manual as previously mentioned. Lastly, the limitations and shortcomings of the model should be described, and necessary model updates should be documented.

Configuration and Change Control

Both the integrated model and documentation deliverables require configuration control due to their frequent updates and changes. It is recommended to follow a process of producing the initial deliverables in the model and documentation as early as possible in the design

phase, and finalizing the model and documentation at the end of the design phase. There are several ways to approach the problem of configuration and change control, and this section will outline one type of approach.

As many individuals and teams may be changing different elements, a SubVersion (SVN) repository system should be used to ensure that one update does not cause problems for the entire integrated model. A commonly used freeware SVN client that can accomplish this task is Tortoise SVN. These model changes should have supporting addendums to both the user's manual and design document. At the final deliverable of each phase, these addendums will be formatted into the user's manual and design document as necessary. At the beginning of the next phase, the recently "finalized" model and documentation should be placed in a "controlled" folder and not be modified (i.e. write permissions locked). Any modifications should be done in the "working" folder, with addendums to supporting any changes. Note that modifications done in the "working" folder should still be controlled utilizing SVN processes, as several personnel may be updating the model elements concurrently.

This structure, displayed in Figure 3-2, would create five different controlled user manuals and design documents in each stage. Although this seems to be wasteful, particularly with a SVN repository in place, this ensures that each phase and its corresponding model and documentation is clearly discernible. This also allows for a clear example for any subsequent mission or designs using a similar IEM approach.

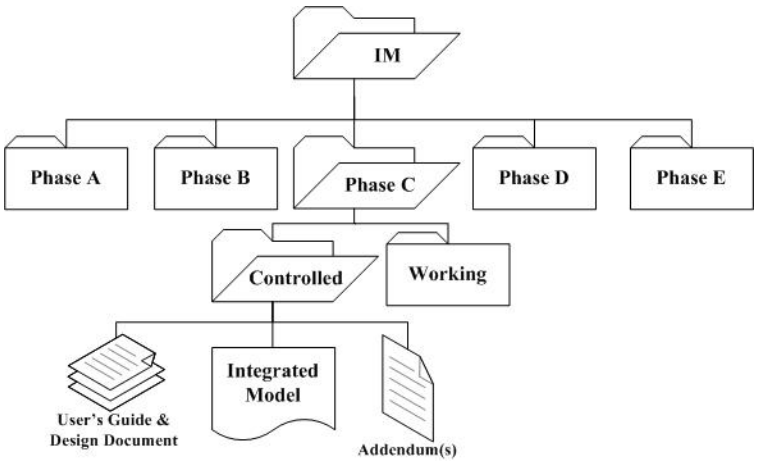


Figure 3-2: IEM Configuration Control Example

3.2 Conceive — Phase A

3.2.1 Phase A Integrated Model Design

In the early concept studies and development stages, modeling can be utilized to identify any uncertainties as well as to perform trade studies of major architectural or design changes. One such example is determining the optimal design of a new payload system, such as a segmented mirror design for improving the areal density, or mass per unit area of the primary mirror, in imaging missions [5]. Cohan's dissertation gives a detailed example of how integrated modeling can be used in preliminary conceptual studies phase (Pre-Phase A) to assist in the development and design of a technology.

This thesis is more focused on Phase A and concept development phase as a starting point, as this is when the integrated system is first developed. This phase begins with the development of a mission statement, objectives, and success criteria. This mission leads to a baseline system with certain payloads and primary subsystem. The integrated model can be used to develop the CONOPS, identify system requirements, and establish the feasibility of the mission and system architecture identified. The initial integrated model should perform the following:

1. Quantify mission objectives with Figures of Merit (FOM) and create a plan to measure and record the FOM
2. Identify major payload specifications and requirements
3. Identify primary subsystem(s) and mission parameters
    - Determine the uncertainty and familiarization of each subsystem
    - Identify key requirements on each interface or subsystem
    - Estimate the intended depth of detail required for each subsystem model to perform mission
4. Determine integrated model platform and supporting personnel and resources
5. Create initial model of system
    - Model must integrate payload and key subsystem(s) to validate mission

The first requirement for the integrated model is to support the development of the MAT and TDA. The figures of merit (FOM) identified will later be used in the MAT to

measure the mission success. Examples of FOM include coverage or collection time, data downloaded, images taken, or ΔV performed. The FOM should be clearly identified so that they are measurable and trackable and are not a simple binary measure of whether something has happened or not. The FOM also will be used later to assist in the TDA output.

The second requirement is to determine the major payload requirements. Different payloads require different degrees of support from the spacecraft bus, and this is important to know to determine which subsystems are most important to mission success. An example is that a communications satellite with omni-directional antennas may require little attitude control, but a similar satellite with a high-speed laser communications platform will require much finer control. Not all payload requirements are fully specified at this point, so this step is primarily a qualitative assessment based on previous experience or a best estimate of the driving factors.

The next step is to identify the primary subsystems and interfaces that stem from the mission objectives and payload requirements. This is one of the most important steps in forming the outline for the initial integrated model as it is the cornerstone to the scope and detail of the model through the rest of the design process. Examples of primary subsystems are the power, attitude control, thermal, guidance, communications, avionics, and propulsion systems. Additional mission parameters that may be very important are the orbital characteristics, mass, volume, cost, schedule, lifetime, and operational capabilities. For each primary subsystem or mission parameter, the team should determine the level of confidence and uncertainty associated with each. Additionally, in this step, the team should identify how the different systems interface and depend on each other. Determining the coupling and integrated nature of the subsystems will depend heavily on the next step, which is to determine the depth and detail that the model should give for each system. This intended depth is for the intended flight integrated model, and is *not* the level of detail necessary in the Phase A model.

The four measures should be used to determine the initial subsystems and mission parameters to model. Additionally, secondary subsystems that are important to the mission but not critical to its success should be noted at this time for future implementation into

the integrated model. The decision of which mission elements to model can be done with a varying degree of effort and planning, depending on the level of understanding of the mission. If the mission has a clear concept or straightforward objective, it is possible to pick the subsystems or parameters without any formal quantification of the elements. To perform a more quantifiable method of choosing which mission elements to model, a rating or weighting system can be used. Requirements framework such as Quality Functional Deployment (QFD) [21] can be modified as well to identify major mission parameters.

An example of a quantifiable rating system is shown in Table 3.1. The weights associated with each are a Boolean flag for the importance of the subsystem or parameter, and then a integer 0 to 4 for the uncertainty level, detail required, and integrated nature. These weights could be modified as desired for a particular mission. The basic mission identified for the example is the on-orbit demonstration and characterization of a low-thrust electric propulsion system.

Table 3.1: Integrated Model Elements Decision Selection for Low-Thrust EP Mission

Subsystem	**Importance (0 or 1)**	**Uncertainty (0 to 4)**	**Detail Level (0 to 4)**	**Integration (0 to 4)**	**Score (J_{el})**
Power	1	3	3	4	10
Thermal	1	2	3	2	7
Avionics	1	2	1	1	4
Propulsion	1	2	4	3	9
Comm.	1	2	2	2	6
Guidance	1	2	2	1	5
Attitude	1	2	3	2	7
Structure	0	2	1	2	0
Payload	Note: Propulsion system represents payload				—
Mission Parameters	**Importance (0 or 1)**	**Uncertainty (0 to 4)**	**Detail Level (0 to 4)**	**Integration (0 to 4)**	**Score (J_{el})**
Orbit	1	2	4	3	9
Mass	1	1	1	2	4
Volume	0	1	1	2	0
Cost	0	1	3	1	0
Lifetime	1	2	2	1	5
Operations	1	3	2	3	8

The decision of which mission elements to model by flight can be chosen by formulating a cost function or weighting equation. The exact equation, like the weighting and scaling

factors, can vary. The equation used in this example is shown below, with I representing importance, U as uncertainty, D as detail required, and C as the coupling or integration term.

$$J_{element} = I \cdot (U + D + C) \tag{3.1}$$

The results of equation 3.1 for this example are shown in the last column in Table 3.1. With each subsystem or mission parameter ranked and weighted, the final action in this step (Step 3) is to decide which elements will be included in the model. This can be done by placing cutoff values on numerical scores, by using the numerical data to make a more detailed qualitative decision, or simply deciding without any formal process as mentioned before. In this example, the integrated model would consider the effects of everything with a score greater than 0 by the end of the system development and design (Phase D). To start, elements with a threshold score of 8 or above would be modeled by the end of Phase A, which would lead to an initial integrated model with the power, propulsion, orbit, and operations systems represented. The cut-off value for deciding when each element should be included in the model will be primarily based on the availability of resources. The model should be used to support the decision making and mission validation processes in support of Phase A deliverables like SDR.

To generalize this process, as it should be modified for each mission, the different subsystems and mission parameters can be generalized into the following criteria:

- Primary Elements — Payload, essential subsystems, and mission drivers
  Example: Propulsion, power, orbit, & operations ($J_{element} \geq 8$)
- Secondary Elements — Important subsystems that interface with primary elements
  Example: Thermal, communications, & attitude ($8 > J_{element} \geq 6$)
- Constraints — Mission characteristics that affect primary and secondary elements
  Example: Mass and lifetime ($8 > J_{element} \geq 4$)
- Supporting Elements — Other subsystems or parameters that support model elements
  Example: Avionics and guidance ($6 \geq J_{element} > 0$)

To reiterate, these examples are specific to Table 3.1. This general nomenclature will be used in Table 3.2 later and throughout the rest of this thesis.

In Step 4 of the Phase A IEM process, the team must decide which platform the integrated model will utilize as well as the resources that will be allocated to its development. There are two major classes of platforms to choose between: commercial, off-the-shelf (COTS) software or internally developed software. COTS software packages such as AGI's Satellite Tool Kit (STK), a.i. solution's FreeFlyer, or Princeton Satellite System's Satellite Control Toolbox (SCT) come pre-programmed with many tools that are necessary in any space mission, and usually allow for user-defined functions or add-ons, whereas user-defined software is developed in-house. There are advantages and disadvantages to both classes of software, so it is up to the personnel of the team to determine which type is best for their use. The recommendation of the author is to utilize COTS if it does not have to be greatly modified to meet the final intended use of the integrated model. An example would be using STK to perform communication coverage analysis for a communications payload. However, it is recommended that if the integrated model is being developed to analyze the interaction between different subsystems and specific mission parameters, a user-defined software that is built specifically for the mission may be a better choice. An analysis and discussion of this reasoning can be found in section 5.3.1. The decision of COTS versus internally developed software is not necessarily black and white. Included in the internally-developed criteria are basic COTS software packages such as MATLAB® or Microsoft Office Excel®. A combination or hybrid method of utilizing the maturity of COTS space packages with the flexibility of internally-developed software is also an advantageous option. Many different software platforms and solutions exist, therefore it is highly encouraged to utilize the tools that are familiar to the team, if possible, as the primary decision choice.

The second half of choosing a platform is assigning a workforce and requirements to the integrated model development. The hypothesis of this thesis is that an integrated evolutionary model will *indirectly* free resources through improvement of design choices, ease of feedback, and lower operational costs. The integrated model itself will require a significant amount of effort that must be accounted for in the program's Work Breakdown Structure (WBS). At this point, the team should outline the intended functionality and usage of the model through its lifetime in a set of integrated model requirements for each phase of the satellite. An example of these requirements based on the definitions of elements

(primary, secondary, and supporting) and constraints explained previously is shown in Table 3.2.

Table 3.2: Example of Integrated Model Requirements by Phase

Phase	**Requirements**
Phase A Concept Development	Model interaction of primary elements Create implementation plan for secondary elements Validate mission and demonstrate mission completion
Phase B Preliminary Design	Incorporate secondary elements in model Update primary element models to PDR design Incorporate constraints into model Verify any testing data or prototypes with model Outline plan for implementing supporting elements Perform trade studies and parameterize majority of inputs Create TDA and MAT outline Validate mission and demonstrate mission completion
Phase C Complete Design	Incorporate supporting elements into model Update primary, secondary, and constraint elements to CDR design Use prototypes, hardware, and testing data to verify model Limit variables and parameterization to select element characteristics Create TDA and MAT software tools Feed in initial command and telemetry formats into SIM and TDA Validate mission and demonstrate mission completion
Phase D Build & Test	Freeze model parameter selection to match QM or FM design Simplify any unnecessary elements or model features Verify model and hardware with prototypes and tests Complete TDA and MAT and test with QM or FM telemetry Create on-orbit integrated model execution procedures Incorporate flight software or command structure (if necessary) Validate mission and demonstrate mission completion with MAT
Phase E Operations	Perform any on-orbit calibration or tuning of SIM, TDA, or MAT Maintain TDA, MAT, and SIM to interface with SAT data Update SIM for any major operational changes (SW, CONOPS changes) Update SIM, TDA, MAT to analyze and resolve any anomalous behavior

The requirements in each phase will be further explained in the rest of the chapter along with examples in Chapter 4. The requirements, workforce, and framework for the integrated model that are chosen in Phase A can be revisited and modified throughout the design process as the familiarization with certain concepts changes, new information is received, or if the mission concept and scope significantly changes.

The fifth and final step in developing the integrated model in the conceptual development

phase is to create and execute the first cut of the integrated model. This means that the primary elements should interact with each other to obtain an output that can validate the success of the mission. To assist in the software development and model organization, a N-squared (N^2) diagram can be utilized. N^2 diagrams illustrate the linking between different model elements by showing input and output parameters that feed to and from each element. This allows the work to be separated and the modules created and modified concurrently. An example of a very simple N^2 diagram is shown in Figure 3-3 between the power, propulsion, orbits, and operations system for the EP characterization mission.

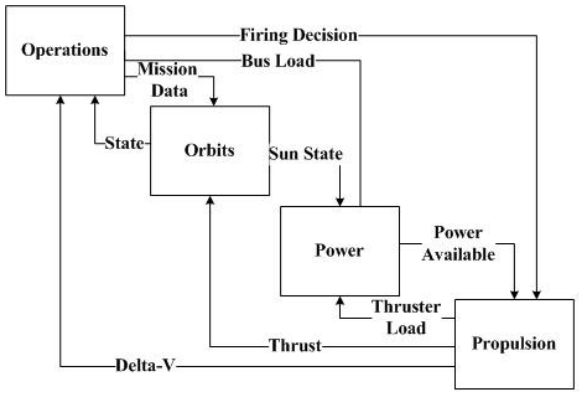


Figure 3-3: Example Phase A N^2 Diagram for EP Mission

Arrows that come from the top of a block represent inputs to a function that come from a previous module, which are shown as arrows that stem from the right of a block. The arrows that originate from the left of a block represent outputs that need to feedback to a previous model element that has already been run. These outputs then connect to the bottom of a block. An ideal N^2 model will minimize the feedback paths to simplify the model. However, in a complex system, many feedback paths become necessary, so it is up to the model developers to decide on the best arrangement for both software performance and intuitive nature.

As Phase A is still dealing with a concept and the majority of the satellite performance parameters will not be defined, the initial integrated model should have a focus on parameterization rather than model accuracy. A first order approximation of a primary element is acceptable for most cases when dealing with major trades. An example of this would be to use a simplified orbital propagator for determining the orbital characteristics such as

coverage time, eclipse time, or thrusting time for a constrained electric propulsion system.

The integrated model can be described by Figure 3-4. This figure is the first building block of what will turn into Figure 3-1. The SIM block has been represented by the initial integrated model which consists only of the primary elements, labeled as P_1. The subscript of "1" is used to describe the revision number of the set of elements, and is usually correlated to the level of detail. The integrated model will produce an output that will specify if the mission objectives are met; however, this is not as detailed at the MAT will be in future phases. The SAT block is displayed as an oval, which is used to differentiate a physical object with an idea. The same labeling and description will be used for the diagrams for the other phases of design.

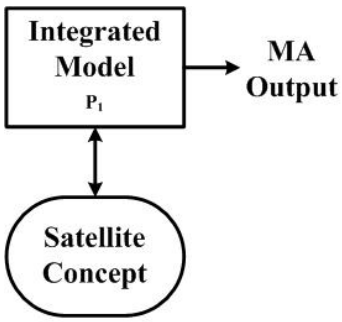


Figure 3-4: Phase A IEM Design Diagram

3.2.2 Phase A Integrated Model Example — ExoplanetSat

The Phase A integrated model design outlined was executed on the Exoplanet Cubesat (ExoplanetSat). The mission of ExoplanetSat is to use a cubesat to detect Earth-like planets orbiting bright Sun-like stars using the transit method. The team was midway through Phase A when they implemented the IEM approach. Versions of the requirements identified on page 42 already existed, albeit not specifically for use with integrated modeling. The team also had access to the Phase B and C CASTOR integrated models that will be described later. The results of implementing the IEM design in the ExoplanetSat design are described in Section 5.1.1.

3.3 Preliminary Design — Phase B

3.3.1 Phase B Integrated Model Design

Upon successful completion of SDR or the equivalent milestone, the effort should shift from concept evaluation to design. Subsystem and assembly requirements will become more specific as the interfaces become better defined. If integrated modeling was not a part of the system development, the basic physical relationship and performance verification steps would begin occurring in this phase. By PDR, a design usually is verified with different budgets to include mass, data, power, communications, thermal, and others. Therefore the effort of the integrated model in Phase B is to incorporate the results of the subsystem-specific studies into an integrated performance model of the mission. This can be done by taking key parameters from the individual analyses and using these values as the integrated model variables, or by creating a subsystem model that allows for both stand-alone subsystem analysis as well as is incorporated as a function into the integrated model. The benefit of using the detailed models directly in the integrated model is that system level trades can be performed using multidisciplinary optimization techniques much easier than if the subsystem results were independent of the model.

Before moving directly into the implementation of the subsystem models, the team should step back and ensure that the plan that was created in Phase A should be implemented as written. Questions to answer include:

- Has the mission significantly changed?
- Are the requirements that were created in Phase A (Table 3.2) still valid?
- Is the platform that was used in Phase A the best for future analysis?
- Are the primary and secondary elements and constraints all important for measuring mission success?
- Are there any other elements or characteristics that need to be added to the modeling framework?

If the answer is "yes" to any of these questions, then the team should decide if any of the integrated model plans need to be adjusted.

The primary role of the integrated model in Phase B is to integrate subsystem designs to verify that the requirements are met and most importantly, the validation of the mission. As much as possible, the design parameters in the mission and subsystem elements should be flexible. Additionally, few, if any, subsystem or system behaviors modeled that are passed up in the design should be scrapped in this phase.

If a N^2 diagram or similar diagram exists, it should be updated with the different mission element interfaces. Even if the input or output parameters are hard-coded and unaffected by the mission layout, the model structure should contain all possible interfaces between the model elements and functions. Based on the requirements from Table 3.2, the primary elements, secondary elements, and constraints should all be included in the Phase B model. However, the supporting elements will need to be defined, even if they are only constants.

The integrated nature of the model, while assisting in system level trades and mission assurance tasks, can inhibit a subsystem team's ability to perform their own trades. One such solution to this issue is to keep the subsystem models independent of the integrated model. This allows the team to work independently of other teams, test new designs quicker and more efficiently, and then upgrade the integrated model later. An alternate approach is to have in the integrated model the ability to turn "on" and "off" different functions of the satellite. This allows subsystems to only maintain one model and to analyze the effect of the other subsystems independent of each other. The major drawback for this method is increased programming to account for all of the variations of switching functions "on" and "off," as the model requires outputs from these modules.

In Phase B, it is recommended that the team outlines the plan for the TDA and MAT. Although these plans are likely to change, it is important for the team to describe the envisioned space to ground telemetry segment. The TDA plan consists of the following elements:

- Telemetry (TLM) specifications
- Ground segment TLM distribution and storage specifications
- Expected subsystem performance tests
- TDA execution and user interface
- TDA output and verification success criteria

Telemetry specifications includes the different channels that will be transmitted to the ground, their frequency of collection and transmission, their format and range of values, and any other characteristics. Although the word telemetry usually is defined as health data from the satellite, in this context it also includes data generated on the ground (e.g. commands sent) or payload data. The ground segment TLM distribution plan describes how the TLM will be parsed by the ground, and most importantly for the TDA, where it will be located and how it will be accessed. Ideally the TLM will be stored in a controlled database that is available to all stakeholders; however, there are many different types of data, and more simplified ground segment designs may have data stored in different locations or in many segmented files.

The purpose of the TDA is to verify the performance of the satellite on-orbit, so it is necessary for the team to create the on-orbit verification plans. These tests can be analyzed using the TDA. In Phase B, it is only necessary to outline the expected tests and their quantitative measurement requirements. The TDA execution and its look and feel will also be considered in this phase. Much like deciding the medium for the SIM, the TDA format and its operations will depend on the mission and the available resources. Possible formats include COTS telemetry analyzers with user-defined functions [22], a specialized interactive Graphical User Interface (GUI) for the specific satellite, or specialized programs that decode and analyze the telemetry and provide a pre-defined output report. Lastly, the TDA plan should include the expected output reports, graphs, or messages, as well as the success criteria for any tests. The first execution of the TDA will take place in Phase C with subsystem functional or algorithm tests and Phase D with integrated system functionality testing through a flatsat, qualifications model, and finally the pre-launch flight model. The TDA plan in Phase B should outline the expected tests to take place in these different phases.

The MAT plan should also be outlined in Phase B. The mission objectives and success criteria should have been quantitatively identified in Phase A, and serve as the baseline for the MAT. The MAT plan describes the input sources, MAT analysis techniques, measurable output, and execution and user interface used for the final MAT. The MAT input sources are relatively straightforward and will primarily come from the TDA outputs or directly

from flight data. These inputs will then be manipulated as necessary for the mission to provide an output to the user or stakeholder that is as clear and concise as possible. The input and outputs should already exist, so the focus of the MAT plan in Phase B is to specify the inner-workings of the MAT analysis as well as the type of program that it will look like. Just like the TDA, COTS tools may be available for some missions, and the execution of the MAT will differ for each mission.

The output of Phase B is a further defined integrated model and simulation with primary elements, secondary elements, and constraints modeled, as well as a plan for designing and implementing the TDA and MAT. The simulation will have parametric inputs used for system and subsystem comparisons, and ideally will have the ability to cycle different systems "on" or "off." The integrated model has sufficient output behavior to perform mission assurance functions and demonstrate that the mission objectives will still be met, just like in Phase A. Figure 3-5 shows the diagram of the Phase B IEM, and specifically shows the addition of the new elements, the switching function, and the planning of the TDA and MAT. This diagram uses the same format as Figure 3-4, although there are a few new formats. The dotted outlines and lines represent an intended or planned objects and path of information, whereas the solid lines represent real objects and information flow. The switch next to the C_1 represents the ability to switch the different mission element models "on" and "off." The primary element representation has been incremented from P_1 to P_2, illustrating that this element has been updated since the Phase A model.

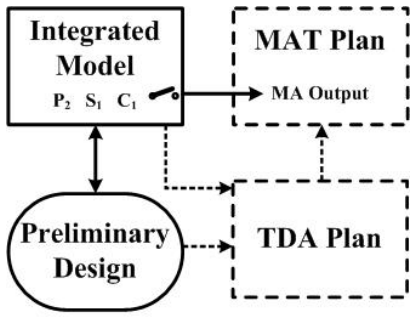


Figure 3-5: Phase B IEM Design Diagram

3.3.2 Phase B Integrated Model Example — MOTV

The Phase B integrated model can be illustrated through the MOTV simulation that was created in the Fall 2008 semester of MIT's graduate satellite engineering course (16.851). This model, created by MIT masters candidates 2LT Joe Robinson, 2LT John Richmond, Chris Pong, and Aaron Johnson, was developed to validate the design of the MOTV satellite. This satellite, which was also called OSMV for a period of time, was being designed and built by a MIT undergraduate satellite design capstone course (16.83x). The resulting satellite design and its lessons were the framework for the CASTOR satellite explained throughout the thesis.

The primary mission of the MIT Orbital Transfer Vehicle (MOTV) satellite in Fall 2008 was to achieve 2 km/s of ΔV utilizing the same DCFT in the CASTOR mission. Additionally, the team wanted to demonstrate the use of deployable, sun-tracking solar arrays and a gimbaled engine mount to allow for continuous sunlight operation of the thruster to achieve both orbital altitude changes and plane changes.

As no earlier model existed, the team started by identifying the key subsystems necessary in analyzing the integrated performance of MOTV. These systems included the orbits and thruster operations subsystem to model the DCFT thrust and maneuvers, the EPS system to model the power balance and available power to the thruster, the ACS/GNC subsystem to identify the expected pointing and losses from knowledge and pointing errors, and the thermal subsystem to model the effect of a 200 Watt thruster input power on the rest of the satellite. Additionally, a simple communications subsystem was modeled as well to do baseline link margin equations and analyze coverage statistics. Subsystem functions for each were created in MATLAB and driven by a MATLAB-based GUI that allowed for the modification of nearly every parameter used in the simulation. Figure 3-6 was presented at the final presentation in the class, and illustrates the different functions used in the MOTV model. Additionally, Figure 3-7 shows the design of the GUI used to launch the MOTV model and display the results.

The outputs from the MOTV simulation can be seen in the right screen shot in Figure 3-7. These are analogous to the mission assurance outputs in the IEM design. While

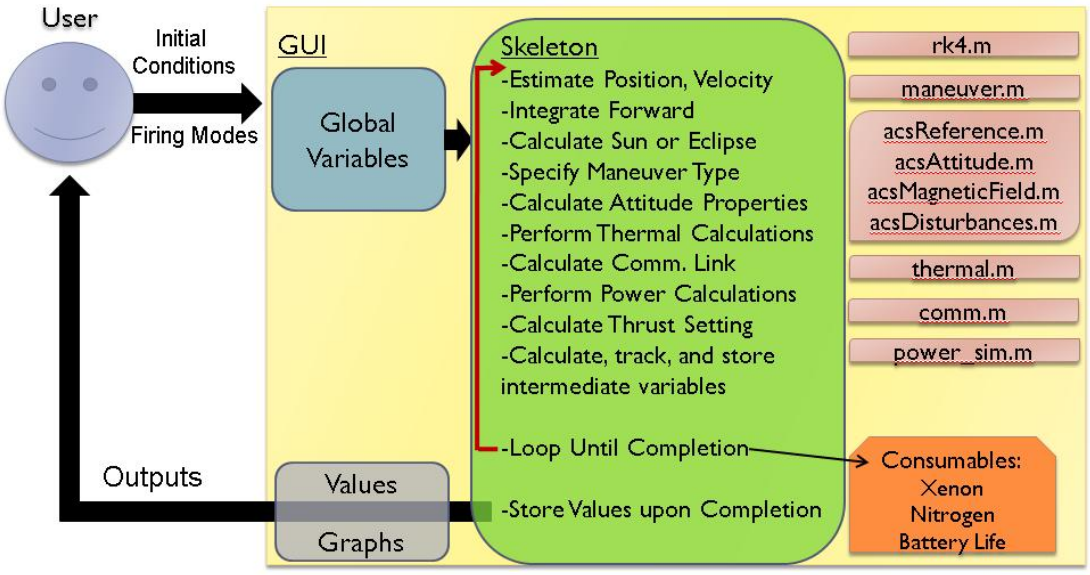


Figure 3-6: MOTV Simulation Design [32]

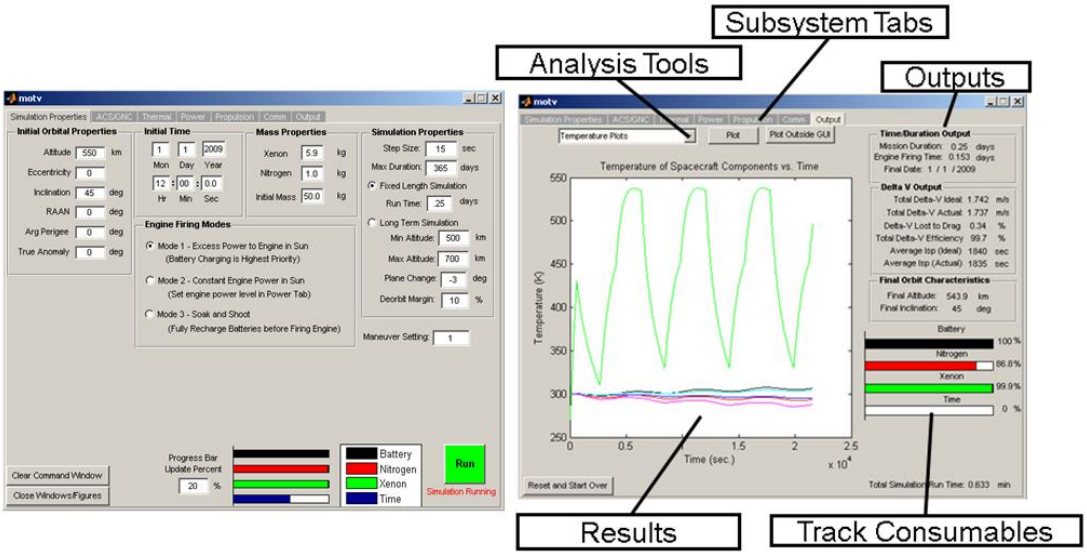


Figure 3-7: MOTV Graphical User Interface Layout [32]

no additional MAT is present, the MOTV simulation (SIM example) produces the output necessary to prove that the mission is met. In the case of MOTV, the primary output was total ΔV produced, with secondary outputs of expendables used, mission duration, and average thruster Isp. Subsystem performance data from each of the modeled elements was saved and was easily plotted by selecting a drop-down menu with more than 10 pre-defined graphs.

The MOTV simulation input allowed a user to manually adjust any of the input parameters for the subsystems. However, no optimization was done to identify the best set of parameters due to the model being implemented near the end of Phase B and the beginning of Phase C. Additionally, the subsystem switching feature was not available on this model. Therefore if a subsystem was desired to be ignored, either the parameters for the other system must have been set to ensure that the outputs were always in the desired range, or the user had to go into the code and manually program the desired state. Section 5.1.2 analyzes the results from this model and discusses its role in the IEM design outlined in this thesis.

3.4 Complete Design — Phase C

3.4.1 Phase C Integrated Model Design

After successful completion of Phase B and the PDR, the satellite moves into the Phase C, or the complete design phase. The difference from the phases is that while Phase B produces a preliminary design, Phase C concludes with a complete design that identifies all necessary components and devices to meet the mission and system requirements. Any uncertainties or trades should be analyzed and identified by the end of Phase C and prior to CDR. This distinction leads to a greater emphasis on the Phase C integrated model to be tailored specifically to the satellite of interest. While many COTS preliminary design tools exist, few tools exist for a CDR-level design. This is because the tools cannot be as general or modular as in Phase B. However, if the integrated model and simulation framework from Phase B is detailed enough, the upgrade to Phase C requires few major architectural changes. The bulk of the changes will come into the details and lower-level functions of the different elements.

Like Phase B, the complete design phase starts by reviewing any assumptions and the previous model architecture. The same questions asked on page 50 should be asked at this time. Reasons for significant architectural changes to the model design include moving from a COTS oriented preliminary design tool to a more specialized integrated model due to limitations in the flexibility of the COTS software. Additionally, many of the assumed benefits

for a taking a specific approach during the Phase B model design may have been found to be invalid during the coding of the mission elements, and therefore a more reasonable or familiar approach may be desirable in Phase C. Through this process, a plan should be generated for what is the intended integrated model look and feel at CDR, and the steps from Table 3.2 (or the equivalent) should be revised as necessary to complete these tasks.

The supporting elements, or those subsystems and tasks that were identified as important for the mission, but not vital for analyzing key mission decisions, should be added at this time. These elements should have already be parameterized or have assumed values, so at this stage the difference will be making more dependent and integrated relationships that spread across the model. An example of a supporting element identified in the Phase A example from Table 3.1 is the guidance system. The previous models assume a certain position accuracy or power demand for certain subsystem models, but they may not have been integrated as a general GNC model with inputs and outputs like the others. The supporting elements do not necessarily need to be specified at a high level of detail; however, they should behave as their own functional element or object so that if necessary, they can be upgraded independent of the rest of the system model.

The primary elements, secondary elements, and mission constraints should be upgraded as necessary to match the CDR-level of design. As the major system and subsystem trades have been analyzed, the emphasis is put on modeling the specific hardware and component details. This includes incorporating any component testing results into the model to change parameters, as well as establishing detailed models for dynamic properties, or properties that will change with time and that depend on the behavior of other subsystems or elements.

All elements and constraints should be summarized in a detailed N^2 diagram or functional flowchart. Each input and output path should be identified to the lowest level of detail possible. This means that each dynamic property or variable should be identified, to include its units and expected range of values. An example of this is that a property that was previously described as ACS power required may be parsed into component power levels and duty cycles. Also, global variables and their use should be defined and tracked to ensure there are not duplicate variables being called by different functions. The complete N^2 diagram and architecture process is challenging, and it is expected that it will be

repeated occasionally in this phase as the simulation structure is modified.

As the integrated model becomes more detailed to match a more certain satellite design, it is useful to decrease the amount of variables that are parameterized, along with any major unnecessary trade study functions. Although useful in the conceptual and preliminary design phases, few major system trades will occur in Phase C. The emphasis should now be placed to accurately model the specific satellite design and prepare for component, subsystem, and system testing to verify the model in Phase D. Certain elements will need to remain as input parameters, such as the time, orbital characteristics, and system operations decision logic. Simulation characteristics such as step size, run duration, subsystem on/off toggles, and reporting options will also be input parameters through the mission duration.

There are a few different classes of input variables or parameters that would have naturally been implemented in the integrated model design by the end of the Phase B design. The classes of parameters that have been discussed or are inherent to the simulation at this point are:

- Architectural parameters (i.e. major system trades)
- System design parameters (i.e. types of components or systems)
- Simulation parameters (i.e. SIM operating options)
- Flexible satellite parameters (i.e. will change with time)

Architectural parameters were used exclusively in Phase A, and should be removed from the Phase C model if still present. The system design parameters are most likely the design variables used in any Monte Carlo analysis or multi-system design trade. While very valuable in Phase B, in Phase C, few major design changes occur and they require significant analysis due to the increasingly integrated nature of the satellite design. Therefore the ability to easily input these system design parameters should be removed from the model for all systems that have high confidence. Certain subsystem trades may occur and a limited number of system design parameters may stay in the model during Phase C for subsystem designs with high uncertainty. However, by the end of Phase C and for CDR, the satellite design is assumed to be at a high confidence level, so no system design parameter inputs will remain in the simulation.

The simulation requires several input parameters through all phases of the SIM life cycle; however, these inputs will change through each phase. An example of a simulation input that changes is the selection of which subsystems to use in the simulation, referred to previously as "on" and "off" toggling. In Phase A this is not a feature of the simulation, and as elements are being added in the different phases, this input will require more options and user feedback. As subsystem cannot really be turned "off" due to the need for changing outputs for accurately simulating other subsystems of interest, more simulation inputs may need to be created to specify the basic behavior for this system. Lastly, the input parameters described as the "flexible satellite parameters" will be input conditions throughout the SIM lifetime as well. Like the simulation parameters, these will change throughout the different phases. A clear example of this is the orbital properties of the satellite. It is possible for small satellites to not have defined orbits until later satellite design phases. Therefore, in early phases, the orbital inputs will be nearly completely open-ended. Once into the operations stage though, the orbit parameters will be much more solidified, and this input may only be a NORAD TLE file.

Additional input parameters exist that are more important in the later phases of design. These parameters can be classified into the following classes:

- Semi-flexible satellite parameters (i.e. some change expected in life)
- Tuning parameters (or uncertainty variables) (i.e. adjustable for calibration with test or on-orbit data)

These classes of parameters are not as straightforward as those previously described. The semi-flexible satellite input parameters are values that are usually fixed or assumed to be a value; however, the value may change as either software changes, the configuration of the satellite changes in testing or on-orbit, or if new information is found. These parameters are different from previously described input classes because they should not be inputs into the simulation, but rather anticipated updates to the simulation if any major events occur. These parameters should be properly annotated through the coding so it is clear how to maintain and update them in the future. The adjective "semi-flexible" is given as these should not change often, and therefore are not input conditions to the simulation.

The tuning parameters class is used to describe the properties that affect the dynam-

ics and behavior of the different elements. These parameters can be measured or tested throughout the build, test, and operations phases, and should be updated in the model as the value is more certain. An example of such a parameter is the efficiency of a power distribution system. The value can be taken from a hardware specification sheet, but through component and system testing, and even during operations, this value may be seen to change. The tuning parameters are very important in using the IEM in Phase D to verify the design and models, as well as for the TDA once on-orbit. Tuning parameters could also be described as uncertainty variables, as they are modified as the hardware performance becomes more and more certain.

Table 3.3 summarizes the different classes of input parameters by identifying their use, showing a simple example, and describing the phases of the design cycle that they are used in. As noted by this table and described earlier, in Phase C, all architectural and almost all system design parameters should be removed from the SIM. The semi-flexible satellite parameters and tuning parameters should be identified for each component that will be incorporated and modeled in the SIM in preparation for hardware delivery. Once these functions are in place, the SIM can be used to do acceptance testing and requirement verification on hardware components, assemblies, and upon integration, the system. These steps are described in the Section 3.5.1 as hardware acceptance testing and system function testing is typically done after CDR.

Table 3.3: SIM Input Parameter Classes

Parameter Class	**Use**	**Example**	**Phase**
Architectural	Concept Comparisons	Identify # of satellites	A
System design	System Trade Studies	Battery Type	A-B
Simulation	Run SIM as desired	Duration	A-E
Flexible satellite	Dynamic characteristics	Orbit	A-E
Semi-flexible satellite	Match satellite state	Inertia Matrix	C-E
Tuning	Incorporate testing data	Calibrate Magnetometer	C-E

Phase C incorporates one of the more important characteristics in the IEM approach, the command feedback and input feature. As described earlier, the commands generated from the satellite will be fed back into the SIM to give a common base of comparison, or *a posteriori* comparison. This ability may seem arbitrary or unnecessary for certain missions,

but this will be a critical part of the TDA and some MAT designs. This feature gives a wide array of additional capabilities and options for using the SIM, although they come at a design price. The simulation must be written in a manner to complete the expected behavior without the command input, yet be flexible enough to allow the commands to overwrite the programmed behavior. Ideally the design team will have determined the major commands that will be either autonomously generated on-orbit or uplinked from a ground station. The depth of commands that will be incorporated into the SIM will depend on their importance to the mission and their effect on the rest of the system. For example, the command to turn on a transmitter may not be of great importance to the mission, but it may be necessary to model the data transfered, the power required for an amplifier, and therefore an additional thermal heat source. The commands can be separated into the four levels, labeled as Satellite Command Levels (SCL-#).

- SCL-1 — Mission critical (e.g. operate payload)
- SCL-2 — Major system functions (e.g. point satellite or transmit data)
- SCL-3 — Subsystem specific (e.g. dump momentum or perform diagnosis tests)
- SCL-4 — Hardware or component specific (e.g. fire torque rod or downlink file)

The SIM should first incorporate the mission commands (SCL-1). These commands are typically straightforward to integrate into the simulation due to their nature of wrapping around the detailed portion of the model. The SCL-2 system level commands are similar, and can be implemented and overwrite the anticipated natural dynamic of the SIM. However, commands at the SCL-3 and SCL-4 levels can become very tedious to code and may add complexity to the SIM design and its output. One such example is the command to fire a torque rod or move a reaction wheel. While these commands are generated on-orbit and are important in modeling the specific attitude health of the satellite, they add significant computational cost to the integrated model due to their high frequency command laws. To summarize, the choice of the commands to use in the SIM requires a significant amount of planning. For the Phase C SIM, the SCL-1 commands should be included as a default. Certain SCL-2 commands can be included if time permits.

The suggested method of implementing the commands is to utilize the same command

string or command variable as given to the satellite. These commands should also be associated with a time tag to mark their execution time. This list of commands can then be input into the SIM, which can either directly read the commands or go through a command parser, and then when the time for the next command is triggered, the command will be executed in the SIM. When running without command inputs, the SIM will autonomously create and execute necessary commands. This is done for a few reasons. The first is to create the command-run dependency for the simulation to execute, which allows the simulation to run with either internally generated commands or processed satellite commands. Additionally, when using a command input file, the SIM should ignore internally generated commands that would conflict with the loaded command file. When the SIM reads a command input file, it can determine which commands to toggle auto-command generation "off" in the simulation. Lastly, the command-run dependency and autonomous command-generation inside the simulation allows for another source of data to compare the SIM data to the SAT data in the TDA. To clarify the command input and automatic generation logic, refer to Figure 3-8.

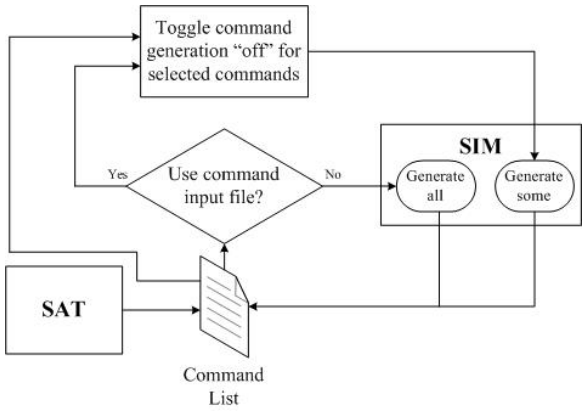


Figure 3-8: SIM Command Input and Generation Decision Chart

This flowchart shows that in all cases, a command list is generated in the SIM output. The decision to use the command input leads to the possibility of automatically generating "some" commands, rather than the default value of generating "all" commands. The command list displayed is shown to be generated by either the SIM or the SAT, however it could be hand-coded or expanded for use on avionics or software prototypes as well. Although the command input feature adds complexity to the model, it allows for greater utilization

of the SIM in the final design stages, testing stages, and ultimately in the operational phase and running the TDA.

The TDA and MAT will be created and working to an initial capacity by the end of the design phase. The plan for creating both tools was outlined in Phase B and should be revisited prior to execution. These plans will specify the intended behavior for both the TDA and MAT. There are some basic requirements that are general to all TDA designs in the IEM approach.

1. The TDA must analyze the TLM and simulation output for readings out of operational or survival range
2. The TDA shall be composed of subsystem and system processing tools as necessary for the given mission
3. The TDA must have data-correlation features and the ability to update specified tuning parameters for the SIM
4. The TDA must be able to launch the SIM in specific subsystem testing modes

The first of these requirements is that the TDA must analyze the telemetry and simulation data for readings that are out of the operational or survival range. This is a basic requirement and is easily met by either a Graphical User Interface (GUI) that color-codes TLM values or by automatically generating a report that identifies the time and values for out-of range readings. Common GUI color coding is to display in-range readings as green, to color readings that are approaching their boundaries as yellow, and readings out of limits as red [16]. SMC Standard Practice HM-RB-2001-1 is a useful document if starting a human computer interface from a relatively new starting point.

The second requirement for the TDA to contain TLM processing tools ensures that the analysis and processing tools often used only by a specific analysis team are available in the TDA. Certain analysis tools may not be necessary to include in the TDA if they are not important in the integrated behavior of the system. Such an example is a structural loads analysis tool in a system that has no on-orbit mechanical operations. An example of a common processing tool that should be used in the TDA is an attitude filter that reads attitude sensor TLM and outputs the pointing angles and rates of the satellite.

The third requirement is to have a data-correlation feature for the SIM tuning param-

eters. As defined before, the tuning parameters are the conditions that often are assumed or not entirely certain, and the TDA is used to verify the actual value of these parameters through ground-based and space-based testing. The data-correlation feature is necessary for each test or series of data that is expected to use the integrated model and simulation for verification. This is a major decision for the design and testing team, as testing is expensive in cost, manpower, and time criteria. A tolerance must be specified for each tuning parameter to ensure that the results meet the expected accuracy and are physically feasible. Note that care should be taken in adjusting certain tuning parameters to match data, particularly for autonomous modification or data fits.

Lastly, the TDA should be able to launch the SIM in a specific configuration necessary to effectively compare detailed subsystem data for verification tests. An example is for analyzing the expected thermal conditions of the satellite, the attitude and power values of the satellite fed into the TDA will become inputs into the SIM, which would run with the attitude and power systems toggled "off." As there are significant programming challenges with these increasingly complicated input-types to the SIM, it is important again to recognize which tests or analysis are expected to be verified with the SIM. These details should have been specified in Phase B in the TDA plan, and should be revisited and finalized prior to TDA creation.

The initial MAT plan was also created in Phase B. This plan describes the specific functions necessary for detailed mission analysis. The primary example that is used in this thesis is the thrust characterization of a low-thrust electric propulsion system. The inner-workings of the MAT for this example include a high-precision orbital propagator as well as a thrust and efficiency characterization tool based off of both direct measurement and position-based extraction. This specific example may also include a thruster plume image analysis tool to determine the thruster efficiency from the characteristics of the plume during operations. In Phase C, this plan should be executed to produce a working MAT with the first iteration of the desired functions included. By CDR, the MAT will utilize the inputs from the TDA and SIM to output a validated mission as identified in the figures of merit.

By CDR and the end of Phase C, the set of tests necessary to verify the MAT's analysis of the expected telemetry from any functional prototype, QM, or ground-based FM testing

must be identified. Ground-based testing of space-based payloads is almost always limited due to the desire for the payload to work in the space environment, so it is very likely that the ground-based functional testing will not be able to simulate the full range of expected telemetry from the satellite. The SIM is purposefully designed to allow for an alternate source of comparison for certain impossible or very costly tests (e.g. six degree of motion dynamics or microgravity environment), but the mission cannot be validated solely by the SIM performance. This is a fundamental problem for testing spacecraft, which is why it is important to decide the scope of the Phase D testing on the design of all IEM elements.

The integrated model status at the end of Phase C is summarized in Figure 3-9. The major additions since Phase C are the MAT and TDA are real elements and not just plans (shown as having a solid outline), the command (CMD) input from the SAT to the SIM, the incorporation of supporting elements (s_1), and the increased detail in the other elements. The dashed lines representing the commands, telemetry, and simulation data all signify that these sources of data are not finalized. The SIM to MAT arrow is solid as the IEM will still perform mission validation functions, as it has in every other phase.

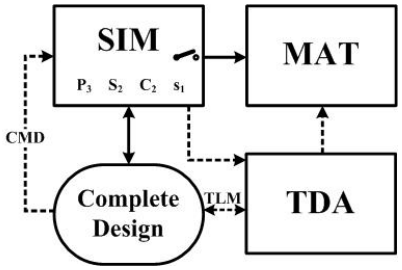


Figure 3-9: Phase C IEM Design Diagram

3.4.2 Phase C Integrated Model Example — CASTOR

A CASTOR integrated model was created in the similar framework as the MOTV model described in the Phase B section, through the Fall 2009 16.851 Satellite Engineering class. The authors of this model were all graduate students from the MIT Space Systems Laboratory who were working on the CASTOR satellite. These members included 2LT Matt McCormack, 2LT George Sondecker, 2LT Corey Crowell, and masters candidate Ryan McLinko. The task was to update the previous MOTV model to many of the changes that occurred

in the nine months of work to what became the CASTOR satellite. These updates included changing components used in the design, incorporating additional supporting subsystems, and adding detail. Due to the timing of the class with the CASTOR design time line, the CASTOR integrated model encompassed the end of Phase B and the beginning of Phase C.

Although this simulation was created prior to the writing of this thesis, it added many features described in the Phase C IEM design section. These features include adding a command input capability into the simulation, modeling supporting elements, and restructuring and simplifying the SIM interactive behavior. The CASTOR simulation allowed for the equivalent of SCL-1 commands and select SCL-2 commands to be triggered at specified times. This process was not designed specifically to the method described in this thesis, so there were no other commands automatically generated during the simulation processing. The avionics system, which can be described as a supporting element, was added into the CASTOR simulation as it was not present in the MOTV simulation design. The communications, ADCS, and thermal models were updated to be more detailed and match the CASTOR design much more closely. Lastly, as Figure 3-10 displays, the GUI was simplified to be only one screen, as opposed to allowing a whole screen for each model element as done in the MOTV design. This was possible as the design was more certain and unnecessary model inputs were allowed to be removed.

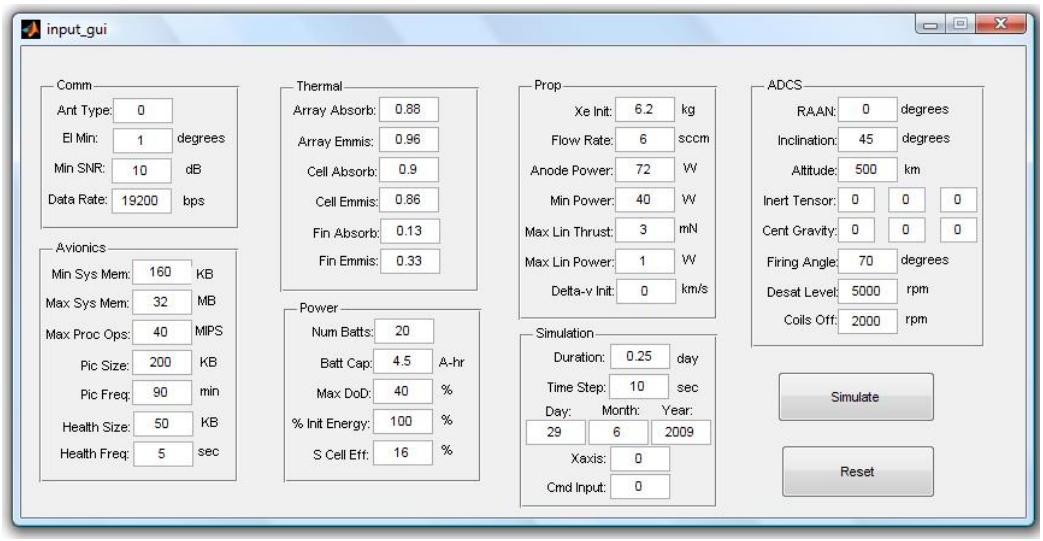


Figure 3-10: CASTOR User Interface

Another major update to this simulation was the programming structure. Although the simulation was still run in MATLAB, the programming style was transfered from the functional decomposition architecture to object-oriented programing (OOP). This was done to increase readability and code organizations, as well as to simplify portions of the coding. The simulation also output an event list that described when components went out of their expected or operational range, which is a function performed by a TDA. The simulation also produced limited mission assurance output in measuring the different expendables and tracking the thruster performance and total firing time. While neither a TDA or MAT were specifically created or distinguished, they were integrated in the design of the simulation.

The portions of the IEM Phase C model that were not done in the CASTOR simulation include the switching of subsystems, the command generator, or a TDA design that allows data and simulation output comparison. The results and discussion of the initial CASTOR integrated model can be found in Section 5.1.3.

3.5 Implementation and Testing — Phase D

3.5.1 Phase D Integrated Model Design

The completion of Phase C is a vital point in the satellite design cycle as the team will transition from the design phase to the manufacturing, assembly, integration, and test phase. Completion of Phase C assumes a technically feasible and validated design, but it does not assume verification of the components or the integrated performance of subsystem and system hardware. Therefore the primary goal in Phase D is to successfully execute the design identified in CDR by performing bottom-up verification during the assembly process. The integrated model is very important throughout this phase, as it will utilize the testing data to verify the hardware functionality, validate the mission, and fully prepare for SIM for the operations phase.

To fully prepare for test data, all SAT elements modeled in the SIM must be updated. This includes updating any changes to hardware specifications, eliminating any unnecessary model or simulation components, and finalizing any interface properties between subsystems or components. Few changes in the satellite hardware occur in Phase D, so these updates to

the SIM prior to testing will be minimal. The largest satellite development task that occurs in Phase D is the implementation of the software design. Therefore the largest updates in the SIM will be incorporating any satellite software changes or additions. It is possible and may be advantageous to execute the flight software code in the SIM. This is not a requirement as the primary task of the SIM is to model and simulate the satellite's performance with minimal effort. However, for certain tasks such as testing new controllers, it is valuable to use the flight software directly in the SIM as the software is already written. Therefore the major trade becomes whether it is more desirable to spend resources on creating the interface between the flight software package and the SIM software, or if it is more desirable to modify the SIM software to mimic the flight software behavior. This decision will primarily depend on the team's familiarity and comfort with the different types of software.

The command list and its input and output interfaces with the SIM will also need to be revised throughout Phase D. Revisions include format changes to the command structure, addition or removal of commands used in the SIM, and changes to the command parser. For formatting changes, it is important that the SIM commands will be continually verified to match the satellite commands in both format and meaning. These changes will likely occur frequently, therefore the SIM command functions must be vigilantly maintained to ensure changes to the satellite commands are accurately represented in the SIM.

In the IEM designs for Phases A-C, the SAT block was replaced with conceptual, preliminary, and complete designs. In Phase D, the SAT block can be represented by a real, physical system for the first time. This satellite system is broken into three building blocks: the satellite flatsat, a qualifications model, and a flight model. The flatsat, which also can be referred to as an Avionics Test Bed (ATB), models the satellite design sans structural elements. That is, all components with data and electronic interfaces are integrated together. This is the first version of the integrated satellite system, and allows for the greatest flexibility in running with the SIM and performing hardware tests.

The Qualifications Model (QM) is the first revision of the complete integrated satellite. It includes the flatsat components and the structural elements, and usually at least one each flight components for each unit. The QM is used for system integration testing and environmental qualifications testing, as well as performing a "trial run" for the flight model

integration. The QM is a useful ground-based testing platform during operations as it has almost the exact same hardware and components as the FM. If a satellite programs cannot afford a QM, it is strongly suggested that a flatsat or ATB is produced for ground-testing any software upgrades that may come during Phase E.

The Flight Model (FM) is the final, completed satellite that will be launched into orbit. This model will be tested to acceptance levels for environmental testing, as well as similar avionics and software testing as done with the QM. It is important to distinguish any differences between the FM and QM hardware, as the QM will be the only tangible representation of the FM upon launch. The QM and FM must use the same flight software to ensure that any functional testing of software on the QM will respond the same as it will on the FM.

The IEM development emphasis is different for each hardware configuration (ATB, QM, FM). Therefore the rest of this section will be separated into a different section for each of the three hardware phases, labeled D.1, D.2, and D.3, respectively.

Phase D.1 — Flatsat/ATB Development and Component Testing

As components arrive and are tested, the SIM and TDA will be used for mission and hardware verification purposes for the first time. This process is iterative, as shown in the following steps of the hardware and SIM verification process:

1. Predict single-scope hardware test result based on expected component characteristics
2. Perform component tests to compare and verify component characteristics
3. Input characteristics into SIM and TDA as applicable
4. Test component in integrated matter using SIM (if possible and necessary)
5. Use TDA to compare hardware performance
6. Modify any SIM or TDA characteristics (e.g. tuning parameters) as necessary to match reality
7. Repeat as necessary in different operational modes (if applicable)

This sequence is used as part of the requirement verification testing that will already be in place for the component, only it also updates the SIM and TDA in the process as

well. This is advantageous as it documents the actual design in the SIM and begins to modify the tuning parameters to match the actual design. This experience will also assist in understanding what telemetry is used and how it will be analyzed in the TDA.

An example of this process is the verification of a battery charging unit. A basic verification method used is to measure the voltage and current coming from a power source or solar panel simulator and compare that to the power being provided to the batteries. The most basic test is to show that the battery charging unit fully charges the batteries. However, much more detail can be given in this test including the behavior of the power efficiency and heating profile of the charging device and as well as the change in current provided to the batteries as the cells change in temperature. The timing of this sequence is also important in order to verify that the batteries can be charged to their intended amount in the sunlight portion of the orbit. If it is significant enough to be modeled, the battery charge dynamics through the sunlight and eclipse cycle will be output in the SIM, using the basic efficiency results from the simple test. The SIM test conditions can then be performed in ground testing on the battery charging device to have a similar source of comparison in the TDA. The TDA can then be used to identify any improper assumptions or correlate simulation parameters to testing data.

It may be clear that this process can grow to be very expensive. Therefore, it is important, as stressed throughout this chapter, to decide the importance and level of detail of each test in Phase D. A general rule is that as soon as the component or system has met its requirement, the team should move to the next step, barring any anomalous behavior that may affect the integrated performance of the system.

In order to perform these tests and gain telemetry to match that used in the TDA, the avionics system and flight software must be integrated and working into the ATB. The ATB does not need to have every component and software element integrated to perform the initial component verification function. However, the ATB does need the following two basic tasks, particularly for the IEM framework:

- Synchronize clock with desired time (UTC format recommended)
- Sample, store, and downlink telemetry from components in expected format

The majority of the flight software is written during flatsat development. Depending on the role of implementing actual flight code (described in the previous section), this stage can lead to many disconnects between the software on the satellite and the SIM assumptions. Once again, it is important to stress the criticality of documenting changes to the flight code and its decision making logic on that in the SIM. This is especially important for software functions that have limited ground performance testing, as often only a single data point exists to validate the functionality of a line of flight code.

In each subset of Phase D, just like all previous phases, the integrated model must successfully validate the mission. The major difference with the task in Phase D.1 from that in CDR is that the difference in expected performance is no longer coming from design changes but rather from a difference in the expected performance of the hardware and the actual performance. This mission validation process may be trivial to run if the only changes to the SIM are in tuning parameter values. If this is the case, the Phase C SIM and MAT can perform the MA function and give the expected mission output. Even if this process is trivial, the output may not be; the integration of component testing and verification data may lead to degraded performance measures. If this is the case, the team must decide whether to accept a loss of performance, or to try to mitigate this loss with either new hardware or different operating characteristics.

The last IEM feature that needs to be updated in the Phase D.1 is the MAT data analysis function. The expected telemetry providing the mission assurance details, such as the thrust details and images given in the CASTOR example, may be created by their respective hardware units at this time. The MAT must be able to read and process the data from these components or assemblies and perform their intended mission assurance functions. Specific analysis and processing functions in the MAT will be updated throughout Phase D as the data output is understood more in the framework of the integrated system.

As more components are incorporated into the flatsat, the integrated performance of the flatsat can be tested following the same procedures as with individual components. The SIM eventually will be directly fed by the flatsat commands. At this point, the satellite enters Phase D.2.

Phase D.2 — Qualifications Model Development and System Testing

The difference between Phases D.2 and D.1 is not as obvious as those between the other major phases, due to the lack of a scheduled review to signify the end of the phase. The systems testing often will take place during the flatsat fabrication and testing phase, leading to confusion in distinguishing these phases. The primary distinction for the purpose of discussion and comparison in this thesis is when mission and system level commands (SCL-1 and SCL-2) are given to the functional satellite model, whether the ATB or QM, and these commands lead to a response by the majority of the satellite hardware elements. Component verification will still occur as additional parts come in, but the primary emphasis is on the systems testing. The components verification still will be performed by the instructions given at the beginning of Phase D.1.

The command input to the SIM will be used frequently in Phase D.2 to verify the system performance and behavior of the subsystem components. The system testing must be done after verifying the component, subassemblies, assemblies, and subsystems in a sequential manner to properly tune the correct parameters and identify the culprit for any unexpected behavior. If used throughout this entire process, the command input tool in the SIM will automatically be validated through the full range of commands expected. The benefit for the programming of this tool will be apparent when open-loop commands are given to the satellite testbed, and the performance can be analyzed in real-time with direct measurement units such as multimeters or analog sensors, along with post-processing analysis using the SIM and TDA results generated from the command file input and test telemetry.

Upon completion of the QM, the SIM will be used to perform a mock operations sequence. Although not all systems will always be available or working due to limitations in hardware or delayed delivery of flight software, it is important to identify how the SIM will be used in the operations phase. At this point, it is advantageous to involve other portions of the mission such as the ground segment hardware or operations team to test the SIM and QM. The SIM can be used to run any early mission tasks, particularly subsystem health verification tasks, and then the QM and TDA can be used to verify that the SIM output produces the results observed and expected. These procedures will be updated and

improved as necessary and repeated in the final acceptance testing of the flight model.

Lastly, the mission assurance tool should once again be revised and run with the SIM to ensure that the satellite still meets its intended mission. Upon successful validation of the mission using a fully verified SIM, the team is prepared to build the final flight unit.

Phase D.3 — Flight Model Development and Final Testing

Phase D.3 will execute the same procedures used in Phases D.1 and D.2 for both components, subsystems, and the integrated system. It is recommended that the QM SIM from Phase D.2 is kept separately from the FM SIM, as the hardware may have some slight differences. At each stage in the final FM component verification, the results should be compared to the QM SIM. This may be as simple as keeping the QM tuning parameters saved and readily available to run into the TDA against the FM telemetry. Any difference in performance between the FM and QM must be annotated clearly and in detail to ensure that any ground-testing during operations includes the differences.

The same initial operations tests will be performed on the FM to ensure that any addition of hardware (e.g. three reaction wheels instead of one) properly works in the integrated system. This is also another time to incorporate alternate mission elements into the testing to perform another dry-run for the post-launch operations.

An additional requirement in Phase D is that the IEM operational users guide needs to be completed. Portions of this document should have been created in earlier phases (as described in page 39) to ensure that the team could effectively utilize it as a design tool.

Phase D is summarized Figure 3-11. This figure shows the major change in the SAT block to go from a completed design to a series of real-life systems. Additionally, it shows the TDA in between all of the other elements, as it takes a primary role in the verification of the satellite hardware. The telemetry and command lines (TLM and CMD) are represented as different lines than before as the data can be transmitted wirelessly to the ground station, or through a wired RF link straight to the satellite. There is a two-sided arrow for both of these as the output from each can still affect the design and configuration of the satellite. In the SIM block, each element has been incremented in its detail level to represent the additional details added to the components and finalizing any subsystem behavior modeled.

Lastly, the line between the TDA and MAT is solid as the MAT is now completely functional and will produce its expected output when satellite telemetry is input.

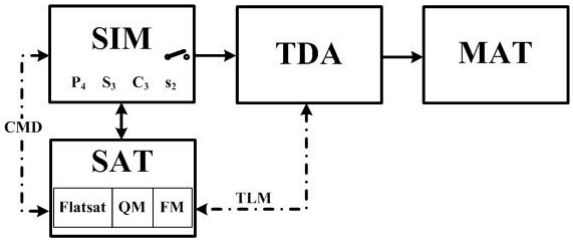


Figure 3-11: Phase D IEM Design Diagram

3.5.2 Phase D Integrated Model Example — CASTOR

A Phase D model does not exist at the time of this thesis. The development of the Phase D CASTOR SIM is the primary topic of Section 4.1, and the development of the CASTOR MAT through a low-thrust trajectory tool design is the discussed in Section 4.3.

3.6 Operations — Phase E

3.6.1 Phase E Integrated Model Design

The purpose of all IEM design efforts prior to launch was to maximize operational efficiency and productivity in the mission. Ideally, the satellite arrives into orbit in perfect health and functions as expected, with the SIM producing results that are shown by the TDA to match the satellite telemetry to the expected levels. Upon on-orbit checkout and commissioning of all subsystems, the MAT can be used with the entire simulation and analysis package to display the success of the mission.

However, this is a very ideal case and is not plausible due to the number of uncertainties that will still be looming at launch. It is important to anticipate these challenges and coordinate proper manpower to the operations phase in order to properly manage the satellite and all the analysis tools available. The first area to expect challenges is in the on-orbit checkout and commissioning of each component and sub-system. It is vital to completely verify that the components and software is performing as expected. Through the IEM sequence and Phase D testing, this should have been accomplished no less than two times,

and likely will have been done with the QM in the final operations readiness sequence prior to launch. All of the capabilities built into the SIM and TDA to include command input, subsystem switches, and tuning parameter calibration, were designed to facilitate smooth debugging by identifying the root cause of any anomalous behavior that may be observed in commissioning.

If any hardware element is found to be unresponsive or perform to a limited capability, the SIM can be updated to match the new satellite state. Depending on the hardware element, this can be either a simple change in a parameter, or it may involve modifying many of the underlying logic elements in the code. Assuming that on-orbit repair is out of the realm of any small satellite, the only option available for correcting the errors is through the modification of the software or CONOPS. If the CONOPS is changed, the SIM must be updated if it is intended for any future prediction capabilities, though it may not be necessary to update the SIM if the primary mode is to run it via the command input.

Even if there are no major failures, significant software modifications should be expected and anticipated. As soon as desired change in the software or operations of the satellite is specified, the SIM must be used to show that the change will perform the intended function. After the SIM prediction, the change will be implemented in the QM, and the output will be compared to the SIM with the TDA. After successful verification of expected behavior on both the SIM and the QM, the new software or commands can be uploaded to the FM. It is recommended that this is done for relatively small changes in software or command processes as well, as small changes in the flight code will be relatively simple to implement in the SIM.

The primary task for the integrated model during Phase E is to continually monitor the bus performance while also measuring the mission success and tracking the payload performance. The TDA will be used to monitor the bus performance and produce reports on the health of certain systems. The space environment is not constant and sensors are not always perfect, so it is possible to show very accurate results in the TDA one day and a disparity a few days later. It is important to not over-analyze the tuning parameters in the TDA and understand the accuracy of the sensors and the fluctuations in the space environment. One such example is calibrating magnetometers to the magnetic field of the

earth in LEO. The IGRF magnetic model of the earth will not perfectly match the actual magnetic field vector at any given location in orbit as it fluctuates with time, and the magnetometer characteristics may change with temperature. Both of these can cause the magnetometer strength to not match the IGRF field strength at any point in time. This specific example is explored in Section 4.2.2. The mission success measurements will require effort and vigilance as well. The MAT is used in the mission operations phase to measure and track the mission performance data and ultimately to measure that the satellite has met its objectives. Changes in the SIM or operational plans can be analyzed to see how they affect the MAT trends.

Any updates or modifications to any elements of the integrated model in Phase E should be documented and saved for later projects. While this feedback loop is not shown in any of the diagrams, it is beneficial for the organization to increase their corporate memory of the IEM process so that it can be done easier and at a lower cost for the next mission. If a follow-up satellite is expected to be produced, it is very important to identify what elements could not be tested or successfully modeled prior to launch.

The Phase E IEM diagram is shown in Figure 3-12. This shows each block as working and real elements in the process. The only lines that are not solid are those that represent the telemetry and commands coming from the satellite and going to the TDA and SIM, respectively. Both of these arrows have changed to be uni-directional, as the satellite is not changed as it had been in previous phases. The different elements and their detail level has been removed, as the SIM is in its intended final level throughout the Phase E design. The switching element is still present in the SIM. The TDA is shown to feedback into the SIM for the first time, signifying that this is the only available feedback path to match the SIM output with the data. Software uploading capabilities are not shown in this figure as they are not directly a part of the IEM and are intended to be minimized.

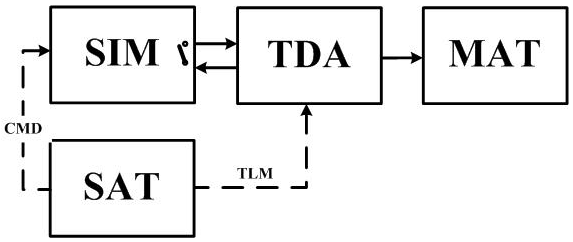


Figure 3-12: Phase E IEM Design Diagram

3.6.2 Phase E Integrated Model Example — FalconSAT-3

The development and application of an on-orbit modeling and simulation tool in the FalconSAT-3 satellite is described in Section 4.2 . This section primarily focuses on the TDA design, as no integrated model exists at the time of this thesis.

Chapter 4

Applied Integrated Model Design

This chapter describes the implementation of the different software elements in the IEM design (SIM, TDA, MAT). Many elements of the Phase D IEM design are implemented in the MIT CASTOR satellite simulation and are described in Section 4.1. This section details the scheduling aspect of the simulation necessary to meet the IEM principles, describes the handling of different model and simulation data, and explores many basic subsystem models necessary for full system interaction in the simulation. Section 4.2 uses the USAFA FalconSAT-3 satellite and its data to explore the design and implementation of a TDA created during the third year of on-orbit operations. Section 4.3 baselines the design of the CASTOR MAT.

4.1 CASTOR Simulation

The CASTOR simulation described in Chapter 3 was revised to provide more features inherent to the IEM design. The main features added or modified from the legacy simulations (see Sections 3.3.2 and 3.4.2) are the scheduling, data input, and subsystem detail level. The scheduling task was added to create a robust and flexible command-based and event-based simulation, and is described in Section 4.1.2. The data input capabilities and a description of the methods for passing data, values, and states throughout the simulation is described in Section 4.1.3. A high level overview of the entire simulation structure and its input and output behavior to a planned CASTOR TDA is described first in Section 4.1.1.

4.1.1 Simulation Structure

The CASTOR IEM-based simulation overview is most clearly described through a user's point of view. This section will look at the simulation from the most external structure, and work into the handling of information and conclude with a full description of the model information flow via a N^2 diagram.

As mentioned in the IEM description, two general types of information can be output from the simulation: *a priori* and *a posteriori* behavior of the satellite. The difference between the two is the use of actual commands that the satellite executed. While constructing the simulation, it became clear that two forms of *a posteriori* results exist. There are three main types of SIM outputs that are useful to analyze, shown in Figure 4-1 as S-1, S-2, and S-3.

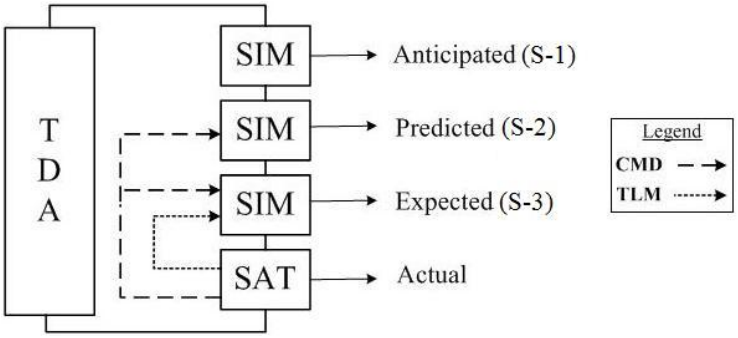


Figure 4-1: Level of Comparison between SIM and SAT Results

Figure 4-1 associates names with the S-1 through S-3 levels to attempt to distinguish the level of certainty in the results based on the SIM inputs. The first SIM instance, S-1, represents the *a priori* case. This is described as the anticipated case, and does not use any flight commands or telemetry. The next instance, S-2, is the *a posteriori* case with commands only. This is described as the predicted case, and the SIM will produce an output that should match the flight telemetry much closer. Lastly, the S-3 case is another *a posteriori* case, with the addition of telemetry being fed into the SIM. This is described as the expected case, as it is expected that the SIM output will match the actual results (TLM). The different cases each have important applications to the mission in the TDA architecture. The S-1 instance is used in all prediction cases, and is also useful in comparing

expected commands generated versus those from the satellite for post-processing. The S-2 case is useful for matching predicted and actual telemetry. Lastly, the S-3 case can be used for verifying one specific model, by using actual telemetry values in place of typical subsystem models.

Therefore the SIM must be able to generate results for a flexible set of input parameters. It is the job of the TDA and the user to select the proper settings for the SIM. The rest of this section is based off of one generic case, meaning that the SIM is executed only once, with the understanding that it is possible to run the SIM multiple times in the TDA. The input parameters to the simulation, including all types specified in Table 3.3, are assigned to the `inits` structure in the SIM MATLAB code. This structure is defined in Table 4.1.

Table 4.1: Inits Structure Breakdown

Nomenclature	**Description**
`inits.state`	Satellite and environment states
`inits.data`	Any telemetry or data to pass into satellite or subsystem objects
`inits.values`	Initial conditions for satellite subsystem properties
`inits.sim`	Simulation settings*
	*Includes time, command list, event list, and scheduler settings

The SIM and most of its underlying elements are coded using Object Oriented Programming (OOP) techniques. The very basic behavior of object oriented programming and design is to create so-called "objects," which have two attributes: properties and methods. Properties are values, data, or states, and are associated with the object. Methods are functions that are specific to an object, and represent the dynamics and behavior of the object. Many sources exist for a complete description of OOP, and MATLAB has an assortment of documentation and examples available on the Internet [9]. While helpful in understanding certain nomenclature used, OOP understanding is not vital in this thesis. One method that all objects use is the constructor function. This function instantiates, or creates, the object. In the case of the simulation, the constructor function instantiates the command list, event list, scheduler, and satellite objects as well. All of these are saved as properties of the simulation object. Of these, the satellite object is the only property of the simulation that has additional objects to instantiate. These objects are the different subsystems of

the satellite that are modeled in the simulation. Therefore the `inits` structure described previous is used to instantiate all of these objects. A graphical description of the SIM layout is shown in Figure 4-2.

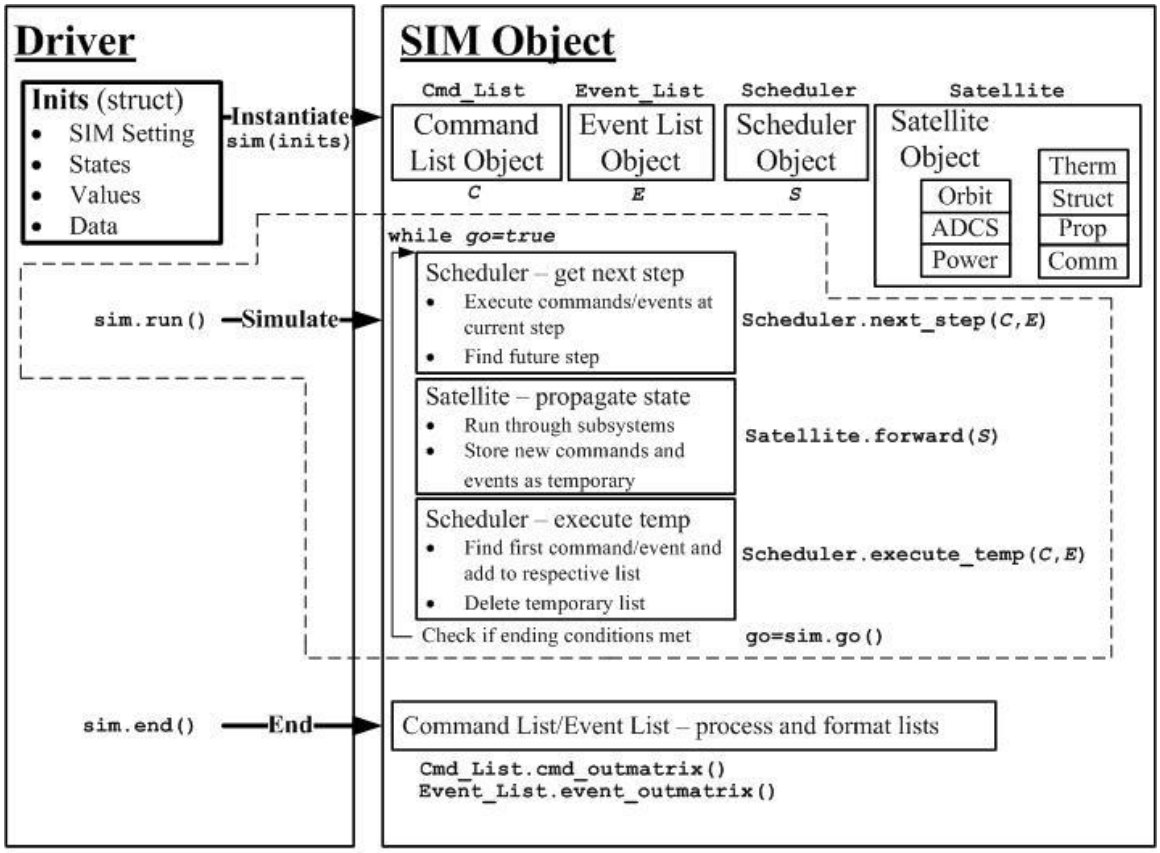


Figure 4-2: CASTOR Simulation Layout

Figure 4-2 outlines the major run function in the simulation with a dashed line. The `sim.run()` represents the syntax of the simulation method called "run." This method is called by the driver script or object (such as TDA or GUI), and then the simulation runs through the loop identified in the right half of the picture. The loop's stopping conditions is controlled by the local variable `go`, which is computed in each simulation step with the method `sim.go()`. The while-loop begins with executing any commands or events that occur at the current time, and determining the intended length of the next simulation step. This process, the `Scheduler.next_step()` method, is discussed in greater detail in Section 4.1.2. The simulation may have a fixed step size or a dynamic, event-based step duration. The scheduler object uses both the command list and event list objects, and this is the only

point where the satellite state is modified.

Upon completion of the `Scheduler.next_step()` method, the satellite object's method `Satellite.forward()` is called. This method calls in the scheduler, which contains information of the current expected simulation step size, and any subsystem step sizes needed to run through the subsystem models. At this point, the subsystem models are called as necessary to propagate the satellite's state forward. The N^2 diagram best explains this flow of information through models, and can be seen in Figure 4-3. This diagram only shows the dynamic properties that will change. Any state changes, component or system values, or other properties are handled internally in the subsystem models or externally via state changes called from the scheduler object.

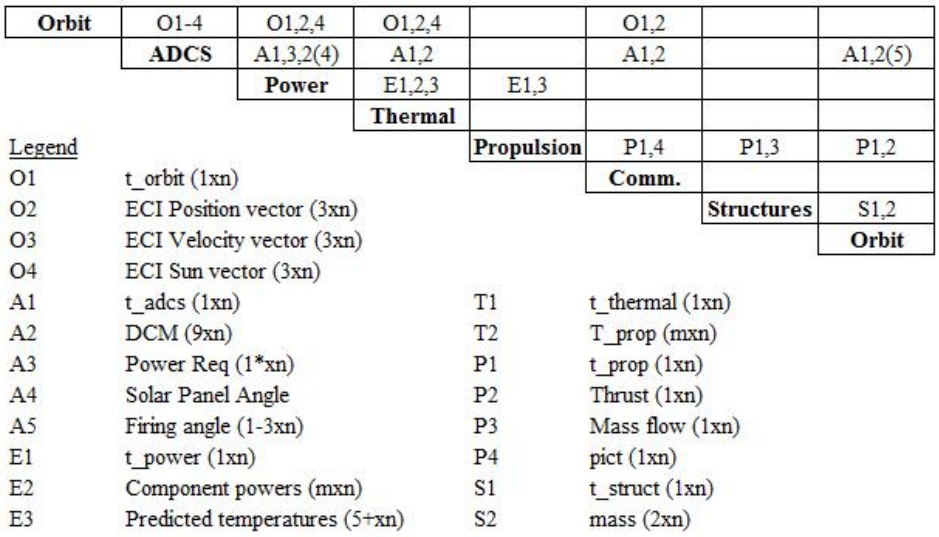


Figure 4-3: CASTOR Satellite Simulation N^2 Diagram

The passing of data between the models is explained further in Section 4.1.3. In the propagation forward, any new events or commands triggered will be passed through to the scheduler and saved into the property `Scheduler.temp`. When the `Satellite.forward()` method is completed, the simulation calls the `Scheduler.execute_temp()` method to add the new commands and events to the proper list for calling at the next time step. The simulation and scheduler settings for fixed or dynamic step sizes will affect how accurate the timing is executed.

After the necessary commands and events are added to their respectful lists, the property

`Scheduler.temp` is cleared. This ensures that no commands or events are executed multiple times. At this point, the simulation runs an internal method `sim.go()` to determine if the exit conditions from the loop are met. These exit conditions can include reaching the simulation end time, running out of a consumable, or being interrupted by a user. If the `go()` local variable returns "false" (0), then the simulation object will return to the driver script, which will execute the `sim.end()` method. While it is possible to add this to the `sim.go()` method, it is left in the driver script in case there is any potential to have a user call it directly. The `sim.end()` method currently only formats the command list and event list to have readable summaries. This method could have other autonomous data or simulation operations if desired to be run at the end of every simulation. It is assumed that the analysis and representation of data (via graphs and reports) is handled by the TDA.

4.1.2 Scheduler, Command List, and Event List Objects

The first revision of the CASTOR simulation described in Section 3.4.2 included the ability to use an external command file to alter the satellite's state in the simulation. This was done by creating a command list object, which stored the commands from the input list, and a command object, which collected the commands for each time interval t_i to t_{i+1}. In this simulation, a constant time step of $\Delta T = t_{i+1} - t_i$ was used. At the beginning of each time step, the command object was created and passed into each of the subsystem models. The individual models were written to search for specific command strings in the current command object. If the command string was found, the satellite state was changed.

One such example of the use of commands is the operation of the thruster. A variable `thruster_on` existed and was assigned in the propulsion model. If the command list was being utilized, the propulsion model searched the list for the correct string; if no command existed in the current time step, `thruster_on` was assigned as false. If no command list was used, a set of criteria based on the orbit was used to determine the value of `thruster_on`. This logic is straightforward and accurate when implemented using certain assumptions. However, these assumptions do not meet IEM design principles and objectives and constrain the flexibility of the simulation. In the new design, the command-based nature of the simulation led to a change in the general model structure. A new scheduling task was

added to allow for the event-based execution of the SIM.

By using a scheduler and a strict event-based and command-based simulation, the assignment of the `thruster_on` variable is done at the precise time that it was sent. If no command file input exists, the propulsion subsystem model will perform the same logic to ensure that it is the proper time to operate the thruster, and at that time, a command will be sent to the scheduler and eventually passed into the command list object.

A graphical depiction of the implemented scheduler and command based design is shown in Figure 4-4.

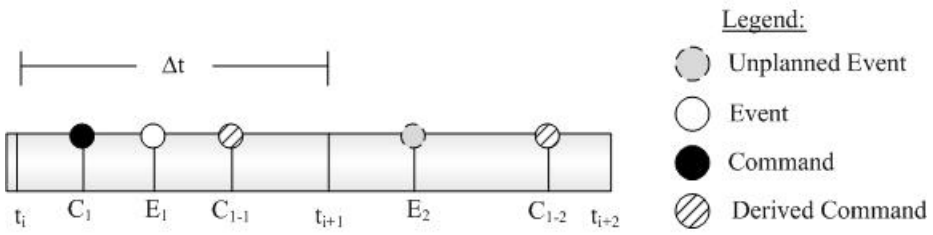


Figure 4-4: Scheduler and Discrete Event Diagram

This figure shows a fixed time step of ΔT, which was assumed in the previous simulations. While this option is still available, the discrete event approach is suggested. In both methods, the scheduler will determine the future time, called t_{next}, that the satellite (and therefore each subsystem model) will be propagated to. In the fixed ΔT case, t_{next} is simply $t_i + \Delta T$, or as Figure 4-4 shows, t_{i+1}. In the discrete event approach, the scheduler decides t_{next} based on when the next event (command or state change) will occur. If no interruptions occur (i.e. no new events or commands), the scheduler simply identifies the next event and propagates forward to this time and repeats. In Figure 4-4, this means that the simulation time would go between $t_i \rightarrow t_{C_1} \rightarrow t_{E_1} \rightarrow t_{C_{1-1}}$. The unplanned event, E_2, would interrupt the simulation's satellite propagation method while it was propagating to $t_{C_{1-2}}$. The interrupt will be sent to the scheduler object, which will add the new event or command to a temporary list, as well as reassigning t_{next} to be the interrupt time. The satellite propagation method will continue with a new t_{next} value, and any other interrupts will continue in a similar fashion until all models have reached t_{next}. At this time, the scheduler will add the new event or command to the respective list, and then will continue

in the while loop outlined previously in Figure 4-2. Figure 4-5 on page 88 clarifies this interrupt process further after discussing the use of time steps and data flow in greater detail.

The derived command is another addition made possible with the new simulation structure. Certain high level commands, such as the firing of a thruster, will produce a sequence of additional commands that take a finite amount of time to execute. An example is turning on the Xenon flow to the DCFT prior to providing power to the anode. This finite time, if known, can easily be added to the command callback method. This is the function executed in the simulation that changes the satellite's state based on the execution of a command. Any additional commands (called derived commands) can be added to the command list. If the delay time is shorter than the simulation step size (fixed time step), or zero (discrete events), then the command will be executed at the same time as its parent.

The event list object performs very similarly to the command list object, with the distinction that events do not directly change the behavior or state of the satellite. When executing the event list callback methods, any state changes for the satellite are called by executing commands. In the case of operating the thruster autonomously when a certain orbital event is triggered (such as reaching the desired firing angle with respect to local noon), the event will add the `thruster_on` command to the command list, which will then be executed using the command list callback method as previously discussed. Events can trigger state changes to the satellite's environment; however, all changes to the satellite state that would occur in flight software or through ground commands will occur through the execution of commands.

The advantages of using discrete events is that the timing is much more accurate, and it increases the computational efficiency of the simulation by not needing to save the state at a fixed time step. Additionally, this method is improved over the initial CASTOR model as there are fewer assumptions in the execution of the commands. Previously, the start and stop times for each command needed to be defined, and in the new method, only start times are necessary, which is more realistic to the actual format of flight command files.

4.1.3 Subsystem Models and Data Distribution Overview

The scheduler object has three major output properties that are used by all systems: the current time (t_i), the future time (t_{next}), and the subsystem step sizes (δt_{sub}). The current time and future time, as described in the previous section, are inputs to each subsystem function, and are used to propagate the state forward. In the discrete event case, if a new event occurs between these times, an interrupt is triggered and the simulation returns to the scheduler with a new command or event being added and causing the t_{next} property to be changed.

However, to put this into practice, it is necessary to delve into the details of the subsystem models. Satellite subsystems can have much different response times and properties, requiring significantly different step-sizes during propagation. Certain subsystems are linear and can do the entire propagation in one step (such as modeling fuel mass using a constant flow assumption), whereas other subsystems require a high frequency sample rate to ensure accuracy (e.g. ADCS). This is where the parameter δt_{sub} is used. This parameter is the step size used within the individual subsystem model. It allows for the simulation to retain accuracy in the high frequency parameters, while allowing for long duration simulation runs.

Figure 4-5 illustrates the different δt_{sub} steps for the different satellite subsystem models for propagating to the time of the next command. This diagram "zooms in" to the case of propagating from $t_{C_{1-1}}$ to $t_{C_{1-2}}$ in Figure 4-4, with E_2 interrupting the propagation. The gray box represents the update in the t_{next} property after triggering the event in the preliminary orbits model. This illustrates the fact that the rest of the models will propagate for less time, and thus collect and save less data.

An example of the subsystem sampling parameter and how it will be modified is by looking at the communications subsystem. For the CASTOR satellite, the communications subsystem is mostly inactive, unless the satellite is in view of the ground station. Therefore δt_{comm} could be set to a relatively high step size, such as 2 or 3 minutes, when not communicating. In this time, the subsystem would only look at the orbital position and find if the ground station is in view of the satellite. If it is in view, the exact time of initial communications would be found, and an event would be triggered for that time. When the

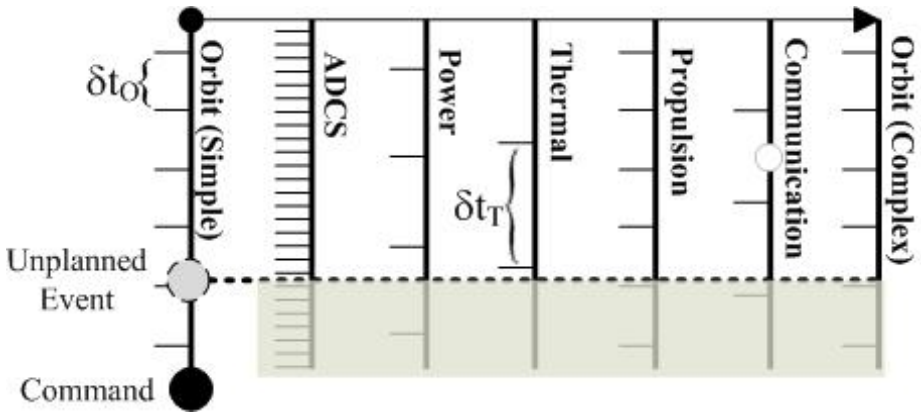


Figure 4-5: Satellite Propagation Subsystem Time Steps and Interrupt Diagram

scheduler sees that the satellite is in view of the ground, it will move back one time step and update or propagate at a rate much faster than δt_{comm}, on the order of 5 to 10 seconds in this case. A bisecting method can be used to minimize the convergence time. Once the viewing event is triggered and a command is executed signifying an actual communications period, δt_{comm} will be changed in the event callback to a higher rate to obtain proper coverage and signal strength details. Upon exiting a satellite contact, the time step will be changed back to a lower sampling rate in.

Figure 4-6 shows an even further zoomed-in instance of Figures 4-4 and 4-5 for the communications subsystem interrupt just described. This figure shows the actual interrupt time as $\odot$, and demonstrates how the time steps along δt_C until the event flag is triggered. Once the flag is tripped, a while loop is executed until the event resolution meets its requirement, τ_{crit}, which is generally 1 to 5 seconds in this simulation. The gray box on the right of the diagram represents the same excess area as Figure 4-5 that was truncated by a previous model updating for an event.

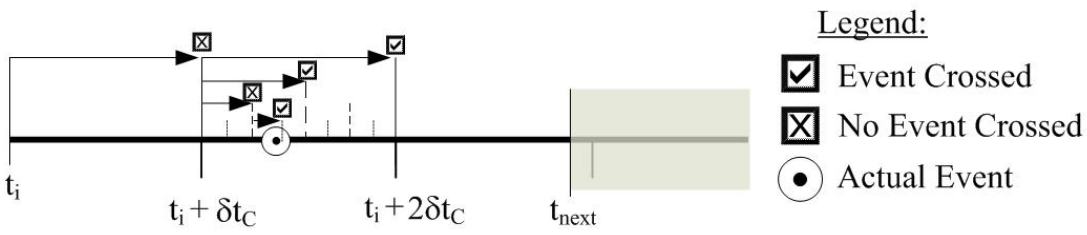


Figure 4-6: Example Subsystem Time Step and Interrupt Diagram

The bisecting algorithm used to find the precise time of the event is shown below. Note that for a fixed time step, the event timing is given simply by t_{next}.

$\tau = \delta t_{sub}$

while $\tau \geq \tau_{crit}$

    `check` $= f(t + \frac{\tau}{2})$

    if `check` $= false$

        $t = t + \frac{\tau}{2}$

    end

    $\tau = \frac{\tau}{2}$

end

The data distribution between subsystems and the simulation output is another significant topic that may not be trivial. This process can be simplified by fixing the ΔT and passing all of the state data between subsystems, as was done in the MOTV and initial CASTOR models. However, as Section 5.1.3 will show, this becomes costly as the detail of the model increases.

It is important to carefully track the data input and output into each subsystem model, and the overall data saved in the simulation. This is done with the assistance of the detailed N^2 diagram shown in Figure 4-3. It also requires an understanding of the telemetry format and individual channels created by the satellite. The design of the data distribution for an individual subsystem function is summarized by the following:

- Read time step from scheduler and generate expected time-tagged data matrix
- Check subsystem mode (pass through if in telemetry mode)
- Read relevant data from other systems necessary for subsystem of interest
- Perform any interpolation or manipulation of input data
- Run subsystem model (propagate to t_{next})
- Pass detailed, relevant data needed by other subsystems out

The relevant data used in the subsystem could be either a single value or a list of times and values passed from another subsystem. The source of this will always be the simulation

subsystem functions, although the flight telemetry could be included in this output, if desired. The data will be converted, if necessary, to a usable format for the subsystem after it is read in. An example of this would be converting an attitude format (such as quaternions or DCM) from the ADCS subsystem into a more usable angle-off-sun format for the power subsystem. Any interpolation of values is done at this point, as it is likely that the sampling will be different for each subsystem.

If a telemetry file exists with the specific subsystem data, it will have been read at the beginning and will already be stored in the subsystem object. The telemetry of interest for the given time will be read into the subsystem function and the data can either be passed straight through, or used to run through more detailed models. An example is the actual attitude position and rates from the satellite. If the only output required from the ADCS model is to provide the angular position and rates, the data can be simply passed through. Some changes in format or sampling may be desired, and additional calculations may be required to get local intermediate variables or global variables not included in the telemetry file.

The subsystem is then propagated with the data of interest at the defined step size, δt_{sub}. If an event or command occurs, the subsystem will interpolate as necessary to find the time of the change as detailed previously. If no event occurs, resulting data will be saved in two formats. The data of interest to the TDA or user (as defined previously) will be saved to the subsystem object as a property, and the data necessary for accurate simulation results will be passed locally to the other subsystems in the data structure. This data will only include the state data between t_i and t_{next}, which is important for long-duration simulations that have growing data matrices.

The default data saved into each subsystem object's telemetry property are the same measurements and calculations collected in the telemetry files on-board the satellite. This is done to make comparison easy between SIM and SAT data in the TDA. At the end of the simulation, the data matrices (structures or objects) will be saved in a usable format for the TDA.

In summary, it has been discussed how the simulation and all of its elements are initialized (`inits` structure), how the satellite state changes (command callbacks), how data is

passed during the propagation (N^2 diagram), and how telemetry is saved (subsystem object property). This is a complete set of information necessary to fully represent the satellite. Table 4.2 summarizes all the different information types. Section 5.1.4 describes the results of implementing this design on the CASTOR satellite.

Table 4.2: Simulation Information Classes and Types

Information	**Description**	**Nomenclature**
Initial Conditions	Create all simulation objects	`inits` structure
State	Current Satellite and Environment Conditions	`state` structure
Data	Temporary time tagged subsystem values	`data` structure
Telemetry	Subsystem specific data storage	`subs.tlm` property

4.2 FalconSAT-3 Telemetry and Data Analyzer

The discussion of developing a TDA is done with the FalconSAT-3 (FS-3) satellite. FS-3 was designed, built, and is still operated by USAFA's SSRC, and has provided a large quantity of both technical and programmatic data through its over three years of operations. This section will describe the development and progression of different on-orbit data analysis tools through many FS-3 unplanned operational events. This discussion will identify technical and programmatic lessons that can be applied to the IEM design, as well as general design and operations of small satellites. The FS-3 case study will show the evolution of the FS-3 telemetry and data analysis tool (FS-3 TDA). Section 5.2 will identify the importance of this tool in recent FS-3 events. The overall lessons from the FS-3 program that can be applied to small satellites will be discussed in Section 5.3.3.

4.2.1 FS-3 Data Overview

FalconSAT-3 (NORAD Catalog #30776) was launched on 9 March 2007 as a secondary payload aboard an Atlas-5 rocket [12]. To properly discuss its telemetry, the different sources of data for FS-3 must be described at the onset. The satellite was launched with a multiple data formats that could be collected and sent to the ground. These primary data formats used are:

- Real Time — All sensors, file system health, command and events
- Whole Orbit Data (WOD) Files — All sensor values
- PLANE Files — PLANE Payload files
- Attitude Files – Attitude Subsystem Files

The real time data is sent every time the satellite is in contact with the ground station. The sensor TLM is sent at a rate determined by operators, usually 5 to 30 seconds between samples. This data is saved by ground tools as it comes in. However, the data is sent only once by the satellite, so if the ground software is not running or the signal strength is not sufficient (e.g. antennas not pointed at satellite), this data can be incomplete. Two different TLM packets are sent individually, and it is possible for only one to be read and the other missed. Additionally, the satellite's avionics system sends health and status updates such as time, file system status, and acknowledgments of commands that have been processed by the satellite. These are all logged as ASCII data into a text file; however, these messages are also sent open loop as before, so they are only recorded if the signal strength is sufficient and the ground software is actively logging the data.

The other three types of data are started, stopped, and downloaded by operators upon request and they are saved into the satellite's memory during collection. WOD data is collected when telemetry data is requested between satellite contacts, PLANE files are created when running the PLANE payload, and attitude files are created when a more detailed attitude analysis is desired.

A little over a half year into the mission, it became clear that the data collection system in place was not sufficient. There was a significant trade-off in collecting the WOD files. If no WOD files were created, data is only collected during ground contacts, which is no more than 40 minutes a day for the FS-3 mission. If WOD files were written continuously, the data budget quickly goes into a negative margin, i.e. the data is being collected at a higher rate than can be transmitted. The original mode of operations is to only collect WOD files for a limited amount of time, such as one or two orbits, so the data budget stayed positive and an snapshot of the health and status was given. However, this process made any analysis challenging, as data still came at somewhat sporadic intervals. It was decided

to create a new data format, which only recorded 12 critical telemetry channels versus the 137 collected before. This format, called Limited WOD (LimWOD), allowed for continuous data collection and improved analysis and monitoring capabilities. The channels collected in LimWOD are battery voltage, battery temperature, solar panel currents, magnetometers, and time. Table 4.3 summarizes the different data types and their typical frequency of collection.

Table 4.3: FalconSAT-3 File Types

File Type	**Description**	**Collection Frequency**
WOD	All sensor readings	Only during significant tests
LimWOD	Critical sensor readings	Continuously
PLANE	PLANE payload data	Only during PLANE operations
Attitude	Attitude calculations	During control mode activation

The LimWOD file format was implemented in January 2008, and files have been nearly continuously collected since that time. Prior to this, WOD files were primarily analyzed using Microsoft Excel. This was done as the data could quickly be manipulated and graphs could be made and formatted as desired for the given test. As the LimWOD files came in at a much greater frequency and with only a limited set of data, a general analysis tool in Excel was created. This tool allowed the data to be analyzed via graphs by pasting in the raw LimWOD data. As too much data became hard to visualize, a new spreadsheet was created for each calendar day of data. However, this method also had limitations. The first limitation was the forced human intervention in adding the data. Operators or analysis were forced to open the LimWOD file (Comma Separated Value (CSV) format), and copy and past the data from the files into the excel template. Depending on the sampling rate, 12 to 60 files are created each day, so this became a tedious activity. The next major limitation was the ability to quickly and easily find or analyze desired data. The LimWOD data was sorted by day in individual files. Not only was this excessive from a data storage standpoint (CSV files much smaller than Excel files with graphs), it was also not very useful for data analysis. For example, when data for a specific set of time was desired to view, a common practice was to temporarily delete the data that outside the specific time range.

After a few months of utilizing the LimWOD Excel analysis tool, a MATLAB based tool was created to process and analyze the data. While taking a much greater level of programming competency, the MATLAB LimWOD reader tool allowed multiple days of files to be read and saved into MATLAB. This was primarily developed in support of gathering data for an altitude filter that required an easy way to gather data for long date ranges. While useful in this, the MATLAB LimWOD reader did not overcome the limitation of flexibility in data manipulation also present in the Excel analysis tool.

In the summer of 2009, MIT SSL partnered with USAFA SSRC to assist in the analysis of FS-3. At this time, the author quickly pursued the design of a new analysis tool. With over a full year of near continuous LimWOD file collection, this tool needed to quickly look at telemetry over many different time ranges. This led to the creation of the LimWOD Analysis GUI discussed in Section 4.2.3.

4.2.2 Telemetry, Data, and Operational Events

FalconSAT-3 has had many operational events in its three year history. An operational event is used in this context as any significant change in satellite state (hardware or software), as well as any off nominal behavior or anomaly. The purpose of this section is to guide through how the telemetry and data was used to analyze these operational events, and relate how these events can be discussed in the general IEM context.

LTAS Event

An operational event in FS-3's Low Thrust Attitude System (LTAS) occurred on 2 November 2007. Four 3-axis stacks of thrusters are located on FS-3, and one axis in one of the stacks was being tested to ensure that the it was still operational on the day of the event. The thruster was turned on for six minutes, which was at the time the longest duration of the mission. The initial analysis of the data using excel provided a graph shown in Figure 4-7.

It appeared from this graph that the thruster initially fired as expected, shown by the blue spikes in the main capacitor sensor. At t=2 minutes, the current from the 9 Volt regulated line suddenly jumped and remained continuously high until the command was

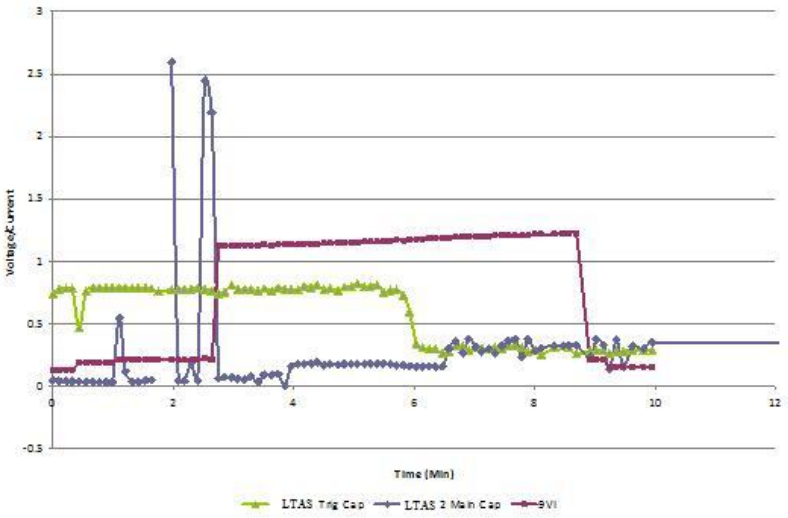


Figure 4-7: Initial Analysis of 2 Nov 2007 LTAS Data [31]

sent to turn off the LTAS thruster six minutes later. This constant current was not seen in any of the other thrusters, as they all show a somewhat random draw of current from LTAS as the sensor is sampled at different points along the pulse. For this reason, it was concluded that this thruster malfunctioned and likely shorted at the time of the event. A comparative graph for what nominal operations of a working LTAS thruster is shown in Figure 4-8.

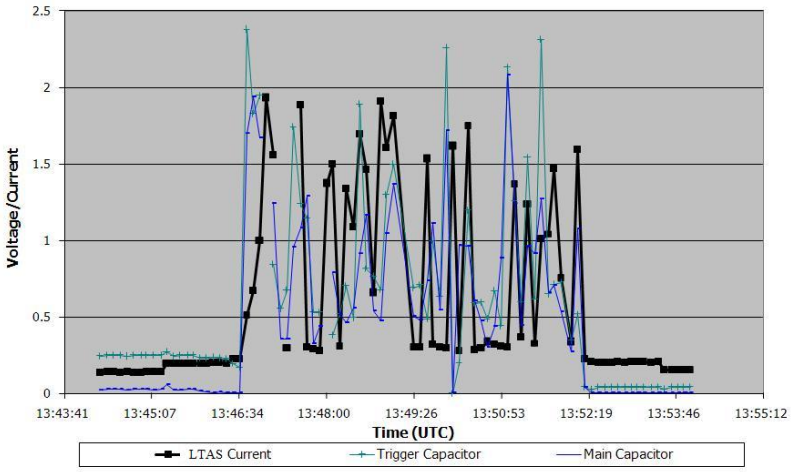


Figure 4-8: Nominal Telemetry of Operating LTAS Tube (8 Nov 2007 LTAS Data)

This conclusion was also based on the results of the first LTAS test that occurred on 9 May 2007. In this test, multiple LTAS clusters and thrusters were tested to demonstrate that

the payload survived launch and was operating as expected. The method for verification was showing the peaks in both the 9V regulator's current sensor and capacitor values. Originally it was determined that the thruster in question (+X, thruster 2) fired successfully due to the increase in the current drawn by LTAS. This can be seen in Figure 4-9. Therefore, after acceptance that the thruster fired successfully on orbit and looking at Figure 4-7, it was concluded that the LTAS +X cluster thruster 2 failed shortly into testing.

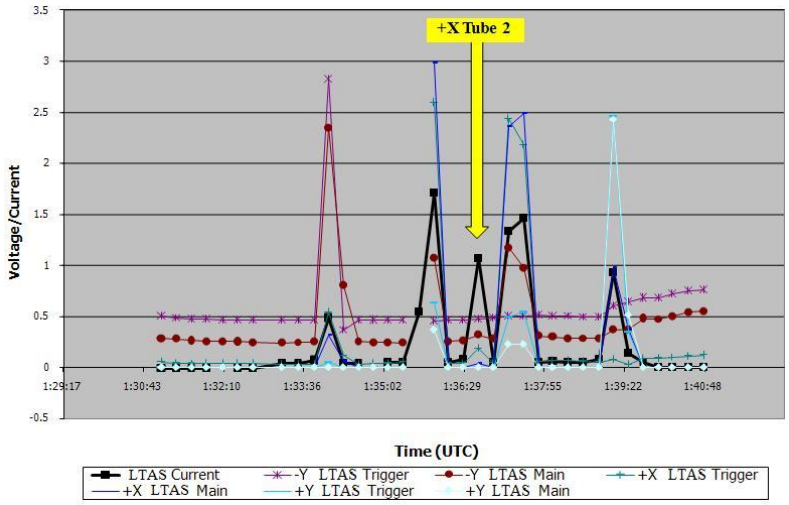


Figure 4-9: Initial LTAS Testing Results (9 May 2007 LTAS Data)

This event is relevant to this thesis as the data produced in Figure 4-7 was incorrectly created. Figure 4-10 shows the actual results from the event. This figure clearly shows that there was a jump in the current, and that the capacitors do not reach the voltage that would be expected (seen by Figure 4-8).

The extra data in the initial results is from a plotting issue with the Excel spreadsheet used for data analysis. The data was copied and pasted to create the graphs, and the graph plotted both the correct data and data from a different day. Once this was found, the initial conclusion of this thruster functioning after launch was questioned. When re-analyzing the data and reviewing Figure 4-9 again, it became clear that the one data point showing a jump did not meet the criteria for a working LTAS. Additionally, the capacitance values recorded are significantly lower than those recorded during the firing of the other two thrusters on that cluster, and match the order of magnitude shown in Figure 4-10. With a better understanding of the data and reanalyzing the original verification tests from May

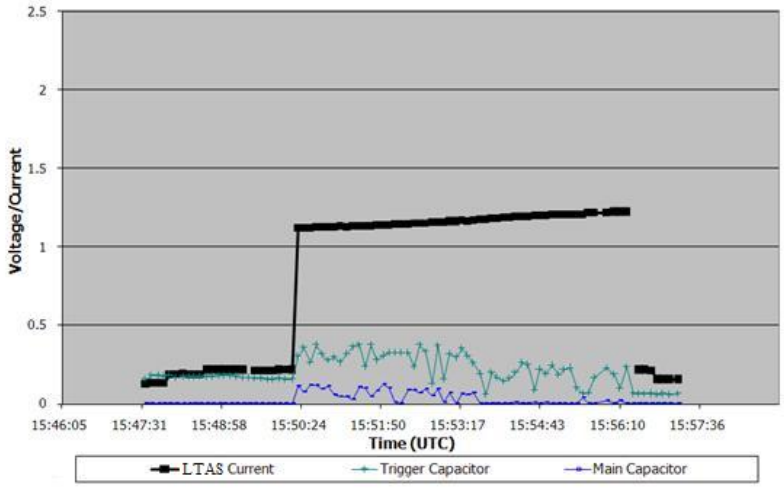


Figure 4-10: Corrected LTAS 2 Nov 2007 Data Plots)

2007, the FS-3 team concluded that the LTAS +X thruster 2 did not malfunction during the November test, but rather it never worked on-orbit.

The lessons from this example are the following:

- It is possible to misdiagnose a system with limited data
- Repeatable analysis processes are desired to compare data in the same way each time
- Spreadsheet tools such as Excel are susceptible to human error
- Re-analyzing data can be productive, even if the results seemed clear at the time, as the previous analysis may have not had the amount of data or understanding of the system

WOD Timing Event

In November 2007, the gravity gradient boom on FS-3 was deployed. In this time, many WOD data collections were performed to look at the rotation rate of the satellite between contacts. The WOD data seemed to have strange gaps between the stop time of one file and the start time of another and was attributed initially to processor and file system issues. However, when one analysis of the spin rates was done by looking at the magnetometer data in both the real-time sensor TLM and the WOD downloaded TLM, it became clear that the WOD files had a timing problem. This data comparing both sensor readings for the same timescale are shown in Figure 4-11.

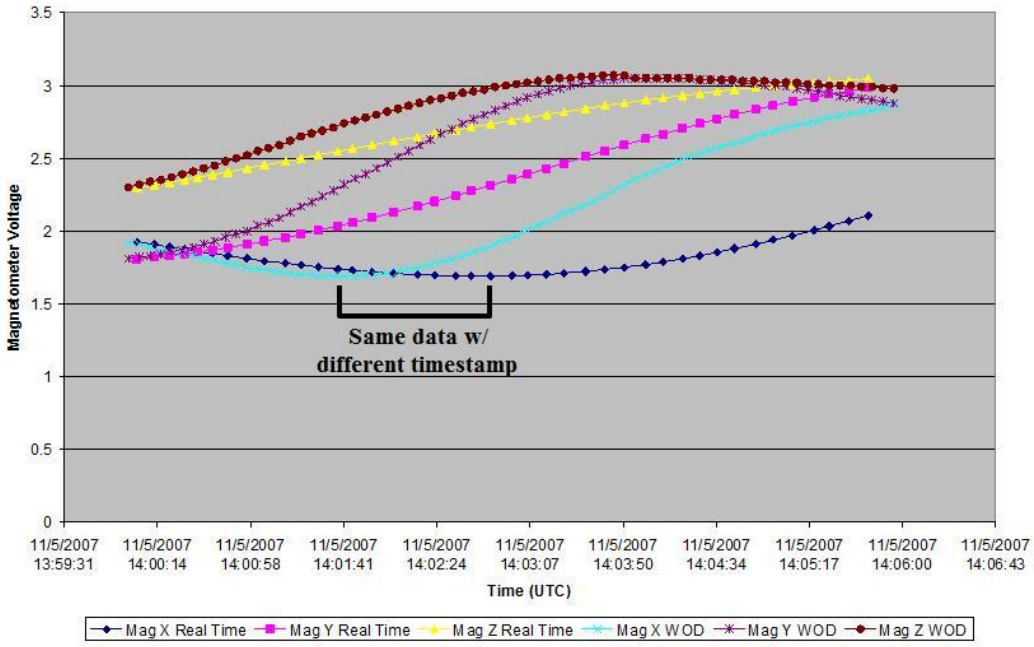


Figure 4-11: Comparison Between WOD and Real-Time Magnetometer Values

Figure 4-11 shows how each WOD magnetometer value begins at the same point as the real-time values, which is due to the WOD file being initialized at that time. As time moves forward, the WOD data lags the real-time data. The data for the same value, such as the minimum value in the X magnetometer, happens earlier in the real-time data than in the WOD data (as shown by the black line). Upon investigation of this mismatch, it was determined that the problem was not with the WOD collection on-board the satellite, but primarily with the ground processing of the data. The header in the WOD file contains the start time for data collection and intended sample rate of the data at this time. The ground processing tool assumes that the data is always collected at this requested interval and that it never changes. This assumption is not valid, as the data rate is often modified by operators. Even if the desired sample rate is fixed, the actual sampling of the data is not constant. The issue was resolved by adding the sample time to the WOD data by treating it like a sensor. The ground processing program has not been updated, leading the CSV file generated by the decoder to still have incorrect time steps. Therefore it is necessary to have a processing tool take the CSV files and manipulate the time to produce correct time tags.

This event and the issue of timing in general invoke several themes that relate to the IEM design:

- Timing sources between all satellite telemetry formats should be verified on the ground and after launch
- The satellite clock should match all ground clocks (correct for any processor drift or ground timing issues)
- A central database that stores all the TLM and data formats with correct time tags is preferred

Torque Rod Strength Event

FalconSAT-3 has three magnetic torque rods fixed to the orthogonal body axes. These torque rods are the primary source of attitude actuation in FS-3. They were individually tested prior to integration to determine their effective dipole, which ranged between 0 and 5 Am^2. Upon integration with the system, acceptance testing was done by testing the highest strength setting and showing a change in the magnetic field with a compass, as well as looking at the power draw and current used by the individual torque rods.

In Fall 2007, a B-dot controller was run to slow the rotation rate of the satellite. The B-dot algorithm implemented on FS-3 simply outputs a command to a torque rod that is opposite of the change in direction of the sensed magnetic field from the Earth. That is, if the x-axis magnetometer shows an increasing magnetic field, the command would be given to the x-torque rod to create a dipole in the -X direction. While there are additional proportionality and multi-axis options in the general B-dot logic, this B-dot controller was utilized on FS-3 for simplicity in implementation. When implemented on orbit, the B-dot algorithm had little effect on the satellite. Commands were generated by the satellite showing the switching polarity of the torque rods as the satellite rotated through the Earth's magnetic field; however, no significant change in rotation was observed.

The cause for this issue was found in January 2008 when a contractor who supports FS-3 was performing tests on the FS-3 Qualifications Model (QM). The individual found that the different strength settings on the torque rods produced nearly random current values in both the sensed current to the torque rod and the power distribution unit. These results

are shown in Figure 4-12. The test was done on both torque rods available and in both directions.

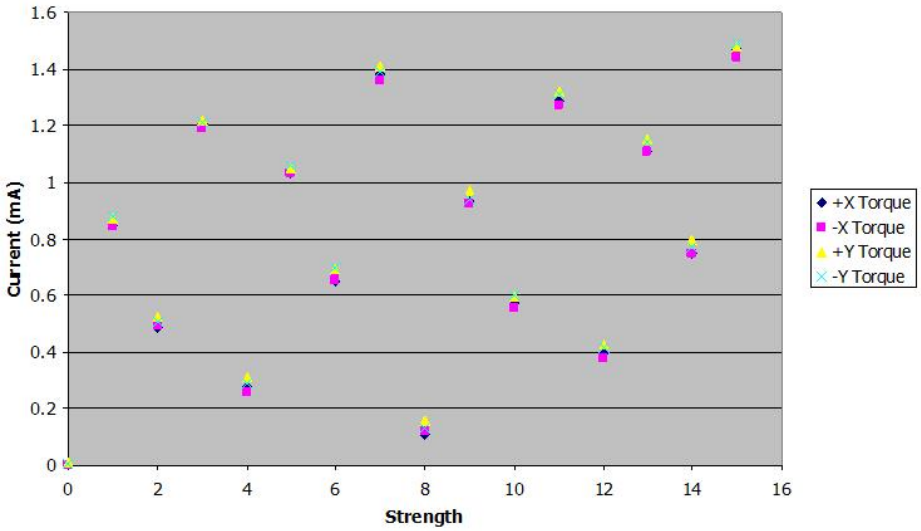


Figure 4-12: Torque Rod Current vs. Initial Strength Setting [38]

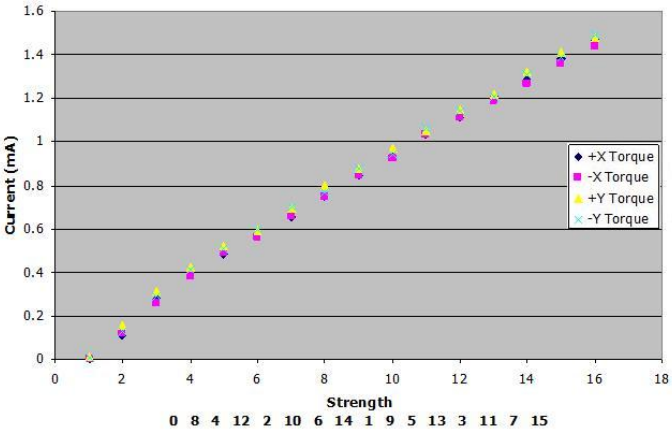
The cause for the random appearance in Figure 4-12 was an inversion of the strength setting, represented by a 4-bit nibble. The strength setting that was used in the B-dot controller was the setting of 8, or 1000 in binary. The inversion caused this to be represented as 0001, or the second lowest strength setting. At this strength, negligible torque were created, and thus no change in dynamics was observed. Figure 4-13 shows that when accounting for the nibble inversion, the current from the torque rods increases monotonically as expected. Testing on the FM verified that this inversion existed and behaved on the ground.

This event has a few lessons that can be applied to the IEM design:

- Verification steps should test through the range of possible commands and not just extremes

- Verification done on the ground that can also be performed on orbit should also be done on orbit, and with the same tools

- Utilizing telemetry and analysis tools for verification both in testing and on-orbit checkout can assist in complete analysis



Expected Strength	Expected Word	Actual Word	Actual Strength
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
8	**1000**	**0001**	**1**
9	1001	1001	9
10	1010	0101	5
11	1011	1101	13
12	1100	0011	3
13	1101	1011	11
14	1110	0111	7
15	1111	1111	15

Figure 4-13: Torque Rod Current vs. Revised Strength Setting [38]

Torque Rod Polarity Tests

Although the strength issues were fixed, uncertainty existed in the polarities of the different torque rods. As the polarity is critical to lower the rotation rate using the B-dot controller, this topic was one of the first examined by the SSL team. Figure 4-14 shows the results from this test. This data was created using a MATLAB script and previously saved processed data sets. The test set-up consisted of commanding torque rods on and off at different strength and polarity settings to see the effective change in the magnetometers.

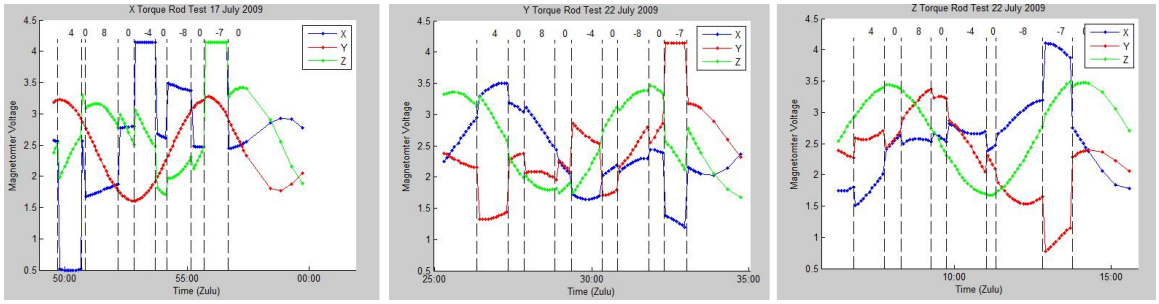


Figure 4-14: Torque Rod Polarity Test Results

The data shown in Figure 4-14 showed that each torque rod displayed different relative magnitudes for a given strength. A magnetic model was created by SSL PhD candidate Jaime Ramirez utilizing the geometry and placement of the torque rods with respect to the magnetometer to show the expected results for a given strength. The output of this model is shown in Figure 4-15. This figure shows that the general response seen on the

magnetometers from the open loop torque rod commands matches the predicted model in all cases.

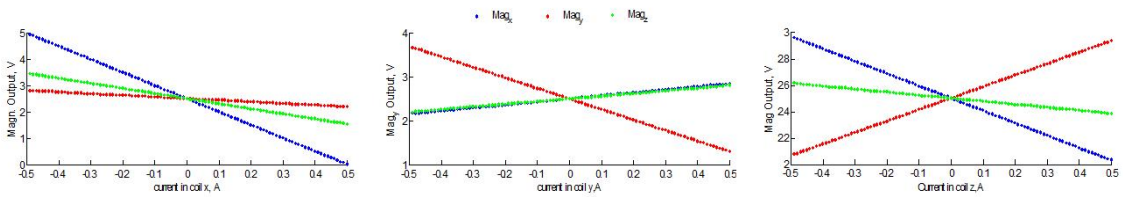


Figure 4-15: Torque Rod and Magnetometer Model

This testing sequence validated that the torque rod polarity was correct in all cases. Using a MATLAB script to perform the analysis allows the results to be re-created or modified as necessary. Since this was a one-time test, it was not necessary to implement this feature in a repeatable medium.

Magnetometer Calibration Event

FalconSAT-3 uses a 3-axis magnetometer as its primary source of attitude knowledge. This magnetometer outputs a voltage that corresponds to an observed magnetic field (B-field) in the three body axes. A team of SSL students and faculty investigating the FS-3 attitude system in June 2009 found that the coefficients for converting voltage to measured B-field (in Tesla) were not accurate. This was found by finding the magnitude of $\boldsymbol{B}$ in both the body and Earth Centered Inertial (ECI) coordinate frames. While direction of $\boldsymbol{B}$ will be different between the two frames, the magnitude should be equal. The expected magnitude of the $\boldsymbol{B}_{ECI}$ comes from the International Geomagnetic Reference Field (IGRF) model of the Earth's magnetic field, and the magnitude of $\boldsymbol{B}_{body}$ comes from the magnitude of the converted magnetometer strength readings.

An overlay of the plots of the two magnitudes with time is shown in Figure 4-16. This shows that $\boldsymbol{B}_{body}$ deviates significantly and periodically from the expected IGRF-based magnitude. While the IGRF is only a model of the Earth's magnetic field and has a error associated with it, the deviation from the magnetometer outputs is orders of magnitude greater than this error. The MIT team hypothesized that either the magnetometer was improperly calibrated or an unexpected magnetic source exists in FS-3. In the case of

improper calibration coefficients, new coefficients could be tested to see if the data fits better for all ranges of time. This method will also work assuming any magnetic source was body fixed and constant. While these assumptions may not be valid, they were made initially as the problem becomes much more complex with dynamic magnetic sources.

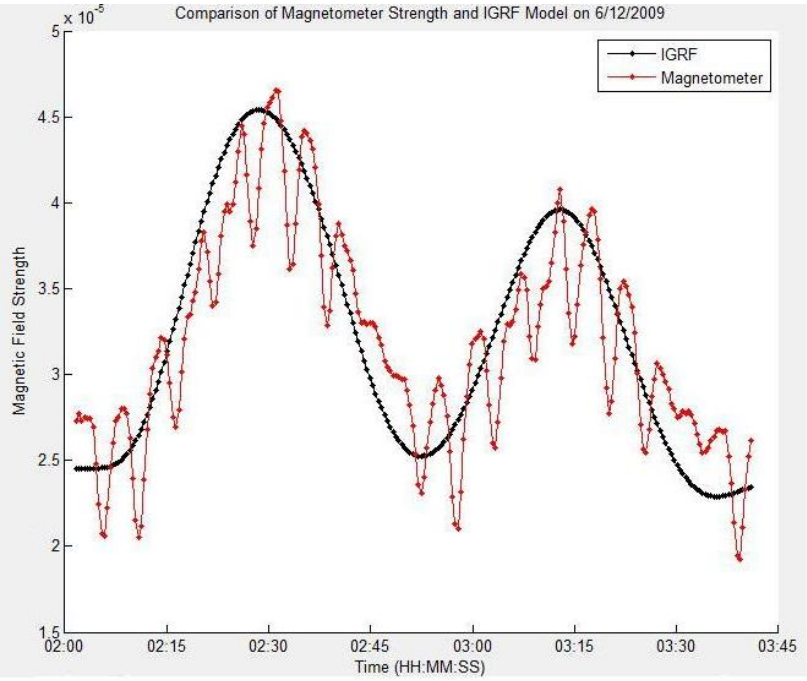


Figure 4-16: Initial Comparison of Magnetometer and IGRF $\|\boldsymbol{B}\|$

A non-linear least squares method was developed by SSL PhD candidate Jacob Katz to determine the magnetometer conversion coefficients that fit the IGRF best. Each magnetometer axis ($\hat{i}$) outputs a voltage ($v_{\hat{i}}$) which is multiplied by a constant ($a_{\hat{i}}$) and added to a bias term ($b_{\hat{i}}$) to obtain the magnetic strength reading in Tesla (T) or μT. An error term e_k is found by taking the difference between the magnitude in the body-axis B-field and the inertial B-field. The least squares method minimizes the error for each term to output the three components of $\boldsymbol{a}$ and $\boldsymbol{b}$. These steps are represented mathematically in equations 4.1 to 4.3.

$$\boldsymbol{B}_{body} = \boldsymbol{a}^T \boldsymbol{v} + \boldsymbol{b} \tag{4.1}$$

$$e_k = \|\boldsymbol{B}_{body}\|_k - \|\boldsymbol{B}_{IGRF}\|_k \tag{4.2}$$

$$(\boldsymbol{a}, \boldsymbol{b})_{opt} = argmin_{\boldsymbol{a},\boldsymbol{b}} \sum_k e_k^2 \tag{4.3}$$

The least squares method was implemented for different date ranges, outputting several values for $\boldsymbol{a}$ and $\boldsymbol{b}$. While the method did not always output the same values, a general set was found that matched all data sets well. The results with the new coefficients are seen in Figure 4-17. These results are not expected to match completely due to uncertainty in satellite position based on TLE propagation, uncertainty in the IGRF model, and time varying properties of the magnetometer (such as changes in reading with temperature).

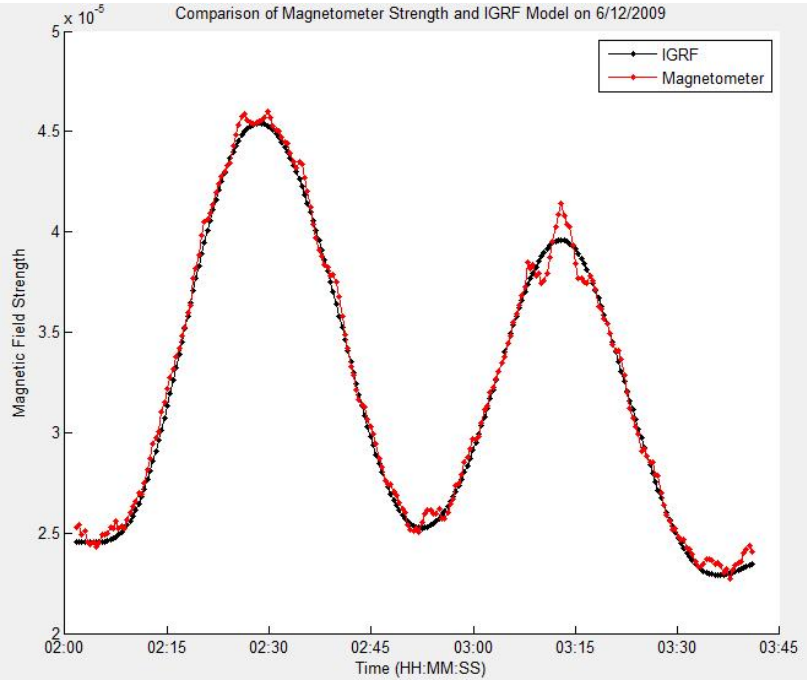


Figure 4-17: Comparison of Magnetometer and IGRF $\|\boldsymbol{B}\|$ (New Coefficients)

The magnetometer coefficients are significant for any filter to determine the attitude of FS-3. If the coefficients are wrong, the accuracy of any knowledge estimate will be affected, and it is possible that the filter could diverge, even if the satellite was stable. For this reason, the discovery that the magnitude created with the initial coefficient values did not sufficiently match the expected IGRF magnitude is significant.

The major lessons from this event that relate to the topic of IEM are the following:

- Tuning parameters can be tested before flight, but should be verified on-orbit as well
- Like torque rod testing, the same logic or algorithms used for verification on the ground can be used on orbit
- Certain tests need to be repeatable on sets of data from different dates.

Additionally, this event was described as it led to the development and use of the LimWOD Analysis GUI. The design and implementation of this GUI is described in Section 4.2.3.

4.2.3 FS-3 LimWOD GUI Design

LimWOD GUI Motivation and Development History

The previous section showed several examples of graphical analysis tools used in FS-3 operations. The addition of continuous LimWOD collections led to the development of a repeatable process for analyzing the large amounts of data. While an Excel template provided a short-term solution, it was easy to unintentionally manipulate, and did not effectively display long term trends or short term dynamics of the system. The MATLAB-based LimWOD GUI was developed to leverage the previous experience with the Excel-based template and several MATLAB-based data processing tools to allow for greater availability to visualize and process data. This section describes the initial design and further add-ons made since July 2009.

The FS-3 LimWOD GUI analysis tool was created initially to support the long-term analysis of magnetometer data by the MIT SSL team. As 18 months of near-continuous LimWOD data existed (February 2008 to July 2009), the SSL team needed a tool to quickly parse through this data. As mentioned previously, an assortment of tools were available that processed the LimWOD files for a specific range of dates and returned data matrices and graphs of the data through the range of dates. These tools were utilized as the input source to the GUI, and then the rest of the GUI was used for data processing and flexible graphical analysis.

The initial data processing tools built into the GUI were a SGP4 TLE propagator and IGRF magnetic model. These were implemented to create the graphs shown in Figures 4-16

and 4-17. These tools existed and were used previously to support attitude analysis, but the magnitude of the vectors was not looked at until July 2009.

Within two weeks, a data processing tool was created that performed the following functions. These functions are described in further detail on page 108.

- Collect data for a range of dates and store it with correct time steps in large data array
- Allow for graphical depiction of each channel of data with start and stop time limits down to the second
- Propagate TLE through time to generate the IGRF magnetic model of the satellite through its orbit
- Compare converted magnetometer strength with IGRF magnetic field strength
- Perform best-fit of IGRF strength to find acceptable $\boldsymbol{a}$ and $\boldsymbol{b}$ coefficients
- Modify coefficients manually to compare strengths

Several upgrades have occurred since the initial development of the GUI. These updates can be classified into the following different categories: maintenance, additional analysis features, and enhanced user interaction.

The maintenance category represents any modifications needed to be able to successfully run the analysis tool. While this can couple with unintentional changes from additional analysis or user interaction features, it is mostly a function of improper assumptions on the long-term input set. Examples of such assumptions can be the date format or data file structure. When analyzing data that passes between months or years, certain tools assumed either a location or an improper structure of the date, leading to errors or unintended data being created. These improper assumptions are addressed at the lowest level, if possible, to lead to more robust and portable analysis tools. Additional maintenance functions include a changing environment in either the satellite or ground segment, such as changing the data structure collected by the satellite or the ground handling of the data.

Maintenance changes and actions should fix improper assumptions necessary to run analysis, or identify any assumptions that limit the analysis capabilities. One such example of the latter is that the LimWOD GUI is only meant to analyze the channels of telemetry collected in the LimWOD format. Originally, the full WOD files were not even read by the

analysis tool; however, upon testing campaigns that led to collection of many full WOD files, the supporting tools to read the data were modified so that full WOD files were read and able to be analyzed in the LimWOD GUI. Only those channels normally collected in the LimWOD format are available for analysis in the LimWOD GUI, even though a much larger set of data exists. Therefore an important restriction made is to have the LimWOD GUI tool only analyze the select channels collected in LimWOD format, even if more data is available.

The next type of update to the GUI is adding additional analysis features. This is done to support different operational tests or further studies of existing FS-3 data. Examples of additional features added are the expected B-dot logic commands, viewing the length of the time step between data samples, running attitude filters, and calculating the expected solar panel angles off the filtered attitude results. This class of updates utilizes the investment of data processing and aggregation of analysis tools to build more and more defined analysis capabilities. These analysis features will be described in further detail in page 108.

The final class of updates to the GUI are used to enhance the utility and user interaction with the tool. A significant amount of analysis comes from visual identification of graphical data, leading to the necessity to clearly depict the data in graphs. This includes proper time labels, axis labels, legends, and titles for each graph. While it is possible to perform analysis without these cosmetic properties, there are many short-term and long term benefits to having readable and presentable graphs output such as clear communication of data to stakeholders and additional analysts. Another form of user interaction updates comes from the ability to sort through the data quickly. This has been accomplished in many ways through the GUI development to enhance the user's interaction and ease to access data, as described on page 119. Another example of user interaction is the selection of input conditions such as TLE files to use, selection of dates to analyze, computer path of the data, and satellite or analysis files to analyze. This has been done in different ways such as text-based input dates, a string of the path or files, and interactive selection tools. While these updates do not affect the results in any way, enhanced user interaction and a more robust source of inputs allows a greater amount of personnel to utilize the GUI's features.

LimWOD GUI Features

With an overview of the GUI and its modifications, this section describes the actual implementation of the different analysis and interactive techniques. Many of these are specific to the MATLAB GUI framework and FS-3 mission; however, many of the techniques can be approached in a broader viewpoint for general TDA design practices. The different topics described in this section is summarized in Table 4.4

Table 4.4: LimWOD GUI Design Outline

Importing and Loading Data and Files in LimWOD GUI Design

The LimWOD GUI requires several different input files to perform all of its functions. At a minimum, the GUI needs to pass the location of the data files, the start date, and the stop date to the function `wod_search`. This function searches for the files that are in the directory between the dates. The files are saved into a character array and a "for" loop calls `wod_lim_process` for each file to save the data into a numeric data matrix. Maintenance modifications made in this process include calling `wod_full_process` inside of the pre-written `wod_lim_process` function in the case of full WOD collections, along with ensuring only the data between the dates is saved by calling the previous day's last data file and saving data that was written into the start date of interest, and truncating after midnight on the stop date of interest. The data is found after the button "Generate Data" is pressed, as shown in Figure 4-18. Additionally, this figure shows the data location, shown to the right of the "CSV Path" text. This path is defined due to the data structure being undefined during the initial development. A later modification (user enhancement) ensured this path is defined correctly upon launching the GUI, as all analysis software and satellite data is now linked in a subversion repository accessed by both MIT and USAFA.

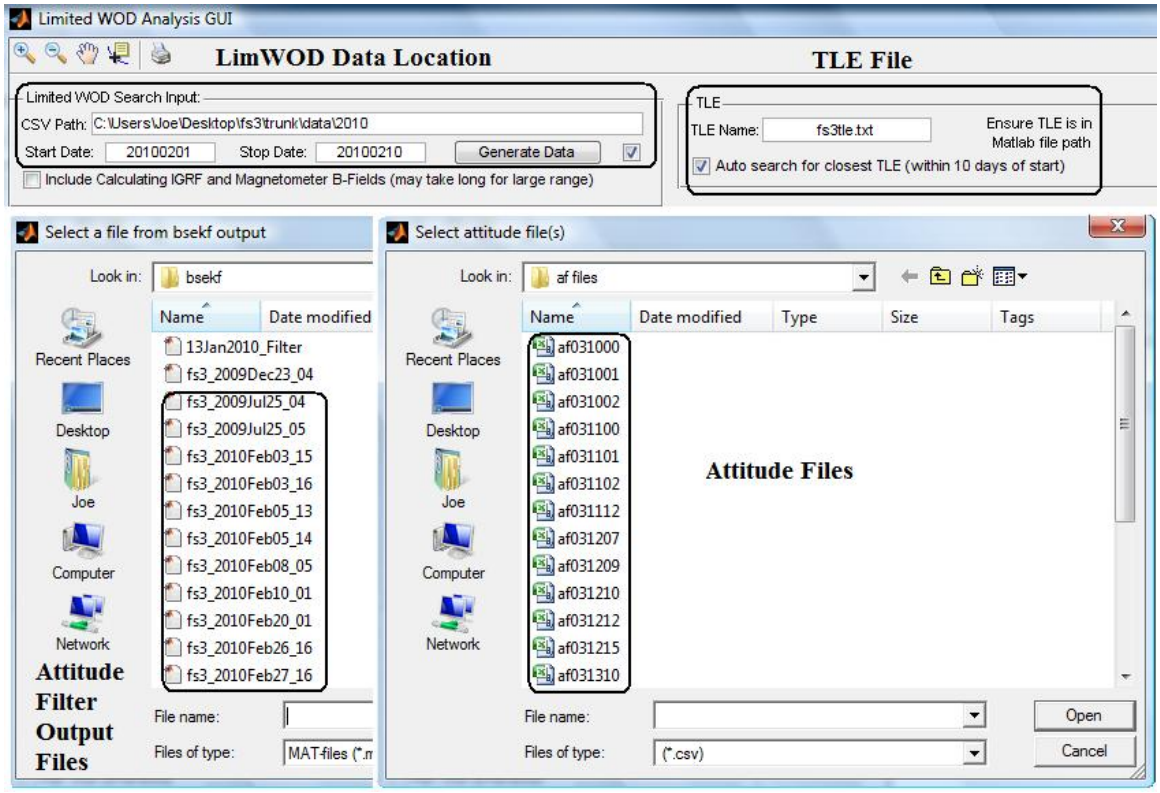


Figure 4-18: LimWOD GUI Data Import Formats

The Two Line Element (TLE) file is selected in a similar way as the data folder, with the exception that it assumes the TLE is in the correct location already. Using the correct TLE file is important as the SGP4 propagation method used is only accurate for a limited amount of time. The TLE is used to propagate the satellite's position forward in time, and is primarily used in the FS-3 satellite for comparison with the IGRF model. As analysis can take place for a wide variety of times, it is important to have a flexible input file. The GUI has a search function that assumes that the files are named "fs3tleYYDDD.txt", where the day (DDD) is in increments of 20 days, and it will automatically display the name of the file if the "Auto search" button is selected. This is not ideal as it forces a user to create and name the different TLE files, and a user enhancement modification could be made to automate this process; however, the accuracy is sufficient with a 10 day maximum lag and TLE files are maintained by those frequently doing analysis.

The two pop-up windows shown in the bottom half of Figure 4-18 represent modifications in both analysis features and user enhancement that occurred later in the LimWOD GUI

development. The left graphic shows a collection of data files created by the BSEKF Attitude Filter that will be described later (page 116). The user has the option of analyzing these files to compare the solar panel currents to the estimated solar angle for each panel given the attitude estimate from the filter. This was done in the verification process of the BSEKF filter, and since many data sets existed, it was added to the LimWOD GUI capabilities. The user enhancement was made in the discovery of the MATLAB `uigetfile` function, which allows the user to select file(s) from a standard pop-up window. This is advantageous as it is much more intuitive than typing the file name and path into a box, along with creating a simpler graphical interface with the user. The same import option is done with the addition of reading satellite attitude files for controller verification, which will be described further (page 116). MATLAB code in Appendix A, page 181 shows how the function `uigetfile` is used to produce a cleaner, user interactive import option for FS-3 attitude (CSV format) files.

The current procedures to analyze the data begin with loading in the data for the day of interest. This was the initial purpose of the LimWOD GUI, although with the addition of different data sets, it is not always necessary to load an entire day of data. Future revisions could have the analysis files (such as satellite-created attitude files or BSEKF output files) initiate the satellite data search so only the data from that time is compared. Additionally, if all LimWOD and WOD data was placed into a database with correct timestamps, the data searching function would be much more intuitive and quicker. This was initially performed prior to launch with WOD files and real-time TLM, but the WOD timing event caused this feature to become obsolete.

To summarize, multiple sources of data are used in the LimWOD GUI. The ability to import this data has improved throughout the use of the LimWOD analysis GUI. The TLE file importing process is semi-automated, although it still uses assumptions that users create proper filenames in the correct folder. Data searching could be improved in future revisions to allow for analysis files to call the data required, or to have a central database for all WOD data.

SGP4 Propagator and IGRF Model

The IGRF magnetic model of the Earth is necessary in order to properly analyze the FS-3 magnetometer data. As the magnetic field of the Earth is a function of longitude, latitude, and altitude, the satellite's position vector must be known for proper analysis. The requirement to know the satellite's position leads to utilization of the Simplified General Perturbations 4 (SGP4) propagation technique. This propagation method utilizes mean elements and a general perturbation technique to find the satellite's position and velocity at a future time [41]. TLE provided by NORAD/DoD are used as the orbit epoch, and are loaded as text files as explained earlier. Multiple files from the Aerospace Mission Analysis MATLAB (amam) set are used to load the TLEs and run the SGP4 technique.

The SGP4 propagation and IGRF model require a significant amount of computation when compared to other elements of the LimWOD GUI. For this reason, the GUI is designed so this calculation only occurs once for each given set of input dates and data. Neither the satellite position vector or IGRF vector in the inertial reference frame will change for a given set of dates and TLE, so this assumption is valid. The magnetic readings derived from the FS-3 magnetometers are possible to change with modifications to the coefficients ($\boldsymbol{a}$ and $\boldsymbol{b}$), but this will only save over the previous data, and not re-run the IGRF model.

One element of the IGRF that has not been applied to the LimWOD GUI is the transition from the IGRF, a predictive model, versus the DGRF (Definitive Geomagnetic Reference Field), a retrospective collection of data based on measurements [39]. This is a research area available for further work, especially in the area of magnetometer calibration, as the figures shown (Figures 4-16 and 4-17) do not give the indication that the new coefficients represent the true magnetic field.

Magnetometer Analysis Tools

The FS-3 magnetometers provide the best measurement of the FS-3 attitude by sensing the strength of the magnetic field in each of the three orthogonal body axes. Preliminary analysis of one orbit of data from June 2009 showed that the magnitude of the magnetic field output by the magnetometers oscillated at a much higher frequency then expected. The development of the LimWOD GUI analysis tool allowed this magnitude to be compared

to the expected magnitude output from the IGRF model. Figures 4-16 and 4-17 showed two such outputs of these graphs that were taken from screenshots of the GUI. This graph was produced by plotting two columns of the matrix `Bdata` against each other, given the same time. `Bdata` consists of eight columns: three IGRF components in the orbital reference frame (LVLH frame), the IGRF magnitude, three body-axes components from the magnetometers, and the magnetometer magnitude. All columns are in Tesla units.

Whenever the IGRF data is needed, all columns of `Bdata` are calculated. When calculating the magnetic field relative to the body axes as sensed by the magnetometer, the GUI uses values of $\boldsymbol{a}$ and $\boldsymbol{b}$ that are shown in Figure 4-19. The user can manually adjust these parameters or default to the original or revised set of coefficients used. The least squares method described in equations 4.1 to 4.3 is performed when the "Calculate New Values" button is pressed. Additionally, a flag is tripped if the magnetometer coefficients are ever changed, either manually or by pressing one of the buttons. Whenever the data is to be plotted, the last four columns of the `Bdata` matrix are recalculated using the new coefficients. As stated before, the IGRF data is not recalculated as it will not change, and due to its large computational nature.

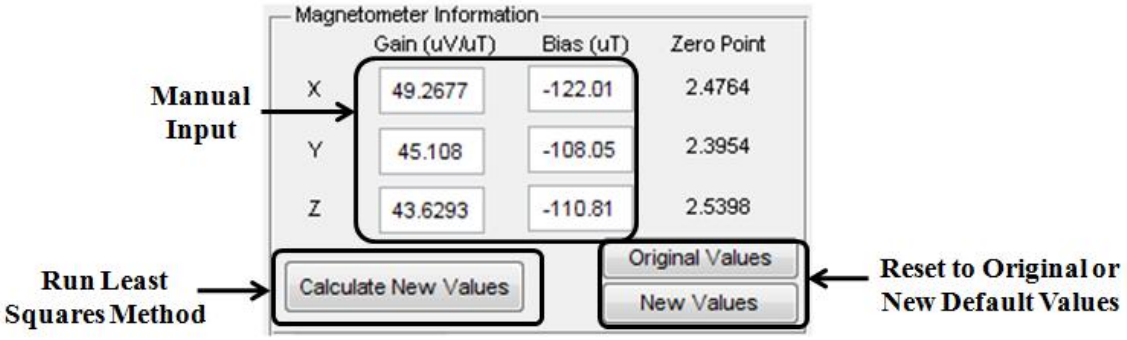


Figure 4-19: LimWOD GUI Magnetometer Coefficients Selection

Another feature of the LimWOD GUI is to fit the IGRF data for a limited time span. If periods of time are missing, either from not being collected by the satellite, or not being downloaded from the satellite, the least squares algorithm will not converge correctly. Therefore whenever the "Calculated New Values" button is pressed, it will utilize the date and time range used by the plotting function to determine the best coefficients. This creates a problem if the time span is too short, as the best fit of the data may only be a

local optimum, and not a global fit that would work for all values of time. This feature is also useful if magnetic disturbances are present, such as during open loop firing of torque rods.

Time and Sampling Techniques

The issue of timing is critical in analyzing data from several sources. The WOD timing event described on page 97 was one instance of improper time tags. Additional care needs to be placed between any comparison of ground time and satellite time. The FS-3 processor clock has a drift of approximately 5 seconds a day, and during the first contact of the day, the satellite clock is synchronized with the ground station time. This procedure assumes the ground station time being sent is the correct GMT time, which is not always the case. It is possible for the computer sending the time to have a different value of time than the computer logging the real-time data. The function `craread` was written to analyze the real-time text log files (CRA files) in MATLAB. This function outputs the time difference between the ground clock and the computer clock, both graphically and in a data array. An example of a standard analysis showing four contacts within a 24 hour period is shown in Figure 4-20. This data is useful as significant offsets in the time can disrupt analysis, especially for looking at the timing of commands generated based on dynamic readings such as torque rod commands generated during the B-dot logic algorithm. It should be noted that the `craread` function is independent of the LimWOD GUI, as it deals with another data type. However, this analysis tool could be transitioned into the GUI if desired.

Another issue of timing that was unexpected at launch was the dynamic nature of the sample size. This was the critical assumption that led to the invalidation of the original ground-based sample size. The sample size can change significantly with a ground based command. The sample rate can be modified in five second increments between five seconds and one minute in duration. This change in timing is very predictable, as it is only triggered by a ground-based command. However, The sample time can also change on its own, due to higher priority taskings with the flight processor or due to processor clock drift. These changes are shown with 6 April 2010 data in Figure 4-21. This graph is generated by the GUI and was plotted externally to allow for additional labeling on the Y-axis.

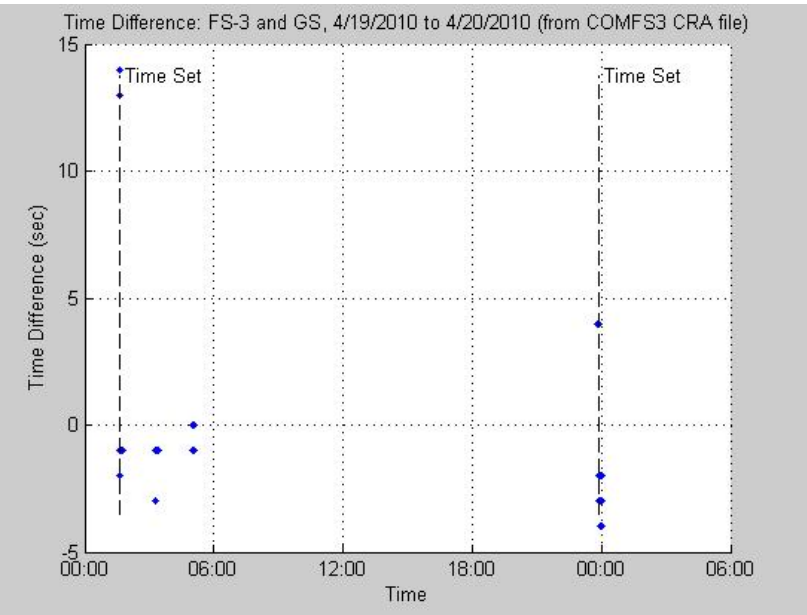


Figure 4-20: Comparison Between Ground and Satellite Times

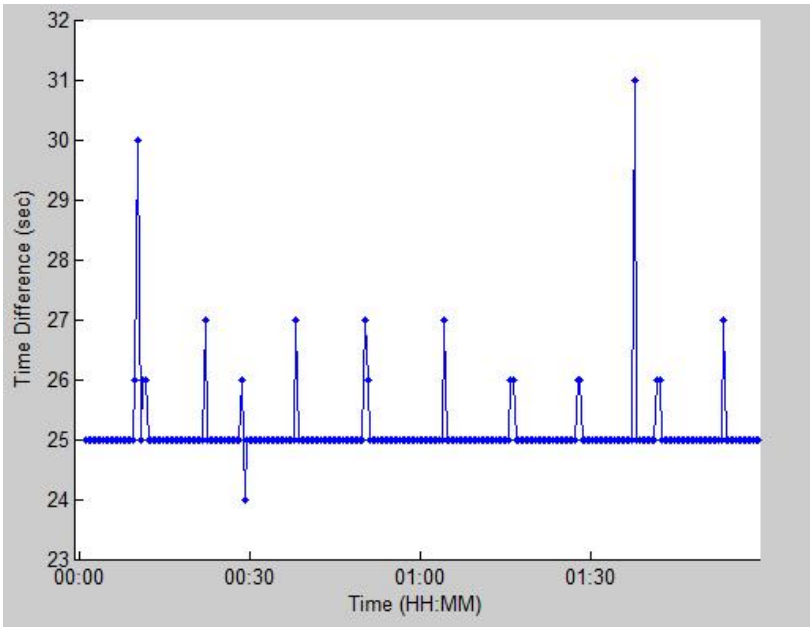


Figure 4-21: Satellite Time Between Samples

The ability to plot the time between samples was a late add-on to the GUI that will become important in analyzing aliased data on FS-3. Aliasing challenged analysts several time during FS-3's lifetime, and was identified as the cause for several unexplained behaviors in April 2010. In general, dynamics in the space environment are very slow, especially for relatively simple satellites like FS-3. For this reason, aliasing of data and sensor readings

is generally not a problem. In the case of FS-3, a number of spin-up events occurred that led to time segments with aliasing of attitude sensor data. The timing jumps can assist in identifying aliasing and determining how badly aliased a signal is. This will be discussed further in Section 5.2.2.

The satellite time written to LimWOD and WOD files is in Unix time. This time, which is in the format of seconds since January 1 1970, is not understandable and must be converted. MATLAB has several built-in functions that allow for representation of dates. Additionally, the date format of Julian Date is also used when propagating the satellite or other astronomical bodies. Therefore it became necessary to carefully ensure that the time being used is in the proper format. Table 4.5 following functions are used to deal with time in MATLAB, their use, and their developer.

Table 4.5: MATLAB Timing Functions

Function	**Use**	**Developer**
`time`	Conversion of different time formats	Author
`datenum`	Converts date string to MATLAB number	MATLAB
`datevec`	Converts MATLAB date number to date vector	MATLAB
`datestr`	Converts MATLAB date number/vector to string	MATLAB
`datetick`	Auto-formats x-axis labels to string	MATLAB

The function `time` was developed to convert between many different formats used. These formats include a date string (e.g. 5/14/2010 05:30:00), Julian Date (days since 4713 BCE), GFSC Modified Julian Date (days since Jan 5 1941), USNO Modified Julian Date (days since Nov 17 1858), Unix Time (seconds since Jan 1 1970), Day of Year (doy.xxx), Year & DOY (YYdoy.xxx), and MATLAB time (days since 1 Jan 0000). This function was written to allow for a general conversion between all formats, and is not optimized for run time. Therefore, `datenum` and `datevec` should be used for converting any large array of MATLAB time. If `time` is necessary to call often, it should be modified for optimum performance by identifying input and output type prior to performing any manipulations. The code for the function `time` is provided in Appendix A on page 179.

The MATLAB provided function `datetick` is very useful for graphing purposes. This

led the general time format in all FS-3 MATLAB-based analysis to be in the MATLAB date number format. By using this time as the x-axis data and running the `datetick` function, the x-axis is automatically formatted to an understandable format. If any zooming or panning functions are added, the `datetick` function should be automatically called when the data is panned or zoomed. This will be discussed further in the graphical interaction section beginning page 119.

FS-3 Attitude Analysis Tools

Several tools were developed by MIT SSL to analyze the attitude behavior of FS-3. These tools supported the verification of sensors, actuators, and software, as well as analyzing attitude data to obtain estimates and visualizations of the vehicle's attitude. This section will describe the different types of analysis done, and the use of the MATLAB tools to provide the results. Sensor verification will not be described as the magnetometer calibration methods have been discussed already, and actuator verification was explained previously when describe torque rod strength and polarity checks on page 99.

The SSL analysis team created several tools to verify the logic and commanding of the B-dot algorithm. An initial MATLAB script looked at the magnetometer data and performed the equivalent B-dot logic through a range of data. This logic included a five point averaging filter used on the satellite for filtering high frequency oscillations. As it was possible that this filter invoked a timing delay in the logic, it was important to demonstrate how the commanded output from the B-dot algorithm would relate to the actual rotation of the satellite. This B-dot script was implemented into the LimWOD GUI framework, and a button was added to allow for the visualization of changing B-dot commands in the graphical display window. This button calls the B-dot analysis program and identifies the time that the command to the torque rods would change if the B-dot algorithm was running. In this specific mode, this is only based off of the Z-axis magnetometer, so the commands will change upon a slope change in the Z-axis signal. Figure 4-22 shows the addition of the solid and dashed lines in the magnetometer graph display in the LimWOD GUI. This figure identifies both the button used to plot the lines, as well as the noticeable offset in the timing between the slope change and the command.

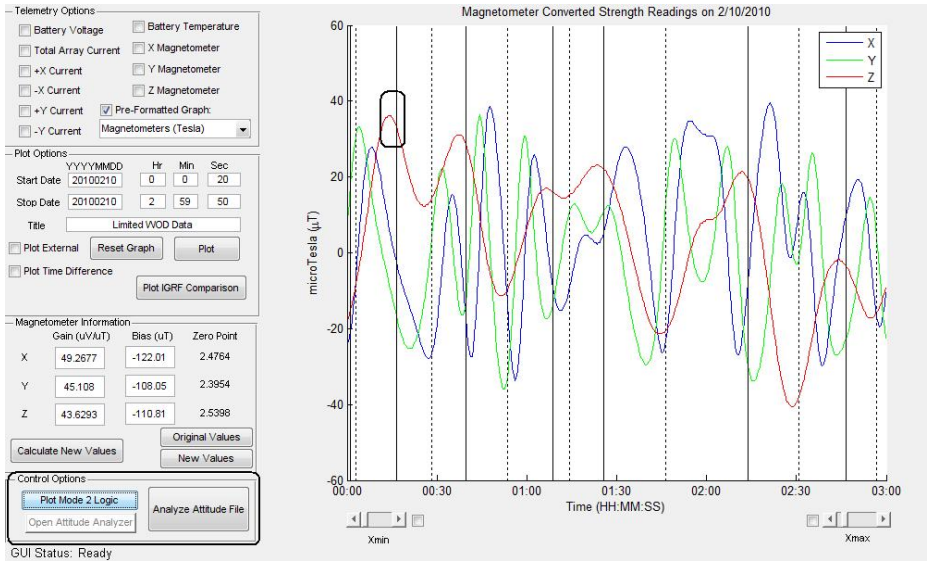


Figure 4-22: LimWOD GUI Screenshot of B-dot Commands

The satellite software was changed to lower this timing delay, as it is significant during periods of faster rotation. Additionally, the software was modified so that attitude log files were written to give the actual commands that the B-dot algorithm produces on orbit. This allows for verification of the command logic. A button was added in the GUI that allowed a user to select a set of attitude files to analyze. Upon selection, the attitude files were read into MATLAB and stored into a data matrix with the correct time tags to allow for comparison with magnetometer data and the previous prediction of B-dot commands. This was successfully implemented in March 2010, although the visualization of the data was performed in an external script that uses the data provided by the GUI. This comparison is shown in Figure 4-23. The solid line represents the expected command based on the magnetometer data, and the solid dots represent actually instances of commands as written in the attitude file. The results show that the commands are nearly exact between the two sources, with a small exception near the transition points.

It is important to note that although Figure 4-23 was created in a script, it relied on output generated from using the LimWOD GUI. This was done to ensure the analysis and graphing methods were accurate prior to implementing them in the GUI.

Attitude determination techniques to determine the angular position and angular velocity of the satellite were also performed on the FS-3 data by the SSL team. Katz developed a

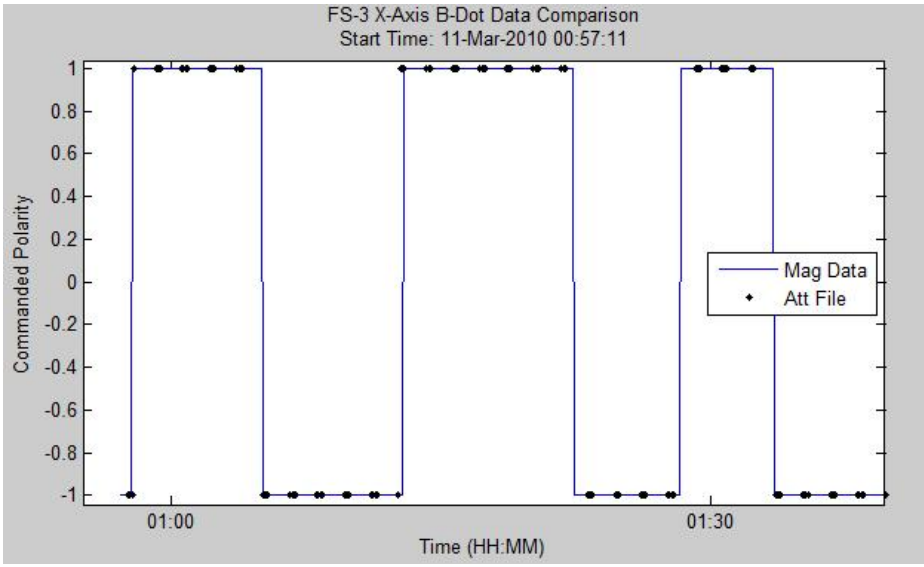


Figure 4-23: FS-3 B-Dot Logic Comparison with Attitude File and Magnetometer Data

Backwards-Smoothing Extended Kalman Filter (BSEKF) that sought to determine the attitude position and rates based solely on magnetometer data. The FS-3 BSEKF design was based on a BSEKF implemented on radar imaging described by Volpe [43]. This technique was very computationally expensive and was therefore written in a stand-alone MATLAB script; this script utilizes the common functions used in the GUI for loading data, propagating the TLE, and calling the IGRF model. The output of the BSEKF was analyzed visually in two ways: visualization through STK and with solar current comparisons in the LimWOD GUI. The STK visualization was performed by saving the time history of the attitude quaternion, angular velocity output by the BSEKF. This data was used to create STK attitude ('*.a') files. These files were loaded into a predefined STK scenario and the scenario was animated forward in time to see if the resulting attitude seemed plausible. Figure 4-24 shows a snapshot of this visualization technique.

The visual analysis performed in STK was limited to concluding when the filter clearly did not converge properly. This would be represented by sudden jumps in angular velocity, and could have been visualized equally well by looking at the continuity in the quaternions.

The additional analysis of the BSEKF output was done in the LimWOD GUI; this analysis method provided an improved source of verification versus the STK inspection. The LimWOD GUI was modified to read in the BSEKF output and then perform analysis to

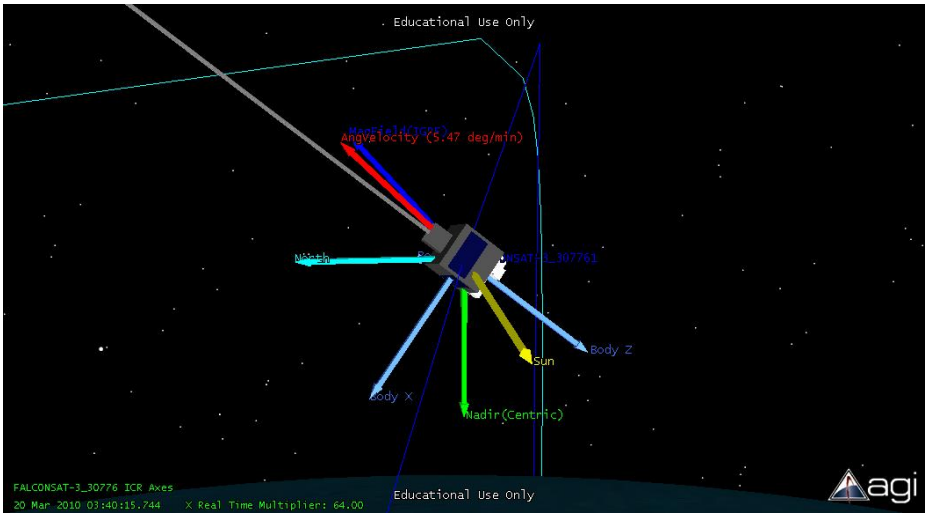


Figure 4-24: FS-3 STK Visualization

determine the expected angle between the sun and each solar panel face. In short, this calculation determines which solar panel should be receiving current at any given time. This data is useful as it can be compared to the solar panel current telemetry for an independent verification of the magnetometer-based attitude filter; this comparison is performed in Section 5.2.1.

Graphical Interaction

The graphical analysis of FS-3 data had proved to be of great importance by both unsuccessfully and successfully identifying and characterizing events. Therefore the LimWOD GUI was designed with usability, flexibility, and ease of graphical analysis in mind. The first requirement for the LimWOD GUI was to display and easily navigate through the different channels of telemetry in the Limited WOD files. In the initial GUI design, this was done by allowing the user to select the individual channels of telemetry desired to plot on the y-axis. The time axis (x-axis) limits were modified by inputting start and stop time limits in editable text boxes. As the graphs generated were not customized, an external plotting option allowed the figure to be generated outside of the GUI. This allows the user to use an assortment of plotting customization tools available with any MATLAB figure. Figure 4-25 identifies these options.

Figure 4-25 shows an example of graphing battery voltage along with magnetometer

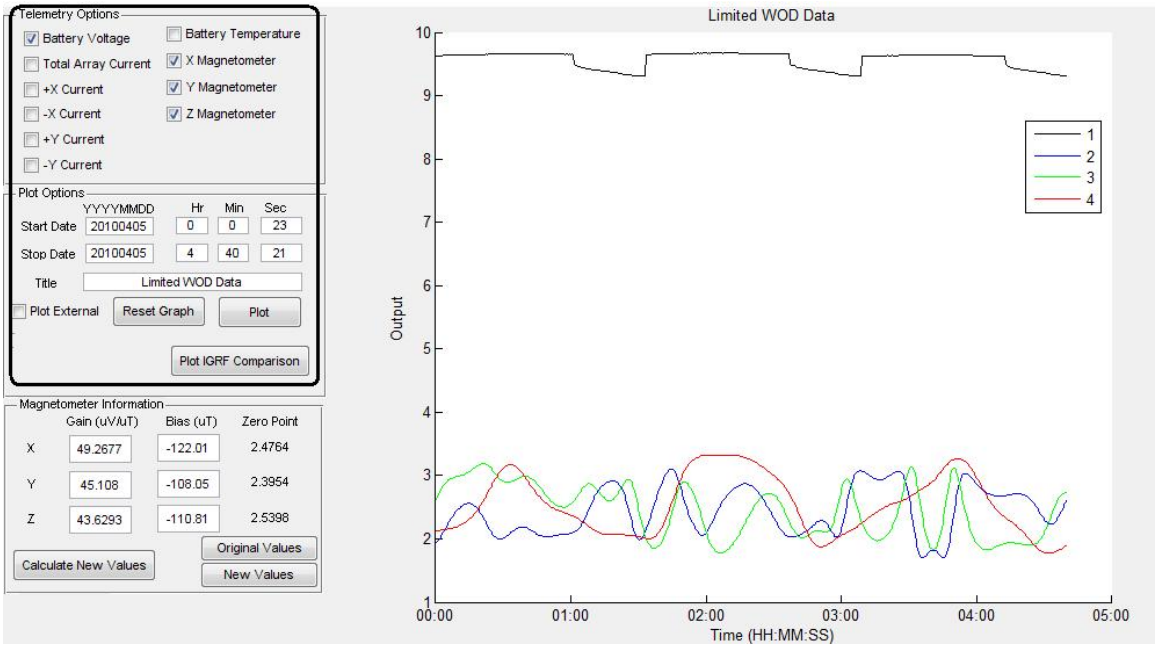


Figure 4-25: Initial LimWOD GUI Capabilities

data. The graph shows that the legend and Y-axis are not specified, which can lead to confusion to a user. While it is desirable to have flexibility in plotting the individual channels, the developer was not able to design a flexible yet detailed design in a short amount of time. While this was partly solved with the external plotting option, it had two major downfalls: the lack of changing resolution in the timescale during zooming and the amount of effort necessary to create repeatable graphs. This inspired the addition of selectable times and pre-made graphs, which were added in the next version. The ability to change the time of the data was improved in the second revision by adding two check boxes, representing the minimum and maximum desired time, which allowed the user to click on the graph at the desired start or stop time to set the respective time in the same editable text boxes. The pre-made graphs were added to quickly generate common sets of graphs that were used by analysts at USAFA and MIT. The results of the second revision are seen in Figure 4-26.

The second revision included the addition of many basic MATLAB plotting functions to make the graphs generate clear, detailed results automatically. A summary of these modifications, numbered in Figure 4-26 is included. While not shown in this graph, this

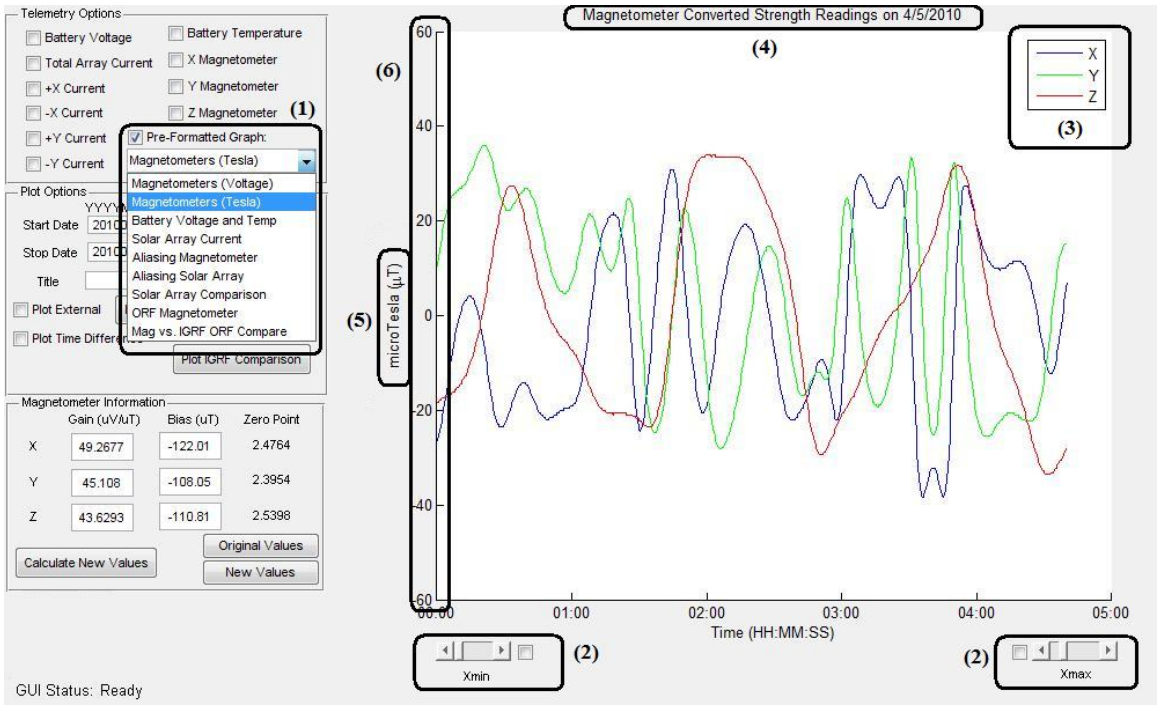


Figure 4-26: Updated LimWOD GUI with Enhanced Graphical Interface Options

version also created data points on each value if there were under 300 points in the graph so that the detailed points could be displayed for small amounts of data while not saturating large amounts of data.

1. Pre-loaded templates of graphing. Includes basic collection of common telemetry plots along with graphs created with additional analysis techniques not available with the previous functions
2. Slider and check box to automatically set start/stop date by clicking on the graph
3. Formatted legend specific to the graph
4. Title with automatic start and stop date text generation for clear results
5. Specific y-axis label for data selected
6. Set axis limit within bounds of sensors to allow for clear comparison in values between different dates

This revised GUI allowed users to quickly produce graphs for the three basic sets of data (battery, magnetometer, and solar currents) clearly and in a repeatable format. Additionally, the ability to click on the graph and reset the time limit allowed for much quicker analysis of a large amount of data.

Most updates made in the GUI after this revision were to increase the analysis capabilities with the B-dot logic, BSEKF loading and solar panel comparison, and attitude file loading and processing. However, a few small yet significant changes were made that allowed the GUI to be greatly improved in usability. These changes were made after the developer discovered the MATLAB toolbar menu in the GUI options. The GUI was created with MATLAB's GUI Development Environment (GUIDE), and while the developer had developed MATLAB GUI's in the past (such as the MOTV GUI), a few capabilities were not realized until after the second version of the LimWOD GUI was finished. This toolbar menu allows the option to zoom in/out, pan, and select data with the data cursor. While these tools are available by plotting data external of the GUI, the time formatting produced issues with zooming and data labeling. Previously when data was zoomed in or out, the time limits would not be updated. This made identifying the time step of large quantities of data to be difficult. The clickable time limit option made it possible to identify the data and re-plot it in the GUI, but was not desired to be a standard procedure for all users. The data labeler also was not desirable, as it shows the MATLAB date number instead of a date string in the tooltips.

The updates to the final GUI therefore were the addition of the zooming, panning, and data identification features into the GUI, as well as fixing all of the issues with the time. Several functions were added as callbacks so that both zooming and panning the data updated the x-axis labels and date ticks, as well as changing the datatips to display the date, time, and y-axis output. These timing properties are also passed to the external plots, allowing for full flexibility to the user to sort through data and create the desired graphical products. The only addition of future work that has not been satisfied is the updating of the x-axis time label as the date changes. Currently it attempts to display the a selected format of MM/DD HH:MM:SS based on the amount of time being graphed, but this is not a robust method and could be improved to make the graphs as detailed as possible. The final version of the GUI with additional features displayed is shown in Figure 4-27. Appendix A, page 181 provides the MATLAB code that can be used to set the graphical time, zoom, and pan features.

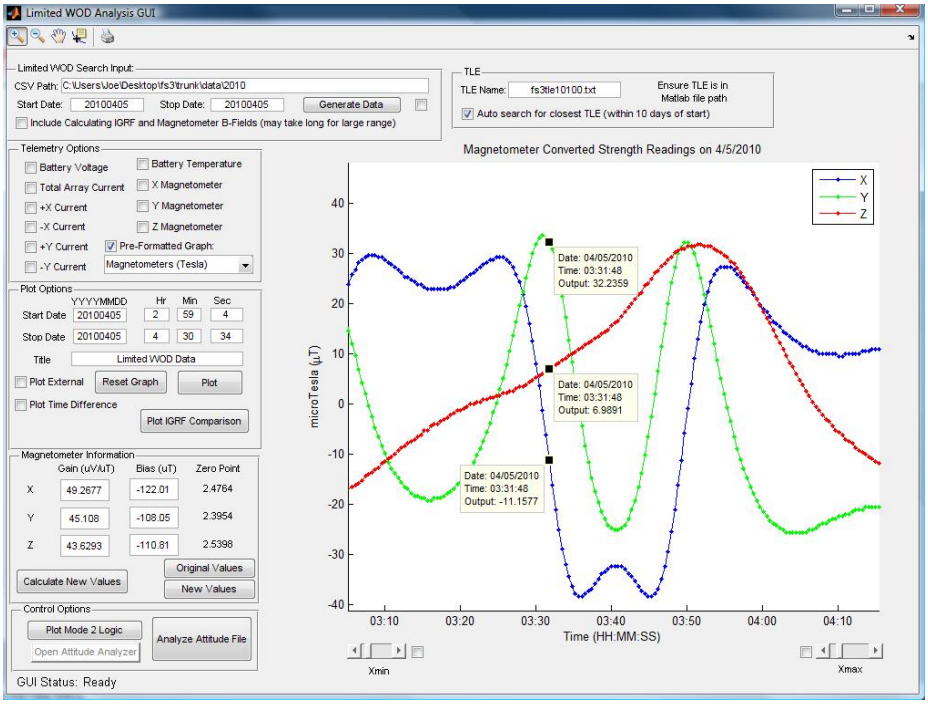


Figure 4-27: Final LimWOD Graphical User Interface

The development of the interactive and graphical nature of the LimWOD GUI has several broad lessons that can be used in a general TDA design. These include:

- At a minimum, the TDA should be able to plot different data channels over time
  - Desire to have an ability to quickly sort through a large array of time
  - Critical to use correct time steps in plotting data
  - Leverage COTS SW products such as Excel or MATLAB that have built-in tools
- An interactive GUI design allows for easy graphical production by many users
- The development time and features of the TDA graphical design are a function of developer's knowledge and abilities

4.2.4 FS-3 Telemetry and Data Analysis Tools Summary

This section began by discussing the different data analysis tools used in the FS-3 program during operations, and then described several major operational events. The relevance of these events to the IEM topic, especially the TDA, is the decision of the type of analysis tool to use. A trade off occurs between the ease of use, accuracy, and the amount of effort necessary to generate results. This section has identified how three years of on-orbit

operations and experience has yielded a variety of telemetry display and analysis tools, from basic Excel plots to a interactive GUI design. An overview and of these tools is shown in Figure 4-28.

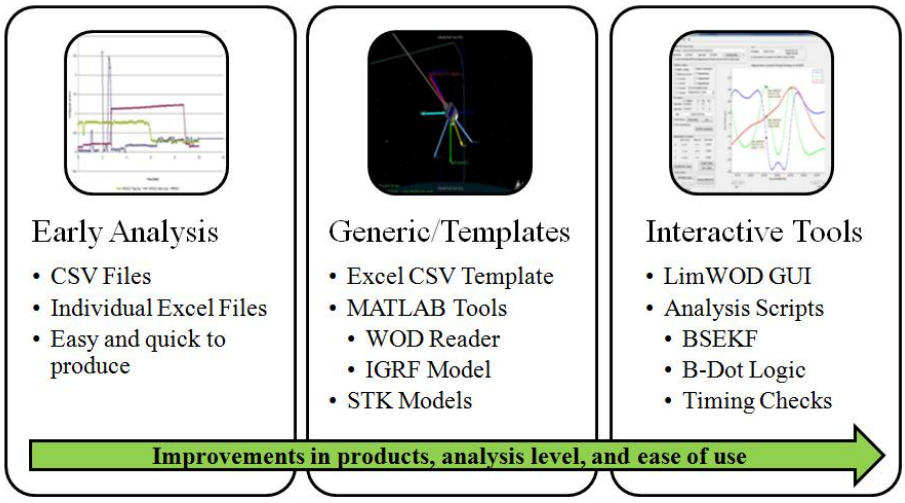


Figure 4-28: Summary of FS-3 Data Analysis Tools

It must be stressed that data analysis programs were available and ready for use for FS-3 prior to its launch, and the analysis plan met the requirements for the FS-3 mission. However, due to improper time stamps, the primary database that stored and plotted telemetry was not sufficient to analyze data. Additionally, the addition of continuous LimWOD collection and the growing duration of the mission created a significant amount of data to parse through. The additions of new analysis tools such as the Excel templates, the MATLAB tools, and eventually the LimWOD GUI were done to create data products quicker and easier, as well as to allow for greater analysis of the data.

In summary, previous successful and unsuccessful graphical analysis techniques and lessons were used to develop an accurate, usable, and flexible graphical tool through the LimWOD GUI. Through nine months, the GUI has been upgraded to allow for improved analysis techniques and enhanced usability. The final version allows a user to quickly go through days of data in a matter of seconds. An unintended final test of all FS-3 analysis tools occurred to analyze a major FS-3 event in early April 2010, and is discussed in Section 5.2.2.

4.3 CASTOR MAT Design

The CASTOR mission success will be measured by analyzing the trajectory of the satellite through time. However, as previously stated, this is not a trivial task due to performance limitations on the CASTOR satellite. This section will describe the baseline MAT design and the intended methods used to determine the thruster's performance. It will also explore a general design of a low thrust trajectory tool that could be useful for similar classes of satellites.

The topic of low-thrust and electric propulsion trajectories has been researched at an increasing frequency over the last thirty years. A significant amount of this work has been performed on optimization methods for low-thrust transfers. The first and simplest class of these transfers is the constant thrust, spiral trajectory case. This is one of the only closed form solutions, and can be completed utilizing the method of variation of parameters [3, 41]. Kimbrel described additional analytical solutions to the three-dimensional GTO to GEO trajectory case, assuming constant thrust and no disturbances [20]. Kimbrel's thesis also lists a variety of documents focused on low-thrust optimization.

Software tools have also been created through the years to provide flexible numeric simulations to optimize different trajectories, such as SEPSPOT (or SECKSPOT) [33], MITEOR [20], or recently Copernicus [28, 19]. NASA's Copernicus software represents one of the leading trajectory optimization tools, as it allows a user to select the wide variety of vehicle, environmental, thruster, and operational conditions and constraints for a given problem.

Each of these optimization tools and techniques have a driving assumption that the satellite will be designed to meet the given conditions assumed in the case. This assumption is not always true, as is the case for CASTOR. The CASTOR satellite is designed with the understanding that certain assumptions that are typical of more complex satellites, such as accurate pointing, are not designed in the system due to a resource burden. Additionally, the power system limitations of the satellite do not allow for continual thrust over the entire orbit or even the sunlit portion of the orbit. These assumptions also do not hold in the case of FS-5, where the purpose of the propulsion systems on board is to create the source

for characterization sensors for its plasma sensors. The ΔV created from the operations of the propulsion system is significant nonetheless, and must be predicted and measured for successful operations. In these cases, an optimal control technique is less important, and a trajectory prediction or estimation tool is more important.

The trajectory analysis tool for the CASTOR MAT should perform the following two main functions:

- **Orbit Prediction:**
  - Feed dynamic thrust and attitude data into the model
  - Utilize models of the LEO environment and its disturbances
  - Predict new orbit
- **Thruster Characterization:**
  - Feed actual thruster state and attitude data into model
  - Accurately model the actual LEO environment
  - Characterize thruster performance (i.e. thrust, efficiency)

Inherent in this tool are the satellite properties, which includes the mass, area, and drag properties necessary to ensure an accurate prediction of the disturbances on the spacecraft. The predictive model aspect of the tool is used to predict where the satellite will be at any point in time given its thrust and attitude inputs. This is equivalently the same as the SIM with command and attitude input into the detailed orbits model. The thruster characterization function is different than the SIM, as it is used to determine the thrust. It is possible to utilize tuning parameters and multiple runs of the SIM to attempt to match the thruster properties to the actual orbital changes if the capabilities exist in the TDA/SIM environment. A stand-alone MAT tool was initially planned to perform these tasks independent of the SIM, so that it could be tested on different satellites (such as FS-5). Additionally, it is important to calibrate the disturbance models on-orbit based on actual observed changes to the orbit during periods without thrusting (such as commissioning). This will allow for increased resolution in the distinction between orbital changes caused by the thruster and those caused by the orbital environment.

The CASTOR MAT should also contain a plume image analysis tool, which will store and process images taken of the DCFT's plume during operations. It is expected that

analysis methods will be created to study the properties of the image, such as the plume divergence angle and color. These analysis methods should be contained in the MAT, and it may be useful to create expected environment images as a function of time (orbit location) and attitude, to quickly check if the camera is pointing at a bright light source like the sun or the Earth.

Verification of the trajectory tool should be done utilizing either COTS programs or flight data from pre-existing LEO low-thrust missions. COTS programs such as FreeFlyer or STK will allow input of maneuvers and attitude through programming interfaces and user-defined scripts. The biggest factor in these results is the model of the orbital environment and propagation methods. The trajectory tool can also be verified with flight data from another mission, although one must find an existing mission that provides the data openly.

In conclusion, a stand-alone MAT tool for CASTOR is still in the very early stages of design. This tool would not optimize the trajectory of CASTOR, but rather attempt to determine the thruster's performance for the given attitude and propulsion system inputs. While no stand-alone tool exists, the SIM is able to perform many of the tasks with its command-driven and telemetry-input capabilities. This is not as desirable as a stand-alone and modular tool, but it should be enough to successfully determine the performance of the thruster. Additionally, any development of an image analysis tool should be incorporated in the CASTOR MAT and SIM.

Chapter 5

Results and Analysis

The IEM design approach uses integrated modeling throughout a satellite design and operations sequence. This chapter looks at the results of IEM applied principles or applicable examples throughout the satellite life cycle. The use of integrated modeling in the design is described in Section 5.1 through previous integrated model results, as well as IEM-based results. Section 5.2 uses FalconSAT-3 to show the applicability of a basic TDA element in the operations phase. Section 5.3 will provide a discussion of key principles and lessons learned through the IEM development and implementation.

5.1 Integrated Model Results

This section will describe the results of the integrated models described previously in Chapters 3 and 4. This includes the results of models performed prior to IEM creation (MOTV and initial CASTOR models), as well as models utilizing the IEM approach (ExoplanetSat and new CASTOR model). The results will describe how the integrated models assisted in the satellite development effort, how well they met the IEM principles, and if there are any additional lessons that can be taken relating to the use of modeling and simulation in the design process.

5.1.1 Phase A — ExoplanetSat Model Results

The ExoplanetSat example represents the only implementation of the IEM design by a party other than the author. This was done to both support the ExoplanetSat team as well as gain one data point of validation in the IEM design. As described in Section 3.2.2, the mission of ExoplanetSat is to detect Earth-like planets orbiting bright Sun-like stars using the transit method. This mission is being designed in the volume constraints of a standard 3U cubesat (10x10x34 cm) [26].

The first step in the Phase A IEM process is to quantify the mission objectives with FOM and subsequently create a plan to measure and record the FOM. The ExoplanetSat team determined that the key figure of merit for the mission to be the total system intensity noise for the target star [35]. Specifically, the maximum total system intensity noise threshold was set at 10 parts per million (ppm). This FOM was chosen as the transit of an Earth-like planet around the star of interest would create a temporary decrease in the intensity of the star. The figure of 10 ppm was chosen as it is ten times lower than the expected intensity decrease needed to identify the star. If the noise floor is too close to the identification limit, it is not possible to decisively state that a planet has been found.

These FOM are specified into the different noise sources that must be characterized to determine the value for the FOM for different designs. These primary noise sources include shot noise, read noise, dark current noise, and jitter noise. The first three noise sources are due to the optical properties and uncertainties of the imaging system or light intensity, and the jitter noise is a function of the spacecraft's stability while imaging. The ExoplanetSat team has identified these different sources and is performing both modeling and testing on the different aspects of these noises.

The second step of the IEM Phase A process is to identify the major payload specifications and requirements. The ExoplanetSat team identified a COTS Single Lens Reflex (SLR) camera lens with a CCD detector. The specifications desired for the lens and detector were known prior to performing the IEM process and are shown below.

- The lens shall have an outer diameter no greater than 90 mm
- The lens will ideally be less than 100 mm long

- The lens aperture will need to have as large an aperture as possible; this translates to a low F-number lens
- The lens mass will ideally be less than 600 g
- The CCD will have as low dark current noise as possible
- The CCD will have as low read noise as possible
- The CCD will have at least 25 square degrees of sky coverage

These specifications include volume, mass, and performance specifications. However, not all specifications are fully defined, as the total system is still in the late concept and early design phase. Ideally, these specifications would be requirements with clear numbers rather than "ideally" and "as low as possible;" however, the payload team understands the challenges of integrating the entire system in the small size of a cubesat, so they identified their preferences. By putting these specifications on paper, others are able to give feedback to the team, such as the necessity to define threshold requirements for certain interfaces (e.g. the lens mass shall not be greater than 700g).

The third step in the IEM process is to identify the primary subsystems and mission parameters. This was explained as having three main steps: determination of uncertainty for each system, identification of interface requirements, and an estimation of the detail required in the model. The ExoplanetSat utilized the same rating and selection metrics shown in this thesis to determine which subsystems and parameters to model first. Table 5.1 shows these results.

The ExoplanetSat team realized it had many technical challenges that it must solve. The two paramount systems that have been worked on longest are the detection (payload) system and the ADCS system. These two scored very high as expected. However, due to the experience that the team already had with these subjects, the highest scoring subsystem or parameter was the thermal subsystem. Although the importance of the thermal characteristics was a topic being investigated by the team, the results of this weighting were nonetheless surprising. The team decided on setting a threshold score of ten to model for the initial integrated model, which includes the thermal, ADCS, and payload systems. It should also be noted that this rating system identified a key programmatic issue of having many complex and important integrated systems with very little experience or background

Table 5.1: ExoplanetSat Integrated Model Elements Decision Selection

Subsystem	**Importance (0 or 1)**	**Uncertainty (0 to 4)**	**Detail Level (0 to 4)**	**Integration (0 to 4)**	**Score**
Power	1	4	2	2	8
Thermal	1	4	3	4	11
Avionics	1	4	1	4	9
Comm.	1	3	1	1	5
Guidance	0	4	1	1	0
Attitude	1	2	4	4	10
Structure	0	3	1	3	0
Payload	1	2	4	4	10
Mission Parameters	**Importance (0 or 1)**	**Uncertainty (0 to 4)**	**Detail Level (0 to 4)**	**Integration (0 to 4)**	**Score**
Orbit	1	1	3	2	6
Mass	1	1	4	4	9
Volume	0	1	4	4	9
Cost	0	1	3	1	0
Lifetime	1	1	3	1	5
Operations	1	1	2	4	7

knowledge on them. While this was known to certain team members, it was never numerically rated until this study.

The fourth requirement in the five-step list is to determine the integrated model platform and supporting personnel and resources. The ExoplanetSat team chose MATLAB as the primary analysis platform. The first cited reason for this choice is that the optical simulation frequently performs many operations, such as FFT's, convolution, and rotations, during any simulation run. Additionally, many different students have access to MATLAB and it can be run on nearly everyone's computer system. Lastly, the ADCS model and optical models developed by SSL Graduate Students Chris Pong and Matt Smith were already being developed in MATLAB. Note that this reasoning was last, as it is still early in the program, and alternative simulation and modeling tools exist. Some of these other tools that have been researched and are under consideration from the team for use are Zemax and Code V for optical simulations and Simulink for attitude simulations.

The final step is to complete the first integrated model. The ExoplanetSat team currently has a quasi-integrated model, meaning the results of the attitude model into the optical model. While this may be sufficient if no feedback exists between the imaging sys-

tem and the attitude system, this framework is very limited to future growth. Lastly, the thermal subsystem was identified as a major element for the model, and therefore a preliminary thermal model should be included in the integrated model. Two types of thermal modeling routines are being looked at by the ExoplanetSat team: utilizing COTS thermal software such as Thermal Desktop, or utilizing a MATLAB-based thermal design code for small satellites created by MIT masters student John Richmond [30].

5.1.2 Phase B — MOTV Results

Introduced in Section 3.3.2, the MOTV model provided the first integrated verification of student-designed MOTV satellite. Preliminary models in the ADCS, GNC, thermal, and power subsystems and a orbits model allowed for many operational scenarios to be run. Major input options included the desired attitude or pointing mode for the satellite, the frequency of guidance updates necessary, and the types of maneuvers desired. Additional subsystem characteristics such as solar panel area or thermal properties also greatly affected the final results. An example of the full mission lifetime is shown in Figure 5-1. The total satellite lifetime in this run was 186 days with a total expected ΔV of 625 m/sec utilizing 2.6 kg of Xenon fuel.

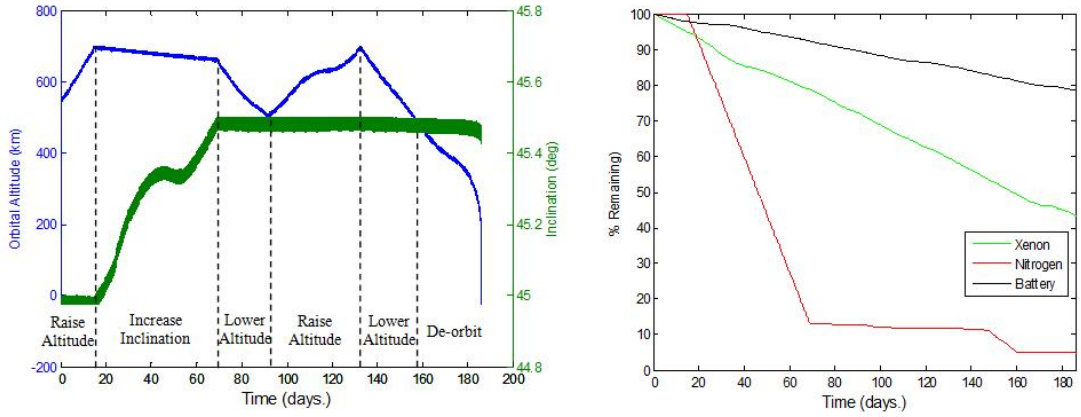


Figure 5-1: MOTV Simulated Mission Output [32]

Design Impact

The MOTV model identified two major design issues with the satellite. The first issue identified involved the attitude system's lifetime and performance characteristics. The design called for a set of nitrogen thrusters, one reaction wheel, and a gimbaled engine to provide pointing control to the satellite. The large magnetic dipole from the permanent magnets in the DCFT ($\approx 24\ Am^2$) led to a large disturbance torque that the nitrogen thrusters had to fight at a much higher rate than expected. Without any cancellation of the engine's magnetic dipole, the simulation showed that nominal operations would lead to depletion of the 1000 grams of nitrogen gas within three days. This result led to a design change, as well as a change in the CONOPS. The design change called for a permanent magnet with an equivalent dipole mounted in the opposite direction to cancel out the magnetic effects. Additionally, the satellite CONOPS was modified to eliminate control during eclipse and to use the engine gimbal as the primary source for pointing.

The second major design issue involved the thermal design for MOTV. The preliminary thermal results showed that the satellite temperatures greatly exceeded their bounds. Some question was placed in the validity of certain model elements, as it was not validated against any standard software. Regardless of the validity of the model's result, the modeling process and interaction with the team indicated that the thermal design of the satellite had a large amount of uncertainty. An example was determining the thermal coefficients of absorptivity or emissivity used for the composite solar panel structure. This design was largely unknown, as the carbon-fiber face sheets, aluminum honeycomb, solar cells, and epoxies all had different properties, many of which were unknown. By attempting to model this system, it became clear that more work needed to be done on this assembly, as well as the thermal analysis as a whole.

As seen in Figure 5-1, the inclination change maneuvers forced slew maneuvers about the axes orthogonal to the reaction wheel, causing a large consumption of nitrogen for a 0.5 degree inclination raise. The nitrogen usage caused the satellite (via the simulation) to initiate it's de-orbit procedures (when nitrogen remaining fell under 10%), even though over 40% of the xenon gas remained. This result forced the question of the utility of performing

inclination changes in the mission.

These results all affected the CASTOR satellite design. The attitude system changed to three reaction wheels, with magnetic torque rods used for momentum dumping instead of nitrogen thrusters. The thermal system was identified as a critical system, motivating students to pursue aggressive modeling and thermal testing campaigns at the component, assembly, and integrated system levels. Lastly, inclination changes were eliminated from the mission to simplify operations as well as providing more time to characterize the thruster over many repeatable sets of maneuvers.

Modeling and Simulation Results

The MOTV model accomplished the majority of its initial requirements. It established the feasibility of the satellite design through integrated modeling. Additionally, simply asking detailed design questions to the undergraduate team led to the identification of unknown issues. The simulation was executed inside of an interactive MOTV GUI, which allowed the user to select the different simulation cases, change subsystem parameters, and eventually analyze results. These results included a summary of mission data including average specific impulse (Isp), total ΔV, and ΔV lost due to drag. Additional preloaded graph templates of subsystem health were also available to easily display results.

While accomplishing its requirements, many lessons were learned in the implementation of modeling and simulation during the design process. One of the first issues that had to be solved is the separation of work and the integration methods for ensuring that individuals could work independently on their subsystem and re-integrate into the simulation. The process in place was for each subsystem model to be in a stand-alone MATLAB m-file and to set all parameters that a user could select in the initial GUI as global variables. This process worked fairly well, as long as only one person performed model integration tasks.

However, one certain integration issue was found that occurred even when all errors were fixed in the simulation. Certain simulation cases were desired to be run to test updates to a model, such as an upgraded ADCS model, were impacted by other models, such as a temperature going out of limits in the thermal model. These issues made it much more difficult to test new methods or detail in certain models. The first solution to this class of

problems was to change the input conditions to make the model perform as desired. This was done in the lifetime simulation shown by modifying all thermal properties to ensure the system stayed in an acceptable temperature and changing the engine dipole to 0.1% of the original strength. The second type of solution, which also allowed for faster run-times, was to hard code in values inside the while loop that runs the simulation, and commenting out the undesireable subsystem or model in the code. This solution, while temporarilly effective, created unclear code and often negatively affected results two or three days after the temporary fix was implemented.

This integration issue was one of the driving factors for adding the switching function early into the IEM design. The ability to switch models "off" allows users to have flexibility in implementing new features as well as improving the run-time for specific simulation cases.

5.1.3 Phase C — Early CASTOR Simulation Results

The MOTV integrated model was rewritten for the CASTOR satellite in Fall 2009. This objective for this model was to update the MOTV model to the CASTOR design and increase the depth of the model. As mentioned in Section 3.4.2, this model added several features and details that were not present in the MOTV simulation design. This section will describe several of the added capabilities and a description of how the results from these additions impacted the IEM design framework.

Added Simulation Features

The first class of updates are modifications to match the new design. One example is the change in solar panel configurations between the two designs. The previous design featured sun-tracking solar panels, which allowed for certain assumptions in the model. With the lower risk CASTOR satellite design with fixed solar panels, the power system now needed the solar panel incidence angle to calculate the available solar power. Similar updates were necessary in the ADCS configuration change from nitrogen thrusters to reaction wheels and torque coils.

The addition of new subsystem models is the next type of updates to the previous MOTV model. Communications and avionics models were added in the simulation upgrade. The

communications model was used heavily in selecting the best configuration of antennas and their placement on the satellite. This model used the ADCS output to determine communications coverage during nominal operations as well as tumbling cases. This integrated simulation allowed the CASTOR communications team to identify the best antenna selection and configuration. This configuration is not intuitive as it utilizes a higher gain antenna with a more directional radiation pattern as the primary antenna for the tumbling case. Only with the integrated simulation could it be shown that the average coverage time increased with the higher gain antenna. An avionics model was also added to support the design decision to lower the amount of computational power available on the satellite to save on mass, volume, and complexity. This model tracked the number of processor operations necessary for each time step based on subsystem activity. Additionally, a data and memory budget were added to the model to support the implementation of an imaging system on CASTOR.

Existing subsystem models from the MOTV simulation were enhanced to include a greater amount of depth in the CASTOR model. The attitude subsystem will be described to show the further amount of depth in the model. Figure 5-2 shows the angle between the axes normal to the solar panel face (+Z axis) and the sun. This graph illustrates the dynamic nature of the controller modeled in the CASTOR simulation. Additionally, this graph also shows how the satellite is modeled to rotate into a velocity-fixed orbit during thrusting, and how the angle difference approaches zero at local noon. The attitude model grew off of the previous model with disturbance torques with one reaction wheel, and added the additional reaction wheel and torque coil actuators into a closed loop controller.

Another feature added to the simulation design was the command input capability. This capability allowed certain commands to be read and executed in a defined time window. Described in Section 4.1.2, this feature operated fully "on" or "off" for a select number of major mission commands. If a command input file was loaded, the pointing mode and thruster state would be decided based on the command intervals listed. If no command file was loaded, the simulation would utilize the operational logic coded into each subsystem to change these states. The reason for adding the command input feature was the same as the reason it was placed into the IEM design: to allow actual satellite behavior to be

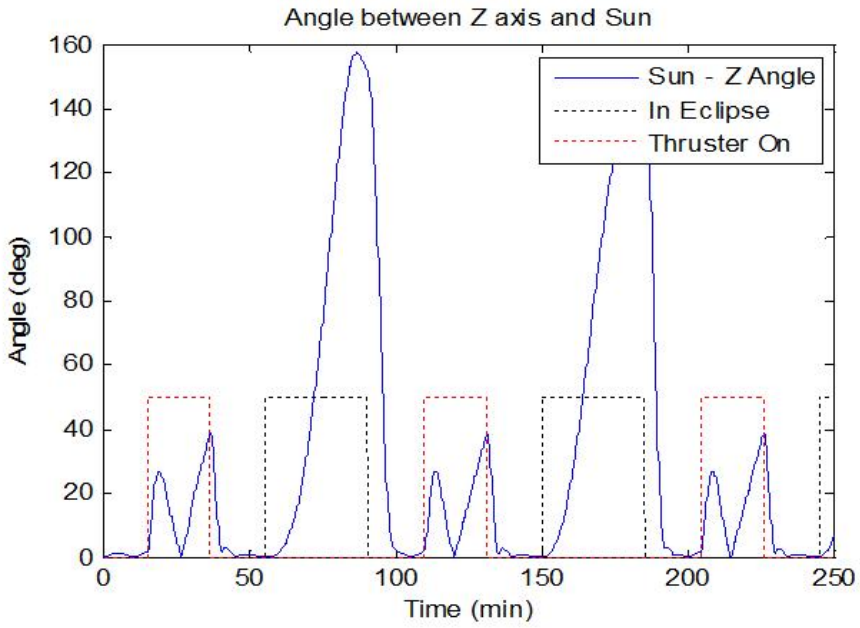


Figure 5-2: CASTOR Integrated Model Attitude Output [7]

simulated most accurately. The addition of this feature allowed for the identification of its challenges and limitations. These challenges resulted in the significant re-design in the simulation architecture to meet the IEM requirements.

Lastly, a log was added to the model that stored all data that deviated its bounds. This can be seen in Figure 5-4 in the next section describing model output. This addition was expanded with the event log in the IEM-motivated CASTOR simulation.

Model Development Feedback and Observations

This example is unique as the author did not have any direct involvement with the model creation or coding. This section is used to discuss feedback of the model developers on both the utility and challenges of the simulation development. All four of the developers were CASTOR team graduate student mentors and therefore all of the developers were knowledgeable with the satellite design. This was advantageous as the simulation features were built directly to support the design of CASTOR, allowing the developers to perform design trades during model creation and modification.

The communications and avionics model developer, 2LT Matt McCormack, discussed the benefits of creating the integrated model in support of the CASTOR teams he mentored

[23]. The communications analysis was significantly affected by changes in the satellite's attitude, and therefore the integration with an attitude simulation was critical to verifying the design. Additionally, the avionics model assisted in determining the type and number of processors necessary to complete the mission with necessary margin. However, discussions with 2LT McCormack regarding the future IEM design led to the omission of the avionics model in the final version. It was decided that while important in identifying the proper design of the avionics system, the avionics model would not have added any detail to the model in its current form. The options were to increase the fidelity and detail of the avionics model to characterize the software and hardware interaction, keep the current model in the simulation, or remove the system from the model. The personnel cost necessary to improve the model exceeded the expected utility, and since the existing model would output the same results for each orbit, the model was removed.

The ADCS model developer, 2LT Corey Crowell, described the benefits of writing the ADCS model as part of the integrated model. He stated that an ADCS model was necessary to create at the time, even if the integrated model was not being assembled [6]. 2LT Crowell cited the sizing of the reaction wheel and the torque coils as one of the primary design decisions requiring an ADCS model. While certain preliminary design references and techniques exist, such as referencing SMAD, the team needed to show that its design was feasible for their particular mission. However, 2LT Crowell readily acknowledged that the integration of the ADCS model into the simulation framework was difficult. In particular, the change to object-oriented MATLAB forced him to make his model compatible with the new structure. His process for making any changes in the integrated model was to test his new methods in a stand-alone ADCS script first, and then attempt to copy and paste into the integrated simulation object-oriented model. The code could not be simply copied and pasted, requiring significant debugging effort for each update. While the resulting code successfully worked with in the simulation framework, the transition from a functional programming technique to object-oriented actually resulted in more confusing and slower code.

The MATLAB introduction to Object-Oriented Programming describes how OOP is is used to manage software complexity by using classification systems and design patterns

in programming [9]. OOP is meant to simplify the structure and readability of code and therefore allow for ease of modification or maintenance. However, past studies have shown that OOP is not necessarily easier for novice programmers, such as those found in the university environment [42]. The initial CASTOR design demonstrated that OOP advantages are not realized if the developers of the code are not familiar with the programming techniques. A discussion summarizing all lessons learned with OOP and changing programming techniques during the program is given in Section 5.3.1.

A few general comments can be made from the CASTOR simulation's code and its comparison to the previous MOTV model. The first comparison deals with the run-time between the models. Both simulations were given same initial conditions, specifically with start time, stop time, and step size. A step size of 20 seconds was used for a 12 hour duration. The MOTV model was over 8 times faster than the CASTOR, a comparison of 0.8 minutes and 6.8 minutes. These were completed on the same machine one after the other to try to keep the simulation computer's resources identical. The results clearly show that the MOTV simulation runs faster than the CASTOR simulation. Figures 5-3 and 5-4 display the differences in the resulting output from the MOTV and CASTOR simulations, respectively.

The slower run time can be attributed to the added depth in the models as well as the programming structure. Of these two, the programming structure and simulation architecture was the driving factor to the increased run time. The model used OOP techniques in a way to maximize readability and clarity in the code, but it sacrificed performance as a result. Through the semester of coding, it became increasingly important to generate accurate results and less important to produce efficient code. To solve integration concerns, it was decided to pass all subsystems objects as inputs to each other, and to store all data in two data-output objects. These objects, called `Tagged List` and `State Vect` were identified using the MATLAB profiler tool as the most costly in terms of performance, shown in Figure 5-5.

The cause for these slow performance is that these objects called lower-level operators, such as opening and writing to a file, over 1 million times in the 12 hour time simulation. Figure 5-5 show that six of the top eight slowest functions were due to data distribution.

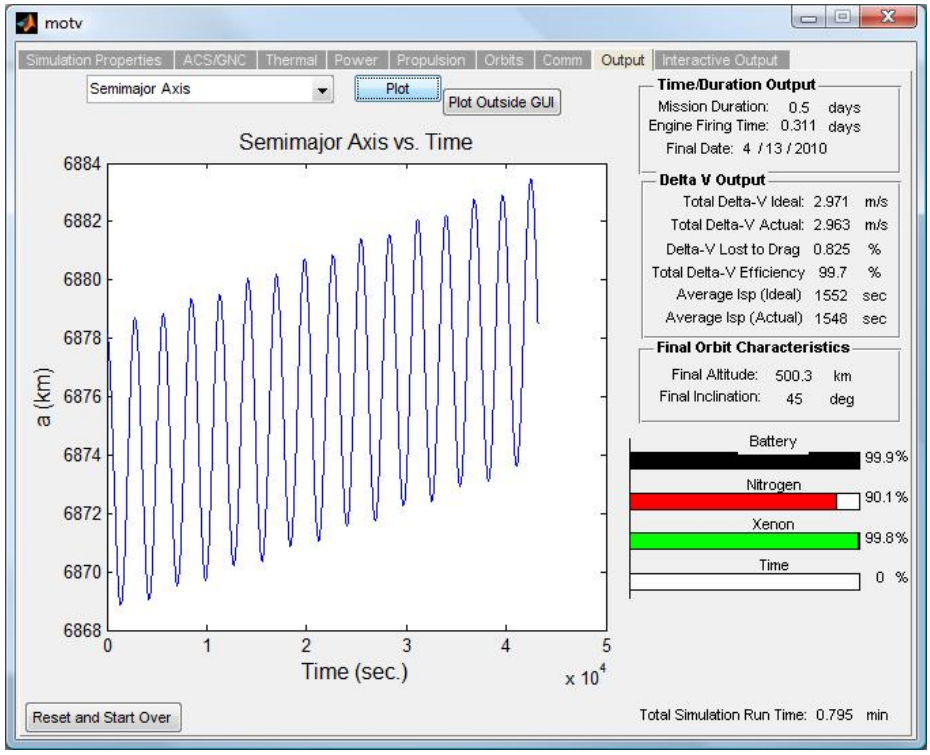


Figure 5-3: Screenshot of MOTV Simulation Output

```
Current Progress: 0%
Current Progress: 10%
Current Progress: 20%
Current Progress: 30%
Current Progress: 40%
Current Progress: 50%
Current Progress: 60%
Current Progress: 70%
Current Progress: 80%
Current Progress: 90%
Current Progress: 100%
WARNING: Null (IGNORE) tag for  at Tagged_List:28 for t=0 by 0
WARNING: Above Max Temp Rqmt tag for Battery at Sys_Thermal.deriv_gen:421 for t=40 by 300
WARNING: Below tag for Minimum SNR at Sys_Comm.comm_qual:610 for t=2260 by 3.0217
WARNING: Below Min Temp Rqmt tag for Solar Array 2 at Sys_Thermal.deriv_gen:391 for t=328
WARNING: Below Min Temp Rqmt tag for Solar Array 3 at Sys_Thermal.deriv_gen:400 for t=328
Elapsed time is 411.477221 seconds.
>>
```

Figure 5-4: Screenshot of CASTOR Simulation Output

The slow run time results led to significant simulation changes in data distribution and storage in the updated CASTOR design.

The increased run time is also due to the added depth of the model. In general, this increase is inherent as increased fidelity will add a larger amount of operations necessary to perform. To speed up the simulation time, one would need to simplify the model and reduce the amount of operations performed. This is done in the general IEM method and

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
Tagged_List>Tagged_List.unique_item	1497	217.244 s	217.244 s	
State_Vect>State_Vect.push_state	123120	39.897 s	39.897 s	
Tagged_List>Tagged_List.new_item	1496	33.930 s	33.664 s	
Sys_ADCS>Sys_ADCS.deriv_gen	2160	64.871 s	18.170 s	
Data_Structure>Data_Structure.pushcur	2160	13.242 s	11.005 s	
State_Vect>State_Vect.grab_state	32400	10.297 s	10.297 s	
Sys_ADCS>Sys_ADCS.pderiv	8644	12.066 s	9.065 s	
State_Vect>State_Vect.State_Vect	54000	8.389 s	8.389 s	
Satellite>Satellite.sim_subsys	2160	214.593 s	8.305 s	
Sys>Sys.find_comp	53328	7.282 s	7.282 s	
Data_Structure>Data_Structure.getlast	2160	14.375 s	7.107 s	
Sys_Thermal>Sys_Thermal.deriv_gen	2160	57.979 s	6.727 s	
Sys_Power>Sys_Power.total_power	2160	6.963 s	5.401 s	

Figure 5-5: MATLAB Profile Results for CASTOR Integrated Model

specifically in the CASTOR design by having different time steps for different subsystem models, as well has having the switching ability to increase or decrease the amount of detail in each model. The initial CASTOR simulation was primarily a design verification tool used for short-duration studies. No entire mission studies were performed with this simulation like those done with MOTV due to this excessive run time.

5.1.4 Phase D — Complete CASTOR Integrated Model Results

The IEM design framework outlined in Section 4.1 was implemented in a new CASTOR simulation. The simulation framework was set up to support the Phase D development of CASTOR and is intended to remain through operations. This section will summarize these architectural changes, describe their results, and describe how the simulation must be managed, updated, and maintained.

Summary of Changes

Section 4.1 described the design of the changes to the CASTOR simulation. To summarize, the new simulation successfully implemented IEM SIM design principles: command input/output and intelligent auto-generation, incorporation of subsystem switching, and

telemetry input. Additionally, while not inherent in the IEM, changes also occurred in the scheduling and data management to improve the simulation performance. Lastly, STK outputs were also added for further analysis, verification, and visualization capabilities.

Command-Based Simulation Results

The command-based architecture was successfully implemented in the new simulation design. The verification test performed was to test all command input conditions and show that the simulation output the desired data. The following simulation input parameters were modified during the test:

- **Command input file** — Load file or no file
- **Autogeneration settings** — Default, never, or always (for each command type)
- **Step size** — Fixed step or event-driven

The test demonstrated that all twelve cases behaved as expected. The most important case is loading a command file and showing the that default autogeneration settings correctly turned off the command generation for the command types located in the file. The two additional autogeneration settings (never and always) are written into the simulation to allow the user to fully control the behavior of the simulation. The primary verification source is the command matrix output, which shows the time of execution, command string, command source (e.g. file or event), and a Boolean identifying if the command was executed in the simulation. Additionally, each executed command is written to a command file in a more readable format, and this represents that the commands were successfully executed. The step size settings are not important for showing the autogeneration capability, but they were both tested to ensure the command changes did not create an infinite time loop in the scheduler.

Implementation of the command-based simulation also identified an unexpected benefit of the IEM design approach: the ability to identify and furthermore develop areas of the design that are not at the necessary maturity level. The CASTOR commanding structure was not defined at the time of the latest simulation development, so a structure and format of commands was developed by the author. Since this is the first attempt at creating the

command format, it is expected that commands structure and specific strings utilized by the ground and satellite will be modified prior to launch. With this expectation, the simulation command handling was written to clearly identify where the interface between real-world command and simulation command is performed.

The maintainability of the simulation command handling is a very important characteristic in the CASTOR SIM. Whether the command strings or formats change, additional classes of commands are created, or a desired command callbacks are added, a member of the CASTOR team must update the simulation. The use of OOP methods should assist in improving the maintainability to an extent, but the command handling is still a highly coupled and complex interaction of many of the different simulation objects. Therefore it is recommended to any future personnel working with the simulation to not modify the command list, event list, or scheduler objects other than in the identified methods.

In summary, the CASTOR simulation command-input structure was greatly modified to meet the broad IEM objectives. The new simulation has a working command-based architecture that utilizes the same commands as the satellite to modify the satellite's state. Commands executed are time tagged and saved to a command file throughout the simulation, and can be used eventually to feed into an integrated avionics flatsat configuration or compare to actual satellite commands. A command input ability allows simulation or satellite command files to be read and used in the simulation. Results have shown that the simulation will intelligently turn on and off the automatic generation of different command types based on their existence in the input file.

Subsystem Switching Results

The subsystem switching method was described as an issue in both the MOTV and initial CASTOR simulations. The newest simulation deals with this issue by having different modes in each subsystem model. These modes are integers greater than -1, with each integer representing a higher level of detail. The -1 mode is reserved for telemetry input, where the model is passing the telemetry through. The zero (0) mode represents the "off" conditions of the subsystem, where only the necessary pass-through data is created. The other mode settings are up to the designers of the subsystem models. However, all modes

must save data to the `data` structure to be used by other subsystems, so therefore no subsystems are really ever off.

The orbits model can be used to show how the subsystem switching should be analyzed to ensure it is worth the programming effort. The new minimized feedback N^2 diagram shown in Figure 4-3 called for two different orbital models. The first was intended to be a simple, quick method to determine the position, velocity, and orbital conditions, and the second was to be a more accurate model of the orbital environment and include the effects of the thruster and attitude subsystems. This is somewhat related to the subsystem switching, as there are two different ways to create the same output. After running the simulation, it was shown that the initial orbital model is actually more computationally demanding than the more detailed model (shown in Table 5.3). This result is an element of the programming and number of additional functions called for checking thruster logic, and is an area of the model that needs improvement. The lesson that comes from this result is to ensure that the additional work of adding different subsystem modes is worth the investment in terms of performance or accuracy.

The implementation of all subsystem models with multiple modes has not been performed at this time, and this is one of the two largest areas for further work. A simplified ADCS model has been implemented in the new model, as the complex model has not been completely converted. Programmers of the future models are encouraged to program different methods in their respective subsystem model objects at the beginning. This will produce code that is easy to maintain and enhance.

Telemetry Input and Data Management Results

The ability to load in and utilize telemetry was successfully shown through the ADCS model. The initial telemetry used was the time-stamped attitude values for CASTOR created from a previous simulation run. This was input into the `inits` input structure, and the ADCS mode was changed to telemetry input and pass-through mode (mode -1). The telemetry input was saved into the `ADCS.tlm` property upon instantiation of the ADCS object and this property was used to fill the `data.adcs` structure during each call of the `sim.run()` loop.

As expected, the simulation performance increased, shown by a lower total run-time. The ADCS step size was set to 5 seconds for a 12 hour simulated duration. Table 5.2 shows the timing results of the ADCS subsystem methods in comparison to the simulation as a whole for the two cases. While the timing is a function of many other simulation and subsystem input conditions, it is clear that the loading of telemetry improved the performance and run-time of the simulation.

Table 5.2: CASTOR ADCS Model Timing Results for Telemetry Input

	ADCS Constructor	**ADCS Model**	**ADCS Object**	**Total Simulation**	**% ADCS**
No TLM	0.52 s	2.42 s	2.94 s	14.3 s	20.6%
TLM-fed	0.53 s	0.13 s	0.66 s	12.0 s	0.66%

The telemetry input conditions assumed in this analysis were that the telemetry will be available for the entire simulation duration and it is a one to one match with the simulation data format. Neither of these assumptions are valid for actual implementation of the simulation with real telemetry, so additional work must be done on each subsystem model to ensure that the telemetry input will be formatted into the correct simulation output. It is also possible to keep these assumptions, and perform the analysis and conversions necessary in the TDA. The operational implementation of comparing telemetry to simulation data is a major area of discussion, and is dealt with in the FS-3 solar panel comparison description in Section 5.2.1.

The modifications made to the data structuring of the CASTOR simulation also greatly improved the performance. The previous CASTOR profile results shown in Figure 5-5 highlighted how the data passing structure was the most significant factor in the simulation run time. The improvements came from having a more efficient model-to-model input-output structure, shown through a reduced N^2 diagram, as well as using temporary data storage instead of global data storage. The improvements have come at the cost of losing a centrally located state matrix that could be called by any model at any time; this loss was accepted once a clear N^2 structure was set and each model uses only what it requires from the other models.

Scheduling Results

The scheduling task was initially added to facilitate the command-driven simulation framework. It was identified that the command-driven simulation could be done in two ways: fixed time step and event-driven, where events included any state change in the natural environment (e.g. orbit) or command. The CASTOR scheduling object was written to allow for both types of cases and to ensure the new framework was compatible with the previous models. The results clearly show that the event-driven simulation is advantageous over the fixed time step in nearly all areas.

The results given in this chapter are for a simplified satellite model. The simulation architecture is complete and fully functional; however, for ease of analysis, the satellite is only modeled with an orbits, ADCS, and propulsion model. Table 5.3 breaks down the different step sizes used for each model, the individual models' respective run-times, and the total run-time for the satellite. Two orbit times are displayed as the orbits model is called twice, as previously discussed.

Table 5.3: CASTOR Simulation Timing Based on Step Size

Run	δ_{orbit}	δ_{ADCS}	δ_{prop}	T_{orbit}	T_{ADCS}	T_{prop}	**Total**	**Overhead**
1	50	5	60	3.3 + 2.0	2.4	0.09	9.7	1.91
2	25	5	60	5.9 + 3.5	2.5	0.08	13.9	1.92
3	5	5	5	27.3+16.9	2.5	0.09	48.7	1.91
4	$\Delta T = 60$			3.2 + 2.5	0.72	0.39	13.0	6.19
5	$\Delta T = 30$			5.5 + 4.7	1.33	0.65	24.1	11.92
6	$\Delta T = 5$			53.6+42.9	18.25	7.58	175.8	53.47

Note: 12 hour simulation run. All times are in seconds

These results provide insight into what areas of the code are degrading the performance of the simulation. While it is clear that the orbital model is taking a significant amount of time relative to the other models, the purpose of this comparison is not to critique the efficiency of the models, but show how the step sizes affect the entire simulation's output. The last column shows that for the event-based cases (1-3), the simulation overhead is independent of the individual subsystem sample times. This is inherent in the event-based design, as the number of calls to each subsystem model is the same for all cases. However, in the constant time step case, the simulation overhead (total time minus model time)

increases as the time step decreases. This is intuitive, as the number of times each function is called has greatly increased. Additionally, the magnitude of the overhead time for fixed step is significantly higher than the event-based case.

The clear conclusions from these results show that the event-driven CASTOR simulation is preferable over the constant time step method. The addition of the scheduling task, event list, and command list added a large deal of overhead for the fixed time step method, so it is necessary to constrain this conclusion to event and command-driven simulations. Additionally, if several additional models are added that frequently interrupted the propagation to the next time step, the run-time in the event-based case would increase whereas the run-time in the fixed time step case would remain the same. Despite this, the discrete scheduler will still likely be faster in any case where accurate and detailed data is required due to the individual time steps for each model. Further work could be done to intelligently save data that is usually lost during event-interrupts to minimize the effects of the interrupts.

A less significant trade study was also performed comparing the performance of the bisector method, illustrated in Figure 4-6, to a stepping method. This trade was used to identify the best way to find the time of an interrupt in the event-based case. The test was performed on the sun/eclipse boundary event interrupt in the orbits model. Table 5.4 shows the results of the two cases. The given input conditions were a 6 hour simulation duration with a 25 second orbit initial time step and a one second step or bisector convergence threshold.

Table 5.4: Interrupt Timing Identification Method Results

Method	**Mean** (ms)	$\boldsymbol{\sigma}$ (ms)
Stepping	14.0	2.0
Bisector	3.5	0.1

These results are not surprising. Nearly the only difference in run time is the number iterations in the given search loop. The number of iterations needed in the bisector method can be predicted from the subsystem time step and bisector convergence using the following:

$$n = \text{ceil}\left(\frac{\log \frac{\delta t_{sub}}{\tau}}{\log 2}\right) \tag{5.1}$$

The number of steps needed in the stepping method is not known, and can be treated like a discrete uniform distribution between 0 and δt_{sub}. The calculation of the expected value and variance of the discrete uniform distribution is simply the midpoint of the time interval, as shown in equation 5.2.

$$E(n) = \frac{\delta t_{sub}}{2\tau} \tag{5.2}$$

These results led to the implementation of the bisector method in the simulation over the stepping method. This method is performed inside of the individual subsystem models. This is advantageous to make the simulation efficient and minimize the pass-through of data and objects. However, this comes with a cost at the readability and maintainability in the code. The bisector interrupt identification method would be better maintained and organized in the scheduler object. It was decided to keep this interrupt identification in the subsystem model and ensure that any future programmers know how to deal with the interrupts. This is not a major source of additional work, as the event-based nature of the simulation forces subsystem model programmers to deal with interrupts.

The addition of a scheduler object and its event-based methods in the CASTOR simulation has improved the simulation in terms of both performance and structure. The scheduler successfully controls the simulation time and ensures that every command and event is processed in all simulation cases. In regards to the IEM design principles, the scheduler provides the simulation with an efficient method for handling the various commands and events in the simulation. The scheduler design in its current form could be implemented successfully through Phase E and satellite operations.

STK Output Results

The simulation added the ability to generate STK ephemeris and attitude files from the satellite's orbit and ADCS telemetry. This addition allows for the utilization of STK's visualization and analysis tools. The visualization is helpful for identifying any abnormalities in the ephemeris or attitude data, but is mainly used as a presentation, training, and education tool. The analysis tools available in STK such as coverage statistics, reports, and

graphs allow users to either verify certain model results with a correspond STK model or to use the additional tools to generate further results.

A MATLAB to STK interface exists in the simulation to automatically launch a scenario with newly generated ephemeris and attitude data. This feature allows MATLAB to call STK methods and control any feature of STK. While this feature could be used to run the STK engine throughout the simulation, this is not an active interest in the CASTOR model. Future work could be done to compare STK orbital propagation methods to those utilized in MATLAB for accuracy and performance.

Summary and Remaining Work

This section has demonstrated two major points: the new simulation successfully meets the IEM principles and the new architecture greatly improves the performance in the simulation run-time. There are a few significant areas of work left in the CASTOR simulation, as shown below.

- **Subsystem Model Completion** — The subsystem models have been unchanged from the previous CASTOR simulation, and only outlines exist for most models in the new simulation. The orbits, ADCS, and propulsion models do exist, but work remains on each of these models.

- **Simulation Accuracy** — The discussion has been primarily on performance, as that was a major deliverable for the new model; however, the accuracy of the simulation must also be shown. The new simulation structure, specifically different subsystem step sizes, should assist in meeting accuracy requirements.

- **Command and Event Implementation** — The commands and events features have successfully been demonstrated in the simulation, but work remains to match the real CASTOR system with the simulation. This requires a large initial effort as well as periodic maintenance as the commands and event characteristics change in the CASTOR hardware or software designs.

Lastly, the full implementation of the TDA and MAT elements of the IEM has not been completed in this thesis. This is an area of work that has been baselined but not performed. Emphasis was put on demonstrating an actual flight TDA design in FS-3 to gather lessons for future tools. These results are shown in the next section.

5.2 FalconSAT-3 TDA Results

The FalconSAT-3 TDA, or the FS-3 LimWOD GUI, was described extensively in Section 4.2.3. Two examples are given in this section to illustrate how the GUI's tools were used. Each example discusses how the results apply to the IEM process, particularly the TDA design.

5.2.1 FS-3 BSEKF Verification Using LimWOD GUI

The BSEKF attitude filter written by MIT PhD candidate Jacob Katz processed magnetometer data to produced attitude positions and rates for FS-3. Described previously on page 117, the filter output was viewed in a STK animation as well as with solar panel currents in the LimWOD GUI. While neither of these methods gave quantitative results, they were an independent source of analysis. The solar panel current comparison in the LimWOD GUI was thought to be a better source of independent verification, as it utilized a different source of telemetry that was not used in the filter.

The solar panel currents are not a reliable source of accurate attitude information in periods of low power draw. The Battery Charge Regulator (BCR) on FS-3 will regulate the amount of current drawn from each solar panel based on the power required by the batteries and the bus. The nominal power draw from the satellite is much lower than the available solar panel power, and therefore the current from the solar panel providing power is much lower than its maximum. In these periods of time, the only comparative analysis that can be performed between the solar panel currents and the BSEKF attitude output is to see if the solar panel that should be in the sun is the one providing current. The switching from one panel to the next as the sun moves around the body-fixed coordinate system (due to spinning and orbital motion) is not entirely predictable in the BCR's logic, and therefore the accuracy of the angular measurement of the body axes to the sun is low. If the power required by the satellite is much higher, the solar panels will give a much better resolution on the incidence angle between the satellite's body and the sun. This is generally not desirable, as it is pushing the limits of the power system of the satellite and the accuracy is best when there is zero or negative margin in the power budget. These cases

of high power draw are most prevalent during communication, as the transmitter requires a large amount of power.

The LimWOD GUI was used to visualize BSEKF output in both power cases. The results shown here are focused on the high power case, as it provides more insight to the attitude output. Figure 5-6 shows the solar panel comparison results for a case in mid-January 2010.

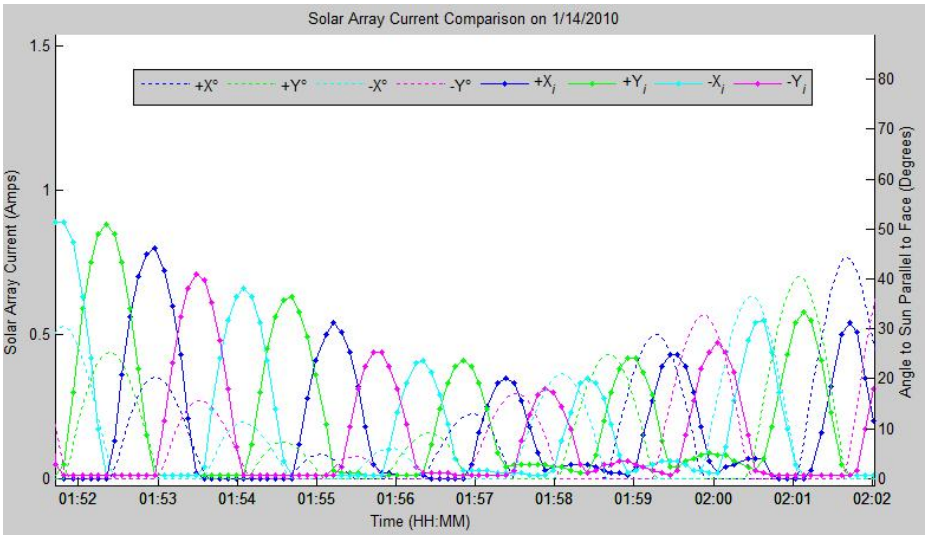


Figure 5-6: FS-3 LimWOD GUI Successful Solar Comparison

Figure 5-6 shows that the solar array angle estimate represents the actual readings very well. The results are not ideal, as the primary (left) y-axis is in amps and the secondary (right) y-axis is in degrees, but the purpose of the graph is to show the general trend in switching between the different panels. The graph shows that the switching between actual solar panel currents (solid lines with markers) matches the switching between the expected solar panel angles. The conclusion from this is that the BSEKF output seems to give a reasonable attitude estimate despite using magnetometer input only. The graph also shows that as the expected panel angle reached a minimum, or the satellite's rotation axis was nearly parallel to the relative sun vector, the measured solar panel current reached a minimum. This is harder to see from this figure, but the maximum solar current for each panel is over 1 amp. The comparison at this minimum is not as good, shown by the half-sinusoids for each axes becoming out of phase; however, the graph shows the curves

converge upon reaching higher angles to the sun.

The BSEKF output did not always match the solar panel currents, as shown in Figure 5-7. This case is shown in a later date during a significant power draw as well.

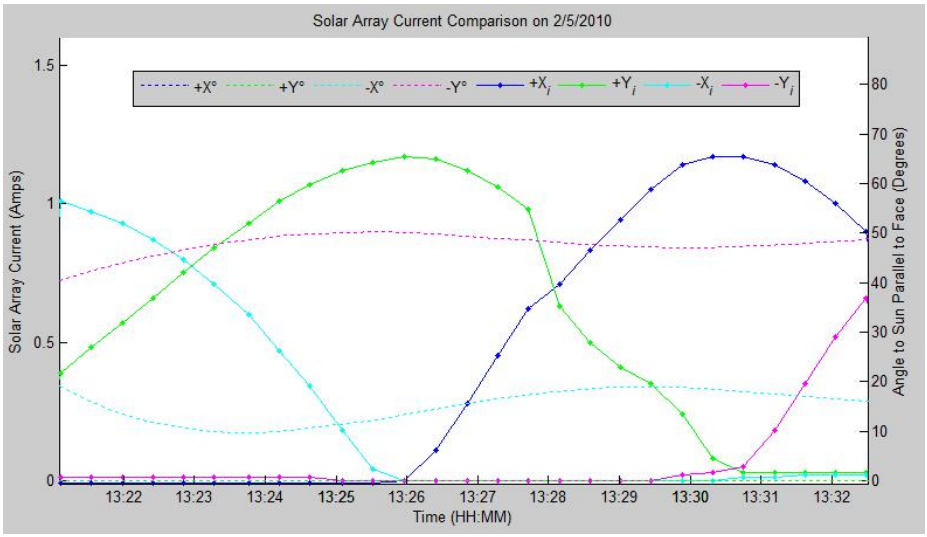


Figure 5-7: FS-3 LimWOD GUI Unsuccessful Solar Comparison

Figure 5-7 shows that the BSEKF output behavior does not match the actual solar currents. Specifically this is shown through the rate and the axis that should be receiving sunlight. The -Y and -X panels are expected to be in the sun for the entire 10 minute window, but the actual solar panel data shows that the +Y and +X panels were in the sun in this period. While it is much harder to qualitatively verify that the filter is providing an accurate knowledge estimate of the attitude, it is much easier to state that it is not correct by only looking at Figure 5-7.

The addition of the solar panel comparison in the LimWOD GUI allowed for quick visual verification of the filter's accuracy. While it did not quantitatively verify the results due to a large uncertainty caused by the BCR's management of solar panels, it allowed for easy identification of infeasible data. This led to the conclusion that the BSEKF did not provide accurate attitude solutions for all cases and needs improvement prior to full implementation.

This example is relevant to the IEM discussion as it gives an example of a comparison between real data and expected data in the TDA. The concept of comparing actual and

expected is simple, but each type of comparison may have different intricacies. These intricacies are a major resource burden and should be considered prior to setting lofty goals. Specific to the BSEKF verification example, it is possible to look only at the time periods of negative power margin, which would therefore give a reasonably accurate incidence angle for each solar panel. This angle could then directly be compared to the BSEKF output and a quantitative measure of error could be performed. While this is a much better result, the majority, if not all, of the solar panel data occurs during positive power margins. It was decided that resources were better spent on creating a quick verification of solar data showing that the BSEKF was plausible or not.

5.2.2 FS-3 April 2010 Event Analysis Using LimWOD GUI

A series of unexpected results were observed in early April 2010. This event tested utilized all of the FS-3 analysis tools and techniques available, as well as provided motivation to create new tools. This section will describe first the timeline of events and then describe how a range of analysis tools were used to determine what occurred. Without these tools, it is likely that this event would not have been identified, which would have jeopardized the FS-3 mission and the vehicle's safety.

The underlying result of this event was a major spin-up of the FS-3 satellite about its yaw (minor) axis. While different hypotheses are currently being investigated for the exact cause of this spin-up, the most likely is a significant (multiple orders of magnitude) increase in the amount of torque imparted by the Low-Thrust Attitude System (LTAS). This system consists of four clusters of three-axis thrusters, which are expected to generate $\approx$100 μN of thrust in a series of short, $\approx$15 μs pulses at a rate of 2 Hz. On the 5th and 8th of April, a long-duration test was performed on two LTAS thrusters firing simultaneously. It is hypothesized that rather than an equivalent 30 μs per second of equivalent thrusting time (duty cycle of 0.003%), the LTAS thrusters both went into a mode that caused the duty cycle to be increased by several orders of magnitude. The expected change in angular velocity for the first test was less than $1 * 10^{-5}$ rad/sec, or 0.0005 deg/sec, whereas the observed change in velocity was greater than 0.5 rad/sec, at least four orders of magnitude greater. This great difference is the cause for technical skepticism that the LTAS actually

caused this torque, but as the data will show, this is the most likely out of all options.

With an understanding that the events in early April sped the satellite up significantly, the following timeline of events is presented in Table 5.5. Events in italics represent data being available for analysis, and events in bold identify unexpected or unplanned events.

Table 5.5: FS-3 April 2010 Summary

#	**Date & Time(Z)**	**Event**
1	4/5 0605	Began operations of two LTAS thrusters for 5 hr test
2	4/5 0605+	**Significant angular acceleration about yaw axis**
3	4/5 0924	**SW shuts off LTAS due to battery voltage limit reached**
4	4/5 1104	Command sent and acknowledged to turn off LTAS thrusters
5	4/6 0600	B-dot algorithm tests with active torque rods (off at 0920Z)
6	4/6 1120	*Real time log file (CRA file) saved and placed in SVN*
7	4/8 0127	MIT email that CRA files show unexpected change in torque rod polarities
8	4/8 0913	Began operations of two LTAS thrusters for 20 hr test
9	4/8 0913+	**Significant angular acceleration about yaw axis**
10	4/8 1520	**SW shuts off LTAS due to battery voltage limit reached**
11	4/12 1613	*Initial WOD files (6Apr) and attitude files placed in SVN*
12	4/12 1722	MIT email recommending hold on further tests due to high rotation rates
13	4/14 1702	*5 Apr WOD files placed in SVN*
14	4/14 1906	MIT email identifying off-nominal magnetometer and power readings
15	4/14 2214	Addition of new MATLAB analysis tools specific to 5 Apr
16	4/15 0336	Test to verify both thrusters of interest individually produce same result
17	4/15 0516	Test to show other thrusters in clusters function nominally
18	4/15 2329	USAFA email to LTAS provider to provide situation report
19	4/16 0213	*Small amount of 8 Apr WOD files placed in SVN*
20	4/16 2131	USAFA ID of 1 March 2010 short-duration LTAS test with similar results
21	4/17 0238	MIT email describing uncertainty and confusion due to new data
22	4/20 0242	*Remaining 8 Apr WOD files placed in SVN*
23	4/20 2137	MIT email summary identifying aliasing issue and 4/8 1520Z event
24	4/22 2325	USAFA email summary identifying 4/5 0924Z event

This table shows the wide variety of response time to the different events. The amount of data collected between the 5th and the 8th required multiple days to downlink, and ground station issues at USAFA such as high winds prevented this data from coming down sooner. The table shows how very soon after data was posted to the SVN (shared and controlled subversion repository between MIT SSL and USAFA SSRC), it was analyzed and a resulting assessment was made. While these assessments did not describe the whole picture, slowly different elements were pieced together to determine what is believed to be

the best explanation for each of the events. The next section is devoted to showing the data for the different events that led to these conclusions.

FS-3 Data from April Spin-up Event

This section will describe in order the data used to analyze the events of April 5-8. This order is based on the data analyzed and assessments made, and the dates of the data presented are not necessarily chronological due to the availability of different data sources.

The first data source analyzed was the CRA file from 6 April (#6 in Table 5.5). This text file contains the commands acknowledged by the satellite, including the individual torque rod commands. These commands identify the torque rod axis fired, as well as the direction of the current in the torque rod (also described as polarity). By looking at how frequently the current direction is changing for a given torque rod, one can gain a general idea of the rotation rate calculated by the B-dot algorithm.

After the corresponding WOD and attitude data (Table 5.5 #11) arrived, it was clear that it was not that the controller misbehaved, but rather that the satellite was spinning faster than expected. This was shown both through magnetometer and solar panel current data. This was shown very quickly through the LimWOD GUI and also through the attitude file script written specifically for this test, with results shown in Figure 5-8 (can be compared to 4-23).

The data from the LTAS tests that arrived (Table 5.5 #13) was not in the usual LimWOD form, but the full WOD form. No analysis tool had been written for this data type, as very little full WOD data had been collected since the implementation of full WOD. Therefore two MATLAB functions were quickly written: one to read the full WOD CSV files and fix the timing, and the other to format the full WOD data into the usual LimWOD format so that the data could be analyzed by the GUI. This allowed all previous features on the data to be available. The LimWOD GUI quickly produced the magnetometer, solar current, and battery graphs that can be seen in Figures 5-9 to 5-11.

Figure 5-9 shows a large disturbance in the X-axis magnetometer at the start of the test. Additionally, the Y-axis magnetometer frequency increases at the this time. The Z-axis magnetometer rate is not significantly affected, but a high frequency noise is seen in

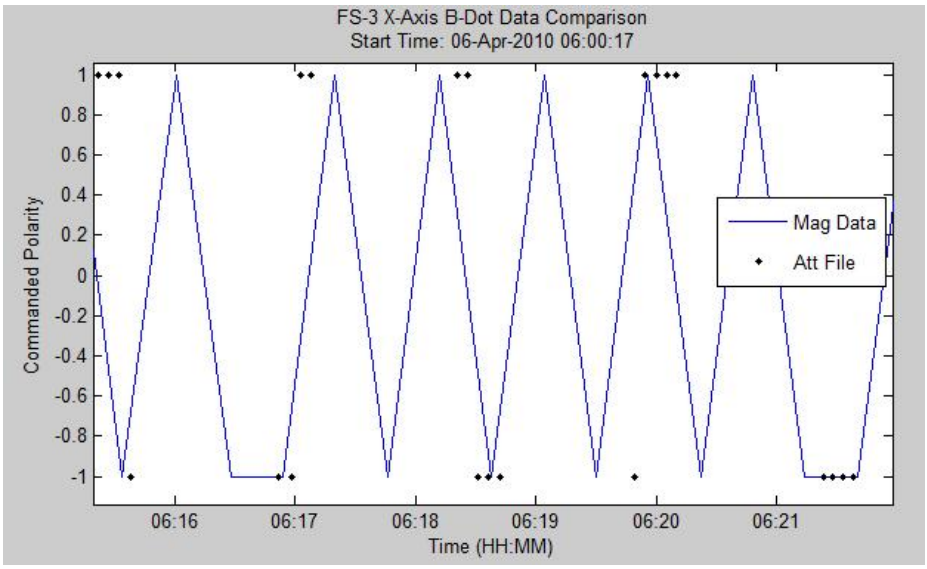


Figure 5-8: B-Dot Analysis Script Results for 6 Apr Test

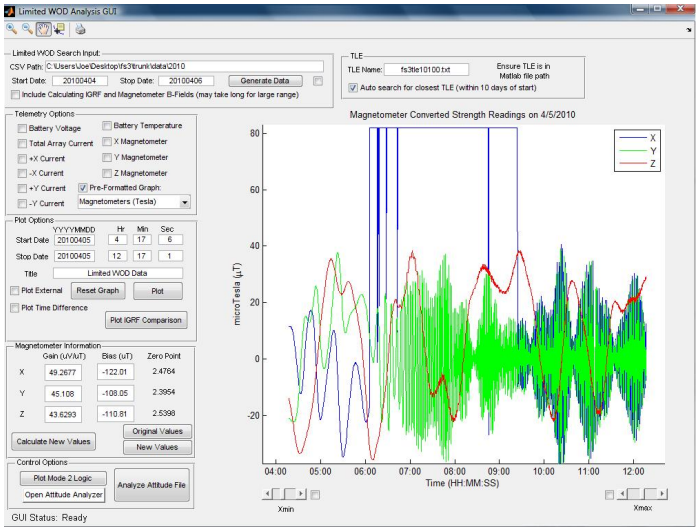


Figure 5-9: LimWOD GUI Magnetometer Output for 5 Apr Data

the signal. The increase in the rotation rate seen in the Y-axis magnetometer is also seen in the solar array currents, shown in the Figure 5-10.

The solar array current in Figure 5-10 also show a significant increase in the amount of current pulled from the solar arrays. It appears that the satellite is drawing more power than it is producing, shown by the total current (black line) being significantly higher than any individual channel (BCR pulling from both panels to produce power). This is also seen in Figure 5-11, which shows the battery voltage and temperature.

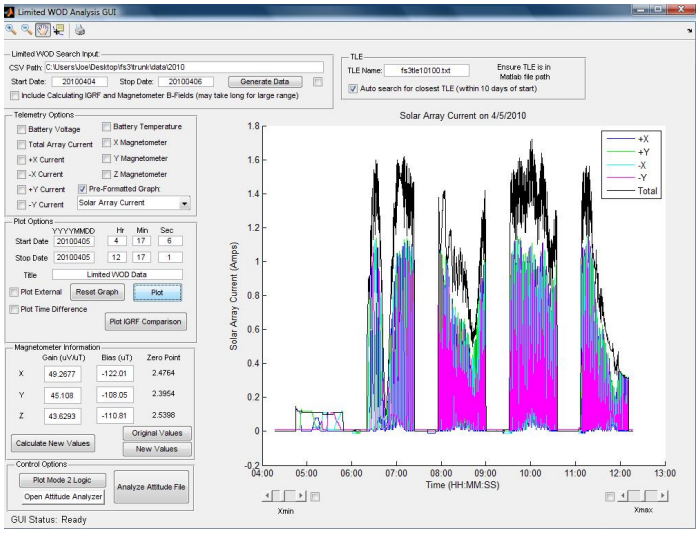


Figure 5-10: LimWOD GUI Solar Current Output for 5 April Data

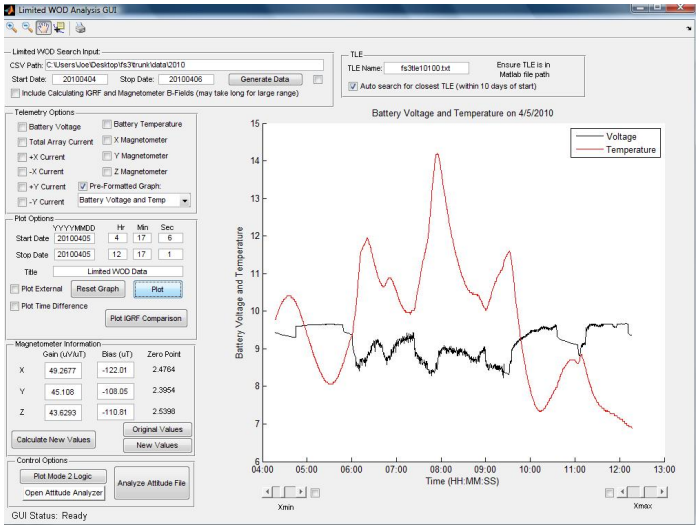


Figure 5-11: LimWOD GUI Battery Output for 5 April Data

The battery data shown in Figure 5-11 shows how the batteries are not able to be fully charged during the LTAS test. The voltage will usually only go down in eclipse and then nearly instantly increase upon entering the sun (as shown in the left of the figure). The graph shows that the batteries are still supplying power while in the sun, showing that the power margin is negative overall.

This quick analysis led to the MIT email identified by #14 in Table 5.5. Also, as additional channels of data were available in the full WOD file, an analysis script was quickly written to view different telemetry channels, such as the LTAS power lines (#15).

These results showed that as the command to LTAS was sent, the power going through the LTAS line increased quickly, therefore summarizing that power was being drawn by the LTAS units. Additionally, the capacitance readings from LTAS were shown to cycle upon sending the commands, representing the charging and discharging of the capacitors.

The resulting data that came in for 5 April and 8 April showed major changes in the satellite magnetometer and solar array data, leading to the conclusion of a major event changing the rotation of FS-3. Figure 5-12 shows how the magnetometer data changed before and after the 5 Apr LTAS test.

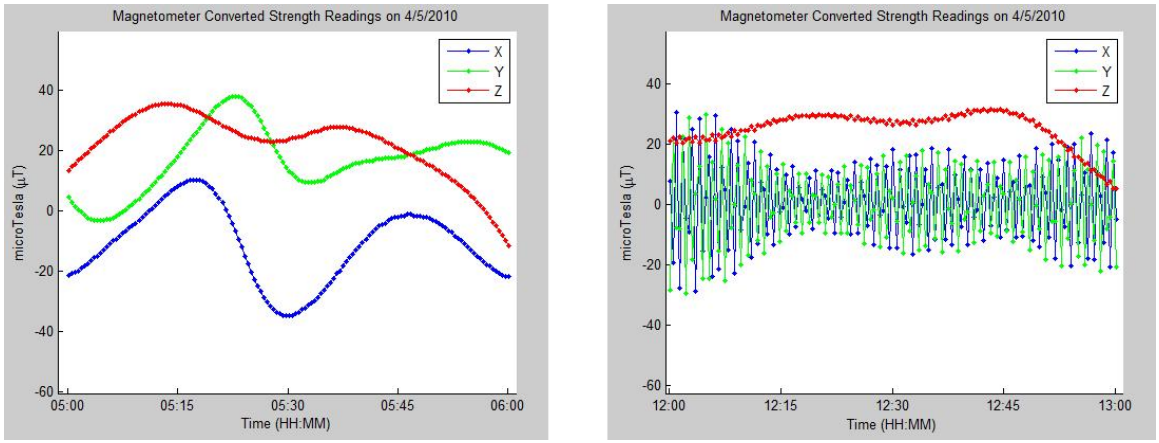


Figure 5-12: Magnetometer Difference Before and After 5 Apr LTAS Test

The latter observed magnetometer readings (right plot) were described to be aliasing. This topic and additional background information is given below as it is relevant to the discussion of the spin-up event.

Aliasing and Under-Sampled Sensors

When dealing with rotational dynamics, aliasing can be described as a function of rotational period and sampling rate. In general, the Nyquist frequency is the minimum frequency that a signal (i.e. sensor) can be sampled at to allow reconstruction of the dynamics [44]. This frequency is defined as $f_{ny} \equiv \frac{2}{T}$, where T is the period (or bandwidth). Thus the sampling rate f_s must be greater than f_{ny} to prevent aliasing. This is the theoretical limit for a known waveform, and in the case of a three dimensional rotation through a changing reference frame, this becomes much more difficult to model. FS-3 has this problem with

both magnetometer readings when compared to a rotating IGRF (relative to the orbit), along with solar panel currents compared to a rotating sun (again, relative to the orbit).

In short, aliasing was a problem when data was sampled less than twice in a given rotation. This problem can be solved easily by increasing the frequency of sampling, but in certain cases this leads to too much data for continuous collection or is not possible for the hardware available. Yet, this seemed to be a straightforward occurrence that was understood that if the data from the magnetometers or solar panels showed choppy data, the data was aliasing and the sampling frequency needed to increase. Figure 5-13 shows an example of the data starting to alias on 5 April due to angular acceleration in the positive yaw ($+\hat{Z}$) direction (Table 5.5 #2).

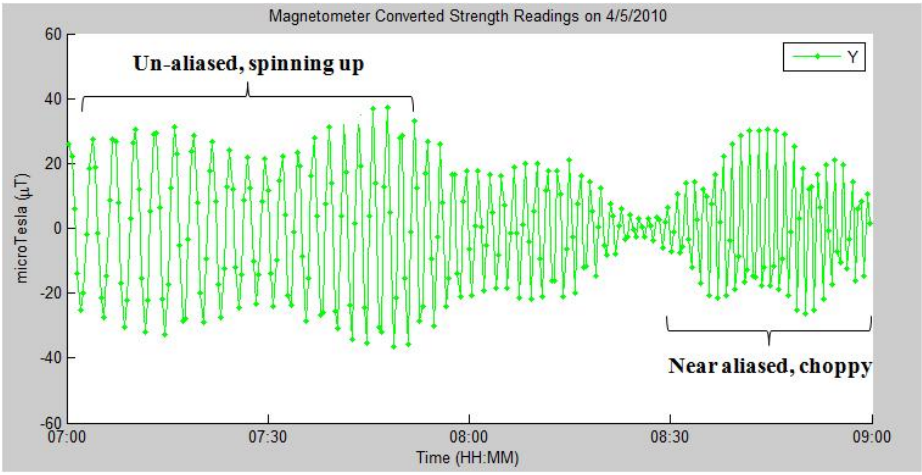


Figure 5-13: Initial Aliasing Due to Angular Acceleration

The problem that perplexed analysts for well over a year is the appearance of slower rotation that can be seen during periods of aliasing. This problem occurs when the sampling frequency is near the period of rotation. Numerous times analysts believed that the satellite rotation rate slowed down based on both magnetometer and solar current readings, when in fact the satellite was rotating significantly faster. An event on 8 April 2010 led to the discovery that the satellite in fact was not slowing down, but in this case, increasing its rotation rate. Figure 5-14 shows this phenomenon, which initially was read as an opposite angular acceleration about the yaw axis (Table 5.5 #21).

The major issue with the description of an opposite body-fixed acceleration is that the

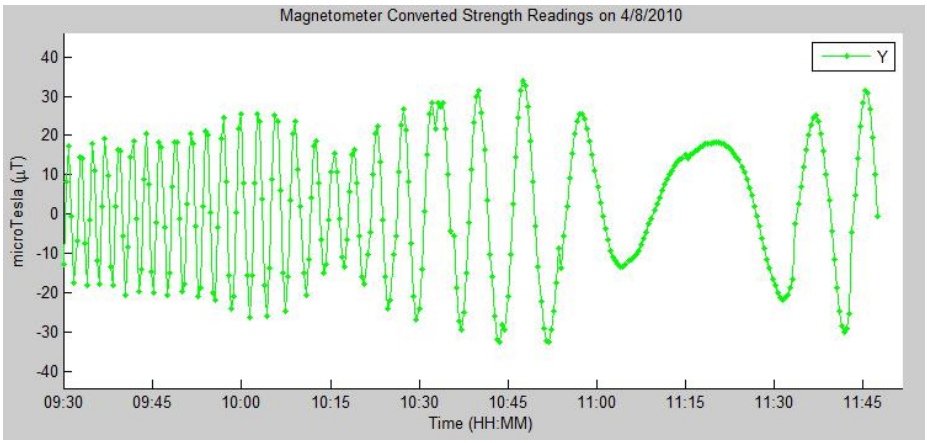


Figure 5-14: Apparent Slow Down Due to Aliasing

same sequence of commands that caused an acceleration shown in Figure 5-13 was performed in this case as well. Therefore, it seemed extremely unlikely that the satellite was actually slowing down. Upon further analysis, two additional sources of telemetry provided data supporting a theory of an illusion of slowing due to aliasing and the reality of an increase in the rate of rotation. The first source is a combination of the time difference between samples and unexplained jumps in the magnetometer reading. As displayed previously in Figure 4-21, the sampling time of the satellite is not necessarily constant. When looking at the Y-axis magnetometer channel in both cases in Figure 5-15, it can be seen that when the time step changes, the magnetometer reading jumps and no longer looks as smooth or periodic.

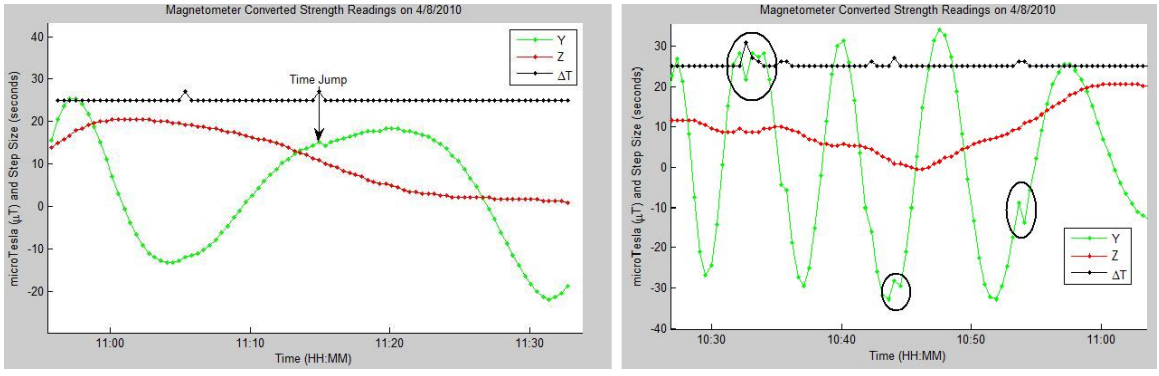


Figure 5-15: Aliasing Indication Due to Time Jump

Originally, these magnetometer jumps were thought to have occurred due to the same unknown reason that causes occasional changes in sample frequency, but this hypothesis

had no basis of fact. The hypothesis that these jumps occurred due to an unrealized state of aliasing fits all data much better. The aliasing phenomena when the sampling rate is nearly equal to the rotation rate (or an integer multiple thereof) can be visualized as a strobe light on a periodic mechanism, such as a pendulum. The strobe light represents the sample rate, and the pendulum represents the rotational period of the satellite. Imagining a fixed inertial viewpoint above the pendulum and a strobe light facing towards the ground, one could plot the strobes as a function of the sample frequency multiplied by the period. This variable, f_sT is equivalently the average number of samples per revolution. Figure 5-16 displays a range of f_sT for a case where the period is one second and for 20 revolutions.

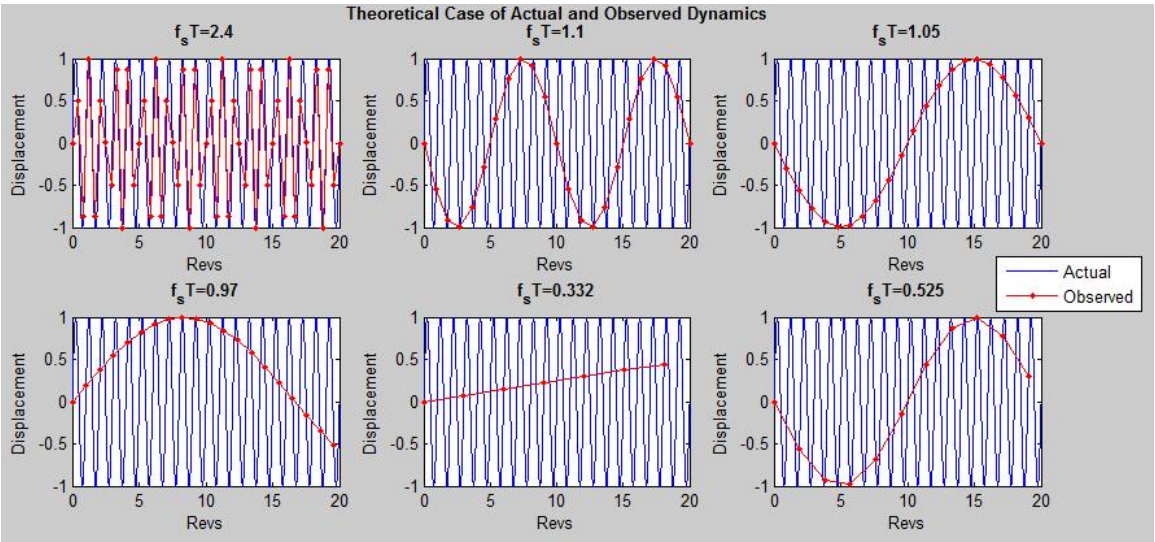


Figure 5-16: Aliasing Phenomena for Under-Sampled Data

Figure 5-16 demonstrates that near an integer of $(f_sT)^{-1}$, or $\frac{1}{f_sT}$, the data will appear to be sinusoidal, although at a frequency much less than the actual frequency. The first case shown of $f_sT = 2.4$ is actually not aliased, but shows how choppy data can be seen as the f_sT initially approaches the Nyquist limit of 2. The next two cases shown on the top ($f_sT = 1.1$ and $f_sT = 1.05$) demonstrate how the observed undersampled signal creates a signal that can be represented as a sinusoid. These images show that by approaching the first integer of $\frac{1}{f_sT}$, the frequency of the apparent sinusoid decreases. This is also shown in the case where $f_sT = 0.97$, only the apparent signal now moves the opposite direction than those when $f_sT > 1$. The last two examples demonstrate that this low-frequency

apparent behavior also occurs in integer multiples of $\frac{1}{f_s T}$. The $f_s T = 0.525$ case show that it is possible for an observed signal to be represented by multiples of the true rotational frequency.

If the sample size was fixed on FS-3, it would be impossible to tell that the data was aliased for a fixed rotational period; however, by having a sample size that is not completely fixed, the jumps in the data show that the low frequency observed dynamics are actually undersampled representations of a much higher frequency. This explained the apparent slow-down as the changes in the rotational period, T, led to the transition of the $\frac{1}{f_s T}$ fraction to pass an integer value.

The validation source for aliased data came from the solar arrays. The solar array currents are sampled at the same rate as the magnetometers and therefore display the same type of results showing a slow down and spin up as the satellite accelerates. However, the solar panel temperature sensors, while sampled at the same rate, have much different measuring characteristics as the temperature dynamics have large time responses. Figure 5-17 shows a comparison between the solar currents and temperatures at the beginning of the first test, where the rotation period was assumed to be relatively slow. These results were created using the new WOD processing and analysis script (#15).

Looking between the currents and the temperatures, one can see that the panel temperatures increase and decrease as a function of current draw from the panel. The rotational direction and rates in the solar panels currents are seen to be approximately the same as the those of the temperatures. It should be noted that this is one of the few cases where the power margin is negative, so the solar panels are producing their maximum available power. The decrease seen around 0645 is due to the angle of rotation being parallel with the sun.

These results can be compared to the point of the slowest observed rotation on 8 April, shown in Figure 5-18 to prove the effects of aliasing.

Figure 5-18 shows that the solar array currents appear to be at a much lower rate than shown in Figure 5-17. The currents appear to show that the rotation switches from the +Z direction to the -Z direction, as the solar panels progress from +X to +Y to -X, and then turns back to +Y. Jumps in the solar data are also seen, and correspond to the timing jumps

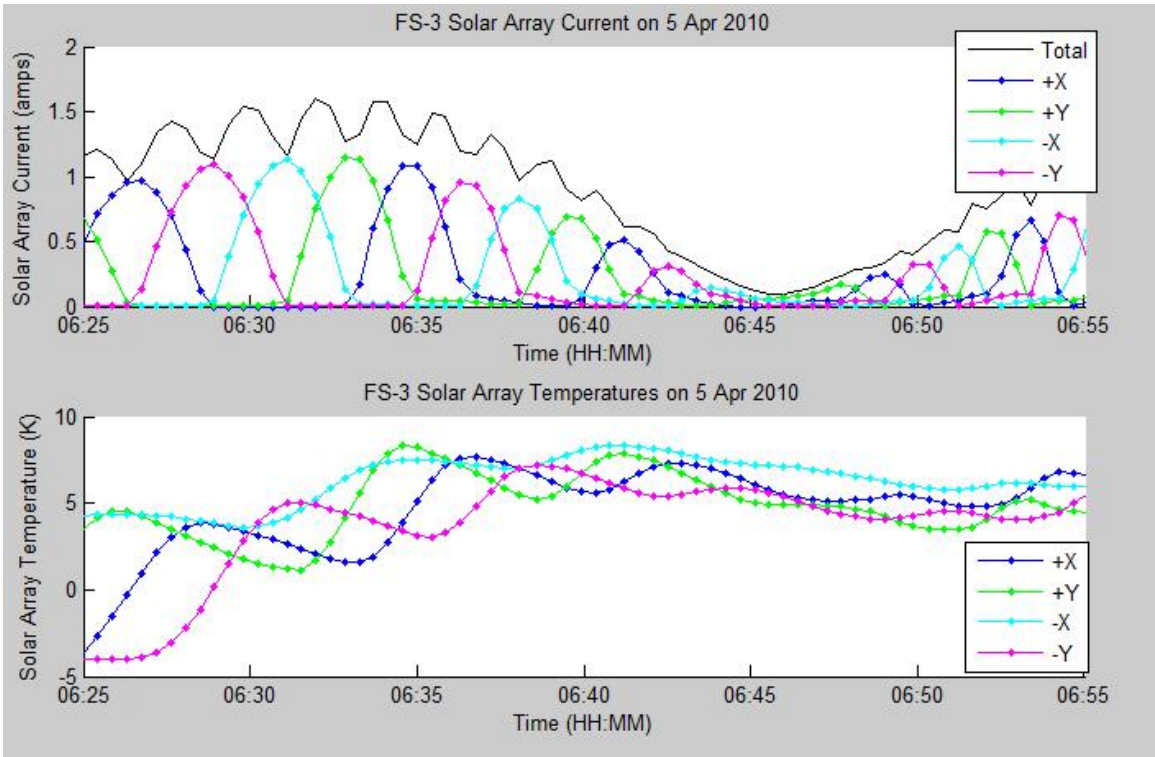


Figure 5-17: Observed FS-3 Solar Array Data During Slow Rotation

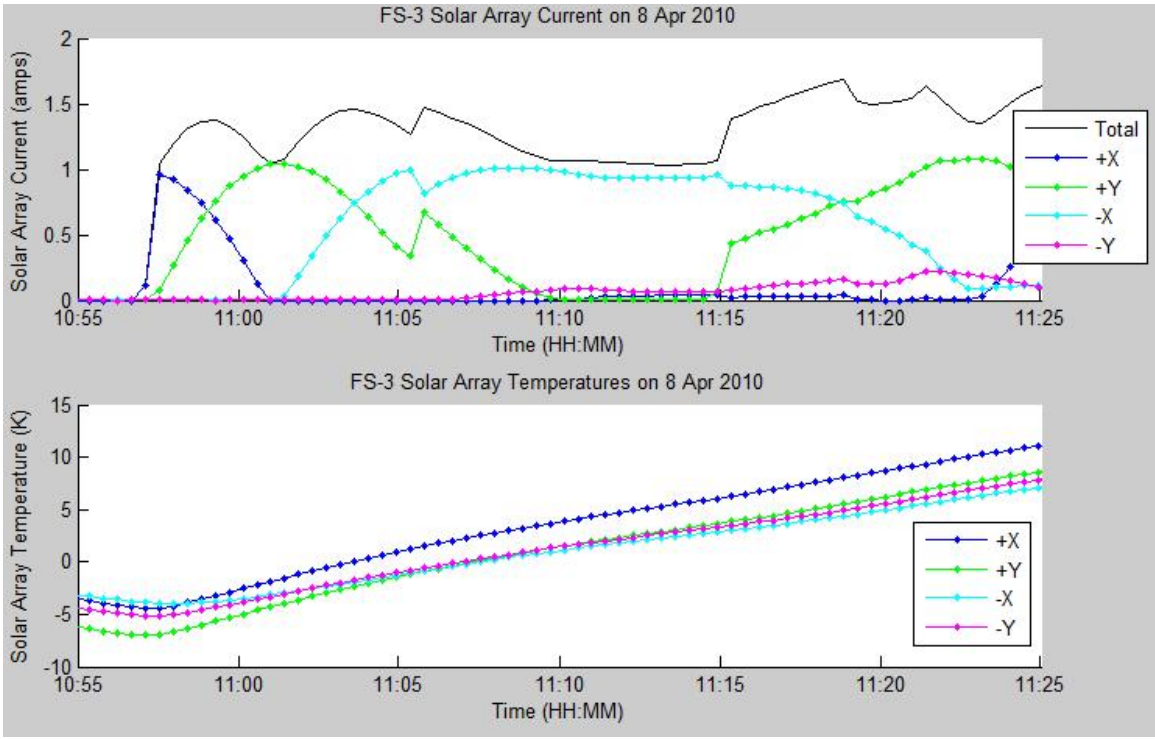


Figure 5-18: Observed FS-3 Solar Array Data at Apparent Slow-Down

just like the magnetometer. The true validation source in these graphs is the temperature data, which shows that although the current shows a slow rotation, the temperatures are all increasing at an approximately equal rate. This represents a rotisserie mode characteristic of a spinning satellite.

The reason for discussing this apparent slow down phenomena due to aliasing is to understand that once a signal is under the Nyquist sampling rate, it is usually thought to be indistinguishable. For FS-3, this infers that unless the sample rate can be increased further, the true rotation will not be able to be observed. The minimum resolution sample size in FS-3 is 5 seconds, which would represent a Nyquist limit of a 10 second rotation; however, all data during the tests was collected at a minimum of 25 seconds. The question desired to be answered was how fast the satellite is spinning. This question led to an effort to determine the rotation rate based off of data that was usually determined to be thought of as aliased.

This characterization effort has led to several interesting discoveries for sampling sensor data near or slower than the spin rate. Firstly, it was determined that the Nyquist criteria of a minimum two samples per rotation did not accurately represent the specific situation for FS-3, and the true criteria for determining the spin rate requires only one sample per rotation prior to aliasing. The cause of this is the fact that two sensors with a known phase shift are measured at the same point in time. This allows for the determination of the exact quadrant that a representative angle is defined in, as long as the phase between the sensors is known. If the phase is not known to be positive or negative (i.e. the direction of spin), then the typical sampling criteria apply. Secondly, the time jumps in the sample rate allow for a different way of meeting the Nyquist criteria. Even though the data is theoretically undersampled, it is hypothesized that a deviation in the sample rate that is under the new criteria of once a sample can be used to determine the true angular velocity. That is, if the satellite is spinning with a 4 second rotation rate, it would be possible to determine with three data points spaced 25 and 26 seconds apart. This case has been shown to be theoretically possible, but there is still ambiguity in both the direction rate and multiples of the frequency that limit the current results. The determination of spin rate based on undersampled data is an active area of research being pursued by MIT and USAFA.

Conclusions from Spin-up Event

This section began by identifying a timeline for the different events operational, analysis, and communication event that occurred in support of the April 2010 spin-up event. This table illustrated how quickly data was analyzed and recommendations and conclusions were made on characterizing the unexpected events. USAFA FS-3 operations staff have acknowledged that without these tools and the identification of the LTAS problem, further tests would have likely have occurred. These tests would have potentially damaged the spacecraft mission and lifetime. A few lessons and applications to the topics of IEM, TDA, and satellite operations from this event are given below:

- **TDA Design** — The event demonstrated that a telemetry analysis tool proves to be very useful in characterizing an unknown event. The ability to quickly graph the accumulated data made it apparent that the satellite spun-up and drew a higher amount of power than desired. The ability to zoom and pan through the data allowed for identification of major events with accurate time tags.

  TDA Design Recommendation: At a minimum, be able to generate graphs for a specified period of time; the ability to dynamically change the time window of the data is beneficial.

- **Implementation of New Tools** — Despite having the LimWOD GUI ready, additional tools were still necessary to analyze the full WOD data format. Additionally, the work being done due to determine the spin rate despite under-sampled data will create new tools. These new tools will be developed first as individual scripts and then implemented in the GUI for past, present, and future analysis.

  TDA Design/Personnel Recommendation: Understand that maintenance and upgrades in the data analysis programs are required for a satellite after launch. If source code exists, it must be commented and documented well to allow for successful maintenance. The fastest resolution is to contact the original developer, which is far less costly if done in-house.

- **Satellite Operations Procedures** — The spin-up and power draw from LTAS was completely unexpected. As Table 5.5 #20 describes, when looking back at the previous data, the same result was shown. Had this been identified prior to the tests, no long duration tests would have been planned. Additionally, operators receive real-time data from the satellite during all contacts, which should have shown a few indicators of off-nominal performance. The real-time analysis is difficult; however, it is possible that between starting the 5 April test and starting the 8 April test, an operator could have spotted a problem and possibly stopped either test. A more concrete lesson is to download and analyze data from a significant test prior to starting the next test. In hindsight, the 20-hour test planned on 8 April should only have occurred after successfully showing the 5-hour test was successful.

  Operational Recommendation: For any test, be vigilant for off-nominal data readings

or trends that occurs immediately after the start of the test, during the test, or in conclusion of the test. Additionally, analyze a sufficient amount of data showing that the test was successful and the satellite behaved as expected prior to starting the next test.

5.3 Discussion

5.3.1 Modeling and Simulation Design Discussion

The subject of modeling and simulation has been discussed throughout the design and implementation of the IEM approach. This section discusses the lessons learned in the last two years in the topic of modeling and simulation for small satellites.

Choosing a Modeling and Simulation Platform

A spectrum of tools are available to perform the modeling and simulation in the IEM process, from specifically developed home-built tools to the full utilization of COTS software platforms. A trade exists between the performance, capabilities, accuracy, and resources (cost, time, personnel) required for the given architecture. The spectrum is full of hybrid methods that utilize COTS programs calling user-functions as well as user-defined programs calling COTS functions, and no point on the spectrum can be considered "best" for all cases. This thesis has demonstrated that flexibility is essential to any architecture, so COTS products utilized must allow for user-defined programs or scripts to be run.

COTS platforms like STK and FreeFlyer are advantageous to use as they come with an assortment of pre-defined functions that are common to the space environment. They both allow for external or internal programming to utilize their simulation engine and capabilities by other tasks, which is important for model flexibility. Additionally, support is available via tutorials, training, and customer service. COTS platforms are likely the best way to verify the general mission concept in Phase A or even Phase B; however, these programs lack the specificity required for meeting the later IEM requirements. The command-input, detailed models, and TDA elements of the IEM design require a large amount of programming specific to the given mission. While it is possible to do this in these COTS platforms, especially FreeFlyer with its programming interface, it is not a trivial task. Lastly, COTS

platforms require licenses, which may come at a cost to the organization, or, if donated, require maintenance to keep current. At critical timing junctions such as before major design reviews, the inability to access or modify the simulation or models can create a single point of failure for model-based design.

Perhaps the largest trade in the platform decision making process deals with the available resources and programming experience of the design team. Assuming that the models and simulations created will change throughout the program, it becomes clear that programmers are required in each phase. It is therefore recommended to use the experience of the personnel as the primary factor for choosing the modeling and simulation platform. The examples given in this thesis utilize MATLAB primarily because of the experience of all programmers with MATLAB. Additionally, common astrodynamics functions and tools have been written for MATLAB and available open source.

Another factor in the decision process is whether to reuse existing software tools (to include COTS platforms) or to establish a new set of tools. Reuse almost always appears as a better option, as the tool already exists and works. However, reuse has a large potential to push the schedule back due to unrealized changes in the software, as well as hiding improper assumptions that will affect the accuracy of the model. Robert Glass wrote that the software field generally considered the point where more than 10%-20% of the software had to be modified as the benchmark for when it is better to start from scratch [13]. An additional example with COTS software reuse issues can be shown through Global Positioning System (GPS) issues on the International Space Station (ISS) [14]. In short, reuse of software can have surprising costs if not done carefully on any software project, to include modeling and simulation.

Programming Methods

The evolution of the CASTOR model from the original MOTV model to its current form demonstrates how even for a given programming language, the methods used can change significantly over time. Specific to the MATLAB example, the transition from a functional architecture to OOP proved to be a challenge for many of the team members. This lesson showed that the programming methods and architecture changes become more costly as the

design matures. This lesson can be applied to changing software platforms or merging two capabilities together as well. If identified and done early in the program, these methods can be learned and implemented; however, after integrating many of the different capabilities together and writing certain design assumptions into a simulation or modeling architecture, these changes can cause setbacks to a program's resource budget.

If multiple developers are involved in the model development, it is important to identify how the work will be divided between the members. The experience over the last two years has shown that regarding different programming abilities, the best choice is to allow the programming to be near the lowest skill level. In the case of MATLAB programming, this meant to either train the personnel on OOP or to allow the personnel to create their models at the level of understanding that they had. One "architect" (or a simulation team) should be chosen to manage integration of the simulation and ensure personnel understand how to program as needed.

Model Verification and Validation

The subsystem, environmental, and orbital models used must be verified and validated prior to use for design analysis. The most common verification methods include hand-calculation, test cases on analytical or closed-form solutions, as well as comparison with a common COTS software tool. The ground testing and operations campaigns identified in the Phase D IEM design are the primary validation sources. This validation is not a perfect case, as the environment of the tests will likely not match the orbital conditions assumed in the simulation. For this reason, the initial commissioning sequence during operations will be the primary validation source of the completely integrated model. By allowing subsystem on/off switching and telemetry input, the simulation can still be effective even if a certain model does not match the real data.

Certain verification and validation checks can be performed on the functional structure of the models and simulation well before launch. All command, telemetry, and events should be tested to ensure all use cases and operational modes are represented. This testing sequence should occur throughout Phase D, and should be performed for both the actual satellite software and hardware as well as the simulation for the different cases. The command input

and output files and TDA design allows for a clear comparison between real and simulated system.

5.3.2 Personnel in the IEM Process

This thesis has a central theme of the importance of personnel throughout the life cycle of a small satellite design. It has just been suggested that the personnel expertise should drive the choice of modeling platform and approach. It has also been suggested that there should be a central modeling and simulation "architect," who is responsible for the integration and use of the integrated model and its elements. These suggestions may seem infeasible for smaller programs, especially university satellite programs, as personnel may not be readily available with the experience or even time to work on systems modeling. A rebuttal to this argument on resource allocation is that the activities described in the IEM design will pay off later in the design and operations of a satellite.

Lack of Time to do IEM

If skilled personnel exist and time is the driving factor to not perform modeling, then it is likely that time will not be available to solve problems seen in testing or operations. The IEM process invests time early in a project to assist with critical decisions throughout the design cycle, with a large emphasis on Phases D and E returns on this investment. Modeling and simulation re-work and restructuring is also costly in time, as seen through the MOTV and CASTOR examples. This thesis has attempted to give a large amount of background information to future satellite design teams so that they can look towards operational use early in the design process. The argument that there is not enough time to implement basic IEM modeling methods in the design process uses a faulty assumption: the satellite will operate as expected.

Lack of Personnel or Expertise to do IEM

The second reason for not performing modeling in the design phase is not having the personnel or expertise to perform modeling. If the program cannot create a model, it is unlikely that they can create an operational satellite. An inherent requirement of designing

a satellite is an understanding of the different components, assemblies, subsystems, and complete system. Therefore this class of reasoning is likely not that personnel do not exist, but rather skilled programmers do not exist in the program. This reasoning is also questionable, as software is critical to the successful operations of the satellite, so competent programmers must exist at some point in the program.

The IEM approach actually attempts to bring in and develop these programmers much earlier in the design process on purpose. A common theme heard and observed in the university satellite development is that software programmers do not understand the system implications, and the system engineers and program managers do not understand the software implications on their respective choices. It can be argued that this is a general theme throughout the aerospace industry, but this thesis will focus on the smaller organizations and universities. By investing the time in Phase A to find skilled programmers and attempt to model the mission and its major elements, early interaction is forced between the software personnel and mission personnel. Experience has shown that both classes of personnel perform better when they understand the other's area of responsibility. Programmers are most useful in Phase C and D of the program, when the flight software is written and modified. At this point, the simulation framework should be sufficient that no major simulation re-designs are necessary. Therefore, it is explicitly recommended that skilled programmers are used in Phases A-C to create the SIM, MAT, and TDA framework and thereby getting a better systems-level concept of the mission before they begin flight and ground software design and coding.

A more specific case for not performing certain IEM design items is in the detail of the subsystem models. The design team should strive for correct level of detail in a SIM subsystem model due to its importance in the mission and not on previous experience; however, the lack of experience in a major design element may be a programmatic issue with the satellite. If the team cannot model a subsystem successfully, most likely it is not well enough understood by the team. The solution is to either removed the subsystem complexity from the design or invest more time to understand it better and be able to model it. A prime example for this is the ExoplanetSat identification of the thermal model being a major element in their design. The team must decide to either modify the entire design so

that the thermal effects are not as critical or invest the time in researching, understanding, and modeling the thermal effects.

Turnover

Personnel turnover is a issue in the IEM process as well as in the design team in general. USAFA has an especially unique situation where its entire development team, sans a small number of faculty, leaves at the end of each May. The workforce is not replaced until August, when a new senior class enters the program. The IEM products can be used to train and develop unfamiliar or unskilled personnel. Originally conceived as operator training tools, these products are much more interactive than documents and presentations, and thus can be used by multiple types of users to gain familiarity with the system. It is recommended that a program limits critical turnover (i.e. greater than 75+% of workforce) as much as possible, to allow for direct knowledge transfer between personnel. In universities, this is done by utilizing graduate students on programs, having a diverse class breakdown (i.e. Senior, Junior, etc.), and having independent research activities during periods without class.

While it is important for all areas of the satellite design, turnover can cripple IEM process. It has been shown that the IEM products and design can assist throughout the design process, but the return on the early investment of time and effort comes in the Phase D and E implementation of the IEM elements. If a successful IEM architecture is produced, but the personnel who are familiar with it leave the design team, it is likely that the IEM tool will not be used. This can be prevented by the program utilizing the IEM approach throughout the design and planning personnel accordingly. Specifically, the IEM "architect" described should have the IEM development as his or her main responsibility, and a "architect protégé" should assist in the IEM development and serve as the replacement for the primary upon graduation or another event.

Personnel Discussion Summary

The modeling and simulation examples in this thesis have demonstrated how the products have been used. The MOTV design produced two outputs: verification results of the

design from integrated results (intended output), and identification of issues that had not been identified through interactions with the design team in support of model development (unintended). Additionally, the recent CASTOR design work implementing the commands and telemetry input into the simulation identified that both did not exist, and therefore the initial formats were created. The IEM process is meant to provide products and verification to the design; however, by asking the right questions, the process also identifies unknown areas and assists in their development.

5.3.3 FalconSAT-3 Lessons Applied to IEM

FalconSAT-3 has been discussed at great length in Sections 4.2 and 5.2. The following list describes lessons learned for consideration in small satellite or university satellite design.

- **Predict the expected** — No prediction capabilities existed for FS-3 at launch. The primary data analysts (cadets) joined the program after the previous knowledge source graduated shortly after launch. The IEM design uses the SIM, TDA, and MAT as a predictive tool in support of operations, and had such a tool been available, many operational anomalies in FS-3 would have been discovered or predicted early on. These products also can serve as training and familiarization tools. Even without fully understanding the satellite's behavior, a new analyst should be able to see if data is in bounds or deviates unexpectedly.

- **Prepare for the unexpected** — FS-3 operations and analysis has proved to be challenging, even three years after launch. Not everything will work as designed nor will every assumption used in the satellite and simulation design be true. Flexibility and a broad TDA visualization assist in quick identification and investigation of anomalous behavior.

- **Collect data continuously** — The communications and data budgets and avionics design should allow for continuous data collection of critical health data. Understanding early in the design process that data will be saved continuously allows the design team to prepare better for a data management and telemetry analysis plan, rather than adding it after launch like in FS-3.

- **Timing** — Ensure time stamps are correct and synchronized for all types and formats of data. Limit assumptions on timing and time stamps as much as possible, and if assumptions are made (e.g. constant time step), verify in ground testing and on-orbit operations that they are true.

5.3.4 Thesis Questions Revisited

Section 1.2.6 provided a list of the main topics and questions that this thesis set to answer. This section describes how the questions have been addressed through the design, implementation, and results of the IEM approach.

University Satellites

What is the correct role and effort that integrated modeling should have throughout the design and operational life of a university satellite?

The thesis has suggested an approach for using integrated modeling throughout the design cycle. Specific to university satellites, this approach is thought to verify the design, assist in identification of unknown technical problems, serve as a training, familiarization, and documentation source, and serve a critical role in the functional testing and operations of the satellite. The IEM approach identified a general architecture to follow and stressed that the specific details for the models will depend on the application and mission. A event-based MATLAB approach was described in Section 4.1 for an example of a simulation that met the IEM architecture. The personnel discussion section describes how integrated modeling is an investment of time, and programmers can be utilized earlier in the program to build the simulation platform and become knowledgeable with the mission prior to writing flight software. The benefits of a IEM can be seen through the FalconSAT-3 examples, where only partial elements of the IEM design, such as a basic TDA plotting tool, assisted greatly in the rapid identification and analysis of anomalous data.

Mission Assurance

How can integrated modeling validate mission assurance requirements without requiring additional commitment of time or manpower from the design team?

The IEM method suggested in this thesis is centered around mission assurance, whether in the design process or during operations. The purpose for spending the time, effort, and money on a satellite is to perform a certain mission. Therefore at each phase of design in the IEM approach, a mission assurance output is produced, to include building the specific MAT by Phase C. While this question cannot be successfully answered, as the necessary tools developed in the IEM process require time and manpower from the design team, the time invested into the integrated modeling process is used to ensure mission success.

Space Engineering Academy

How can integrated modeling be implemented into a structured satellite design and build curriculum such as SEA?

The IEM design approach is well-matched to the SEA curriculum. By implementing the IEM approach in the SEA program, SEA participants will be able to ensure their satellite design is meeting the intended mission. The operational tools developed will allow future SEA classes to quickly analyze orbital data and focus on the next satellite design. The existing tools and their respective developmental lessons will assist in the development of the next iteration of the integrated model. This process, while foreign at the onset, will become standard practice in the SEA satellite design.

CASTOR

How will integrated modeling be used to ensure that the CASTOR satellite will meet its mission objectives after launch?

The IEM process described in this thesis has been implemented in the CASTOR satellite program. Successful completion of the CASTOR SIM, TDA, and MAT elements will allow for complete systems testing and successful operational analysis. The difficulties in predicting and measuring the thruster's performance will be overcome by utilizing an integrated performance model and MAT throughout the CASTOR operations. The TDA and SIM elements will also allow for the subsystem and component health to be monitored to identify and characterize anomalous behavior. Using the IEM process and its resulting tools will allow for on-orbit characterization of a new electric propulsion technology (DCFT) with a relatively low-cost spacecraft.

FalconSAT

How can the lessons learned through FalconSAT-3's lifetime be applied to future satellite designs such as CASTOR and FalconSAT-5 so that the on-orbit anomalies can be prevented or resolved more quickly?

The overall lessons learned from FS-3 that apply to the IEM process were summarized in Section 5.3.3, and those specific to the TDA design were described in Section 5.2.2. FS-3 has shown that at a minimum, a graphical analysis tool should exist to quickly view and analyze telemetry, and was proven to be critical in the April 2010 event. The IEM design is much more extensive, and would have been of great use in the FS-3 mission.

Chapter 6

Conclusion and Future Work

This thesis has outlined an Integrated Evolutionary Model (IEM) approach for developing small satellites. This approach suggests that an integrated performance model developed and maintained through each phase of the design sequence greatly assists in design verification and mission assurance. A MATLAB discrete event, command-based simulation framework was developed and for the CASTOR satellite and used as an example of the IEM SIM element. FalconSAT-3 data analysis tools were described to support the discussion of the TDA design. A discussion was completed on how the IEM approach can be applied to the SEA program and university satellites.

6.1 Conclusion

This thesis developed a model-based approach to small satellite engineering with a hypothesis that this approach will save resources. This will be verified through the SEA program as well as future MIT SSL satellites implementing the IEM approach. Feedback on IEM use and applicability will be obtained, although it will likely not be clear if the IEM approach truly saves resources in these arenas. This thesis has highlighted how IEM tools developed for FS-3 operations (i.e. TDA in the LimWOD GUI) were critical in the identification, investigation, and resolution a serious attitude dynamics and power system event that occurred in April 2010. These tools saved time and energy for the supporting personnel, as well as limited the risk to the satellite, as the satellite would likely have been tested beyond

its design limits had the event not been identified. For this reason, the SSRC is now performing changes to utilize the IEM approach in operations and design. While this is only one event, it was performed with a simplified TDA design, and shows the potential of using the IEM approach for future satellite development.

6.2 Future Work

The IEM approach was developed from the experience and lessons of several small satellite programs. However, the method identified in the IEM approach has never been fully tested from Phase A to Phase E. Therefore a major area of future work is to implement IEM at the start of a program and attempt to measure its utility throughout the program's lifetime. This is planned to be performed in the SEA program.

Additionally, a significant amount of work remains for the CASTOR satellite integrated model. For the SIM design, the subsystem models must be completed, and the command and telemetry designs, once complete, must be implemented and maintained. Adding the ability to add commands and callbacks with a clean interface rather than in the source code is a potential area of improvement. The TDA structure needs designed and created, then used in ground testing to verify the satellite software and hardware. The CASTOR MAT needs to be created by either adding elements to the propulsion system in the SIM design or by creating a stand-alone program as outlined in Section 4.3. This tool should be tested to show that the thruster can be characterized given a given set of expected attitude and thruster data. The MAT should also be able to analyze the pictures taken of the DCFT plume. The completed set of tools must be integrated to show the complete verification of the CASTOR satellite mission as well as for verification with flatsat hardware and software tests.

Appendix A

MATLAB Code Examples

This appendix gives several applied examples of MATLAB code used throughout the thesis. The purpose of showing these examples is to assist a user unfamiliar with MATLAB to perform certain functions. The MATLAB help tools and online support are very useful resources to gain additional information on any of the functions or methods used. An electronic copy of all code used in this thesis is available upon request from MIT SSL or the author.

Time Code

```
%***Begin time.m function ***
function [ tout ] = time(time, intype, outtype)
% function [ tout ] = time(time, intype, outtype)
%   Time.m changes the time for different inputs to different outputs,
%   primarily by using the MATLAB built in timing functions
%
% List of types:
% 1 - Date string (example: 'MM/DD/YYYY HH:MM:SS')
% 2 - Julian Date (days since 4713 BCE)
% 3 - GSFC Modified Julian Date (days since Jan 5 1941) (JD-2400000.5-29999.5)
% 4 - USNO Modified Julian Date (days since Nov 17 1858) (JD-2400000.5)
% 5 - Unix Time (seconds since Jan 1 1970)
% 6 - Day of Year (decimal) (output only)
% 7 - Year, Day of year (YYDOY.xxxx) (output only, string of 10 characters)
% 0 - Matlab time (days since 1 Jan 0000)
%
% Inputs: time  -number or string of input time (can be vector or struct of
%                strings)
%         intype- Format of input
%         outtype- Format of output
% Output: tout - Converted time

%Conversions
matlab_jd=1721423.5-365;
unix_matlab=719529;
```

```
switch intype
    case 1 % String input
        mattime=datenum(time);
    case 2 %Julian date input
        mattime=time-matlab_jd;
    case 3 % GSFC Modified Julian Date input
        mattime=time+2400000.5+29999.5-matlab_jd;
    case 4 %USNO Modified Julian Date Input
        mattime=time+2400000.5-matlab_jd;
    case 5 %Unix Input
        mattime=time/86400+unix_matlab;
    case 0 %Matlab time input
        mattime=time;
    otherwise
        error('Improper input type, see time.m documentation')
end

switch outtype
        case 1 % String output
        tout=datestr(mattime);
    case 2 %Julian date output
        tout=mattime+matlab_jd;
    case 3 % GSFC Modified Julian Date output
        tout=mattime-2400000.5-29999.5+matlab_jd;
    case 4 %USNO Modified Julian Date output
        tout=mattime-2400000.5+matlab_jd;
    case 5 %Unix output
        tout=(mattime-unix_matlab)*86400;
    case 6 %DOY output
        [Y M D h m s]=datevec(mattime);
        tout=mattime-datenum(Y,1,1,0,0,0);
    case 7 % YYDOY output
        tout=yydoy(mattime);
    case 0 %Matlab output
        tout=mattime;
    otherwise
        error('Improper output type, see time.m documentation')
end

function tout=yydoy(mattime)
%Gets the time in yydoy.xxxx format for a given set of matlab times
[Y M D h m s]=datevec(mattime);
doy=mattime-datenum(Y,1,1,0,0,0);
yyyy=num2str(Y);
tout={};
for i=1:size(mattime,1)
    if doy(i)<10
        str=['00' num2str(doy(i))];
    elseif doy(i)<100
        str=['0' num2str(doy(i))];
    else
        str=num2str(doy(i)); %three character or more array
    end
    str=[yyyy(i,3:4) str];
    if size(str,2)==5; %if exactly on day (no fraction)
        str=[str '.0000'];
    elseif size(str,2)==7 %tenth of day
        str=[str '000'];
    elseif size(str,2)==8 %hundredth of day
```

```
        str=[str '00'];
    elseif size(str,2)==9 %thousandths of day
        str=[str '0'];
    elseif size(str,2)>10 % many digits
        str=str(1:10);
    end
tout{i,1}=str;
end
tout=char(tout);

%***End time.m function ***
```

Selecting Files with MATLAB

```
%*** Example of Using "uigetfile" in limwodGUI.m code ***
curdir=cd; %Get string of current directory

%Set the default path for attitude files (using SVN file structure)
afdir=[curdir(1:(length(curdir)-7)) '\data\af files\'];

%Use uigetfile.m to select the file(s)
[filename, pathname]=uigetfile([afdir '*.csv'], 'Select attitude file(s)',...
                               'MultiSelect','on');
if isequal(filename,0)
    return %Return if user cancel
end

addpath(pathname) %Add attitude file path to directory

%Perform analysis on files given by "filename" (not shown)

%*** End uigetfile example ***
```

Zooming and Panning with Datetick

```
%**** Begin set_graphs.m function ****
function set_graphs()
% set_graphs automatically sets zooming, panning, and data cursor callbacks
% for plots using matlab time on the x-axis. Call after the plot function

datetick
h=zoom;
set(h,'ActionPostCallback',@redo_tick)
f=pan;
set(f,'ActionPostCallback',@redo_tick)
fig=gcf;
dcm_obj = datacursormode(fig);
set(dcm_obj,'UpdateFcn',@data_fcn)

function redo_tick(obj,evd)
%For updating the ticks when zooming or panning
datetick('x','keeplimits')

function txt = data_fcn(empt,event_obj)
```

```
pos = get(event_obj,'Position');
txt = {['Date: ',datestr(pos(1),'mm/dd/yyyy')],...
    ['Time: ',datestr(pos(1),'HH:MM:SS')],...
    ['Output: ',num2str(pos(2))]};
%**** End set_graphs.m function ****
```

Basic Plotting

```
% **** Begin example.m ****
% example.m provides a script to demonstrate the time and set_graphs
% functions as well as basic plotting functions in MATLAB

%Create data to plot
to=now; %current time (in matlab time)
t=to:1/86400:(to+10/24/60); %1 second samples for 12 minutes
w=2*pi/(1/24/60); %angular frequency for 1 minute period
y=sin(w*(t-to));

%Plot
plot(t,y)
set_graphs %Call set graphs function
doy_start=time(t(1),0,7); %Get YYDOY string function
title({'Example output of sinusoid';['Start Time: ' datestr(t(1),21)...
    ', YYDOY: ' doy_start]})
ylabel('Amplitude')
xlabel('Time')
legend('Y_{example}','Location','SouthOutside')

% **** End example.m ****
```

Bibliography

[1] *SMC Systems Engineering Primer & Handbook*. Space and Missile Systems Center, third edition, April 2005.

[2] NSS 03-01. *Guidance for DoD Space System Acquisition Process*. Department of Defense, December 2004.

[3] Richard H. Battin. *An Introduction to the Mathematics and Methods of Astrodynamics*. Reston: AIAA, 1999.

[4] Lucy E. Cohan. Integrated Modeling to Facilitate Control Architecture Design for Lightweight Space Telescopes. Master's thesis, Massachusetts Institute of Technology, 2007.

[5] Lucy E. Cohan. *Integrated Modeling and Design of Lightweight, Active Mirrors for Launch Survival and On-Orbit Performance*. PhD thesis, Massachusetts Institute of Technology, 2010.

[6] Corey Crowell. Personal Interview Regarding CASTOR Model Development, April 2010.

[7] Corey Crowell, Matt McCormack, George Sondecker, and Ryan McLinko. CASTOR 16.851 Final Presentation, November 2009.

[8] Olivier de Weck. Integrated Modeling and Dynamic Simulation for the Next Generation Space Telescope. Master's thesis, Massachusetts Institute of Technology, 1999.

[9] MATLAB Digest. Introduction to Object-Oriented Programming in MATLAB <`http://www.mathworks.com/company/newsletters/digest/2008/mar/matlab_oop.html`>, March 2008.

[10] Dov Dori and Edward F. Crawley. *Object-Process Methodology: A Holistic Systems Paradigm*. Springer-Verlag New York, Inc, 1999.

[11] Worldwide CDIO Initiative: A Framework for the Education of Engineers. <`http://www.cdio.org`>.

[12] Samuel Gay and Nicholas A. Schmiegel. Falconsat-3 and the Space Environment. In *48th AIAA Aerospaces Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, number AIAA 2010-182, 2010.

[13] Robert L. Glass. Reuse: What's Wrong with This Picture? *IEEE Software*, March/April 1998.

[14] Susan F. Gomez and Michael L. Lammers. Lessons Learned from Two Years of On-Orbit Global Position System Experience on International Space Station. In *ION GNSS 2004*, Long Beach, CA, September 2004. NASA Johnson Space Center.

[15] Object Modeling Group. *MDA Guide Version 1.0.1*. June 2003.

[16] HM-RB-2001-1. *Human Computer Interface Display Conventions for Space Systems Operations*. Space and Missile Systems Center Standard Practice, 1 January 2001.

[17] INCOSE-TP-2003-002-03. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. International Council on Systems Engineering (INCOSE), June 2006.

[18] Cyrus D. Jilla. Separated Spacecraft Interferometry — System Architecture Design and Optimization. Master's thesis, Massachusetts Institute of Technology, 1998.

[19] Greg Johnson and Sebastioan Munoz. Copernicus, A Generalized Trajectory Design and Optimization System. Presentation, November 2003.

[20] Michael S. Kimbrel. Optimization of Electric Propulsion Orbit Raising. Master's thesis, Massachusetts Institute of Technology, 2002.

[21] Wiley J. Larson and James R. Wertz, editors. *Space Mission Analysis and Design*. El Segundo: Microcosm, third edition, 1999.

[22] Andrew W. Lewin. Streamlining Satellite Development, Testing, and Operations Using a COTS Command and Telemetry Package. In *Small Satellite Conference*, 2010.

[23] Matt McCormack. Personal Interview Regarding CASTOR Model Development, April 2010.

[24] NASA-STD-7009. *Standard for Models and Simulations*. National Aeronautics and Space Administration, July 2008.

[25] NASA/SP-2007-6105. *NASA Systems Engineering Handbook*. National Aeronautics and Space Administration, December 2007.

[26] Isaac Nason, Jordi Puig-Suari, and Robert Twiggs. Development of a Family of Picosatellite Deployers Based on the Cubesat Standard. In *Proceedings of the IEEE Aerospace Conference, Inst. of Electrical and Electronics Engineers*, volume 1, pages 457–464, Piscataway, NJ, 2002.

[27] Inaba Noriyasu, Hidemi Hase, Hiroyuki Miyamoto, Yoshiyuki Ishijima, and Shiro Kawakita. A Satellite Simulator and Model Based Operations in Quasi-Zenith Satellite System. In *AIAA Modeling and Simulation Technologies Conference*, number AIAA 2009-5813, 2009.

[28] Cesar Ocampo. An Architecture for a Generalized Spacecraft Trajectory Design and Optimization System, 2003.

[29] Troy W. Pannebecker. Integrated Spacecraft Design Tools. Master's thesis, Naval Postgraduate School, 1999.

[30] John Richmond. An Adaptive Thermal Model Architecture for Small Satellite Applications. Master's thesis, Massachusetts Institute of Techonlogy, 2010.

[31] Joseph Robinson. The Relationship between Engineering and Operations in a Small Satellite Program. In *AIAA Region V Student Conference*, 2008.

[32] Joseph Robinson, John Richmond, Chris Pong, and Aaron Johnson. MOTV 16.851 Final Presentation, December 2009.

[33] Lester L. Sackett, Harvey L. Malchow, and Theodore N. Eduelbaum. Solar Electric Geocentric Transfer With Attitude Constraints: Analysis. Technical Report NASA CR-134927, The Charles Stark Draper Laboratory, Inc., Cambridge, MA, August 1975.

[34] K.E. Siegenthaler, J.J. Sellers, D.A. Miller, T.J. Lawrence, D.J. Richie, , and D.J. Barnhart. The Undergraduate Satellite and Rocket Design, Fabrication and Launch Program at the US Air Force Academy. In *International Symposium IGIP/IEEE/ASEE*, Fribourg, Switzerland, September 2004.

[35] Matt Smith. Written Communications on ExoplanetSat IEM Process, April 2010.

[36] TOR-2007(8546)-6018. *Mission Assurance Guide*. The Aerosapce Corportation, July 2007.

[37] Defense Acquistions University. *Systems Engineering Fundamentals*. 2001.

[38] USAFA CSOPS. *FalconSAT-3 On-Orbit Handbook*, May 2008.

[39] IAGA Division V-MOD Geomagnetic Field Modeling: IGRF Proper Use. <`http://www.ngdc.noaa.gov/IAGA/vmod/igrfhw.html`>, January 2010.

[40] David Vallado. Fundamentals of Astrodynamics and Applications Software Routines. <`http://www.smad.com/vallado/`>.

[41] David A. Vallado. *Fundamentals of Astrodynamics and Applications*. El Segundo: Microcosm, third edition, 2001.

[42] Iris Vessey and Sue A. Conger. Requirements Specifications: Learning Object, Process, and Data Methodologies. *Communications of the ACM*, 37(5), May 1994.

[43] Kyle Volpe. Application of the Backward-Smoothing Extended Kalman Filter to Attitude Estimation and Prediction using Radar Observations. Master's thesis, Massachusetts Institute of Technology, 2009.

[44] Rodger E. Ziemer and William H. Tranter. *Principles of Communications: Systems, Modulation, and Noise*. Wiley, 2002.