Quantative Selection and Design of Model Generation Architectures for On-Orbit Autonomous Assembly

by

Swati Mohan

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

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author ....
Department of Aeronautics and Astronautics
March 15, 2010

Certified by ....
David W. Miller
Professor of Aeronautics and Astronautics
Thesis Committee Chair

Certified by ....
Olivier de Weck
Associate Professor of Aeronautics and Astronautics and Engineering Systems Division

Certified by ....
Steven Dubowsky
Professor of Mechanical Engineering and Aeronautics and Astronautics

Certified by ....
Alvar Saenz-Otero
Research Associate, MIT Space Systems Laboratory

Accepted by ....
Eytan H. Modiano
Chairman, Department Committee on Graduate Students

Quantative Selection and Design of Model Generation Architectures for On-Orbit Autonomous Assembly

by

Swati Mohan

Submitted to the Department of Aeronautics and Astronautics
on March 15, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

On-orbit assembly is an enabling technology for many space applications. However, current methods of human assisted assembly are high in cost and risk to the crew. Thus, there is a desire to automate the on-orbit assembly process using robotic technology. Automation introduces additional challenges in the design of the assembly, particularly within the control systems. During an assembly sequence, an assembler robot can undergo multiple reconfigurations of its geometry and dynamics as it attaches to and releases from individual modules. The particular problem addressed in this thesis is how to account for mass and stiffness property variations that occur with changes in configuration. Proper model generation for each configuration is critical to maintain control system stability and efficiency. This thesis explores two specific challenges associated with this problem: (1) the design of a model generation architecture to where module mass property information is known, but specific configurations are unknown; and (2) the selection of a model generation architecture that is appropriate for a given assembly architecture.

Literature review of the possible model generation architectures revealed a gap in the literature, when models are aggregated online based on module mass property information. The challenge is resolved through the design of an architecture, called Online Model Calculation. Online Model Calculation uses module information obtained at the time of attachment to generate the model for the current configuration online. This is accomplished through the parameterization of the control algorithms with respect to a property vector. The property vector contains mass property information (ex. mass, inertia) that is used to generate the model. The design of Online Model Calculation, both in terms of framework and algorithm parameterization, is successfully implemented and validated on hardware. Results show a tracking error performance improvement when the correct model of the system is used in the control system over an unupdated model of the assembler along. Online Model Calculation balances *a priori* knowledge about the possible configurations with identification of the model online.

The challenge of selecting a model generation architecture is accomplished through the development of a process that downselects feasible architectures into a single optimal architecture. This process is based on a set of generalized, quantitative metrics that are used to compare architectures from the perspectives of the control system, spacecraft, and assembly operation levels. It is exercised on three case scenarios, using a simulation tool that is developed to evaluate the model generation architecture metrics for a given assembly architecture. Results clearly show that for different assembly scenarios, different model generation architectures perform best. The quantification of this performance difference and the process for selecting the appropriate architecture constitute a key contribution of this work.

Thesis Committee Chair: David W. Miller
Title: Professor of Aeronautics and Astronautics

Thesis Committee Member: Olivier de Weck
Title: Associate Professor of Aeronautics and Astronautics and Engineering Systems Division

Thesis Committee Member: Steven Dubowsky
Title: Professor of Mechanical Engineering and Aeronautics and Astronautics

Thesis Committee Member: Alvar Saenz-Otero
Title: Research Associate, MIT Space Systems Laboratory

Acknowledgments

This research was performed at the MIT Space Systems Laboratory and supervised by Dr. David W. Miller. This research was supported by multiple sources: NASA SBIR Contract No. NNM07AA22C Self-Assembling Wireless Autonomous Reconfigurable Modules; NASA's Harriet G Jenkins Pre-doctoral Fellowship Program; and the Amelia Earhart Fellowship Program. I would like to thank my entire committee for their guidance and support, as well as the thesis readers Dr. Daniel Kwon and Dr. Daniel P. Scharf for their helpful comments.

I would like to thank the entire SPHERES team over the past five years. Working with them has been a pleasure. In particular, SPHERES team members Dr. Simon Nolet, Christophe Mandy, Dr. Enrico Stoll, Jacob Katz, and Andrew Wang have been especially helpful in providing guidance on the SPHERES system and help running tests when I could not be present at MIT. Also, thanks to my former officemates, Sarah Shull, Lucy Cohan, Christophe Mandy, and Alessandra Babuscia. It was a pleasure sharing an office with you.

Finally, I would like to thank my family for their bountiful love and support. To Santhosh, thank you for putting up with the hectic travel schedule and being so understanding and loving. I could not have asked for or imagined a better husband. To my parents, thank you for your unwavering faith and support in me. Your pride in me made me believe I could accomplish anything.

Contents

List of Figures

List of Tables

Nomenclature

$\alpha_{location}$	attachment point location
α_{status}	status of attachment point, whether an object is attached to it
α_{type}	type of attachment point
η_{ctrl}	Control period
$\hat{a}$	estimated parameter vector used in adaptive control
κ	stiffness matrix associated with spring forces
Λ	Time constant associated with integral term
Ω	Duty cycle
ψ	integration term used in control to sum up error
τ	Integration time period
$\vec{D}_{act}$	actuator force directions
$\vec{D}_{sensor}$	sensor measurement directions
$\vec{F}_{act}$	actuator force magnitudes
$\vec{F}_{ref-sensor}$	sensor reference point
$\vec{F}_{sensor}$	sensor magnitude and range
$\vec{I}$	Inertia tensor

$\vec{r}_{act}$	actuator locations with respect to assembler
$\vec{r}_{cg}$	center of mass
$\vec{R}_{ref-act}$	actuator reference point
$\vec{r}_{sensor}$	sensor locations with respect to assembler
Ξ	Mixing Matrix
ζ	Damping ratio
e	state error
f	Actuator commands
imp	actuator impulse
K_d	Derivative Gain
K_i	Integral Gain
K_p	Proportional Gain
p	property vector used in online model calculation
p_{am}	property vector of the assembler
p_{pl}	property vector of the module
u	control input
w_n	control bandwidth
ACRRES	Autonomous Control Reconfiguration for Robotic Exploration Systems
ALMOST	Assembly of Large Modular Optical Space Telescopes
CM	Center of Mass
CMG	Control Moment Gyroscopes

DOF	Degrees of Freedom
EELV	Evolved Expendable Launch Vehicle
EKF	Extended Kalman Filter
EMFF	Electro-magnetic Formation Flight
EVA	Extravehicular Activity
HST	Hubble Space Telescope
ISS	International Space Station
JWST	James Webb Space Telescope
LEO	Low Earth Orbit
M	System Model
m	mass
NASA	National Aeronautics and Space Administration
PADS	Position and Attitude Determination System
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
RMS	Root-mean-square
SPHERES	Synchronized Position Hold Engage Reorient Experimental Satellites
SWARM	Self-Assembling Wireless Autonomous Reconfigurable Modules
TDMA	Time Division Multiple Access
TRL	Technology Readiness Level
UDP	Universal Docking Port

Chapter 1

Introduction

On-orbit assembly is a key enabling technology for many space missions, such as space telescopes. Current methods of on-orbit assembly are limited to human-assisted assembly. Human-assisted assembly, though very flexible and capable, has the high cost and risk associated with manned flight. This is epitomized in the assembly of the International Space Station (ISS), a very complex structure successfully assembled through human extra-vehicular activities (EVA) and tele-operation of robotic arms. The assembly has taken over 41 launches of the Space Shuttle and spanned more than a decade. The high cost, time, and risk associated with current on-orbit assembly methods leads to a desire to automate the process using robotic technology. Introducing automation creates additional challenges in the design of the assembly mission. Much of the work done to date in relation to on-orbit autonomous assembly is either high level concept studies or low-level controller design. Few methodologies or design practices are in place to help engineers design with autonomous assembly in mind.

This thesis develops tools to guide engineers to design specifically for autonomous assembly. This chapter explains the benefits of on-orbit assembly and the motivation to move toward autonomous assembly. Then, this chapter describes the exact problem under consideration, the relevant literature, and the specific contributions of this work with a road map to the remaining chapters.

1.1 Motivation

On-orbit assembly is a technique that enables space construction, the ability to build large space structures. The size of a spacecraft is limited by launch vehicle specifications, such as launch mass and payload fairing size. Mass can be directly correlated to capability, such as the mass of additional science instruments or fuel mass which restricts mission duration. Payload fairing size can also restrict capability, such as the diameter of a space telescope mirror. On-orbit assembly bypasses these limitations by using multiple launches to send the spacecraft up in pieces.

The benefits of on-orbit assembly have been demonstrated in-flight through the construction of the Russian MIR space station and ISS. The MIR space station was built using on-orbit assembly from 1986 to 1996; similarly, the ISS began assembly in 1998 and is scheduled to be complete in 2010. The success of these structures demonstrates the great benefit of on-orbit assembly. Due to their large size, space stations are able to serve as orbital platforms, specializing in long duration microgravity sciences [57]. The science discoveries made on MIR and ISS (e.g. [9], [75], [78], [71]) demonstrate that on-orbit assembly is a key tool for future space missions. There are three types of missions that can significantly benefit from on-orbit assembly: human exploration missions, space tourism, and scientific missions.

Human exploration missions

NASA's long-term plans for human exploration focus on exploration of the Moon and Mars [4]. To accomplish this, concept studies include in-space construction of large structures, such as lunar outposts. Using human-assisted assembly to build a lunar outpost can become prohibitively expensive if each construction trip requires sending humans to the Moon. [84] [22]

Space tourism

As more people travel into space, the need for on-orbit destinations for them to visit increases. Large space structures are prime space tourism destinations. Economics dictates that the total time for assembly of the large space structure should be as short as possible to maximize utilization and profit. The historical data for on-orbit assembly shows that the construction of a space station takes on the order of a decade. Driving reasons for this long assembly time are the preparation time required to train and launch humans and to build new tools for each particular mission. Thus, current methods of on-orbit assembly do not satisfy the needs for the commercial space industry, leading companies to consider alternative options. Although other techniques such as deployable [54] or inflatable [1] structures have been considered, even designed, only on-orbit assembly has been demonstrated in space.

Scientific missions

Many future science missions are dependent on on-orbit assembly to transition from concept to reality. There are three issues that affect scientific missions: size, location, and cost.

Three examples of scientific missions that need on-orbit assembly are space telescopes, fuel depots, and solar power stations. The effectiveness of these missions is determined by their size. For example, the larger the diameter of the primary mirror of a space telescope, the greater the angular resolution, which increases the science benefit. Current telescopes are limited by launch payload fairing diameters (ex. Hubble Space Telescope), or by complex deployable mechanisms (ex. James Webb Space Telescope).

Some missions, particularly for space telescopes, prefer to be in locations outside of Low Earth Orbit (LEO), such as the Earth-Moon or Sun-Earth Lagrange points. Current methods of on-orbit assembly do not allow for these locations because there is no mode of transportation to allow humans to travel to these locations.

Most scientific missions are under strict budget restrictions, which makes it impossible to bear the burden of the additional cost of human spaceflight. For example, total mission cost is greater for manned missions because launching on a man-rated launch vehicle is more expensive than launching on an evolved expendable launch vehicle (EELV). The cost of launching a payload on the Space Shuttle in 2000 was roughly \$4.7K/kg. The cost of launching the payload on an similar heavy lift launch vehicle, such as Russia's Proton rocket, was as low as \$1.9K/kg [3] (Costs are given in FY2000 dollars).

Tele-operation, as a construction method, has also been used as a human-assisted form of assembly. Though tele-operation provides benefits associated with the response of a human who is capable of anticipating failures and/or obstacles, the real-time execution of this method provides drawbacks as missions become more complex and are farther away from Earth. If operating from the ground, communication delays prevent real-time commanding, necessitating a rudimentary form of automation. If operating from orbit, once again, the issues associated with human space flight arise. The three main limitations of current on-orbit assembly methods are summarized below:

1. The time frame associated with human assisted assembly (roughly a decade) is too long for future space missions.

2. Locations for assembly are limited by where humans can travel, namely LEO and possibly the Moon.

3. The cost of assembly is expensive due to the additional cost associated with astronaut participation, such as the use of human-rated launch vehicles.

These limitations can all be addressed by implementing autonomous assembly using robots. Robotic assembly serves as a low-cost, low-risk option to human-assisted assembly. A complete robotic assembly can occur at the onset of the mission, reducing the assembly time from years to weeks, even days. Using robotics also enables assembly missions in significantly more locations than human-assisted assembly can reach. The space industry has demonstrated the capability to reach unique and far reaching locations robotically, such as a comets, asteroids, and almost every major planetary body in the solar system. The robots and spacecraft can be launched on an EELV, instead of a human rated launch vehicle, which leads to a significant cost reduction for large space structures with multiple launches. Also, a structure that is assembled is inherently modular, which facilitates on-orbit servicing and upgrades [40] [61] [7] [68].

There has been little research conducted to address these limitations other than concept studies. Automation introduces several issues across multiple fields, such as assembly sequencing, robotics, and control. The next section provides a description of the challenge considered in this work.

1.2 Problem Definition

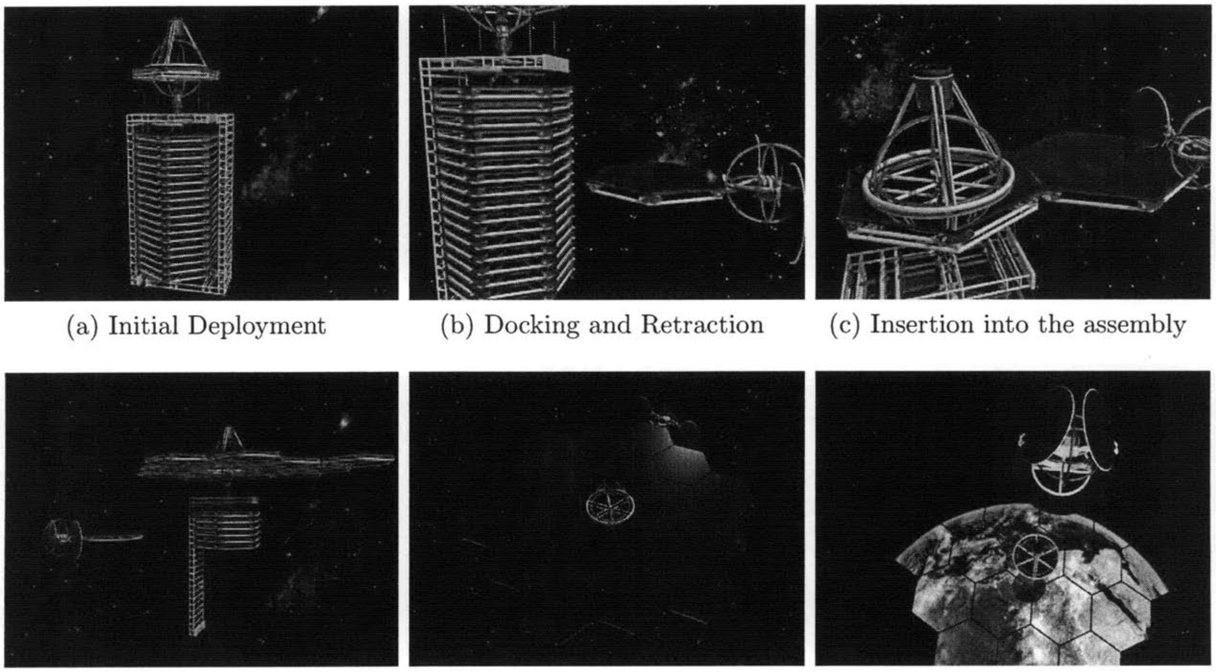
There are many key challenges associated with designing an autonomous assembly mission, such as standardized interfaces, assembly sequencing, hardware design, and control system design. Of the various challenging areas of assembly work, the control system is of particular interest. The control system is responsible for execution of the assembly sequence and therefore, the overall performance of the assembly. The design of the control system impacts the maneuverability and resource efficiency. Failure to properly design the control system can lead to overall failure of the mission. A key aspect of control system design is the dynamics model that is used to represent the system. This thesis considers the specific problem of generating the model used in the control system.

In autonomous assembly architectures, a robot, called the *assembler*, is responsible for maneuvering a set of *modules*. These modules are the components that combine to form the assembled structure. There can be many types of assemblers, such as self-assembly, tugs, or robotic arms. In self-assembly missions, the robots are both the assembler and module, because they maneuver themselves into the desired final position. In missions using tugs or robotic arms, the assembler is physically and functionally separate from the module, and is responsible for maneuvering modules into position. Assembly missions that use techniques other than self-assembly face a challenging control problem associated with model changes due to the attachment of assembler and modules. The dynamics change each time the assembler attaches to and releases a modules. This changes requires an associated change in the control algorithms to maintain requisite precision.

A *configuration* is defined as a unique physical attachment of assemblers and/or modules. When the assembler attaches to a module, the configuration of the assembler changes. Many configurations may be present during the assembly process. The change from one configuration to another through the act of attachment or detachment is defined as a *transition*. The set of all transitions in their order of execution specifies the assembly sequence. The transitions are important because they define the changes in the model, and associated control, that can occur throughout the assembly process.

The transition from one configuration to another can introduce large mass and stiffness variations. For example, attaching to a rigid module results in the addition of mass and inertia from the module, and attaching to a flexible module can also introduce flexible dynamics into the system. Though there are many, possibly drastically different, configurations associated with a single assembly process, the assembler usually consists of a single control system. In order to handle the multiple configurations, the control system must be sufficiently reconfigurable to take in a model of the configuration and generate the proper control commands. A key aspect in the design of the control system is the generation of the model that is appropriate for the current configuration and how the model is made available to the control system. The **model generation architecture** is the algorithm framework used to calculate the model of the configuration and propagate it into the control system. The choice of model generation architecture impacts the overall performance of the control system, such as transition time and resource utilization efficiency.

An example assembly scenario is depicted in Figure 1-1. Figure 1-1 shows a telescope assembly using an Electro-magnetic Formation Flight Vehicle (EMFF) as the propellant-less assembler [44]. The steps shown in Figure 1-1 are initial deployment (Figure 1-1a), docking and retraction (Figure 1-1b), insertion into the assembly (Figure 1-1c), repetition for all modules (Figure 1-1d), completion of assembly (Figure 1-1e), and operations (Figure 1-1f). Each of the eighteen modules can have distinct mass properties. Thus, the control system for the assembler shown must properly maneuver in at least eighteen different assembler-module configurations to complete the assembly process successfully.



(a) Initial Deployment (b) Docking and Retraction (c) Insertion into the assembly

(d) Repetition for all modules (e) Completion of the assembly (f) Operations

Figure 1-1: Example assembly scenario - Space telescope assembly via EMFF assembler

Problem Under Consideration

How does one select and design a model generation architecture to maintain control system performance at each configuration change throughout the assembly sequence, in spite of large mass and stiffness property variations?

The model generation architecture must be designed to handle all of the configurations, while maintaining performance and versatility. Performance in this work is defined as stability, resource efficiency (e.g. fuel and time), and trajectory accuracy. Versatility is defined as the ability to adapt to changing mission needs or modules.

An example of increasing versatility in the model generation architecture is the minimization of hardcoded transitions and properties used. Minimization of hardcoded properties, for example, means the model generation architecture is capable of identifying the properties online. Online identification allows the assembler to attach to any module, which increases its versatility because the same assembler can be used for multiple assembly operations.

Scope

This thesis directly addresses the problem of generating control system models for the myriad of assembler-module configurations that arise during the assembly sequence. It does not consider the configuration changes associated with the build-up of the assembled structure, although techniques similar to those developed in this work can be applied. Thus, the configuration changes due to the model aggregation of self-assembly modules are not considered, only the maneuvering of the modules is considered.

1.3 Literature Review

Review of four main areas of research was conducted: sequence planning, robotics, reconfigurable control systems, and assembly concept studies. Review of these areas helped to identify the science need for autonomous assembly and the research issues being explored for autonomous assembly in general. The review of literature on these four topics is not included in this chapter because it does not directly relate to the problem under consideration. It only sets a context for the state of the autonomous assembly field. The literature review of these topics is available in Appendix A.

The area of literature most relevant to the research presented in this thesis is reconfigurable control systems. Reconfigurable control systems are designed to handle changing plant, actuator, or sensor dynamics. Part of the design of the reconfigurable control system involves model generation. This thesis considers reconfigurable control system designs in terms of the model generation architecture that they employ.

1.3.1 Model Generation Architectures

Model generation architectures can be classified by the amount of *a priori* information available to the designer. The architectures can be conceptualized on a spectrum where they are distinguished by how they employ different techniques of model generation based on the *a priori* information available (Figure 1-2). The right end consists of architectures that require all mass property information to be available *a priori*, a control design approach such as gain scheduling can be used. The left end includes architectures where no mass property information is necessary, a some form of online system identification would be required. Research has been performed in many areas of this spectrum. Three examples of literature are selected and placed on their corresponding location on the spectrum, shown in Figure 1-2. These papers are described in detail in this section to characterize their differences.

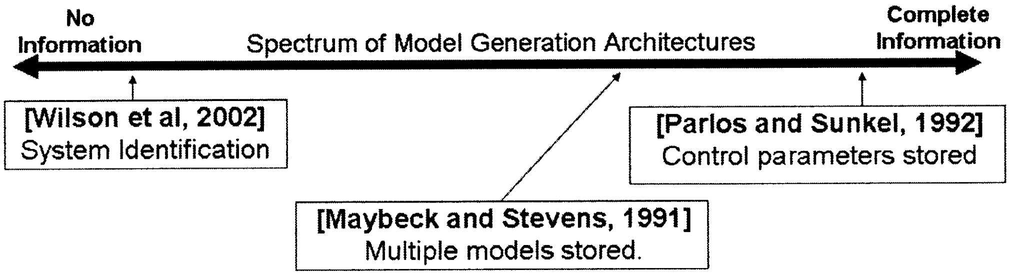

No Information
Spectrum of Model Generation Architectures
Complete Information
[Wilson et al, 2002]
System Identification
[Parlos and Sunkel, 1992]
Control parameters stored
[Maybeck and Stevens, 1991]
Multiple models stored.

Figure 1-2: Spectrum of Model Generation Architectures

Starting with the system identification end, Wilson et al. use a recursive least squares approach to determine the center of mass and inertia of a spacecraft [81] by analyzing gyroscope measurements. Their approach imparts known torques onto the spacecraft and observes the resulting gyroscope measurements. These measurements are fit to an assumption of the model structure to identify parameter values that minimize the mean squared error between the measurement values observed and the measurement values generated based on the assumed dynamics structure. This technique is a key example of the end of the spectrum where no *a priori* information is available. The mass property information is generated purely through system identification, by exciting the system and analyzing its response to derive a model.

Maybeck and Stevens' [50] approach of multiple model adaptive control uses a bank of pre-designed Kalman filters and Command Generator Tracker (feedforward) / Proportional-Integral (feedback) controllers for each of the different anticipated failure states. The final model is a conditional probability-based combination of the individual models using the residuals of the Kalman filter. This method also uses a hierarchical setup of the Kalman filters in order to detect multiple failure states without having to enumerate every combinatorial failure case. Some advantages of this method are that it allows for the seamless transition between configurations, especially in degraded states. Maybeck and Stevens provide a robust way for transitioning between configurations because the models for all possible transitions are continuously processed in the form of a Kalman filter. The continuous processing of the models is necessary when the transition time is unknown, as in Maybeck and Stevens; however, it is excessive for assembly scenarios because it requires running estimators for models that will not be needed until a specific known time in the future. In an assembly mission, the time of transition can be pinpointed to the time of attachment of a module. Thus, the multiple Kalman filters need not be run continuously, but only during the transitions. Once the proper configuration is identified, the filters running incorrect models can be terminated. The execution of several Kalman filters simultaneously is computationally expensive, especially as the number of models stored increases. One additional limitation of this work is that their approach is set-up only to handle configuration changes due to failures. In assembly missions, actuators and sensors can not only be lost due to failure, they can be also be added or change their geometry with respect to the center of mass, as a result of assembly.

Parlos and Sunkel present an approach for attitude control of the Space Station Freedom (precursor to the ISS) under significant mass property variations from the berthing of the Space Shuttle [58]. Their approach uses gain scheduling to update the attitude control law during the berthing scenario. The system is linearized about multiple torque equilibrium attitudes, and the inertias about that equilibrium are used in the calculation of the gains for an LQR controller. They successfully demonstrate attitude control of the spacecraft under inertia variations of approximately 30%.

However, Parlos and Sunkel assume a look-up table exists with the necessary control parameter values. The creation of the look-up table requires complete knowledge of the system, including all possible configurations. This model generation architecture has memory storage and ground development implications associated with creating the look-up table.

The review of the three papers in this section characterizes the differences across the spectrum. The differences in model generation architectures have performance implications on the control system, such as memory storage, computation time, identification time, and fuel consumption. However, model generation architectures have not been designed to exploit the properties of autonomous assembly.

1.3.2 Open Areas in the Literature

Two open areas, identified in the literature review, are addressed in this thesis. The first area is an architecture design problem, while the second is an architecture selection problem.

Model Generation Architecture Design

The first area focuses on the design of a specific model generation architecture to exploit the structure of an autonomous assembly mission. In assembly missions, a potential scenario exists where the module properties are known, but how they attach to the assembler is not known *a priori* in the control system. Three examples of when this situation can arise are (1) due to complex maneuvering where automated obstacle avoidance algorithms impact the state and attitude of the assembler as it attaches to the module, (2) if the module is damaged or in an unknown state when the assembler attaches to it, or (3) if a launch sequence changes and the assembly occurs in steps not previously expected. The design of a model generation architecture to account for these situations allows the control system to be decoupled from the assembly sequence. The decoupling of the assembly sequence from the control system design could lead to benefits such as decreased ground development time and increased versatility of the control system to handle multiple configurations.

An open area exists in model generation literature to account for scenarios where the module properties are known *a priori*, but the configurations and transitions are unknown. Aggregation of the model online based on module properties received during attachment exploits the information available while maintaining a versatile system. Though algorithms for calculating properties of aggregated systems exist, a framework has not been developed to identify the information required to calculate the aggregated model, how that information is transmitted between modules, and how the control system incorporates the aggregated model. The design of this architecture also requires the parameterization of a control system such that it can easily transition between models. This parameterization has not been addressed in literature. Overall, this is an important gap to fill because it serves as a middle ground between having complete information about the system and using resources to identify the properties online.

Model Generation Architecture Selection

The selection of a model generation architecture is important due to the control system performance implications. To make a proper selection, one must address the entire spectrum of architectures. Specifically, one must address how to compare architectures across the spectrum and how to select a single appropriate architecture for a given assembly mission. There are multiple aspects of this problem, such as

- How can one compare model generation architectures equally despite the large implementation differences?
- How does one know if a model generation architecture is appropriate for the assembly mission under consideration?
- What is the impact of the selection of a model generation architecture on the overall assembly performance?

The selection of a model generation architecture requires a good knowledge of: the available architectures in literature; the assumptions associated with each architecture; and the performance of each architecture based on a set of metrics. These aspects have not been addressed in literature. A listing of possible model generation architectures does not currently exist. For a given model generation architecture, an analysis of implementation assumptions does not exist, nor does an analysis of how the assumptions relate to the assembly mission.

Comparisons between model generation architectures are not currently possible because a set of generalized, quantitative metrics does not exist. The current metrics compare either the performance of the control system (e.g. stability, convergence properties) or the assembly mission performance (e.g. assembly time). These metrics do not directly capture the performance of the model generation architecture. Generalized metrics must be developed to capture the use of resources for each type of model generation architecture.

A process for selection is also needed. It is currently difficult to make an informed selection since it is not known what information about the assembly mission is important or how model generation architectures perform compared to each other. An appropriate design is one that is selected using the available *a priori* information specific to that assembly mission and which maximizes system performance specific to the assembly mission. System performance is defined as optimizing the value (ex. versatility), while minimizing the costs (ex. fuel costs). System performance is determined by the optimizing objective functions, which are created based on the metrics that can compare model generation architectures.

The objective of selecting an appropriate model generation architecture is to improve the control system performance, thereby improving overall assembly performance. Therefore, educated implementation of a model generation architecture for a given assembly mission requires knowledge of how the model generation techniques impact control and assembly performance. Literature studies show how an autonomous assembly architecture selection drives mission performance, such as mass and cost. Open areas that exist are how an autonomous assembly architecture drives lower-level control system performance, and how a model generation architecture limits or enables specific assembly architectures. It is important to determine how the model generation architecture affects the overall assembly performance in a quantitative manner to enable proper selection and implementation.

1.4 Thesis Contributions Summary

The objectives of this work relate directly to the open areas specified in Section 1.3.2. The following list gives a summary of the contributions of this thesis. The first three contributions address the design a model generation architecture to balance *a priori* information and on-orbit resource consumption, while the latter contributions relate to selecting a model generation architecture.

The specific thesis contributions are:

- Developed a process for the selection of a model generation architecture based on quantitative metrics that capture computational differences and control system performance implications

- Quantitatively determined the impact of the selection of a model generation architecture on the assembly mission performance

- Developed a framework such that a model for a configuration can be computed online based on assembler and module properties received at the time of attachment

- Developed and implemented a method for parameterizing a control system design such that it can accommodate multiple configurations

- Validated the parameterized control system design on hardware, in a representative space environment

Detailed explanation of the contributions is available in Chapter 9. The major contribution of this work is the **development and validation of tools to address the selection and design of model generation architectures for on-orbit autonomous assembly**.

1.5 Thesis Overview

1.5.1 Definition of Reconfiguration

The main assembly architecture considered in this work is a single assembler that maneuvers individual modules to complete the assembly. The following definitions are used consistently throughout this thesis and have very specific meanings.

- **Architecture**: the structure of the algorithm or concept. Specifically, *model generation architecture* refers to the structure of how the model information is processed, such as inputs, outputs, and component calculation algorithms. *Assembly architecture* refers to the structure of the assembly mission specified through parameters such as the type of assembler used and number of modules.
- **Assembler**: a robotic vehicle that maneuvers modules into their desired locations. The module can be a separate object or the assembler itself (as in self-assembly cases). The assembler is responsible for the control system, in terms guidance, navigation, and control.
- **Module**: an individual, mostly passive, object that is a component of the mission payload requiring assembly. The object is moved by the assembler.
- **Configuration**: a unique physical attachment of assemblers and/or modules
- **Transition**: the change from one configuration to another through the act of attachment or detachment
- **Sequence**: a chronological set of transitions
- **Scenario**: a detailed description of an assembly mission, including assembly architecture parameters, configurations, transitions, as well as mass property information about the assemblers and modules,
- **Case scenario**: a specification of a single scenario used for analysis
- **Metric**: a standard of measurement used to compare different algorithms

- **Model**: the mathematical representation of the dynamics for a configuration used in the control system

1.5.2 Assumptions

The following assumptions are made in this thesis to manage the scope of the research:

- The thesis does not consider the configuration changes due to the aggregation of the assembled structure.
- An assembler has knowledge of its own mass properties.
- Gravitational disturbances, such as J2 effects, are small enough to be neglected during assembly.

1.5.3 Road map

This thesis addresses the implementation of model generation architectures from both a design and a selection perspective. These two components are schematically shown in the thesis road map, Figure 1-3. The spectrum of model generation architectures is shown schematically in the center of the figure.

The design component of the thesis, as depicted by the up/down arrow, considers one particular model generation architecture for an open area, namely where modules are known *a priori*, but transitions and configurations are not known. The design developed in this thesis, called Online Model Calculation, uses property information obtained at the time of attachment to calculate the model for the configuration. Chapters 2 through 5 detail the overall framework, control system parameterization, design, software implementation, and performance on hardware. Chapter 2 provides the theoretical framework for both rigid and flexible modules. The architecture design is implemented on hardware to verify and validate the theoretical design developed. Chapter 3 describes the hardware testbed used in this work. The chapter specifically focuses on the physical hardware components of the Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES) testbed. Chapter 4 describes the

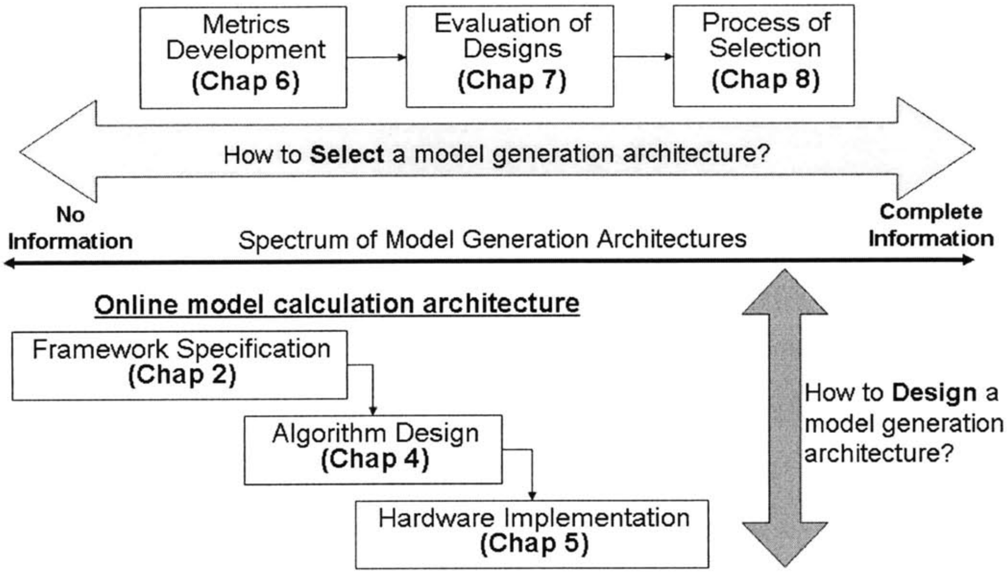

Metrics Development (Chap 6)
Evaluation of Designs (Chap 7)
Process of Selection (Chap 8)
How to Select a model generation architecture?
No Information
Spectrum of Model Generation Architectures
Complete Information
Online model calculation architecture
Framework Specification (Chap 2)
Algorithm Design (Chap 4)
Hardware Implementation (Chap 5)
How to Design a model generation architecture?

Figure 1-3: A schematic road map of the thesis in perspective to the spectrum of reconfigurable control system designs

parameterization of the control system based on the SPHERES testbed. Chapter 5 presents the results from hardware implementation on the SPHERES testbed.

Above the spectrum line, the horizontal line represents the selection component of the thesis. The chapters associated with this area consider architectures from all areas of the spectrum. Chapters 6, 7, and 8 compare, contrast, and evaluate architectures across the spectrum to develop a methodology to select one for a given mission. These architectures are considered from a high-level and are not implemented on a hardware system. Chapter 6 provides the enumeration and classification of model generation architectures, as well as the derivation of a set of metrics that can be used to compare architectures. Chapter 7 presents an assembly simulation that executes the full dynamics and control aspects of the autonomous assembly scenario. This simulation can be used as a tool to calculate metrics for a given assembly scenario for a range of model generation architectures, both at the control system and assembly mission level. Chapter 8 presents a process for the selection of a model generation architecture, along with the underlying assumptions and validated metrics. Chapter 8 also exercises the process on three case scenarios.

The results and contributions of this work are summarized in Chapter 9. This chapter also provides direction for future work in the area of model generation architectures, as well as ways that this work can be incorporated into general autonomous assembly design research.

Chapter 2

Online Model Calculation Framework

The model generation architecture designed in this work is called **Online Model Calculation** and seeks to fill a particular gap identified in literature. This design provides a balance between calculating and storing all properties in advance and identifying the model online. Online Model Calculation calculates the new model at the time of transition, based on the current model, attachment status, and mass property information about the attached module. The attachment and mass property information only needs to be obtained at the time of transition, thus does not need to be stored or known *a priori* by the assembler. It is assumed that the functionality exists to receive the properties by communication. The Online Model Calculation model generation algorithm combines individual property structures of attached objects into the property structure of the combined system.

Consider the an assembler is assigned a module to move. The module communicates its property structure, analogous to a business card. The property structure contains all of the information about that module that is needed by the assembler to develop a module of the aggregate assembler-module system. Properties can include the module's mass, center of mass, inertia tensor, docking port frame, vector from the docking port to the center of mass, and sensor and actuator locations. The assembler combines the module's property structure with the assembler's property structure to build a configuration model that is used to design the control system.

This chapter presents the framework for Online Model Calculation. The following sections give an overview of a reference baseline system, description of the reconfiguration framework, and details on the model calculation algorithm. In particular, the contributions of this chapter are the development of a property structure to capture and transmit the model of a vehicle, and the development of a framework that monitors this structure and performs the necessary updates when the configuration changes. Chapter 4 describes the parameterization of the control algorithms to make them reconfigurable. Chapter 5 details the implementation on the SPHERES hardware testbed and validates the design through the experimental results.

2.1 Approach

The overall approach to designing an Online Model Calculation algorithm is outlined in five major steps. These steps allow for the implementation of Online Model Calculation on any physical system.

1. Identify model framework of the baseline physical system
2. Identify physical properties to be updated and develop a property structure p
3. Develop model calculation algorithm based on the baseline system
4. Parameterize the sections of control software to be a function of the changing physical properties
5. Implement framework using model calculation algorithm and parameterized functions

Figure 2-1 shows the five step approach as a block diagram. The approach consists of two paths, developing the model generation architecture and developing the corresponding control system. Developing the model generation architecture consists of determining the model framework (step 1), defining the property structure for the

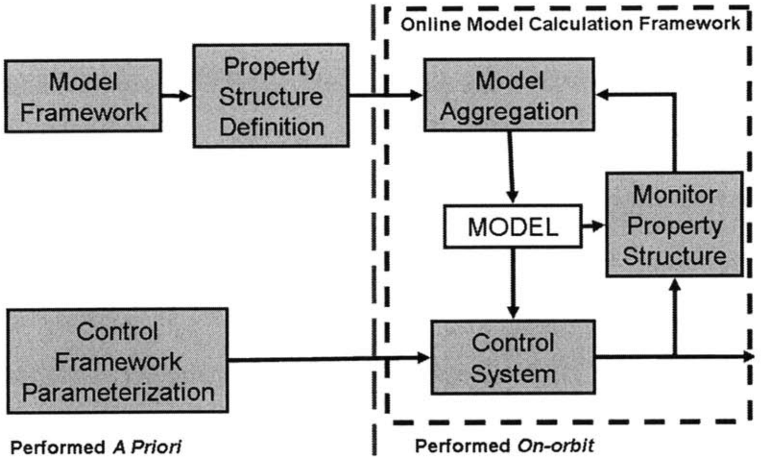

Online Model Calculation Framework
Model Framework
Property Structure Definition
Model Aggregation
MODEL
Monitor Property Structure
Control Framework Parameterization
Control System
Performed *A Priori*
Performed *On-orbit*

Figure 2-1: Online Model Calculation Approach

baseline system (step 2), and developing the model aggregation algorithm (step 3). The development of the corresponding control system is to parameterize it to handle the changing model (step 4). The final step, implementing the framework online, is shown in the black dashed box. Figure 2-1 clearly shows the steps that must be performed on the ground prior to operations and the steps that are performed on-orbit. The remainder of this chapter describes these steps.

2.2 Determining a Model Framework

The first step in the approach to designing a model calculation architecture is to identify the model structure of the baseline physical system. A baseline physical system is used to generalize the possible configurations of assembler and module attachments. This baseline system is used to identify which properties should be updated, as well as providing a standard against which to compare performance.

An example baseline system is described in this work to demonstrate the design process. The generalized configuration assumed in this work is a rigid body assembler, with a module that can be either flexible or rigid. This generalized configuration is depicted in Figure 2-2. Though the example baseline system, represented in Figure 2-2, may not be directly applicable to all systems, it is a good starting point to capture the rigid body properties of any system. This baseline system can be augmented to

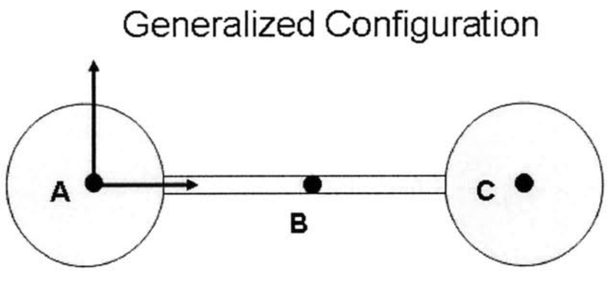
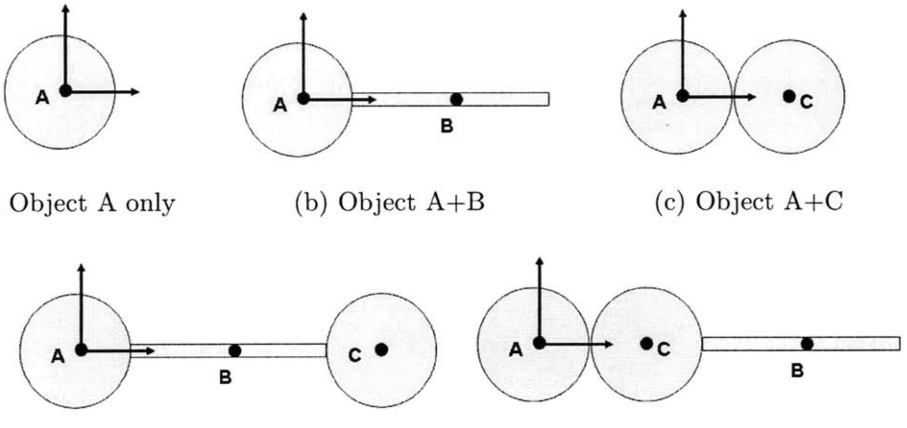


Figure 2-2: Cartoon Representation of Generic System



(a) Object A only

(b) Object A+B

(c) Object A+C

(d) Object A+B+C

(e) Object A+C+B

Figure 2-3: Possible Configurations of Baseline System

represent multiple assembly methods by including objects that capture the additional dynamics.

The generalized configuration has three objects: A, B and C. Object A represents an assembler, while Object B and C represent flexible and rigid modules, respectively. One constraint levied on the configurations is that Object A must always be present, as it is the object providing the maneuvering capability. There are five possible configurations: (1) Object A only; (2) Object A+B; (3) Object A+C; (4) Object A+B+C; and (5) Object A+C+B. Figure 2-3 show a graphical representation of each of the possible configurations.

Configuration 1 (Object A only) represents an assembler by itself. Configuration 2 (Object A+B) represents an assembler with a flexible appendage. An example of this configuration is a tug docked to a solar panel. Configuration 3 (Object A+C) represents an assembler attached to a rigid payload. An example of this configuration is a tug docked to a telescope mirror segment. Configuration 4 (Object A+B+C) can represent an assembler attached to an in-the-loop flexible module, with a rigid end mass on the other end (Object C). Finally, configuration 5 (Object A+C+B) represents an assembler attached to a module with a flexible appendage. The three main configurations considered in this work are Object A, Object A+C, and Object A+B+C. The tools used to design a model generation architecture for these configurations can easily accommodate the remaining configurations.

Object A: Assembler

The assembler has the majority of the actuation and sensing capability. The global coordinate frame is centered at the geometric center of Object A, and is maintained throughout the different configuration changes. Online Model Calculation requires the following information to be known about Object A:

- Reference Point: Geometric Center A
- Body Frame
- Mass, Inertia, Center of Mass with respect to A
- Actuator configuration: Location w.r.t A, Direction, Force, Health, Type
- Sensor configuration: Location/Orientation w.r.t A, Direction, Field of View, Type, Bias/Scale factor
- Attachment locations/directions w.r.t A

Object B: Module - Flexible Component

Object B is the flexible component of the module, defined by its mass, inertia, and stiffness properties. Online Model Calculation requires the following information to be known about Object B:

- Reference Point: Geometric Center B w.r.t A

- Body frame

- Location of attachment w.r.t B and w.r.t A

- Center of Mass w.r.t B

- Mass, Stiffness (characterized by the Object without any attachments), Inertia

Object C: Module - Rigid Component

Object C is the rigid component of the module, defined by its mass, inertia, and attachment location. Online Model Calculation requires the following information to be known about Object C:

- Reference Point: Geometric center C w.r.t A

- Body frame

- Location of attachment w.r.t. B and w.r.t A

- Mass, Inertia, Center of Mass w.r.t C

2.3 Properties to be Updated

The second step in the approach is to identify how the change in mass properties influences the model generation architecture. Figure 2-4 shows a representative control system block diagram, with the software elements that must be updated outlined in a solid line. The physical elements that change due to a configuration change (i.e. plant, measurement matrix (H), and actuator matrix (B)) are outlined in a dashed line in Figure 2-4. Within the estimation loop, the plant model in the estimator and the measurement model, which contain the sensor configuration, must be updated. The actuator model is updated through the control allocation algorithm, which converts the control input into actuator commands. The controllers must also be updated, whether it is the control law itself that is updated or simply the gains.

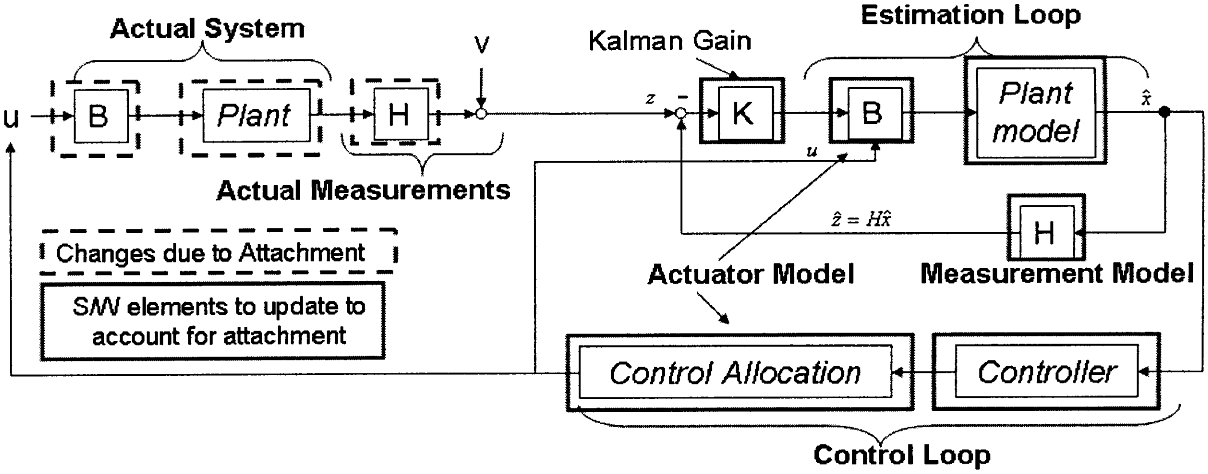


Figure 2-4: Control system block diagram with necessary elements to update highlighted

Each software element can be analyzed to identify the physical properties that are used which require updating. The physical properties constitute three major model components: plant model, actuator model, and measurement model. The plant model contains the physical mass property information of the system: mass, inertia, stiffness, and center of mass location. The actuator model contains the actuator configuration: location, direction, and force of each actuator; actuator health; and the location of the reference point from where the location of the actuator is measured. Similarly, the measurement model contains the sensor configuration: type of sensor, location and direction of sensor, sensor health to know which sensors should be used, and reference point/axis for each sensor.

The property structure p captures the properties that are needed to generate the model. The structure p is passed to each algorithm in order to calculate its respective model. The combination of all of the models mentioned above (plant, actuator, and measurement) constitutes the model of the system. In state space terminology, the model specifies the A (state transition), B (input), C (output), D (feed forward), H (measurement), and K (control gain) matrices. In additional to the models, the property structure also captures the attachment dynamics and status. This allows the system to identify when a configuration change occurs. Thus, p requires the following information:

- Plant: mass, inertia, stiffness, and center of mass
- Actuator: actuator configuration (location, direction, force), actuator health,
- Measurement: sensor configuration (location, direction, type), sensor health, sensor noise properties, reference point/axis
- Attachment: attachment locations, type of attachment, attachment status

The property structure p is specified in Equation 2.1.

$$p = \begin{bmatrix} m & \vec{r}_{cg} & \vec{I} & \\ \vec{r}_{act} & \vec{F}_{act} & \vec{D}_{act} & \vec{R}_{ref-act} \\ \vec{r}_{sensor} & \vec{F}_{sensor} & \vec{D}_{sensor} & \vec{F}_{ref-sensor} \\ \alpha_{location} & \alpha_{type} & \alpha_{status} & \end{bmatrix} \tag{2.1}$$

The components of the structure are

- mass (m),
- center of mass ($\vec{r}_{cg}$) specified in the body frame of the object, from the reference point to the center of mass,
- inertia ($\vec{I}$) inertia as given about the reference point for Object A and about the center of mass for Object B and C,
- actuator locations ($\vec{r}_{act}$) specified in the body frame of the object, from the reference point to the actuator,
- actuator force magnitude ($\vec{F}_{act}$) specified in the body frame of the object,
- actuator directions ($\vec{D}_{act}$) specified in the body frame of the object,
- actuator reference point ($\vec{R}_{ref-act}$) specified from the reference point of the object,
- sensor locations ($\vec{r}_{sensor}$) specified in the body frame of the object, from the reference point to the sensor,

- sensor type ($\vec{F}_{sensor}$) specified in the body frame of the object,
- sensor direction/axis ($\vec{D}_{sensor}$) specified in the body frame of the object,
- sensor reference point ($\vec{R}_{ref-sensor}$) specified from the reference point of the object
- attachment locations ($\vec{r}_{interface}$) specified on Object from the reference point to the interface and on Object B and C from the interface point to the reference point,
- attachment type (α_{type}) which specifies the dynamics and orientation of the attachment,
- and attachment status (α_{status}) which indicates if a module is attached

The property structure includes components that are vectors and matrices, denoted with an arrow in Equation 2.1. The attachment status field can be binary, or can have different values associated with different methods of attachment (ex. docked vs grappled). Nominally, the size of p can be updated dynamically to account for new properties that may arise due to module attachment. The particular elements of p are based on the system under consideration. Equation 2.1 captures the necessary list for the baseline system given in Figure 2-2. For other system, p could be updated to include environment characteristics, stiffness properties, dimensions, and surface area of components of the object. In general, p serves as a profile of the system that can be manipulated in a similar fashion to how the physical system is being manipulated. A detailed description of the components of p is given in Chapter 4, where the control system is parameterized to accept these components to calculate control inputs.

2.4 Model Generation

The third step in the approach is to develop the algorithm to aggregate the model, which allows for the maintenance and generation of the current model. The inputs to this algorithm are the property structure of the assembler (p_{am}), the property

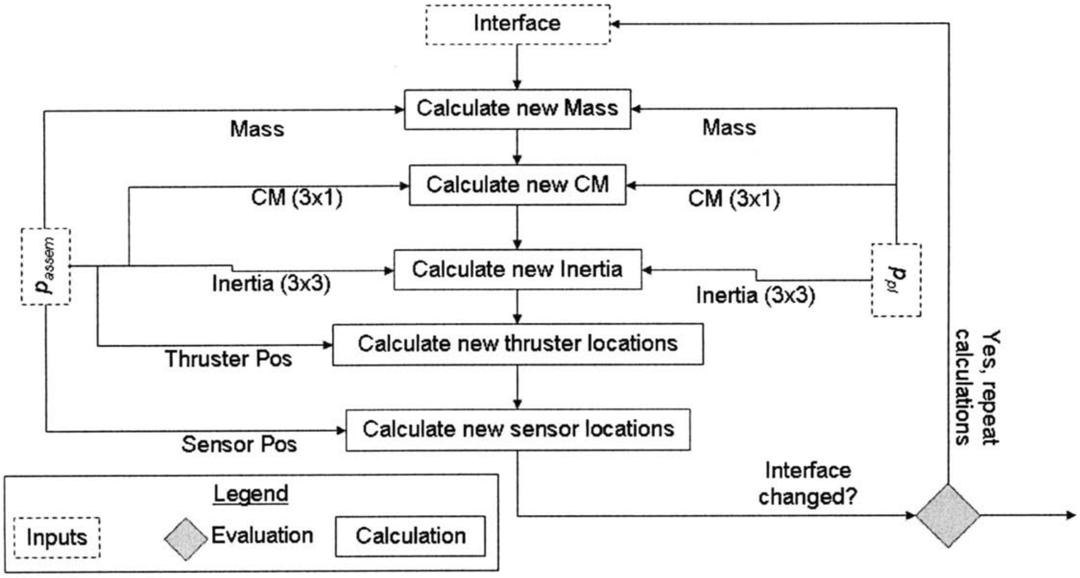


Figure 2-5: Model generation flow diagram example for a passive module

structure of the module (p_{pl}), and the interface where the module has been attached. Upon attachment, two property structure inputs are combined to yield a new property structure that is representative of the combined system. Figure 2-5 is a block diagram that shows the flow of the calculation of the different elements of p.

The first step of model calculation in Figure 2-5 is the calculation of the new mass. The new mass is the sum of the assembler and module masses.

$$m_f = m_{am} + m_{pl} \tag{2.2}$$

After the mass is calculated, the next step in Figure 2-5 is the center of mass computation. The center of mass is recomputed based on the module information, maintaining the assembler's reference frame. The center of mass of the module is determined in the assembler's reference frame by using the attachment information.

$$\vec{r}_{cg} = \frac{1}{m_f}\left[m_{am}\vec{r}_{cg,am} + m_{pl}(\vec{r}_{interface,am} + \Phi\vec{r}_{interface,pl} + \Phi\vec{r}_{cg,pl})\right] \tag{2.3}$$

Equation 2.3 gives the new center of mass with respect to the center of the assembler. The interface location on the module with respect to the center of the module

($\vec{r}_{interface,pl}$) and the interface location with respect to the center of the assembler ($\vec{r}_{interface,am}$) are used to convert the module center of mass to be with respect to the center of the assembler. The variable Φ is the rotation matrix that converts the vectors specified in the module's frame into the assembler's frame. It is determined using the module attachment orientation, which is specified as a quaternion rotation from the module's frame to the attachment port's frame, and the assembler's attachment orientation, which specifies the rotation from the assembler's attachment port to the assembler's frame.

Next, the inertia is similarly obtained by using the parallel axis theorem. First, Equation 2.5 shows the equation to calculate the assembler's inertia about the reference point, where $\vec{d}$ is the vector from the assembler center of mass to the desired reference point O (Equation 2.4). The reference point can be chosen arbitrarily and pre-specified in the model calculation algorithm.

$$\vec{d} = \vec{r}_O - \vec{r}_{cg} \tag{2.4}$$

$$I_{am,O} = \begin{bmatrix} I_{xx,am,cg} + m_{am}\sqrt{d_2^2 + d_3^2} & I_{xy,am,cg} + m_{am}(d_1 d_2) & I_{xz,am,cg} + m_{am}(d_1 d_3) \\ & I_{yy,am,cg} + m_{am}\sqrt{d_1^2 + d_3^2} & I_{yz,am,cg} + m_{am}(d_2 d_3) \\ & & I_{zz,am,cg} + m_{am}\sqrt{d_1^2 + d_2^2} \end{bmatrix} \tag{2.5}$$

Equation 2.6 gives the general form for the parallel axis theorem where:

- O is the reference point on the assembler, that is maintained throughout the configuration changes,
- P is the reference point on the module,
- $I_{am+pl,O}$ is the final aggregated inertia about O,
- $I_{am,O}$ is the inertia of the assembler about O,
- $I_{pl,P}$ is the inertia of the module about P,

- $\vec{b}$ is the shift vector from O to the center of mass of the module,
- c_{pl} is the first moment of inertia, given by $c_{pl} = m_{pl}\vec{r}_{P\Rightarrow cg}$ where $\vec{r}_{P\Rightarrow cg}$ is the vector from P to the module center of mass
- $\vec{1}$ is the 3x3 identity matrix.

$$I_{am+pl,O} = I_{am,O} + I_{pl,P} + m_{pl}\left(b^2\vec{1} - \vec{b}\vec{b}^T\right) + \left(2\vec{b}\bullet\vec{c}_{pl}\vec{1} - \vec{b}\vec{c}_{pl}^T - \vec{c}_{pl}\vec{b}^T\right) \qquad (2.6)$$

Equation 2.6 can be simplif

$$I_{am+pl,O} = I_{am,O} + I_{pl,cg} + m_{pl}\left(b^2\vec{1} - \vec{b}\vec{b}^T\right) \qquad (2.7)$$

The last items in Figure 2-5 to be calculated are the actuator positions and sensor positions. These are maintained with respect to the center of the assembler and only need to be updated if the module includes actuator or sensors. If the module does include actuators or sensors, their locations are converted to be with respect to the center of the assembler by using the interface location on the module and on the assembler.

2.5 Implementation Framework

The approach for integrating Online Model Calculation with a control system design is accomplished through a framework that allows for generic implementation in a real-time system. The difference between this framework and a nominal control system framework is the incorporation of the property structure p as shown in Figure 2-6. The framework has three main components, as shown in Figure 2-6: initialization, control loop, and reconfiguration loop.

The purpose of the initialization component is to initialize p and the baseline assembler model. The assembler starts out with its original model, without any module attachments (Object A configuration). The center block represents the model,

M. This model is maintained throughout the assembly. The model is an input to the control loop to run the parameterized estimator, controller, and control allocation algorithms. The control loop is responsible for providing the actuator commands, and would nominally form the entirety of the control system if no reconfiguration is included.

In the reconfiguration loop, the system is continually checking if p has changed. Specifically, if the attachment status variable changes, then a configuration change has occurred. The property structure of the module is obtained either by communication with the module or from prior knowledge. The new model is calculated by using the property structures from the assembler (p_{am}) and the module (p_{pl}). The central model, M, is updated with this new model, as shown in Figure 2-6. The framework is setup such that the control system always receives the latest model to be used in the estimator, controller, and control allocator. The continuous monitoring of the property structure and maintenance of the model constitutes the reconfiguration loop.

2.6 Conclusions

A five step approach is given in this chapter to develop an Online Model Calculation design. The description of the steps is given via example on a sample assembly scenario. The generalized configuration developed in this work is for the scenario of a rigid body assembler that can attach to either flexible or rigid modules. This general configuration is carried throughout the approach description, including Chapters 4 and 5. Though this work considers a single assembly architecture, the approach for developing this algorithm, as well as the parameterization of the control system algorithms can be extended to other assembly architectures. The assembly architecture assumptions lie in the specification of the baseline system and in the definition of the components of p.

Inherent assumptions of Online Model Calculation that limit its generality are as follows:

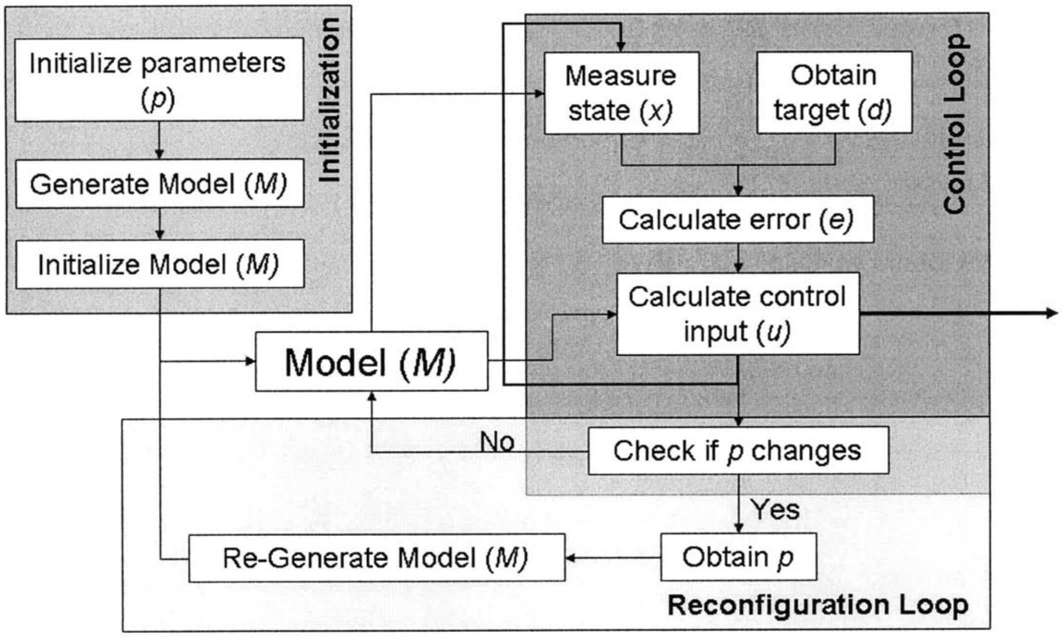


Figure 2-6: Block diagram of reconfiguration framework

- All necessary mass property information is known and pre-coded to be available during the assembly mission. In practice, this requires significant ground development work to properly identify all necessary properties. Also, significant design and implementation may be required to store and transmit the properties.

- The aggregated dynamics can be calculated by relatively simple math manipulation of the module properties. Though this is valid for simple systems, particularly rigid body systems, some assembly architectures may exist with such complex assembler or module dynamics that an aggregated model to required accuracy can not be generated using just the module properties.

- The control system can maintain desired performance in all configurations. If the control system is not design to handle a particular configuration, successful maneuvering will not occur even if the proper model is generated.

- The control system can be expressed as a function of the property structure p. The framework shown in this chapter assumes that the control system can take in the property structure and calculate the necessary parameters. Not all control systems are designed such that parameters, such as control gains, can be written as a function of mass properties. Extending this parameterization to complex control systems which require extensive simulation modeling for parameter selection is difficult.

Despite these limitations, Online Model Calculation provides a critical first step in performing autonomous model generation. The novel contributions of this chapter are the development of the framework for re-calculating the model online and the development of a property structure p. The property structure p captures the necessary mass properties of the system in a form that is easily transmitted between vehicles. This enables the Online Model Calculation framework depicted in Figure 2-6. The consolidation of all mass property information into a single structure p allows for a simplified monitoring and maintenance framework. The control system framework is set-up such that the attachment status variable in the property structure is continuously observed. If the variable changes value, the property structures are used to generate the new model. The parameterization of all of the control system algorithms allows for easy propagation of the model into the control loop. These elements of (1) capturing relevant mass properties, (2) framework to monitor attachment status, and (3) an algorithm to calculate the new model are needed for any autonomous model generation algorithm. Online Model Calculation provides an initial design for autonomous model generation, that can be augmented and integrated with other model generation techniques as more research is performed in this area.

Chapter 3

Hardware Overview

This chapter provides a description of the hardware facilities used for this research. The key hardware and software components are briefly described so that the reader is familiar with the terms and references present throughout the thesis. Hardware testing is used to confirm the simulation results on a realistic system and properly validate the algorithms developed in this work.

The main hardware testbed in this work is the Synchronized Position Hold, Engage, Reorient Experimental Satellites (SPHERES) testbed. SPHERES is a testbed designed to provide a fault-tolerant environment for the development and maturation of control and estimation algorithms for formation flight, docking, autonomy, and reconfiguration. The SPHERES testbed is utilized in three test environments. The first environment is a flat table ground testbed. The second environment is a microgravity flight testbed operated by astronauts on the ISS. Finally, the third environment is a flat floor ground testbed called SWARM (Self-Assembling Wireless Reconfigurable Modules), which includes the attachment of a flexible beam to explore control and estimation issues associated with flexible motion.

3.1 SPHERES - Ground and ISS

The SPHERES testbed consists of self-contained, identical, free-flyer satellites (Figure 3-1). Three satellites are located on the ground, at MIT Space Systems Laboratory,

Figure 3-1: SPHERES satellite

and three satellites are located on the ISS. The ground satellites are used to test algorithms prior to uplink to the ISS. Each SPHERES satellite is a complete spacecraft, equipped with metrology, propulsion, computing, and communication capability. Each satellite has a Velcro panel, which serves as a rudimentary docking system. Additionally, each satellite has an expansion port, which is used on the ground satellites to augment the functionality by attaching external payloads. The components of the SPHERES testbed are the satellites, a laptop computer that serves as a ground station, and five beacons that form the Position and Attitude Determination System (PADS). The five beacons define the working area in which measurements can be made and provide the global reference frame. The following sections describe the aspects of the SPHERES testbed that were directly used in this work. Detailed information about the SPHERES satellites can be found in References [29], [66], and [13]. Only elements specifically used in the work will be discussed. The elements for this work are mass properties, communication, sensors, expansion port, and propulsion. Table 3.1 gives the pertinent mass properties of the SPHERES satellite and Figure 3-2 shows the reference body frame of the satellite.

3.1.1 Communication

The SPHERES communication system was important for this work because some of the algorithms developed in this work require inter-satellite communication. SPHERES uses two communication channels. The SPHERES-to-SPHERES (STS) channel is

Table 3.1: SPHERES satellite mass properties

Mass (kg)			Center of Mass (mm)		
			X	Y	Z
4.3			0.48	-1.19	1.08
Body Frame Orientation			Center w.r.t. Geometric Center(m)		
X	Y	Z	X	Y	Z
Expansion Port	Battery Door	Regulator Knob	0	0	0
Inertia (kg m^2)					
Ixx	Iyy	Izz	Ixy	Ixz	Iyz
2.29E-2	2.42E-2	2.14E-2	9.65E-5	-2.93E-4	-3.11E-5
Velcro Location (m)			Expansion Port Location (m)		
X	Y	Z	X	Y	Z
-0.1023	0	0	0.1023	0	0
Ultrasound Rx			Gyroscopes		
Range	Resolution	Noise	Range	Resolution	Noise
(m)	(mm)	(mm)	(°/s)	(°/s/count)	(°/s/(Hz)$^($1/2))
3	10	2	±83	0.0407	0.05
Accelerometers			Thrusters		
Range	Resolution	Noise	Number	Thrust	Variability
(mg)	(μg/count)	(μg rms)		(kgms$^-$2)	(kgms$^-$2)
±25.6	12.5	7	12	0.11	0.01

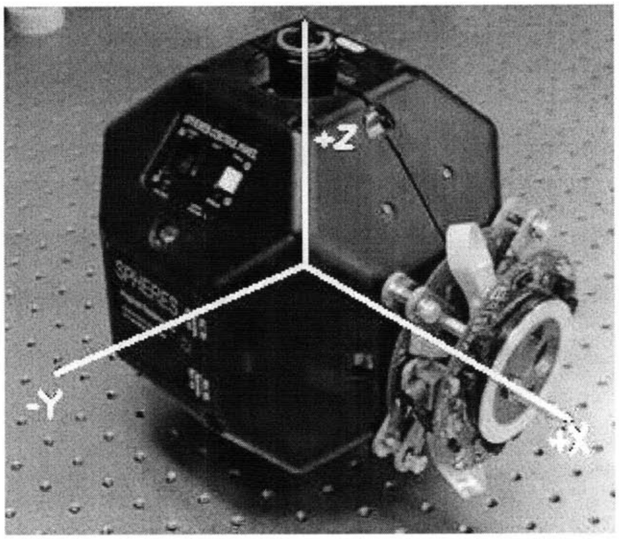


Figure 3-2: SPHERES Body Frame Axes

used for inter-satellite communication, enabling cooperative and coordinated maneuvering between satellites during tests. The SPHERES-to-Laptop (STL) channel is used to transmit data and telemetry to the laptop station. Each channel is on an independent radio frequency, either 868 MHz or 916 MHz. More detail can be found in Reference [66]. The communication bandwidth is limited to a total of 70 packets per second, where each packet is 32 bytes. This must be shared among all of the satellites in operation. The communication delay between sending and receiving is usually on the order of a few milliseconds, but can be up to 200 ms at worst case due to Time Division Multiple Access (TDMA) protocol. The maximum frequency of data transmission of the SPHERES communication system 5 Hz. The amount and frequency of data transmission possible with the SPHERES hardware was a limiting constraint in the implementation of the algorithms. Modifications in implementation were necessary to specifically account for these constraints, which are described in detail in Chapter 5.

3.1.2 Sensors

The sensor configuration of the SPHERES testbed was used heavily in the development and implementation of estimation algorithms. The SPHERES PADS consists of inertial sensors and ultrasound beacons and receivers. Inertial sensors include three single-axis gyroscopes and three single-axis accelerometers, providing three-axis inertial measurements. The ultrasound system consists of 24 ultrasound receivers and one beacon on each satellite. There are five external wall-mountable beacons. Estimation is based on sequenced time-of-flight measurements from the beacons to the receivers to determine a range. A state estimator is then used to provide real-time position, velocity, attitude, and angular rate information for each SPHERES satellite up to 5 Hz (Figure 3-3). More detail can be found in References [29] and [56].

The accelerometers are used in this work as a velocity truth measure. The test volume, framed by the location of the beacons, is important because it constrains the maneuvering space. Thus, the waypoints and target locations for the experiments described in Chapter 5 were constrained to be within the test volume defined by the

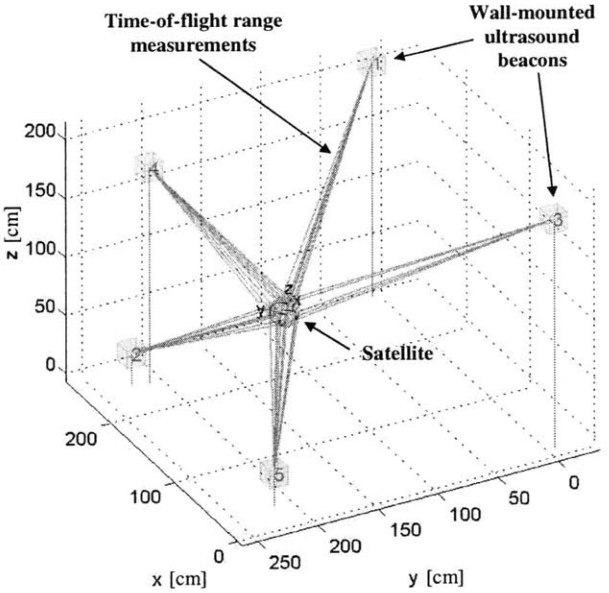
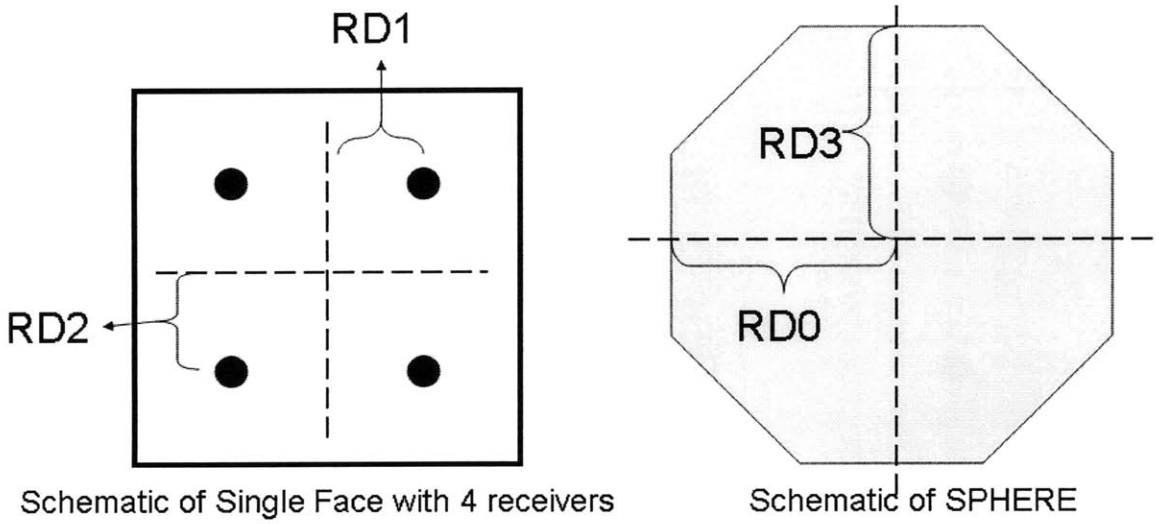


Figure 3-3: Geometry of PADS. Beacon locations define operational volume. Time of flight ranging used from at least four beacons to get full state determination.

location of the five beacons.

The ultrasound sensor placement is specified here because this geometry was used to develop a reconfigurable estimation scheme that combines ultrasound measurements from multiple satellites (described in Chapter 4). The physical locations of the ultrasound receivers are given by Table 3.2. Ultrasound receivers are grouped in sets of four located on faces around the satellite. The faces are identified by their location in the body frame. Due to the geometry of the placement of the face (along one axis and centered in the other two axes), faces can be identified by a single body axis specification (e.g. +X, -Y). RD0 equals 0.1023 m, which is the distance from the center of the SPHEREs satellite to the face on which the receivers are located. RD1 equals 0.0392 m, which is half the distance between receivers on a single face, in the horizontal direction. RD2 equals 0.0394 m is the half of the distance between receivers on a single face in the vertical direction. Finally, RD3 equals 0.1026 m which is the distance from the center of the SPHERES satellite to the +Z face. Figure 3-4 shows the receiver distances schematically. Figure 3-2 shows the body frame of the



Schematic of Single Face with 4 receivers

Schematic of SPHERE

Figure 3-4: Geometry of Receiver locations on a face

satellite, as a reference. The horizontal and vertical distances specified in Chapter 3-4 correspond to the body frame depending on the face the receivers are located on. For example, if receivers are located on the +X face, the vertical direction is the Z and the horizontal distance is the Y direction.

3.1.3 Expansion Port

The SPHERES expansion port provides the mechanical mounting, serial communication, and a power interface to augment the current functionality of the satellite. Figure 3-5 shows the expansion port uncovered on the SPHERES satellite. The expansion port connector enables the payload to interface with the satellite's main processor and metrology sensors. Thus, the payload is able to acquire and transmit information back and forth through the interface. This interface is used extensively on the ground with a number of payloads that have been designed, such as a tether mechanism [15], optical pointing payload [48], and a universal docking port [62]. Two particular payloads that are used heavily in this work are the Universal Docking Port (UDP) (Figure 3-7) and the SWARM propulsion module. The SWARM propulsion module is described in more detail in Section 3.2.

The UDP is a genderless docking port that enables a rigid attachment to another

Table 3.2: Ultrasound Receiver Locations with respect to the geometric center of the SPHERES satellite

Face	Rx Number	X	Y	Z
+X	1	RD0	-RD1	RD2
+X	2	RD0	RD1	RD2
+X	3	RD0	RD1	-RD2
+X	4	RD0	-RD1	-RD2
+Y	5	RD2	RD0	-RD1
+Y	6	RD2	RD0	RD1
+Y	7	-RD2	RD0	RD1
+Y	8	-RD2	RD0	-RD1
+Z	9	-RD1	RD2	RD3
+Z	10	RD1	RD2	RD3
+Z	11	RD1	-RD2	RD3
+Z	12	-RD1	-RD2	RD3
-X	13	-RD0	RD1	-RD2
-X	14	-RD0	RD1	RD2
-X	15	-RD0	-RD1	RD2
-X	16	-RD0	-RD1	-RD2
-Y	17	-RD2	-RD0	RD1
-Y	18	RD2	-RD0	RD1
-Y	19	RD2	-RD0	-RD1
-Y	20	-RD2	-RD0	-RD1
-Z	21	RD1	-RD2	-RD0
-Z	22	RD1	RD2	-RD0
-Z	23	-RD1	RD2	-RD0
-Z	24	-RD1	-RD2	-RD0

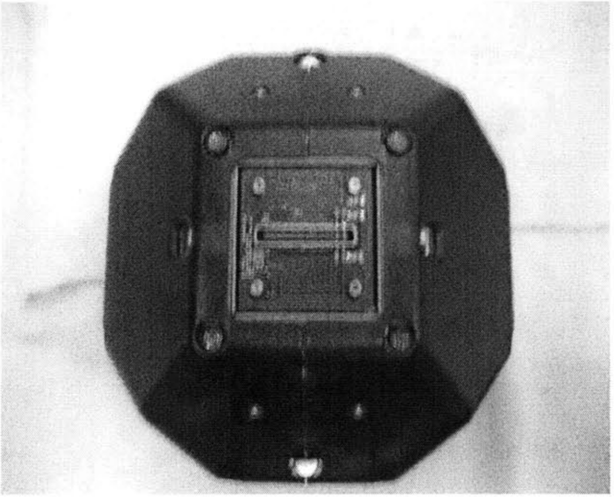


Figure 3-5: Expansion port on the SPHERES satellite

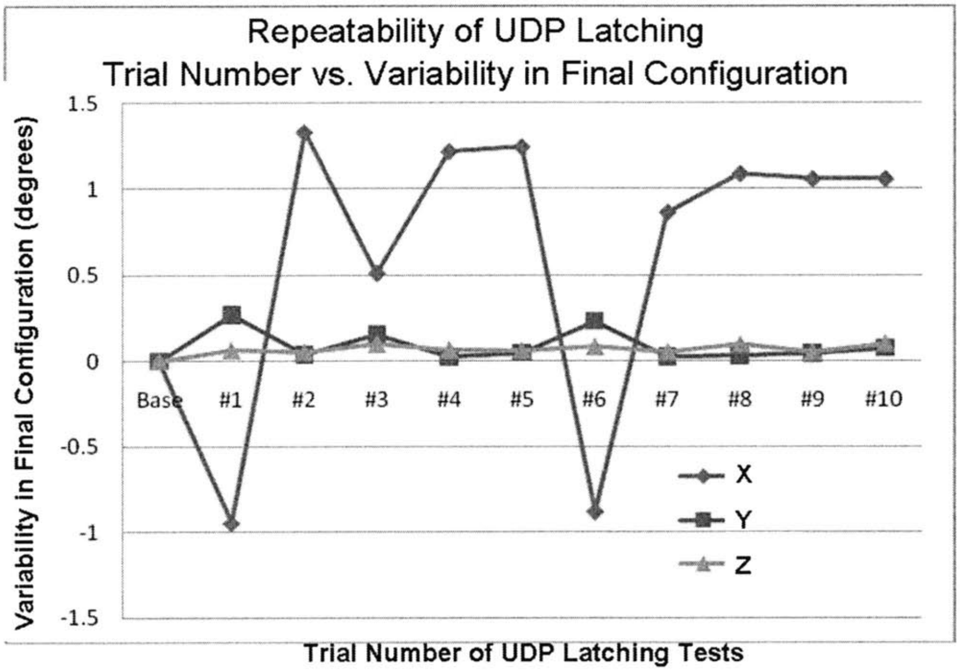


Figure 3-6: Results for UDP accuracy characterization

UDP. Each UDP has a metrology ring, with three beacons and three receivers. These sensors are used in addition to the SPHERES satellite's sensors in a estimator that provides the state of the satellite relative to other UDPs. The mechanical design of the UDP is a pin-hole mechanism. The pin enters the hole and trips a photo sensor, which initiates the two counter-rotating cams that close to grip the pin. The UDP also has an electromagnet to aid in smooth docking and undocking. Accuracy testing performed on the UDP revealed a precision of approximately 2° about the X axis, 0.267° about the Y axis, and 0.1° about the Z axis, as shown in Figure 3-6 [59]. More detail can be found in Reference [62].

The UDP is important because it provides a rigid, precise attachment mechanism compared to the Velcro that is present on the SPHERES satellites. The UDP was used extensively for the tests performed in the SWARM environment, described in Chapter 5. The UDP is a critical component for assembly, particularly future ISS tests, because unlike Velcro, it allows for both docking and undocking.

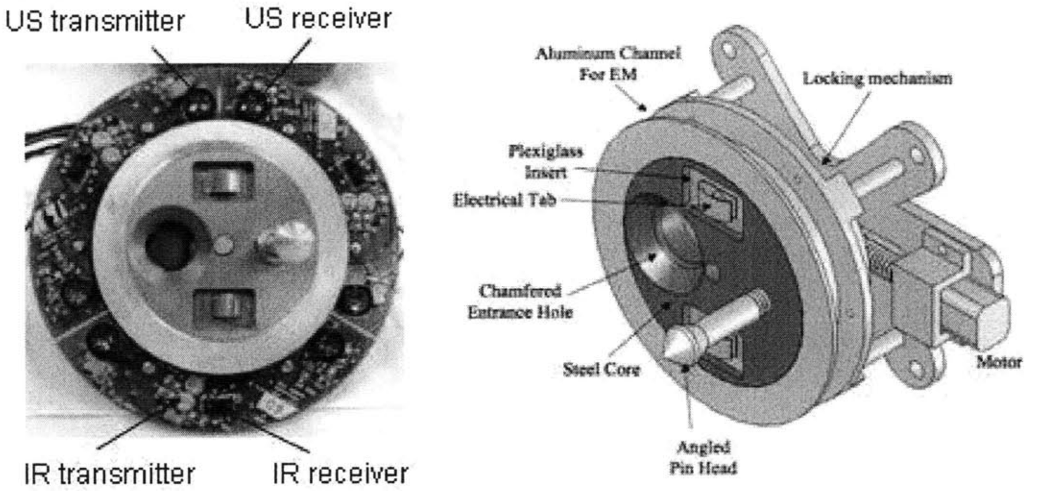


Figure 3-7: Universal Docking Port. Left: Picture of hardware, with sensors labeled. Right: CAD drawing of UDP

3.1.4 Propulsion

Each satellite has the ability to maneuver in six degrees of freedom (DOF) propelled by a cold-gas thruster system fueled by CO_2. The satellite has a single propellant tank with 170g of liquid CO_2. This fuel amount restricts the maneuvering time of the tests performed in Chapter 5. In particular, the assembly demonstration tests used nearly a full tank for a single run. As assembly testing expands, the fuel available will become a limiting constraint on the hardware.

Twelve thrusters are symmetrically positioned around the satellite to provide control about all three axes, enabling simultaneous attitude and translation control. The layout of the thrusters are shown in Figure 3-8. The numbered arrows in Figure 3-8 indicate the direction in which the gaseous CO_2 is expelled by that thruster. The thruster geometry is specified by the direction of the force and torque produced by that thruster. For a single satellite, the geometry is given by Table 3.3. For multiple SPHERES satellites attached to each other, the geometry is determined depending on how the satellites are attached. Table 3.4 gives the thruster geometry for two satellites connected by their -X (velcro) face. In this configuration, one satellite is designated the assembler. This satellite is responsible for determining the thruster configurations for both satellites. Thus, the thruster configuration is specified in the

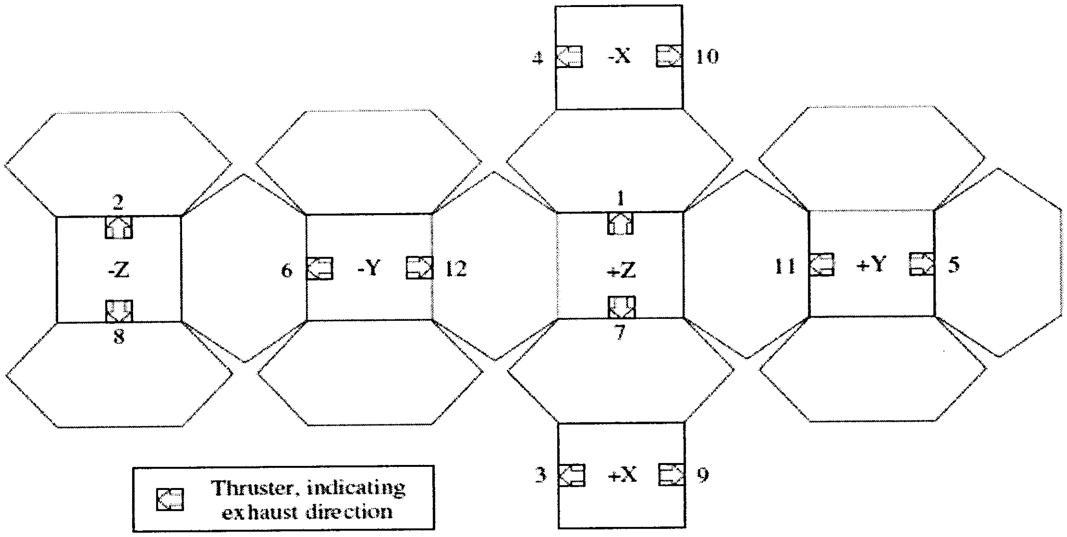


Figure 3-8: Two-dimensional exploded view of thrusters one the SPHERES satellites [29]

assembler's frame. The thruster configurations specified in Table 3.3 and 3.4 are used as input to the control allocation algorithms described in Chapters 4 and 5. These tables can be used to determine the proper thruster to fire to generate the desired force or torque. For example, from Table 3.3, to generate a pure force in the +X direction, one would fire thrusters 1 and 2. To generate a pure torque about the +Z axis, one would fire thrusters 3 and 10.

The thruster geometry has significant implications for the control system, particularly when multiple satellites are attached. The SPHERES thruster geometry is the minimal set required for six DOF motion. When two satellites are connected, on any primary face, two thrusters on that face become blocked by the other satellite. The plume impingement effects cause the thrusters on the attachment face to become ineffective. The result is that the axis of attachment is an uncontrollable and unstable direction, if only a single SPHERES satellite's thrusters are used. Thus, in this thesis, when two SPHERES satellites are connected, thrusters on both satellites are used so that the system is still controllable.

Table 3.3: Mapping of the SPHERES thrusters to Forces/Torques on the satellite

Thruster	**Force**			**Torque**		
Number	x	y	z	x	y	z
1	+1				+1	
2	+1				-1	
3		+1				+1
4		+1				-1
5			+1	+1		
6			+1	-1		
7	-1				-1	
8	-1				+1	
9		-1				-1
10		-1				+1
11			-1	-1		
12			-1	+1		

3.1.5 Software

SPHERES software consists of an operating system (*SPHERESCore*) and additional user-selectable library functions. *SPHERESCore* is responsible for handling interrupts and interfacing with the hardware. The library functions provides guest scientists with the ability to use pre-defined utility functions to expedite programming testing [24]. The coding language used on the hardware is C, while code for the simulation can be programmed in C or MATLAB. It is through this operating system framework that the reconfigurable algorithms developed in this work are implemented. In addition to updating *SPHERESCore* functionality, this research also makes use of some of the library functions to support testing. Examples of the library functions used include math utilities, data compression algorithms, and basic controllers.

Two updates were made to *SPHERESCore*. The first update was to develop a reconfigurable estimator that could handle multiple configurations. This estimator is described in detail in Chapter 4. The second update was to update the satellite physical properties file. This file loads the satellite mass and thruster properties, such as mass, inertia, center of mass, thruster strength, thruster direction, and thruster location. The values stored in Flash memory (specific to each satellite) are automatically loaded when a satellite is reset. The satellite-specific flash values are maintained in

Table 3.4: Thruster directions of two SPHERES satellites attached via their -X faces, as Forces/Torques matrix, in Assembler satellite's frame. Thrusters on the Assembler are denoted with an 'A', while thrusters on the Payload are denoted with a 'B'

Thruster	**Force**			**Torque**		
Number	x	y	z	x	y	z
1/1A	+1				+1	
2/2A	+1				-1	
3/3A		+1				+1
4/4A		+1				+1
5/5A			+1	+1		
6/6A			+1	-1		
7/7A	-1				-1	
8/8A	-1				+1	
9/9A		-1				-1
10/10A		-1				-1
11/11A			-1	-1		
12/12A			-1	+1		
13/1B	+1				+1	
14/2B	+1				-1	
15/3B		+1				+1
16/4B		+1				+1
17/5B			+1	+1		
18/6B			+1	-1		
19/7B	-1				-1	
20/8B	-1				+1	
21/9B		-1				-1
22/10B		-1				-1
23/11B			-1	-1		
24/12B			-1	+1		

the file *spheres-physical-properties.c.* The mass and thruster properties are assigned as working variables, local to this file, and accessed by 'get' and 'set' functions. The existing functionality only includes values for a single SPHERES satellite. Through the research presented in this thesis, this file was updated to include multiple configurations. The functions in this file were used as the primary interface to access the mass properties of the satellites, which is a critical part in the reconfiguration framework described in Chapter 2.

3.2 SWARM

The third testing environment is the SWARM environment. SWARM was developed as a ground extension to the SPHERES testbed. The purpose was to enhance the capability of the nominal SPHERES testbed to fully test algorithms related to autonomous assembly, on the ground. The SWARM environment was important to this work because of the flexible beam attachment. The presence of a flexible beam module allowed the algorithms developed in this work to be validated not only for a rigid module, but for a flexible module as well. This is important because it expands the generality of this research, demonstrating that the algorithms developed can accommodate varying stiffness properties as well as mass properties. The SWARM testbed consists of the SPHERES satellites, UDPs, a structural base, a propulsion module, and a flexible beam. The SPHERES satellites and UDPs are mentioned in a previous section of this chapter. The section describes the remaining three components.

3.2.1 Structural Base

The structural node is a necessary structural augmentation to the SPHERES satellite to enable the satellite to attach to more than one module. It is a plate with mounting posts attached. Figure 3-9 shows a structural base with a SPHERES satellite attached. The SPHERES satellite sits in the center of the node, resting on a square plate. On all four sides, there are connections where a mounting post can be attached. Figure 3-9 shows two mounting posts attached, where the right post has

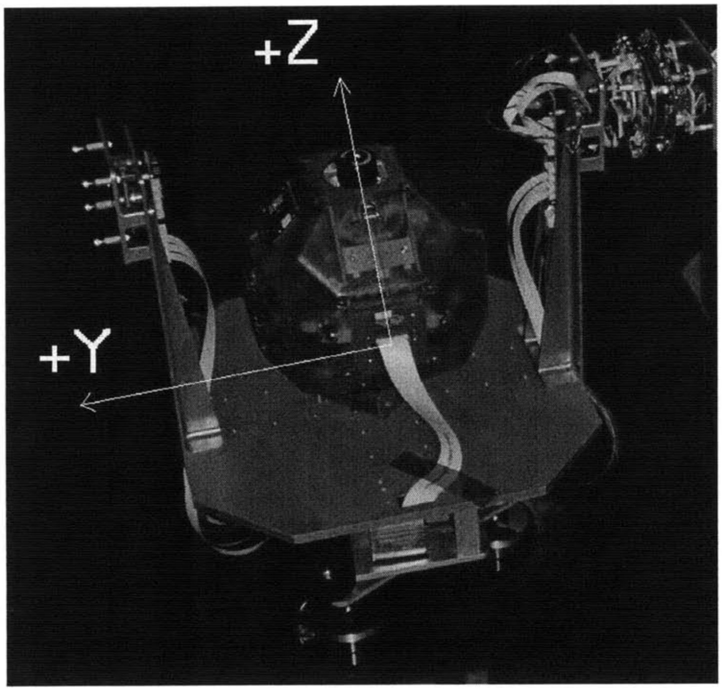


Figure 3-9: SPHERES satellite on a SWARM structural base

a UDP attached to the top of the post. The structural base gives the SPHERES satellite the ability to have multiple docking ports attached. This allows for more seamless assembly, as the SPHERES satellite can now act as either the assembler or module.

3.2.2 Propulsion Module

Attached to the bottom of the plate of the structural base is the SWARM propulsion module. The propulsion module is connected to the satellite via the expansion port, described previously in this chapter. The SPHERES thrusters were designed to provide enough force to maneuver in microgravity. However, the force of the SPHERES thrusters are insufficient to maneuver on the ground when modules are attached. Therefore, the SWARM propulsion module was designed to provide additional force in the horizontal plane during ground testing. Figure 3-10 shows the placement of the propulsion module with respect to the structural plate. The placement of the thrusters below the SPHERES satellite ensures that no plume impingement effects occur. The propulsion module contains sixteen thrusters, four in each direction, +X,

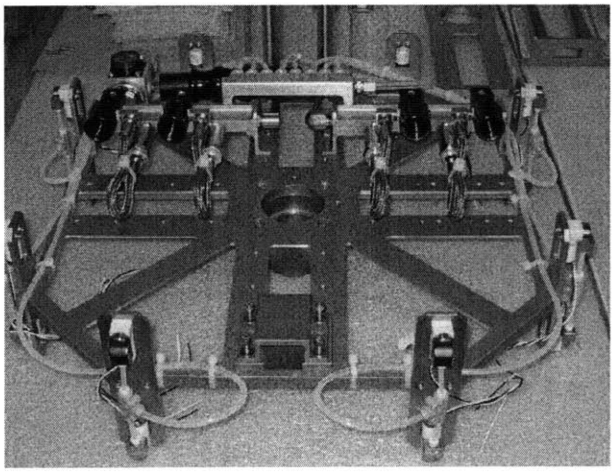


Figure 3-10: SWARM propulsion module, attached to the structural node

+Y, -X, and -Y. Table 3.5 gives the thruster configuration for the propulsion module. Four thrusters are generally fired simultaneously, with an effective thruster force on the order of 1 N.

The SWARM propulsion module is important to the research in this thesis for two reasons. First, it enables maneuverability in the ground environment. Without this additional propulsion module, the SWARM tests described in Chapter 5 would not have had enough thruster authority to maneuver successfully. Second, the method of accessing and commanding the propulsion module thrusters, which are external to the SPHERES satellite, demonstrate the parameterized control allocation developed in Chapter 4. Similar to the SPHERES satellite thruster configuration, Table 3.5 is also an input into the control allocation algorithm developed in this work.

3.2.3 Flexible Beam

One of the main objectives of the SWARM testbed is to enable control and assembly of flexible structures. Thus, the flexible beam was designed with two main requirements. The first requirement was that the deflection of the beam when excited by a nominal SPHERES firing pattern be larger than 1 cm, which is the accuracy necessary for docking using the UDPs. The second requirement was that the first mode frequency should be less than 0.5 Hz. This requirement arose because the maximum control

Table 3.5: SWARM Propulsion Module Thruster Force Directions

Thruster	**Force**			**Torque**		
Number	x	y	z	x	y	z
1		-1				-1
2		-1				+1
3	+1					-1
4	+1					+1
5		+1				-1
6		+1				+1
7	-1					-1
8	-1					+1
9		-1				-1
10		-1				+1
11	+1					-1
12	+1					+1
13		+1				-1
14		+1				+1
15	-1					-1
16	-1					+1

rate of the SPHERES is roughly 1 Hz, thus, using the Nyquist criterion, the nominal observable frequency of the beam is 0.5 Hz. The flexible beam in SWARM is a four link beam, consisting of of four Aluminum links, with torsional springs made from steel shims connecting the links. Figure 3-11 shows a schematic of the physical beam with dimensions, while the hardware setup is shown in Figure 3-12. This beam configuration was experimentally tested to have a first modal frequency of 0.235 Hz. A LED is attached to the free end of the beam and is used to estimate the beam deflection, using a camera that is mounted to the SPHERES satellite.

This thesis makes use of previous work performed using the SWARM environment. Katz [42] developed an estimator to track the deflection of the beam using the LED/Camera system, which is used as developed in this work. The model of the beam used in this thesis is a simplified representation which models the beam as a single flexible link. The beam is then represented with a approximated length and stiffness value. For more information on the derivation of the beam dynamics, please refer to Katz [42].

Katz also developed 2D adaptive controllers to maneuver with the flexible beam.

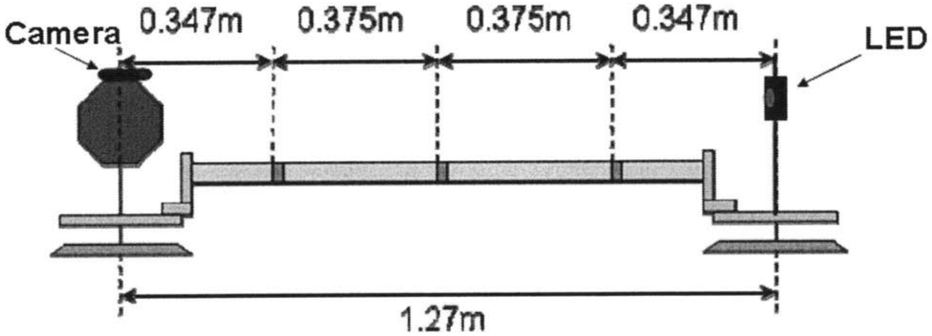


Figure 3-11: SWARM Beam Configuration

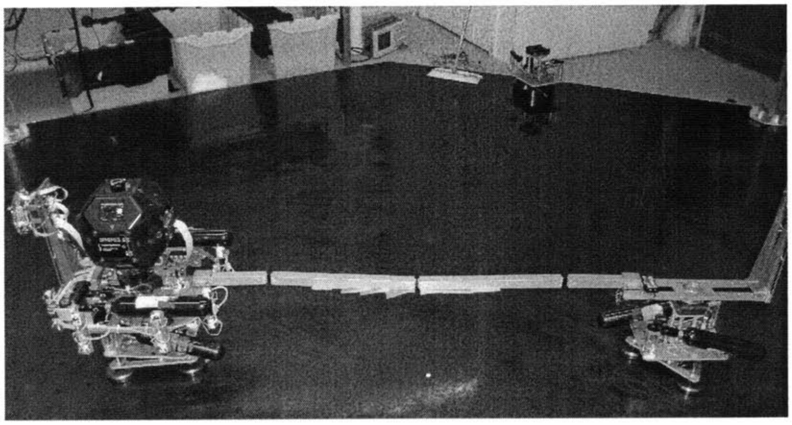


Figure 3-12: SWARM with Beam attached

One of the main adaptive controllers used in this work was developed by Katz as a 2D controller and tested only in simulation. This work upgrades the controller to be 3D and conducts tests in simulation and hardware.

3.3 Summary

The SPHERES testbed was the hardware and software baseline for this study. Not only were the algorithms developed implemented on SPHERES, but the model of the SPHERES satellites is used throughout this work as a representative assembler. The content of this chapter provided an overview of the main aspects of the SPHERES testbed as they relate to this work. To get a detailed understanding of the aspects mentioned or how they interact, please consult the references specified. Chapters 4 and 5 provide more detail on the software algorithms used on SPHERES, with implicit reference to the hardware components described in this section.

Chapter 4

Online Model Calculation: Control System Parameterization

Chapter 2 provides the framework and implementation approach for Online Model Calculation. Demonstration of Online Model Calculation requires implementation of a control system to take in the model and calculate the corresponding control inputs. To accomplish the seamless propagation of the model into the control system, the control algorithms are parameterized to accept the property vector p. Using the property vector as an input, the control algorithms can generate the necessary control parameters specific to the model.

The control system considered in this work is a simple design geared toward demonstrating the parameterization feasibility. The baseline control system used is based on the SPHERES hardware testbed described in Chapter 3. Three main control system algorithms are discussed in the following sections: estimator, controller, and control allocation algorithms.

4.1 Estimator

The research in this thesis considers an Extended Kalman Filter (EKF) as the baseline estimation algorithm for the assembler. This type of estimator is routinely used in industry due to its good performance and ease of implementation. Four aspects of the estimator are considered for parameterization: state propagation, actuator propagation, sensor measurement incorporation, and initialization. This section begins with a brief review of the EKF algorithm, then addresses each of the four aspects of the estimator.

4.1.1 EKF Review

Consider a generic second order system given by Equation 4.1, where the dynamics vector q represents the position and attitude of the assembler, J is the mass/inertia matrix, D is the damping or friction matrix, and G is the stiffness matrix. A state vector x can be written as $x = [\ q \quad \dot{q}\ ]^T$. The state space representation of this model is given by Equation 4.2. The state transition matrix is given by A and B is the actuator matrix.

$$J(q,\dot{q})\ddot{q} + D(q)\dot{q} + G(q) = u \tag{4.1}$$

$$x_{t+1} = A(x_t) + B(x_t, u_t) \tag{4.2}$$

The EKF is a two step process of first predicting the estimate through propagation, then updating the estimate based on measurements. The Predict step propagates the estimated state forward from time $t-1$ to time t. The Update step updates the predicted state at time t with measurements from time t. This is depicted in Equations 4.3 and 4.4, with the covariance (P), process noise (Q), Kalman gain (K), measurement matrix (H and h), measurement (z), and sensor noise (R). The measurement matrix h is used to convert the state into an estimated measurement, while H is used to propagate the covariance.

$$\begin{array}{lr} \textit{Predict} & \begin{array}{r} \hat{x}_{t|t-1} = A(\hat{x}_{t-1|t-1}) + B(\hat{x}_{t-1|t-1}, u_t) \\ P_{t|t-1} = AP_{t-1|t-1}A^T + Q_t \end{array} \end{array} \tag{4.3}$$

$$\textit{Update} \quad \begin{aligned} \hat{x}_{t|t} &= \hat{x}_{t|t-1} + K_t y_t \\ K_t &= P_{t|t-1} H_t^T \left(H_t P_{t|t-1} H_t^T + R_t \right)^{-1} \\ y_t &= z_t - h(\hat{x}_{t|t-1}) \\ P_{t|t} &= (I - K_t H_t) P_{t|t-1} \end{aligned} \tag{4.4}$$

A change in the model affects both Predict and Update steps in the estimator. Four aspects of the estimator are parameterized with respect to the property vector p. Parameterization to account for changes to the A matrix are considered in Section 4.1.2, while the parameterization to account for changes to the B matrix are considered in Section 4.1.3. Parameterization to account for changes to sensor configuration, such as number or type of receivers, is addressed in Section 4.1.4. Finally, Section 4.1.5 addresses initialization issues when estimator switches between models.

4.1.2 State Propagation

State propagation is a critical part of the estimation framework. Therefore, to maintain the estimator performance, it is important to parameterize the state propagator to seamlessly accept the model.

The structure of the A and B matrices from the state space representation in Equation 4.2 stays the same for any second order system with respect to the dynamics matrices given in Equation 4.1. A is the state transition matrix, which describes how the state changes over time under no external forces. The state propagation must be updated if there are changes to x or A. To account for a changing A matrix, the J, D, and G matrices can be parameterized such that they are properly calculated for any combination of the objects in the baseline physical system (Figure 2-2).

The steps of the derivation of the parameterized dynamics matrices are based on the Euler-Lagrange method.

1. Determine complete state vector as a function of the presence of an Object

2. Determine the Lagrangian L, where $L = T - V$

   (a) Calculate the kinetic energy (T)

(b) Calculate the potential energy (V)

3. Extract J by $J = \frac{d}{d\dot{q}}\frac{dT}{d\dot{q}}$

4. Extract D using J

5. Extract G by $G = \frac{dV}{dq}$

6. Determine A matrix based on derived forms of J, D, and G

Determine state vector

Equation 4.5 gives the dynamics vector q, which includes the three element position vector of Object A ($\vec{r}_A$), the attitude of Object A (θ). The variable $\vec{\delta}$ is a three element vector that captures the deflection of Object B in the x, y, and z directions. The deflection is only included in the state vector if Object B is attached. The variable φ_B is boolean variable that indicate whether Object B is attached ($\varphi_B = 1$) or not ($\varphi_B = 0$). The attitude of Object A (θ) can be represented as a three element vector with the x, y, and z Euler angles, or as a four element quaternions vector. In this thesis, tests that are run in 2D use Euler angles for simplicity, while 3D operations, such as on the ISS, use quaternions.

$$q = \begin{bmatrix} \vec{r}_A & \vec{\theta}_A & \varphi_B \vec{\delta} \end{bmatrix}^T \tag{4.5}$$

These components included in q are selected based on what properties must be monitored when the Objects are rigidly attached. The position and attitude of Object A are necessary in all configurations. The inclusion of the deflection of Object B is necessary because the deflection impacts the maneuvering of Object A, when Object B is attached, and it cannot be computed based on Object A dynamics. The position of Object C does not need to be included in the state vector because it's state can be completely defined by Object A's state, and Object B's deflection if in Object A+B+C configuration. Even though Object C's position is not included in the state vector, its contribution to the dynamics are included in the calculation of the dynamics matrices.

The corresponding state vector $x = \begin{bmatrix} q & \dot{q} \end{bmatrix}$, where q is given by Equation 4.6.

$$q = \begin{bmatrix} \vec{r}_A & \vec{\theta}_A & \varphi_B \vec{\delta} & \dot{\vec{r}}_A & \dot{\vec{\theta}}_A & \varphi_B \dot{\vec{\delta}} \end{bmatrix}^T \tag{4.6}$$

Calculate Lagrangian and dynamics matrices J, D, and G

To calculate the dynamics matrices (J, D, and G), one must (1) calculate Object positions and velocities; (2) use the positions and velocities to calculate kinetic and potential energy; (3) form the Lagrangian using the kinetic and potential energy; and (4) calculate the dynamics matrices using the Lagrangian.

Equation 4.7 defines the global position of each object. The variables of Equation 4.7 are defined as follows:

- The variables $\vec{r}_A$, $\vec{r}_B$, and $\vec{r}_C$ are the three element position vectors for Object A, B, and C respectively. The variables $\vec{r}_B$ and $\vec{r}_C$ are written in terms of $\vec{r}_A$ because they are rigidly attached and Object A is the central reference frame.

- The variables φ_B and φ_C are boolean variable that indicate whether Object B or C is attached.

- The matrix ϑ specifies the three dimensional rotation matrix based on the input angle vector (e.g. $\vartheta(\delta)$).

- The variable $\vec{r}_{\alpha A}$ is the distance from the center of A to the attachment location where Object B is attached.

- The variable $\vec{r}_{cgB}$ is the distance from the attachment location on B to the center of mass of B.

- The variable $\vec{r}_{cgC}$ is the distance from the attachment location on C to the center of mass of C.

- The notation $\dot{\theta}^{\times}$ indicates the derivative of the angle arranged to form the cross-product when multipled to another vector, $\dot{\theta}^{\times} = \begin{bmatrix} 0 & -\dot{\theta}_Z & \dot{\theta}_Y \\ \dot{\theta}_Z & 0 & -\dot{\theta}_X \\ -\dot{\theta}_Y & \dot{\theta}_X & 0 \end{bmatrix}$.

$$\begin{aligned} \vec{r}_A &= \begin{bmatrix} x & y & z \end{bmatrix}^T \\ \vec{r}_B &= \vec{r}_A + \varphi_B \vartheta(\vec{\theta}) \left[\vec{r}_{\alpha A} + \vartheta(\vec{\delta})\vec{r}_{cgB}\right] \\ \vec{r}_C &= \vec{r}_A + \varphi_C \vartheta(\vec{\theta}) \left[\vec{r}_{\alpha A} + \vec{r}_{cgC} + \varphi_B \vartheta(\vec{\delta}) L_B\right] \end{aligned} \tag{4.7}$$

Equation 4.8 defines the global velocity of each object, by taking the derivative of Equation 4.7. The variables $\dot{\vec{r}}_A$, $\dot{\vec{r}}_B$, and $\dot{\vec{r}}_C$ are the three element velocity vectors for Object A, B, and C respectively.

$$\begin{aligned} \dot{\vec{r}}_A &= \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \\ \dot{\vec{r}}_B &= \dot{\vec{r}}_A + \varphi_B \theta^{\times} \vartheta(\vec{\theta}) \left[\vec{r}_{\alpha A} + \vartheta(\vec{\delta})\vec{r}_{cgB}\right] + \varphi_B \vartheta(\vec{\theta}) \left[\hat{\delta}\vartheta(\vec{\delta})\vec{r}_{cgB}\right] \\ \dot{\vec{r}}_C &= \dot{\vec{r}}_A + \varphi_C \dot{\theta}^{\times} \vartheta(\vec{\theta}) \left[\vec{r}_{\alpha A} + \vec{r}_{cgC} + \varphi_B \vartheta(\vec{\delta}) L_B\right] + \varphi_C \vartheta(\vec{\theta}) \left[\varphi_B \hat{\delta}\vartheta(\vec{\delta}) L_B\right] \end{aligned} \tag{4.8}$$

The position and velocity are used to calculate the kinetic and potential energy. The Lagrangian (L) is the difference between the kinetic energy (T) and the potential energy (V), $L = T - V$. The kinetic energy T can be determined by summing up the translational and rotational kinetic energy for each Object, as shown in Equation 4.9. The variables m_A and I_A are the mass and Inertia of Object A, m_B and I_B are the mass and Inertia of Object B, and m_C and I_C are the mass and Inertia of Object C.

$$\begin{aligned} T &= T_{trans} + T_{rot} \\ T_{trans} &= \tfrac{1}{2} m_A \dot{r}_A^T \dot{r}_A + \varphi_B \tfrac{1}{2} m_B \dot{r}_B^T \dot{r}_B + \varphi_C \tfrac{1}{2} m_C \dot{r}_C^T \dot{r}_C \\ T_{rot} &= \tfrac{1}{2} \dot{\theta}^T I_A \dot{\theta} + \varphi_B \tfrac{1}{2} \dot{\theta}^T I_B \dot{\theta} + \varphi_C \tfrac{1}{2} \dot{\theta}^T I_C \dot{\theta} \end{aligned} \tag{4.9}$$

By substituting the velocities from Equation 4.8 into Equation 4.9, the kinetic energy is expressed with respect to the state vector specified in Equation 4.6.

The potential energy (V) is due to either gravitational or spring forces. For this thesis, gravitational forces are neglected as assembly generally occurs in close proximity operations or deep space (e.g. Lagrange point). Also, autonomous assembly occurs fast enough that orbital drift, J2 effects, and similar perturbations can be neglected. For the baseline system specified in Chapter 2, only Object B has potential energy associated with deflections. Equation 4.10 gives the potential energy associated with the deflections in Object B based on the spring constant matrix κ.

$$V = V_{grav} + V_{spring} = 0 + \varphi_B \frac{1}{2} \vec{\delta}^T \kappa \vec{\delta} \tag{4.10}$$

The J matrix is defined as $J = \frac{d}{d\dot{q}} \frac{dT}{d\dot{q}}$. J is obtained by differentiating the kinetic energy expression given in Equation 4.9 with respect to the state vector (Equation 4.6) and expressing it in matrix form.

The D matrix can be computed using the fact that $\dot{q}^T \left(J - 2D\right) \dot{q}$ is a skew symmetric matrix. D, an n by n matrix, is defined in Equation 4.11, where n is the size of q.

$$D_{ij} = \frac{1}{2} \sum_{k=1}^{n} \frac{dJ_{ij}}{dq_k} \dot{q} + \frac{1}{2} \sum_{k=1}^{n} \left(\frac{dJ_{ik}}{dq_j} - \frac{dJ_{jk}}{dq_i} \right) \dot{q}_k \tag{4.11}$$

The complete D matrix can be determined by individually calculating the elements of D from Equation 4.11 using the expression obtained for J.

The G matrix is defined as $G = \frac{dV}{dq}$ and can be obtained by arranging the potential energy expression given in Equation 4.10 into a matrix form with respect to the state vector and differentiating.

Determine parameterized A matrix

The A matrix is parameterized to be generated from the property vector p, as given in Equation 4.12. The A matrix can be calculated for the specific configuration from the dynamics matrices J, D, and G, which are a function of the property vector p.

$$A = f(p) = f(J(p), D(p), G(p)) \tag{4.12}$$

The general differential equation can be rearranged into a state space representation, where the state vector $x = \begin{bmatrix} q & \dot{q} \end{bmatrix}^T$.

$$\begin{aligned}\dot{x} &= Ax + Bu \\ \dot{x} &= \begin{bmatrix} 0 & 1 \\ -J^{-1}G & -J^{-1}D \end{bmatrix} x + \begin{bmatrix} 0 \\ J^{-1} \end{bmatrix} u\end{aligned} \tag{4.13}$$

The selection of the state vector in Equation 4.13 arranges the position elements for all objects before the derivative elements. However, since Object A is always present, it is beneficial to arrange the state vector, such that the elements for Object A are at the top, followed by the elements for Object B and C. This makes the addition and removal of states associated with configuration changes simpler since the indices for Object A will not change. The state vector for this arrangement is given in Equation 4.14, where $v = \dot{r}$ and $\omega = \dot{\theta}$.

$$x = \begin{bmatrix} x & y & z & v_x & v_y & v_z & \theta_x & \theta_y & \theta_z & \omega_x & \omega_y & \omega_z & \delta_{x,B} & \delta_{y,B} & \delta_{z,B} & \dot{\delta}_{x,B} & \dot{\delta}_{y,B} & \dot{\delta}_{z,B} \end{bmatrix} \tag{4.14}$$

4.1.3 Actuator Propagation

In the Predict step of the estimator, the actuator model is incorporated into the velocity estimate to improve accuracy when the actuators are firing. The actuator propagation calculates the acceleration generated by the actuators using mass property information. The actuator propagation must be parameterized to take in the current mass and actuator configuration and output the new velocity estimate. It is assumed that that the assembler is commanding all actuators, and thus has knowledge and control of all actuators, even if they are are not located on the assembler. The actuator propagation algorithm is parameterized to take in the actuator magnitude ($\vec{F}_{act}$), locations ($\vec{r}_{act}$), and directions ($\vec{D}_{act}$), as well as the mass (m) and center of mass ($\vec{r}_{cg}$) and output the velocity estimate $\vec{v}$ (Equation 4.15).

$$\vec{v} = f(p) \Rightarrow \vec{v} = f(\vec{F}_{act}, \vec{r}_{act}, \vec{D}_{act}, m, \vec{r}_{cg}) \tag{4.15}$$

The first step in the actuator propagation algorithm is to determine the actuator forces present at the current time. Equation 4.16 shows the equation to determine the actuator forces present, where

- $\vec{F}_{act}$ is the magnitude of the force produced by each actuator. It is a vector whose length is equal to the number of actuators.
- $\vec{Act}_{on}$ is vector of boolean variables that indicate the on (currently firing) or off (not firing) status for each actuator. It is a vector whose length is equal to the number of actuators. $\vec{Act}_{on}$ is an input to the estimator and not an element of the property vector p.
- f is the force imparted onto the system by each actuator. It is a vector whose length is equal to the number of actuators.
- k is a subscript used to denote a specific actuator.

$$\vec{f}_k = \vec{F}_{act,k}\vec{Act}_{on,k} \tag{4.16}$$

Next, if there are any actuators on, the body impulse generated for the current time step is calculated using the actuator configuration. Equation 4.17 shows the body impulse generated by the thrusters that are on, where $\vec{D}_{act}$ is a nx3 matrix (n is the number of actuators) that has the x, y, and z force directions produced for each actuator. The impulse is then converted from the body frame to the global frame using the current attitude estimate.

$$\vec{imp}_{Body} = \vec{D}_{act}^{T}\vec{f}dt \tag{4.17}$$

The actuator propagation must also account for changes in center of mass. A change in center of mass ($\vec{r}_{cg}$) has an impact on velocity because actuator commands can cause a torque as well as a force. Equation 4.18 gives the torque impulse generated by the actuators. The variable $\vec{D}_T$ is a nx3 matrix of the x, y, and z torque directions determined by $\vec{D_T} = \vec{r}_{act} \times \vec{D}_{act}$, where $\vec{r}_{act}$ is a nx3 matrix of position vectors corresponding to the locations of the actuators with respect to the center of the assembler.

$$\vec{imp}_T = \vec{r}_{cg} \times (\vec{D_T}^T \vec{u}) \tag{4.18}$$

Finally, Equation 4.19 shows how the velocity is updated to account for the impulse using the mass and inertia of the system, where v^+ is the velocity estimate after actuator propagation and v^- is the velocity estimate before actuator propagation.

$$\vec{v} = \vec{v} + \frac{\vec{imp}_{global}}{m} + \frac{\vec{imp}_T}{\vec{r}_{cg} m} \tag{4.19}$$

4.1.4 Measurement Incorporation

A change in sensor configuration can be due to the addition or removal of new sensors or a change in sensor type. The measurement matrices H and h as well as the sensor noise matrix R, as given in Equation 4.4, must be parameterized to account for changing sensor characteristics. To account for the changing sensor configurations, these matrices must be generated online based on the information in the property vector p.

A key issue to consider when incorporating new sensors into the estimator is whether the additional measurements will add value. The addition of multiple estimates of the same position can help to reduce the error of the estimate, but also adds noise associated with each sensor. Also, the placement of the sensors is important because the farther the sensor is from the point one is trying to estimate, the more errors arise when trying to convert from the location estimated to the located sensed.

The exact mapping of the H matrix is heavily dependent on the type of sensors used. Two types are considered in this work, inertial and position sensors.

Inertial Sensors

Inertial sensors measure the angular rotation rate or acceleration of the object with respect to a body fixed frame. Examples of inertial sensors are gyroscopes and accelerometers. Gyroscopes are used as the baseline inertial sensor in this work to demonstrate parameterization. Gyroscopes provides a direct measurement of the angular rates of the vehicle. Equation 4.20 shows the parameterization of the gyroscope measurements, where ρ is the axis along which the gyroscope is measuring, β is the gyroscope bias, and χ is the gyroscope scale factor.

$$z_{omega} = f(p) \Rightarrow z_{omega} = f(\rho, \chi, \beta) \tag{4.20}$$

The estimated angular rates can then be calculated as in Equation 4.21, where z_{raw} is the three element raw measurement vector received from the gyroscope [56].

$$z_{omega} = \rho\left(\chi z_{raw} - \beta\right) \tag{4.21}$$

Given the state vector in Equation 4.14, the h vector is given as

$$h(x_k) = \begin{bmatrix} 0_{3x3} & & & & & & & \\ & 0_{3x3} & & & & & & \\ & & 0_{3x3} & & & & & \\ & & & 1 & 0 & 0 & & \\ & & & 0 & 1 & 0 & & \\ & & & 0 & 0 & 1 & & \\ & & & & & & 0_{3x3} & \\ & & & & & & & 0_{3x3} \end{bmatrix} x_k \tag{4.22}$$

The corresponding h vector for this sensor simply extracts the rotational rates from the state vector. Thus, it does not change for different types of gyroscopes or mass properties.

Position Sensors

Position sensors measure the position of the object with respect to an external reference frame. For position sensors, a beacon / ultrasound receiver scheme is considered here. This sensor scheme can be extrapolated into many types of sensors, such as

GPS and relative ranging sensors. Equation 4.23 shows the parameterization of h and H to be generated from the beacon and receiver locations, where $\vec{r}_{Rx}$ is an nx3 matrix of receiver positions with respect to the center of the assembler where n is the number of receivers. The variable $\vec{r}_{Tx}$ is a mx3 matrix of beacon positions with respect to an inertial reference frame where m is the number of beacons.

$$\begin{aligned} h = f(p) &\Rightarrow h = f(\vec{r}_{Rx}, \vec{r}_{Tx}) \\ H = f(p) &\Rightarrow H = f(\vec{r}_{Rx}, \vec{r}_{Tx}) \end{aligned} \tag{4.23}$$

Equation 4.24 shows the derivation of the h as the distance between the receiver on the vehicle and the beacon it is sensing with respect to a inertial frame, where $\vec{r}_A$ is the position of Object A in the inertial frame. The variable Θ is a transformation matrix that converts the receiver location given in the body frame to the global frame using the current attitude estimate. [56]

$$h(x_k) = \left\| \Theta^T \vec{r}_{Rx} + \vec{r}_A - \vec{r}_{Tx} \right\| \tag{4.24}$$

Equation 4.25 shows the generation of the H matrix from Equation 4.24 such that the position and attitude estimates are calculated from the position measurement, where j is the index of the state vector. The H in Equation 4.25 is a linearized matrix used to propagate the covariance estimate.

$$H = \left\{ \begin{array}{ll} \frac{\left[\Theta^T \vec{r}_{Rx}\right]_j + [\vec{r}_A]_j - [\vec{r}_{Tx}]_j}{h} & , j = 1:3(position) \\ \frac{\left(\Theta^T \vec{r}_{Rx} + \vec{r}_{pos} - \vec{r}_{Tx}\right)^T \left[\frac{d\Theta^T}{dq_{j-6}} \vec{r}_{Rx}\right]}{h} & , j = 7:10(attitude) \\ 0 & , otherwise \end{array} \right\} \tag{4.25}$$

Noise Characteristics

The modification to the noise matrix R depends on the nature of the sensor configuration change. It can be parameterized with respect to the sensor configuration, namely the number and type of sensors. If identical sensors are added, the size of R grows but maintains the same element values. If a different type of sensor is used, the noise characteristics need to be accounted for. Equation 4.26 gives a generic expression for

R, where $\vec{r}_{Rx}$ is the matrix of receiver locations, R_{Rx} is the noise for a single receiver, and R_{gyro} is the noise for a single gyroscope.

$$R = f(p) \Rightarrow R = f\left(\vec{r}_{Rx}, R_{Rx}, R_{gyro}\right) \tag{4.26}$$

Equation 4.27 shows one way to generate the noise matrix R using the type of sensor and number of sensors averaged in that measurement, where n_{Rx} is the number of sensors used to measure that state, and j is the diagonal index of the matrix R. The value for n_{Rx} depends on pre-filtering techniques as well as receiver placement.

$$R = \left\{ \begin{array}{lll} \frac{R_{Rx}}{n_{Rx}} & ,j = 1:3 & position \\ \frac{R_{Rx}}{n_{Rx}} & ,j = 7:10 & attitude \\ R_{gyro} & ,j = 11:13 & rotationrate \\ 0 & ,otherwise & \end{array} \right\} \tag{4.27}$$

4.1.5 Initialization

When the model changes, the estimator must be re-initialized with the new model. Nominally, EKFs are initialized based on an initial guess of the state vector. The convergence properties of the estimator are dependent on the accuracy of this initial guess. During a transition in an assembly mission, one has information about the state of the assembler just prior to the configuration change, specifically the state estimate and the corresponding covariance. Table 4.1 shows the four options of how to use the information, to improve the accuracy of the initial guess used to initialize the estimator for the next configuration. The values in the columns "State" and "Covariance" indicate whether the state and covariance information should be kept or reset to default values.

Case 2 should not be implemented on an actual system because one is discarding the state information and keeping the converged covariance information. This makes the estimator believe that it has a accurate initial state, which is not the case. In most situations, the measurements received will indicate a significantly different state than the initial state. Thus, the estimator could reject all new measurements because

Case	State	Covariance	Performance
1	Reset	Reset	Acceptable
2	Reset	Keep	Diverges
3	Keep	Reset	Acceptable
4	Keep	Keep	Acceptable

Table 4.1: Initialization Setup Truth Table

the covariance leads it to believe the state is correct and the measurements are faulty.

Case 1, resetting both sets of information, will work provided the reset state is close enough such that the EKF converges. However this case could lead to additional time and resources used, especially if there are multiple configuration changes during the course of a mission and one is reconverging the state at each transition.

However, if one does keep the information, how is that information incorporated into the new estimator? First, consider the impact of keeping the state information as in Case 3 through the impact on the EKF predict equation (Equation 4.28), where k denotes the current time step, x_k is the estimate after including the measurement, $\hat{x}_k^-$ is the estimate prior to including the measurement, z_k is the measurement, K_k is the Kalman gain calculated from the covariance, and h_k is the matrix to convert the state into an estimated measurement.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h_k(\hat{x}_k^-)) \tag{4.28}$$

If one keeps the state information, the value of $h_k(\hat{x}_k^-)$ will be closer to the measurement, z_k. Thus, overall, the state estimate will have less error during the convergence period. Stability could also an issue, if initialized with a default state vector far from the current state. The EKF is linearized about the current state estimate, therefore, a better initial state estimate can improve the stability. Keeping the state information, however, has little impact on the covariance or the convergence time. Considering the covariance update equation, one can determine the impact of keeping the covariance value. The value of the covariance at the end of the previous configuration has most likely converged, thus it would be much smaller than any initial covariance that one would initialize the EKF with. Since convergence is determined by the covariance being less than a certain value, keeping the covariance should cause a decrease in convergence time.

If minimal motion is expected during transition, Case 4 should be selected because it maximizes the use of prior information to minimize transition estimation error and convergence time. If large motions are expected during transition, Case 2 or 3 would be more robust because they assume uncertainty in the state by discarding the covariance information.

4.2 Controller

Two types of controllers are considered in this work: Proportional-Integral-Derivative (PID) controllers and Adaptive controllers. These two types are selected to present the parameterization of distinctly different algorithms. For each type of controller, a single design is presented and parameterized. The demonstration on these controllers demonstrates the feasibility of the parameterization. Selection and design of a controller for an assembly mission requires significant work to ensure stability and performance for all possible configurations. The parameterization technique may not be appropriate for all controllers, but is demonstrated in this work to be be a viable option.

4.2.1 PD/PID Controllers

Proportional-Derivative (PD) and Proportional-Integrative-Derivative (PID) controllers generate the control input from pre-set gains that are calculated based on the mass properties of the system. These controllers can be used for both attitude and position with similar form, and are well suited to parameterization for rigid body systems.

The control laws are a function of the state error and of a set of gains, K. The gains hold the mass property information and dictate the tracking performance of the controller. The gains must be re-calculated for the new model to parameterize these controllers for multiple configurations. The gains are parameterized to be a function of the inertia tensor, mass, and actuator force magnitude, as given in Equation 4.29.

Though the bandwidth also may need to change for different configurations, it is a function of the mass and actuator force magnitude. For a rigid body assembler, the bandwidth can be kept the same if the necessary actuation force is available. Once saturation of the actuators is reached, the bandwidth should be scaled down proportional to the increase in mass.

$$u = f(p) \Rightarrow u = f(K(p)) = f(\vec{I}, m, \vec{F}_{act}) \tag{4.29}$$

The PD control law is given by Equation 4.30, where e_x, e_y, and e_z are state errors (e.g., attitude, position) and $\dot{e}_x$, $\dot{e}_y$, and $\dot{e}_z$ are the derivative errors (i.e., angular rate, velocity). The control input is given by u. K_p and K_d are the proportional and derivative gains. These gains can be constant or in matrix form to account for differences between axes.

$$u = K_p \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} + K_d \begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_z \end{bmatrix} \tag{4.30}$$

The PID control law is given by Equation 4.31, where K_p, K_i, and K_d represent the proportional, integral, and derivative gains for each axis. The variable ψ is the integration term, which is defined as $\psi = \psi + e\tau$ for each axis with τ being the integration time period.

$$u = K_p \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} + K_d \begin{bmatrix} \dot{e}_x \\ \dot{e}_y \\ \dot{e}_z \end{bmatrix} + 2K_i \begin{bmatrix} \psi_x \\ \psi_y \\ \psi_z \end{bmatrix} \tag{4.31}$$

The gains are calculated using Equation 4.32 for attitude and Equation 4.33 for position, where w_n is the control bandwidth, m is the mass, $\vec{I}$ is the inertia tensor, ζ is the damping ratio, and Λ is the time constant associated with the integral term. The gains calculation equations for these controllers are from Wie's Spacecraft Attitude Dynamics and Control [80]. The selection of w_n and ζ are nominally from picking desired performance characteristics (such as rise time and settling time) and converting that into w_n and ζ based on the characteristic polynomial of the system (eg. $s^2 + 2\zeta w_n s + w_n^2 = 0$). The parameters ζ and Λ can be tuned to the assembler's dynamics and maintained through the configuration changes. The control bandwidth w_n can be originally set to a tuned value based on the assembler's dynamics, then scaled proportionally to mass or inertia as the configuration changes.

$$K_p = \left(w_n^2 + \tfrac{2\zeta w_n}{\Lambda}\right) I \quad K_i = \left(\tfrac{w_n^2}{\Lambda}\right) I \quad K_d = \left(2\zeta w_n + \tfrac{1}{\Lambda}\right) I \tag{4.32}$$

$$K_p = \left(w_n^2 + \tfrac{2\zeta w_n}{\Lambda}\right) m \quad K_i = \left(\tfrac{w_n^2}{\Lambda}\right) m \quad K_d = \left(2\zeta w_n + \tfrac{1}{\Lambda}\right) m \tag{4.33}$$

4.2.2 Adaptive Controller

The adaptive controller described in this section is based on a direct adaptive tracking controller designed for nonlinear manipulator control by Niemeyer and Slotine [55]. This baseline adaptive controller is selected because it can easily incorporate both flexible dynamics associated with Object B's deflections as well as the rigid body motion of Object A and C. The recursive structure for the calculation of the control vector makes implementation computationally efficient. This controller was developed in detail for the configuration of Object A+B+C in Katz [42] and is parameterized here to work for the baseline system given in Chapter 2. The adaptive controller is parameterized with respect to the property vector p as in Equation 4.34. For an adaptive controller, the parameterization amounts to the proper initialization of the parameter vector $\hat{a}$ and the proper selection of gains.

$$u = f(p) = f(\hat{a}(p), K_p(p), K_d(p)) \tag{4.34}$$

The adaptive controller seeks to generate the control vector u given a desired reference trajectory q_d and a mass property vector a (for which some or all of the parameters may be unknown). It uses an adaptation law for the unknown parameters such that the actual position q tracks the desired position q_d. The adaptive control law is given by Equation 4.35, where s is a velocity tracking error defined as $s = \dot{q} - \dot{q}_r$, where $\dot{q}_r = \dot{q}_d - \lambda(q - q_d)$ with gain λ.

$$u = Y\hat{a} - K_d s \tag{4.35}$$

The variable $\hat{a}$ is a estimated parameter vector, which consists of the mass properties from Objects A, B, and C, as shown in Equation 4.36. The size of $\hat{a}$ is $10 + 13\varphi_B + 10\varphi_C$. The corresponding adaptation law to propagate $\hat{a}$ is given by Equation 4.37.

$$\hat{a} = \begin{bmatrix} m_A & I_{xx,A} & I_{yy,A} & I_{zz,A} & I_{xy,A} & I_{yz,A} & I_{xz,A} \\ m_A * r_{cm,x} & m_A * r_{cm,y} & m_A * r_{cm,z} & & & & \cdots \\ \varphi_B m_B & \varphi_B I_{xx,B} & \varphi_B I_{yy,B} & \varphi_B I_{zz,B} & \varphi_B I_{xy,B} & \varphi_B I_{yz,B} & \varphi_B I_{xz,B} \\ \varphi_B m_B * r_{cm,x,B} & \varphi_B m_B * r_{cm,y,B} & \varphi_B m_B * r_{cm,z,B} & & & & \cdots \\ \varphi_C m_C & \varphi_C I_{xx,C} & I_{yy,C} & \varphi_C I_{zz,C} & \varphi_C I_{xy,C} & \varphi_C I_{yz,C} & \varphi_C I_{xz,C} \\ \varphi_C m_C * r_{cm,x,C} & \varphi_C m_C * r_{cm,y,C} & \varphi_C m_C * r_{cm,z,C} & & & & \cdots \\ \varphi_B k_{x,B} & \varphi_B k_{y,B} & \varphi_B k_{z,B} & & & & \end{bmatrix} \tag{4.36}$$

$$\dot{\hat{a}} = -K_p Y^T s \tag{4.37}$$

The variables, K_d and K_p, are tracking and adaptation gains that must also be appropriately set for each configuration. Nominally, the selection of the configuration gains should be done via simulation modeling. However, due to the adaptive nature of the algorithm, a single set of gains may be applicable for all configurations, if selected properly.

The variable Y in Equation 4.35 is a dynamic regressor that includes the model dynamics. The dynamic regressor must be updated such that it can be generated for each configuration based on the model properties. The dynamic regressor is defined as

$$\tilde{J}(q)\ddot{q}_r + \tilde{D}(q,\dot{q})\dot{q}_r + \tilde{G}(q) = Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\hat{a} \qquad (4.38)$$

where $\tilde{J} = \hat{J} - J$, $\tilde{D} = \hat{D} - D$, and $\tilde{G} = \hat{G} - G$. Y is calculated by estimating the dynamics matrices using the properties in $\hat{a}$ to compute the mass and inertia contributions from each object. The parameterized version includes three phases: upwards, downwards, and adaptation. The first phase performs an upwards calculation to determine the forces associated with each object based on the velocity and angular rate. The desired translational velocities are initialized for Object A directly, since they do not require any rotations. The angular rates for Objects A and C and deflections of Object B are calculated by summing up the contribution at each attachment point. The corresponding force at each attachment point is calculated. The second phase performs a downwards calculation to sum up the forces at each attachment to compute the total force required at Object A, which is the feedforward term $Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\tilde{a}$. The third phase constitutes the adaptation phase, which uses the error between the desired velocities and actual velocities and the force contribution to determine the necessary change in $\hat{a}$. The details on the calculations for each phase can be found in Niemeyer and Slotine [55].

4.3 Control Allocation

The control allocation algorithm considered is a pulse width modulation algorithm that converts the control input vector into actuator commands. The control allocator converts the control vector into actuator commands using a Mixing matrix (Ξ), control period, duty cycle, and minimum and maximum pulse widths. The parameterization of the control allocation algorithm, as given in Equation 4.39, generates the Mixing matrix online.

$$\vec{Act}_{on} = f(p) \Rightarrow \vec{Act}_{on} = f(\Xi(p)) \qquad (4.39)$$

The mixing matrix generally is a hardcoded matrix since it is static for a single configuration. Parameterization of the control allocation algorithm generates the mixing matrix online based on the actuator model. The variables of the p vector used are center of mass, location of actuator, direction of actuator, and force of actuator.

$$f(\Xi(p)) = f(\vec{r}_{cg}, \vec{r}_{act}, \vec{D}_{act}, \vec{F}_{act}) \tag{4.40}$$

The parameterized control allocation needs the following inputs to generate the mixing matrix:

- $\vec{r}_{act}$ is the nx3 matrix of actuator positions with respect to the geometric center of Object A, where n is the number of actuators
- $\vec{r}_{cg}$ is the center of mass with respect to the geometric center of Object A
- $\vec{D}_{act}$ is the nx3 matrix of actuator force directions with respect to the body frame of Object A
- $\vec{F}_{act}$ is the nx1 vector of actuator force magnitdues

Equation 4.41 shows the calculation of the mixing matrix using the actuator force/torque configurations. The calculated quantities are

- $\vec{r}$ is the matrix of actuator positions with respect to the center of mass
- $\vec{D}_T$ is the nx3 matrix of actuator torque directions with respect to the body frame of Object A
- Ξ is the mixing matrix given as the combination of the force and torque directions.

$$\begin{aligned} \vec{r} &= \vec{r}_{act} - \vec{r}_{cg} \\ \vec{D}_T &= \vec{r} \times \vec{D}_{act} \\ \Xi &= [\vec{D}_{act}; \vec{D}_T] \end{aligned} \tag{4.41}$$

The rows corresponding to non-active actuators can be removed by using an actuator health variable ($\vec{Act}_{health}$), which starts as an identity matrix of size equal to the number of actuators. If an actuator fails, that index of $\vec{Act}_{health}$ is set to zero. Equation 4.42 shows the calculation for the corresponding mixing matrix to include only healthy actuators.

$$\Xi_{health} = \vec{Act}_{health}\Xi \tag{4.42}$$

The use of the active actuator input allows for flexibility in the system. It can be used to implement additional constraints, such as for fault detection or plume impingement. The mixing matrix is then used to calculate the actuator commands, with Equations 4.43 and 4.44. The control input u_{ctrl} is multiplied by the inverse of the Mixing matrix to get the actuator forces (u_{act}).

$$u_{act} = \Xi_{Health}^{-1} u_{ctrl} \tag{4.43}$$

The actuator forces are converted into individual actuator on-off commands (Act_{on}) in seconds (Equation 4.44), using the control period (η_{ctrl}), duty cycle (Ω), and actuator force magnitude (F_{act}). The actuator commands are first scaled such that the maximum force is less than or equal to the maximum pulse width ($maxPulseWidth = \eta_{ctrl} * \Omega$). More information can be found in Hilstad [29].

$$\vec{Act}_{on} = \frac{\eta_{ctrl} * \Omega * u_{act}}{F_{act}} \tag{4.44}$$

4.4 Conclusions

This chapter describes the parameterization of a simple control system such that it can seamlessly take in the model and properly estimate and control for the current configuration. The parameterization of the algorithms for the estimator, controller, and control allocation are described. The techniques described for the selected algorithms presented in this chapter can be expanded to other control algorithms. This parameterized control system is validated through implementation on the SPHERES hardware testbed; the results are shown in Chapter 5.

The parameterization process is specific to the control system design. Though the algorithms that require parameterization remain the same, the nature of the parameterization is very dependent on the design of the control system. Two main conclusions are derived:

1. Selection of the control system algorithms is critical in enabling a successful parameterization. Certain algorithms are more suited to being written as a function of the mass properties.

2. It is helpful to start with a control system that has been tuned for the desired performance for the assembler, then use the parameterization technique to enable it to accommodate other configurations.

The parameterization of this baseline control system is geared toward validating the Online Model Calculation design. The proper design of a parameterized control system is a significant area of research.

Chapter 5

Online Model Calculation: Implementation on SPHERES

This chapter describes the implementation of the Online Model Calculation design. This thesis accommodates the following two types of modules:

1. Rigid body: Rigid body modules can be modeled as a single lumped mass. Once they are attached to the assembler, the location and orientation of the module can be entirely specified by the state of the assembler. In some scenarios, modules can also have sensing or actuation capability. Model generation for a rigid body module involves accounting for changes in mass, inertia, actuator configuration, and sensor configuration. Accounting for the additional actuation and sensing capability of the module could lead to several key benefits, such as fuel savings on the assembler; greater mobility for the assembler-module system; and more accurate state estimates if sensors on the assembler are blocked when the module is attached.

2. Flexible body: Flexible body modules are objects whose state cannot be entirely determined from the state of the assembler. These objects include deflection dynamics which are characterized by stiffness parameters. Flexible modules present a unique challenge because not only do the mass properties change upon attachment, but the stiffness properties change as well. In this work, flexible

modules are assumed to be passive objects with no actuation capability, but do have deflection sensing capability. The combined system is an underactuated, but observable system. Reconfiguration to account for flexible module involves accounting for the flexible dynamics in the control algorithms, as well as the mass properties.

This chapter starts with a description of the hardware setup and prerequisite software algorithms that were developed to aid in testing. The next three sections focus on the parameterized control system results: estimator, controller, and control allocation respectively. Each algorithm section describes the implementation of the parameterized algorithms and presents results from simulation and hardware testing. The following section describes the results for two integrated tests, which demonstrate the entire Online Model Calculation design. The chapter is concluded with a discussion of the validation of the Online Model Calculation design.

5.1 Testing Set-up

Many hardware and software components were used in obtaining the testing results presented in this chapter. The hardware components used in testing are described in detail in Chapter 3. This section describes the specific configurations used and the correlation to the hardware, the development of a path planner to facilitate testing, and upgrades to the SPHERES simulation made to account for multi-body dynamics.

Three configurations are used throughout this testing: Object A only, Object A+C, and Object A+B+C. Object A in this work is a SPHERES satellite. Object C, a rigid body module, is also a SPHERES satellite. To form the Object A+C configuration, the Object A satellite and Object C satellite are attached through their Velcro faces as shown in Figure 5-1. The Object C satellite is an active module; it has both actuation and sensing ability. Object B, the flexible module, is a four-link segmented beam. The beam is attached to Object A satellite via a mounting plate, as described in Section 3.2. Figure 3-12, depicting the hardware set-up of the SPHERES satellite attached to the beam is repeated below for clarity. The Object

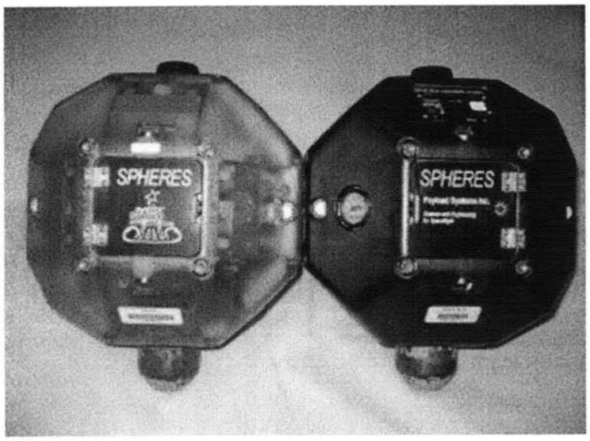


Figure 5-1: Two SPHERES satellite attached via Velcro

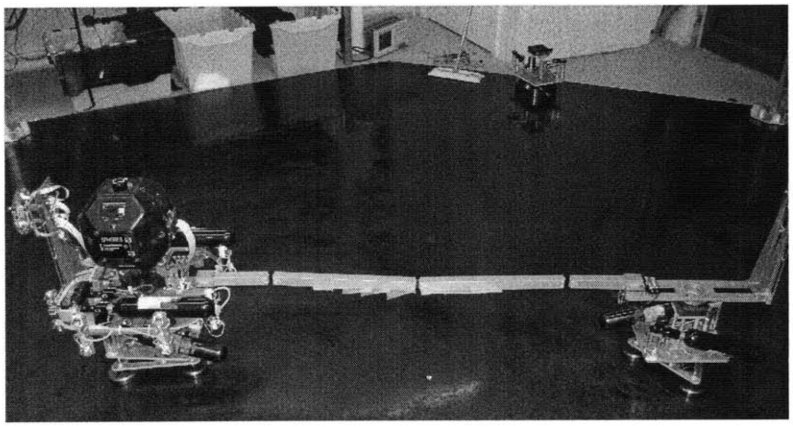
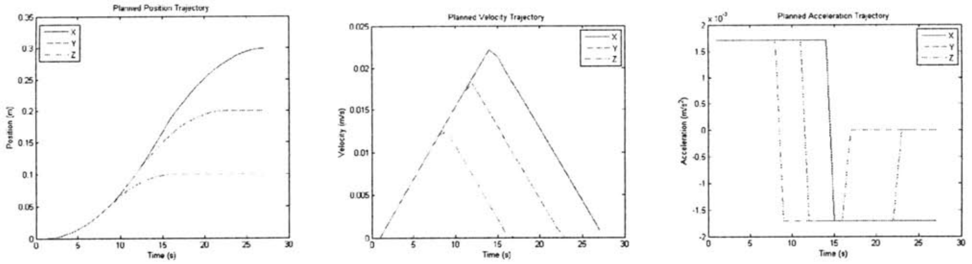


Figure 5-2: SWARM with Beam attached

B flexible module has no actuation capability, but it does have sensing capability via an LED/Camera system that is used to measure the deflection of the beam. Experimental testing was performed in three environments: in simulation, on ground hardware, and on ISS hardware. The environments are denoted SIM, GND, and ISS respectively.

An additional algorithm necessary for the implementation of the controllers is a path planner. The adaptive controller in particular is designed as a trajectory tracking controller and does not work with only waypoint targets. Thus, a bang-bang path planner was developed and implemented on the SPHERES hardware. The planner calculates the path from the initial state to the final desired state. Example position, velocity, and acceleration profiles are shown in Figure 5-3. The resolution of the path



(a) Position (b) Velocity (c) Acceleration

Figure 5-3: Example planned trajectory using on-board path planner

calculated is 0.1s, and is output at the resolution of the control period.

Finally, a modified version of the SPHERES simulation was created to handle the configuration of two satellites attached via Velcro. This simulation used an attached flag, such that if the satellites were docked, it would use a model of the docked configuration, including the additional thrusters. This simulation was compared to ISS tests from Test Session 13 (September 2008). Figure 5-4 shows the performance of the simulated position versus the actual position that occurred during the test. Though the tracking performance is tolerable, some of the performance differences may be due to the simplification of the model in the simulation. Unmodeled dynamic effects include Velcro alignment issues, thruster force damping due to velcro spring effect, communication delay in thruster firing, and deviations between the actual mass and inertia and the average values modeled in the simulation. Updates have been made to the simulation based on the ISS data received to date. The updates will be compared to flight data for the next test session, scheduled for late spring 2010 in Test Session 22.

5.2 Estimator

Four main areas of the estimator are parameterized: state propagation, actuator propagation, measurement incorporation, and initialization.

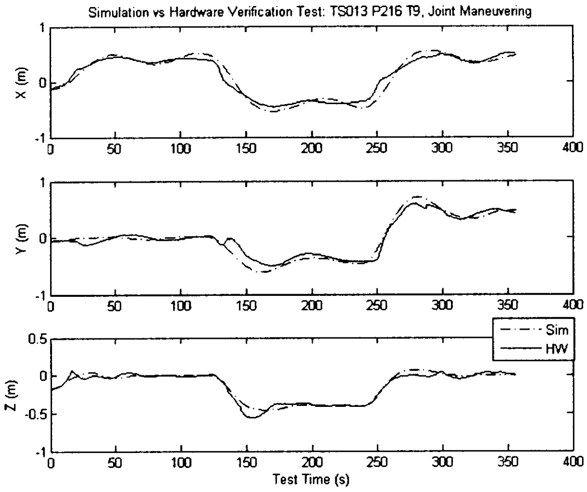


Figure 5-4: Verification of Multi-body simulation versus ISS data

5.2.1 State Propagation

The state propagator is only updated for flexible dynamics. For the rigid body modules, there are no additional states (e.g. deflection) because the state of the module is completely defined by the state of the assembler. Thus, only the actuator propagation needs to be parameterized for rigid body modules.

Experiment

The estimator needs significant updates to account for flexible dynamics. First, in order to account for Object B, additional states must be added to account for the deflection of the beam, as shown in Equation 4.6. For implementation ease, the propagation of the dynamics of the beam deflection are performed separately. The true beam is a four-link beam, while an approximation is used in this calculation that assumes a single mode deflection. The velocity contribution from the beam deflection is added to Object A's state separately. The additional states added are the deflection δ along the y axis and the deflection rate $\dot{\delta}$.

For the 2D set-up, the effects of the beam deflection on the assembler can be determined by performing a torque balance on the system. The deflection of the beam creates a torque in the perpendicular (Z) direction, which causes a corresponding force at Object A based on the ratio to the center of mass. Equation 5.1 gives the y velocity including the beam propagation component: where v_y is the velocity of Object A in the y direction, L is the length of the beam, x_{cg} is the x component of the center of mass, k is the beam stiffness, δ_y is the beam deflection, m is the mass of Object A, and dt time duration of the beam deflection propagation in seconds. The superscripts $^-$ and $^+$ indicate the velocity estimate before and after incorporating the beam deflection dynamics.

$$v_y^+ = v_y^- + \frac{L - x_{cg}}{x_{cg}} k \delta_y \frac{1}{m} dt \tag{5.1}$$

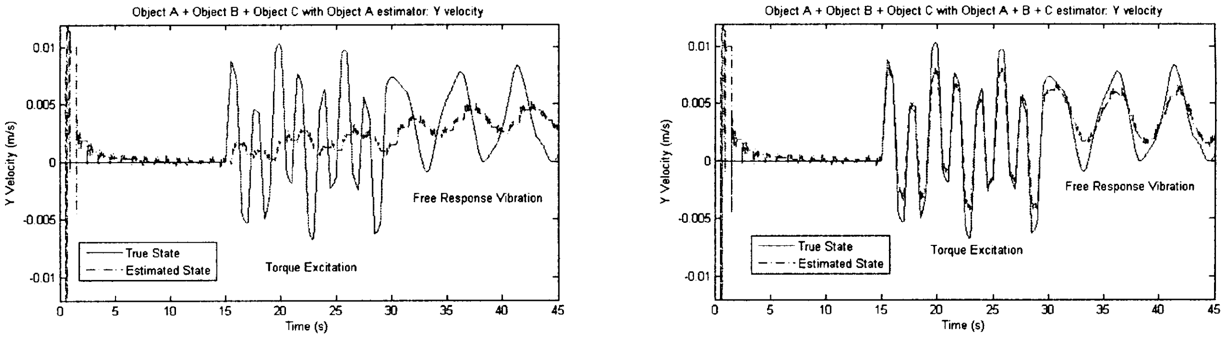
The torque magnitude for the simulation tests was increased to 5 times the actuation torque of the hardware set-up to fully excite the system. The simulation tests are set up with the following sequence:

- t = 0 - 15 s: estimator initialization
- t = 16 - 30 s: torque excitation phase
- t = 31 - 45 s: free vibrartion, no thruster firings.

The magnitude of the SWARM propulsion module thrusters are not sufficient to excite the beam to achieve the desired oscillations, given the friction dynamics of the floor. The oscillations achievable with the SWARM propulsion module are not large enough to impact the y velocity of the assembler. Thus, hardware results of this test are not included.

Results

Figure 5-5 shows the simulation results for the beam deflection component of state propagation. Figure 5-5a shows the estimator performance versus the true state when using an estimator that only models Object A. The velocity is mostly constant and does not track the true velocity. The slight increase of the velocity from t=15 s to t=45 s in Figure 5-5a is due to the differential position measurements that have been propagated.



(a) Object A estimator

(b) Object A+B+C estimator

Figure 5-5: True State vs Estimated State for SWARM + Beam (Object A+B+C), Simulation

Figure 5-5b uses an estimator based on the model for the Object A+B+C configuration. It shows much better tracking of the true state. Exact tracking performance is dependent knowledge of the stiffness parameter k.

Analysis

The importance of updating the assembler state propagator to account for the flexible dynamics depends on the total actuator capability to excite the beam, as well as the ratio between the length and stiffness of the beam and the mass of the assembler. Figure 5-6 shows the effect of the beam deflection on the y velocity estimate as a function of Object A (assembler) mass and beam length, assuming constant stiffness. For very large assembler masses, the beam deflection is not large enough to impact the velocity of the assembler. For very small beam, the energy associated with the beam deflections is small enough that it does not impact the velocity of the assembler. Therefore, the effect of the beam deflection on the assembler is a critical consideration only when the beam and assembler are of comparable size.

Knowledge of where the assembly system under consideration lies on Figure 5-6 is important in determining if the parameterization to account for the beam deflection is necessary. Other factors to consider are the stiffness of the beam and the ability for the actuators to excite beam dynamics. Stiffer beams have smaller deflections, which in turn produce smaller disturbances on the assembler. The actuator dynamics

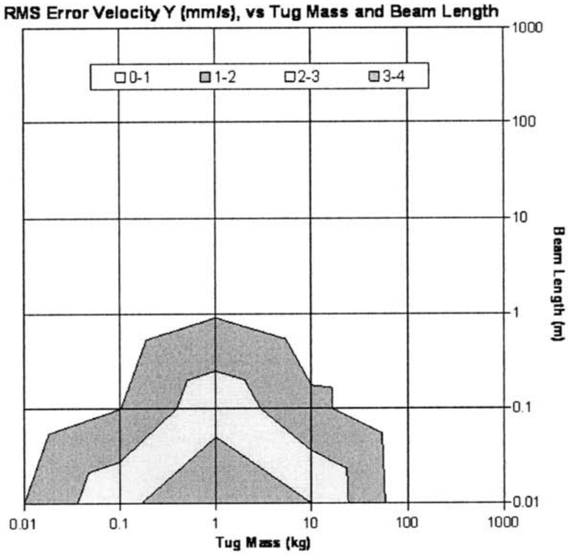


Figure 5-6: Increased Y Velocity error due to beam deflection as a function of Object A mass and Beam Length

impact the excitation of the beam to cause deflections. Some factors to consider are the maximum excitation force on the beam from the actuators and the frequency of actuator with respect to the vibration frequencies of the beam.

5.2.2 Actuator Propagation

Equation 5.2 gives the actuation propagation relation, where: v^+ refers to the velocity estimate after the propagation, v^- refers velocity estimate prior to propagation, f is the force generated by the actuators, dt is the duration the actuators are on, and m is the mass of the assembler.

$$v^+ = v^- + \frac{f}{m}dt \tag{5.2}$$

The mass is obtained from the property vector p, while f is obtained through summing the commanded thrusters on the SPHERES satellites. Figure 5-7 shows the algorithm updates for the estimator thruster propagation.

In the baseline algorithm, the thruster commands for the assembler's thrusters are converted into the impulse imparted based on the commanded duration and the thruster force. The impulses are converted into a velocity contribution based on

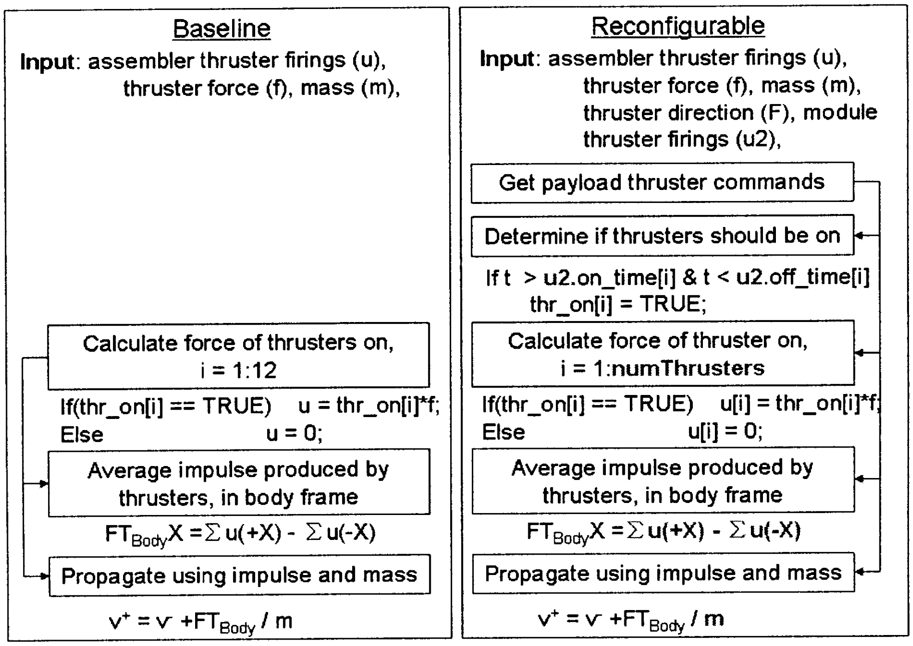

Baseline
Input: assembler thruster firings (u),
thruster force (f), mass (m),
Calculate force of thrusters on,
i = 1:12
If(thr_on[i] == TRUE) u = thr_on[i]*f;
Else u = 0;
Average impulse produced by
thrusters, in body frame
FT$_{Body}$X = Σ u(+X) - Σ u(-X)
Propagate using impulse and mass
v$^+$ = v$^-$ +FT$_{Body}$ / m
Reconfigurable
Input: assembler thruster firings (u),
thruster force (f), mass (m),
thruster direction (F), module
thruster firings (u2),
Get payload thruster commands
Determine if thrusters should be on
If t > u2.on_time[i] & t < u2.off_time[i]
thr_on[i] = TRUE;
Calculate force of thruster on,
i = 1:numThrusters
If(thr_on[i] == TRUE) u[i] = thr_on[i]*f;
Else u[i] = 0;
Average impulse produced by
thrusters, in body frame
FT$_{Body}$X = Σ u(+X) - Σ u(-X)
Propagate using impulse and mass
v$^+$ = v$^-$ +FT$_{Body}$ / m

Figure 5-7: Process Flow Diagram comparing baseline thruster propagation to reconfigurable thruster propagation

the mass. In the reconfigurable algorithm, first the module thruster commands are obtained and the corresponding thruster configuration. The total number of thrusters is determined. Then, the impulse is calculated for all of the thrusters present, both on the assembler and module. The impulse is converted into the velocity contribution using the current mass of the configuration, accounting for any modules that could be attached.

Experiment

The update of the thruster propagation has the biggest impact on the velocity estimate. To identify the impact on the velocity, a test was created for the Object A+C configuration. Figure 5-8 shows a schematic of the test set up, with the desired velocity profile based on the thruster firing pattern. The test is split into three phases:

- Phase 1 (t = 0 - 20 s): estimator initialization
- Phase 2 (t = 21 - 30 s): assembler fires in the -X direction, for 600 ms every 1 s
- Phase 3 (t = 31 - 40 s): module fires in the +X direction, for 600 ms every 1 s

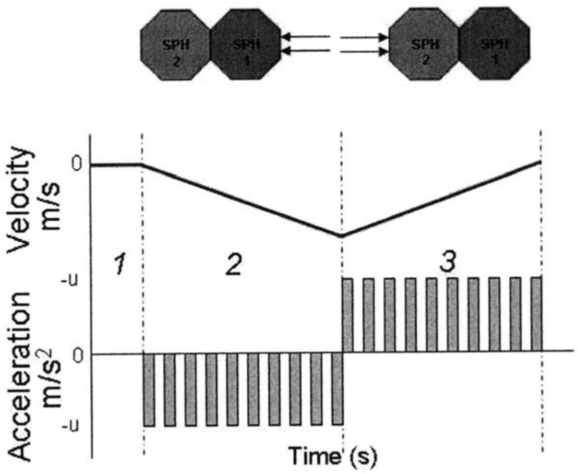
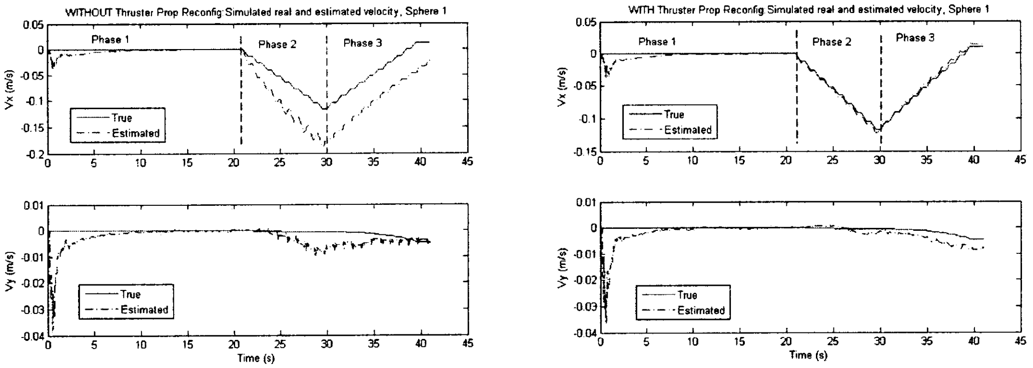


Figure 5-8: Test setup for Estimator Thruster Propagation tests

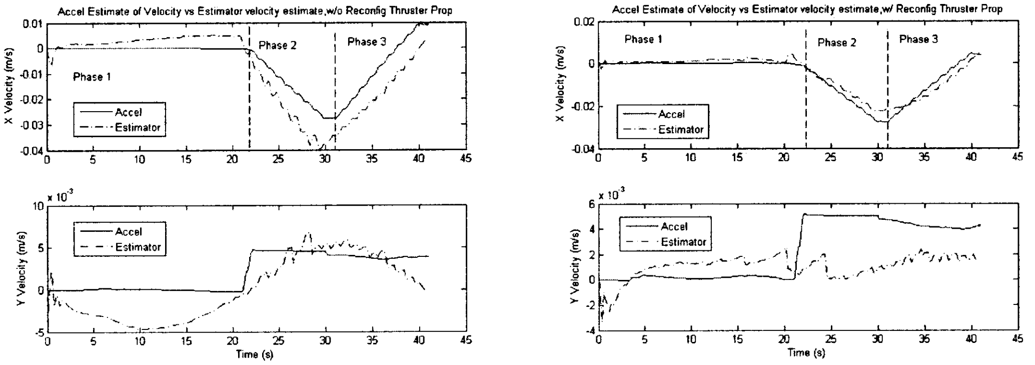
Results

Figures 5-9, 5-10, and 5-11 show the estimator results from the actuator propagation tests. Figure 5-9 shows the velocity performance in simulation, while Figures 5-10 and 5-11 show the performance in hardware. In simulation, the solid line represents the true velocity state, while the dashed line represents the estimated state. For the hardware, the true state is the measured velocity obtained by integrating the accelerometer measurements. When using the Object A model, the estimator does not know the module is attached, so it does not account for its added mass or thruster firings. From t=20 s to t=30 s, the assembler is firing thrusters. The estimator on the assembler accounts for the assembler's firing, but accounts only for assembler's mass, which is half of the total mass. Thus, the velocity estimate overshoots the true velocity. From t= 30 s to t= 40 s, the module is firing thrusters. The velocity estimate has a delay from following the true state in this phase because it does not incorporate the module's firings. It must rely on the ultrasound measurements to update the velocity state. These two issues are removed when using the Object A+C model. Therefore, the estimated velocity tracks significantly better when using the Object A+C model than the Object A model. The jump in the true Y velocity in Figure 5-10 reflects a additional force which appears after stiction is overcome when the satellites starts firing. The additional force could be due to a slope on the table or



(a) Object A Mass Properties
(b) Object A+C Mass Properties

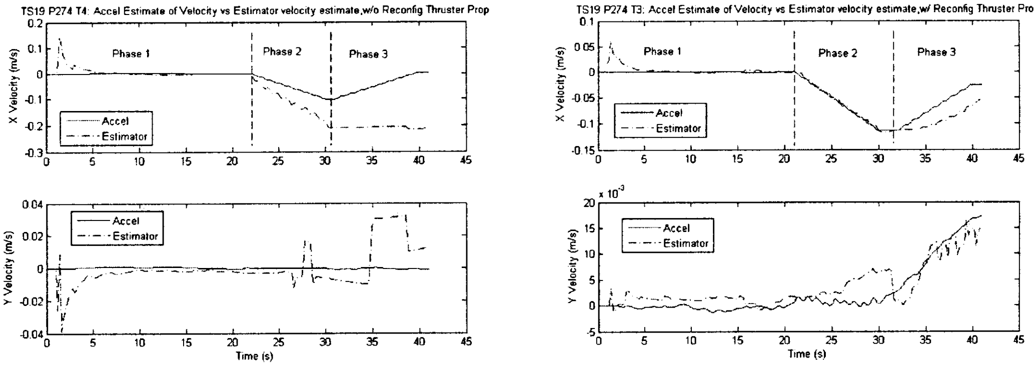
Figure 5-9: Estimator velocity performance with and without new mass and thruster properties, Simulation



(a) Object A Mass Properties
(b) Object A+C Mass Properties

Figure 5-10: Estimator velocity performance with and without new mass and thruster properties, Ground Hardware

small component of the thruster firings that is in the y direction. In general, updating the thruster propagation in the estimator improves the velocity estimate, which can improve fuel usage. In Figure 5-11, the EKF is seen to rely more on the model as opposed to the ultrasound measurements. This is reflected in the flat velocity in Phase 3. If the estimator relies more on the model than the measurements, the velocity would not change if there is no firing.



(a) Object A Mass Properties

(b) Object A+C Mass Properties

Figure 5-11: Estimator velocity performance with and without new mass and thruster properties, ISS hardware, Test Session 19 P274 T4 and T3

Analysis

The necessity of the parameterization of the actuator propagation is dependent on the ratio of propagation to measurement incorporation and the ratio of assembler mass to module mass. The following criteria should be considered when deciding if parameterization of the actuator propagation is necessary.

- Measurement incorporation occurs fast enough and is clean enough such that actuator propagation has little effect on the performance of the velocity estimate
- The module mass is significant fraction of the assembler-module system, upwards of 15%. Previous work has shown that small modules (5% of the assembler mass) have little impact on the control system performance [53].
- Additional module actuators impact forces onto the system that are greater than disturbance noise.

5.2.3 Measurement Incorporation

Measurement incorporation was performed for the Object A+C configuration to combine measurements from the module's ultrasound receivers. The initial attempt at incorporating the measurements was to communicate the filtered measurements from each beacon in real time from the module to the assembler, and then incorporate the received measurements into the assembler's estimator. However, due to the restrictive communication structure on SPHERES and how the estimator processes measurements as soon as they are received, this approach was determined to be impractical.

Experiment

In order to determine a realistic performance improvement due to combining sensor measurements, without being limited by specific hardware constraints, the approach used in this thesis is to post process raw measurement data obtained from the hardware into a Kalman filter. This approach has the benefit of using actual hardware data, so it provides a realistic level of sensor noise. Post processing can be performed for a variety of configurations using the same measurement set, allowing for a direct comparison between methods.

Accuracy of the estimator in determining the position of the center of the assembler was tested by varying the number of beacons used (3, 4, and 5) and receiver configurations (using receivers from either the assembler or both the assembler and module). The positioning of the satellites was such that the module was directly between the assembler and one of the beacons. Nominal operations use 5 beacons, while 3 beacons is the minimum needed for the state estimator to converge. The reason for testing convergence for 3, 4, and 5 beacons is to determine the blockage effect. The module satellite blocks the assembler's face that is directly pointing to Beacon 1 (Figure 5-12). Thus, when the assembler processes the measurements from Beacon 1, it must use measurements received from a side face. The performance in 3 beacon operations differs significantly based on which beacons are used compared to the location of the assembler. This Object A+C configuration is compared against the baseline Object A configuration.

The following three configurations were tested:

1. Object A: Assembler receivers used, all 24 receivers used

2. Object A+C: Assembler receivers used

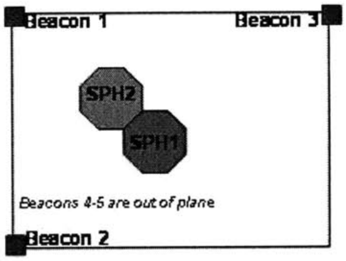


Figure 5-12: Setup for joint sensing tests

- 20 assembler receivers used, 4 assembler receivers blocked by module

3. Object A+C: Assembler and module receivers used

   - 20 assembler receivers used, 4 assembler receivers blocked by module
   - 20 module receivers used, 4 module receivers blocked by assembler

For each configuration, approximately 15 tests were run to collect raw measurement data. The raw measurements were post-processed for three cases: 3, 4 and 5 beacons. Then the average of the RMS error for each run was computed, then averged to provide an overall performance mean and standard deviation of the estimator for each case.

Results

Figures 5-13 and 5-14 show the performance of the position estimate when ultrasound receivers are incorporated from both assembler and module satellites compared to baseline performance. Figure 5-13 shows the simulation results, while Figure 5-14 shows the results of performance based on measurements taken on the hardware.

The simulation results (Figure 5-13) showed that for 4 and 5 beacons there is no noticeable difference between using 40 receivers and using 20 receivers. The marginal improvement in the 3 beacon case is still within one standard deviation, so no distinctive difference can be inferred between the two methods. Competing effects are in play that result in a minimal net benefit. The benefit to using more receivers is to obtain a better estimate by reducing the noise. However, after a threshold number of receivers, the improvement is minimal. The threshold value is dependent on the

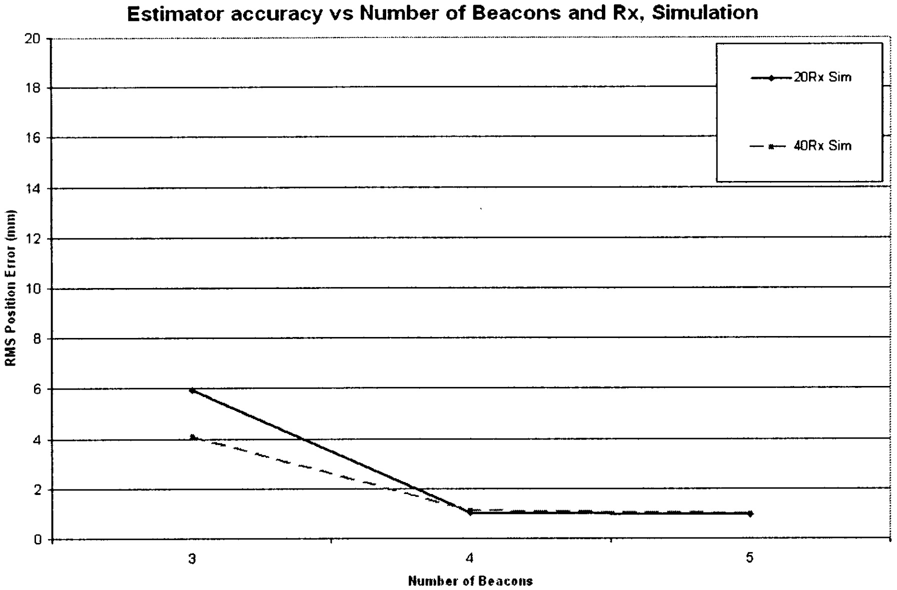


Figure 5-13: Results for Joint Sensing tests, Simulation

specific scenario. In this set-up, all measurements are being pre-filtered, so the cumulative effect of multiple receivers does not average in but results in a better single estimate. A factor causing a decrease in performance is the method for calculation of the estimated measurements. The EKF uses the estimated attitude to convert the receiver locations from the body frame to the inertial frame. The noise of the attitude estimate has a larger affect on the estimated measurement the farther the receivers are from the center of the satellite. The receivers from the assembler were located at a radius of 10 cm, while the modules's receivers were between 20 cm and 30 cm away from the center of the assembler. The combination of these two effects, among others, causes a negligible net performance change in simulation.

The trends are more distinct in hardware. The performance using 4 and 5 beacons is approximately the same. This trend reverses in 3 beacon operations, when using both assembler and module receivers performs better in hardware than using assembler receivers only. This performance is attributed to the body blockage issue that is not modeled in the simulation. The presence of the module satellite causes blockage

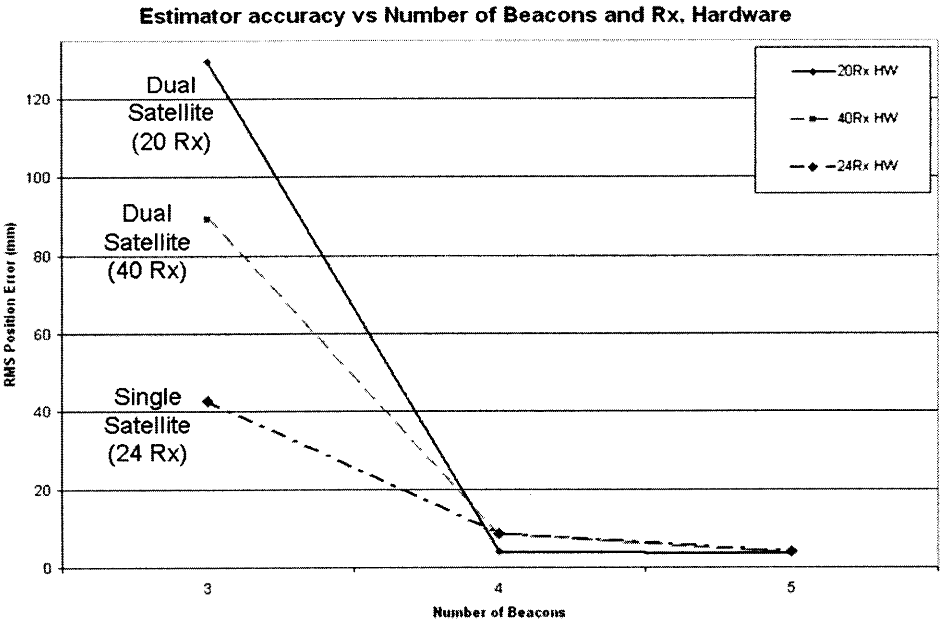


Figure 5-14: Results for Joint Sensing tests, Hardware

and slightly degraded measurements due to signals bouncing off of the module. The body blockage of the assembler to Beacon 1 is a bigger source of error when there are only 3 beacons instead of 4 or 5. When using the module's measurements, it can use the face directly facing the beacon, which leads to a better estimate.

However, the baseline performance is still better than the performance even when using all 40 receivers. The main source of error which causes the Object A+C configuration using Object A+C sensor model to not perform as well as the baseline configuration is most likely due to the variability of the attachment. The satellites are attached via Velcro, which can be a very imprecise mechanism. Discrepancies between assumed receiver locations on the module and the actual locations leads to an inaccurate model, and hence slightly worse estimator performance. This source of error can be minimized by using a rigid mechanical interface with high accuracy and repeatability of capture. A suitable replacement for the Velcro docking port is the Universal Docking Port described in Chapter 3.

Analysis

Given these results, the incorporation of additional sensors, particularly if they are the same type as those that already present, provide benefits if

1. the additional sensors cause a significant reduction in uncertainty,
2. the additional sensors replace sensors that can no longer be used due to blockage.
3. obtaining measurements from the additional sensors does not introduce a significant time delay
4. the attachment mechanism is sufficiently precise to allow for the incorporation of the module measurements while maintaining the estimator accuracy

5.2.4 Initialization

Determination of the proper initialization of the estimator is important after changing configurations. Using current information of state and covariance can help reduce convergence time and improve the accuracy of the estimate during the convergence period.

Experiment

The set up for the initialization tests used the Object A+C configuration with the satellites stationary in the center of the test volume. For each test, the test sequence started with a 10 s estimator initialization, followed by 10 s firing by Object A in the -X direction, 10 s firing by Object C in the +X direction, configuration switch from Object A estimator model to Object A+C estimator model, and repeat of the firing sequence. Even though the satellites were physically constrained to be stationary, the firing incorporates some uncertainty into the estimator that differentiates between the two models.

Three tests were performed which vary how the switch from the Object A model to the Object A+C model was executed.

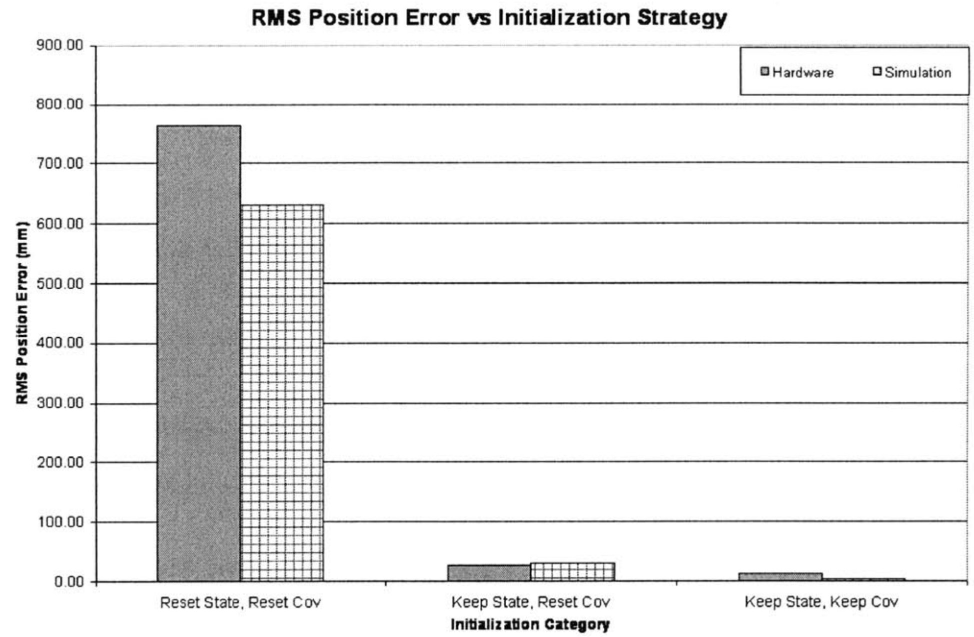


Figure 5-15: Hardware vs Simulation results of RMS position error during transition for estimator initialization strategies

1. Reset State, Reset Cov: The estimator was disabled, then re-initialized using default initialization values for the state and the covariance.

2. Keep State, Reset Cov: The estimator was disabled, then re-initialized using the last estimated state and a default covariance.

3. Keep State, Keep Cov: The estimator was disabled, then re-initialized using the last estimated values for the state and covariance.

The RMS error in estimated position when re-initializing the estimator is the metric of choice. Tests were performed in both simulation and hardware.

Results

Figure 5-15 shows the difference between the estimator initialization strategies for both hardware and software for RMS position error. Both hardware and simulation values match closely, indicating that the trend is valid.

Figure 5-15 confirms that resetting the state and covariance leads to a high RMS error during convergence. Keeping the state leads to a marked improvement of a approximately a factor of 16. The benefits to having an improved accuracy during convergence are related to safety. Though the assembler may not begin maneuvering until the estimator has converged with the new configuration, it is important to have accurate knowledge of where it is throughout the transition. The state estimate is used to monitor for collisions and to perform obstacle avoidance. The improved RMS accuracy allows the assembler to maintain obstacle avoidance during the transition.

Analysis

The selection of the appropriate initialization case depends on the motion of the assembler during the transition. For the initialization tests performed in this thesis, the satellites were mostly stationary during transition. Thus, the last estimated state and covariance are nearly identical to the current state and covariance after transition. For similar cases, where the motion during transition is small, Case 3 (Keep State, Keep Cov) should be used as it minimizes position error during transition and convergence time.

However, for assembly scenarios where large motion is expected during transition, Case 2 (Keep State, Reset Cov) is a safer choice. Case 2 still allows for the benefit of initialization the estimator from the last known state, but provides more time for convergence which might be needed if the large motion causes the assembler to be farther from the initialized location than expected.

It is generally always preferable to keep last estimated state, rather than discarding it to use a default value. Although Case 1 still works, it has significantly larger RMS position error during transition than Case 2 or Case 3. The difference in RMS position error depends on how far the default state used to initialize the estimator is from the actual state.

5.3 Controllers

Two controllers were implemented in this work, PID/PD control and adaptive control. Please note that some of the plots presented in this section are referenced in Section

Table 5.1: Attitude controller PD/PID gains for different configurations,$w_n = 0.4$

Configuration	Mass (kg)	Inertia (kg/m^2)	PD Gains		PID Gains		
Object A (ISS)	4.3	0.0225	0.0036	0.0135	0.00450	0.0001800	0.01434
Object A+C (ISS)	8.60	0.0450	0.00720	0.0270	0.009	0.00036	0.2869
Object A (GND)	12.43	0.067	0.01072	0.04020	0.01340	0.0005360	0.04271
Object A+C (GND)	24.86	0.134	0.02144	0.08040	0.02680	0.0010720	0.08543

Table 5.2: Position controller PD/PID gains for different configurations, $w_n = 0.2$

Configuration	Mass (kg)	Inertia (kg/m^2)	PD Gains		PID Gains		
Object A (ISS)	4.3	0.0225	0.0036	0.0135	0.00450	0.0001800	0.01434
Object A+C (ISS)	8.60	0.0450	0.00720	0.0270	0.009	0.00036	0.2869
Object A (GND)	12.43	0.067	0.01072	0.04020	0.01340	0.0005360	0.04271
Object A+C (GND)	24.86	0.134	0.02144	0.08040	0.02680	0.0010720	0.08543

5.4 to demonstrate the implementation of the control allocation.

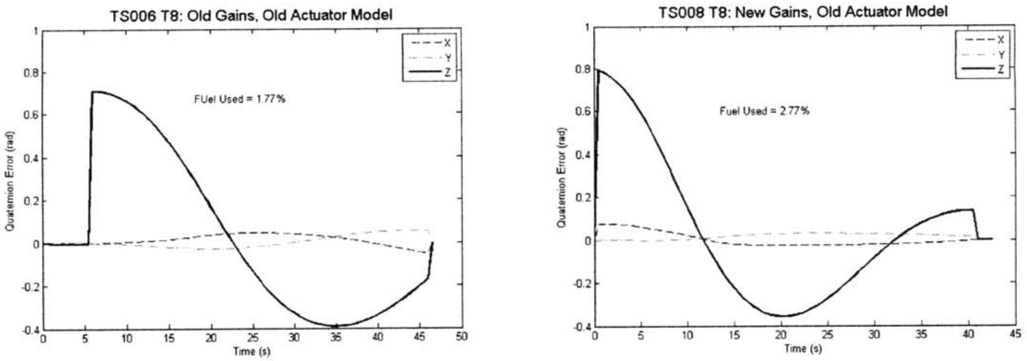
5.3.1 PD/PID Control

The PD/PID controller is tested under two types of target specifications: waypoint and trajectory specification. Waypoint specification provides a single target location and tests the step response of the system. The controller for waypoint targets is run at 1 Hz. Trajectory specification provides a target state as a function of time and tests the tracking performance of the system. The controller for trajectory targets is run at 2.5 Hz.

Waypoint Experiments

Table 5.1 and 5.2 list the attitude and position gains for the different configurations tested. The gains are calculated using Equations 4.32 and 4.33. The control bandwidth used to calculate the gains is $w_n = 0.4$ rad/s for attitude and $w_n = 0.2$ rad/s for position. The gains use unit damping ($\zeta = 1$) and an integration time constant of $\Lambda = 20$ s. The ground gains listed account for the SPHERES satellite as well as a single-puck air carriage on which the satellite floats on the 2D test table. The SPHERES satellite on its own, Object A only, is able to achieve $\pm 1^\circ$ in attitude and ± 2 cm in waypoint position control. Details for results of the Object A only configuration can be found in References [56] and [53].

The improvement due to the update in gains is determined by comparing two tests.



(a) Object A Gains

(b) Object A+C Gains

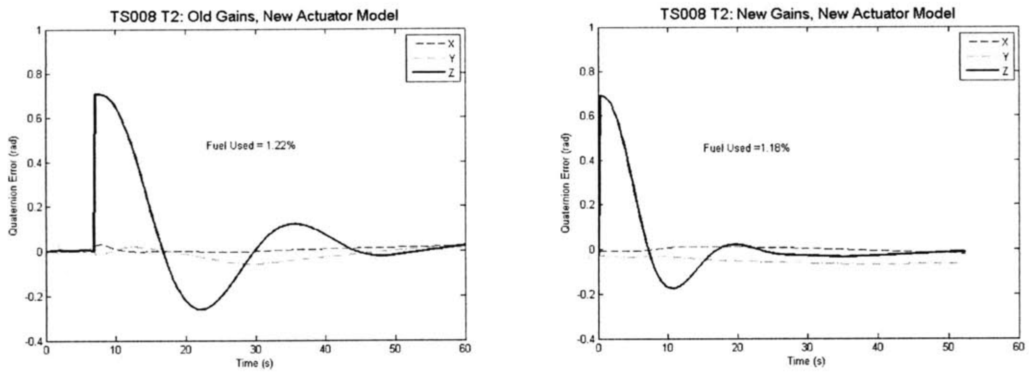
Figure 5-16: Attitude Error using Object A actuator model

The physical configuration used for both tests was the Object A+C configuration. In both tests, the satellites execute a 90^o rotation about the Z axis, shown in Figure 3-2, while maintaining attitude about the other two axes. In the first test, the attitude gains used are those calculated for the Object A configuration. In the second test, the attitude gains used are those calculated for the Object A+C configuration.

Waypoint Results

Figure 5-16a shows the results of the attitude error versus time, when using the Object A thruster configuration. At time t = 5 s, the satellites begin the rotation, after estimator initialization. The attitude in X and Y are nominally zero, and Figure 5-16a shows that the satellites are maintaining that angle. Results show a percent overshoot of 52 % (47^o) in Z and a fuel usage of 1.77% when using Object A gains.

The second test uses the gains calculated for the Object A+C configuration. Figure 5-16b shows the performance of the rotation with Object A+C gains, when only using the thrusters on Object A. As can be seen in Figure 5-16b, the percent overshoot decreases to 45% (41^o). Also, the settling time decreases by approximately 15 s from Figure 5-16a. However, the overall performance is not desirable because it does not achieve the desired attitude within the time specified. This poor performance can be attributed to the location of the center of mass outside of the thruster envelope. When two satellites are attached, the center of mass is located at the attachment



(a) Object A Gains

(b) Object A+C Gains

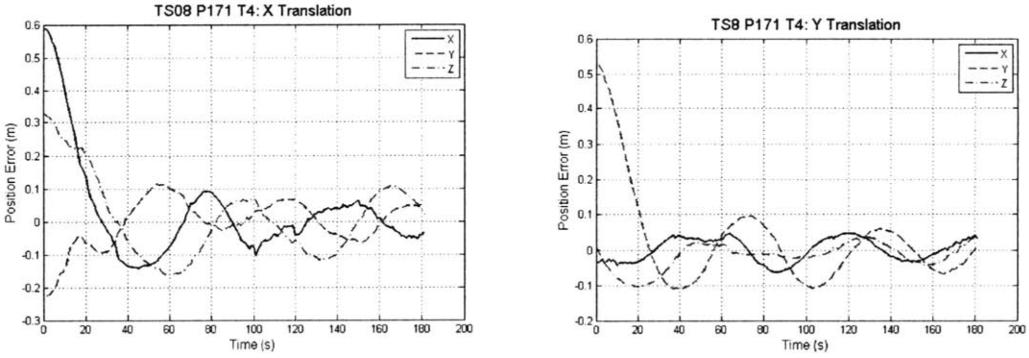
Figure 5-17: Attitude Error using Object A+C actuator model

point. If using only one satellite's thruster, the center of mass is outside the thruster envelope. Thus, the torque capability degrades depending on how the center of mass shifts.

Figure 5-17 shows the results for Object A gains versus Object A+C gains when using the Object A+C actuation model, which gives a better indication of the performance improvement. The percent overshoot decreases from 33% (30^o) in the Object A gain case to 22% (20^o) in the Object A+C gain case. Also, the settling time decreases from 40 s in the Object A gain case to 20 s in the Object A+C gain case. Finally, the overall performance improvement is seen through the decrease in fuel usage from 1.22% to 1.18% of a tank.

Position control was also demonstrated when two satellites were attached, and both satellites actively firing thrusters. A test was performed in SPHERES Test Session 8 that consisted of two single axis translations. Figure 5-18 shows the error between the desired waypoint and state for the two single axis translations. The performance showed the settling time of approximately 80 s, and an error performance of ±10 cm.

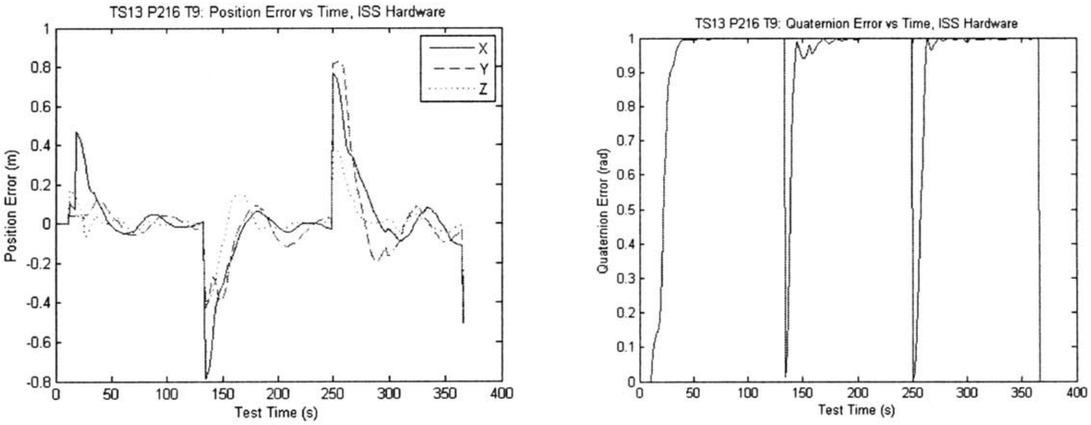
A multi-target test was performed (Test Session 13) in Object A+C configuration, where the satellites performed joint position and attitude control. The satellites were commanded to maneuver to three location targets, during the course of the test. Figure 5-19 shows the error performance for a translation in all three axes



(a) X Translation

(b) Y Translation

Figure 5-18: Single axis translation tests



(a) Position Error

(b) Attitude Error

Figure 5-19: Error for Object A+C configuration multi-target test, ISS hardware

simultaneously. Figure 5-19 shows the position and attitude error for the multi-target test. The spikes in the position and attitude errors indicate when the location target changed. The attitude performance is very good, since the quaternion error stays close to one, indicating zero total attitude error below 5^o. The rise and settling times are also quite small, indicating good performance. The position performance is acceptable, though not quite as good as the baseline performance.

Waypoint Analysis

The performance of Object A+C when using the proper model, both gains and actuator model, is worse than Object A waypoint control performance achievable. Sources

Table 5.3: Attitude controller PD/PID gains for different configurations,$w_n = 0.6$

Configuration	Mass (kg)	Inertia (kg/m^2)	PD Gains		PID Gains		
Object A (ISS)	4.3	0.0225	0.0036	0.0135	0.00450	0.0001800	0.01434
Object A+C (ISS)	8.60	0.0450	0.00720	0.0270	0.009	0.00036	0.2869
Object A (GND)	12.43	0.067	0.01072	0.04020	0.01340	0.0005360	0.04271
Object A+C (GND)	24.86	0.134	0.02144	0.08040	0.02680	0.0010720	0.08543

Table 5.4: Position controller PD/PID gains for different configurations, $w_n = 0.8$

Configuration	Mass (kg)	Inertia (kg/m^2)	PD Gains		PID Gains		
Object A (ISS)	4.3	0.0225	0.0036	0.0135	0.00450	0.0001800	0.01434
Object A+C (ISS)	8.60	0.0450	0.00720	0.0270	0.009	0.00036	0.2869
Object A (GND)	12.43	0.067	0.01072	0.04020	0.01340	0.0005360	0.04271
Object A+C (GND)	24.86	0.134	0.02144	0.08040	0.02680	0.0010720	0.08543

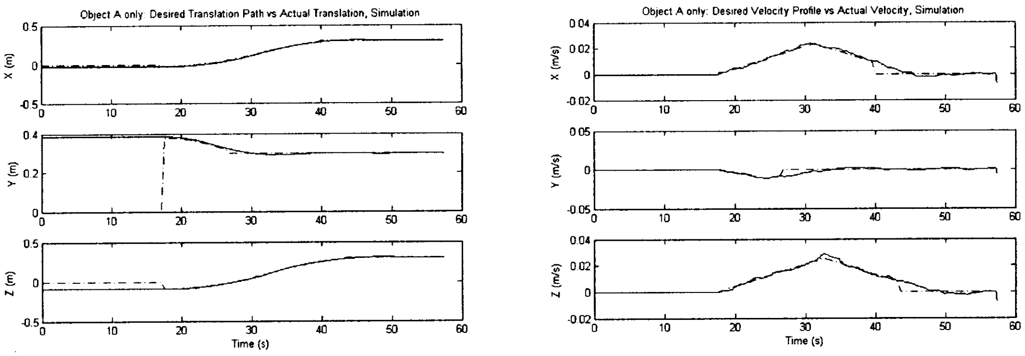
of error in this test include use of an estimator with a single satellite model, delay in the thruster firings on the module satellite, unmodeled damping in the velcro attachment, and inaccuracies in the mass properties modeled. The use of the Object A model in the estimator has an important effect, particularly on the velocity estimate which is not measured directly. Propagating the thruster firings with a Object A model would cause the estimator to over estimate the velocity estimate. The controller would cause repeated overshoot, as is present in Figure 5-18.

A second minor source of error is the delay in thruster firings caused by communication delay and commanding. Due to limited communication bandwidth, only the thruster durations were communicated to the module satellite. A small error is caused by not centering the thruster pulse in the firing window. Also, the velcro attachment is assumed to be a rigid attachment in the controller design. The nonlinear effects of the velcro attachment could cause a small amount of damping, which reduces the overall force imparted onto the satellites.

Trajectory Experiments

The PD/PID controller was run at 2.5 Hz when using trajectory targets to improve tracking performance on the ground flat table testing environment. Tables 5.3 and 5.4 give the attitude and position gains for the trajectory tests. The gains were calculated using a control bandwidth of $w_n = 0.6$ rad/s for attitude and $w_n = 0.8$ rad/s for position.

The trajectory tests performed have the satellites move to a desired waypoint,



(a) Position Tracking

(b) Velocity Tracking

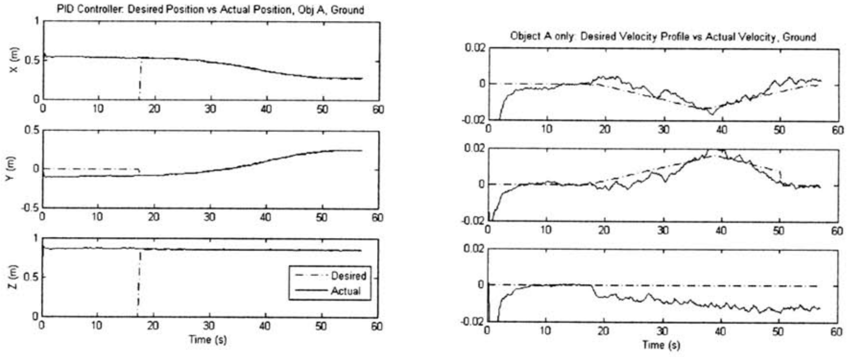
Figure 5-20: Object A PID Trajectory Tracking Performance, Simulation

using a planned trajectory given in position, velocity, and acceleration. A feedforward term, the desired acceleration times the configuration mass, is included in the controller to account for the acceleration profile. The attitude is maintained for all tests.

Trajectory Results

Figure 5-20 shows the simulation tracking performance of the Object A configuration. Figure 5-20a shows the position tracking performance, while Figure 5-20b shows the velocity tracking performance. As seen in the plots, the controller tracks very well, under 1 mm tracking error. Figure 5-21 shows the corresponding tracking performance on the ground hardware testbed. Figure 5-22 shows the error performance on the ground hardware, in the presence of disturbances such as friction and slopes on the test table. The controller maintains error to 1 cm or less during the trajectory tracking on the ground hardware. This test provides the baseline performance against which to compare the other configurations.

The test was repeated for the Object A+C configuration. Figure 5-23 shows the tracking performance in simulation, while Figures 5-24 and 5-25 show the performance on the ground hardware. As seen in Figure 5-24a, the position tracks fairly well during the maneuvering phase. However, once the satellites have reached the end of the trajectory, the tracking error increases as the controller is not able to accommodate



(a) Position Tracking

(b) Velocity Tracking

Figure 5-21: Object A PID Trajectory Tracking Performance, Ground Hardware

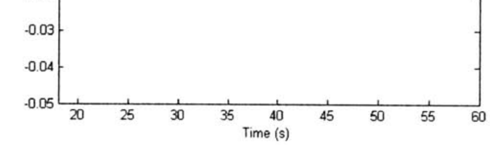
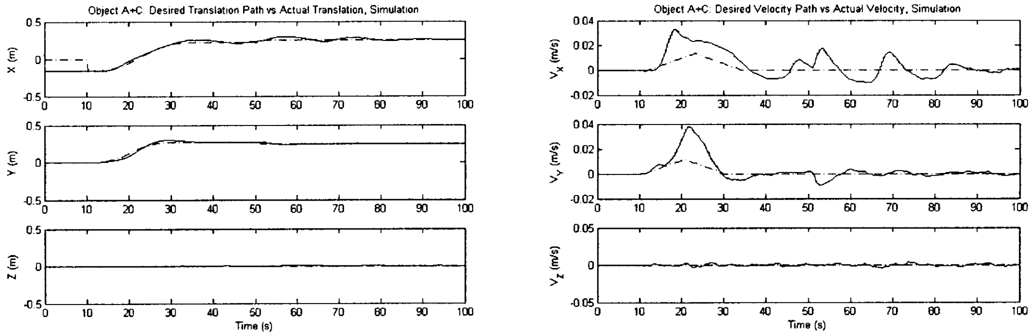


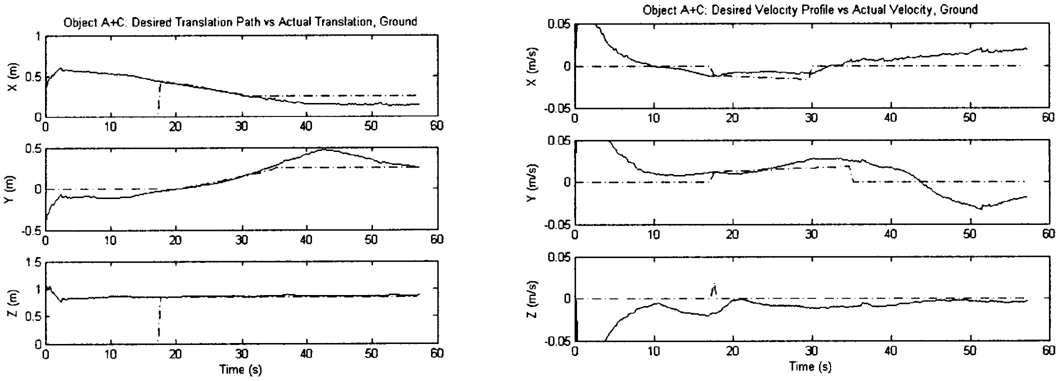
Figure 5-22: Object A PID Trajectory Error Performance, Ground Hardware



(a) Position Tracking

(b) Velocity Tracking

Figure 5-23: Object A+C PID Trajectory Tracking Performance, Simulation



(a) Position Tracking

(b) Velocity Tracking

Figure 5-24: Object A+C PID Trajectory Tracking Performance, Ground Hardware

for the disturbances on the table, given the momentum of the attached satellites.

Trajectory Analysis

In general, the tracking performance is achievable in both configurations. The performance can be tuned in three ways. First, the plan can be updated for this configuration so the maximum velocity is slower, since the thruster force remains the same but mass doubles. Second, the bandwidth can be updated so that the tracking phase and position maintenance phase have different control bandwidths selected based on the expected maneuvering in that phase. Third, the selection of duty cycle and control period may be dependent on the configuration, based on the disturbances. The quantitative effect of the table disturbances can be confirmed by the successful completion

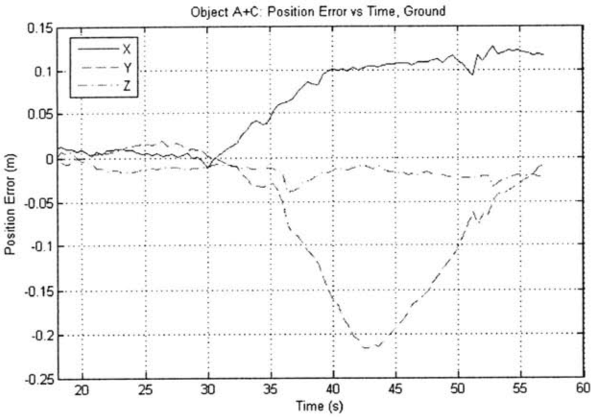


Figure 5-25: Object A+C PID Trajectory Error Performance, Ground Hardware

of these tests on the ISS and the comparison of performance, since the ISS testing environment has minimal disturbances. The tests are scheduled to be run in Test Session 22, sometime in the spring of 2010.

5.3.2 Adaptive Control

The adaptive controller implemented is a direct adaptive control algorithm. As specified in Chapter 4, this controller is based on an adaptive tracking controller developed by Niemeyer and Slotine [55]. A 2D baseline version of this algorithm was implemented by Katz [42] for the specified flexible module set-up in simulation, but not tested on hardware. This work updates the algorithm to 3D and parameterizes the algorithm to be reconfigurable for multiple configurations.

Experiment

The first parameterization is in the specification of the state length. This parameter sets the number of variables and indicates the types of objects present. The state length is given by

$$l_{state} = 6 + 3\varphi_B + 0\varphi_C \tag{5.3}$$

Object C does not add in any additional parameters because its state can be fully determined from the state of Object A and B.

The next step is to list the force directions affected for each state variable. The axis direction matrix is given by Equation 5.4, where the last three rows corresponding to the axis direction of Object B. The bottom three rows are only used if the state length is initialized to include Object B. The direction matrix is linked to the state vector (position, attitude, and deflection), as specified in Chapter 4.

$$d = \begin{bmatrix} & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \\ 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ \varphi_B & & & & & \\ & \varphi_B & & & & \\ & & \varphi_B & & & \end{bmatrix} \tag{5.4}$$

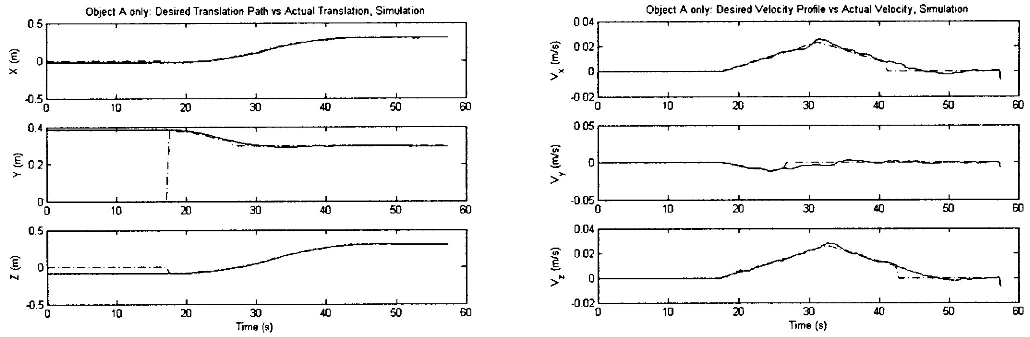
The third step parameterization occurs in specification of which of the parameter vector indices to use in the force calculation. For states 1 through 6, the mass properties for Object A should be used. If the state is greater than 6, the mass properties for Object B should be used.

The final step is the population of the estimated parameter vector $\hat{a}$ to include the mass properties for the configuration.

Table 5.5 specifies the adaptive controller gains used for each of the different configurations. The adaptive controller was tested for the SPHERES satellite only configuration (Object A only) in order to form a baseline tracking performance. The same adaptive controller was tested in the Object A+C configuration, both in simulation and hardware. Neither the controller form nor gains changed, only the initialization of the parameter vector a.

Table 5.5: Adaptive controller gains for different configurations

Configuration	Mass (kg)	Inertia (kg/m^2)	Position Gains		Attitude Gains		Adaptation Gain
Object A (ISS)	4.3	0.0225	6.5	2.0	0.4	0.25	1
Object A+C (ISS)	8.60	0.0450	6.5	2.0	0.4	0.25	1
Object A (LAB)	12.43	0.067	6.5	2.0	0.4	0.25	1
Object A+C (LAB)	24.86	0.134	6.5	2.0	0.4	0.25	1



(a) Position Tracking

(b) Velocity Tracking

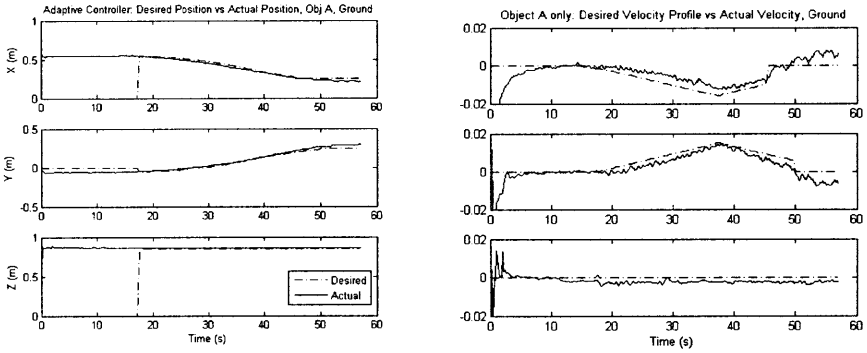
Figure 5-26: Object A Adaptive Control Trajectory Tracking Performance, Simulation

Results

Figure 5-26 shows the performance of Object A in simulation and Figure 5-27 shows the performance in hardware. Simulation results show tracking to ±1 mm, while hardware results show tracking error less than ±5 cm.

Figures 5-29 and 5-30 show the results for Object A+C configuration position tracking. Figure 5-31 shows that the tracking error for Object A+C also is less than ±5 cm. The adaptive control tracking performance for Object A+C is comparable to the tracking performance seen for Object A. This demonstrates that the controller maintains the same level of performance for both configurations.

This adaptive controller was run in simulation for a transverse translation maneuver for Object A+B+C configuration. Figure 5-32 shows the position tracking results of simulated position (solid) versus desired position (dashed). In Figure 5-32a, the adaptive controller is set to Object A only configuration, while Figure 5-32b has the Object A+B+C adaptive controller. Thus, one can see the distinct tracking improvement when using the controller designed for that configuration.



(a) Position Tracking

(b) Velocity Tracking

Figure 5-27: Object A Adaptive Control Trajectory Tracking Performance, Ground Hardware

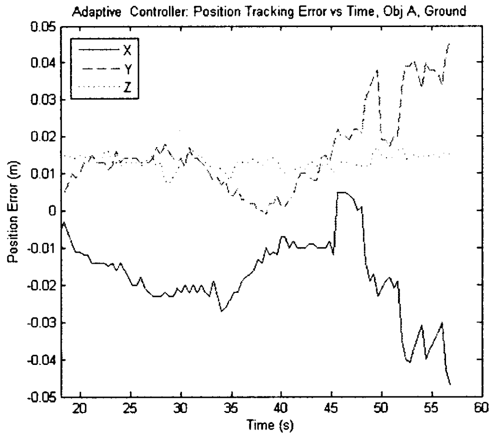
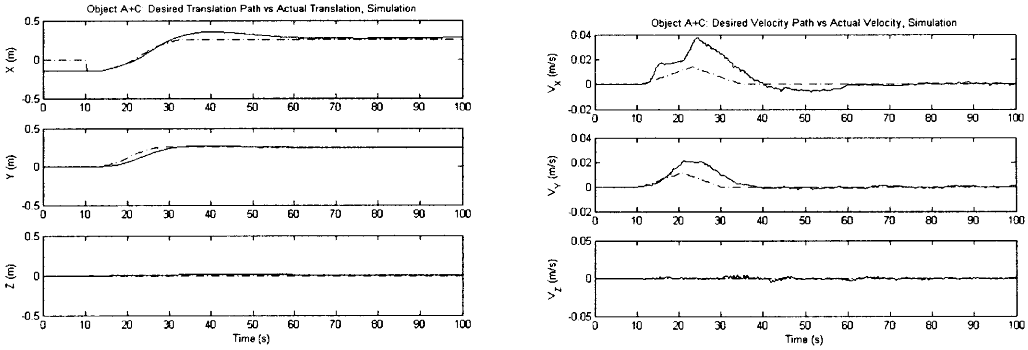


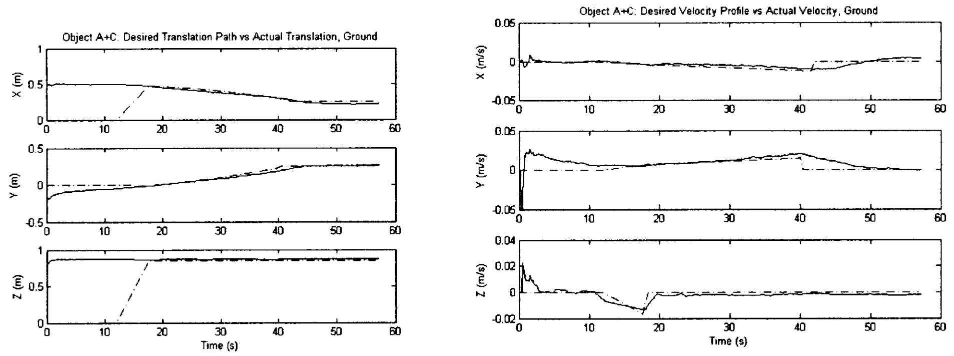
Figure 5-28: Object A Adaptive Control Trajectory Error Performance, Ground Hardware



(a) Position Tracking

(b) Velocity Tracking

Figure 5-29: Object A+C Adaptive Control Trajectory Tracking Performance, Simulation



(a) Position Tracking

(b) Velocity Tracking

Figure 5-30: Object A+C Adaptive Control Trajectory Tracking Performance, Ground Hardware

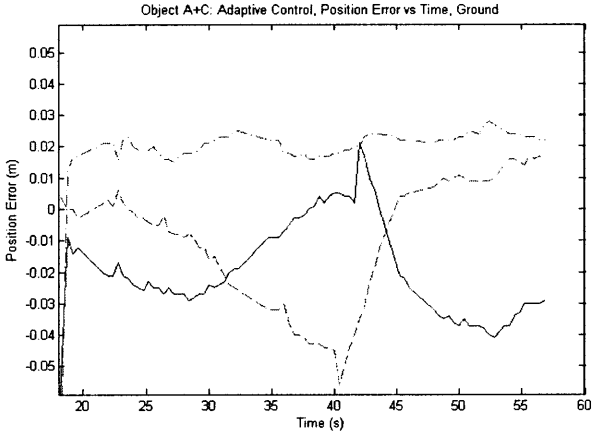
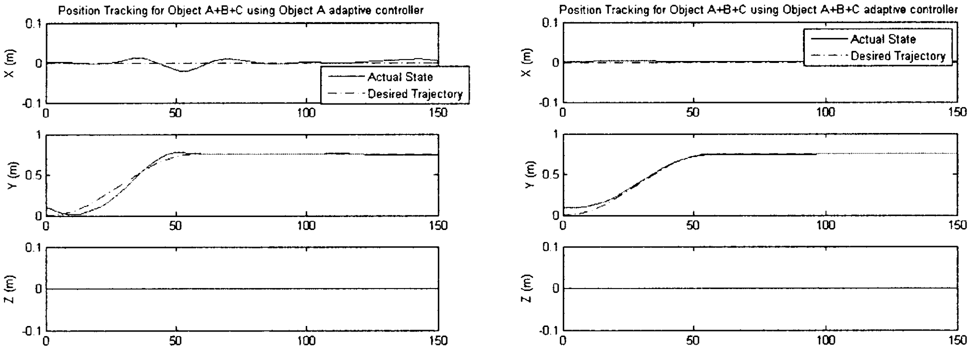


Figure 5-31: Object A+C Adaptive Control Trajectory Error Performance, Ground Hardware



(a) Object A controller

(b) Object A+B+C controller

Figure 5-32: Estimated State vs Desired State for SWARM + Beam (Object A+B+C) using an adaptive controller, Simulation

Analysis

Results presented in this section show good tracking performance when using the adaptive controller for Object A, Object A+C, and Object A+B+C configurations. The maintenance of the tracking error under all three configurations demonstrates the successful parameterization of the adaptive controller. The adaptive controller is naturally suited to online autonomous model generation with the specification of $\hat{a}$. Online Model Calculation specifies the layout of masses and initial guesses for $\hat{a}$. Adaptation improves $\hat{a}$ to achieve trajectory tracking. Online Model Calculation enables adaptive controller to work with varying configurations. Because of the inherent adaptation, the same set of gains could be used for all configurations, which simplifies the parameterization.

5.3.3 Comparison between PD/PID and Adaptive control

Table 5.6 shows the RMS tracking error for each configuration for PID versus Adaptive control. For the Object A configuration, the adaptive controller performs slightly worse than the PID controller, shown in Figures 5-20 to 5-22. For the Object A+C configuration, the adaptive controller shows slightly better performance than the PID controller shown in Figure 5-25.

This performance improvement in the Object A+C configuration can be attributed to the adaptive controller's ability to adjust the inherent model based on the tracking performance. This has the effect of partially accounting for unmodeled mass properties or disturbances. This allows the adaptive controller to maintain the same level of performance for both configurations. The overall performance can then be tuned for the desired performance specifications. The Object A PID control gains have been significantly tuned over the past 5 years. However, only minimal testing was performed to select the adaptive controller gains. Thus, it is likely that the proper selection of gains enables the adaptive controller to match the PID control performance.

Table 5.6: RMS Position Tracking Error Performance in meters between PID and Adaptive Controllers for Object A and A+C configurations

Configuration	PID	AC
Object A	0.0064	0.024
Object A+C	0.0954	0.0337

5.4 Control Allocation

The control allocation algorithm on SPHERES is a pulse width modulation scheme based on the thruster geometry. The control allocation algorithm is updated to use the additional thrusters that are external to the assembler satellite. These additional thrusters could be on a module satellite or a propulsion module.

Experiment

Figure 5-33 shows the differences between the implemented baseline and reconfigurable control allocation algorithms. The variables highlighted in red bold are properties that are obtained through the property vector. The reconfigurable version of the control allocation for Object A+C accounts for the addition of 12 more thrusters and the change of the center of mass of the system. Due to the addition of 12 thrusters, the system now has redundant thrusters and blocked thrusters. A variable (Act_{health}) is introduced to allow for the selection of particular thrusters out of the complete set. For this work, the thrusters on the attachment face are disabled. Of the redundant thrusters, the heuristic that is chosen to select the thrusters is the selection of the thrusters furthest from the center of mass to allow for the most torque.

Two tests were performed to compare the control performance when a large module is attached. Object A+C configuration was used for the first test. The assembler satellite was actively maneuvering, while the module satellite was a proof mass and did not actuate during the test. The satellites execute a 90^o rotation about the Z axis. The attitude is maintained about the other two axes. The Object A+C configuration was also used for the second test, but actuators were able to be used on both satellites. Thruster commands were calculated on the assembler satellite, then communicated

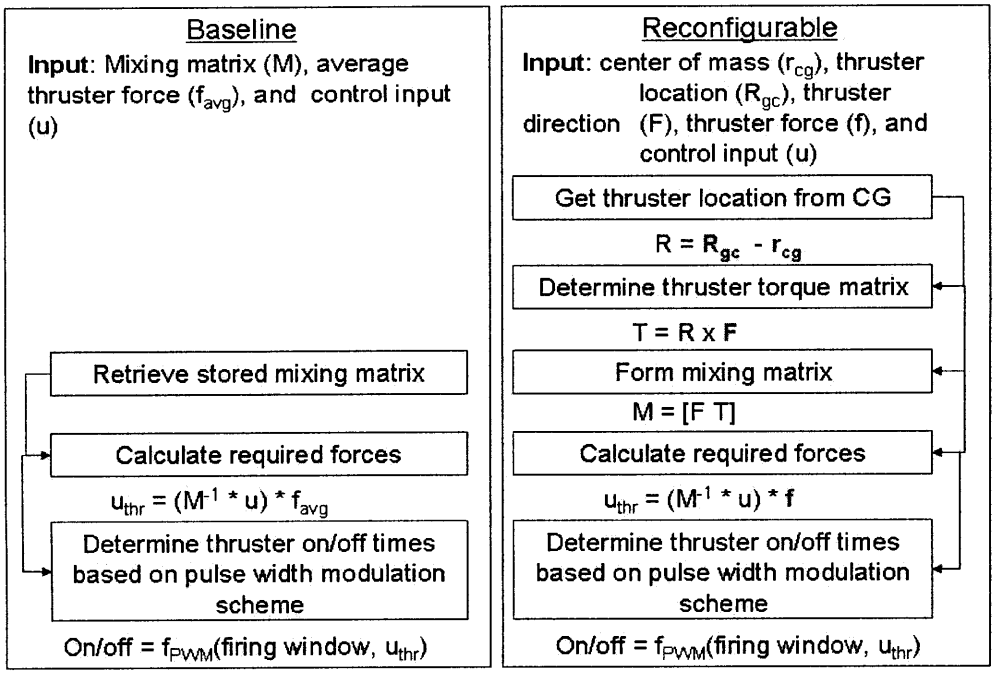

Baseline
Input: Mixing matrix (M), average thruster force (f_{avg}), and control input (u)
Retrieve stored mixing matrix
Calculate required forces
$u_{thr} = (M^{-1} * u) * f_{avg}$
Determine thruster on/off times based on pulse width modulation scheme
On/off = f_{PWM}(firing window, u_{thr})
Reconfigurable
Input: center of mass (r_{cg}), thruster location (R_{gc}), thruster direction (F), thruster force (f), and control input (u)
Get thruster location from CG
$R = R_{gc} - r_{cg}$
Determine thruster torque matrix
T = R x F
Form mixing matrix
M = [F T]
Calculate required forces
$u_{thr} = (M^{-1} * u) * f$
Determine thruster on/off times based on pulse width modulation scheme
On/off = f_{PWM}(firing window, u_{thr})

Figure 5-33: Implementation of Baseline versus Reconfigurable Control Allocation

to the module satellite. The test maneuvering is identical to the previous test, where the satellites execute a 90° rotation about the Z axis, while maintaining attitude in the other two axes. This updated thruster geometry leads to a uniform distribution of thrusters around the new center of mass.

Results

Figure 5-16b shows the results for the attitude error using only the assembler's thrusters. The overshoot for the single satellite configuration is roughly 41°. The satellites did not settle to the desired attitude during the 45 s allotted for the maneuver. This reflects that even though the gains are updated for the mass, the performance degradation is caused by the fact that the center of mass is outside of the thruster envelope. Use of the thrusters on the attachment face are not permitted so the system is unstable and uncontrollable along the direction of attachment.

Figure 5-17b shows the attitude error plotted versus test time when using the gains calculated for the Object A+C configuration for the ISS environment. The overshoot in Figure 5-17b is about 20°, which is a factor of two improvement over

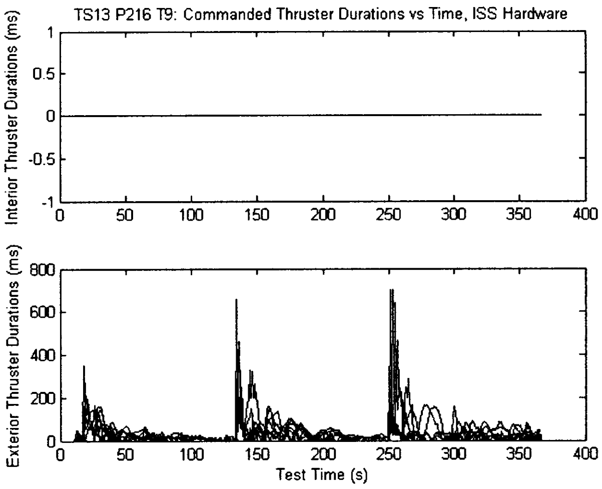


Figure 5-34: Commanded thruster durations for Object A+C configuration multi-target test, ISS hardware

Figure 5-16b with the Object A actuator model. Also, in this case, the satellites were able to converge to the desired attitude, with a settling time of 20 s.

The thruster commands for the translational joint maneuvering test shown in Figure 5-19 are shown in Figure 5-34. It is seen that the control allocation algorithm does not command the interior thrusters, located on the attachment face. This is as expected for this configuration because the interior thrusters do not provide any thrust due to plume impingement from the attached satellite. The exterior thrusters are commanded with an initial large pulse, consistent with a high initial position error (corresponding to a large control input), and decreasing as the position error decreases. Thruster saturation is an important issue to track because it leads to decreased performance due to loss of desired actuation. Saturation would appear as a flat top to the thruster firings. The peaks of the thruster firings in Figure 5-34 are all rounded. Thus, the fact that the thrusters are not saturated in Figure 5-34 demonstrates proper selection of the duty cycle and control period.

This control allocation was also used for controller testing for the Object A+B+C configuration in Katz thesis [42] to account for the SWARM propulsion module thrusters. Figure 5-35 shows the performance of three different controllers using the same control allocation algorithm. Figure 5-35 plots (top) the desired attitude

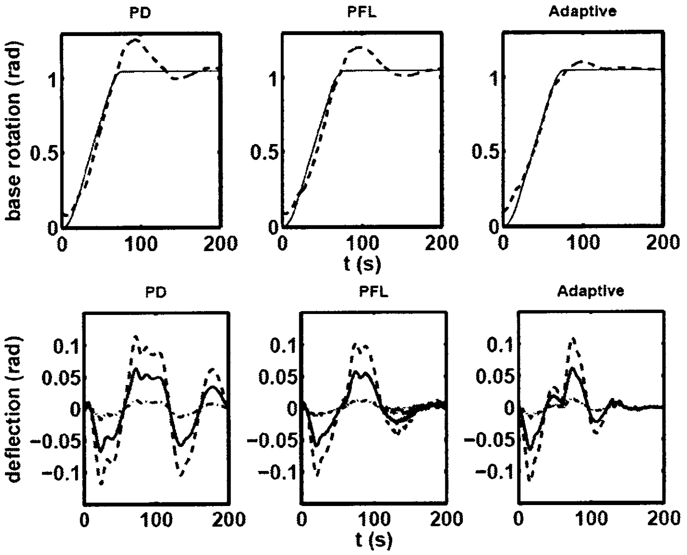


Figure 5-35: Performance of three controllers for Object A+B+C configuration using the parameterized control allocation algorithm

versus actual attitude for a rotation of 90°, and (bottom) the corresponding deflections for each link in the beam while performing the rotation. The successful rotation tracking, while maintaining deflection under 0.1 rad, demonstrates proper control. The successful control implies the successful implementation of the control allocation and demonstrates its proper conversion of control input to thruster commands for a variety of controllers.

Analysis

The successful parameterization of the control allocation algorithm was demonstrated for Object A+C and Object A+B+C configurations. Object A+C configuration testing accounted for Object C being both active and passive. Communication was used to transmit thruster commands from the assembler to the module. The control allocation algorithm does not quantitatively account for the communication delay in sending the packets. It is assumed that the communication delay is small. In microgravity conditions, this assumption is valid since there is minimal motion during the few milliseconds of communication transmission. The impact is greater on ground tests because friction and slopes on the table lead to larger disturbances during that communication transmission.

Communication delay is not an issue for the Object A+B+C configuration, because commands are transmitted through the expansion port connection. The pulse width modulation scheme used could present issues when actuating flexible dynamics. The successful damping of the deflection also shows that this control allocation algorithm can also be used for flexible module, given the proper control frequency selection. The control frequency should be selected to be faster than the vibrational dynamics associated with the flexible module to be able to control it.

5.5 Integrated Tests

The demonstration of the Online Model Calculation design is contingent on the implementation of both the framework (proper transmission and model calculation based on p) as well as the parameterized control algorithms. Two types of integrated tests were conducted to demonstrate the model generation architecture end-to-end. The first type of test is in support of autonomous assembly. These tests validate that the Online Model Calculation design developed successfully works for the application envisioned. The second type of tests is a remote control test. The successful implementation of the Online Model Calculation design on the remote control tests demonstrate that the framework and algorithms can be extended to non-assembly applications. Thus, the Online Model Calculation design can be included in the spectrum as a valid model generation architecture because it is not just applicable for autonomous assembly applications.

5.5.1 Docking and Assembly

Many iterations of docking and assembly tests were coded and implemented as part of the testing process. Two key tests are discussed here. The first test is docking when attached to the flexible beam. The second test is an assembly maneuvering for a single rigid module.

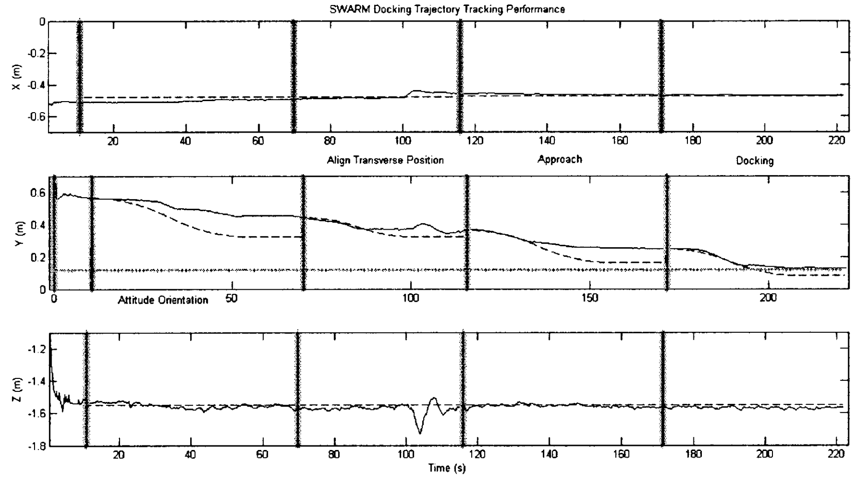
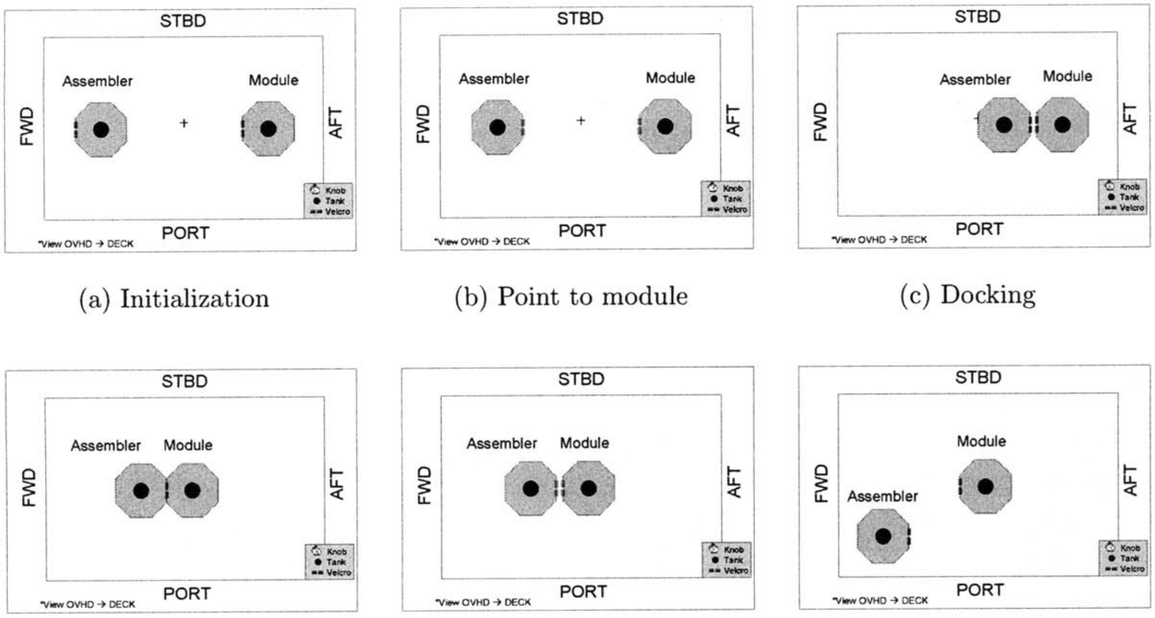


Figure 5-36: Object A+B+C maneuvering and docking to a fixed structure, Ground Hardware

Docking

Figure 5-36 shows the adaptive control tracking performance of Object A+B+C maneuvering to dock to a fixed structure. The docking occurs along the Y axis. The goal position that indicates docking is depicted by horizontal bar. The docking occurs as a sequence of several component maneuvers. The changes between maneuvers are depicted by the vertical bars. The test had several maneuvers: estimator initialization, attitude orientation, alignment in the transverse direction, approach, and docking. The areas where the actual state ceased to follow the trajectory and remained stationary were periods when the satellite's air carriage got stuck on the flat floor. This problem was remedied at the start of the next maneuver due to the large thruster pulse at the beginning of a maneuver as part of the bang-bang path planner. This test resulted in a successful docking of the free end of the flexible beam, indicating that the adaptive controller was successful in maintaining accuracy in position and attitude, as well as the beam deflection. The adaptive controller used in this test was 2D sliding mode adaptive controller, developed by Katz [42], a precursor to the direct adaptive controller.



(a) Initialization

(b) Point to module

(c) Docking

(d) Joint Maneuvering

(e) Undock

(f) Maneuver to Next module

Figure 5-37: Schematic of Assembly Maneuver Sequence

Assembly Maneuver

A full assembly maneuver was performed with a rigid module. The assembly maneuver test consisted of three main phases. Phase one involves an assembler satellite docking to a module satellite. The sub-phases for the docking are the same as those described in the previous section. In Phase two, both satellites are attached and maneuvering jointly to the center of the test volume. This phase mimics the movement of a module from its initial location to its final location in the assembled structure. Finally, in Phase three, the assembler satellite detaches from the module satellite and moves to another location, as if in preparation to dock to another module satellite. Figure 5-37 shows a schematic representation of the maneuvering of the satellite.

Figures 5-38 through 5-42 show the performance of the assembly maneuver. Figure 5-38 successfully shows the general execution of the assembly sequence and the framework of the Online Model Calculation design. Figure 5-38 shows the trajectory tracking performance, where the solid line is the actual trajectory and the dashed line is the desired trajectory. At each phase change, the property structure was

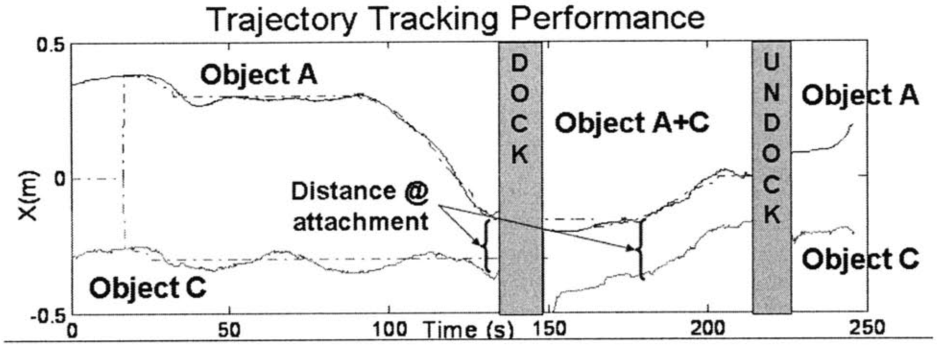


Figure 5-38: Trajectory Tracking Performance for assembler and module, Assembly Maneuver, Hardware

successfully transmitted and the new model calculated and set. Otherwise, the desired motion of the next phase would not occur successfully. Trajectory tracking performance is maintained in each configuration. Object A+C is an uncontrollable configuration if using the Object A model. Thus, successful trajectory tracking after docking demonstrates a successful model update.

Figure 5-39 shows the relative position between the assembler and the module in simulation. During the docking phase, the relative position decreases according to the desired trajectory profile until it reaches 0.20 m separation. At this distance, the two satellites are docked. This relative position is maintained throughout the joint maneuvering phase, since the two satellites remain attached. Once the assembler undocks, the relative position increases again as the assembler performs an open loop maneuver to push back. Figure 5-40 shows nearly identical, though noisier, behavior for the ground testing. Figures 5-39 and 5-40 show that the relative attitude is consistently maintained throughout the maneuvering, such that the satellites are pointing their docking faces (-X Velcro faces) at each other.

Figure 5-41 shows the trajectory tracking error for the assembly maneuver, in all three phases. A PD/PID controller was used during this test to follow the Bang-Bang trajectory. The maintenance of the tracking error in simulation was below 2 cm and approximately 5 cm in ground hardware. The maintenance of the same tracking performance throughout the different configurations demonstrates the successful parameterization of the control algorithms. Though the error is greater in the hardware

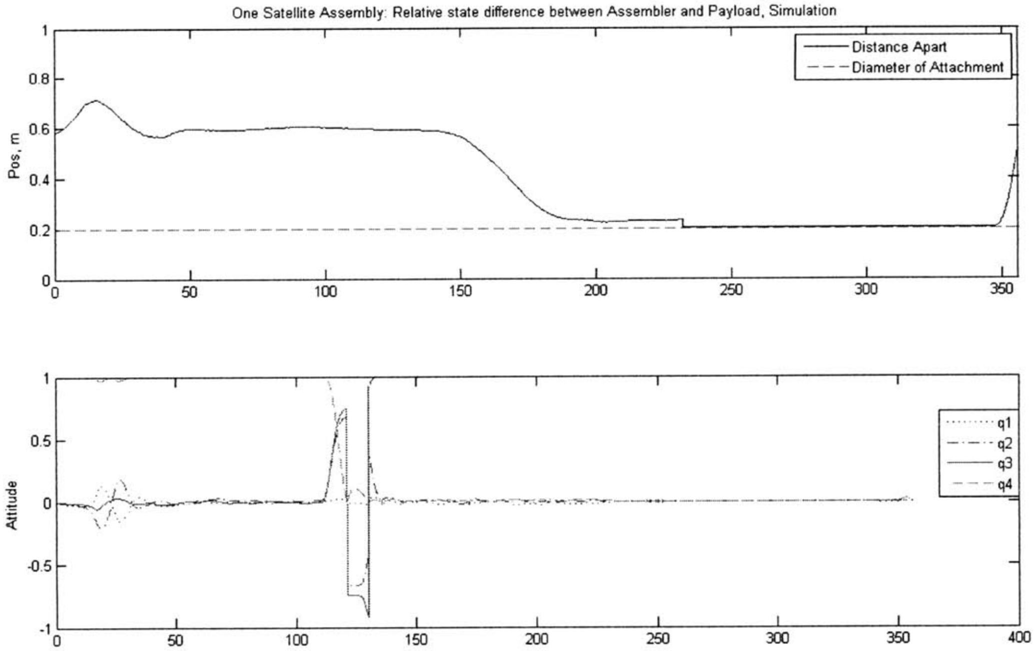


Figure 5-39: Relative Position Between assembler and module, Assembly Maneuver, Simulation

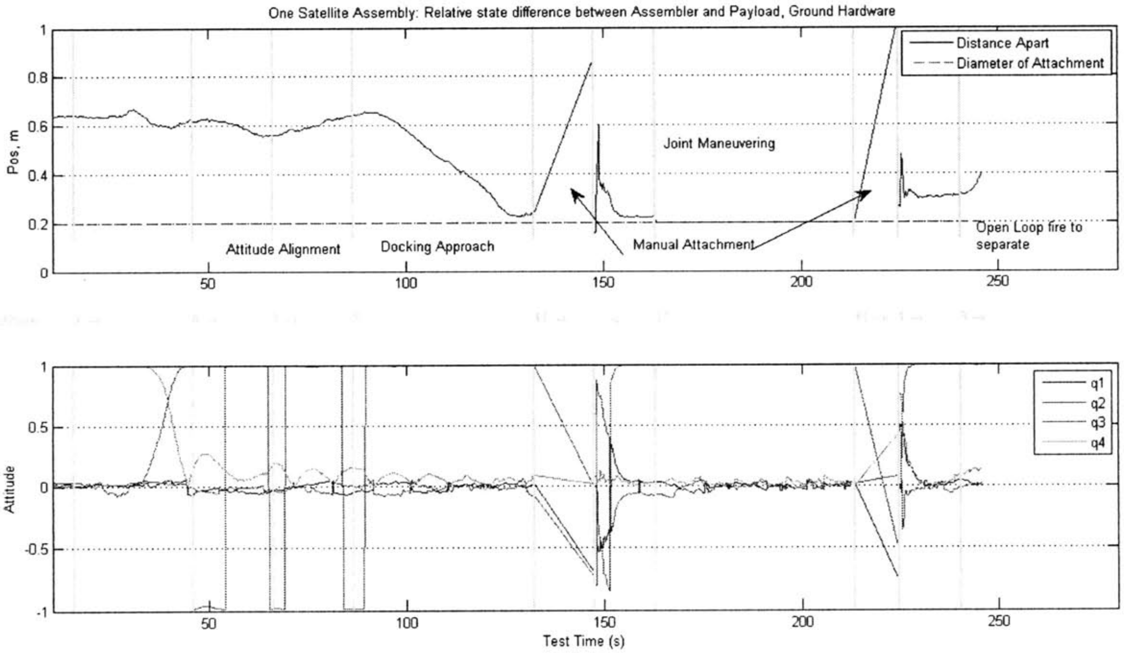
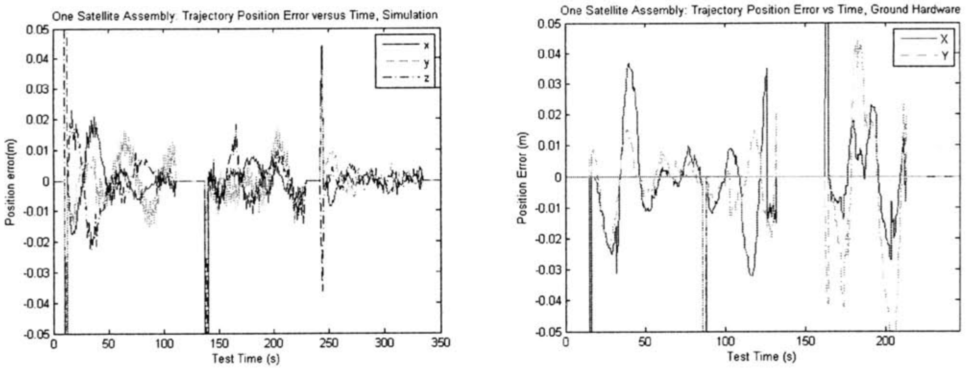


Figure 5-40: Relative Position Between assembler and module, Assembly Maneuver, Ground Hardware



(a) Simulation

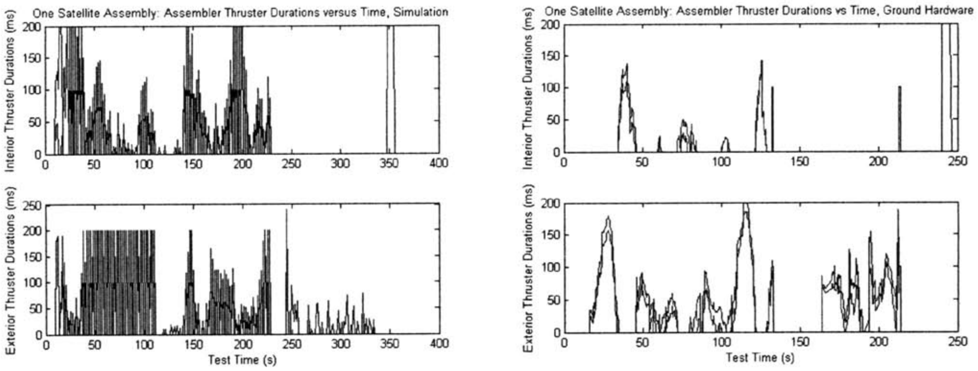
(b) Ground Hardware

Figure 5-41: Position Tracking Error of assembler, Assembly Maneuver

tests, it is sufficient for docking to occur. The tracking performance can be improved by tuning the baseline control system design, such as tuning the path to be within the actuator constraints and tuning the control bandwidth. Also, these tests are designed primarily for the ISS testing environment. Thus, they do not account for friction or table disturbances in the selection of gains or duty cycles. Only the additional mass of the air carriages is accounted for in the control system for ground testing.

The successful implementation of the control allocation algorithm is demonstrated by plotting the thruster firing durations for the thrusters on the attachment face versus the exterior thrusters. When the assembler and module are attached, the interior thrusters located on the attachment face are disabled to prevent plume impingement. Figure 5-42 shows the thruster firing durations for these thrusters. When the assembler is not attached to the module, these four thrusters are enabled and can be actuated. When the assembler and module are attached, these thrusters cannot be actuated, but the exterior thrusters are successfully actuated. Figure 5-42 demonstrates the successful parameterization of the control allocation algorithm to align to the thruster constraints.

Components of this test were performed on the ISS. However, issues with satellite reset and improper beacon setup prevented a complete successful run. This test is scheduled to be run in Test Session 22, in Spring 2010. When the test is completed successfully, this test will constitute the first on-orbit autonomous assembly maneuver



(a) Simulation

(b) Ground Hardware

Figure 5-42: Thruster Firings of assembler and module, Assembly Maneuver

demonstrated.

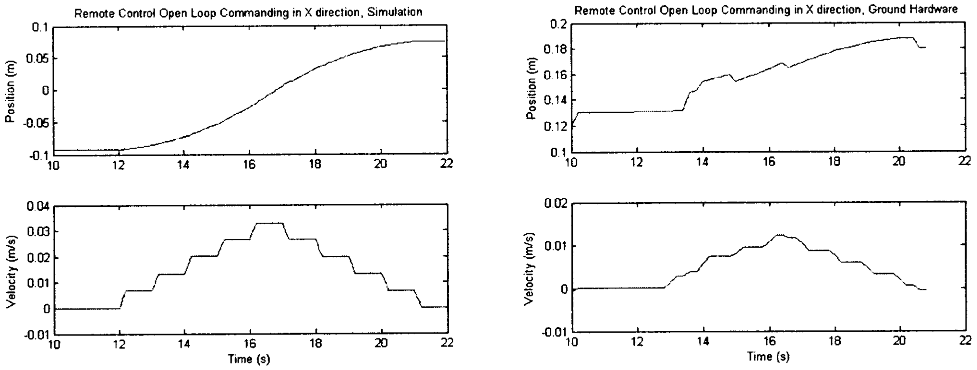
5.5.2 Remote Control

The Online Model Calculation design can also be applied to distributed systems. The application considered is a remote control scenario where a module satellite can estimate its state, but does not have the computation capability to run a path planner or controller. An assembler satellite can obtain the state of the module satellite, via communication, run the computation to determine the actuator commands and communicate those back to the module satellite.

In order to properly execute the remote control scenario, the assembler must generate a model of the module. The module properties are received, but instead of aggregating it with the assembler properties, they are used to generate the model of the module. The generated module model is then fed into a control system that monitors the module state and calculates necessary actuator commands. It is similar to the control allocation tests performed in the Object A+C configuration with joint firing, except the satellites need not be physically connected.

Two simple tests were performed. The first 10 s in simulation and the first 15 s in ground hardware tests are estimator initialization.

1. The assembler satellite communicates open-loop thruster commands to the module satellite to perform a body translation in the X direction. The thruster



(a) Simulation

(b) Ground Hardware

Figure 5-43: Module satellite position state, remotely commanded open-loop

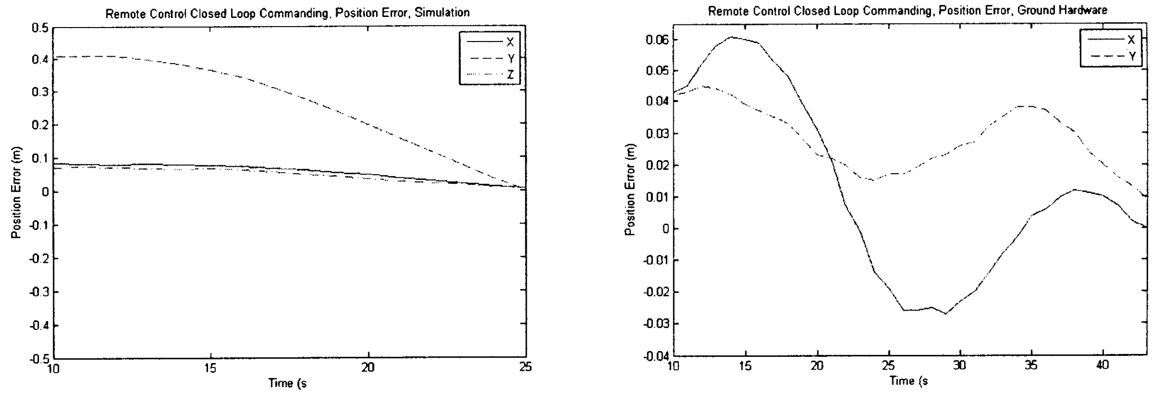
commands are not based on the module's current state, only on the module's thruster configuration. The assembler issues commands to fire the -X thrusters for 15 s, then to fire the +X thrusters for 15 s.

2. The module satellite sends its current state to the assembler satellite. The assembler satellite uses the state and the module model to generate a control input to maneuver the module to a specific location target.

Figure 5-43 shows the position of the module satellite for test one, both in simulation and hardware. The movement in the X direction denotes successful receipt and actuation of the thruster commands from the assembler satellite.

While the first test demonstrates open-loop remote control, the second test demonstrates closed-loop remote control. Figure 5-44 shows the position error of the module satellite from the desired location. The decrease in the position error to approach zero shows that the module satellite is able to reach its desired waypoint target. Issues with overshoot or settling performance can be mitigated with the use of a path planner. Also, selection of the gains can be tuned to account for the communication delay between assembler and module.

The use of the Online Model Calculation framework was useful in this application to generate the module model on the assembler and run it in a parameterized control system. By using the parameterized control system developed for Online Model Calculation, the same algorithms are able to be used for both assembler and module,



(a) Simulation

(b) Ground Hardware

Figure 5-44: Module satellite position error, remotely commanded closed-loop

even though they might have different models. The usefulness of Online Model Calculation for this application demonstrates its versatility and its ability to be extended for non-autonomous assembly applications.

5.6 Conclusions

This chapter presents the experimental implementation of parameterized control algorithms on the SPHERES testbed, both in simulation and on hardware. Two types are modules are explored, rigid and flexible modules. Experimental results show definite performance improvement for the estimator, controllers (PID and adaptive), and control allocation. The full framework was tested through two integrated tests, an assembly test and a non-assembly test. The successful performance of the assembly test in properly changing configurations is demonstrated by the successful maneuvering and estimation, which validates the Online Model Calculation framework and design. The ability to use the Online Model Calculation framework and parameterized algorithms in a distinctly different application, such as the remote control tests, shows how the design can be extended to other non-assembly applications.

Implementation on hardware unearthed issues that revealed configuration conditions that necessitate use of the new model. The necessity of incorporating the flexible dynamics depends on the level of excitation from actuation and mass ratio between Object B and Object A. The mass ratio between the assembler and module also impacts the necessity for parameterization of the estimator actuator propagation and controllers. For changing sensor configurations, the utility of incorporating the module sensors is based on the number of sensors, location of sensors, and delay in incorporation of the measurements. Overall, the successful implementation of the parameterized control algorithms on the two distinctly different modules demonstrates the validity of the parameterization of the control algorithms.

Chapter 6

Metrics for Model Generation Architecture Comparison

The design of a control system plays a large role in determining the success of a space mission. The control system is responsible for achieving position and attitude targets, under strict accuracy and resource consumption requirements. Current space missions generally have a single configuration. Thus, their control systems use a single model and are tuned to achieve the desired performance for all foreseeable operating conditions of the single configuration. Missions that employ autonomous assembly, however, are fundamentally different because the control system must incorporate configurations that change throughout the mission. Therefore, the control system must be reconfigurable.

A key aspect in making the control system reconfigurable is generating the model of the configuration and determining how it is processed by the control system. There are many types of model generation architectures. The selection of an appropriate model generation architecture can increase control system accuracy, save fuel, decrease computational processing time, and/or decrease development time. The most common practice for selecting a model generation architecture is experience, such as comparing to similar past missions. However, there is no precedent for on-orbit autonomous assembly. The closest example is on-orbit autonomous docking, such as DARPA's Orbital Express [43]. Docking is only the first step of assembly and does not cover the range of configuration changes that can occur during assembly. Therefore, in order to effectively design a control system, engineers should quantitatively compare model generation architectures and select an architecture based on performance objectives.

A key innovation of this thesis is the comparison and evaluation of model generation architectures. The comparison of model generation architectures can be used in the concept design phase to evaluate assembly scenarios. Comparisons of model generation architectures allow engineers to account for trades very early in the design process. The final selection of the architecture would therefore be more informed and optimized. To enable comparisons between model generation architectures, the first step is to determine the traits by which to compare architectures. The effective choice of metrics, to characterize both computational load and resulting control system performance, is important in making the comparison as helpful as possible. The focus of this chapter is the development of metrics to quantitatively compare model generation architectures.

This chapter starts with a review of the types of model generation architectures used in reconfigurable control systems, with an emphasis on their differences. The next section describes the metrics developed in this work that focus on three areas: control, spacecraft, and assembly mission performance. The metrics are then combined into a set of objective functions that can be used to evaluate model generation architectures. The evaluation of assembly architectures using these metrics and objective functions is discussed in Chapter 7. Finally, the metrics and the evaluation methodology are integrated into a process for the selection of a model generation architecture, described in Chapter 8.

6.1 Types of model generation architectures

This section provides a detailed description of different types of model generation architectures. The types of model generation architectures are determined through an extensive literature search. They are arranged on a spectrum, based on the nature of the information available, such as the mass properties of the system. Section 6.1.1 describes the spectrum and defines three key variables for determining where an architecture is placed on the spectrum. The different types of model generation architectures present on this spectrum are divided into four major categories. These categories are described in Section 6.1.2, including the advantages and disadvantages of each category with relation to implementation for autonomous assembly missions.

6.1.1 Spectrum Definition

The range of model generation architectures can be classified by the amount of *a priori* information known about the assembly, such as module mass and stiffness properties. By knowledge of these properties, we mean the information is made available to an assembler during the assembly execution. Though many, or all, of these properties may be known on the ground, the properties may not be stored onboard an assembler. Thus, the information content available to an assembler during the assembly execution is important in designing the control system. For this work, it is assumed that an assembler has full knowledge of its own properties, when no module is attached.

There are three critical variables by which the model generation architectures can be distinguished: *transitions*, *configurations*, and *module properties*. Knowledge of the *transitions* indicates that an assembler knows the timing of the assembly sequence. This specifically includes the time at which the transitions will occur, but not necessarily the starting and ending configuration that occurs at that transition. Knowledge of the *configurations* indicates that an assembler knows the dynamics model for each configuration used during the assembly. Though an assembler knows all possible configurations, it does not know when in the assembly a configuration will be used, which configurations will be used, or the configuration sequence. Finally, knowledge of the *module properties* indicates that an assembler knows the mass properties of the module when it is unattached to an assembler. Knowledge of how the module attaches to an assembler is not known throughout the assembly, but can be acquired at the time of attachment. To enumerate all possible situations, each of the three

Table 6.1: Types of model generation architectures based on available information in a binary truth table

Type	Transitions	Configurations	Module Properties
1	Unknown	Unknown	Unknown
2	Unknown	Unknown	Known
3	Unknown	Known	Unknown
4	Unknown	Known	Known
5	Known	Unknown	Unknown
6	Known	Unknown	Known
7	Known	Known	Unknown
8	Known	Known	Known

critical variables is given a boolean designation. A value of *true* indicates that the information is known, while a value of *false* indicates the information is unknown. The full set of possible architecture types is given by Table 6.1, using a binary truth table format.

As seen in Table 6.1, there are eight types of architectures. The spectrum definition can be identified by the progression of the types of architectures seen in Table 6.1. Architectures of Type 1 have no information about the system stored on an assembler. Architectures of this type must identify the properties online, or employ a controller with the capability of learning or adapting during maneuvering. Architectures of Type 2 have knowledge of the module properties. This allows an assembler to calculate the model at the time of attachment, even though the time and nature of the attachment is not known ahead of time. Similarly, architectures of Type 3 and Type 4 have knowledge of the configurations. An assembler can set the proper configuration, once the attachment occurs and specify which configuration is used. Architectures of Type 5 have knowledge of the transitions, but not the configurations or properties. Thus, these architectures, though they can include monitoring close to the transition time, must employ identification or learning schemes similar to Type 1. Architectures of Type 6 are similar to architectures of Type 2, but with additional monitoring during the known transition times. Finally, architectures of Type 7 and 8 have knowledge of the transitions, as well as the configurations. This is sufficient information to pre-plan the entire assembly sequence in the control system. Based on these classifications, one end of the spectrum can be defined as types of architectures that have little to no knowledge of the system. These types of architectures necessitate learning or adapting to the properties online. On the other end of the spectrum are architectures where all of the information is known. This allows for the entire assembly sequence to be pre-planned and specifically coded in the control system. The intermediate range includes designs that have partial knowledge. This spectrum is graphically represented in Figure 6-1 and includes the location for the types of architectures specified in Table 6.1.

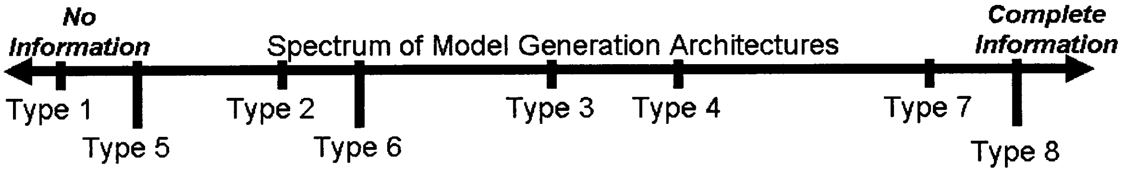
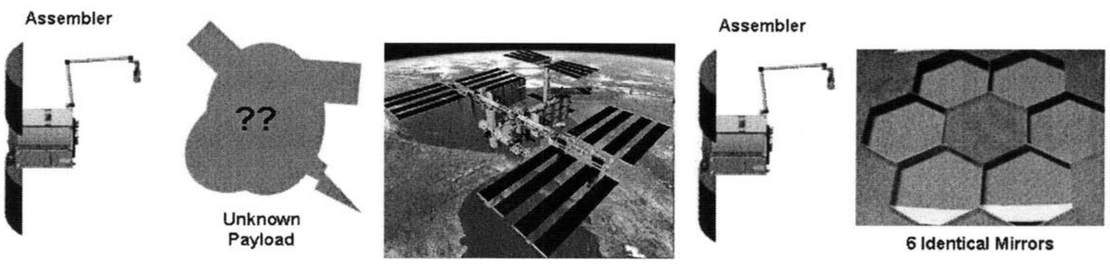


Figure 6-1: Spectrum of model generation architectures with arrangements of types of architectures from Table 6.1

Figure 6-2 gives example missions for selected types of model generation architectures. These are meant to give the reader an example of what "real-life" missions would fall under each type. For example, for Type 1, architectures in this type arise when an assembler attaches to a module in an unknown state (i.e. unknown mass and inertia due to failure or impact), as would occur in a rescue, contingency mission, or docking to a hostile target. Type 4 includes architectures where the configurations and module properties are known, but the time and order of assembly is not known. Missions that fall in this camp are those similar to the ISS, which has unknown transition times associated with launch delays. Fabrication delays could cause changes to the order of assembly, so although the properties and configurations are known, the exact time of when the transitions occur are not known. Type 8 describes architectures where all desired information is explicitly known. One example is the assembly of a modular space telescope, which has a set of identical modules that are assembled individually.

The availability (or lack thereof) of information to derive the model dynamics drives how the model generation architecture is designed. In reconfigurable systems,



Category 1 Category 4 Category 8

Figure 6-2: Example missions for selected categories of architectures from Table 6.1

reconfiguration through model generation is dependent on what information is available. Based on Figure 6-1, four major categories of model generation architectures are derived. The use of four categories allows for easy explanation of the advantages and disadvantages across the spectrum.

6.1.2 Categories of model generation architectures

There are four main categories for model generation architectures, each of which covers a section of Table 6.1 and Figure 6-1. The four model generation categories are: system identification, online model calculation, multiple model storage, and gain scheduled. Each model generation category is described in detail in the following subsections, specifically including a (1) definition of the category, (2) discussion of the advantages and disadvantages, (3) location of the category on the spectrum, (4) assumptions necessary to implement the architecture, and (5) examples of architectures from literature.

System Identification

The system identification category refers to all architectures that identify or learn about the model properties during the assembly execution. Traditional system identification derives a mathematical model of the system through analysis of input and output signals [49]. Practically, for a spacecraft system, this method amounts to adding actuation inputs and analyzing the resulting motion. The time needed to obtain the measurements is proportional to the level of fidelity needed in the model. The fidelity of the model identified increases with increasing amounts of measured data. Some architectures in this category require a model structure and seek to identify parameters to fit the observed data to the model.

The benefits of this category is the minimal amount of information necessary to implement this method. Thus, these methods can be implemented on almost all systems. The disadvantages include resource consumption required to excite the system to obtain the data (such as time and fuel), possibility of unsafe input excitation, computational load, and that the fidelity of the model is proportional to the amount of data obtained. Time and fuel are very precious resources on-orbit. Overall mission constraints on these resources may limit how much can be expended to obtain data for system identification. Also, the trajectory tracking performance during the identification or learning phase may not meet minimum performance requirements. This is an important consideration, particularly for obstacle avoidance issues.

This category is located on the far left of the spectrum, where no information is present. Since no information is required for implementation, the system identification category places no assumptions on the information necessary to implement this category.

Significant research has been performed in this field. One example is Jacques' strategy for identifying large order multi variable models from transfer function data [38]. This algorithm was implemented and demonstrated on the Middeck Active Control Experiment [2], flown on the Space Shuttle and ISS. Wilson et al. demonstrated online gyro-based system identification using a recursive least-squares method [81]. Chandler et al. used a static system identification method for a reconfigurable adaptive controller to handle changes in plant dynamics due to actuator failures [12]. These algorithms seek to maximize accuracy of the model determined, with minimal computation and resource consumption.

Online Model Calculation

The online model calculation category refers to architectures that calculate the model of an aggregated system by combining the initial model of an assembler with properties of the module attachments. For example, these model generation architectures start with the model of an assembler. At each configuration change, it uses property information of the attached module and information on how the module is attached, to generate the new model of the combined system.

The benefits of this category are minimal information storage on an assembler, capability to aggregate models, decoupled control system from the assembly sequencer, and accommodation of multiple configurations from a minimal knowledge base. This allows the same assembler to be used for multiple assembly missions with little to no modification to the control system, as long as the inputs are maintained. The disadvantages of this category are that the mass property information must be known, implementation may require some form of communication between an assembler and a module, and the computational load of the model calculation algorithm. Also, a key disadvantage of these architectures is the inherent assumption that all dynamics of the aggregated system can be captured through knowledge of how an assembler and module are attached. In complex systems, the aggregation algorithm may not be able to capture all dynamics effect, leading to a lower fidelity model.

This category is located in the middle-left of the spectrum, where basic information is available, but must be manipulated to a form that is usable in the control system. Architectures in the online model calculation category assume knowledge of the properties of an assembler, properties of the module, and knowledge of how the module is attached to an assembler. However, the information is only needed at the time of the configuration change.

Online model calculation has not been implemented much in literature, particularly for spacecraft systems. Some of the reconfigurable control literature that can fit into this category are self-assembly techniques, as the model aggregation as the vehicles attach to each other is a classic example of how online model calculation can be used to increment the model as each vehicle attaches itself. Examples of research in self-assembly include LeMaster et al.'s demonstration of automated rendezvous, docking, and self-assembly tasks between a group of three modular robotic spacecraft emulators [46]. Ukegawa and Natori developed a concept of self-assembly using autonomous modules to construct future space structure, with a stochastic relaxation process for deadlock avoidance [77].

Multiple Model Storage

The multiple model category includes architectures which store a model of each configuration. The proper control and estimation parameters, such as control gains, are determined as needed after the model is set. The control system is parameterized based on model input. For example, the controllers would take in the mass and inertia and generate the control gains needed.

The benefits of this category are full knowledge of the dynamics of the configuration, control system parameterization to take in model input, and simplified transitions between configurations. The simplified transitions between configurations is a significant benefit, particularly when the order of configurations is not known. The key disadvantage is that all of the information about the configurations must be known at the start of the assembly execution and stored on the assemblers.

This category is located in the middle-right of the spectrum, where the majority of information is present but the timing is not known. Architectures in the multiple model category assume full knowledge of the configurations. The control system does not need to know the individual module properties because it knows the configurations of its possible module attachments and of the overall structure.

The majority of the reconfigurable control system designs that consider failed or degraded states use model generation techniques from this category because failure configurations are known, but not the time at which configurations will become active. Maybeck and Stevens present a method of multiple model adaptive control which uses a bank of pre-designed Kalman filters and Command Generator Tracker (feedforward) / Proportional-Integral (feedback) controllers for each of the different anticipated failure states (i.e. configurations) [50]. Boskovic et al. switch between a bank of controllers to accommodate estimated damaged states on the Boeing's Tailless Advanced Fighter Aircraft[11]. Examples of work performed on reconfigurable control allocation, which accounts for failure of actuators, include References [17],

[14], and [20].

Gain Scheduled

The gain scheduled category includes architectures where the control system parameters are pre-determined for the entire assembly. The pre-calculated control and estimation parameters for each configuration are hardcoded into an assembler. All necessary parameters are known, so no calculation is required. Some architectures in this category also require knowledge of the transitions, which requires the assembly sequence to be pre-defined in the control system.

The benefits of gain scheduled architectures are that the control performance for each configuration can be tuned based on the mass properties, transitions between configurations are seamless, and there is no downtime associated with model calculation since the model generation is done on the ground. The disadvantages for architectures in this category are the *a priori* computation requirements, knowledge of all configurations must be available and fixed, storage of all properties on-board, and the high level of coupling between the assembly sequence and control system. Little flexibility exists for these architectures to account for changes, such as different module, failure states, and changes to the assembly sequence. Thus, the entire control system would likely need to be re-designed for the next assembly mission.

This category is located on the far right of the spectrum, where all information is known. Architectures in this category assume full knowledge of the mass and stiffness properties of the system, configurations, as well as the control and estimation gains. The transitions are generally also known for these architectures.

A key example of the use of gain scheduling in a control system design is Parlos and Sunkel's method for gain scheduled attitude control of Space Station Freedom under significant mass property variation from the berthing of the Space Shuttle [58]. Meressi and Paden use gain scheduling of H_∞ controllers to accommodate a two-link flexible manipulator [51]. Theodoulis and Duc linearize a nonlinear system about a range of small operating points, based on angle of attack and Mach number. A global gain scheduled control law is obtained by interpolating between gains of the different

Table 6.2: Categories of model generation architectures versus assumptions

No.	Assumptions			Category			
	Transitions	Config properties	Module properties	System ID	Online model calc	Multiple model storage	Gain scheduled
1	Unknown	Unknown	Unknown	√			
2	Unknown	Unknown	Known	√	√		
3	Unknown	Known	Unknown	√		√	√
4	Unknown	Known	Known	√	√	√	√
5	Known	Unknown	Unknown	√			
6	Known	Unknown	Known	√	√		
7	Known	Known	Unknown	√		√	√
8	Known	Known	Known	√	√	√	√

controllers for the linearized system to cover all operating regions [74].

Category Summary

Each category can only be implemented if the necessary information is available. Table 6.2 expands Table 6.1 to show which category can be implemented based on the knowledge of the three variables (transitions, configurations, and module properties). The √ indicates that necessary information is available to implement architectures from that category.

For example, system identification can be implemented under all architectures because it is not dependent on any knowledge of the system. Gain scheduling and multiple model storage need the information about the mass and stiffness properties of the configuration, so may only be implemented when configurations are known. Likewise, online model calculation is dependent on knowledge of the module properties. Table 6.2 does not specify which is the appropriate category to implement, just whether there is sufficient information to implement the architecture. To determine which category is appropriate to implement, one must compare them using performance metrics, described in Section 6.2.

6.2 Performance Metrics

The purpose of the metrics is to provide a quantitative way of comparing different model generation architectures. There are three main requirements for the determination of the metrics:

1. The metrics must sufficiently capture all impacts that the model generation architecture has on the overall assembly process.

2. The metrics must be quantitative and calculable.

3. The metrics must be written as a function of assembly architecture parameters (e.g. N_{trans}) so that they can be calculated for any assembly scenario. Metrics that cannot be represented in such a way must be constant for a given design, despite differences in implementation or architecture.

This section starts by providing a rationale of how the reader can generate appropriate metrics for their system, then follows with a set of example metrics and corresponding derivations.

Metrics Determination

The selection of the metrics specifies the traits of the assembly process and control system design that are important to the designer. To determine these traits, one should start at a low-level and work outwards to identify key areas. Larson and Wertz' Space Mission Analysis and Design [79] provides a good initial reference to identify the different components of a space mission, such as orbit, launch vehicle, ground operations, and the spacecraft bus.

The first step is to start within the control system to identify control performance based on the model generation architecture. Metrics to evaluate control system performance are found throughout control literature. Examples of control system performance metrics used in the literature are trajectory tracking error, stability margins, convergence properties, accuracy of model identified or estimated, robustness, and resource consumption. Computational performance of the model generation architecture should also be accounted for, such as memory storage and processing time.

After identifying the metrics to evaluate the control system separate from hardware or mission implementation, the next step is to determine the performance within the context of the spacecraft system. One must determine what spacecraft components the control system interacts with during design and operations. It is important to clearly identify the inputs to the control system (e.g. sensors, power, path target commands, disturbances), as well as outputs of the control system (e.g. actuator commands, communication packets). The next step is to trace these values outward to identify the subsystems affected (e.g. power, propulsion, communication) and the nature of the information exchange. These interactions are used to identify the traits that should be monitored at a subsystem level. Traits to track include general resource utilization of the control system, as well as tasks that are run based on control system inputs. This process can be repeated at the spacecraft system level, depending on the complexity of the system.

Finally, after identification of the spacecraft level traits, one performs a similar analysis on the control systems interaction at the assembly mission level. The metrics used in this category are commonly found in literature on concept studies or trade analysis. Examples of papers that identify mission level metrics for assembly applications are Stephens and Willenberg's metrics for in-space telescope assembly [70] and Basu et al.'s proposed autonomous assembly of a space telescope [8].

Three areas levels of interactions are considered in this work: control system, spacecraft, and assembly mission performance. The following sections describe each area, specify the metrics in those areas, and describe the rationale for the selection of the set of pertinent metrics. Also, heuristic expressions are determined for each metric based on assembly architecture parameters. The metrics provided here are not comprehensive, but rather are meant to be a guide for an engineer to start with.

6.2.1 Control System metrics

The performance of the control system is dependent on the design and implementation of a model generation architecture. A key assumption in this work is that all control system designs must satisfy a required level of trajectory tracking performance in steady state to successfully assemble the system. This allows the comparison to be more focused on the method of model generation rather than the selection and tuning of controllers. The metrics in this section are:

- Total control downtime ($T_{ctrl-down}$)
- Performance drop while transitioning (P_{drop})
- Fidelity of model needed (F)

Control downtime

Each transition in the assembly process requires time to generate and update the model. During this time, an assembler should not maneuver because the control system is not fully initialized. The control downtime metric is a summation of the time that an assembler is not maneuvering due to reconfiguration during these transitions. Lower values of control downtime are desirable because it means more time is spent maneuvering, which completes the assembly faster. The total control downtime can be expressed as

$$T_{ctrl-down} = N_{trans} * T_{ctrl-down-per-trans} \tag{6.1}$$

where N_{trans} is the number of transitions and $T_{ctrl-down-per-trans}$ is the downtime associated with each transition. The downtime associated with each transition is made up of several components, as shown in Equation 6.2.

$$T_{ctrl-down-per-trans} = T_{comm} + T_{flag} + T_{calc} + T_{id} \tag{6.2}$$

The four components of the downtime for each transition are:

- T_{comm}: the time associated with receiving the model via communication,

- T_{flag}: the time associated with switching a flag to indicate the new configuration,
- T_{calc}: the time associated with calculating the new model, and
- T_{id}: time spent exciting the system to obtain data to identify the new model.

Some of these variables, such as T_{flag}, are a function of the hardware specifications (e.g. processor speed), while others, such as T_{id} are more dependent algorithm implementation. Equation 6.2 weights these values equally as they are all in the same units (seconds), thus have equal weight on the overall assembly time. However, an alternate version would be to include a weighting term to each value, such that the user can specify which terms are more important for their particular mission.

T_{comm} is a function of the type of communication, communication bandwidth, and packet size. Equation 6.3 gives the time it takes to communicate a set of data, where the communication bandwidth is given by w_{comm} expressed in packets/sec, s_{data} is the amount of data to be transferred in kilbytes, and s_{packet} is the size of a packet expressed in kilobytes per packet.

$$T_{comm} = \frac{s_{data}}{w_{comm} s_{packet}} \tag{6.3}$$

In many cases T_{flag} is small enough to be neglected. However, in cases where an assembler is very processor restricted or if the confirmation to switch configurations must be received from outside an assembler, T_{flag} may become significant. For processor restricted assemblers, T_{flag} can be expressed as a function of the cache size on the processor and processor memory latency. The time to retrieve a variable from main memory is longer than retrieving the variable from cache memory, thus the amount of space available in the cache memory determines the overall time it takes to switch the flag.

T_{calc} is a function of both the algorithm and the hardware. The computational profile of the algorithm, such as total number of floating point operations and nature of instructions, determines the computational load. For example, the load changes depending on the number of for-loops, memory accesses, and matrix manipulations. The total execution time of the algorithm is based on the hardware processing capability of the computer, such as multi-threading, parallel processing, instructions per second, floating point operations per second, built in library functions, and clock speed. Some level of optimization is present as algorithms can be coded to make full use of the capabilities of the processor. Thus, T_{calc} must be determined through partial implementation on hardware or comparing to similar algorithms to determine a reasonable value.

For system identification, T_{id} controls the accuracy of the model generated. Therefore, if a minimum accuracy for the generated model is set, the minimum number of measurements needed can be calculated. Using the number of measurements and the time to actuate and obtain a measurement, there is minimum T_{id}. N_{idTest} equals the number of tests and T_{length} equals the length of each test in seconds. This particular scheme is based on Wilson et al. [81].

$$T_{id} = N_{idTest} * T_{length} \tag{6.4}$$

A minimum value for N_{idTest} can be obtained by setting a requirement on the error covariance of the generated model, σ_{max}. Equation 6.5 gives the minimum value for N_{idTest} based on a desired error covariance of the generated model (σ_{max}), the noise characteristics of the observed data (zero mean with variance λ), and the input spectrum ($\overline{R}$).

$$N_{idTest} = \frac{\lambda}{\sigma_{max}} \overline{R}^{-1} \tag{6.5}$$

Equation 6.6 gives the input spectrum $\overline{R}$, where y is the output measurements and u are the input values [49].

$$\begin{gathered} \overline{R} = \lim_{N\to\infty} \tfrac{1}{N} \textstyle\sum_{t=1}^{N} \varphi(t)\varphi(t)^T \\ \varphi = \begin{bmatrix} -y(t-1) & \dots & -y(t-n) & u(t-1) & \dots & u(t-n) \end{bmatrix} \end{gathered} \tag{6.6}$$

The control down time is a function of the number of transitions, N_{trans}, which is a property of the assembly architecture. For example, if an assembler is maneuvering one module at a time, the number of transitions is equal to twice the number of module (N_{pl}), to account for an attachment and detachment of each module ($N_{trans} = 2*N_{pl}$). However, the downtime associated with attachment is different than the downtime associated with detachment. For cases when an assembler stores its own model when no module is attached, the downtime associated with the detachment transition only includes the time associated with switching the flag to indicate the return to the original assembler model. There also might be situations when some transitions have models stored, while other transitions have configurations that require identification. Equation 6.1 can be generalized to remove the assumption that each transition has equal downtime, by summing the control downtime time for each specific transition, as given in Equation 6.7. The subscript i denotes the value of that parameter for transition i specifically.

$$T_{ctrl-down} = \sum_{i=1}^{N_{trans}} (T_{comm,i} + T_{flag,i} + T_{calc,i} + T_{id,i}) \tag{6.7}$$

The sensitivity of the control downtime can be seen by differentiating Equation 6.2 with respect to the hardware parameters and assembly architecture parameters. Equation 6.8 gives the equations for the sensitivity with respect to number of transitions (a scenario parameter) and time associated with switching a flag to indicate the new configuration (a hardware parameter). The sensitivity to the other hardware parameters that the control downtime is dependent on, such as T_{comm} and T_{calc}, are identical to that for T_{flag}.

$$\begin{aligned} \frac{dT_{ctrl-down}}{dN_{trans}} &= T_{comm} + T_{calc} + T_{id} + T_{flag} \\ \frac{dT_{ctrl-down}}{dT_{flag}} &= N_{trans} \end{aligned} \tag{6.8}$$

$T_{ctrl-down}$'s sensitivity to scenario is based on hardware parameters. This indicates that a slower processor capability of the hardware leads to more variation of the control downtime. In other words, the slower the processor, the more the model generation architectures are distinguished. As the computer becomes faster, the sensitivity of the control downtime to scenario decreases. Also, the sensitivity of the control downtime to hardware parameters is equal to the number of transitions, which is a measure of the size of the assembly. The larger the assembly, the larger the variation of the control downtime.

Performance drop while transitioning

The "performance drop while transitioning" metric is a measure of how the controlled state differs from the desired state during the transition period. Lower values of performance drop are desirable as this means the controlled state closely matches the desired state. The transition period is defined as the time from the instant the flag is set to the new configuration until the model of the configuration is obtained and the control based upon that new model is initiated. If no model is specifically calculated, the transition period lasts until steady state performance is reached. To ascertain the performance during transition, one must first identify when the system has converged to the proper model. Consider a linear time-invariant second-order system based on $F = ma$ (Eqn. 6.9).

$$J\ddot{x} = u \qquad \dot{x} = Ax + Bu \tag{6.9}$$

The variable u is the control input. Using a proportional-derivative control law, u is given by Equation 6.10, where K_p and K_d are the (strictly positive) control gains, x is the actual state, and d is the desired state.

$$u = K_p(d - x) + K_d(\dot{d} - \dot{x}) + J\ddot{d} \tag{6.10}$$

Equation 6.11 gives an expression for the closed loop dynamics is obtained by substituting equation 6.10 into Equation 6.9.

$$\begin{gathered} J\ddot{x} = K_p(d - x) + K_d(\dot{d} - \dot{x}) + J\ddot{d} \\ J(\ddot{d} - \ddot{x}) + K_p(d - x) + K_d(\dot{d} - \dot{x}) = 0 \end{gathered} \tag{6.11}$$

Equation 6.12 can be rearranged into state space representation, where the state vector is function of the error ($y = [\ e \quad \dot{e}\ ]$), where $e = d - x$.

$$\dot{y} = \begin{bmatrix} 0 & 1 \\ -J^{-1}K_p & -J^{-1}K_d \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} \tag{6.12}$$

Equation 6.13 gives a candidate Lyapunov function (V_L) for the system in Equation 6.12, where A is the state transition matrix. It can be proven that this system is exponentially stable, when P is positive definite.

$$V_L = y^T P y \tag{6.13}$$

Equation 6.14 gives the derivative of the Lyapunov function in Equation 6.13.

$$\dot{V}_L = -y^T Q y \tag{6.14}$$

To determine Q and P such that the system is stable, Q is set as the identity matrix and used to solve for P, using Equation 6.15.

$$A^T P + PA = -Q \tag{6.15}$$

By solving for P, the convergence rate is bounded by $\gamma = \frac{\lambda_{min}(Q)}{\lambda_{max}(P)}$, where λ represents the eigenvalues of Q and P. Since Q is chosen to be I, $\lambda_{min}(Q)$ is 1. P is a function of the control gains (K_p and K_d) and the mass (m), as are its eigenvalues. By assuming that the Lyapunov function is converged when $V_{L,f} = 10^{-6}$, the convergence time can be obtained by Equation 6.16

$$T_{convIdeal} = -\frac{2}{\gamma} ln(\frac{V_{L,f}}{V_{L,0}}) \tag{6.16}$$

Equation 6.16 assumes that the control gains are selected to achieve the desired performance for the given mass. If the properties are slightly off from the baseline, the convergence time will increase. In practice, this value should be calculated by simulation through implementation of the control system. The convergence time determined through implementation in simulation is denoted as $T_{convImpl}$, while the theoretical convergence time is $T_{convIdeal}$. The performance during convergence can be expressed as $RMS = \sqrt{e^T e}$, equivalent to the root mean square of the error, evaluated over the convergence time. Thus, P_{drop} is the difference of the performance of the implemented design (calculated from $t = 0$ to $t = T_{convImpl}$) and the theoretical design (calculated from $t = 0$ to $t = T_{convIdeal}$). Equation 6.17 details this relationship, where the integral is the total RMS error over the specified time period. The units of the performance drop are based on the quantity being evaluated. The performance drop for position could be in meters, while the performance drop for attitude could be in radians.

$$P_{drop} = \int_0^{T_{convImpl}} (\sqrt{e_{Impl}^T e_{Impl}} dt) - \int_0^{T_{convIdeal}} (\sqrt{e_{Ideal}^T e_{Ideal}} dt) \qquad (6.17)$$

Fidelity of model needed

The fidelity of the model, F, needed in the control system puts restrictions on the ground development of the model generation architectures. This unit less metric is used as a measure of ground development work needed to identify the model on the ground. Lower values of the fidelity of the model are desirable because less resources are required on the ground to explicitly calculate the models. There are four values: High, Medium, Low, and None. Each of these qualitative values is given a quantitative value to allow it to be used in objective function calculation. The four fidelity values are specified in terms of the percentage error between the generated model and the true model. The percentage values are selected based on simulation analysis. For example, Figure 6-3 shows the degradation in tracking performance caused by an percent error in the knowledge of the mass of an assembler. Figure 6-3 shows an example of how the percent error of the model correlates to a tracking error. Though knowledge of the system increases over the project life cycle, this metric is meant to capture the ground resources necessary in determining the model that will be stored on board the satellite.

- High:(Value = 4) High fidelity models require knowledge of the system to within

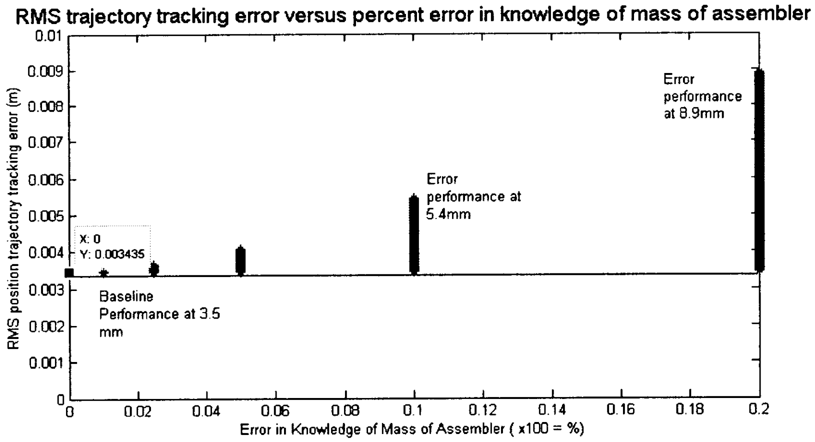


Figure 6-3: Performance change due to error in the knowledge of the mass

5% of the true hardware system. Model determination usually entails performing rigourous system identification of the as-built system.

- Medium: (Value = 3) Medium fidelity models can be generated by a finite element model of the system. These models require knowledge to within 10% of the true hardware system.

- Low: (Value = 2) Low fidelity models are satisfied with simple models, such as CAD models, and are highly tolerant to variations in the model. These models require knowledge to within 30% of the true hardware system.

- None: (Value = 1) This value classifies control systems that do not require a model to implement. Examples are types of system identification methods that generate the model online.

6.2.2 Spacecraft metrics

The design of the model generation architecture also has implications on the rest of the spacecraft design. Consider the common subsystems of a spacecraft: Guidance Navigation and Control (GNC), Command and Data Handling (CDH), Communications (Comm), Structures, Power, and Thermal. GNC is essentially the control system, so the implications for the subsystem are already accounted for in the previous section. This thesis focuses on propellant-based assemblers, so the Power and

Thermal subsystems are not greatly affected by the model change in the control system. If other forms of propulsion are considered, such as electric propulsion, the implications on the Power and Thermal subsystems could be significant. Metrics similar to those discussed for additional fuel used should then be included. The main metrics considered for the spacecraft are:

- Total additional mass (m_{tot})
- Sensing equipment needed (S)
- Communication bandwidth needed (C)
- Total memory used (M)

Total additional mass

The main impact to the Structures subsystem is any additional mass that would be incurred by implementing the model generation architecture. The gain scheduled architecture is used as the baseline because it is the architecture that has the highest technology readiness level and has been implemented in flight. Additional mass is the difference between the mass of the current architecture and the baseline architecture.

There are two components: dry mass and fuel mass. Dry mass consists of any additional hardware required to implement the model generation architecture. This includes the addition of sensors (ex. RFID reader), antennas, or structural mass. One example is the selection of online model calculation that uses communication to transmit properties. The selection of this architecture may necessitate the addition of communications hardware. For this metric, lower values are more desirable, since mass drives cost.

The additional fuel mass consists of fuel expended when trying to generate the model. It is not meant to capture differences between controller performance, thus does not include all fuel used during assembly. Therefore, additional fuel used is only present in architectures using system identification and/or adaptive control. For upfront identification, the additional fuel used is expended in the maneuver to excite the system and take measurements. There are N_{idTest} sets of tests that each last T_{length} seconds long. During a single test of T_{length} seconds, the actuators will not be firing constantly. A firing patten will be employed to excite all of the axes, such as shown in Figure 6-4. From a selected firing pattern, a percent firing is calculated, P_{on}, which gives the percentage of T_{length} where the thrusters are firing. Therefore, the total fuel used to identify the model can be calculated from Equation 6.18, where $\dot{m}_{fuel}$ is the mass flow rate out of the thrusters and P_{on} is the percentage of time when the thrusters are firing. Determination of the additional fuel used for architectures that employ learning or adaptation is generally accomplished through implementation in simulation.

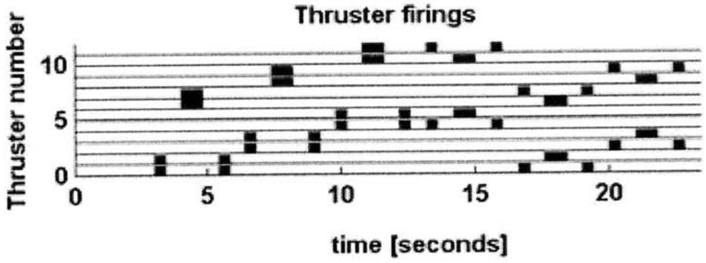


Figure 6-4: System Identification firing maneuver sequence, SPHERES TS03 P103 by NASA Ames, [73]

$$m_{fuel} = N_{idTest} * T_{length} P_{on} \dot{m}_{fuel} \tag{6.18}$$

For Figure 6-4, thrusters were on for 9.6 s in pairs of two, meaning 19.2s of firing time. The total test time was about 25s, where all 12 thrusters could have been on. Thus, the P_{on} is 19.2s divided by 12*20s, which is 8%.

The total additional mass is the sum of the dry mass and the fuel mass.

Sensing and Communication

The Communication subsystem is affected if the method of obtaining the module properties is through communication. If communication is required, it entails additional mass, depending on the type of communication method. If a passive method is chosen on the module, such as RFID, then the additional mass required on an assembler is the mass of the RFID reader. The mass of the RFID tag is negligible in comparison to the mass of an assembler. Therefore, the values for sensing equipment needed is Yes or No. If Yes, then the additional mass is included as dry mass in the total mass metric, and is chosen to be a standard RFID reader mass.

If the module is an active module, with communication ability, then mass properties can be transferred through communication. Thus, the values for Communication bandwidth need is also a binary Yes or No. If the answer is Yes, then the bandwidth, which is a property of an assembler, is used to calculate T_{comm}.

Values of 'No' are more desirable because it leads to less mass for sensing and less bandwidth used for communication.

Total memory required

The main impact on the CDH system is the requirements on processor specifications and data storage. These relate to how much information the different architectures need to store at a time. Different model generation architectures use different amounts of processor memory, particularly those that employ a large calculation or identification algorithms. Lower values of memory required are desirable as the memory saved can be used for other algorithms or data collected. In current design climate, flight qualified processors are somewhat restrictive on processor memory, particularly for nanosatellites. The importance of memory as a driving metric may change in the future based on the trends in flight qualified computation and autonomy tasks. However, it is still a useful quantity to track as it is required to size the processor. When designing a new mission, it is fairly easy to obtain the required amount of memory, even for large autonomous tasks. However, for certain cases, when an engineer may be designing for a system that has already been built, memory might once again be a concern.

Memory properties are processor specific and must be calculated through partial implementation on hardware. For each method, there are two types of memory: overhead and dynamic memory. The definitions of these terms are specific to this thesis. Overhead memory is defined as the minimum memory necessary to implement the model generation architecture. Dynamic memory is defined as the amount of memory to implement each configuration. The total amount of memory needed is the sum of the overhead and dynamic memory, where the total dynamic memory is the product of the memory per configuration and the number of configurations.

Gain scheduling uses only dynamic memory. The memory to store a single configuration (M_{cfg}), includes the necessary gains. Therefore, the total memory is the product of the memory for a single configuration and the number of configurations (Eqn. 6.19). Assuming a single assembler architecture, and that an assembler moves only one module at a time, the number of distinct configurations is the number of different types of modules (N_{types}) plus one to include the configuration of an assembler. Nominally, the memory associated with a configuration is fairly constant for a hardware system because there is a basic set of information that is necessary for all configurations. However, some variations in M_{cfg} may occur due to differences in number of sensors, actuators, etc. Equation 6.19 can be generalized to account for varying M_{cfg}, as is given in Equation 6.20.

$$M_{tot} = M_{cfg}(N_{types} + 1) \tag{6.19}$$

$$M_{tot} = M_{assem} + \sum_{i=1}^{N_{types}} M_{cfg,i} \tag{6.20}$$

Multiple model storage includes both overhead and dynamic memory. The dynamic memory for multiple model storage is the memory to store a single model (M_{mdl}), which includes mass, inertia, center of mass, etc. It also includes a small amount of overhead memory (M_{calc}) in order to calculate the gains from the model. The total memory is the sum of the overhead memory and the total dynamic memory, which is the memory per model times the total number of models (Eqn. 6.21).

$$M_{tot} = M_{calc} + M_{mdl}(N_{types} + 1) \tag{6.21}$$

Similarly, online model calculation has both overhead and dynamic memory. The memory that calculates the integrated model given two models is specified as M_{calc}. The overhead memory contribution is from the storage of an assembler's model

(M_{assem}). The amount of dynamic memory stored is based on the type of online model calculation. If the properties are stored on board, the memory associated with each module's properties is M_{seg}. If the properties are communicated, the dynamic memory associated with the module's properties are not stored in an assembler's memory throughout the entire assembly. The module properties are only stored temporarily while the model of configuration is being calculated. Equation 6.22.a shows the total memory for online model calculation with properties stored, where the number of module configurations is equal to the number of distinct types of module (N_{types}), since the memory for an assembler is specified separately. Equation 6.22.b shows that the total memory for online model calculation with properties communicated is just the overhead memory.

$$\begin{aligned} M_{tot} &= M_{calc} + M_{assem} + M_{seg}N_{types} \quad (a) \\ M_{tot} &= M_{calc} + M_{assem} \quad (b) \end{aligned} \tag{6.22}$$

For system identification, the memory storage is all overhead memory. The overhead memory consists of memory associated with the system identification algorithm (M_{sysid}) and memory to store the current model (M_{assem}).

$$M_{tot} = M_{sysid} + M_{assem} \tag{6.23}$$

6.2.3 Assembly Mission Performance metrics

Assembly mission performance relates to the incorporation of the control system design into assembly planning. It is important to note that these metrics are meant to differentiate model generation architectures, not to evaluate the performance of the assembly or mission in general. Thus, the metrics "assembly time" or "overall fuel used" employed here are used to compare the difference between model generation architectures at the highest mission level. The metrics for this section should identify which architecture is better at being implemented and integrated into the overall assembly planning, and the different requirements each design has on pre-flight ground development work. These metrics are:

- Integration with assembly sequencing (I)
- Technology readiness level (TRL)
- Total assembly time (T_{assem})
- Total fuel used (m_{fuel})

Integration with assembly sequencing

This metric captures the codependency between the assembly sequence and the control system design. Lower values for integration are desirable as higher levels of integration require more ground development. If there is a high level of integration, such as the control system parameters specified entirely by the assembly sequencer, more time is required during development to coordinate, design, and test that interface. If there is little integration between the sequencer and control system, a previously designed control system can be used with minimal integration effort. The decoupling between the sequencer and control system allows the control system to be reused. This metric has four values:

- High: (Value = 4) Control system is fully integrated with the sequence. Any changes to the mass properties of the module or order sequence of assembly requires redesign of control system.
- Medium: (Value = 3) Control system has knowledge of all configurations, but not necessarily their order. Can handle changes to the sequence easily, but not changes to the module properties.
- Low: (Value = 2) Control system has knowledge of each configuration at the given instance. Easily adaptable to changing sequences and module properties.
- None: (Value = 1) Control system has no information about the modules or sequence. Can be used for various missions with no modification to the control

system design. Primarily only applicable to designs such as system identification.

Technology readiness level

Technology readiness level (TRL) is a metric commonly used in the aerospace industry to measure the status of a technology during its development. Higher values of TRL are desirable because architectures with higher TRL have lower risk and more available experience for implementation. The scale used in this thesis is NASA's TRL scale, which from 1 to 9. A TRL of 1 indicates that it is a concept only and has not yet been proven feasible. A TRL of 5 entails implementation and technology demonstration in a controlled laboratory environment. Finally, a TRL of 9 indicates demonstration of the technology in a flight environment. The higher the TRL of an algorithm, the less risky the implementation will be. Also, when used in concept planning, a higher TRL generally requires less cost to be included to the budget to mature the technology to flight. This scale is non-linear, so each step becomes more challenging than the previous, particularly after TRL 5.

Total assembly time and Total fuel used

These two metrics capture the effect of the model generation architecture on high level assembly performance. Total assembly time captures the duration of assembly from just after launch to the completion of the assembled structure. The total fuel used provides an efficiency measure of the assembly and is calculated over the course of the assembly. The total fuel used metric can also be used for propellant-free methods, by capturing the main resource used, such as energy. Lower values of assembly time and fuel used are desirable. The comparison of assembly architectures with these metrics allows for the determination of the effect of the selection of the model generation architecture on high level assembly performance metrics.

Table 6.3: Objective functions with the corresponding values

Objective	Value
Minimize Mass	m_{tot}
Minimize Processing time	T_{proc}
Minimize Time	$T_{ctrl-down}$
Minimize ground development time	$\frac{1}{TRL} + I + F$
Maximize adaptability	$\frac{1}{I}$

6.2.4 Objective functions

The metrics listed above can be used individually, or combined to form multi-objective functions. These objectives functions are given in Table 6.3. In this thesis, five main objective functions are explored, four single cost objective functions and one multi-cost objective function. The first three single objectives are minimization of mass, processing time, and control downtime respectively. It is important to note that these quantities are not the assembly launch mass or assembly time, but are the additional amounts imposed by the implementation of the reconfiguration. The fourth objective function is the maximization of adaptability. Maximization of adaptability requires minimization of the integration between the assembly sequence and the control system, so this is inversely proportional to the integration, I.

The multi-cost objective function is the minimization of ground development time. Minimization of the ground development is comprised of three metrics. It is inversely proportional to the TRL level, since a higher TRL means less ground development needed, and it is proportional to integration of the assembly sequence and fidelity of the model. The higher the integration and fidelity of the model, more time is required during development to complete it. All of these objectives are normalized to the baseline design so that they are compared on their relative differences, as opposed to the actual numerical values.

6.3 Conclusions

Existing literature indicates that current missions compare and select model generation architectures based on experience, without a rigorous definition of the metrics or process used for selection. This chapter provides the first step in formalizing a process for selection by (1) categorizing the range of model generation architectures, and (2) developing a set of quantitative metrics to compare model generation architectures. These metrics encompass control system, spacecraft, and assembly mission performance. The metrics are used to create objective functions, such as minimization of processing time. These metrics serve as a foundation to the process of selection in Chapter 8.

The metrics provided in this chapter are meant as a guide to readers of how to generate relevant metrics. The comparison and subsequent analysis of model generation architectures can be very sensitive to the metrics and objective function definition. In particular, the metrics that attempt to capture the qualitative aspects of mission design, such as integration and ground development work must be analysed to make sure that the weighting is appropriate for the particular assembly mission. Similar issues exist with the definition of the objective functions. The weighting of the different terms in the objective function can greatly influence the resulting design. As autonomous assembly research progresses, key metrics will emerge based on driving issues. Until then, rigorous analysis must be made comparing multiple aspects of the model generation architectures with respect to the assembly mission.

Chapter 7

Metrics Evaluation for Assembly Scenarios

This chapter evaluates the metrics defined in Chapter 6 for two assembly architectures, a space telescope assembly and a self-assembly of autonomous vehicles. The evaluation of the metrics serves to map the model generation architectures to the overall assembly scenario. The impact of the model generation architecture selection on the overall mission performance provides important information that can be used in trade studies to improve the design of the mission. The mapping between model generation architectures and control system/mission performance is accomplished through the development and execution of an assembly simulation tool.

7.1 Scenario Description

Two scenarios, ALMOST and ACRRES (described below), are chosen for the evaluation of the metrics, and are later also used in Chapter 8 to exercise the process of selection. The first scenario is chosen to maximize the configuration changes, while the second scenario is chosen with few configuration changes. The comparison between these scenarios demonstrates the effect of the configuration changes on the overall performance. Thus, it cements the necessity for proper selection of model generation architectures for on-orbit autonomous assembly.

7.1.1 ALMOST

Assembly of Large Modular Optical Space Telescope (ALMOST) [52] is a concept architecture for the on-orbit autonomous assembly of six hexagonal segments into a primary mirror (Figure 7-1) to be assembled using the SPHERES satellites inside the ISS. Each hexagonal segment is 7 inches in diameter and weighs approximately 1.5 kg. Figure 7-2 shows the prototype mirror as part of the SPOT-lite testbed at NASA Goddard Space Flight Center. In this scenario, there is only one assembler, modeled on the SPHERES satellite. Each of the mirror segments is a passive payload that has no actuation or sensing capability. The mirrors start in a stack configuration and are assembled into a ring, as in Figure 7-2.

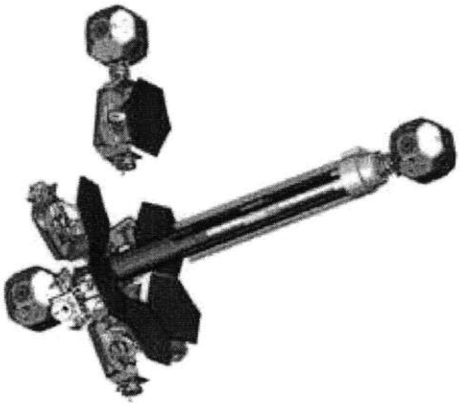


Figure 7-1: NASA GSFC's SPOT telescope architecture modified for ALMOST robotic assembly

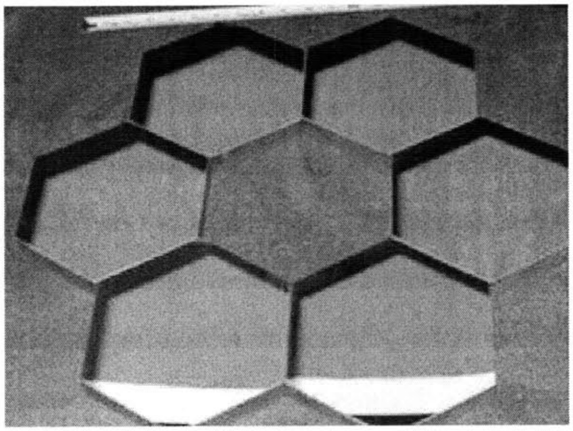


Figure 7-2: NASA GSFC's SPOT-lite mirrors

7.1.2 ACRRES

Autonomous Control Reconfiguration for Robotic Exploration Systems (ACRRES) [35] consists of three assemblers that autonomously self-assemble to form a single spacecraft. Each satellite has full actuation and sensing capability, and is modeled after a SPHERES satellite. Figure 7-3 shows the initial and final configurations for the ACRRES scenario. Only configuration changes during maneuvering are considered in this thesis, not during the aggregation of the assembled structure. Thus, there are little to no configuration changes during maneuvering for ACRRES. The necessary aggregation occurs during build up of the assembled structure.

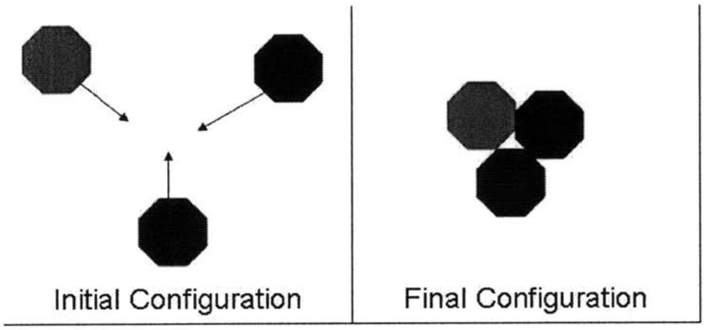


Figure 7-3: ACRRES Initial and Final configurations

7.2 Simulation Description

The simulation executes a full time elapsed modeling of the assembly, with a full embedded control system. The simulation includes actuator and sensor noise to provide a measure of realism. The inputs into the simulation are the scenario to run (1 = ALMOST, 2 = ACRRES) and the model generation architecture to use. Categories of model generation architectures are discussed in detail in Chapter 6. Figure 7-4 shows the overall block diagram of the simulation, with the control and estimation loops highlighted. Each block in Figure 7-4 represents a function call in the simulation, and the black arrows and dots represent the flow of variables between function calls. The following sections describe the initialization, control loop, estimation loop, and metrics sections of the simulation.

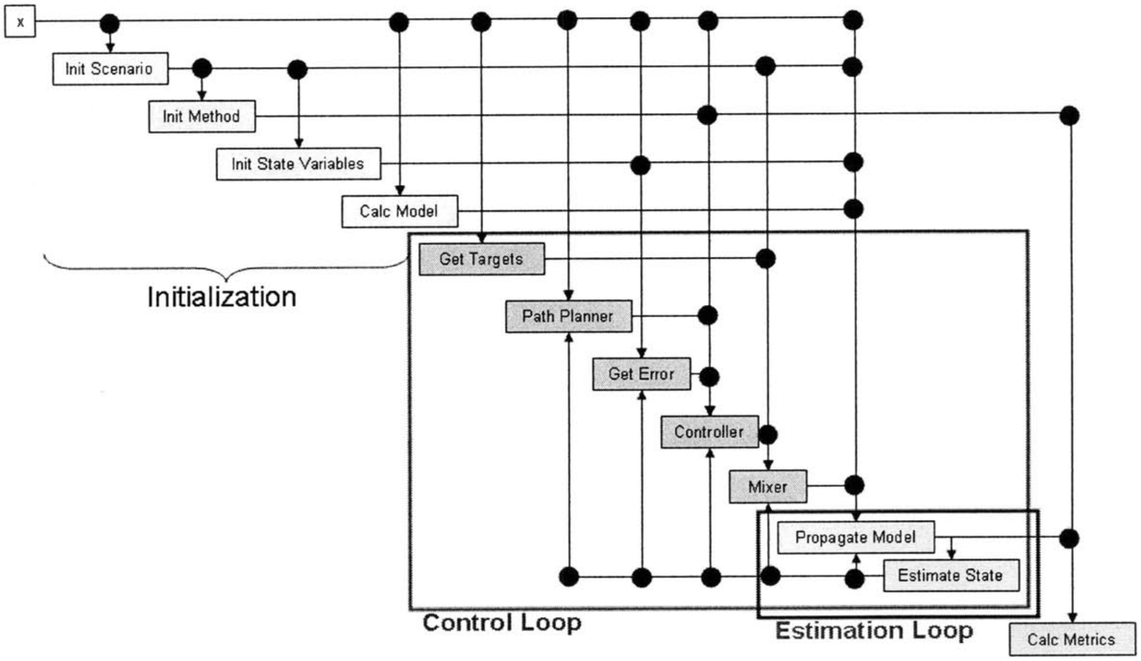


Figure 7-4: Simulation Block Diagram

Table 7.1: Simulation Parameter Description

Type	Name	Description
Design Variable	Number of Modules	Specified by the scenario
Design Variable	Number of Assemblers	Specified by the scenario
Design Variable	Assembler mass properties	Includes mass, inertia, dimensions etc. Based on the SPHERES testbed
Design Variable	Module mass properties	Includes mass, inertia, dimensions etc. Based on the SPHERES testbed
Design Variable	Type of assembler	Tug vs Self assembly
Design Variable	Location of assembly	Currently set to ISS. Can be upgraded in future to set environment disturbances.
Design Variable	Propulsion type used	Propellant. Can be upgraded in future to include electric, etc.
Parameter	Module target state	Specified by the scenario
Parameter	Assembler target state	Derived based on number of assemblers and Module targets
Parameter	Assembler thruster properties	Based on the SPHERES testbed
Parameter	Assembler sensor properties	Based on the SPHERES testbed
Parameter	Specific impulse	Based on the SPHERES testbed
Objective	Module State	Calculated through execution of assembly
Objective	Assembler State	Calculated through execution of assembly
Objective	Assembly time	Time from start of assembly to when all modules in target locations
Objective	Fuel consumption	Cumulative fuel used throughout assembly for all assemblers
Objective	Trajectory tracking error	Difference between actual state and planned path throughout assembler
Objective	Memory storage	Memory required to store on assembler based on model generation architecture

7.2.1 Initialization

The initialization section of the simulation sets up the simulation based on the inputs of the scenario and the model generation architecture. Based on the scenario, the following parameters are initialized: number of modules, number of assemblers, initial states of assembler and modules, final target states for modules, and mass property information for assembler and module (mass, inertia, thruster configuration, and sensor configuration). The model generation architecture sets the controller to be used and is used to calculate the metrics. The simulation uses the mass property information to calculate the dynamics model. Table 7.1 gives a detailed overview of the variables used in the simulation.

InitScenario initializes the simulation to the specified assembly scenario under analysis. The simulation currently accommodates two different assembly scenarios, as described in Section 7.1. For each scenario, there are nine cases which span a range of assembly architectures, varying the number of assemblers, the mass of the assemblers, and the number of modules. The inputs to the InitScenario function are the scenario number and the case number. The main outputs of the InitScenario function are the type of assembly (tug, robotic arm, self, etc), propulsion type used, and the location of the assembly. These variables are incorporated throughout the simulation and allow for a platform for upgrading the simulation for a wide variety of assembly scenarios. The InitScenario also outputs the assembler and module properties, based on the selected scenario. Current properties that are included are mass, inertia, thruster locations, specific impulse, thruster force, and receiver locations. Finally, the initial and final positioning is specified. This includes a list of target locations and attitudes, as well as the attachment status for each target maneuvering. The InitScenario function is modular to allow for the incorporation of additional properties based on the scenario.

After initialization of the scenario properties, the model generation architecture is initialized. The architectures used in this chapter are listed in Table 7.2. Specific detail on each model generation architecture specified in Table 7.2 is available in Chapter 8 (Table 8.1). The InitMethod function initializes the metrics associated with the model generation architecture selected, specifically those that can be calculated prior to the full assembly execution. Thirteen model generation architectures have been implemented, as specified in Table 7.2.

For each model generation architecture, the following metrics are set in this function: total memory required; memory per model; memory per module; processing time required; sensing equipment needed; communication bandwidth needed; additional dry mass; ground system required; ease of changing between models; ease of model incorporation; integration with assembly sequencing; total control downtime; control downtime per maneuver; method of model calculation; and fidelity of measured model needed. The metrics of fuel used to calculate model and performance

Table 7.2: Model generation architectures implemented in the simulation

Architecture	Assumptions		
	Transitions	Config Properties	Module Properties
Gain scheduling (w/transitions)	Known	Known	Known
Gain scheduling (w/o transitions)	Unknown	Known	Known
Gain scheduling (w/transitions) AC control	Known	Known	Known
Gain scheduling (w/o transitions) AC control	Unknown	Known	Known
Multiple model storage	Unknown	Known	Known
Multiple model storage (estimator config only)	Unknown	Known	Known
Multiple model storage AC control	Unknown	Known	Known
Online model calc (prop stored, PID ctrl)	Unknown	Unknown	Known
Online model calc (prop comm, PID ctrl)	Unknown	Unknown	Known
Online model calc (prop stored, AC control)	Unknown	Unknown	Known
Online model calc (prop comm, AC control)	Unknown	Unknown	Known
Model Identification Adaptive control (MIAC)	Unknown	Unknown	Unknown
System identification	Unknown	Unknown	Unknown

drop while transitioning are extracted from the overall control performance from the assembly execution. The metrics that are a function of hardware properties are evaluated based on the SPHERES hardware properties as specified in Chapter 3.

The CalcModel function determines the models necessary for the assembler, by taking in the mass property information of the assembler and the modules to create a state space model assuming double integrator dynamics. This simulation assumes that an assembler moves at most one module at a time. Thus, only two models are necessary for each assembler. The first model is for the unattached configuration, while the second model is for the attached configuration. The mass property and thruster property information are used in the determination of the B matrix, which is continually updated to account for the fuel used. The function calculates the model first in the initialization phase. However, the CalcModel function is also the basis of the PropagateModel function, so is run at each time step.

7.2.2 Control Loop

The control loop section is the largest and most important section of the simulation since the objective is to quantify the impact of model generation architectures used in the control system. It is important to note that the controllers implemented have all been tuned to meet a minimum tracking performance requirement. This allows the control systems to be compared solely on the model generation technique used since tracking performance is highly dependent on the proper selection of gains. The control loop obtains a module target, generates a trajectory to reach that target and maneuvers to the target. When the assembler is within 2 cm of its final target position, the next module target is obtained. The cycle is repeated for each module until the assembly is complete.

Get Targets

For the ALMOST scenario, the set of final module positions output from InitScenario are expanded to consist of four major maneuvers for an assembler based assembly. For each module, the maneuvers consists of an approach the stacked module, retrieval of the module from the stack, and maneuvering upwards with the module, then a final maneuver to push the module into its final position. The GetTargets function takes in the list of total targets and the current progress of the assembly. The function outputs the current target state for the module being assembled.

For the ACRRES scenarios, there is only one maneuver per assembler/module, which is specified by the final target location. Thus, the GetTargets function simply passes through the final module positions received from InitScenario.

Path Planner

The control system is given a path to follow to maneuver from the current state to the specified target state obtained from GetTarget. The trajectory is based on the current configuration of the assembler and the specification of the propulsion type. In this work, a Bang-Bang trajectory profile was used. The maximum acceleration is calculated so that the thrusters are not saturated, where $a_{max} = \frac{1}{10}\frac{f_{thr}}{mass}$. The switching time is calculated as

$$t = \sqrt{\frac{\|d - x\|}{a_{max}}} \tag{7.1}$$

After the path is planned, it is sent to the controller as a time sequence of target states.

Get Error

The assembler attempts to follow the trajectory by calculating the error, which is the difference between desired state given by the trajectory and estimated state. The error is fed into the controller.

Controller

The model generation architecture specified by InitMethod determines the type of controller used. The majority of architectures use a proportional-integral-derivative (PID) controller, which requires all mass property information to be available prior to control system execution. For architectures that employ learning or identification, an adaptive controller (AC) is used. The gains for both PID and AC controllers are calculated based on the configuration of the assembler. The gain calculation laws used in the script are discussed in detail in Chapter 4. The controller outputs a six element control vector of forces and torques.

Control Allocation

The forces and torques control vector gets fed into the control allocation algorithm, which converts the control vector from forces and torques into thruster on/off commands. The control allocation algorithm uses a pulse-width modulation scheme; the control input is converted into firing times, using knowledge of the thruster configuration, thruster force, control period, and duty cycle.

7.2.3 Estimation Loop

The estimation loop includes two major functions: PropagateState and EstimateState. The PropagateState uses the simulation time increment to propagate the state using the model of the current configuration. The EstimateState function implements an Extended Kalman filter (EKF) and simulates measurements in order to provide a realistic state estimate that includes noise. The EstimateState function is modular and can be easily incorporate a wide range of sensors. For this work, the SPHERES hardware testbed is used as a baseline system to model the sensors since information about the type of sensors and noise characteristics are easily available.

The estimation in the simulation models the SPHERES ultrasound receivers and gyroscopes as the sensors on the assemblers. Each assembler has 24 receivers and 3 gyroscopes. Receiver measurements are simulated to include noise based on beacon angle, receiver angle, distance to beacon, and ±1 mm wavelength noise. Gyroscope modeling includes incorporating the bias for each gyroscope, as well as a ±3 mrad/s noise. These sensors are processed using an EKF at 5 Hz. Gyroscope measurements are taken at 1 kHz and incorporated at a rate of 50 Hz. Ultrasound measurements are incorporated as they are received. The 5 beacons ping in sequence, roughly every 30 ms. The EKF then maintains the estimated state vector and covariance.

7.2.4 Metrics

Though the simulation is capable of calculating many objective functions, as shown in Table 7.1, three key metrics are explored in this section to demonstrate the difference in scenario performance due to model generation architectures.

The first metric is the total fuel used for the assembly. This metric is calculated throughout the simulation using the SPHERES thruster mass flow rate and thruster commands. A lower fuel usage per module is desirable as it reflects the efficiency of the assembly.

The second metric is the total assembly time. It is preferable to minimize the assembly time since a longer assembly time reduces the total operational mission duration. The assembly time is comprised of the time to execute all of the paths, but also includes the computation time required. This is particularly significant for the system identification methods.

Finally, the third metric is the time averaged root mean square of the tracking error, for position and velocity specifically. This metric captures how far the assembler strayed from its planned trajectory, which captures the accuracy of the assembly and the performance drop during transitions.

7.3 Results

7.3.1 ALMOST

For the ALMOST scenario, the assembly of the six modules was explored using 13 different model generation architectures (as per Table 7.2) and up to three assemblers. Figure 7-5 shows the total fuel used (cumulative across all assemblers) versus the model generation architecture method. The methods with the high fuel usage are those that employ adaptive control or system identification. This is an expected result. When the control system does not have sufficient knowledge of the system, it must expend resources to identify those properties in order to maintain equivalent control. The remaining methods, those that do not use adaptive control or system identification, perform very similarly because the mass property information used to set the control system parameters is the same though they set the parameters in different ways. This trend is consistent across the number of assemblers used for assembly.

Figure 7-6 shows the overall assembly time versus model generation method used, with the number of assemblers indicated by dashed backgrounds. A clear trend is that as the number of assemblers increases, the total assembly time decreases. This is expected since multiple assemblers can work simultaneously to move modules. For the groups mentioned in Figure 7-5, the increase from one to two assemblers seems the most significant, while the increase to the third assembler is a comparatively smaller improvement, except for methods using adaptive control. While certain methods seem to have a significantly higher assembly time, the total difference between method assembly times is on the order of 5 to 12 minutes. In a mission context, this variation is likely small enough to be insignificant; particularly if the mission is located far enough away that the communication delay is greater than this difference. For example,

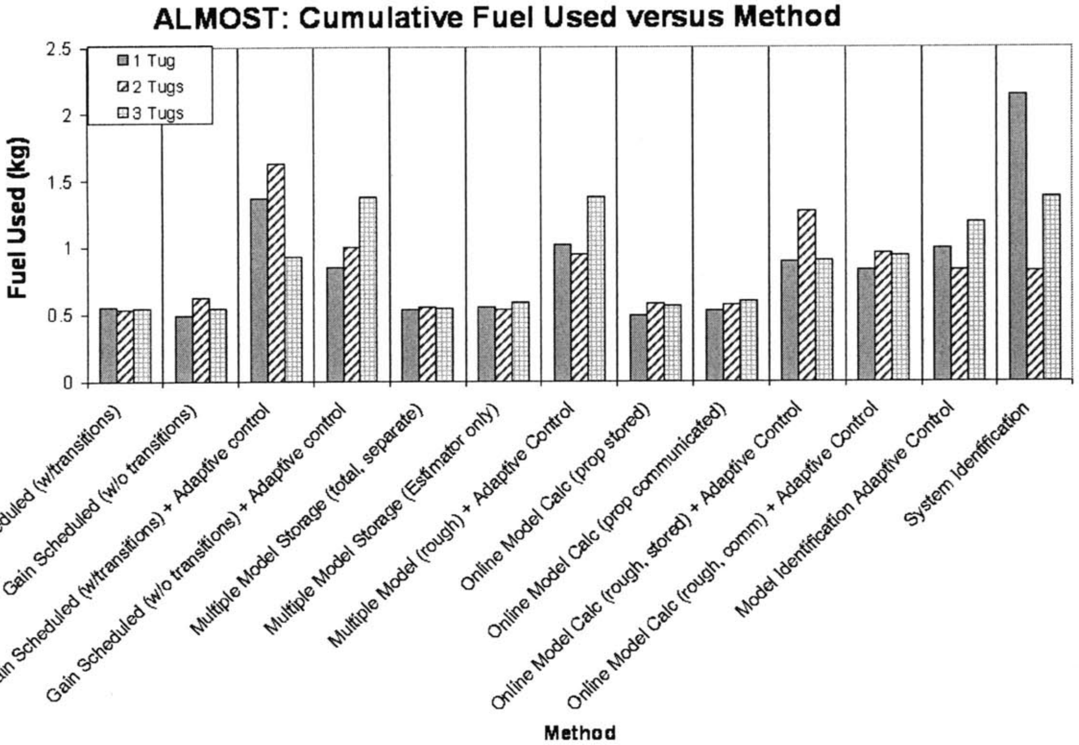


Figure 7-5: Cumulative Fuel Used (kg) vs Model Generation Method, Number of assemblers (dashed), ALMOST

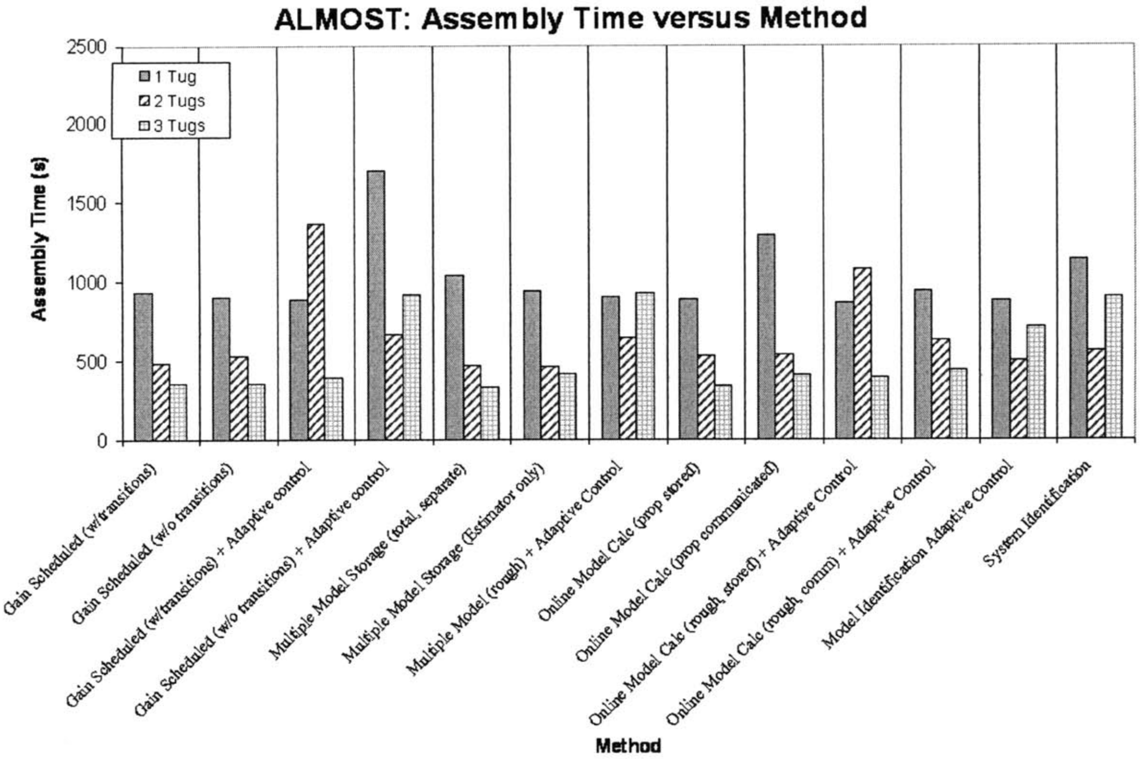


Figure 7-6: Assembly Time (s) vs Model Generation Method, Number of assemblers (dashed), ALMOST

locations in the Earth-Moon system are close enough where this difference is well above the communication time between Earth and the spacecraft. Locations at Mars and beyond introduce a one-way light time delay comparable to the differences in assembly times. Thus, as the communication delay increases, the impact of model generation architecture selection on assembly time decreases.

Figure 7-7 plots the average root-mean-squared (RMS) position error in meters and velocity error in meters per second versus the model generation method. It is expected that the points with lower RMS position error will also have lower RMS velocity error. Similar to Figures 7-5 and 7-6, the methods using adaptive control and system identification have a higher RMS error. This is expected since these methods have less knowledge of the system, and therefore take more time to achieve the desired tracking. A higher averaged error is the result of a high initial error due to unknown properties.

Results from Figures 7-5, 7-6, and 7-7 show that methods that utilize knowledge of

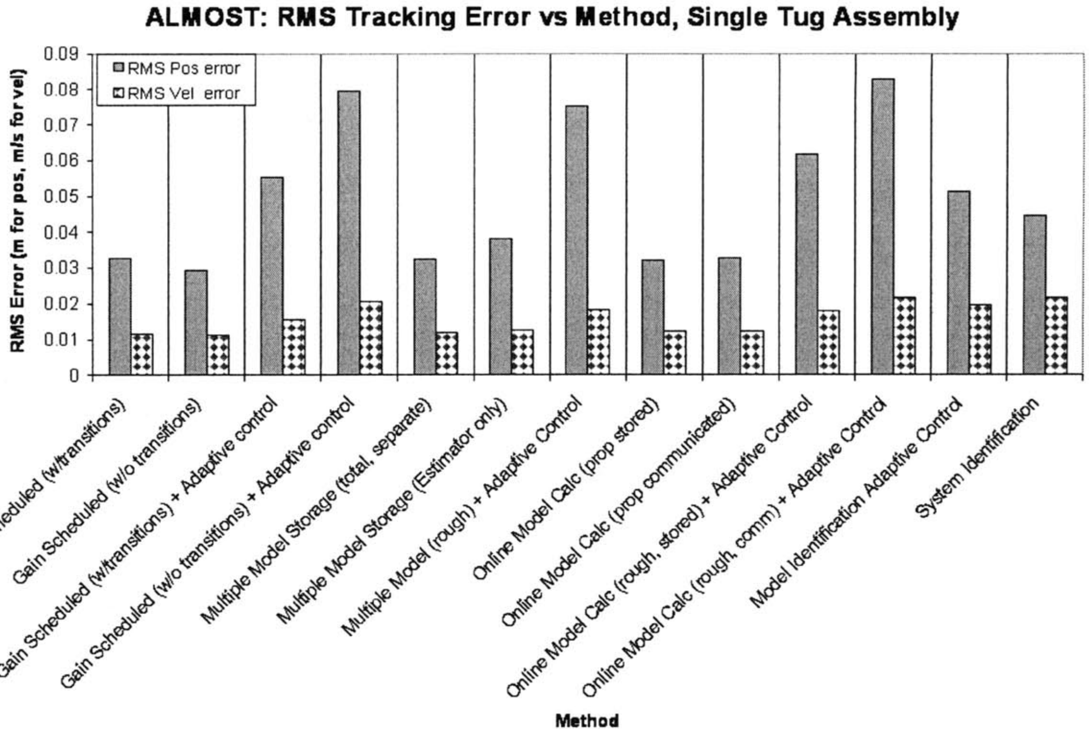


Figure 7-7: RMS Position error (m) and RMS velocity error (m/s) vs Model Generation Method, ALMOST

the mass properties to setup the control system perform better individually in terms of fuel used, assembly time, and error. Figure 7-5 shows that this trend holds for combined fuel and assembly time performance as well. Figure 7-8 plots the assembly time in seconds versus fuel used (kg) for each method. The objective is to have low fuel used and low assembly time. Therefore, the optimal methods are those that do not use adaptive control, as indicated on Figure 7-8. Of these methods, gain scheduling uses the least amount of memory space for this scenario.

These results show that for the particular scenario of a small telescope assembly, the choice of a gain scheduled method leads to better overall mission performance in terms of memory stored, fuel used, assembly time, and RMS error. These results are also useful in quantifying the additional cost in terms of fuel, time, and error if another method is selected for implementation. From Figure 7-8, one can extract the cost of obtaining the necessary mass property information online for this scenario. Compared to a baseline of the method with the best performance (i.e. gain scheduling), system

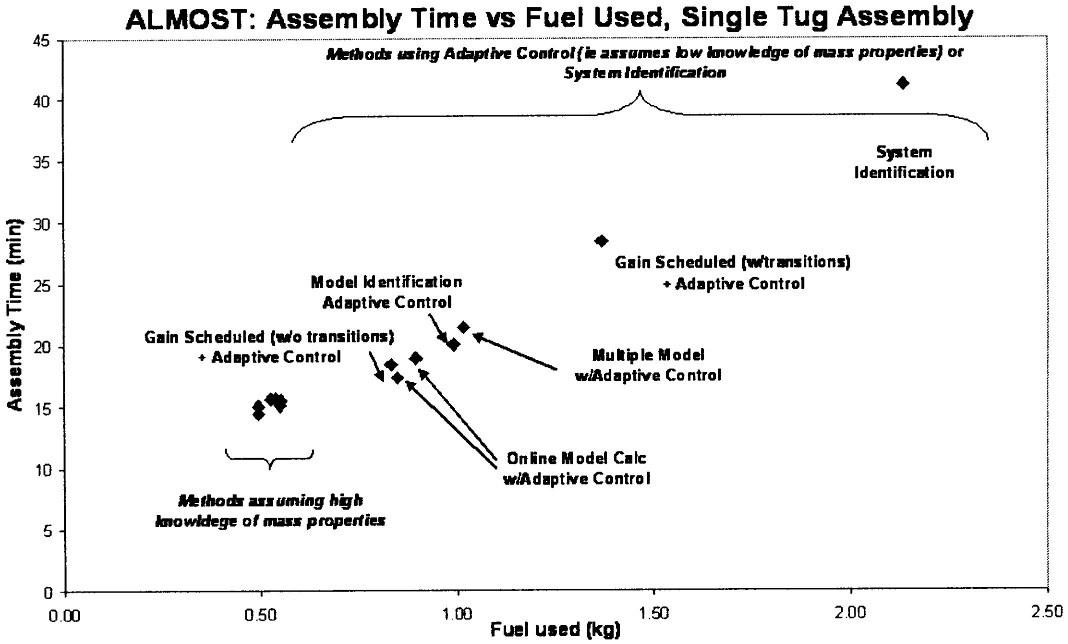


Figure 7-8: Fuel Used (kg) vs Assembly Time (s) vs Model Generation Method, ALMOST

identification methods have fuel costs four times larger than the baseline and time costs 2.5 times larger than the baseline. Understanding what this additional cost is for given assembly scenario allows for informed choices in the selection of a control system design.

Another aspect to consider is the sensitivity of the results presented in Figure 7-8. A brief analysis is conducted to determine how these results change if the six modules are not identical. If the six modules are similar, but assumed identical, the difference between the actual properties and assumed properties would cause slight variations in the fuel, time, and error. The exact nature of the variations are based on how the actual properties differ from the assumed properties. For example, if the actual mass is larger than the assumed mass, this would equate to controlling the system at a lower bandwidth. Fuel used should decrease, but assembly time and tracking error may increase. If the tracking error increases beyond the required performance specified by the assembly architecture, the modules can no longer be considered identical. If the module are not identical and are modeled separately, the performance difference occurs in the memory and computation time, as opposed to the assembly mission metrics of fuel, time, and tracking error. When the module are modeled separately, the tracking performance should be similar for all of the module because the gains are calculated separately for each module. Future work should include upgrading the simulation to include variations between modeled properties and assumed properties to quantitatively determine the sensitivity of the results.

7.3.2 ACRRES

Contrary to ALMOST, ACRRES is a self-assembly scenario, such that each assembler maneuvers itself into position. Thus, there are no configuration changes during assembly for each assembler. This scenario assumes all pieces attach together after all pieces have reached their target locations. Thus, the aggregation occurs for the final stationary assembled structure, which is not considered in the scope of this thesis. This assembly scenario is chosen to clearly show the effect of configuration changes, through comparison with the ALMOST scenario.

Three different cases are considered. Case 1 assigns the mass of a assembler to 4.3 kg, equivalent to a single SPHERES satellite. Case 2 and 3 use a mass of assembler twice and three times the mass of the assembler used in Case 1, respectively. Figure 7-9 shows the performance of fuel used versus method. Compared to Figure 7-5, the fuel used versus method is a more varied for the ACRRES scenario than the ALMOST scenario. The adaptive control methods still use slightly more fuel, through not as significantly as in the ALMOST scenario. The driving contribution of the differences is now the stochastic nature of the maneuvering, rather than a deterministic quantity, like differences in models used.

Figure 7-10 shows the performance of assembly time versus method. For a given assembler mass, the assembly time is fairly constant across methods. This is seen more clearly with M_{tug} = 12.9 kg, which show constant levels almost across the board, with only slightly higher assembly times for methods 12 and 13. Once again, these are the adaptive control and system identification methods. It is expected that these methods have slightly higher assembly times as it takes time to identify the systems.

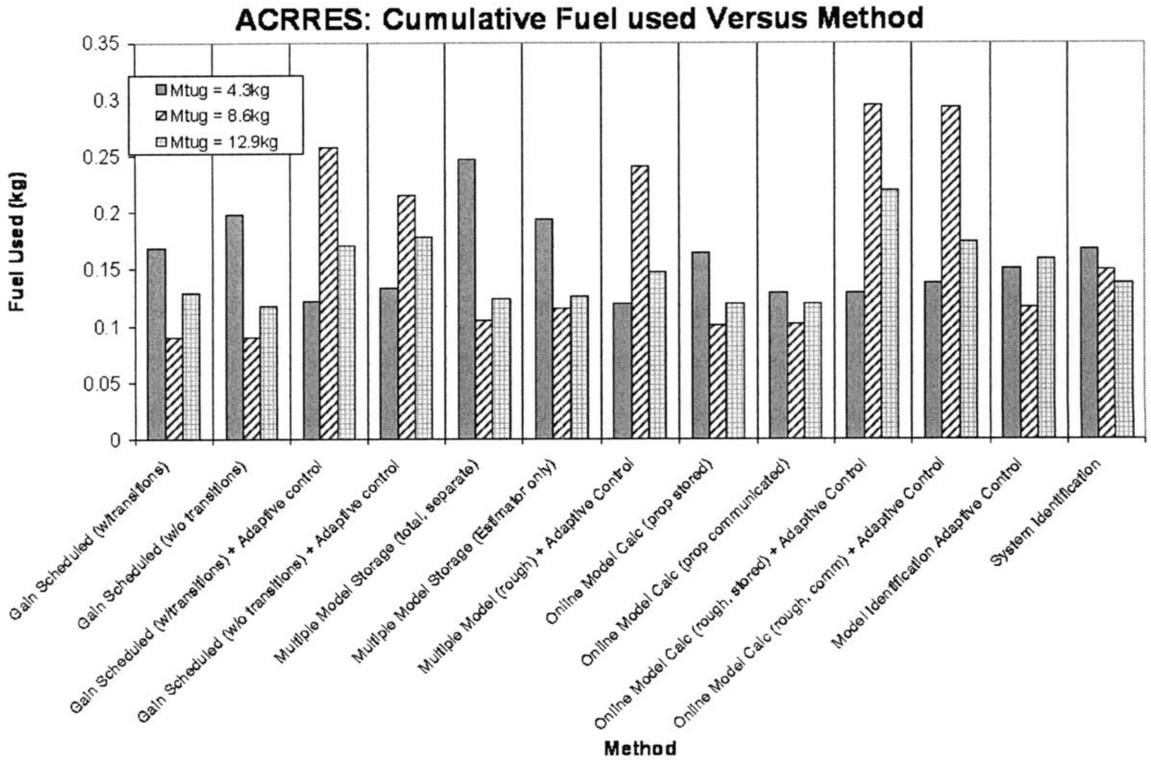


Figure 7-9: Cumulative Fuel Used (kg) vs Model Generation Method, Mass of assembler (dashed), ACRRES

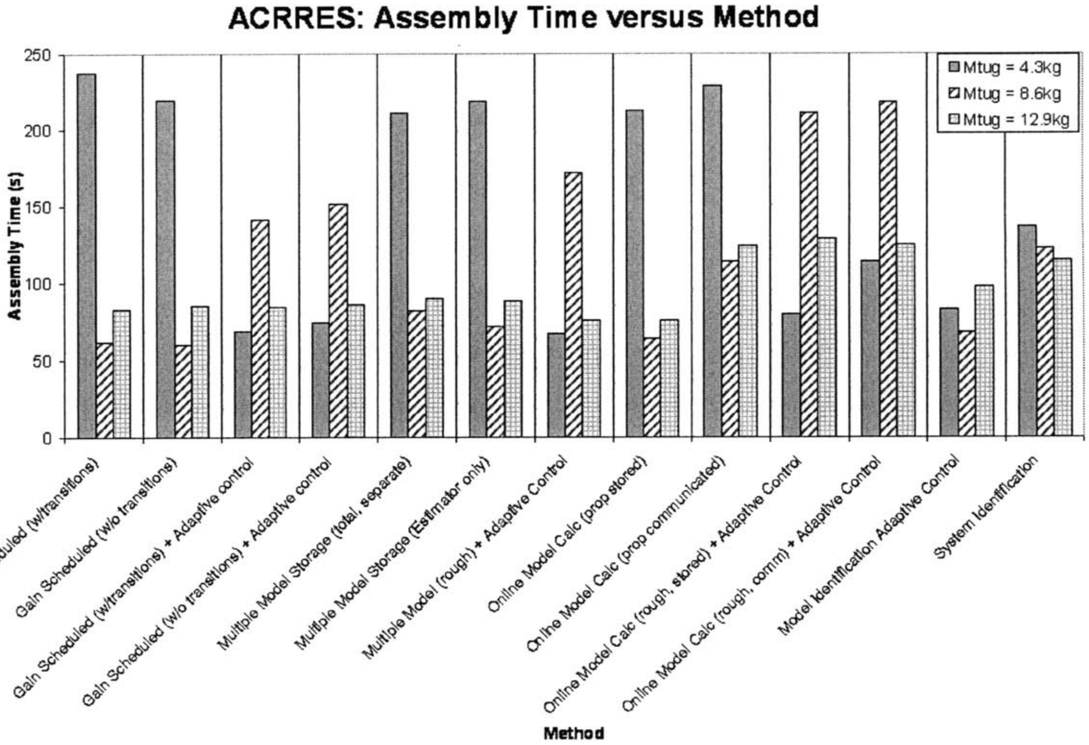


Figure 7-10: Assembly Time (s) vs Model Generation Method, Mass of assembler (dashed), ACRRES

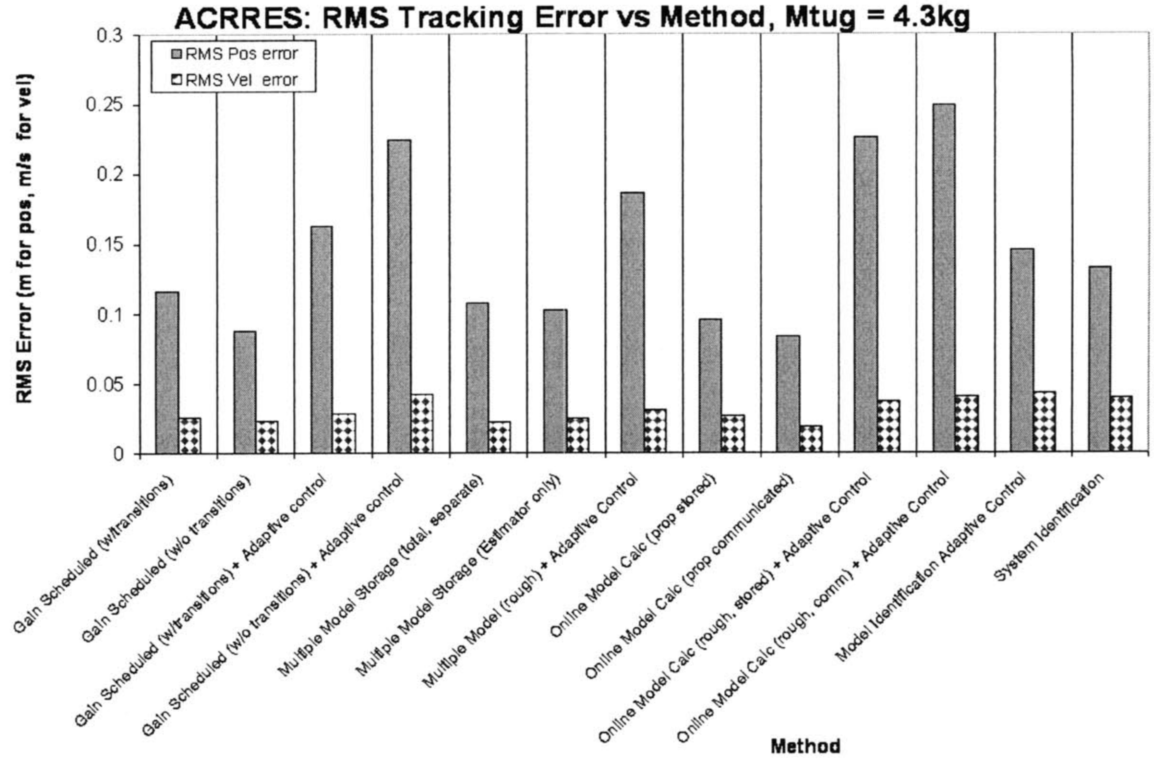


Figure 7-11: RMS Position error (m) and RMS velocity error (m/s) vs Model Generation Method, ACRRES

Figure 7-11 shows the performance of the RMS position error and velocity versus method. Similar to Figure 7-7, the points that have minimum RMS position error also have minimum RMS velocity error. The methods that use adaptive control have slightly higher tracking error, though not as high as in the ALMOST scenario.

Results from figures 7-9, 7-10, and 7-11 show that there are minimal differences in the individual performance of fuel used, assembly time, and RMS position error as a function of the method. Figure 7-12 shows the combined performance of fuel used and assembly time. Similar to the individual performance, the combined performance indicates that many methods provide the minimum fuel used and assembly time. The range of variation of fuel used and assembly time is small, particularly compared to Figure 7-8.

For this scenario, where each assembler maneuvers itself to self-assemble into a formation, there are minimal differences in performance as a function of model generation architecture. System identification architectures have slightly higher fuel used

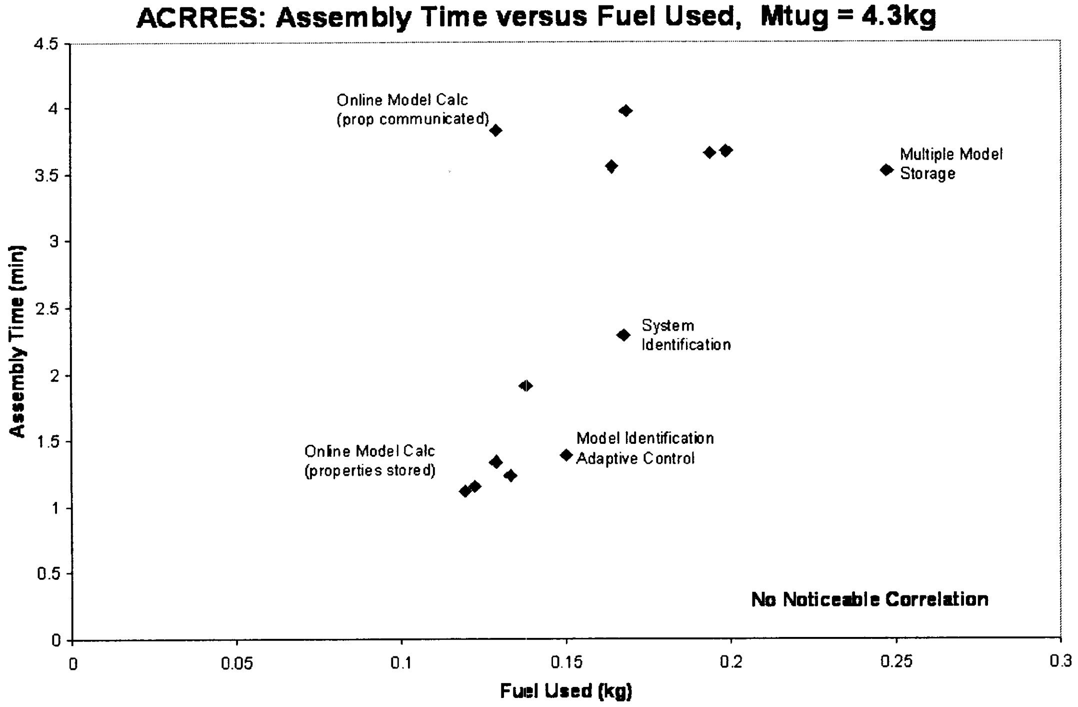


Figure 7-12: Fuel Used (kg) vs Assembly Time (s) vs Model Generation Method, ACRRES

and assembly times, indicative of the extra time needed to identify the properties of the assembler. Therefore, an engineer can select any architecture with equivalent performance. The differences between the architectures exist in the reconfigurable nature of the model generation architecture, its ability to adapt to changing properties. In this scenario, the assemblers do not undergo configuration changes while maneuvering to the desired location. Thus, it is expected that most methods should perform similarly. System identification and adaptive control methods perform slightly differently because the overhead associated with those methods are in fuel and time.

7.4 Conclusions

The assembly simulation is an important tool that enables the evaluation of the metrics performance for the different model generation methods based on an assembly scenario. The simulation was tested on two different scenarios: a six segment telescope tug assembly (ALMOST) and a three assembler self assembly (ACRRES). Results show that methods using knowledge of mass properties, such as gain scheduling, are preferred for the ALMOST scenario. The results also helped to quantify the additional cost on obtaining information if one were to select a method that identifies properties online. Contrary to ALMOST, the results for ACRRES showed that there is little performance difference between the different methods. This is expected as there are minimal mass property changes during maneuvering in the ACRRES scenario, as transitions only occur at the end of assembly.

By demonstrating that the trends are different for different scenarios, the assembly simulation tool shows its benefit in being able to quantify the impact of model generation architectures. This allows for more informed design, leading to more effective and efficient assembly designs. The modular design of the simulation tool allows it to be easily upgraded to account for more assembly scenarios. Future work involves upgrading the simulation to account for obstacle avoidance maneuvers, environmental disturbances, and detailed system identification maneuvering. These upgrades will generate a better fuel consumption estimate. Upgrades to expand the applicability of this simulation tool include modeling assembly via robotic arm, including electric propulsion based assembler, as well providing a user-definable mission concept (initial and final positions, module properties, and assembler type).

Chapter 8

Process for Selection of Model Generation Architectures

The proper selection of a model generation architecture is a critical open area in the current literature. By developing a process that can quantitatively compare and select model generation architectures, implementing autonomous assembly will become easier. This process is helpful not only in during critical design, but also during concept design to perform educated trades and further refine the overall assembly design.

The process of the selection of a model generation architecture developed in this thesis occurs in two phases. Phase one is to identify feasible architectures based on the amount of information available. Phase two is to determine which of the feasible architectures is the appropriate architecture based on a ranking of objective functions. However, enumeration of all possible selections of an appropriate architecture from the feasible set can become combinatorially large, based on how many objective functions and specific architectures are considered. Figure 8-1 enumerates different model generation architecture options to illustrate the breadth of the decision tree, determined by different types of architectures and the objective functions used to compare them. The large scope shown in Figure 8-1 shows that there is a need for a process to navigate this tree.

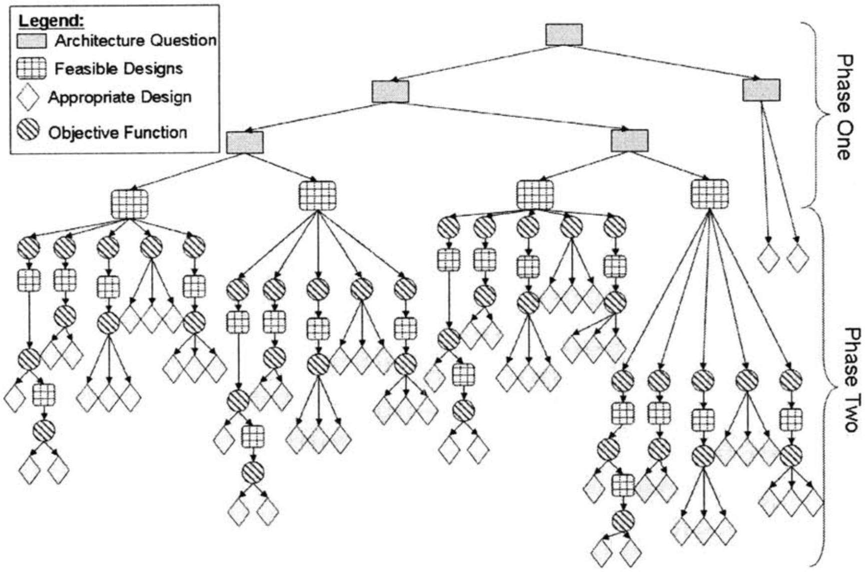


Figure 8-1: Process flow diagram to illustrate breadth

8.1 Definition of "Feasible" and "Appropriate" Design

The terms *feasible* and *appropriate* are used to describe the selected model generation architectures at the end of the phase one and phase two respectively. Each model generation architecture has a certain amount of required *a priori* knowledge about the mass properties.

- Feasible model generation architectures are architectures where the scenario satisfies the *a priori* knowledge requirements.

- Appropriate model generation architectures are the "perceived best choice based on the available information".

It would be incorrect to say that an appropriate architecture is optimal because many of the characteristics are hardware specific. The choice of the assembler could change the outcome of the model generation architecture selection. Thus, the appropriate model generation architecture is the "optimal" architecture, given a specific assembler hardware specification and assembly architecture specification.

8.2 Phase One: Identification of feasible architectures

Phase one of the process performs an elimination based on the knowledge of the assembly scenario. The elimination is based on a set of questions that are used to identify which model generation architectures can be implemented. In Figure 8-2, the boxes with the solid lines are architecture questions, which have a binary yes/no answer. Based on the response, the user follows the branch associated with that response, down to the next question. This process is repeated until they arrive at the set of feasible architectures, indicated by the boxes with dotted lines. These high level questions discern what the *a priori* information is known to the user.

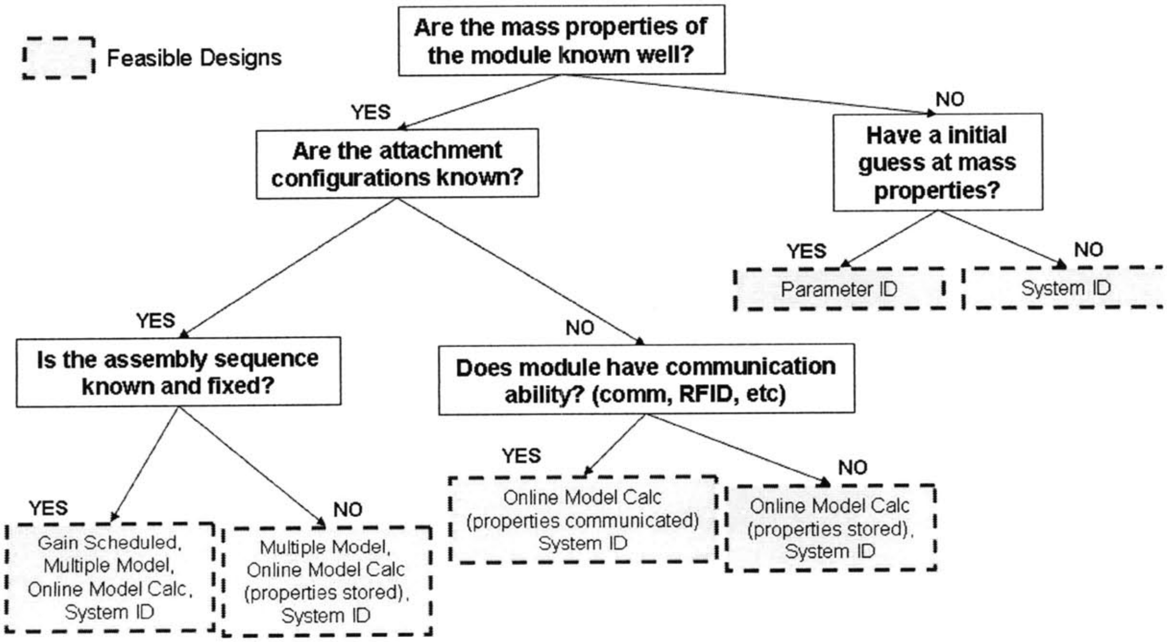

Feasible Designs
Are the mass properties of the module known well?
YES
NO
Are the attachment configurations known?
Have a initial guess at mass properties?
YES
NO
Parameter ID
System ID
YES
NO
Is the assembly sequence known and fixed?
Does module have communication ability? (comm, RFID, etc)
YES
NO
YES
NO
Gain Scheduled, Multiple Model, Online Model Calc, System ID
Multiple Model, Online Model Calc (properties stored), System ID
Online Model Calc (properties communicated) System ID
Online Model Calc (properties stored), System ID

Figure 8-2: Phase One process flow diagram, identification of feasible model generation architectures

8.2.1 Determination of Phase One questions

The questions used in Phase One are derived from the three critical variables described in Section 6.1.1, transitions, configurations, and module properties. The set of questions used in Phase One can expand based on the differentiation between model generation architectures and range of assembly scenarios considered. Each question in Figure 8-2 is described in detail below.

- Are the mass properties of the module known well?: This question is used to determine which regions of the spectrum to consider. Knowing the properties well indicates that all aspects of the spectrum can be considered, while not knowing the properties well necessitates some form of learning or identification. This question determines if the module properties is known.

- Have an initial guess at the mass properties?: This question is used to distinguish between architectures that perform complete upfront identification versus architectures that adapt during maneuvering.

- Are the attachment configurations known?: This question determines if the configurations variable is known. No knowing the configurations further restricts the feasible regions of the spectrum to those that must calculate or identify the model online.

- Is the assembly sequence known and fixed?: This question determines if the transitions variable is known. Not knowing the transitions eliminates architectures that integrate the assembly sequence with the control system.

- Does the module have communication ability?: This question serves to identify model generation architectures that place assumptions of available resources.

8.2.2 Accuracy Knowledge of Properties

An important aspect to consider is how well are properties known. What should be the cutoff accuracy of the mass property knowledge? For control reconfiguration applications, knowing the mass properties well means that one can implement a selected control system and always achieve desired performance. To determine a knowledge accuracy requirement, the baseline design, gain scheduling, is used, since it is the most restrictive method in terms of adapting to uncertainty in plant dynamics. Assume a simple second order system as before ($J\ddot{x} = u$, Equation 6.9). Figure 8-3 shows the simulation performance of an assembler following a spiral trajectory, using a PID controller with unit thruster saturation limits. Different values of percent error of the knowledge of the mass of the assembler were tested. Given the percent error, a random mass is initialized based on the true mass and the bounds determined by the percent error. For example, if the mass of the assembler is 10 kg and the percent error is 50%, then the mass used in the control system is randomly initialized between 5 kg and 15 kg. 500 iterations were run for each percentage error level. A simple noiseless dynamics propagation model is used, so the variation seen in this plot is solely due to the varying mass in the controller. The RMS position error is the difference between the desired trajectory and the actual position, which captures the effect of the error in the knowledge of the mass on the control system performance.

The RMS position error is seen to increase as the percentage of error in the knowledge of the mass property increases. If a required minimum tracking performance is specified, for example maintaining the tracking error to within 30% of the ideal, one can determine a cutoff of roughly 6% error in the knowledge of the mass. Similar analysis can be performed for the architecture under consideration to determine the cutoffs based on the baseline tracking performance and the required performance.

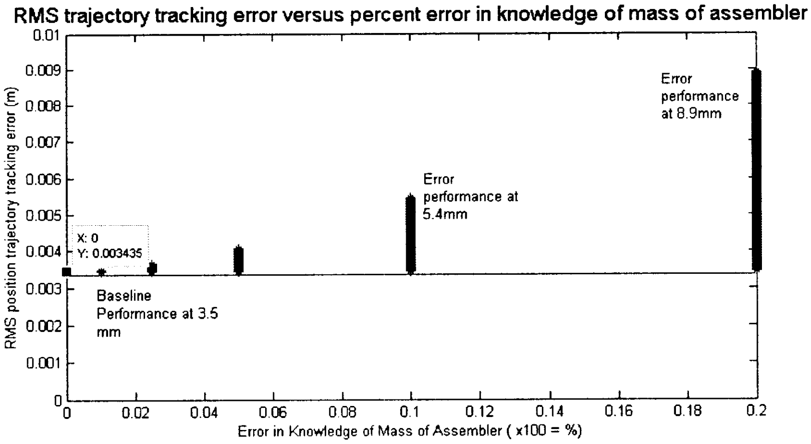


Figure 8-3: Performance change due to error in the knowledge of the mass

8.2.3 Generality of Phase One questions

The Phase One questions provide an initial set of questions to downselect based on the categorization of model generation architectures with respect to the three variables transitions, configurations, and module properties. The contribution of these questions is to show how to downselect regions on the spectrum of model generation architectures using succinct questions organized in a logical flow. These questions can be updated or augmented to tailor them to a specific assembly analysis. Assembly mission information can be used to further refine the feasible set of model generation architectures if factors such as risk, safety, or location eliminate certain architectures. Additional research in model generation architectures may warrant a new variable to distinguish designs, which should then be capture in the Phase One questions.

8.3 Phase Two: Determination of the Appropriate Architecture

After a set of feasible architectures have been identified, the next step is to select a single model generation architecture. The goal of phase two is to use objective functions to pare down the set of feasible architectures into a single architectures. Objective functions evaluate the performance of the model generation architecture, based on metrics such as those specified in Chapter 6. Selection and creation of the objective functions are discussed in detail in Section 6.2.4. The model generation architecture with the best value is the appropriate architecture that is output at the end of phase two. If a selected objective function results in multiple appropriate architectures, two techniques can be used to downselect to a single appropriate architecture, either a hierarchy of objective functions or a multi-objective function. A hierarchy of objective functions can be used to filter the feasible architectures, such that the architectures output of the first objective function are the input to the second objective function. If the desired objective functions have equal ranking but different weighting, a multi-objective function can be created to specify the desired weighting and used to select a single appropriate architecture in one step.

To illustrate phase two, the following objective functions are chosen from Table 6.3: Minimize Control Downtime, Minimize Mass, Minimize Ground Development, and Maximize Adaptability. Based on the first objective, Control Downtime, the selected methods in Figure 8-4 can be filtered to only include Gain Scheduling and Multiple model. Evaluating these architectures on the second objective does not result in reducing the number of feasible architectures since both techniques are perform equally in terms of Mass. The next step is to evaluate based on the third objective function of Ground Development time. Though gain scheduling has a higher TRL, the integration with the assembly sequence is higher for gain scheduled than for multiple model architectures. Multiple model architectures minimize ground development time more than gain scheduled architectures. Thus, the final appropriate architecture is multiple model architectures. The use of the fourth objective function is not needed because a single appropriate architecture is obtained after the third objective function evaluation.

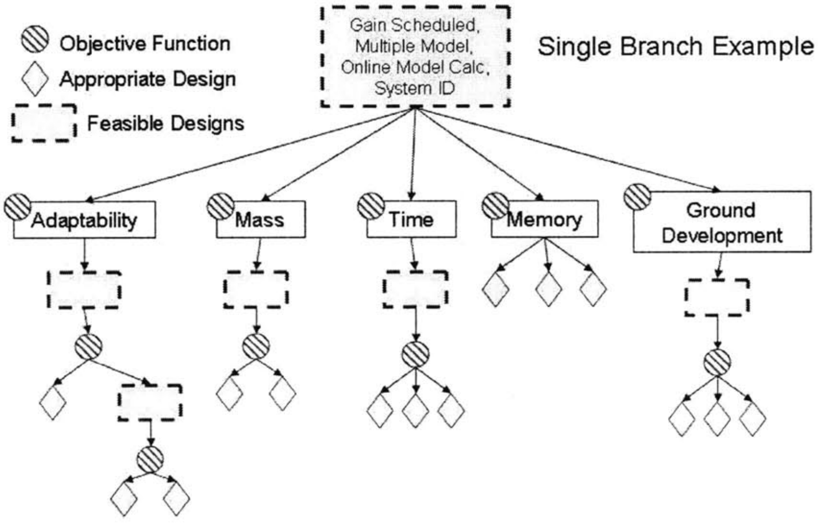


Figure 8-4: Phase two process flow diagram, determination of the appropriate design

Phase two essentially performance a single step optimization, based on the objective function. Thus, the weighting and ranking of objective functions significantly impact which model generation architecture is appropriate. This allows the same process to fulfill many assembly mission needs. However, the choice of the objective function should clearly reflect the mission priorities, otherwise the resulting appropriate architecture may not achieve desired mission performance.

8.4 Table of Model Generation Architectures

An enumerated set of model generation architectures is given in Table 8.1. This set is not all-inclusive, but is meant to be representative of the spectrum of model generation architectures. Table 8.1 gives a description of each architecture and the corresponding assumptions for implementation. The goal is to provide an example set of model generation architectures to demonstrate the metrics and to compare the performance of the architectures for a set of case scenarios. The model generation architectures are part of a control system, using an EKF and PID controller. Architectures that employ learning or adaptation are specifically labeled in Table 8.1. The architectures specified in Table 8.1 are implemented in the simulation described in Chapter 7.

Table 8.1: Set of model generation architectures

Method		Assumptions			Description
No.	Name	Transitions	Config Properties	Module Properties	
1	Gain scheduled (w/transitions))	Known	Known	Known	All information known *a priori*. The times for model transitions are hard coded. Control parameters for each configuration are pre-set based on quantities calculated on the ground.
2	Gain scheduled (w/o transitions)	Unknown	Known	Known	Transition times are not known ahead of time. The gains for each model (based on state/mass properties) are coded in. It requires a flag to be set real-time during assembly execution in order to switch between models
3	Gain scheduled (w/transitions) AC control	Known	Known	Known	Gains are coded into the model based on a low-fidelity model calculated on the ground prior to launch. Adaptive control is used to refine the model based on motion. Transitions known.
4	Gain scheduled (w/o transitions) AC control	Unknown	Known	Known	Gains are coded into the model based on a low-fidelity model calculated on the ground prior to launch. Adaptive control is used to refine the model based on motion. Transitions unknown.
5	Multiple model storage (total, separate)	Unknown	Known	Known	Full models are stored for each distinct configuration. Each configuration has a separate estimator and controller. Transitions between the models are probabilistic, based on measurements of the motion, or flag received indicating next model.
6	Multiple model storage (estimator only)	Unknown	Known	Known	Multiple estimators are stored, based on configuration. Controller is implemented through gain scheduling, based on configuration properties.
7	Multiple model storage AC control	Unknown	Known	Known	Low fidelity models are stored on board the assembler (both estimator and control). Adaptive control is used to refine the model to achieve maximum control performance.

8	Online model calc (prop stored)	Unknown	Unknown	Known	Mass properties of the modules are stored on board the assembler. These properties are used to calculate the actual model online, based on a flag that indicates which module has been docked to.
9	Online model calc (prop comm)	Unknown	Unknown	Known	Properties are obtained at the time of attachment from the module (ex RFID tags on docking ports). These properties are used to calculate the new model based on the module properties and the location of attachment.
10	Online model calc (prop stored, AC control)	Unknown	Unknown	Known	A low fidelity model is calculated online and implemented. Adaptive control is used to improve control performance. Module properties are stored in the assembler's memory.
11	Online model calc (prop comm, AC control)	Unknown	Unknown	Known	A low fidelity model is calculated online and implemented. Adaptive control is used to improve control performance. Module properties are communicated over at the time of attachment.
12	Adaptive control (MIAC)	Unknown	Unknown	Unknown	A Model Identification Adaptive Controller is used to identify the system parameters online during the maneuvering. No information is known prior to identification. Identification is performed while maneuvering.
13	System identification	Unknown	Unknown	Unknown	System identification (using fuel) is performed to identify the parameters in the system prior to maneuvering to the desired target. Identification is performed upfront prior to maneuvering.

Table 8.2: Scenario Description: SPHERES

Type of Assembly	Tug
Propulsion type	Thrusters
Location	ISS
Ntug	1
Nseg	1
Ntypes	1
Dseg	0.20 m
Mseg	4.3 kg
Mtug	4.3 kg
Final Pos	stack
Initial Pos	sparse

8.5 Metrics Validation

In order to have confidence in the ability of metrics to compare performance of model generation architectures, the metrics generated in Chapter 6 are validated on hardware for the architectures described in Table 8.1. Many of the heuristic expressions given in Section 6.2 depend on both assembly and hardware properties. Validation on SPHERES offers two benefits. First, it confirms that the heuristic expressions adhere to common sense and that they can be used to select the appropriate method. Second, by implementing the architectures on SPHERES, it provides a baseline set of values to compare architectures. Though the absolute values may increase or decrease on different hardware, SPHERES can be used as baseline hardware to select an initial appropriate architecture for a wide variety of scenarios.

8.5.1 Exercising the Process

The SPHERES hardware set-up was translated into a scenario to fully exercise the process of selecting a model generation architecture (Table 8.2). The hardware set-up consists of two SPHERES satellites, one acting as the assembler and one acting as the module. The expected outcome is that gain scheduling will be the appropriate architecture under most objectives since all mass properties are known and there are only two configurations. First, process phase one is exercised. The architecture questions are:

Table 8.3: Objective function rankings for SPHERES scenario

Case	Ranking				
1	Time	Mass	Memory	Ground Dev	Adaptability
2	Mass	Memory	Time	Ground Dev	Adaptability
3	Memory	Mass	Ground Dev	Time	Adaptability
4	Ground Dev	Mass	Memory	Adaptability	Time
5	Adaptability	Mass	Time	Ground Dev	Memory

- Are the mass properties of the module known well? → **YES**
- Are the attachment configurations known? → **YES**
- Is the assembly sequence known and fixed? → **YES**

Based on Figure 8-2, all designs are feasible for this scenario. To fully exercise process phase two, multiple rankings of the objective functions were selected. Table 8.3 shows the rankings selected, where each case specifies a ranked list of objective functions. Objective functions are listed in order of their ranking from left to right in the table. The results for these rankings are presented in the next section.

8.5.2 SPHERES Results

Four model generation architectures from Table 8.1 were implemented in the SPHERES testbed: gain scheduling, multiple model calculation, online model calculation with property storage, and online model calculation with properties communicated. Table 8.4 presents the hardware values of the variables described in Section 6.2. Since the architectures listed in Table 8.1 also include two different types of controllers, the memory specifications for the controllers are listed. The variable ϵ indicates a small value, that can be neglected.

Table 8.5 shows the results of exercising the Phase 2 process for the SPHERES scenario. The five objective rankings cases are shown. For each case, the objectives are listed in order of their rank. All of model generation architectures are evaluated for the first objective. The architectures with the best performance are selected and filtered down to the second objective. The -- mark indicates that the architecture has been eliminated from the previous step. This process is continued until only one

Table 8.4: SPHERES hardware performance

Variable	Value	Comments
Gain scheduling		
M_{cfg}	968 bytes	
m_{tot}	0 kg	
T_{flag}	ϵ s	
Multiple model method		
M_{mdl}	416 bytes	
M_{calc}	1280 bytes	
T_{calc}	ϵ s	
T_{flag}	ϵ s	
m_{tot}	0 kg	
Online model calculation		
M_{calc}	5056 bytes	
M_{seg}	416 bytes	
T_{calc}	ϵ s	
T_{flag}	ϵ s	
T_{comm}	1.8 s	
m_{tot}	0.11 kg	dry mass for RFID reader, if passive communication on tug
System identification		
T_{calc}	1 s	uses one control period to update
T_{flag}	ϵ s	
T_{id}	12 s	
m_{tot}	6.7 g	fuel used to identify model
Controllers		
MC_{pid}	88 bytes	
MC_{AC}	2784 bytes	
$m_{fuel-AC}$	ϵ kg	additional fuel used with adaptive control
$P_{drop-AC}$	ϵ	performance drop during adaptive control transition

architecture remains. Note, not all objectives in the ranked list may be used. Also, because this particular scenario only involves one module, the additional fuel mass due to adaptive control is not a big driver.

For the five cases, gain scheduling wins for case 1, 2, and 3. For this particular scenario, since all aspects are known and N_{types} is small, gain scheduling techniques minimize time and memory, so become the appropriate choice when those objectives are ranked high. System identification techniques become the appropriate choice when minimization of ground development and maximization of adaptability are ranked high, as in case 4 and 5. The weighting of the components in ground development objective greatly skews the preferred choice. For the form of the objective function used in this calculation (as specified in Chapter 6), the TRL contribution is not weighted as much as the integration or model fidelity. Therefore, system identification wins because it minimizes the fidelity of the model needed and integration with the sequence. However, if TRL is the driving concern, architectures such as multiple model methods become preferred since they have fairly high TRL but lower integration than gain scheduling.

8.6 Case Scenarios

As defined in Section 1.5.1, a scenario is a description of an assembly architecture that specifies the characteristics of the assembler, modules, initial and final positioning, and location and type of assembly. A case scenario is similar to a case study; it is a scenario invented to test the process of selection, as described in this chapter. Three case scenarios are selected to cover a range of assembly conditions, so that the selection process may be exercised in many different regimes. The following sections describe each case scenario, exercise the process on the scenario using the same set of objectives, and detail the appropriate architecture found. Note that it is not always necessary to apply all objectives to obtain an appropriate architecture. Some objectives, such as memory, can reduce the feasible set to one architecture in a single step, while others, such as time, may have many architectures of equal value. Thus, the ranking of the objective functions is critical since it has a large influence on the

Table 8.5: Results for Table 8.3 cases evaluated for SPHERES scenario

Case	Objective	Model Generation Architecture												
		1	2	3	4	5	6	7	8	9	10	11	12	13
1	Time (s)	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	1.8	ϵ	1.8	24	24
	Mass (kg)	0	0	0	0	0	0	0	0	–	0	–	–	–
	Memory (kB)	2.0	2.0	4.7	4.7	2.2	2.2	4.9	5.6	–	8.2	–	–	–
	Ground Dev	8.1	7.1	–	–	–	–	–	–	–	–	–	–	–
	Adaptability	–	√	–	–	–	–	–	–	–	–	–	–	–
2	Mass (kg)	0	0	0	0	0	0	0	0	0	0	0	0	0
	Memory (kB)	2.0	2.0	4.7	4.7	2.2	2.2	4.9	5.6	5.6	8.2	8.2	5.6	10.3
	Time (s)	ϵ	ϵ	–	–	–	–	–	–	–	–	–	–	–
	Ground Dev	8.1	7.1	–	–	–	–	–	–	–	–	–	–	–
	Adaptability	–	√	–	–	–	–	–	–	–	–	–	–	–
3	Memory(kB)	2.0	2.0	4.7	4.7	2.2	2.2	4.9	5.6	5.6	8.2	8.2	5.6	10.3
	Mass (kg)	0	0	–	–	–	–	–	–	–	–	–	–	–
	Ground Dev	8.1	7.1	–	–	–	–	–	–	–	–	–	–	–
	Time (s)	–	√	–	–	–	–	–	–	–	–	–	–	–
	Adaptability	–	√	–	–	–	–	–	–	–	–	–	–	–
4	Ground Dev	8.1	7.1	7.1	6.1	7.1	7.6	6.1	6.1	6.3	5.3	5.3	3.2	2.2
	Mass (kg)	–	–	–	–	–	–	–	–	–	–	–	–	√
	Memory (kB)	–	–	–	–	–	–	–	–	–	–	–	–	√
	Adaptability	–	–	–	–	–	–	–	–	–	–	–	–	√
	Time (s)	–	–	–	–	–	–	–	–	–	–	–	–	√
5	Adaptability	0.25	0.33	0.25	0.33	0.33	0.33	0.33	0.5	0.5	0.5	0.5	1	1
	Mass (kg)	–	–	–	–	–	–	–	–	–	–	–	1.2	6.7
	Time (s)	–	–	–	–	–	–	–	–	–	–	–	√	–
	Ground Dev	–	–	–	–	–	–	–	–	–	–	–	√	–
	Memory (kB)	–	–	–	–	–	–	–	–	–	–	–	√	–

Table 8.6: Scenario Description: ALMOST

Type of Assembly	Tug
Propulsion type	Thrusters
Location	ISS
Ntug	1
Nseg	6
Ntypes	1
Dseg	0.30 m
Mseg	6 kg
Mtug	4.3 kg
Final Pos	ring
Initial Pos	stack

final model generation architecture selected.

8.6.1 ALMOST

ALMOST is an assembly scenario where a single tug assembles a set of identical mirror segments through docking (Section 7.1.1). The defining characteristics of this scenario are that all the modules are identical and that the tug assembles them one at a time. This minimizes the total number of configurations. Table 8.6 specifies the details of the ALMOST scenario.

The first step in executing the selection process is to answer the architecture questions in phase one. The questions are:

- Are the mass properties of the module known well? → **YES**
- Are the attachment configurations known? → **YES**
- Is the assembly sequence known and fixed? → **YES**

Following the path of questions on Figure 8-2, it shows that all model generation architectures are feasible for this scenario. Since all of the information about the modules, how they dock, and the order is known, any of the architectures can be implemented.

The next step is to pare down the list of feasible model generation architectures into a single architectures by using process phase two. Applying the first objective,

Table 8.7: Scenario Description: ACRRES

Type of Assembly	Self
Propulsion type	Thrusters
Location	ISS
Ntug	3
Nseg	0
Ntypes	1
Dseg	0.20 m
Mseg	N/A
Mtug	4.3 kg
Final Pos	triangle
Initial Pos	sparse

control down time, to the feasible architectures reduces the feasible set to gain scheduled, multiple model architectures, and online model calculation (properties stored). The second objective is to minimize memory, which further reduces the feasible architectures to gain scheduling. Out of the various gain scheduling architectures, the third objective to minimize ground development time leads to the selection of **Gain scheduling with transitions known (Method 1)** as the appropriate method.

8.6.2 ACRRES

In ACRRES, individual vehicles self-assemble into an aggregated structure (Section 7.1.2. All vehicles have actuation and sensing capability, and use distributed control techniques to self-assemble. It is assumed, for this study, that the vehicles are SPHERES satellites. Table 8.7 specifies the details of the ACRRES scenario. ACRRES is chosen as a scenario because it has very few configuration changes.

The first step in executing the selection process is to answer the architecture questions in phase one. The questions are

- Are the mass properties of the module known well? → **YES**
- Are the attachment configurations known? → **YES**
- Does the module have communication ability? → **YES**

Table 8.8: Scenario Description: ISS-like

Type of Assembly	Tug
Propulsion type	Thrusters
Location	LEO
Ntug	1
Nseg	15
Ntypes	15
Dseg	2.95 - 4.91 m
Lseg	1.5 - 13.1 m
Mseg	1,880 - 19,300 kg
Mtug	4000 kg
Dtug	2 m
Final Pos	pre-designated (ISS)
Initial Pos	single item

Following the path of questions on Figure 8-2, it shows that all model generation architectures are feasible for this scenario.

The next step is to pare down the list of feasible model generation architectures into a single architecture by using process phase two. Applying the first objective, control down time, to the feasible architectures reduces the feasible set to gain scheduled, multiple model architectures, and online model calculation (properties stored). The second objective is to minimize memory, which further reduces the feasible architectures to **multiple model storage (Method 5)**. This is a single architecture, thus we have achieved the goal of selecting an appropriate model generation architecture, even before completing all objectives.

8.6.3 "ISS"

This scenario models the autonomous assembly of the ISS via a tug assembler. The number, diameter, and length of the modules are representative of the large modules of the ISS, however, this scenario does not include the solar panels. The tug is sized to be significantly smaller than the average module to stress the necessity for control reconfiguration. Table 8.8 specifies the details of the ISS-like scenario.

The first step in executing the selection process is to answer the architecture questions in phase one. The questions are

- Are the mass properties of the module known well? → **YES**

- Are the attachment configurations known? → **YES**

- Is the assembly sequence known and fixed? → **NO**

Following the path of questions on Figure 8-2, it shows that the feasible designs for this scenario are multiple model method, online model calculation, adaptive control, and system identification. Gain scheduling is not an option because the assembly sequence is not known.

Applying the first objective, control downtime, to the feasible designs reduces the feasible set to multiple model and online model calculation (properties stored). The second objective is to minimize memory, which further reduces the feasible designs to **multiple model storage (Method 5)**. Note that if the order of the objectives is reversed, to minimize memory first, the appropriate choice becomes Online Model Calculation (properties communicated) (Method 9).

8.7 Conclusions

This chapter provides a two step process for selecting an appropriate model generation architecture, given the assembly scenario and objective functions. Phase one of the process uses architecture questions to help the user to downselect to the feasible architectures. Process phase two uses a ranking of the objective functions to downselect the feasible architectures to a single appropriate design. The hardware values for the SPHERES are used to exercise the process on three case scenarios: a telescope assembly, a vehicle aggregation, and an autonomous assembly of the ISS.

Results from the different case scenarios show that different designs are appropriate for the different scenario. Sensitivity analysis shows trends between the design and the number of modules and number of types. The selection of the objective function rankings is also seen to have a significant effect on the selection of the appropriate model generation architecture. Further integration of the process into the assembly simulation could help to identify the sensitivity of the appropriate design to objective function specification, such as changes in ranking or weighting.

The three case scenarios validate the process and provide an example of how this process can be implemented for a real scenario. The fact that the three case scenarios produced different appropriate control system designs demonstrates that the selection of the model generation architecture has a measureable impact on performance of the control system.

Chapter 9

Conclusions and Future Work

9.1 Thesis Summary

On-orbit assembly is critically enabling technology for future space missions. It provides the ability to bypass launch vehicle mass and payload fairing size restrictions. However, current methods of on-orbit assembly employ humans to serve as the key decision makers in the assembly process. This leads to several disadvantages, such as long-lead time to complete an assembly, risk to the crew during EVAs, limitations in assembly locations, and higher cost associated with manned flight. These limitations can potentially be overcome by employing on-orbit autonomous assembly via robotic technology.

There are still many challenges associated with designing autonomous robotic systems, particularly for on-orbit assembly applications. These challenges range from standardized interfaces to autonomous sequence planning to reconfigurable control system design. This work focuses on the particular challenge of designing a model generation architecture to accommodate large mass and stiffness property variations associated with configuration changes during assembly. Background literature review shows that though some work has been done in model generation architectures, much of the work focuses on models that change due to failures. Model generation architectures that focus on mass property changes either assume complete knowledge of the system such that it can schedule out the models, or assume no knowledge at all and use system identification techniques to identify the model. Two gaps in literature are considered in this work: (1) the design of a type of model generation architecture called Online Model Calculation which serves to fill a gap where module properties are known but configurations are not known; and (2) how to compare and select a model generation architecture from a range of possible architectures.

The first gap addressed in this work is the lack of a design that balances the use of *a priori* mass property information with identification of the model online. Specifically, a gap exists when module properties are known, but configurations are unknown until attachment occurs. The model generation architecture developed in this thesis to fulfill this gap is called Online Model Calculation. The approach uses a parameterized control system based on a property vector p that is used to calculate the model online at the time of attachment. Thus, the properties do not need to be stored ahead of time, only need to be obtained at the time of attachment. The framework for this design was presented in Chapter 2 including the development approach, generalized configuration assumed in this work, and model calculation algorithm. Chapter 4 details the parameterization of the control system algorithms based on the property vector p, while Chapter 5 demonstrates this technique through experimental validation on the SPHERES hardware testbed.

The second gap addressed in this work is the selection of an appropriate model generation architecture for a particular assembly mission. The selection of an appropriate architecture can increase control system accuracy, save fuel, decrease computational processing time, and/or decrease development time. Chapter 6 provides a set of quantitative metrics that can be used to compare model generation architectures on multiple levels, as well as a description of the four main categories of model generation architectures. Chapter 7 documents a simulation tool developed to allow a user to evlaluate the metrics described in Chapter 6 for a given assembly architecture scenario. The simulation tool is demonstrated on two case scenarios: a telescope assembly and a self-assembly of three modules. Finally, Chapter 8 combined these elements into a two-phase process that steps a user through the selection of an appropriate architecture. The first phase downselects to feasible architectures based on assembly mission information, while the second phase downselects to a single appropriate architecture based on user-defined objective functions derived from the metrics specified in Chapter 6.

9.2 Contributions

The completion of this work has led to the main contribution of **development and validation of tools to address the selection and design of model generation architectures for on-orbit autonomous assembly**. Several smaller component contributions have been accomplished as well, as listed below.

Contributions from development of Online Model Calculation architecture:

- Developed a model generation architecture called 'Online Model Calculation', which calculates the new configuration at attachment and uses it to re-initialize the variables in the control system, such as control gains, mass property information in the estimator, and thruster configuration information.

- Developed a methodology for parameterizing a control system to support Online Model Calculation implementation

- Identified situations when one should account for the added mass or flexibility of a module based analysis of the performance degradation of maneuvering with and without accounting for the added module.

- Experimental validation of Online Model Calculation on hardware, specifically the parameterized control system algorithms.

- Upgrade of a adaptive controller implemented in [42] from 2D to 3D, and implemented in simulation and hardware.

Contributions from selecting a model generation architecture:

- Developed a two-phase process to select an appropriate model generation architecture

- Development and validation of a set of metrics that can be used to compare different architectures on multiple levels

- Developed an assembly simulation with a high fidelity control system that quantitatively maps the model generation architecture selection on the assembly performance.

9.3 Recommendations for future work

This thesis provides a foundation that allows engineers to compare and assess model generation architectures for autonomous on-orbit assembly mission applications. This is useful because it allows for trades to be made early on in the design process based on quantitative analysis, which can leads to a more efficient design with cost, mass, and time savings. This work can be extended in multiple beneficial directions.

- Validation on different types of assemblers: Much of the implementation and validation performed in this work was done using an assembler with docking as the primary form of attachment. One area of future work is to compare the performances differences between different types of assemblers. Adding in the functionality to compare between tugs, robotic arms, hybrid systems, and potentially also against human-assisted assembly. The decision of human versus robotic assembly may be a driving trade in the future.

- Servicing: Many of the control system challenges associated with assembly can also occur with on-orbit servicing. Online Model Calculation can be modified to the area of on-orbit servicing to provide similar contributions.

- Distributed control systems: This work assumes that the assembler is a centralized control system that has command and actuation ability over all other modules. An interesting future work would be to account for distributed assembler systems. If instead of a single assembler moving a module, there were two or more assemblers, how should the control system set-up be approached?

Exploring the different centralized and decentralized control architectures could show unique performance differences.

- Remote control operations: The implementation techniques developed to enable joint actuator control also present an opportunity for remote control operations, such that a primary module can control a secondary module, for both estimation and control, in cases where the secondary vehicle's processor is damaged. Initial work has been completed in this area to demonstrate feasibility.

- Batch filtered Kalman filter: The use of a batch filtered Kalman filter would allow for more versatility if trying to combine measurements from multiple satellites because it allows for more leeway as to when the measurements are received.

- Control system design for use with Online Model Calculation: The parameterized control system has many factors that drive performance. The design of the control system is a key research area to maintain stability and performance for all possible configurations, and still maintain the parameterized form.

- Model generation and control design for the assembled structure: This thesis considers the model changes due to the maneuvering of the assembler. However, the aggregation of the payload being assembled also incurs model changes. Techniques similar to those used in this work can be used to develop model generation architectures geared toward controlling the structure as it is assembled.

9.4 Concluding Remarks

The two main objectives of this thesis, which were to (1) develop a process for the selection of a model generation architecture and (2) develop a new architecture to balance *a-priori* information with online model determination, were accomplished. These methods were successfully verified and validated on multiple representative assembly scenarios. This work provides a solid foundation upon which to compare and design autonomous assembly missions.

Appendix A

Additional Literature Review

On-orbit assembly is a rich field drawing from many different areas. The literature related to this work was categorized into four main subfields: Sequence Planning, Robotics, Control and Estimation, and Systems Analysis. A full end-to-end on-orbit autonomous assembly must successfully combine aspects from all of these areas. Sequence Planning refers to the autonomous development of a set of trajectories that results in full assembly of the spacecraft. Robotics covers a broad range of the hardware technology that has mobility, docking, or assembly capabilities. Control and Estimation refers to the guidance, navigation, and control issues, primarily focusing on reconfigurable control. Systems Analysis refers to assembly architecture analysis and trade studies. The research for Control and Estimation was presented in Section 1.3.1. The remaining sections are presented here for supplemental information.

A.1 Sequence Planning

The assembly sequence is an important parameter in terms of the control system because it defines the operational state transitions. Thus, literature on sequence planning gives insight into the types of sequences prevalent and on assumptions made of the control system. One important distinction in sequence planning literature is self-assembly versus assembly via a robotic assembler. In the self-assembly of modules, work has been done with potential field methods [36], transition set rules [39], stochastic relaxation [77], as well as case studies for different applications (satellite [72], nanostructures [60]). Self-assembly methods assume that each segment has its own propulsion stage and control system. This results in a highly decentralized control system. The reconfiguration challenges for self-assembly are quite different as they primarily exist in the final stages as segments are attaching to each other. Thus, detailed analysis of self-assembly lies outside the scope of this work, since it does not employ a centralized controller to account for mass and stiffness property changes. Assembly via robotic assembler will be the primary sequencing method considered.

Work on sequence planning via robotic assembly has been done in a primarily mathematical formulation. Precedence knowledge [33], subassembly extraction through liaison graphs [45], cut set methods [63] [5] [25], and Contact State Network [83] are some examples of different approaches that have been considered. Traditional optimization methods of genetic algorithms [10] and simulated annealing [30] have also been done. Though much work has been done on sequence planning, to date the only application that has executed its plan on-orbit is the International Space Station [41] [16].

It is important to understand the assumptions of the sequence planning on the control system. Some methods require a physical model of the object [18] [45][63], while others only require knowledge of the constraints [5] [83]. Another issue to consider is if these methods can be implemented on-line or if they necessitate a precomputed sequence, as this has memory and computation issues, as well as specifies all transitions ahead of time. These issues will be considered by including sequence planning and the assembly sequence itself as an architecture parameter, with potential impact in the design of reconfigurable control systems.

A.2 Robotics

Robotics for assembly can be loosely classified into tugs, arms, and rovers. Examples of tugs include the SPHERES and AMPHIS projects. SPHERES is a formation flight testbed on the ISS used to further autonomous docking and reconfiguration tests [67] [65] [29]. AMPHIS is an autonomous ground docking testbed developed by the US Naval Postgraduate School [64]. Examples of robotic arms are the Ranger vehicle of the University of Maryland's Space Systems Laboratory [6] and the tug-arm hybrid vehicles of the Field and Space Robotics Laboratory at MIT [23]. Custom end effectors for assembly have also been designed [34]. A prime example of a rover is the legged ATHLETE rover [32], which is envisioned to help with assembly of planetary outposts.

In addition to these general classifications, hardware demonstrations of docking and assembly have been done with both manipulators and tugs. MRHE, developed by Marshall Space Flight Center, demonstrated assembly of three tugs, with deployable boom payloads. The MRHE work included mass reconfiguration for assembly, though no sensor reconfiguration [46]. KAMRO is a camera-based rover that demonstrates pick and place operations for a manipulator arm doing product assembly [31]. Minimal reconfiguration work was done since the assembler was much larger than the payload. Similar manipulator arm work was done for automotive assembly on a work bench [82]. Other hardware demonstrations of assembly include the NASA Langley truss assembly using a large external manipulator on a movable platform [21] and human EVA assembly experiments EASE/ACCESS on STS-61B.

Review of the robotics literature reveals that some demonstration of autonomous assembly have occurred. However, all of the demonstrations described have been very application specific. There does not exist a generic framework to enable autonomous assembly for any application. Similarly, reconfiguration has been implemented for some parameters, such as mass and inertia, but not all (ex. thruster configuration, sensor configuration). The different types of assemblers impact the control system framework, especially the model set-up for the assembler, the physical properties that need to be updated, and the capabilities of the assembler. The type of robotic assembler is considered in this work as an assembly architecture design parameter, with potential impact on the design of the reconfigurable control system.

A.3 Assembly Architectures

The design of the assembly architecture has a significant impact on the design of the control system of the tug. However, this impact has not been quantified in literature. In this work, assembly architecture is defined as the specification of the high level mission objectives for an assembly: **what** is being assembly, **where** is it being assembled, **how** is being assembled, **why** is it being assembled, and **who** is assembling it. Answers to the what, where, how, why and who questions specify the architecture. For the scenario in Figure 1-1, a set of segments of known mass (what) is being assembled at Earth-Moon Lagrange Point 1 (where) to form a space telescope (why) through docking/undocking maneuvers (how) by a propellant (who). Architecture parameters are the variables used to answer these questions. Examples are orbit, type of assembler, and number of assemblers.

Literature in assembly architecture has focused on high level comparison of assembly strategies or developing concept architectures for specific applications. Papers that focus on assembly strategies either compare methods of assembly [26], define metrics and requirements [70], or provide software tools to analyze assembly scenarios[19][27]. Space telescopes are a big application driver, with multiple architectures proposed [8] [47] [37]. Solar power stations are another application driver, using manipulators [76], human EVAs [69], or legged vehicles [28]. Finally, work for future manned habitats has a small section, with assembly requirements for the Constellation program [84] as well as assembling parts of a planetary stack [22].

Basu et al gives an architecture for a proposed autonomously assembled space telescope [8]. This paper provides a comprehensive set of architecture parameters such as telescope resolution, requirements for sensors and actuators, number of assemblers, and accuracy requirements at various stages of assembly. The objective in this concept is to minimize the number of segments and to optimize segment size, to achieve a final telescope diameter of 10m. The paper also provides a table of potential robotic assemblers, with specifications for their capability. While Basu et al indicate key drivers, they do not do a quantitative calculation of any of those parameters, nor does it specify the mapping between architecture parameters and the selection of an assembler.

Gralla and de Weck [26] provides a partial quantitative analysis on how assembly strategies can affect overall assembly objectives for human-lunar mission applications. The paper analyzes five methods of assembly: self-assembly, single tug assembly, multiple tug assembly, in-space refueling, and propellant storage on modules. The advantages this paper presents are that it provides a quantitative analysis of how these five architectures map into overall metrics. For example, the paper shows a trend for how number of modules affects the overhead mass. The assembly method is modeled through launch and orbit analysis. ΔV is calculated based on the amount necessary to move the segment to the appropriate orbit. However, for large scale assembly, much maneuvering is done in close proximity operations. Rodgers [62] does consider close proximity operations in his investigation of the assembly of a segmented primary mirror for a space telescope. The main comparison made is between using propellant-based versus electromagnetic-based propulsion systems for the tug. Quantitative trends for subsystem mass and time to assemble are given. However, both Gralla and de Weck and Rodgers require a more detailed analysis to map the interaction into the control system specifically. Only high level objectives, such as total assembly time, are considered. Parameters not considered in these works include the method of sequencing or the type of robotics used (ex. manipulator [21], hybrid manipulator-tug [23], and tug [67] systems).

Appendix B

Simulation User's Manual

B.1 Initialization

The initialization section of the simulation sets up the simulation based on the inputs of the scenario and method. Based on the scenario, the following parameters are initialized: number of payload, number of tugs, initial states of tug and payloads, final target states for payload, and mass property information for tug and payload (mass, inertia, thruster configuration, and sensor configuration). The controller to be used is set by the method; additionally, the metrics specified in Chapter 6 are calculated for the method. The property information is used to calculate the dynamics model used in the simulation.

B.1.1 Init Scenario

InitScenario initializes the simulation to the specified assembly scenario under analysis. The simulation currently accomodates two different assembly scenarios, as described in Section 7.1. For each scenario, there are 9 cases which span a range of assembly models, varying the number of tugs, the mass of the tugs, and the number of segments. The inputs to the InitScenario function are the scenario number and the case number. The outputs of the InitScenario function are:

- Assembly type (*AssemType*): The type of assembly specifies the mechanism by which payloads are connected together. Possible options are tug based, robotic arm, self assembly, as well as hybrid systems. For this thesis, only tug based and self assembling systems are considered, though the functionality exists to augment the simulation to broaden its scope to other types of assembly.

- Propulsion Type (*PropType*): The type of propulsion type used for assembly impacts the control architecture, path planning, and tug sizing. For this thesis, only chemical propulsion systems are considered, though the functionality exists to augment the simulation to include other forms in the future. Possible types of propulsion can include electric propulsion, robotic arms using joint motors, or even human powered.

- Location: The location of the mission can be used to specify the environmental disturbances, such as J2 effects, drag, gravity, etc. Currently, the location assumed is inside the environment inside the ISS. Thus, no gravitational or atmospheric effects are included.

- Tug Properties: If the assembly type is specified as a tug-based assembly, the InitScenario function also initializes the tug properties. These properties are based primarily of the SPHERES satellite. For self-assembling systems, the tugs are both the assembler vehicle and the payload.

  - Number of tugs (*Ntug*): Indicates the total number of tugs used in the assembly scenario. Tugs can simultaneously move pieces.

  - Diameter of tug (*Dtug*): The diameter of tug is used for path planning purposes to indicate where to specify the target locations with respect to the tug's geometric center.

  - Mass of tug (*Mtug*): The value is the dry mass of the tug. It is used in the control system to specify position gains and control performance.

  - Inertia of the tug (*InertiaTug*): The inertia of the tug is used to specify attitude gains and control performance.

  - Specific Impulse (*Isp*): The specific impulse is set if chemical propulsion is chosen as the propulsion type. It is a measure of the efficiency of the propellant and is used to size the fuel mass based on the estimated change in velocity needed.

  - Thruster mass flow rate ($\dot{m}$): The thruster mass flow rate is a hardware characteristic of the thrusters. It is used to calculate the fuel used during assembly, using the commanded thruster on times.
  - Center of Mass (r_{cg}): The center of mass is specified with respect to the geometric center of the tug. This information is used to calculate the model after docking to a payload.
  - Thruster locations (r_{gcSPH}): The thruster locations are given with respect to the center of the tug. The information is used to calculate the thruster commands.
  - Thruster force directions (F_{SPH}): The thruster force directions are given with respect to the body frame of the tug and is used to calcuate the thruster commands.
  - Thruster output force (thr_{force}): The force from a single thruster is used for path planning calculations to know how steep to design the trajectory.
  - Receiver locations (RX_{POS}): The receiver locations are given with respect to the geometric center of the tug. This information is used in the estimator to convert the sensor measurements into an estimated state.
  - Receiver directions (RX_{DIR}): The receiver directions are given with respect to body frame of the tug. This information is used in the estimator to convert the sensor measurements into an estimated state.
  - Control bandwidths ($w_{n_{pos}}$,$w_{n_{att}}$): The nominal control bandwidths for the tug are output to the controller to calculate the nominal gains for the tug only system.

- Payload Properties: If the assembly type specified is tug-based assembly, the payload properties are initialized. These payloads are passive objects with no actuation capability. Some of the properties listed below are for the docked system, when a payload is attached to a tug. It is assumed that only one payload is attached to a tug at any given time.

- Number of payloads ($Nseg$): The total number of payloads to be assembled gives a magnitude dimension to the assembly. More payloads means a larger final structure. The value is a function of the case number. It is varied for each case to give a spectrum of assembly scenarios, small to large.
- Number of distinct types of payloads ($Ntypes$): The number of types of payloads distinguishes how many unique payloads are present. This is a critical piece of information for the reconfigurable control system methods to size the memory and computation required.
- Diameter of payload ($Dseg$): The outer diameter of payloads is used for path planning purposes to determine where to specify the targets for the center of the tug and payload system.
- Height of payload ($Hseg$): The height of the payload is used when calculate how to position the payloads in their initial stowed configuration.
- Mass of payload ($Mseg$): The dry mass of a payload is used to calculate the control gains.
- Inertia of docked structure ($Inertia_{docked}$): The docked inertia is used to calculate the attitude control gains.
- Center of mass in docked configuration ($r_{cgDocked}$): The center of mass in the docked configuration is used to calculate thruster commands.
- Thruster locations in docked configuration ($r_{gcDocked}$): The thruster locations, in docked configuration, are required to calculate the thruster commands.
- Thruster force directions after docking (D_{docked}): The thruster force directions, in the docked configuration, are required to calculate the thruster

commands. The docking could have invalidated some thrusters due to blockage or plume impingement.

- Target properties: Based on the scenario, a initial and final positioning is specified.

  - Tug and Payload initial state ($tug-initial-cfg$, $payload-initial-cfg$): Possible initial positioning geometry are stack or sparse. For stack initial positioning geometry, the initial state of each tug and payload is calculated by placing them all in a vertical row. The tugs are placed at the top, with the payloads below them. A minimum separation distance of approximately 20cm is given between objects. For sparse positioning geometries, the initial location is randomly generated within a given spherical radius. The object can be anywhere within the outer radius of the placement sphere. This positioning geometry is used for self-assembling systems, while the stack positioning geometry is used for tug-based systems.
  - List of target locations and attitudes ($ctrlTargets$, $quatTarget$): The target location and attitudes are based on a final positioning geometry, such as hexagonal ring or triangle. Based on the information about the final positioning geometry, number of payloads, and dimensions of the tugs and payloads, the InitScenario function calculates the final target locations and attitudes for each tug, after assigning an equal number of payloads to each tug. This set of targets is expanded out into maneuver targets for the tug to reach the desired final payload location.
  - Docked status at each target ($dockedStatus$): The docked status value specifies whether the tug is docked during the specific target maneuver. This values is used to select control gains, models, etc.

B.1.2 Init Method

The InitMethod function initializes the metrics associated with the method chosen, specifically those that are not calculated by the assembly execution in the simulation.

Sixteen methods have been implemented, that correspond to those given in Table 8.1. The metrics values are based on the SPHERES properties as specified in Chapter 6. For each method, the following metrics are set in this function: Total memory required; Memory per model; Memory per payload; Processing time required; Sensing equipment needed; Communication bandwidth needed; Additional dry mass; Ground system required; Ease of changing between models; Ease of model incorporation; Integration with assembly sequencing; Total control downtime; Control downtime per maneuver; Method of model calculation; and Fidelity of measured model needed.

The metrics of fuel used to calculate model and performance drop while transitioning are extracted from the overall control performance from the assembly execution.

B.1.3 Init State Variables

The simulation uses a 13 element state vector to track of tug. The elements of this state vector are position (3), velocity (3), attitude in quaternion format (4), and angular rates (3), as given in Equation B.1. The state for each tug is initialized in the tug's initial position as determined by the function InitScenario.

$$x = \begin{bmatrix} x & y & z & v_x & v_y & v_z & q_1 & q_2 & q_3 & q_4 & w_x & w_y & w_z \end{bmatrix} \tag{B.1}$$

Other variables that are initialized for the tug are: the state error, a 13 element vector initialized to zero; the thruster durations, a 12 element vector modeling 12 thrusters initialized to zero; estimator covariance, initialized to a 12 by 12 matrix with the following diagonal entries (0.001, 0.001, 0.001, 0.0002, 0.0002, 0.0002, 0.1, 0.1, 0.1, 0, 0, 0); and the innovation, which is initialized to zero. The 3 element position is tracked for each payloads. InitStatVariables initializes the position of the payload to the payload's initial position as determined by InitScenario.

B.1.4 Calc Model

The CalcModel function takes in the mass properties information about the tug and the payloads and creates two state space models. The first model is for the undocked configuration. The continuous time A matrix is based on a double integrator system, with a submatrix that accounts for the quaternion propagation (Eqn. B.2).

$$A = \begin{bmatrix} 0_3 & I_3 & 0_4 & 0_3 \\ 0_3 & 0_3 & 0_4 & 0_3 \\ 0_3 & 0_3 & \frac{1}{2}\begin{bmatrix} 0 & w_z & -w_y & w_x \\ -w_z & 0 & w_x & -w_y \\ w_y & -w_x & 0 & w_z \\ -w_x & w_y & w_z & 0 \end{bmatrix} & 0_3 \\ 0_3 & 0_3 & 0_4 & 0_3 \end{bmatrix} \tag{B.2}$$

The mass property information is accounted for in the determination of the B matrix. The B matrix is calculated by first determining the linear and angular acceleration from each thruster by multiplying the thruster force by the thruster direction and dividing by the mass or inertia respectively. Equation B.3 shows the formula for the calculation of the linear and angular acceleration, where f_{thr} is the force per thruster, m is the mass, I is the 3x3 inertia tensor, r_{pos} is the position of the thrusters in the body frame of the tug, and D is the force direction of each thruster in the x, y, and z directions. The mass of the satellite is determined based on the dry mass of the tug, the tug's current fuel mass, and whether the payload is attached or not.

$$a_{lin} = \frac{f_{thr} * D}{m} \quad a_{ang} = I^{-1}\left(r_{pos} \times \left(f_{thr} * D\right)\right) \tag{B.3}$$

The overall continuous time B matrix, a 13 by 12 matrix, in the tug body frame, is compiled from these accelerations, as given in Equation B.4.

$$B = \begin{bmatrix} 0_3 \\ a_{lin} \\ 0_4 \\ a_{ang} \end{bmatrix} \tag{B.4}$$

The B matrix given in Equation B.4 is converted from body frame to inertial frame using the current attitude state of the tug via Equation B.5, where I_4 and I_3 are the identity matrcies of size 4 and 3 respectively and

$$\Theta_{B2G} = \begin{bmatrix} q_4^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_3 q_4) & 2(q_1 q_3 + q_2 q_4) \\ 2(q_1 q_2 + q_3 q_4) & q_4^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_1 q_4) \\ 2(q_1 q_3 - q_2 q_4) & 2(q_2 q_3 + q_1 q_4) & q_4^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

$$B_{inertial} = \begin{bmatrix} \Theta_{B2G} \\ \Theta_{B2G} \\ I_4 \\ I_3 \end{bmatrix} B_{body} \tag{B.5}$$

Both A and B matrices are converted from continuous frame to discrete frame using Matlab's *c2d* function with a time step of 0.2 seconds. This process is repeated for the docked case. For the docked case, the mass used is the sum of the mass of the tug and the mass of the payload. The inertia is calculated based on the parallel axis theorem and assumes that the payload docks to a pre-defined docking port located along the X body axis of the tug. The function calculates the model first in the initialization phase. However, the CalcModel function is also the basis of the PropagateModel function, so is run at each time step.

B.2 Control Loop

The control loop section is the largest and most important section of the simulation since the objective is to quantify the impact of control system designs on assembly.

B.2.1 Get Targets

The set of final payload positions output from InitScenario are expanded to consist of three major maneuvers for a tug based assembly. Each payload consists of a maneuver to approach the docked payload, pull out the payload from the stack, move upwards with the payload, then a final maneuver to push the payload into its final position. The GetTargets function is a very simple function; it takes in the list of total targets and the current progress of the assembly. The function outputs the current target state for the payload being assembled. For self-assembly scenarios, there is only one maneuver per tug/payload, which is to move to the final target location.

B.2.2 Path Planner

A path is planned from the current state to the specified target state obtained from GetTarget. The trajectory planned is based on the current mass of the tug (specifically whether it is docked or free-flying) and the specification of the propulsion type. In this work, a Bang-Bang profile was used to achieve minimum time. The maximum acceleration is calculated so that the thrusters are not saturated, where $a_{max} = \frac{1}{10}\frac{f_{thr}}{mass}$. The switching time is calculated as

$$t = \sqrt{\frac{\|d - x\|}{a_{max}}} \tag{B.6}$$

After the path is planned, it is sent to the controller as a time sequence of target states. For tug-based assembly, each payload has three maneuvers. The first maneuver is for the tug to approach the payload in the stack and attach to it. The second maneuver is to pull it out of the stack. Finally, the third maneuver is to move it to its final position.

B.2.3 Get Error

The tug attempts to follow the trajectory by calculating the error, which is the difference between desired state given by the trajectory and estimated state. The error is fed into the controller.

B.2.4 Controller

The controller used is set by the method. The two options are proportional-integral-derivative (PID) control and model reference adaptive control (MRAC). The gains for both controllers are calculated based on the mass and inertia of the tug. The gain calculation laws used in the script are discussed in detail in Chapter 4. The controller outputs a six element control vector of forces and torques.

B.2.5 Control Allocation

This control vector gets fed into the mixer, which converts the control vector from forces and torques, into thruster on/off commands. The mixer uses a pulse-width modulation scheme; the control input is converted into firing times, using knowledge of the thruster configuration, thruster force, control period, and duty cycle. The cycle is repeated for a single path for the length of the path, until the tug is within 2 cm of its final target position. Afterwards, the next payload target state is obtained. The whole cycle is repeated for each payload until the assembly is complete.

B.3 Estimation Loop

The estimation in the simulation models the ultrasound receivers and gyroscopes as the sensors on the tugs. Each tug has 24 receivers and 3 gyroscopes. Receiver measurements are simulated to include noise based on beacon angle, receiver angle, distance to beacon, and ±1 mm wavelength noise. Gyroscope modeling includes incorporating the bias for each gyroscope, as well as a ±3 mrad/s^2 noise. These sensors are processed using an Extended Kalman filter (EKF), which runs at 5 Hz. Gyroscope measurements are taken at 1 kHz and incorporated at a rate of 50 Hz. Ultrasound measurements are incorporated as they are received. The 5 beacons ping in sequence, roughly every 30 ms. The EKF then maintains the estimated state vector and covariance.

B.4 Metrics

Three main assembly metrics are used to compare the different control systems and scenarios from the metrics detailed in Chapter 6. The first metric is the total fuel used for the assembly. This metric is calculated throughout the simulation using the SPHERES thruster mass flow rate and thruster commands. A lower fuel usage per payload is desirable as it reflects the efficiency of the assembly. The second metric is the total assembly time. It is preferable to minimize the assembly time since a longer assembly time reduces the total operational mission duration. The assembly time is comprised of the time to execute all of the paths, but also includes the computation time required. This is particularly significant for the system identification methods. Finally, the third metric is the time averaged root mean square of the error. This metric captures how far the tug strayed from its planned trajectory, which captures the accuracy of the assembly and the performance drop during transitioning

Bibliography

[1] Bigelow aerospace.

[2] Middeck active control experiment. http://ssl.mit.edu/flight/mace.html, 1994.

[3] Space transportation costs: Trends in price per pound to orbit 1990-2000. memo, Futron Corporation, 2002.

[4] The vision for space exploration. Technical report, Nation Aeronautics and Space Administration, 2004.

[5] Maria Joao Abrantesl, C. White, and Ann Nicholson. Minimal infeasible sets of connections: A representation for efficient assembly sequence planning. *IEEE*, pages 275–280, 1995.

[6] David L Akin, Brian Roberts, Kristin Pilotte, and Meghan Baker. Robotic augmentation of eva for hubble space telescope servicing. In *Space 2003*. AIAA, September 2003.

[7] Mark Baldeserra. A decision-making framework to determine the value of on-orbit servicing compared to replacement of space telescopes. Master's thesis, MIT, 2005.

[8] Santanu Basu, Terry S Mast, and Gary T Miyata. A proposed autonomously assembled space telescope (aast). In *Space 2003*. AIAA, September 2003.

[9] R Berisio, L Vitagliano, A Vergara, G Sorrentino, L Mazzarella, and A Zagari. Crystallization of the collagen-like polypeptide (ppg)10 aboard the international space station. *Acta Crystallographica*, 58:1695–1699, 2002.

[10] F. Boizneville, C. Perrard, and J. M. Henrioud. A genetic algorithm to generate and evaluate assembly plans. *IEEE*, pages 231–239, 1995.

[11] Jovan Boskovic, Sai-Ming Li, and Raman Mehra. Reconfigurable flight control design using multiple switiching controllers and on-line estimation of damage related paramters. In *IEEE International Convference on Control Applications*, 2000.

[12] P Chandler, M Pachter, and M Mears. System identification for adaptive and reconfigurable control. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, 18, 1995.

[13] Allen Chen. Propulsion system characterization for the spheres formation flight and docking testbed. Master's thesis, MIT, 2002.

[14] Yoonhyuk Choi, Hyochoong Bang, and Hyunjae Lee. Dynamic control allocation for shaping spacecraft attitude control command. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, number AIAA 2006-6040, 2006.

[15] Soon-Jo Chung. *Nonlinear Control and Synchronization of Multiple Lagrangian Systems with Application to Tethered Formation Flight Spacecraft.* PhD thesis, Massachusetts Institute of Technology, 2007.

[16] Steven Y Chung, John D Wanagas, and Stephen Wright. Impact of restructuring on space station freedom assembly sequence. In *AIAA Space Programs and Technologies Conference.* AIAA, March 1992.

[17] John B. Davidson, Frederick J. Lallmant, and W. Thomas Bundick. Integrated reconfigurable control allocation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, number AIAA 2001 4083, 2001.

[18] Luis S. Homem de Mello. Sequence planning for a robotic assembly of a tetrahedral truss structure. Technical report, NASA Jet Propulsion Laboratory, 1994.

[19] Donald DeLaquil and Robert Mah. Advanced engineering software for in-space assembly and manned planetary spacecraft. In *AIAA Space Programs and Technologies Conference*, 1990.

[20] Nitin Dhayagude and Zhiqiang Gao. A novel approach to reconfigurable control systems design. *Journal of Guidance, Control, and Dynamics*, 19:963–966, 1996.

[21] William Doggett. Robotic assembly of truss structures for space systems and future research plans. *IEEE*, 2002.

[22] John T Dorsey, Judith J Watson, and Robin Tutterow. Structural conceptst for a lunar transfer vehicle aerobrake which can be assembled on orbit. In *AIAA*.

[23] Steven Dubowsky and Peggy Boning. The coordinated control of space robot team for the on-orbit construction of large flexible space structures. In *ISSS International Conference Robotics and Automation*, 2007.

[24] John Enright, Mark Hilstad, Alvar Saenz-Otero, and David W Miller. The spheres guest scientist program: Collaborative science on the iss. In *IEEE Aerospace Conference*. IEEE, March 2004.

[25] Jing Fan, Yang Ye, and Jia-Mei Cai. Multi-level intelligent assembly sequence planning algorithm supporting virtual assembly. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[26] Erica Gralla and Oliver deWeck. On-orbit assembly strategies for human space exploration. In *International Astronautical Congress*, 2005.

[27] Stephen Boyd Hall. Large space structures assembly simulation.

[28] Gregory Scott Hickey, Brett Kennedy, and Tony Ganino. Intelligent mobile systems for assembly, maintenance and operations for space solar power. Technical report, NASA Jet Propulsion Laboratory, 1999.

[29] Mark Hilstad. A multi-vehicle testbed and interface framework for the development and verication of separated spacecraft control algorithms. Master's thesis, MIT, 2002.

[30] D. S . Hong and H. S. Cho. Generation of robotic assembly sequences using a simulated aneealing. In *IEEE International Conference on Intelligent Robots and Systems*, 1999.

[31] Andreas Hormann and Ulrich Rembold. Development of an advanced robot for autonomous assembly. In *IEEE International conference on robotics and automation*, 1991.

[32] http://www robotics.jpl.nasa.gov/systems/system.cfm?System=11. Jpl robotics: The athlete rover.

[33] Y.F. Huang and C.S.G Lee. An automatic assembly planning system. *IEEE*, pages 1594–1599, 1990.

[34] Shin ichiro Nishida and Tsuneo Yoshikawa. A new end-effector for on-orbit assembly of a large reflector. In *IEEE*, 2006.

[35] Michel Ingham, Simon Nolet, David W. Miller, and Swati Mohan. Autonomous control reconfiguration for robotic exploration systems (acrres),. Proposal to NASA Jet Propulsion Laboratory, December 2008.

[36] Dario Izzo and Lorenzo Pettazi. Equilibrium shaping: Distributed motion planning for satellite swarm. In *iSARIS 2005*, 2005.

[37] Dario Izzo, Lorenzo Pettazzi, and Mark Ayre. Mission concept for autonomous on orbit assembly of a large reflector in space. In *56th International Astronautical Congress*, 2005.

[38] Robert Jacques. *Online System Identification and Control system design for flexible structures*. PhD thesis, Massachusetts Institute of Technology, 1994.

[39] Chris Jones and Maja J. Mataric. From local to global behavior in intelligent self-assembly. In *IEEE International Conference on Robotics & Automation*, 2003.

[40] Carole Joppin and Daniel Hastings. On-orbit upgrade and repair: The hubble space telescope example. *Journal of Spacecraft and Rockets*, 43(3), May-June 2006.

[41] James T Kaidy and William G Bastedo. Space station assembly sequence planning: an engineering and operational challenge. 1988.

[42] Jacob Katz. Estimation and control of flexible space structures for autonomous on-orbit assembly. Master's thesis, Massachusetts Institute of Technology, 2009.

[43] Lt Col Fred Kennedy. Orbital express space operations architecture. http://www.darpa.mil/tto/programs/oe.htm.

[44] E.M.C Kong, D.W. Kong, S.A. Schweighart, L.M. Elias, R.J. Sedwick, and D.W. Miller. Electromagnetic formation flight for multi-satellite arrays. *Journal of Spacecraft and Rockets*, 41:659–666, 2004.

[45] Sukhan Lee and Yeong Gil Shen. Assembly planning based on subassembly extraction. *IEEE*, pages 1606–1611, 1990.

[46] Edward A LeMaster, David Schaechter, and Connie K Carrington. Experimental demonstration of technologies for autonomous on-orbit robotic assembly. In *Space 2006*, 2006.

[47] Charles F Lillie. On-orbit assembly and servicing for future space observatories. In *Space 2006*, 2006.

[48] Ryan Lim. Staged attitude metrology pointing control and parameteric integrated modeling for space based optical systems. Master's thesis, Massachusetts Institute of Technology, 2006.

[49] Lennart LJung. *System Identification: Theory for the user.* prentice hall, 2005.

[50] Peter Maybeck and Richard Stevens. Reconfigurable flight control via multiple model adaptive control methods. In *IEEE Transactions on Aerospace and Electronic Systems*, volume 27, 1991.

[51] Tesfay Meressi and Brad Paden. Gain scheduled hinf controllers for a two link flexible manipulator. *Journal of Guidance Control and Dynamics*, 17:537, 1994.

[52] David W. Miller Swati Mohan and Jason Budinoff. Assembly of a large modular optical telescope (almost). In *SPIE Telescopes*, 2007.

[53] Swati Mohan. Reconfiguration methods for on-orbit servicing, assembly, and operations with application to space telescopes. Master's thesis, Massachusetts Institute of Technology, 2007.

[54] NASA. The james webb space telescope. www.jwst.nasa.gov, 2010.

[55] Gunter Niemeyer and Jean-Jacques E Slotine. Performance in adaptive manipulator control. In *Proceedings of the 27th Conference on Decision and Control*, 1988.

[56] Simon Nolet. *Development of a guidance, navigation and control architecture and validation process enabling autonomous docking to a tumbling satellite.* PhD thesis, MIT, 2007.

[57] ISS Program Scientist's Office. International space station experiment and facility results publications. website, February 2010. http://www.nasa.gov/missionpages/station/science/experiments/Publications.html.

[58] Alexander G Parlos and John W Sunkel. Adaptive attitude stability and control for space station / orbiter berthing operations. 1992.

[59] Kevin Readman. Universal docking port repeatability metrology. Technical report, NASA Goddard Space Flight Center, 2009.

[60] Aristides A. G. Requicha. Building shapes by self-assembly. In *Proceedings of the Shape Modeling International 2004*, 2004.

[61] Dr. Charles M. Reynerson. Spacecraft servicing - first order model for feasibility and cost effectiveness. In *AIAA Space 2001 Conference*, 2001.

[62] Lennon Rodgers. Concepts and technology development for the autonomous assembly and reconfiguration of modular space systems. Master's thesis, MIT, 2005.

[63] E Rohrdanz, H. Mosemann, and F. M. Wahl. Highlap : A high level system for generating, representing, and evaluating assembly sequences. *IEEE*, pages 134–141, 1995.

[64] Marcello Romano and Jason Hall. Atest bed for proximity navigation and control of spacecraft for on-orbit assembly and reconfiguration. In *Space 2006*, 2006.

[65] Alvar Saenz-Otero. The spheres satellite formation flight testbed: Design and initial control. Master's thesis, MIT, 2000.

[66] Alvar Saenz-Otero. *Design Principles for the Development of Space Technology Maturation Laboratories Aboard the International Space Station.* PhD thesis, MIT, 2005.

[67] Alvar Saenz-Otero and David W. Miller. Spheres:a platform for formation flight research. In *UV/Optical/IR Space Telescopes: Innovative Technologies and Concepts II conference.* SPIE, August 2005.

[68] Joseph H. Saleh, Elisabeth Lamassoure, and Daniel E. Hastings. Space systms flexibility provided by on-orbit servicing: Part 1. In *AIAA Space 2001 Conference*, 2001.

[69] T. Sours, R. Lovely, and D. Clark. Photovoltaic module on-orbit assembly for space station freedom. *IEEE*, pages 251–254, 1989.

[70] Stuart K Stephens and Harvey J Willenberg. Metrics for in-space telescope assembly techniques. In *IEEE Aerospace Conference*, 2003.

[71] RP Stowe, DL Pierson, DL Feedback, and AD Barrett. Stress-induced reactivation of epstein-barr virus in astronauts. *Neuroimmunomodulation*, 8(2):51–58, 2000.

[72] Hideyuki Tanaka, Noritaka Yamamoto, Takehisa Yairi, and Kazuo Machida. Autonomous assembly of cellular satellite by robot for sustainable space system. *IAC-05-D1.2.04*, AIAA IAC 2005, 2005.

[73] SPHERES Team. Spheres third iss test session report. Technical report, Massachusetts Institute of Technology, 2005.

[74] Spilios Theodoulis and Gilles Duc. Gain scheduled autopilolt synthesis for an atmosphere re-entry vehicle. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2008.

[75] S Trappe, D Costill, PM Gallagher, A Creer, JR Peters, H Evans, DA Riley, and RH Fitts. Exercise in space: Human skeletal muscle after 6 months aboard the international space station. *Journal of Applied Physiology*, 106:1159–1168, 2009.

[76] Hiroshi Ueno, Vickram Mangalgiri, Steven Dubowsky, Takeshi Sekiguchi, Mitsushige Oda, and Yoshiaki Ohkami. Simulation, analysis, and experimenta of on-orbit assembly behavior of flexible structure by cooperative robots. In *ISTS*, 2004.

[77] Katsuyuki Ukegawa and M. C. Natori. Concept of self-assembly of space structures systems using autonomous modules. In *54th International Astronautical Congress of the International Astronautical Federation,*, 2003.

[78] MM Weislogel, R Jenson, Y Chen, SH Collicott, J Klatte, and M Dreyer. The capillary flow experiment aboard the international space station: Status. *Acta Astronautica*, 65:861–869, 2009.

[79] James R Wertz and Wiley Larson. *Space Mission Analysis and Design.* Space Technology Library, 1999.

[80] Bong Wie. *Spacecraft Vehicle Dynamics and Control.*

[81] Edward Wilson, Chris Lages, and Robert Mah. On-line, gyro-based, mass-property identification for thruster-controlled spacecraft using recursive least squares. In *The 2002 45th Midwest Symposium on Circuits and Systems*, volume 2, pages 334–337. IEEE, August 2002.

[82] Dun Xiangming, Sun Bin, Zhang Weijun, and Yang Ruqing. Design of a flexible robot assembly demo system. *IEEE 4th World Congress on Intelligent Control and Automation*, 2002.

[83] Tsuneo Yoshikawa, Yasuyoshi Yokokohji, and Yong Yu. Assembly planning operation stategies based on the degree of constraint. *IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS*, pages TH0375–6, 1991.

[84] Douglas Zimpfer, Peter Kachmar, and Seamus Tuohy. Autonomous rendezvous, capture, and in-space assembly: Past, present, and future. In *1st Space Exploration Conference: Continuing the Voyage of Discovery.* AIAA, Jan 2005.