Reconfigurable Thruster Selection Algorithms for Aggregative Spacecraft Systems

by

Christopher Michael Jewison

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

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Author .....
Department of Aeronautics and Astronautics
May 22, 2014

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

Certified by .....
Alvar Saenz-Otero
Director, Space Systems Laboratory
Thesis Supervisor

Accepted by .....
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Reconfigurable Thruster Selection Algorithms for Aggregative Spacecraft Systems

by

Christopher Michael Jewison

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

The vast majority of today's space systems launch to orbit as completely assembled spacecraft stowed within a single launch vehicle. However, there is demand for large space systems (e.g., large space telescopes, fuel depots, space habitats, and solar power stations) that are overly limited by the lifting capability and fairing size of one launch vehicle. By separating large space systems into modules on multiple launches, these restrictions can be lifted, given a method to assemble and reconfigure the modules once on orbit. Specific control challenges associated with this reconfiguration need to be overcome before on-orbit assembly can be proven to be cost and resource effective. Some of these challenges have been addressed through adaptive and robust controller design, however an area that has not been explored deeply enough is that of thruster selection algorithms. In spacecraft control, thruster selection algorithms determine which thrusters to fire to produce a commanded force or torque on the system. Thus, these algorithms are critical for implementation of new controllers. To solve some of the control challenges associated with a system gaining additional thrusters, the goal of a reconfigurable thruster selection algorithm is to adapt to the new mass properties and thruster layout of the aggregated spacecraft, while optimizing for fuel consumption, precision, and agility. In this thesis, the proposed methodology is presented. A simulation for testing these algorithms is described, and results detailing the success of these reconfigurable thruster selection algorithms are discussed. In addition, results from preliminary testing of these algorithms using the MIT Synchronized Position Hold, Engage, Reorient, Experimental Satellites (SPHERES) facility in the three degree-of-freedom ground testbed are reported.

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

Thesis Supervisor: Alvar Saenz-Otero
Title: Director, Space Systems Laboratory

Acknowledgments

The work in this thesis related to ARMADAS was supported by the United States Air Force Space and Missile Systems Center (SMC) under NASA contract #NNH11CC25C. The research performed on the MIT Phoenix testbed was supported by Aurora Flight Sciences for the DARPA Phoenix project under DARPA-BAA-13-12. The work related to the SPHERES Halo was supported by NASA and DARPA under the InSPIRE-II contract, #NNH11CC26C. In addition, the entirety of this research was conducted with Government support under and awarded by the Department of Defense (DoD), Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. The author gratefully thanks these sponsors for their generous support that enabled this research.

I would like to thank the MIT Space System Laboratory and my advisors, Professor David Miller and Dr. Alvar Saenz-Otero, for allowing me to participate and contribute to the SPHERES program and team. I would like to extend a big thank you to the current graduate student SPHERES team—Bruno, Bryan, David, Drew, Duncan, Dustin, and Tim—for hanging together through thick and thin and always managing to pull solid work together.

Special thanks to Bryan McCarthy and David Sternberg for working by my side for two full years. The level of success with the ARMADAS, Phoenix, and InSPIRE-II projects is largely credited to their dedicated work ethic and perseverance. I will always remember the endless month of January. Also, a special thanks to Bruno Alvisio for being a true Argentinian friend.

Finally, I would like to thank my family for always supporting my decisions and pushing me to do my best throughout my entire life. This thesis could not have been completed without your constant love and support.

Contents

List of Figures

List of Tables

Nomenclature

ADC	Attitude Determination and Control
AFS	Aurora Flight Sciences
API	Application Programming Interface
ARMADAS	Agile Reconfigurable Modules with Autonomous Docking for Assembly and Servicing
ATLAST	Advanced Technology Large-Aperture Space Telescope
CAD	Computer Aided Design
CMG	Control Moment Gyroscope
CONOPS	Concept of Operations
DAC	Direct Adaptive Control
DARPA	Defense Advanced Research Projects Agency
DOF	Degree of Freedom
ExpV2	SPHERES Expansion Port Version 2
FDIR	Fault Detection Isolation and Recovery
GNC	Guidance Navigation and Control
GUI	Graphical User Interface

IMU	Inertial Measurement Unit
ISS	International Space Station
JEM	Japanese Experiment Module
JWST	James Webb Space Telescope
LQR	Linear Quadratic Regulator
MBPE	Model-Based Planning and Execution
MIT	Massachusetts Institute of Technology
NASA	National Aeronautics and Space Association
RARC	Resource Aggregated Reconfigurable Control
SLQP	Sequential Linear-Quadratic Programming
SPHERES	Synchronized Position Hold, Engage, Reorient, Experimental Satellites
SSL	Space Systems Laboratory
SWARM	Self-assembling Wireless Autonomous Reconfigurable Modules
Sys ID	System Identification
UDP	Universal Docking Port
VERTIGO	Visual Estimation for Relative Tracking and Inspection of Generic Objects

Chapter 1

Introduction

1.1 Motivation for On-Orbit Assembly and Servicing

Traditionally, from the beginnings of aerospace engineering, spacecraft have been designed to the limitations of a single launch vehicle. The current state of rocket technology, thus, dictates the types of satellites that can be launched to orbit. These restrictions not only include that the spacecraft must not exceed the size of the launch vehicle's fairing and the lifting capabilities of the launch vehicle's engines, but also that once on orbit, the spacecraft will not be repaired, refueled or augmented with materials sent on additional launches. The idea of the on-orbit assembly and servicing of satellites differs fundamentally from the traditional design of spacecraft in that space systems will no longer be constrained by the use of a single launch vehicle. Specifically, on-orbit assembly and servicing can improve upon three aspects of traditional spacecraft design. First, on-orbit assembly enables the construction of large space structures. Second, space systems can be serviced and upgraded once they are already deployed on orbit (whether that deployment was fully successful or not). Third, the paradigm of monolithic spacecraft can be broken through the use of modular design.

1.1.1 Large Space Structures

Although the sizing of space structures is incredibly mission-specific, there is a common desire across many space communities. Bigger is indeed better (at least for the performance of the system, that is). For the manned-spaceflight community, the implications of a large space system are easily relatable. Humans require large volumes to live comfortably, store consumables (food and water primarily), and house environmental control and life support system equipment. All of these considerations increase the demand on the other spacecraft subsystems as well. In the end, small habitats are generally infeasible for long duration spaceflight. Consider the International Space Station (ISS) shown in Figure 1-1. The ISS is the quintessential example of a large space structure that was assembled on orbit. It took multiple launches over many years to finish the assembly process, but now the ISS spans an area larger than a football field and perpetually houses six astronauts. One can imagine that if the creation of space outposts lies in the future, the habitats would undoubtedly need to be very large.

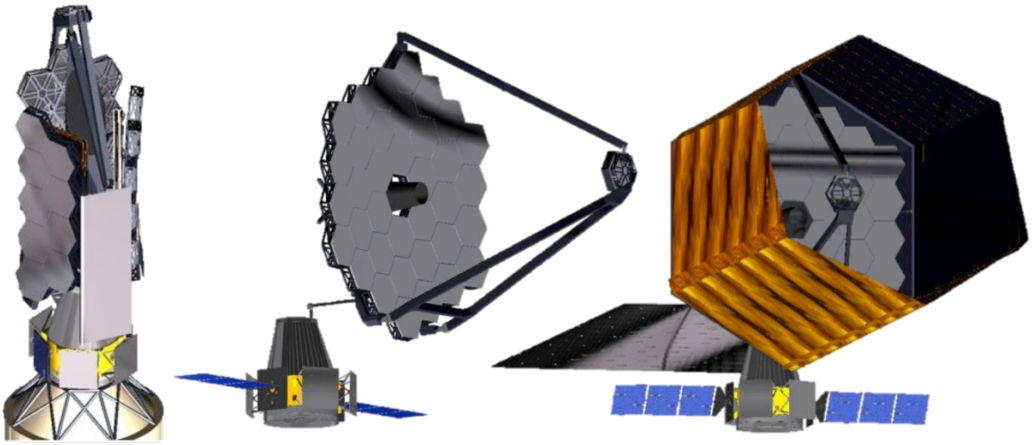
Figure 1-1: The International Space Station [37]

However, it is not just the manned-spaceflight community that would prefer larger

spacecraft. The space telescope and optics communities are particularly interested in larger apertures. As the diameter of lenses and mirrors grows, the angular resolution on the celestial sphere improves according to the following relationship:

$$\theta = 1.22\left(\frac{\lambda}{D}\right)$$

where θ is the angular resolution, λ is the wavelength of light, and D is the diameter of the aperture. With a higher angular resolution on the sky, the telescopes are able to more precisely image celestial bodies at further distances (i.e., further in the past). Currently there is interest is developing larger self-deployed apertures as have been designed for the 6.5m James Web Space Telescope (JWST) [11]. The process of folding this mirror into the launch vehicle, and unfolding everything around it once on orbit involves very high risk. The Advanced Technology Large-Aperture Space Telescope (ATLAST) mission proposes a 16.8m telescope (Figure 1-2) that would require the use of a yet-to-be-developed launch vehicle and still relies on a very risky deployment sequence [30]. Assembling these large, segmented mirrors on orbit would allow launching in multiple launch vehicles and quicker advancement in feasible sizes of mirrors, as the growth is not checked by the slow rate of launch vehicle development.



(a) (b) (c)

Figure 1-2: The ATLAST spacecraft as stowed in launch vehicle (a), as the mirror has been deployed (b) and as the full shroud and heat shield have deployed (c) [30]

Today, the commercial space community primarily consists of communications satellites providing data all across the globe. An important concept in communica-

tions is the gain of an antenna. Satellite manufacturers prefer an antenna that has a very large gain, as it can improve the signal to noise ratio of the system significantly and thus require lower transmission power requirements. The gain of an antenna is proportional to the square of diameter of the antenna as expressed in the following relationship for a circular aperture:

$$G = \left(\frac{\pi D}{\lambda}\right)^2$$

where G in the gain, D is the diameter and lambda is the communications wavelength. Again, currently these antennae must fit in the launch vehicle fairing with their spacecraft bus and could grow larger if given separate launches and assembled on orbit.

Other applications of large space structures include on-orbit electrical power stations that rely on a large area of solar arrays and refueling stations that would require large volumes to store propellant. These applications will likely require assembly on orbit but also hint at the idea of servicing satellites once they are on orbit as to extend their lifetimes.

1.1.2 Spacecraft Servicing

One of the most expensive areas of current satellite design is implementing stringent lifetime and reliability requirements, which dictate a high amount of redundancy and protection to prevent system failures. Even so, once the satellite experiences a critical component failure, its mission abruptly ends and there are generally no efforts to restore its functionality through on-orbit servicing. Aside from component failures, a satellite's lifetime is also limited by the amount of propellant with which it is launched. Once the fuel runs out, the satellite cannot station-keep or point as effectively. This behavior can be dangerous for other satellites, so satellites are generally deorbited or put into a graveyard orbit before they reach this point. On-orbit satellite servicing is the concept that these systems' lifetimes do not need to be limited by these constraints. It may be cheaper and quicker to repair failed satellites or refuel depleted propellant tanks, than to build and launch a replacement.

Spacecraft servicing has been performed in the past very effectively from a performance perspective, but not very cost efficiently. The most important and most widely known servicing missions to date are the five Space Shuttle flights that serviced the Hubble Space Telescope. After each of these missions, the scientific output of Hubble increased significantly as can be seen by Figure 1-3 from Baldessara [1]. These missions both consisted of servicing failures of the telescope and upgrading the components of the telescope. Not only was the lifetime of the telescope increased, but the increased value of the scientific data coming out of Hubble lead to advancements and discoveries that would not have been achievable without the servicing missions.

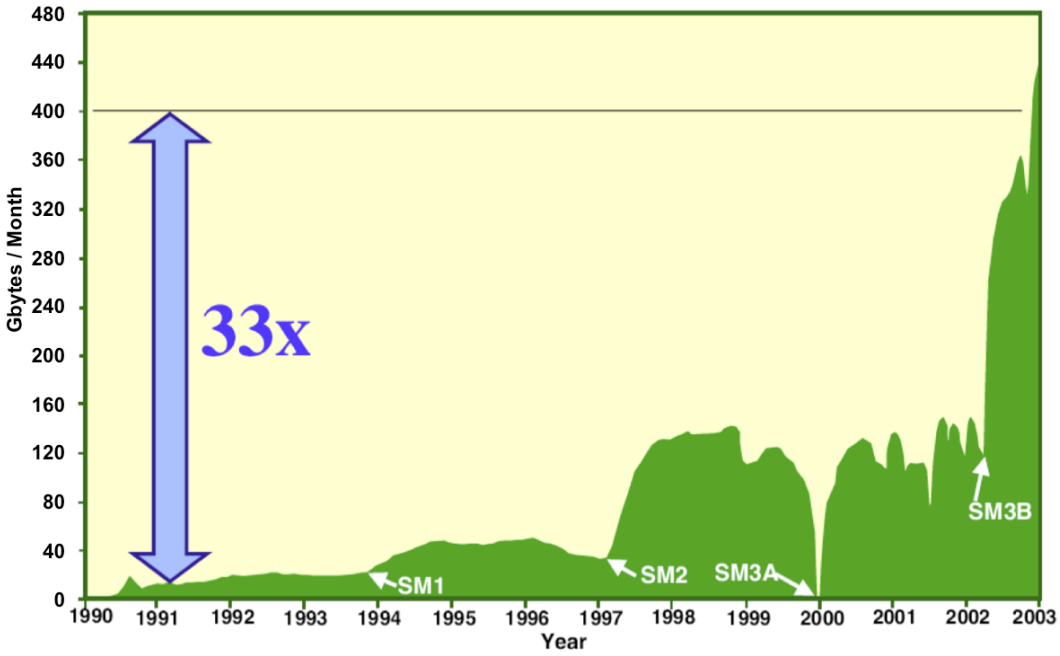


Figure 1-3: Scientific output from the Hubble Space Telescope from 1990 through 2003. Visible improvements in the amount of data collected are seen after each servicing mission (SM) [1]

Recently, there has been a push to avoid the costs of manned servicing missions like those performed on Hubble, by accomplishing these tasks autonomously with robotic spacecraft servicers. The Defense Advanced Research Project Agency (DARPA) Phoenix program aims to do just this through a robotic servicer/tender removing an antenna from a defunct communications satellite and repurposing that antenna in the construction of a new space system [4]. Figure 1-4 shows an artist's

conception of the servicer/tender severing the antenna from the communications satellite. With Phoenix, the goal is to demonstrate the ability to reduce the cost of getting operational satellites on orbit by reusing parts from decommissioned satellites in the geostationary graveyard orbit.

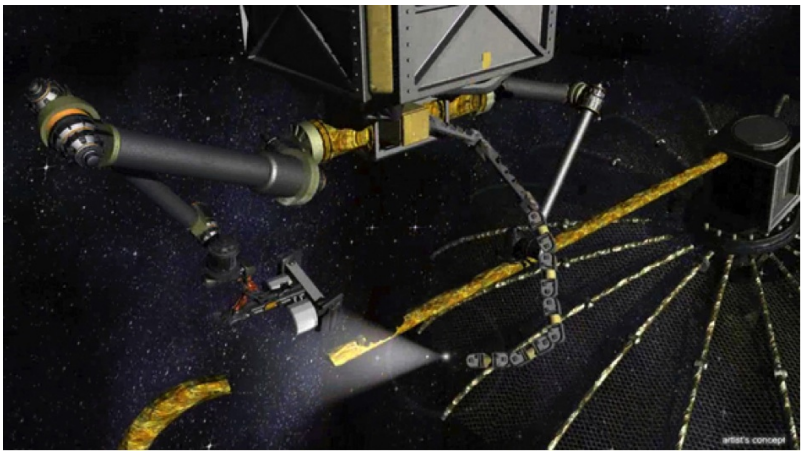


Figure 1-4: Artist's conception of the DARPA Phoenix servicer/tender spacecraft severing the boom of a communications antenna [8]

Economic feasibility studies [3] [38] have been performed that assess the demand of on-orbit servicing, yielding substantial claims that servicing is a viable option to explore in the immediate future. The large number of satellites that are being launched to replace older versions in the same orbital slot suggests that servicing these satellites on-orbit would save time and cost to the service provider and satellite manufacturer. However, no reliable in-depth studies of the overall costs of a robotic servicer spacecraft have been performed, so the economic studies can only be validated against historic satellites that may not accurately represent the tradespace.

Smaller steps are also being made with on-orbit refueling experiments, autonomous rendezvous missions (e.g., Orbital Express), and testbed development of algorithms, including control software and vision-based navigation. Although the full robotic servicing applications, like DARPA Phoenix, may seem out of the current capabilities (or needs) of the traditional satellite provider, they may offer benefits to future, more modular, and universally designed spacecraft.

1.1.3 Modular Spacecraft Design

Modularity in spacecraft design refers to the subdivision of a spacecraft into many modules that each performs a specified function. A basic depiction of a modular spacecraft is shown in Figure 1-5. For example, a spacecraft could be divided into propulsion, attitude control, navigation, computer processing, battery, solar panel, and payload modules, in addition to many other potential modules. Each of these modules would interact through electromechanical docking port interfaces, transmitting data and power and establishing a rigid physical connection between each other. Dividing components into modules can be on the basis of being in the same component family or subassembly. Transcending traditional subsystem bounds is also possible, as it may be optimal for components to be grouped together based on the resources they each use, their thermal properties, or their reliability and mean time between failure. There is likely a significant component, mass and power overhead associated with modularity, as each module will require structural support, thermal management, processing, power, and a docking port. Cellularization, as Barnhart describes, takes modularization another step forward by developing standard interfaces that do not require specific docking orientations or locations to mimic the biological behavior of cells [2]. Under the DARPA Phoenix project, these small modular spacecraft are called *satlets*.

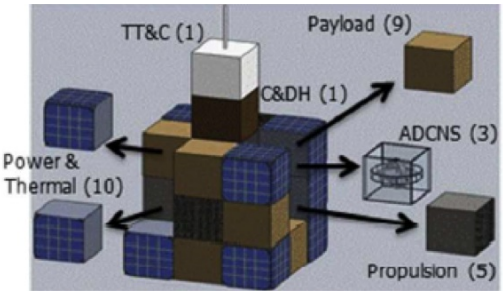


Figure 1-5: Example of a modular spacecraft system [2]

Currently, there are not many obstacles to developing modular spacecraft beyond the perceived additional engineering effort and raw materials required to divide components into modules and manage the additional interfaces between modules. Modularization is not popular, because there is currently no infrastructure on orbit

for modularization to be beneficial. Without robotic servicers and assemblers actively interacting with satellites orbiting the Earth, no satellite provide will see a benefit to modularizing their spacecraft. On-orbit assembly and servicing will, however, greatly increase the value of modularizing the systems. The modules do not have to launch on the same rocket, as they can be collected and assembled once on orbit. This enables the staged deployment of modules using multiple launch vehicles and allows the construction of very large structures once on orbit. This also allows modules to be launched at lower cost on ride-share opportunities if primary payloads have left mass and volume margin in a launch vehicle.

Once the satellite is assembled and is operational, there still remains the primary benefit of modularization: servicing. When spacecraft are design to be divided into separate modules with electromechanical docking interfaces, the hope is that the robotic servicer spacecraft will be able to quickly remove modules that have failed and insert replacement modules. This avoids the technical challenges of the DARPA Phoenix program by eliminating the need to sever antennae booms and other structural components, dock and grapple with unknown targets, and in general, interact with systems that are not designed to be serviced. Potentially the most beneficial module would be a propulsion module, or fuel tank module, that could be simply replaced when the satellite has run low on fuel. In addition, it is expected that the creation of universal modules will benefit from significant reduction in cost due to economies of scale and learning curves.

1.2 Aggregative Spacecraft Systems

An *aggregative spacecraft system* consists of multiple spacecraft modules that exist individually, but can synergistically perform tasks by docking or grappling together and sharing their resources and capabilities. The general idea is that by joining multiple modules, they can perform tasks that none of them could have performed alone. This concept relates to the discussion of modularity in Section 1.1.3. For example, consider an aggregative system that consists of two modules. One module has its

own set of reaction wheels and can control attitude very well to point a telescope aperture at the Earth. The other module has a propellant tank and thrusters and is capable of controlling position and attitude very well for station-keeping purposes and rough pointing for a communications antenna. They were each launched as separate payloads and serve their own purpose, however docking together presents added benefits for both sides. The first module can now station-keep with a propulsion module and lengthen its lifetime by avoiding deorbiting. The second module may now become more fuel-efficient as now it can point using reaction wheels rather than propellant-consuming thrusters.

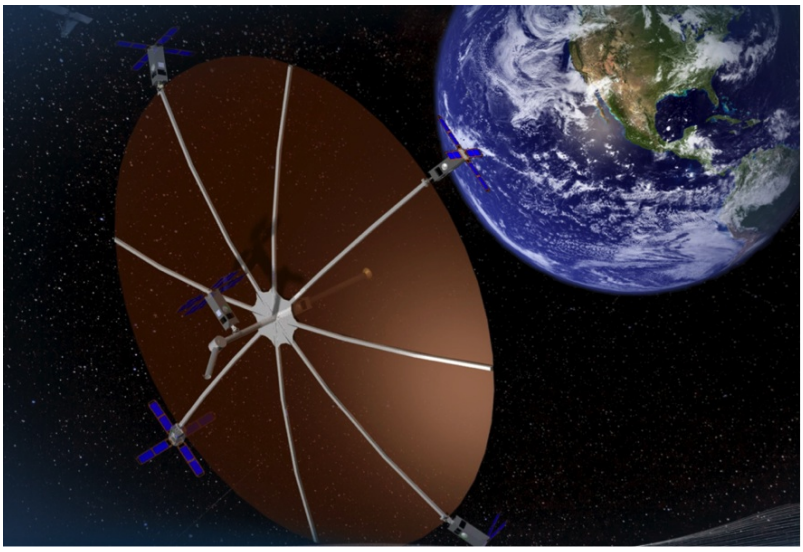


Figure 1-6: The DARPA Phoenix satlets positioned to control and operate a harvested communications antenna [8]

In the scope of this thesis, *aggregative* refers to the ability of the system to aggregate, because it has inherent capabilities suitable for aggregation, while *aggregated* means that the systems has already performed an aggregation maneuver. Aggregated, for example, applies to a set of satellites sharing resources and capabilities only when they have docked, while aggregative would refer to them at all times as it is a classification of the system.

Figure 1-6 shows an example of the DARPA Phoenix program utilizing an aggregative spacecraft system. In this mission, four small spacecraft modules, known as satlets, are positioned around the perimeter of a retired communications antenna.

A fifth is attached to the antenna electrically at the boom in the center. This aggregative system consists of six components, however, as the antenna itself, although not "smart," is still considered an aggregative component. Each of the satlets on the perimeter is a propulsion module responsible for pointing and slewing the antenna to its target on Earth. As they are positioned as far away from the center of mass as possible, the system achieves the maximum actuation capabilities. All of these components are intended to be separate upon launch and only aggregate together once on orbit to extend the life of the antenna from the defunct communications satellite.

Figure 1-7: Proposed project for robotic assembly of a space telescope [25]

Figure 1-7 shows a proposal for using multiple satellites to assemble a telescope on orbit. In this on-orbit assembly process, there are several situations where multiple components dock and undock, and some points when multiple spacecraft are attempting to control a larger structure. When a spacecraft is maneuvering one segment of the telescope mirror (hexagonal pieces in Figure 1-7), it has aggregated mass properties with and has added actuation capabilities to the previously passive mirror segment. When the two spacecraft are docked to the structure, they are cooperatively controlling the system with additional actuation capabilities that greatly exceed what one spacecraft could do alone.

Of course, through the discussion of on-orbit assembly and servicing, most of the capabilities have not been fully developed and tested in relevant environments.

There are a significant number of control challenges present that need to be resolved before on-orbit assembly and servicing can be considered viable. It is not that these problems are extremely difficult to solve, but they require a change in mind frame and expansion of control theory application before true confidence can be gained.

One of the primary control challenges is how to sense and estimate the spacecraft attitude and relative pose once the separate modules have aggregated unique sensors together or have changed the orientation of the sensors with respect to the body. Sensor fusion is a rapidly advancing field and is addressing this concept fairly well, although the problem of reconfiguring estimators to accept new and unknown sensors effectively has not been tackled yet. Another challenge is related to the variation in knowledge of the system as it aggregates. Tolerances stack up in docking interfaces. Jitter can increase due to new vibrational components. Alignment of thrusters and sensors can be varied or blocked based on the spacecraft geometry. All of these things increase the noise in the system and increase the error in the nominal models of the system. This uncertainty needs to be accounted for in the spacecraft's control and estimation systems.

More physically obvious as multiple modules dock and interact with each other, their mass, center of mass and inertia properties have the potential to drastically change. Once a system has aggregated, its mass could more than double, its center of mass could move entirely away from the center of geometry of the individual components, and its principal inertial axes could shift. This change in mass properties affects how the sensing and control algorithms are implemented, as they rely on a dynamic model of the system, which may have been significantly altered. Finally, the actuation capabilities of the system likely will change with the addition of new thrusters, control moment gyroscopes (CMGs), reaction wheels, or other actuation mechanisms.

For this thesis, the research will be scoped to look at how additional actuators and new actuator layouts should be handled when spacecraft have docked. This heavily involves the physical model of the system. As the center of mass changes, the torque that each thruster imposes on the system can change, even completely

switching direction in fairly simple aggregations. Having the knowledge of where the thrusters are located relative to the center of mass is essential. In spacecraft control, a mixer is a piece of software that converts the forces and torques commanded by the control system into applied thruster firing times. The mixer figures out how long to fire specific thrusters to achieve a specific commanded direction and magnitude of force and torque. This mixer is traditionally developed for a specific spacecraft, so has hardcoded values for the thruster locations and forces. In aggregative systems there is a need for a modular mixer that can adapt to include the additional thrusters that have been added to the system as well as account for the movement of the center of mass of the system. This thesis suggests a methodology and algorithm for reconfiguring these mixers on-line, in real time, so the aggregative systems will not have excessive downtime when compared to manually uploading new algorithms and controllers from a ground station. This adds an additional level of autonomy to the system, as there will be full computer control during this switchover to avoid time spent with human intervention.

1.3 Experimental Testbeds

Throughout this research, several experimental testbeds were used to incrementally test this technology in relevant environments. At the core of all of these testbeds are the Synchronized Position Hold, Engage, Reorient, Experimental Satellites (SPHERES), which were designed, built and developed in the MIT Space Systems Laboratory (SSL) during the late 1990s and early 2000s. The Self-assembling Wireless Autonomous Reconfigurable Modules (SWARM) testbed also served as a base for the Agile Reconfigurable Modules with Autonomous Docking for Assembly and Servicing (ARMADAS) program and the MIT SSL's research for the DARPA Phoenix program. The remainder of the hardware and backend software, including the refurbishment and duplication of several pieces of SWARM hardware, was developed and implemented during the course of this thesis research with the help of David Sternberg and Bryan McCarthy.

1.3.1 SPHERES

The SPHERES facility aboard the ISS consists of three small satellites and is used extensively as a risk-reduction platform for enabling new space technologies—specifically those technologies relating to close-proximity operations of multi-spacecraft systems that require long-duration microgravity testing. In Figure 1-8, the three SPHERES satellites are shown operating in microgravity on the ISS. Currently, the Japanese Experiment Module (JEM) or Kibo is home to the SPHERES satellites. The SPHERES testbed also has a ground segment that is made up of several SPHERES satellites spread around the United States. As of the writing of this thesis, there are three satellites at NASA Ames Research Center and two satellites at MIT. This ground segment is used to test out algorithms and hardware before they are sent to the ISS to ensure operational success once on orbit. The MIT Flat Floor Facility is a 16-ft epoxy floor that enables three-degree-of-freedom dynamics through the use of air carriages that float on pressurized carbon dioxide with very low friction.

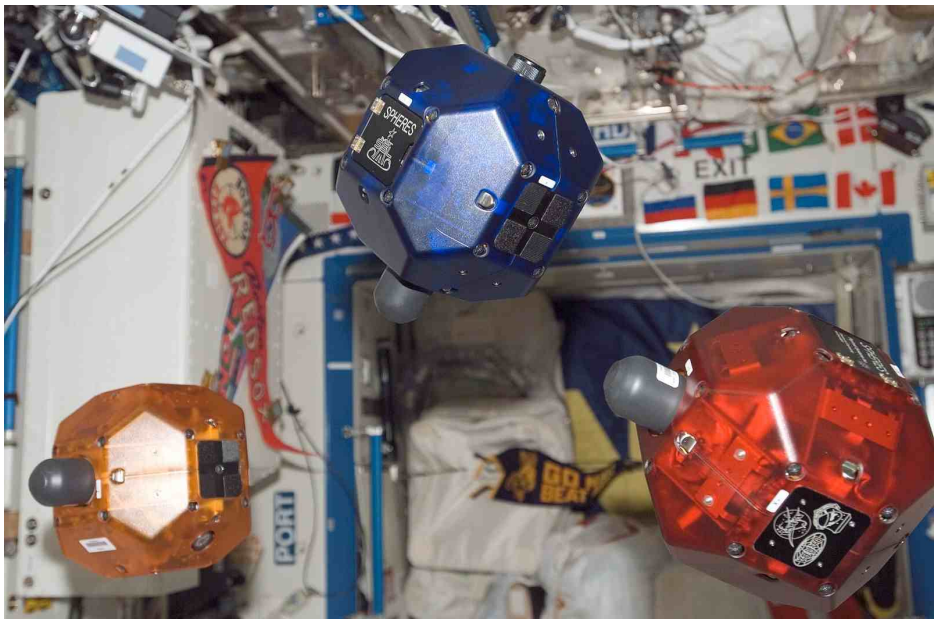


Figure 1-8: Three SPHERES satellites floating in micro-gravity aboard the ISS [27]

The SPHERES satellites use most of the major subsystems typical of any spacecraft: Structures, Communications, Power, Avionics, Attitude Determination and Control (ADC), Guidance Navigation and Control (GNC), Propulsion, and Payload.

The only subsystem missing is a traditional Thermal subsystem, as the SPHERES satellites rely on the room temperature environment inside the ISS and have limited heat rejection mechanisms to protect against hot temperatures. Nonetheless, to remain a captured, safe, reusable and cost-efficient testbed, the SPHERES satellites are not designed to operate in the environment outside the walls of the ISS. To serve as a spacecraft dynamics and control testbed, the most important subsystems are ADC, GNC and propulsion, however the payload becomes very important as well, when additional sensing or actuation capabilities are added to the system. In addition, the satellites need to make it easy for the astronauts aboard the space station to operate and run through experiments efficiently.

The main visible components outside of the 20-centimeter, translucent, plastic shell of the SPHERES satellite are labeled in Figure 1-9. For operability, the control panel on the satellite allows the crew to toggle power to the satellite, enable the satellite for a test, and perform a hard reset on the satellite. The battery doors give the crew quick access to the two sets of 12V batteries on either side of the satellite. These batteries can power the satellite for approximately two hours of testing. The SPHERES satellites have 12 cold-gas thrusters that can actuate the satellite in a full six degrees of freedom. The thrusters' propellant, CO_2, is stored in a propellant tank that is screwed into the bottom of the satellite by the crew. The crew is able to use a knob on the top of the satellite to regulate the pressure of the gas. The satellite uses infrared (IR) pulses to synch time between itself and other satellites, as well as with the ultrasound beacons that surround the test volume. To determine their position and attitude relative to the global frame, the SPHERES satellites use time-of-flight measurements of the ultrasound pulses emitted from the five SPHERES beacons positioned in the overhead and deck planes of the ISS JEM module. Aside from this global metrology scheme, the satellites have inertial measurement units (IMU) with gyroscopes and accelerometers to measure angular velocity and linear acceleration. With all of these sensors, the SPHERES satellites are able to determine their full state of position, velocity, attitude and angular velocity. A Texas Instruments C6701 Digital Signal Processor runs the flight software, which handles all of the user pro-

grammable controllers, estimators, and tasks, in addition to the background tasks such as housekeeping, communications, and scheduling. The flight software is written in C and can be bootloaded to the satellite over an 868 MHz radio. This radio also allows the satellites to communicate with each other and with the SPHERES Graphical User Interface (GUI) on the flight laptop. The ISS crew interfaces with the satellites through the SPHERES GUI, where they are, among other things, able to bootload programs, switch between experiments, start and stop tests, and monitor consumables. Because the testbed is meant to be extensible, the satellites each have an electro-mechanical expansion port (not pictured) to which more advanced or mission-specific payloads can be attached.

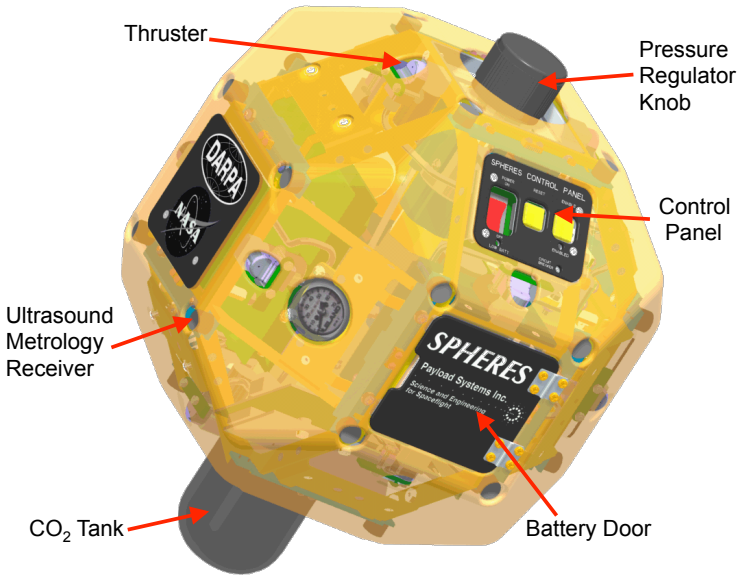


Figure 1-9: A labeled SPHERES satellite

The SPHERES expansion port was recently upgraded to the SPHERES Expansion Port Version 2 (ExpV2). This new interface allows rigid mechanical attachment to the satellite in addition to an upgraded electrical connector, now including two communications channels and several input/output pins. This new expansion port has allowed several payloads to be used with the SPHERES facility aboard the ISS. The first payload to use this was a set of stereo cameras that support vision based navigation and estimation. Additionally, the ExpV2 has enable the testing of a large electromagnetic formation flight device consisting of 0.75 meter diameter coils

that can actuate the satellite using electromagnetic forces. The satellites have also served as a base for microgravity fluid slosh experiments and remote operations via smartphone, and will soon host docking ports. Over the last couple of years, the SPHERES ExpV2 has enabled all of these experiments.

1.3.2 ARMADAS

The ARMADAS testbed was developed in the fall of 2012 and implemented and used in research during the spring of 2013. The testbed serves to enable research in Resource Aggregated Reconfigurable Control (RARC) and Model-Based Planning and Execution (MBPE) for on-orbit assembly and servicing missions. The Concept of Operations (CONOPS) for this project required two mobile satellites equipped with docking ports and obstacles for the individual satellites to navigate around before docking. The SPHERES satellites, the air carriages, and the MIT Flat Floor Facility are used as described in Section 1.3.1 and, along with the SWARM equipment, constitute the majority of the test equipment. This hardware is not intended to launch to the ISS and is only used in the MIT facilities here on Earth.

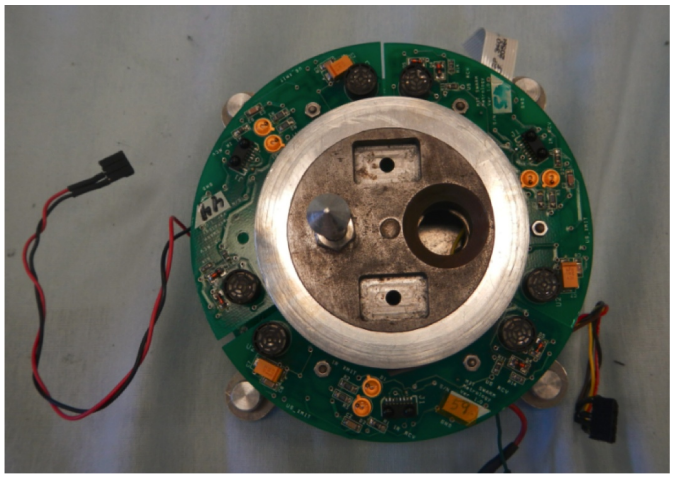


Figure 1-10: The Universal Docking Port ground prototype hardware

The SWARM equipment, shown in Figure 1-11, augments the SPHERES satellites with a docking port, 16 additional thrusters and several additional hardware mounting locations. This hardware was designed for the SWARM project in the early 2000s and also used extensively for flexible-beam docking research [16]. The docking port used

in this testbed is the Universal Docking Port (UDP), a precursor to the SPHERES Docking Port (described briefly in Appendix A) that is launching to the ISS in October 2014 as of the writing of this thesis. The UDP, shown in Figure 1-10, is androgynous, meaning it is able to dock to copies of itself. Each UDP has a lance and a hole, such that when an opposite UDP's lance is inserted into the hole, it trips a photosensor and signals to the SPHERES satellite that capture can begin. The SPHERES satellites can then send a command to close the cams on each of the docking ports, so that the satellites are rigidly connected to each other [32]. The SWARM propulsion carriage, the gold structure shown in Figure 1-11, sits between the SPHERES satellite and the air carriage. An avionics board on the propulsion carriage is connected to the SPHERES expansion port, and controls the thruster firings and passes commands between the SPHERES satellite and the UDP. Nominally four short CO_2 tanks (those used on the ISS) are screwed into the four-port regulator on the carriage and are used as propellant for the thrusters. Optionally, for longer test durations, two long CO_2 tanks (those used on the ground and for air carriages) can be substituted in place of the outside two CO_2 tanks.

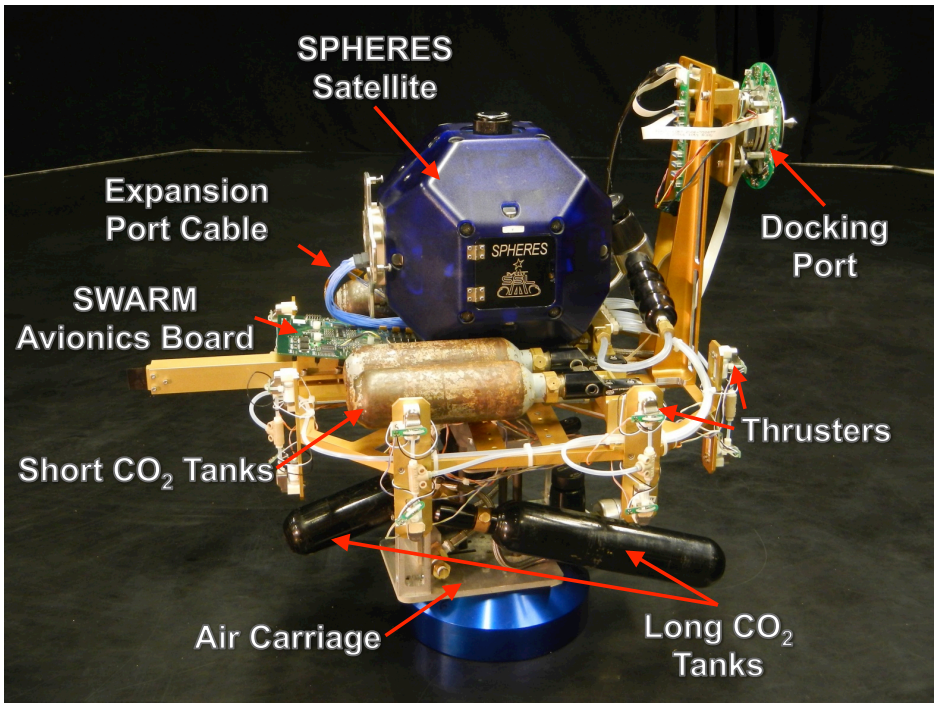


Figure 1-11: The SWARM ground hardware

One full SWARM propulsion carriage was built prior to the fall of 2012 and was not completely functional. For the ARMADAS project, an additional SWARM propulsion carriage was required and the avionics board now needed to talk to the SPHERES ExpV2. This hardware fabrication was performed over the fall of 2012 and was completed in January 2013. In addition, because of ExpV2, the communications software needed to be updated.

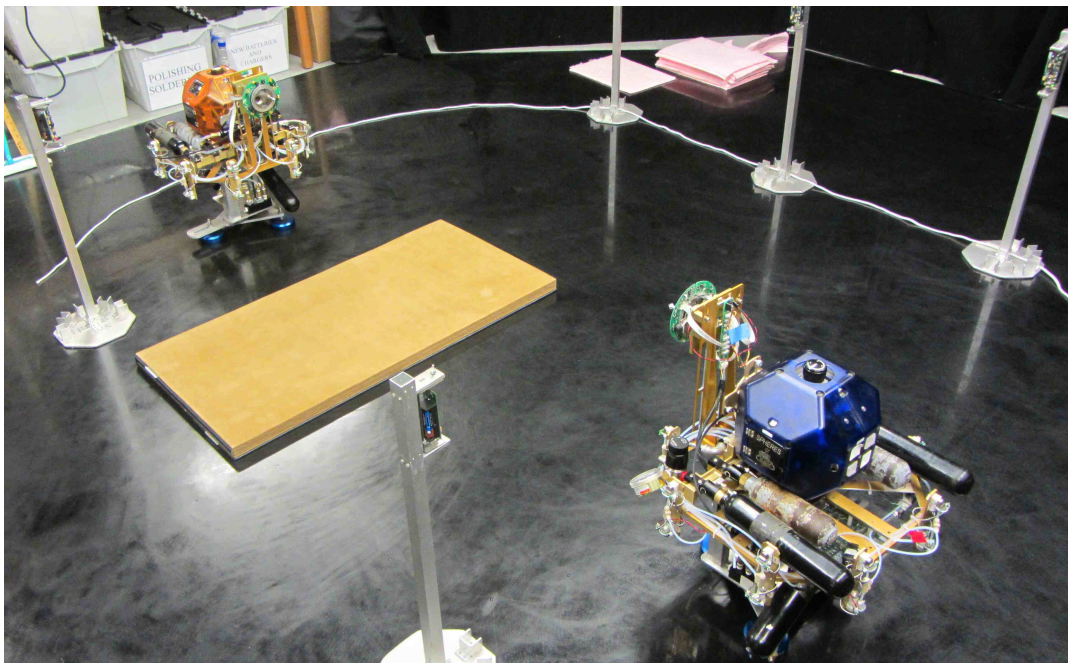


Figure 1-12: The ARMADAS test set-up on the MIT Flat Floor Facility

Two full SWARM assemblies (Figure 1-11) are used in the ARMADAS testbed. A typical setup is depicted in Figure 1-12, where two satellites start on the opposite sides of the MIT Flat Floor Facility and are separated by an obstacle. Further CONOPS are described in Section 5.2.1. The SPHERES metrology beacons are pulled in from their nominal Flat Floor configuration as the docking port structure seems to block the US and IR transmissions and reduce the effective "covered" area of the Flat Floor. This requires an additional step to set up the correct beacon configuration in the SPHERES GUI. To ease our efforts, the beacons were placed in locations that could be specified by seat-track indices that correspond to the ISS JEM module. The obstacle used in this testing is the glass table cover. This obstacle was chosen because it is thick enough that the air carriages rebound when hitting it, it does not scratch

the epoxy floor as it is design to protect sensitive surfaces, and it is light and easily maneuverable.

Operationally, several tips were helpful in creating, maintaining and using this testbed. The test area is roped off with a white piece of rope, because the epoxy floor collects dust and particles very easily. By roping the test area off, the section of the epoxy floor can be kept clean during testing. Cleaning the epoxy floor is time consuming and tedious as even the smallest scratch or sticky spot needs to be meticulously buffed out for the air carriages to glide smoothly. It is suggested that one replenish all consumables, sweep the test area and clean the air pucks before running any test on the ARMADAS testbed. Since consumables run out at different rates, not switching them out every test will result in consistent test failure due to the air carriages or satellites running low on propellant and power. This problem is exacerbated by the extremely long test durations that experiments using this testbed require. The best results always come from fresh consumables.

1.3.3 Phoenix

The Phoenix testbed aimed to assist in performing preliminary testing and risk-reduction for the DARPA Phoenix project and was supported by Aurora Flight Sciences and the NASA Jet Propulsion Laboratory during the spring and early summer of 2013. The main goal of the testbed was to be able to adjust the center of mass and inertia properties of a set of docked satellites with a payload and test relevant control algorithms. This testbed enables the testing of the same controller in multiple system configurations, as is traceable to the DARPA Phoenix project's on-orbit demonstration in which multiple satellites are positioned to control the large antenna of a communications satellite. The CONOPS will be discussed further in Section 5.1.1. The Phoenix testbed consists of two SPHERES satellites each sitting on SWARM propulsion modules and SPHERES air carriages. These satellites are rigidly attached to a proof mass through two adjustable-length aluminum beams. The proof mass consists of an unpopulated SWARM carriage with an adjustable amount of mass sitting on this carriage. This proof mass's position can be changed in two dimensions by

sliding the beam through the T-joint connector and tightening with a hex key. The system uses the same CO_2 air carriages described in Section 1.3.1 to float on the MIT Flat Floor Facility's 5-meter, octagonal epoxy floor as pictured in Figure 1-13. This testbed has extensibility as it can be easily used to support research other than that for the DARPA Phoenix program. The testbed is modular in that it can be connected in multiple configurations and varying length aluminum-beams can be substituted. All connections use the standard hole pattern on the SWARM carriages, so additional carriages or test equipment can be attached using the same heritage design.

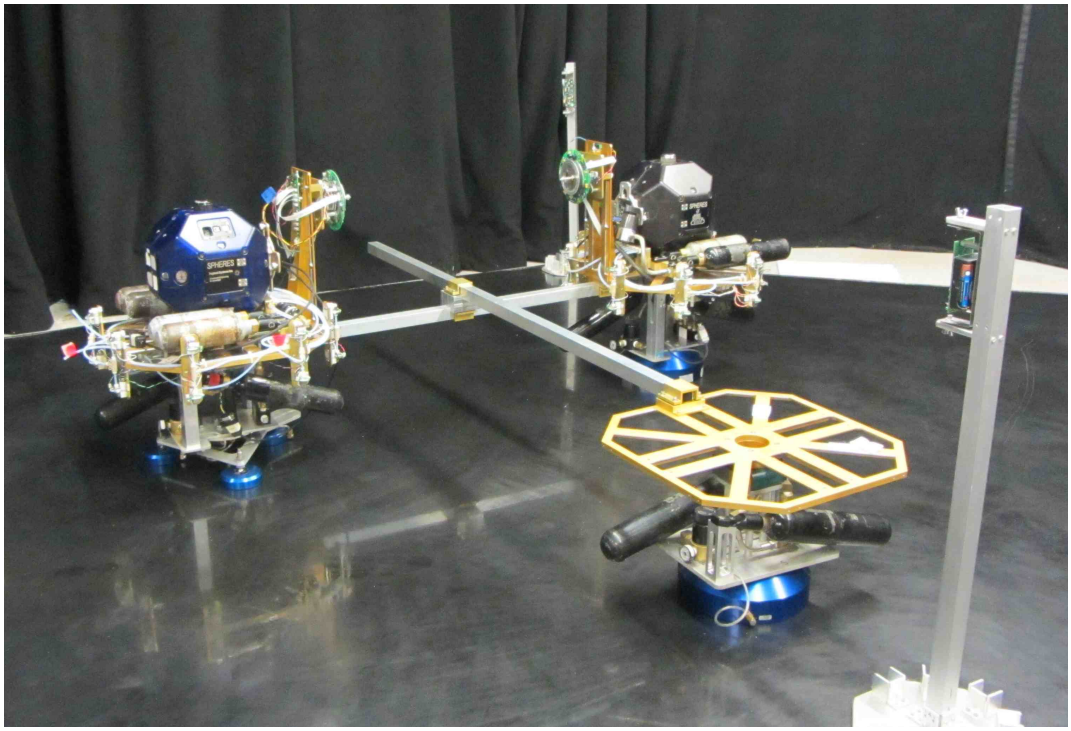


Figure 1-13: The Phoenix test set-up on the MIT Flat Floor Facility

1.4 Research Objective

The overall objective of the research forming the basis of this thesis is to create and test the effectiveness of a reconfigurable thruster selection algorithm for aggregative spacecraft systems under four different actuation modes: nominal, precision, agility and fuel-efficiency. These algorithms will be needed for the cooperative control of multi-spacecraft systems that have docked together and aggregated sensors, actuators

and mass properties, a prevalent topic in the field of on-orbit assembly and servicing of spacecraft. The formal definition of the research objective of this thesis is as follows:

- **To** generate commanded forces and torques in an aggregative, thruster-controlled spacecraft system
- **By** reconfiguring thruster-selection authority matrices (or mixers) in real time
- **Using** algorithms tailored for agile, precise and fuel-efficient actuation modes
- **While** preserving force and torque direction in the presence of saturation

The first statement refers to jointly controlling the thrusters of multiple docked satellites; the second to an on-line reconfigurable mixer; the third to the actuation modes mentioned in the paragraph above and their hybrids; and the fourth to ensuring that saturation is accounted for when multiple thrusters have been commanded.

1.5 Thesis Approach

This thesis consists of six chapters. Chapter 1 hopefully motivated on-orbit assembly and servicing and aggregative spacecraft systems and formulated some background on the research problem of this thesis. Chapter 2 discusses the current literature in the scope of reconfigurable control and thruster selection algorithms and places this thesis amongst the state of the art. Chapter 3 describes the methodology, theory and implementation behind reconfigurable thruster selection algorithms in the context of aggregative systems. Chapter 4 details a simulation created for testing these algorithms and reports results achieved from this simulation. Chapter 5 explains the hardware testing and experimentation used to prove the functionality of these reconfigurable mixers. Finally, Chapter 6 summarizes the thesis and discusses areas of research that should be explored in the future as follow-ons to this work.

Chapter 2

Literature Review

2.1 Aggregative Control Algorithms

In the scope of this thesis, *aggregative control algorithms* include all types of control algorithms that are capable of controlling aggregative systems as defined in Section 1.2. Aggregative control algorithms must be able to maintain control through multiple system reconfigurations, including the docking and undocking of modules with one another. This means that the controller must be either able to adapt to or be robust to these system reconfigurations. The mass properties (mass, center of mass, and inertia) may change by orders of magnitude upon reconfiguration, which in turn drastically modifies the physical plant and system dynamics. Typically, controllers cannot be robust to such large model error, so generally, adaptive techniques or other techniques that update the system model must be implemented.

Specifically, in the context of autonomous on-orbit assembly, an assembler or tug spacecraft will likely be collecting modules and transporting them to an assembly location. While the assembler spacecraft is docking to a set of modules, its controller will need to account for the updated mass properties of the system as it maneuvers and before it performs a burn to the assembly location. Upon reaching the assembly location, more precise movements are required to maneuver the modules into position and dock them with each other without damaging their components.

In the context of on-orbit servicing, maneuvers would be similar to those per-

formed by the assembler spacecraft upon reaching the assembly location. The difference between the servicing and assembly cases is that in the servicing case, there is greater potential for errors in knowledge. With on-orbit assembly, there is exact knowledge of module mass properties and dimensions as they have just been manufactured and launched. Therefore, these properties can be more easily built into the controller in the assembly scenario. For the on-orbit servicing scenario, the customer satellite may have been on orbit for 10 to 15 years before being serviced. Thus there is much less knowledge about the current state of the target spacecraft. Uncertainty is much greater in servicing operations. The presence of uncertainty means that the controller in the servicer must be much more robust and cannot rely solely on a catalogue of known facts about the target satellites. Consequently, the assembly and servicing scenarios will likely favor different control algorithms. However, there may be controllers that can achieve high performance in both situations.

In addition, while investigating the suitability of control systems for aggregative spacecraft systems, specific attributes are desired for both assembly and servicing scenarios. Consider the response time of the controllers and the time required for reconfiguration. The constraint on the reconfiguration time metric is not as tight as it would be for fighter jets, where a millisecond could make a huge difference. However, there is a desire for the reconfiguration to be completed quickly, as minimizing satellite downtime is important. The response time of the controller itself to command inputs can be fairly slow, because fuel-efficiency is vital with on-orbit, propellant-constrained operations and fuel-optimal, bang-off-bang controllers are typically slower. However, under autonomous control, this response time metric could potentially be improved greatly over traditional values. In spacecraft, there are also heavy constraints on the processing time and storage space available to the ADC software. In this sense, there is a desire to reduce the processing and data storage required in these control algorithms. With these metrics, we can narrow some options out, but it is still worthwhile to investigate a large tradespace of controllers.

There are several control methods that have been developed extensively and can be applied to aggregative spacecraft systems. A fair amount of these control algorithms

have come from Fault Detection Isolation and Recovery (FDIR) research. In addition, gain scheduling is an early technique that has been employed extensively in the past. Model switching, online and offline model calculation, and direct adaptive control are also methods that can be implemented with aggregative systems. There still remains a large amount of work to directly apply these control methods to on-orbit assembly and servicing missions, although the majority of the development of the theory behind these control systems has already been completed.

When discussing FDIR research in relation to aggregative spacecraft systems, the algorithms for the recovery portion are most applicable as they typically involve reconfiguring controllers and estimators to perform well with the new configuration of the system. These recovery algorithms usually deal with losing actuators. A fair amount of the work has been done in the aeronautics field as methods to deal with failures of control surfaces or jet engines. This loss of actuators is a problem that needs to be solved in aggregative spacecraft systems, but more important is the adaption of the controller to the addition of actuators. The control algorithms to account for this addition can be inspired by the work already done in FDIR though.

Maybeck and Stevens [21] worked on aircraft flight control systems using a technique known as multiple model adaptive control. This technique uses predesigned Kalman filters and a Command Generator Tracker (feed-forward) with proportional integral feedback controllers for anticipated failure states. Models for all possible configurations are continuously processed and the estimator dictates which model to use in the controller. This results in extremely small transition periods between operational modes—something important for aircraft, but no so for on-orbit assembly and servicing. Boskovic [6] also worked on a model switching, FDIR method dealing with failures due to damage in fighter aircraft. The method uses a variable to track the percentage of damage experienced by the system and switches the controller based on this estimation. This Adaptive Reconfigurable Control scheme only considers a finite number of known failures, so the performance is bounded. Dhayagude [10], who also focused on recovery from aircraft failures, developed a reconfigurable controller that uses linear model following to force the control system output to track a reference

model that exhibits the desired characteristics.

Davidson [9] implemented a method in aircraft control that can probably most easily be translated to the spacecraft regime as it uses multiple actuators that are commanded through a control mixer. Davidson employed frequency-apportioned control allocation and a weighted pseudoinverse to control actuators operating at specific control rate saturations. To account for actuator loss, the weighting for that actuator in the pseudoinverse matrix is set to zero. This is approaching a method of thruster selection logic as described in Section 2.3. Taking another step toward aggregative spacecraft systems, Pong [29] looked at recovery from thruster failure for underactuated spacecraft in both translation and rotation. Pong utilized Model Predictive Control, which relies on an optimizer wrapped around a dynamics model to attempt to minimize a chosen cost. If this work dealt more with overactuated systems and adapting to additions of thrusters rather than removal, it would be perfect for aggregative spacecraft systems. However, optimization in the underactuated regime is fundamentally different than the overactuated regime on-orbit assembly and servicing requires.

Moving away from FDIR, there have been many space missions and experiments flown using gain scheduling as a means to reconfigure a controller. This method essentially preloads a look-up table of gains that can be substituted in a controller based on the current configuration. Normally, gain scheduling is used when there is a very good idea of what configurations the system can take. Gain scheduling also relies on the same controller (with different gains) being able to maintain controllability of the system. Parlos [28] used gain scheduling for attitude control of the Space Station Freedom, a precursor to the ISS, under mass property changes due to Shuttle dockings. Parlos linearized the system about several torque equilibriums to compute linear quadratic regulator (LQR) gains for each configuration prior to launch. All of these gains were stored in a look-up table for easy access once on orbit and a reconfiguration has taken place. The problem with gain scheduling methods unfortunately is that they require prior knowledge of all possible configurations and not only is that infeasible, but can take quite a lot of data storage space.

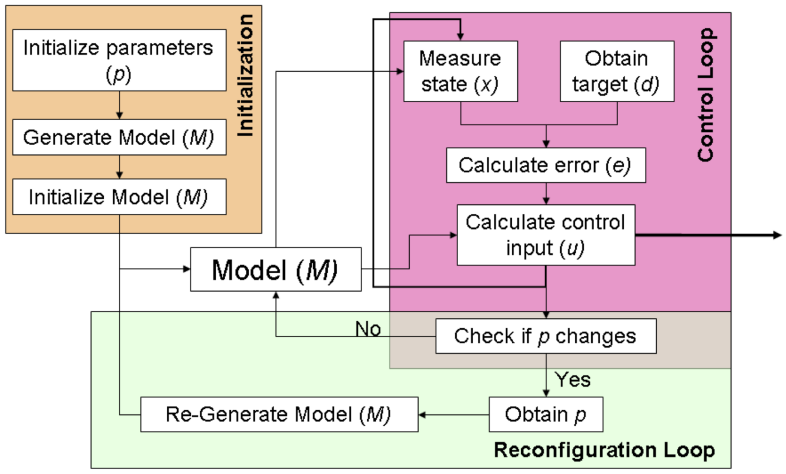


Figure 2-1: Online Model Calculation process [26]

A method that counters the excessive storage space requirement of gain scheduling and multiple model switching is the online model calculation technique developed by Mohan [26]. This method was specifically designed with on-orbit assembly of satellites in mind. In this method, new dynamics models are computed in real time when modules dock together. When the modules dock, they transfer data detailing the physical properties of themselves to each other. Using this data, the reconfiguration algorithm calculates a new dynamic model of the system and new gains through Resource Aggregated Reconfigurable Control (RARC). The RARC method is discussed further in the theses of Sternberg [36] and McCarthy [24] and in the related conference paper [15]. With this method, there is no onboard storage and the only prior knowledge required is for the individual modules to know their own properties. The method is modular and can adapt to any number of additions or removals, as it keeps track of its current state and aggregates or disaggregates as appropriate. This algorithm, as depicted in Figure 2-1, has been demonstrated on the SPHERES testbed both on orbit and on the ground [26][15]. A slightly different approach, essentially offline model calculation, does not require the model to be calculated online in real time. LeMaster [18] worked on a Centralized Connectivity Manager that has an internal model of all satellite modules and updates based on the configuration status of the modules. The individual spacecraft modules get updates from the centralized man-

ager and are able to adapt to the changes. However, in this scenario there must be constant communication between the spacecraft modules and the centralized manager that may be instantiated as a ground station. No actuator or sensor reconfiguration has been developed for this control architecture however.

Direct adaptive control is yet another method that shows promise in the field of spacecraft assembly and servicing. Ulrich [39] applies direct adaptive control techniques to a spacecraft navigating in close proximity to one another. Ulrich uses a model reference adaptive control technique under the presence of uncertainty in spacecraft mass properties and external forces. This controller can be used directly and possibly paired with Mohan's online model calculation method to handle a lot of the uncertainty concerns with the servicing scenario.

Mohan also completed a fairly exhaustive examination of different control methods to be used in on-orbit assembly, as is detailed in [26]. The control methods are compared against required assembly time, fuel-efficiency, and average tracking error. Through the literature survey, it is apparent that there has been substantial development of control methodologies in support of aggregative spacecraft systems, and that there need not be completely new algorithms developed.

The adaptive controller from Ulrich and the online model calculation framework from Mohan (implemented as RARC) provide the basis for the controllers used in this thesis work. This combination suits the control problem posed as there is little processing and data storage required, the reconfiguration can happen quickly, and uncertainty is accounted for with the adaptive control. The implementation of this controller, however, is described in [15], [24] and [36], and is not contained in the scope of this thesis work.

2.2 Determining System Properties Upon Reconfiguration

During the implementation of an aggregative control algorithm described in Section 2.1, there arises a need to determine the current properties, or state, of the system. The controller and estimator need to know the physical properties of the system as far as the location, direction, number, health, and type of actuators and sensors present. These system properties will, by definition, change upon every reconfiguration. Therefore, defining a systematic approach to updating this data is essential for the control algorithm's effectiveness. In literature there has been a number of methods implemented. These methods range from not performing an update at all to look-up table approaches to full system identification maneuvers. The range of approaches is shown in Figure 2-2.

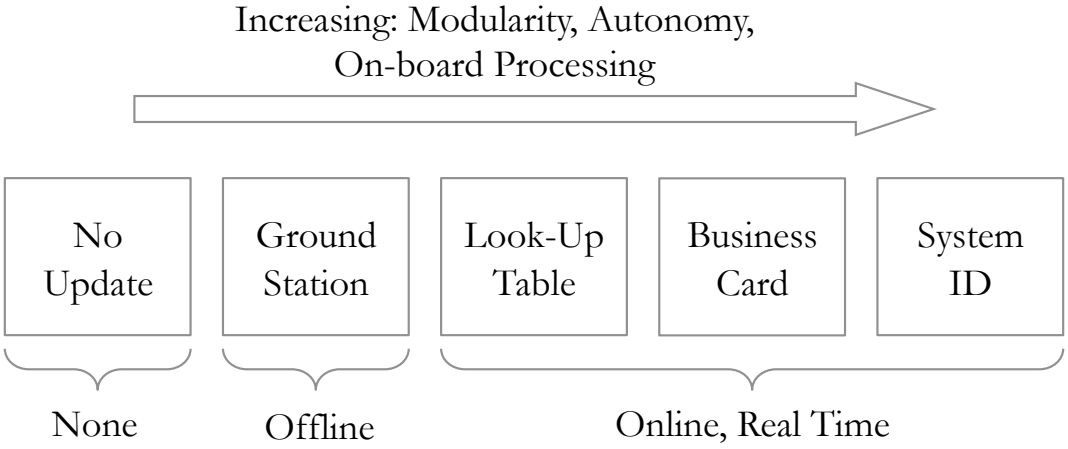


Figure 2-2: Methods of system property update

The first, or actually the zeroth, option to consider is whether or not the system model needs to be updated at all. If adaptive control techniques can be used that do not rely on a high confidence model of the system, there is potential that upon reconfiguration, the system properties would not need to be updated. This assumes that the aggregative controller and estimator is able to operate even when the system properties are possibly orders of magnitude different than what was initially built in to the controller. Generally, the further the true state of the system is from the

modeled state, the longer it will take the adaptive controller to respond to commanded inputs. This is not great for on-orbit assembly and servicing, as quick reconfiguration is preferred. Regardless, it stands as an option and removes the need to update system properties (mass and inertia primarily). The main problem experienced is that there is a high chance that the center of mass will change and cause actuators to produce completely opposite effects on the system than what is initially expected. This is an area that needs to be looked at in further detail, but for the purpose of this thesis, the capabilities of adaptive controllers were considered too weak for the amount of reconfiguration we are considering.

The next step is to determine the system properties offline and transmit the data to the spacecraft using the ground station communications link. This is similar to the method that LeMaster uses with the connectivity manager [18]. The computational burden is lifted from the spacecraft in this case, because computing the new system configuration, determining the new mass properties, and updating the actuator and sensor layouts is completed offline at the ground station. This method adds an element of human intervention and could potentially slow down the process, because the spacecraft would need to wait for an update from the ground station before continuing. If true autonomous assembly and servicing operations are desired, this method could not be used. However, it is a good starting point and has been used extensively in past systems.

Another advancement in terms of autonomy is using a pre-populated look-up table similar to the gain scheduling approach used by Parlos [28]. In this method, a finite set of configuration properties is calculated before launch. The properties are stored in a table on board the spacecraft. The spacecraft is responsible for determining which configuration it is in, and search the look-up table for appropriate system properties. This is a fairly quick method and saves quite a bit of time over the offline method. Unfortunately, the method is limited to the data storage space on the spacecraft to determine the amount of configuration data it can store. Thus, the method is not very modular or extensible to new configurations. This method also pushes the entire burden of dealing with uncertainty to the controller, as the pre-calculated values could

have estimation error based on imprecise knowledge upon table population.

The business card method proposed by Mohan [26] improves upon table look-up, because it adds a high degree of modularity to the system. The method is described in Figure 2-3. In the business card method, each module has knowledge of its own system properties and stores them in a software structure known as a business card. When modules dock with each other, they swap business cards. Now each module knows the properties of the other and can compute the new properties of the system and store them in a new, aggregated business card. The method is modular, in that, after aggregation, the exact process can be repeated since the aggregated system has its own business card. Similar to the look-up table approach, this method also pushes the burden of handling uncertainty to the controller.

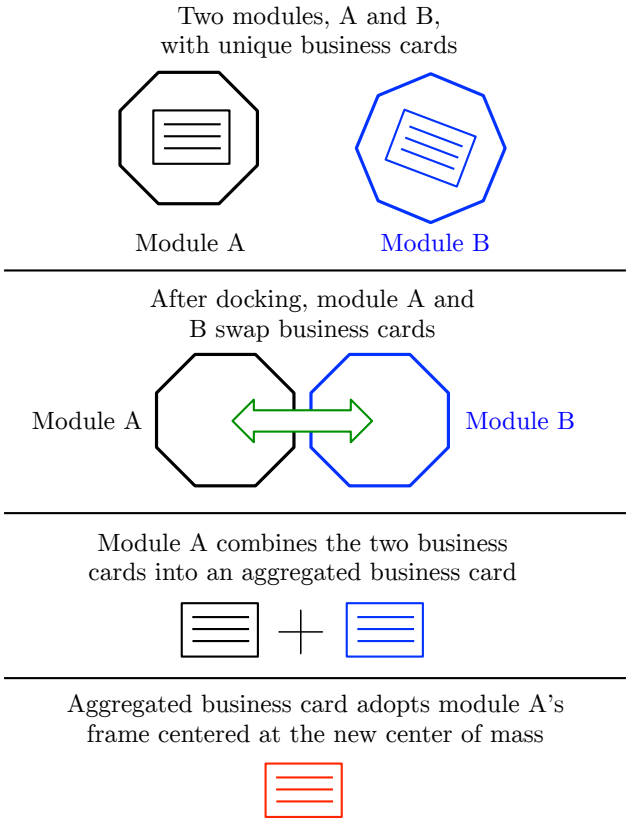


Figure 2-3: Process of transferring business cards

The most advanced method of updating the system properties would be to perform System Identification (Sys ID) after every reconfiguration. Sys ID is the process of

using the on-board actuators to disturb the system and measure the response of the system with the on-board sensors. A map is constructed that directly translates disturbances to system responses. Thus, with this method there is no need to transfer model data between modules or update the system property values directly. This could be used directly in the controller instead of performing online calculations to reconfigure the controller. Other Sys ID techniques are capable of identifying the mass properties of the system. Wilson [42] worked on performing mass-property identification for thruster-controlled spacecraft. With methods similar to those in Wilson's work, the system property values could be updated directly and Mohan's online model calculation methodology could be implemented. Sys ID techniques also have the ability to remove uncertainty from the system, as the system is now being characterized as it reconfigures and does not rely on calculations and measurements performed before launch. Sys ID represents the state-of-the-art in determining system properties autonomously and can be extremely beneficial to on-orbit assembly and servicing missions and operations.

Although the Sys ID process seems best suited to aggregative spacecraft systems, the work in this thesis assumes that the business card method developed by Mohan is being used, as the timeframe that the work was completed did not allow for the extensive development and time-consuming implementation associated with Sys ID.

2.3 Thruster Selection Logic

Throughout the discussion in Sections 2.1 and 2.2, several techniques were discussed on how to handle an aggregated spacecraft system from a control system perspective. Keeping this discussion in mind, there still remains a need for the software to know which thrusters to fire given a certain control input. There needs to be this software-hardware interaction not only within modules but also throughout the entire aggregated system. A global directive is needed to control specific thruster firing times. Depending on the literature source, this piece of software is known by a few names: a *mixer*, *thruster selection logic*, and *jet selection logic*. All of these terms are

referring to the same entity and will be used interchangeably throughout this thesis.

Thruster selection logic, or the mixer, is responsible for translating commanded forces and torques from the controller into required, individual thruster forces or firing times. The typical ADC/GNC architecture for a spacecraft is shown in Figure 2-4. There is an overall guidance and maneuver logic that sends desired positions, velocities, attitudes and angular rates to the controller. The controller then tries to achieve this desired state through the generation of forces and torques on the system. The controller sends these commanded forces and torques to the mixer, which in turn computes the firing times for each individual thruster. The mixer can update the navigation module by sending the executed forces and torques. The navigation module estimates the actual state of the system and sends the systems estimated position, velocity, attitude and angular rate to the controller for feedback purposes.

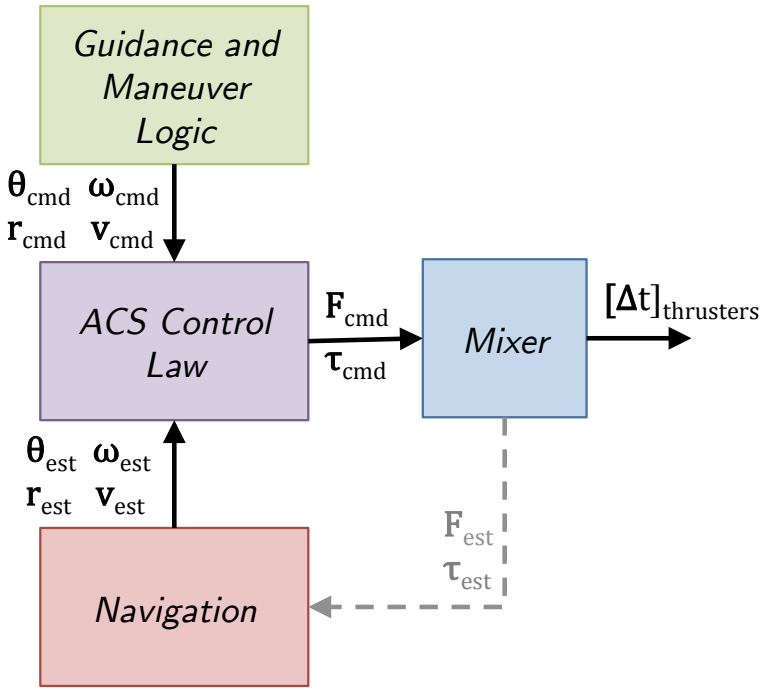


Figure 2-4: Notional ADC/GNC architecture [34]

The concept of a mixer and the first use of the words "jet selection logic" come from the Apollo program [20][40]. The implementation in the Apollo program was minimal however when compared with the state-of-the-art today. In the Apollo program, it

was truly a logic-based selection of attitude control jets based on whether they would issue a positive or negative torque on the system. Several new thruster selection logic techniques have been developed since this time: catalogue tables, pseudoinverse methods, candidate optimal groupings and delta velocity tracking.

Catalogue tables are one of the older methods for selecting thrusters and are based on a look-up table of thruster directions, forces and locations. Choosing a particular set of thrusters from this table allows the algorithm to compute the required impulse from each of the selected thrusters. In recent years, there has been development to have catalogue tables that store optimal thruster configurations for many maneuver types. Martel [19] applied this method to the ADC subsystem of the Automated Transfer Vehicle spacecraft. This method requires a substantial amount of data storage space when compared to the other methods, although it seems to reduce the computational processing required.

The pseudoinverse method relies on the concept of an authority matrix. With an authority matrix, the mixer can directly matrix-multiply the commanded forces and torques to obtain the required thruster firing times. The mechanics of this method will be described in Chapter 3. This makes the computation speed much faster as it essentially only relies on a single matrix multiplication. This concept has been in literature for a while. Originally it was only used with positive/negative thruster pairings as described in Wie [41], but Ratan [31] developed a method that does not rely on this architecture. Shankar [33] extended the concept by adding a weighted pseudoinverse, which can zero out failed thrusters or modify the force provided by individual thrusters. Closely related to the work in this thesis, Hilstad [12] developed the first pseudoinverse authority matrix for the SPHERES testbed as shown in Figure 2-5.

As detailed in Wie [41], the Candidate Optimal Groups (GOGs) approach utilizes an authority matrix as well, although it involves a fair amount of real-time optimization. A linear programming problem is formulated such that the solution will minimize a particular cost [5]. In the literature, this cost is typically fuel consumption. Based on the COGs approach, the feasible set of thrusters in use can be reduced

$$M_{(f)} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \end{bmatrix}$$

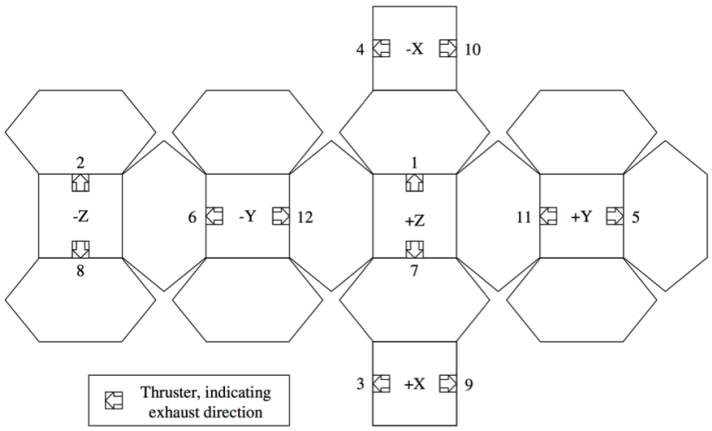


Figure 2-5: Example of the authority matrix (left) and thruster layout on a SPHERES satellite (right) [12]

so that not all sets have to be evaluated when performing the optimization. Sets that will always be suboptimal are removed, so that only the basic feasible solutions remain, known as the COGs. Shoemaker [34] has used COGs in practice in Lockheed Martin's Orion program. There is some overhead required to generate the COGs, but once the groups are created the processing time is reduced significantly when compared with a full optimization.

Another mixer approach uses delta velocity (ΔV) control of the thrusters. This solves the problem of determining how long to fire the thrusters, but not the problem of which thrusters to fire. This method uses the sensors available on the spacecraft to determine when a large enough change in velocity or angular rate has been performed on the system. The command to close the thrusters is only given once the commanded velocity change has been sensed. Jeffrey [13] developed a version of this ΔV control on the SPHERES testbed.

Through this examination of the literature on thruster selection logic, the literature survey of aggregative control schemes in Section 2.1, and the survey on model updates upon reconfiguration in Section 2.2, there is an area that has not yet been explored. No one in the literature has investigated the development of a method to autonomously update thruster selection logic when an aggregative spacecraft system reconfigures. Most of the literature discusses algorithms to optimize the selection of

thrusters in a stagnant spacecraft design. The only work that has been started in this area is investigating the effects of failed thrusters through FDIR algorithms [29]. Looking at how the mixers should adapt to the addition of new thrusters has not been explored. The work in this thesis serves to solve this problem through the development of reconfigurable thruster selection logic algorithms.

Chapter 3

Methodology for Thruster Selection Logic Reconfiguration

As discussed in the previous chapters, the goal of a reconfigurable thruster selection algorithm is to adapt to an aggregative spacecraft system's changes in the number of available thrusters and the location of those thrusters with respect to the center of mass of the system. For more details and a definition of standard thruster selection algorithms, or mixers, see Section 2.3. The algorithm must account for both the force and torque that each thruster will produce on the system when firing. The method must be modular, so that upon further system aggregation or disaggregation, the appropriate thrusters can be utilized for position and attitude control.

The methodology for this thruster selection logic reconfiguration follows in this chapter, but is briefly outlined here. First, there is a requirement for the data transfer of module properties between the new system submodules. Section 2.2 describes available methods, but for the purpose of this thesis, the business card approach will be used [26]. After the data is shared, a new physical system model is computed and stored in an updated, aggregated business card. Next, an authority matrix that maps available thrusters to forces and torques is constructed based on the information provided in this new aggregated business card. With this initial authority matrix, there are multiple paths that the reconfiguration can take depending on whether the CONOPS require precise control, fuel-efficiency, or maximum actuation. Each

of these actuation modes follows a different path to achieve reconfiguration of the thruster selection logic.

3.1 Business Card Formulation

The first step in reconfiguring the thruster selection logic is for all of the modules in the aggregative system to share their physical system properties with each other. The details of this data sharing depend on whether the system will use a master-slave architecture or a cooperative control architecture. In the master-slave architecture, only the slaves are responsible for sending their property data to the master, however in the cooperative control architecture, all modules need to share information with all others. For control, estimation and higher-level tasks, the type of architecture greatly affects appropriate algorithms. Nevertheless, the differences between these architectures will not be discussed further. The following methodology is applicable to both, given that all of the system property data eventually reaches the location where the reconfigurable mixer will be implemented.

The business card approach, developed by Mohan [26], is tailored specifically to aggregative spacecraft systems. The method itself is described in Section 2.2 and Figure 2-3, but the business card is required to contain specific pieces of information to be able to perform reconfiguration on the thruster selection logic. An example business card for a general aggregative spacecraft control problem is shown in Figure 3-1.

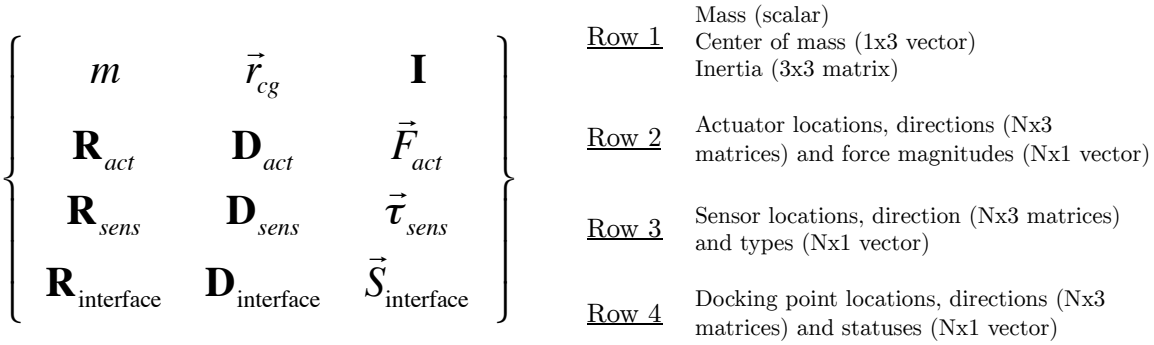


Figure 3-1: Example of a generic business card

Figure 3-1 is meant to represent the software structure of a business card with fields for the appropriate properties. The business card would not be structured in a cell array, or matrix format, but would be a collection of variables assigned values of according size. For the reconfigurable mixer problem, not all of the information in this general business card would be needed. Specifically, the inertia and sensor properties are not required. The required pieces of data are the mass, center of mass, actuator locations, actuator directions, actuator force magnitudes, docking port locations, and docking port directions. For other problems, more information than is shown in Figure 3-1 would likely be needed. For example, structure-control interaction problems would likely need to include stiffness properties and module dimensions.

Before aggregation, each module has its own business card, but after docking, the modules swap business cards so that all of the system's business cards are collected into specific location(s). In these location(s) in software, the business cards must be combined into an overall system business card for the purposes of knowing the new properties of the system and providing modularity for further aggregation and disaggregation. For a reconfigurable mixer, this combination of business cards consists of calculating the new system's center of mass, the thruster locations with respect to the new center of mass, and the thruster directions in the new coordinate frame. In a general setting, the new mass, inertia, and sensor locations and directions are also desired characteristics to calculate when combining business cards. The following shows the mechanics of combining and calculating the new business card parameters.

For the following set of calculations, we are assuming that two modules (Module A and Module B) have docked with each other and now are combining business cards. The first and easiest calculation is finding the new mass of the system, m_{agg}, from the individual masses of Module A and Module B, m_A and m_B respectively. This computation is the simple addition shown in Equation 3.1.

$$m_{agg} = m_A + m_B \tag{3.1}$$

The next mass property to update is the center of mass. Calculating the new system's center of mass, $r_{cg,agg}$, becomes complicated quickly if the modules have misaligned coordinate frames once docked. If the individual business cards have data in vectors with respect to their own coordinate frames, we cannot simply add these vectors together. We first need to convert all vectors into a common reference frame before proceeding. For example, in Figure 3-2 we need to convert each of Module B's vectors (blue) into Module A's coordinate frame (black). We can complete this transformation of coordinate systems using a rotation matrix, ${}^{A}\mathbf{\Phi}^{B}$, from Module B's coordinate frame to Module A's coordinate system. As a side note and for generality, each module is considered to have a reference point (e.g., the geometric center of the spacecraft) that is different than the center of mass of the module. This fact is shown in Figure 3-2, where the solid dot is the module's reference point and the center of mass is shown by the standard opposite-quadrant-shaded circle symbol.

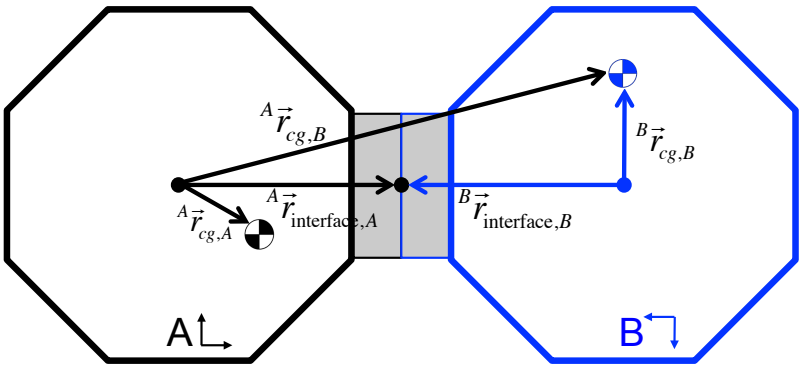


Figure 3-2: Vector locations of two generic, docked spacecraft centers of mass: Module A in black and Module B in blue

The first step in computing the new center of mass is thus finding the location of Module B's center of mass with respect to Module A's reference point in Module A's coordinate frame. The new vector to Module B's center of mass is named ${}^{A}\vec{r}_{cg,B}$. This vector can be computed from the distance vector to Module B's center of mass from Module B's reference point in Module B's coordinate frame, ${}^{B}\vec{r}_{cg,B}$, and the two interface distance vectors from each module with respect to their own coordinate frames, ${}^{A}\vec{r}_{interface,A}$ and ${}^{B}\vec{r}_{interface,B}$. Equation 3.2 shows how to compute the desired ${}^{A}\vec{r}_{cg,B}$.

$$ {}^{A}\vec{r}_{cg,B} = {}^{A}\vec{r}_{interface,A} - {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{r}_{interface,B} + {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{r}_{cg,B} \tag{3.2} $$

Once we compute this parameter, we now know the relevant vectors required for the determination of the center of mass of the system in Module A's coordinate frame, ${}^{A}\vec{r}_{cg}$, as shown in Figure 3-3. The computation of the location of the new center of mass is a weighted average of the two modules' locations to their respective center of mass. Equation 3.3 shows the calculation of this new value without completing the intermediary step shown in Equation 3.2. Equation 3.4 shows the computation of the new center of mass using the result from Equation 3.2.

$$ {}^{A}\vec{r}_{cg} = \frac{1}{m_{agg}}\left[m_A\,{}^{A}\vec{r}_{cg,A} + m_B\left({}^{A}\vec{r}_{interface,A} - {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{r}_{interface,B} + {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{r}_{cg,B}\right)\right] \tag{3.3} $$

$$ {}^{A}\vec{r}_{cg} = \frac{1}{m_{agg}}\left[m_A\,{}^{A}\vec{r}_{cg,A} + m_B\,{}^{A}\vec{r}_{cg,B}\right] \tag{3.4} $$

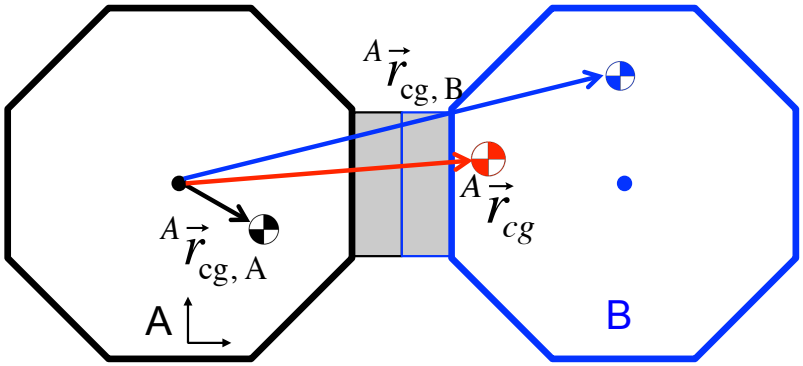


Figure 3-3: Vector location of the center of mass of the generic aggregated system

Moving forward, the next parameter to compute is the updated inertia tensor of the aggregated system with respect to the new center of mass. The new inertia tensor, ${}^{A}\mathbf{I}_{cg}$, is calculated through the Parallel Axis Theorem. The Parallel Axis Theorem is useful in computing the moment of inertia about another point of reference as shown in scalar form as Equation 3.5, where r is the distance between the reference points.

$$ I = I_{cg} + mr^2 \tag{3.5} $$

However, since we are dealing with inertia tensors, this must be represented in a more general form that first requires the calculation of appropriate vectors. In particular, we need to compute the vector pointing from the center of mass of each module to the combined center of mass, ${}^{A}\vec{r}_{A}$ and ${}^{A}\vec{r}_{B}$. This can be done in Equations 3.6 and 3.7 using previously known variables.

$$ {}^{A}\vec{r}_{cg} - {}^{A}\vec{r}_{cg,A} = {}^{A}\vec{r}_{A} \tag{3.6} $$

$$ {}^{A}\vec{r}_{cg} - {}^{A}\vec{r}_{cg,B} = {}^{A}\vec{r}_{B} \tag{3.7} $$

Once we have these vectors, we can compute the aggregated inertia tensor as is shown in Equation 3.8, where ${}^{A}\mathbf{I}_{cg,A}$ and ${}^{B}\mathbf{I}_{cg,B}$ are the modules' moments of inertia about their own centers of mass in their own reference frames, and $\mathbf{E}_3$ is the 3x3 identity matrix (the "E" derived from the first letter of the word "eye" because the inertia tensor already took "I").

$$ {}^{A}\mathbf{I}_{cg} = {}^{A}\mathbf{I}_{cg,A} + m_A\Big({}^{A}r_A^2\mathbf{E}_3 - {}^{A}\vec{r}_A\,{}^{A}\vec{r}_A^{\,T}\Big) + \ \ldots \\ \big({}^{A}\mathbf{\Phi}^{B}\big)\big({}^{B}\mathbf{I}_{cg,B}\big)\big({}^{A}\mathbf{\Phi}^{B}\big)^{T} + m_B\Big({}^{A}r_B^2\mathbf{E}_3 - {}^{A}\vec{r}_B\,{}^{A}\vec{r}_B^{\,T}\Big) \tag{3.8} $$

Now that the mass properties of the aggregated system have been computed and added to a combined business card, we can look at how to determine the new actuator and sensor locations and directions. These computations are fairly straightforward, and are similar in vector math to the computation in Equation 3.2, because we are attempting to convert vectors in one coordinate frame to another. All of the thrusters in Module A keep their locations with respect to the new coordinate frame as it is equivalent to Module A's coordinate frame. For those thrusters in Module B, we need to compute each of their new distances, ${}^{A}\vec{r}_{thruster,i}$, in Module A's coordinate frame. Equation 3.9 uses their location in Module B's frame, ${}^{B}\vec{r}_{thruster,i}$, and performs the same conversion as Equation 3.2. This operation can be performed for any location

originally in Module B's frame, such as sensors or docking ports. Converting the unit vectors from Module B's frame to Module A's frame is a simple left multiplication of the rotation matrix, ${}^{A}\mathbf{\Phi}^{B}$, as shown in Equation 3.10. This can be performed for all directional vectors in the business card, such as thruster directions, sensor directions or docking port directions.

$$ {}^{A}\vec{r}_{thruster,i} = {}^{A}\vec{r}_{interface,A} - {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{r}_{interface,B} + {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{r}_{thruster,i} \tag{3.9} $$

$$ {}^{A}\vec{d}_{thruster,i} = {}^{A}\mathbf{\Phi}^{B}\,{}^{B}\vec{d}_{thruster,i} \tag{3.10} $$

With these computations, we now have updated all of the relevant properties of the system and thus have an aggregated business card for the system. This aggregated business card is available for any algorithm to use as desired. The updated mass properties of the system are typically used in the computation of new gains for control laws, the sensors locations are input into the estimator, and the additional thrusters and new thruster locations and center of mass can now be used in the thruster selection logic reconfiguration process.

3.2 The Authority Matrix

3.2.1 Traditional Construction

The central element in thruster selection logic is the authority matrix. The authority matrix allows the easy integration of a mixer into modern state space controllers as it is used to linearly map an input to an output. For reference, the common state space representation of a control system is shown in Equations 3.11 and 3.12 , where $\mathbf{x}(t)$ are the states of the system and $\mathbf{u}(t)$ are the inputs to the system.

$$ \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{3.11} $$

$$ \mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) \tag{3.12} $$

Thus, because thruster selection logic is interested in controlling and regulating the inputs to the system in the form of thruster forces, the B matrix is closely related to the concept of an authority matrix. In a 6-DOF environment, the inputs to the system are usually the six possible axial forces and torques. In thruster-controlled spacecraft, there are no mechanisms that can directly produce those six forces and torques individually. Each thruster is only capable of producing a positive force paired with a specific torque based on the thruster's distance from the spacecraft center of mass. Therefore, if an individual force is desired, an additional thruster will need to fire to offset the torque created by the first and vice versa for only producing an individual torque. The authority matrix solves this problem. The authority matrix is used to convert any commanded forces and torques into the required forces that individual thrusters should create to collectively produce the total required forces and torques on the system. Also known as the mixing matrix, and to reduce confusing with the state-space control $\mathbf{A}$ matrix, the authority matrix is called $\mathbf{M}$. For the remainder of the thesis, the commanded forces will be represented by a capital F and commanded torques by τ. The individual thrusters' required forces will be represented by a lower case f_i. The authority matrix formulation is shown in Equations 3.13 and 3.14.

$$\mathbf{f} = \mathbf{M}\mathbf{u} \tag{3.13}$$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \end{bmatrix} = \mathbf{M} \begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{3.14}$$

It follows from here, that if desired, one could theoretically solve for $\mathbf{u}(t)$ and replace the $\mathbf{u}(t)$ in the state space controller with $\mathbf{M}^{-1}\mathbf{f}$, so that the inputs to the system are now in terms of individual thruster forces instead of the six commanded

forces and torques. This changes Equation 3.11 to Equation 3.15. However, this representation is not always desired. In this thesis, to remain as general as possible, the authority matrix will be considered separate from the state space controller, as shown in Figure 2-4 in Section 2.3.

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{M}^{-1}\mathbf{f}(t) \tag{3.15}$$

The formulation of the authority matrix is of particular interest to the reconfigurable mixer, because every time the system reconfigures to add and remove thrusters or changes its center of mass, the authority matrix requires an update.

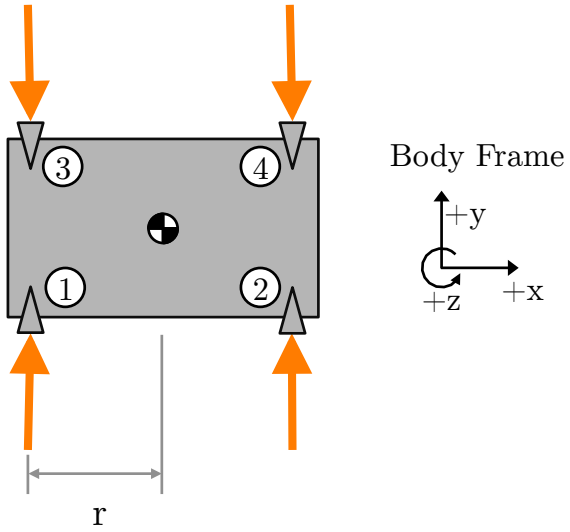


Figure 3-4: Simple example of 2-DOF spacecraft thruster layout

To start out, we will consider a simple 2-DOF system shown in Figure 3-4. In this system, each thruster is considered to produce the same magnitude of force and have the same moment arm to produce the same magnitude of torque. With these assumptions, we can implement a thruster pair formulation by combining thrusters 1 and 3 into the first pair and thrusters 2 and 4 into the second pair. These thrusters are paired together because they produce equal and opposite forces and torques on the system. This removes the need to ensure all required forces are positive. If a positive force is required for one of the thruster pairs, the positive thruster fires (thruster 1 in the first pair), but if a negative force is required, the negative thruster would fire

(thruster 3 in the first pair). In general, the logic is shown in Equation 3.16.

$$f_{i,j} = \begin{cases} f_i & \text{if } > 0 \\ f_j & \text{if } < 0 \end{cases} \tag{3.16}$$

$$\begin{bmatrix} F_y \\ \frac{\tau_z}{r} \end{bmatrix} = \mathbf{M^{-1}} \begin{bmatrix} f_{1,3} \\ f_{2,4} \end{bmatrix} \tag{3.17}$$

Solving Equation 3.17 for $\mathbf{M^{-1}}$ and inverting is the most intuitive method of deriving the authority matrix. In this formulation, the first row of the inverse authority matrix linearly combines the appropriate thrusters to produce a positive force on the system. The second row linearly combines the thrusters to produce a positive torque on the system. The first column multiplies the first thruster pair and the second column multiplies the second pair. For this 2-DOF system, the resulting $\mathbf{M^{-1}}$ is that in Equation 3.18. Taking the inverse yields the authority matrix $\mathbf{M}$ in Equation 3.19.

$$\begin{bmatrix} F_y \\ \frac{\tau_z}{r} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} f_{1,3} \\ f_{2,4} \end{bmatrix} \tag{3.18}$$

$$\begin{bmatrix} f_{1,3} \\ f_{2,4} \end{bmatrix} = \mathbf{M} \begin{bmatrix} F_y \\ \frac{\tau_z}{r} \end{bmatrix} \tag{3.19}$$

Hilstad [12] does a similar analysis in detail for the SPHERES satellites. In Hilstad's analysis, there are 6 DOF, 12 thrusters and 6 thruster pairs. Hilstad makes the same assumptions that all thrusters produce the same magnitude of force and have the same moment arm to produce the same magnitude of torque.

3.2.2 Reconfiguration

Upon every system aggregation and disaggregation, the authority matrix needs to be updated. For example, consider the case where two spacecraft modules, each with their own propulsion systems, have docked. To control the entire system as one, the controller needs to be able to convert forces and torques into thruster firing times that

will maneuver the system as a whole. The individual modules each have their own authority matrices that help them achieve this requirement for themselves, however these matrices will not accurately return appropriate firing times for the combined system. Firstly, the authority matrix must change, because the number of available thrusters has increased, so the commanded forces and torques must be split between all of the thrusters. Secondly, the center of mass has likely changed dramatically, so the torques that each thruster produce on the system will vary significantly, and possibly even completely change direction if the center of mass moves outside the thruster envelope. The effects of this change in center of mass apply even in the case where the docked module did not have its own propulsion system. So, after the modules have docked and have shared business cards as described in Section 3.1, an updated authority matrix is always required.

In aggregative spacecraft systems, the formulation of the authority matrix is not as simple as described in Section 3.2.1. Thrusters are likely to have varying force magnitudes and have different moment arms because the center of mass of the system changes. In addition, there will likely be more thrusters available than system DOF. Thus we cannot make the assumptions we made in Section 3.2.1 and need to create a more general, easily automatable approach to creating an updated authority matrix. We are able to do this solely with the information provided in the aggregated system's business card from Section 3.1.

From the business card, we have the updated location of the aggregated system's center of mass in the aggregated reference frame, we have the updated location of all of the thrusters in the aggregated frame, and we have the directions and force magnitude of each thruster in the aggregated frame. We do not yet have the distance vectors from the center of mass to the thrusters, so this is the first step that must be performed as shown for an individual thruster, i, in Equation 3.20.

$$^{A}\vec{r}_{thr,i} - {}^{A}\vec{r}_{cg} = \vec{r}_{thr,i} \tag{3.20}$$

The thruster force vectors are calculated in Equation 3.21, multiplying the force

magnitude from the business card with the thruster direction in the business card. As is the case with the distance vector, this must be performed for all thrusters.

$$f_{thr,i} = F_{thr,i}\vec{d}_{thr,i} \tag{3.21}$$

Again, it is more intuitive (and helpful for generality later) to derive the inverse authority matrix instead of the authority matrix itself. In the aggregative system, however, the authority matrix will likely be overdetermined, meaning that it will be a rectangular matrix. Because the inverse cannot be taken of a rectangular matrix, the Moore-Penrose pseudoinverse of the authority matrix, $\mathbf{M}^+$, must be used instead as defined in Equation 3.22 and implemented in Equation 3.23.

$$\mathbf{M} = \left(\mathbf{M}^{+T}\mathbf{M}^{+}\right)^{-1}\mathbf{M}^{+T} \tag{3.22}$$

$$\mathbf{u} = \mathbf{M}^{+}\mathbf{f} \tag{3.23}$$

To ensure that all of the information about thruster forces and torques is accurately represented in the authority matrix, the matrix cannot simply be filled in with ones and negative ones as the simple 2-DOF example in Section 3.2.1 was. For the general 6-DOF case, the first three rows are the x-axis, y-axis and z-axis force that the corresponding thruster will produce. The last three rows are the x-axis, y-axis and z-axis torque that the corresponding thruster will produce. The columns are representative of each individual thruster. The force is that from Equation 3.21. The torque is the cross product of the distance vector in Equation 3.20 and the force in Equation 3.21. The generic formulation of this pseudoinverse authority matrix is shown in Equation 3.24. Note that as read on the page, the transpose of $\mathbf{M}^+$ is shown. $\mathbf{M}^+$ has six rows for the forces and torques and a number of columns equal to the number of thrusters (a 6xN matrix).

$$\begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} \vec{f}_{thr,1} & \vec{r}_{thr,1} \times \vec{f}_{thr,1} \\ \vec{f}_{thr,2} & \vec{r}_{thr,2} \times \vec{f}_{thr,2} \\ \vdots & \vdots \\ \vec{f}_{thr,N} & \vec{r}_{thr,N} \times \vec{f}_{thr,N} \end{bmatrix}^T \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \end{bmatrix} \tag{3.24}$$

The moment arm cannot be pulled out of the matrix in this general form as it was pulled out in Equation 3.17, because the thrusters have a variety of moment arms. This results in an interesting situation, where the $\mathbf{M}^+$ matrix is heterogeneous in terms of units; half of the matrix is in units of force and half in units of torque. This has implications discussed later in Section 3.3.1 when directly converting the pseudoinverse, $\mathbf{M}^+$, into the authority matrix, $\mathbf{M}$. The information needs to be represented in the matrix however, so there is not an easy way around this.

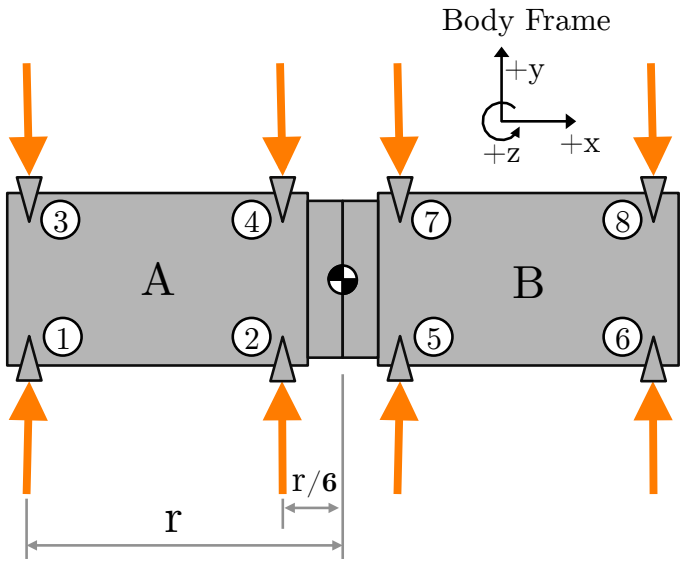


Figure 3-5: Example of 2-DOF, two-module, aggregated spacecraft thruster layout

As an example, consider a situation where two of the 2-DOF spacecraft modules in Section 3.2.1 have docked and have swapped and aggregated business cards. Figure 3-5 depicts this scenario and labels the relevant moment arms. Keeping the

thruster pair formulation, we can find the pseudoinverse authority matrix as shown in Equation 3.25. The cross products in Equation 3.24 reduce to scalar multiplications in the simple 2-DOF case.

$$\begin{bmatrix} F_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -r & -\frac{1}{6}r & \frac{1}{6}r & r \end{bmatrix} \begin{bmatrix} f_{1,3} \\ f_{2,4} \\ f_{5,7} \\ f_{6,8} \end{bmatrix} \tag{3.25}$$

It is not always the case that the thrusters will be able to form thruster pairs; for example, when there are an odd number of thrusters or when the center of mass moves in a way that changes the moment arms so that thrusters do not line up appropriately. Therefore, Equation 3.26 does not assume any thruster pairing. Note that in this formulation the only valid results are those that have positive required thruster forces: $\mathbf{f} > \mathbf{0}$.

$$\begin{bmatrix} F_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ -r & -\frac{1}{6}r & r & \frac{1}{6}r & \frac{1}{6}r & r & -\frac{1}{6}r & -r \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \end{bmatrix} \tag{3.26}$$

The creation of the pseudoinverse of the authority matrix is the first step in reconfiguring the thruster selection logic. The result is an overdetermined matrix. This means that there could potentially be many $\mathbf{f}$ vectors that produce the same $\mathbf{u}$ vector depending on which thrusters are most desirable to the overall CONOPS. The next step is to determine which actuation mode the system should operate in and to implement the appropriate algorithm as detailed in Section 3.3.

3.3 Actuation Modes

There are four distinct actuation modes that can be implemented with reconfigurable thruster selection logic. These actuation modes each require different algorithms to achieve their characteristic behavior: computational ease, precision, fuel-efficiency and agility. These four actuation modes arise because of the additional thrusters available. If positioned correctly, only twelve thrusters are generally needed to command any force and torque direction on a spacecraft. In an aggregative spacecraft system, when there is a surplus of thrusters, the system gains some flexibility in what to do with those extra thrusters. It is with this flexibility that the four actuation modes are born: the nominal mode (Section 3.3.1), precision mode (Section 3.3.2), fuel-efficient mode (Section 3.3.3), and agile mode (Section 3.3.4).

3.3.1 Nominal Mode

The most basic reconfiguration method is known as the nominal actuation mode. It involves the least amount of computation and fewest lines of code, and it requires little more conceptual understanding than what is detailed in Section 3.2.2 about reconfiguring the authority matrix. The nominal actuation mode starts where Section 3.2.2 left off. The pseudoinverse of the authority matrix is now known. To calculate the corresponding authority matrix for the nominal actuation mode, we simply need to take the pseudoinverse of this matrix. The resulting Nx6 matrix can then be used directly as the authority matrix, $\mathbf{M}$. The vector of commanded forces and torques is left multiplied by the authority matrix to directly produce the required forces for each thruster, $\mathbf{f}$.

The only difficult part here is choosing an appropriate method to compute this pseudoinverse or determining if that is actually necessary. Directly using the Moore-Penrose formulation would require the computation of an inverse and a few matrix multiplications. More efficient methods that avoid directly computing the inverse exist including LU decomposition and Cholesky decomposition. For example, a brief overview of how to implement the Cholesky decomposition process to solve for re-

quired firing times follows in Equations 3.27 to 3.31.

First, form the symmetric positive definite matrix, $\mathbf{A}$, as required by the Cholesky factorization in Equation 3.27.

$$\mathbf{A} = \mathbf{M}^{+}\mathbf{M}^{+T} \tag{3.27}$$

Next perform a Cholesky decomposition on $\mathbf{A}$ to obtain the lower triangular matrix $\mathbf{L}$ given by Equation 3.28.

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{T} \tag{3.28}$$

Next use forward substitution to solve for $\mathbf{y}$ in Equation 3.29.

$$\mathbf{L}\mathbf{y} = \mathbf{u} \tag{3.29}$$

Solve for $\mathbf{x}$ in Equation 3.30 through back substitution.

$$\mathbf{L}^{T}\mathbf{x} = \mathbf{y} \tag{3.30}$$

Finally, to obtain the required forces and torques, use Equation 3.31.

$$\mathbf{f} = \mathbf{M}^{+T}\mathbf{x} \tag{3.31}$$

Because we know that the $\mathbf{A}$ matrix is always symmetric positive definite by definition in Equation 3.27, the Cholesky decomposition procedure will be about twice as fast as the LU decomposition method. Regardless of whether the inverse has been computed or Cholesky factorization has been implemented, identical results are returned.

Some would argue that matrix inverses should never be computed as they are less computationally efficient in solving the $Ax = b$ problem. The Cholesky method above should satisfy their concerns. However, this is not always optimal for the system. If the pseudoinverse authority matrix is being updated continuously as components move in the system and change the center of mass, it will indeed be more efficient. If

instead, the pseudoinverse authority matrix is only updated upon docking, or upon major reconfigurations, it is likely more computationally efficient in the long run to initially calculate the authority matrix directly using the Moore-Penrose formulation or similar, because all subsequent calculations will be minimized to a single matrix multiplication rather than solving the Cholesky factorization problem. One matrix inverse computation and many matrix multiplications would be more efficient than many Cholesky factorizations.

The results from this direct use of the pseudoinverse authority matrix are somewhat interesting. As discussed in Section 3.2.2, to preserve all of the necessary information about moment arms and forces, the pseudoinverse authority matrix has heterogeneous units. Accordingly, when the required thruster forces are computed there is not an even distribution across all thrusters. The force magnitude per thruster ends up being proportional to the moment arm of that thruster as can be seen in Figure 3-7. Unintentionally, this scaling creates a more fuel-efficient firing of thrusters. The thrusters with larger moment arms produce larger torques on the system per unit of fuel consumed assuming uniform force capabilities of all thrusters. These thrusters are required to fire longer by the authority matrix and thus there is an element of fuel efficiency involved here. However, this is not always desired, as it limits the full actuation capabilities of the spacecraft. It is also not completely fuel-efficient. This behavior is not present in the translational case. Given uniform thrusting capabilities for each thruster, the translation maneuver will use the appropriate thrusters uniformly as shown in Figure 3-6.

The process is not complete yet however. As of now, only required forces from thrusters have been computed. Variable force thrusters do exist, however the majority of spacecraft use pulse-controlled thrusters. Pulse-controlled thrusters produce an approximately constant amount of force for a given duration of time, rather than a given force. Thus, we want to control thruster firing times not required forces. To compute the firing times from the forces requires the multiplication of the final forces by a scaling factor. This scaling factor is called the slope force, which is affected by the duty cycle, D, of the thrusters. The slope force, S_f is computed to produce

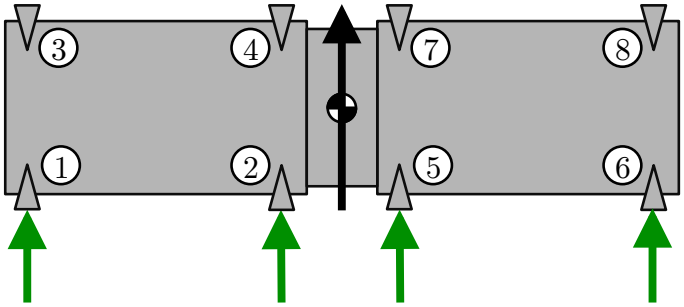


Figure 3-6: The representative thruster behavior of the 2-DOF aggregated spacecraft translating in the nominal actuation mode

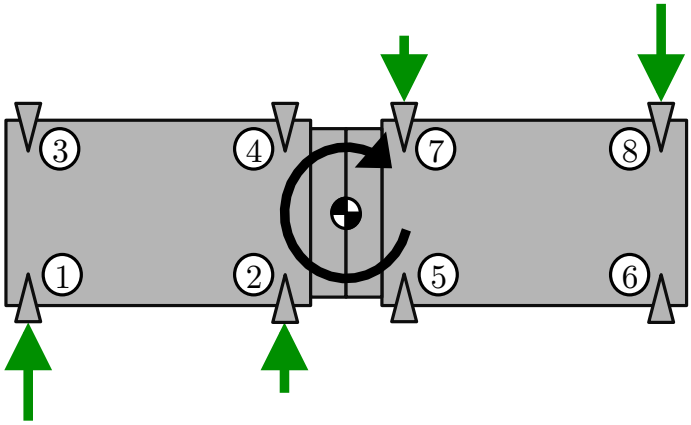


Figure 3-7: The representative thruster behavior of the 2-DOF aggregated spacecraft rotating in the nominal actuation mode

the same impulse that a thruster of constant force would have produced during a full control cycle period, T. It is calculated in Equation 3.32 and used to determine thruster firing times, $\mathbf{t}_{thr}$ in Equation 3.33.

$$S_f = \frac{DT}{F_{thr,i}} \tag{3.32}$$

$$\mathbf{t}_{thr} = S_f \mathbf{f} \tag{3.33}$$

At this point, there is still the possibility that negative thruster times are being computed. This must be accounted for with the determination of thruster pairs.

Thruster pairs can be determined by comparing the force and torques that individual thrusters can provide. Equal and opposite thrusters can be considered pairs. Determining the pairs boils down to a for-loop comparison of the thrusters as is done in the code in Appendix C. Once the thruster pairs have been determined, the negative times should be shifted to the other thruster in the pairing.

A final consideration that the thruster selection logic must consider is maximum and minimum on-times. If a thruster is commanded to fire above its saturation limit, it should only fire for the saturation limit. In addition, all other thrusters must scale their times to preserve the direction of the force and torque. Therefore, all thrusters firing times must be scaled by a value equal to the ratio of the saturation limit to the commanded force. Additionally, if a thruster is commanded for less time than the thruster is capable of opening and closing in, its required firing time should be zeroed out.

This nominal reconfiguration is the simplest and theoretically the most computationally efficient actuation mode. On the other hand, the performance of this actuation mode can be considered worse than the others as it is not tailored for a specific application. This method can be improved upon in the areas of precision, fuel-efficiency, and agility as is discussed in the next few sections.

3.3.2 Precision Mode

In traditional thruster selection logic methods, the control of thruster firing times is cut off when the firing time is too small. Mechanisms that open the thruster valve take a finite amount of time to fully actuate, meaning that the full force of the thruster is not achievable for a period of time while the valve is opening. Additionally, the transient behavior of the force produced by a thruster while it is opening is fairly noisy. It takes time for this transient behavior to settle as seen by Chen's propulsion characterization work [7]. These characteristic thruster on-times directly limit the minimum pulse width that can be commanded to the thruster. This minimum pulse width then limits the minimum possible force and torque that can be commanded by the control algorithm. Effectively, this situation creates a bottleneck for the precision

of the system as very small forces and torques cannot be applied to the system. A deadband exists in the velocity control of the spacecraft because of this gap between zero actuation and the minimum pulse width actuation. This deadband results in undesired backward and forward movement during a position hold maneuver. Nevertheless, this level of precision can be improved without advancing any technology to improve thruster on-time performance.

The precision actuation mode attempts to solve this problem or, at the very least, improve the system's precision performance by a substantial margin. The flight computer of a spacecraft will run in real time at a particular speed. The cycle period of flight computers is generally substantially lower than the thruster minimum on-times. This fact means that thrusters can be controlled at a faster rate than they can physically be turned on and off. Furthermore, the effective minimum pulse width for a thruster pair can be improved to this flight software clock period and remove the bottleneck from the minimum thruster on-time requirement. If a commanded firing time is less than the traditional minimum pulse width of the thruster, the opposing thruster in its thruster pair can be fired and the difference in thruster on-times will be the net effect on the system. Thus, since the thrusters can be turned off at times specified by the precision of the clock period in the flight software, the system improves to the precision of the clock period. For example, on the SPHERES satellites, the minimum pulse width is about 20 ms while the clock period is 1 ms. See Figure 3-8 for clarification.

In an aggregative system, there is more room for improvement in precision in the torque specifically, because of the fact that some thrusters have shorter moment arms than others. Firing the thrusters with shorter moment arms when running into the deadband can ensure that the smallest possible torque is being created on the system. Furthermore, this can be paired with the differential thruster firing technique to be fully precise. Figure 3-9 shows how the differential thruster firing technique would be used on a commanded translation in the simple 2-DOF example. Figure 3-10 shows the two options for improving precision for torque in the simple 2-DOF example.

The differential thruster firing technique can be added to the end of any of the

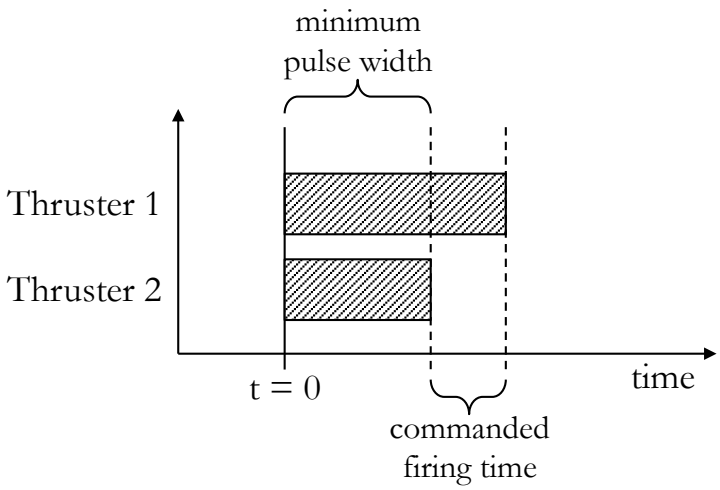


Figure 3-8: Differential thruster firing technique

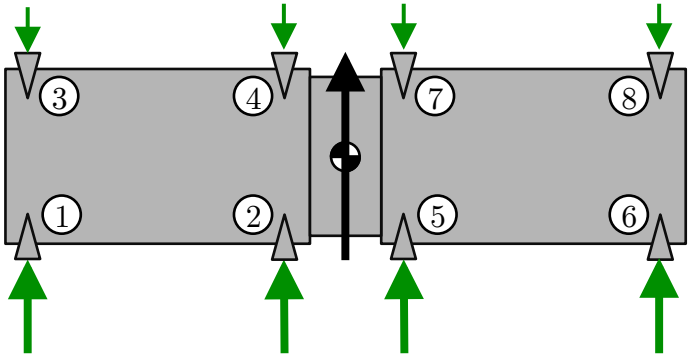


Figure 3-9: The representative thruster behavior of the 2-DOF aggregated spacecraft translating in the precision actuation mode

actuation modes fairly easily and requires only a few extra lines of code and very little additional runtime. As an example of implementation, consider the nominal actuation mode discussed in Section 3.3.1. The same process of using the pseudoinverse with Cholesky factorization to solve for the firing times can be implemented identically. The only difference is that after the firing times have been scaled to account for the saturation limits, some logic statements can be checked to see if the commanded firing time is less than the minimum pulse width but greater than the clock cycle period. If this is the case, add the minimum pulse width to both the thrusters in that pair. Adding the minimum pulse width is the most fuel-efficient way to utilize the differential thruster firing technique, as that is the minimum time the opposing

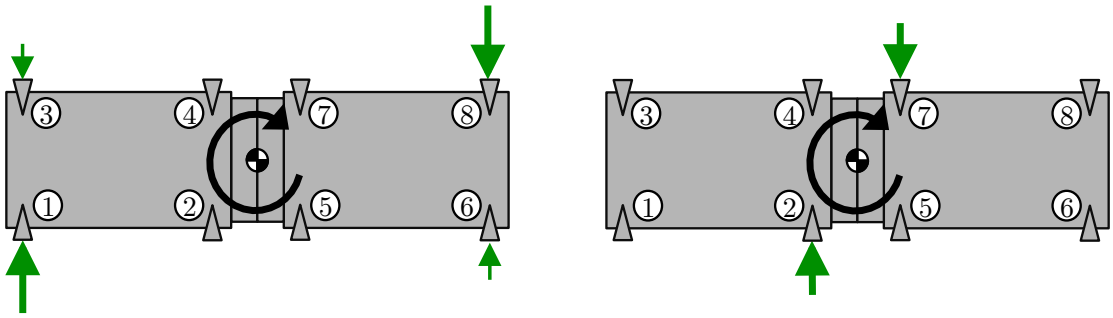


Figure 3-10: The representative thruster behavior of the 2-DOF aggregated spacecraft rotating in the precision actuation mode. Differential thruster firings (left) and minimum moment arms (right).

thruster can effectively fire. See the code in Appendix C for specific implementation added at the end of the nominal actuation mode. One additional step is required if the mixer needs to center the thruster pulses in the control cycle. In this case, to achieve the behavior shown in Figure 3-8, the portion of the firings that cancel out should start at the same time rather than each individually being centered in the control period.

On the SPHERES satellite, the differential thruster firing technique can achieve a minimum commanded impulse of 0.12 mNs where the other actuation modes would only be able to achieve a minimum commanded impulse of 2.4 mNs. Thus, the precision mode theoretically improves the precision of the mixer by a full order of magnitude. The resulting deadband should thus be about 5% of the width of the traditional deadband. Unfortunately, this method will also result in a dramatic increase in fuel consumption of the satellite during position hold maneuvers. If fuel-efficiency is a goal, this technique should likely be avoided. However, if the precise control of spacecraft pointing is required and fuel-efficiency is less of a concern, this actuation mode could be very effective.

3.3.3 Fuel-Efficient Mode

The requirement of minimum propellant consumption is very common in space systems. Extra propellant means extra launch mass and larger mass moments of inertia

to control. This fuel-efficient requirement attempts to reduce the prohibitive costs associated with this additional needed propellant mass. Thus it would be very beneficial for the thruster selection logic algorithm to utilize thrusters in a fuel-optimal manner. In aggregative systems, there is a great opportunity to capitalize on the additional thrusters available for use. Some of these thrusters will likely be more efficient in terms of thruster performance or specific impulse (I_{sp}). Also, some of the thrusters will have larger moment arms and produce more torque per mass of propellant than other thrusters that are closer to the center of mass of the system. The optimal performance of the system, in a fuel consumption sense, would be to use only these most efficient thrusters. The difficulty comes in appropriately selecting these optimal thrusters.

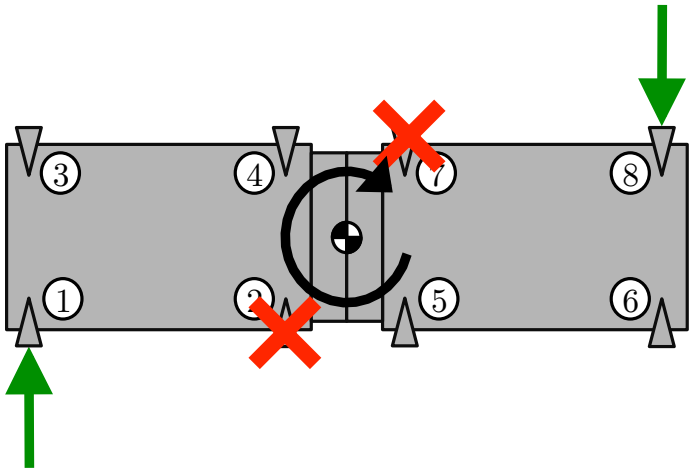


Figure 3-11: The representative thruster behavior of the 2-DOF aggregated spacecraft rotating in the fuel-efficient actuation mode.

For the simple 2-DOF example, all of the thrusters have the same I_{sp} and can be considered identical. Therefore, as shown in Figure 3-11, the outside thrusters are the most fuel-efficient for producing torques on the system as they have larger moment arms. For translation, Figure 3-6 from Section 3.3.1 describes the fuel optimal approach as all thrusters are of equal efficiency for translation. The inside thrusters would never be used for torques in a fuel optimal sense. This seems fairly obvious intuitively, but developing a method to accomplish this autonomously is more difficult

for the general case. The general case is in full 6-DOF and must apply for any number of thrusters with their own unique efficiencies and locations.

Fortunately, the formulation of the pseudoinverse authority matrix forces the problem to be linear. It is thus possible to perform optimization for fuel efficiency using linear techniques. Linear optimization algorithms are well known and have been implemented successfully in every single engineering field. With this rich history, effective optimization algorithms have already been developed. The choice of the algorithm should be tailored to the thruster selection problem however. Some algorithms are faster but make certain assumptions that may not be appropriate for our problem.

The formulation of the single objective linear optimization problem is as follows. The objective function, J, should be the summation of the required propellant from all thrusters to achieve the commanded forces and torques. To calculate this, the propellant cost vector, $\mathbf{c}$, is required. This vector does not need to translate required thruster forces directly to the propellant consumed, but should show the relative differences between thruster efficiencies. This cost vector can be created by comparing thruster I_{sp}'s, or if all thrusters have the same performance, can simply be a vector of ones. After this cost vector is known, the objective function is the transpose cost vector right multiplied by the vector of required thruster forces as shown in Equation 3.34.

$$J = \sum_{j=1}^{N} c_j f_j = \mathbf{c}^T \mathbf{f} \tag{3.34}$$

The optimization is the minimization of this objective function while varying $\mathbf{f}$, the required thruster forces, as shown in the linear optimization formulation in Equation 3.35.

$$\begin{aligned} &\min J(\mathbf{x}, \mathbf{p}) \\ &s.t. \\ &g(\mathbf{x}, \mathbf{p}) = \mathbf{f} \geq 0 \\ &h(\mathbf{x}, \mathbf{p}) = \mathbf{M}^{+}\mathbf{f} = \mathbf{u} \end{aligned} \tag{3.35}$$

The design variables in the $\mathbf{x}$ vector are the required forces for each thruster. Therefore, $\mathbf{x}$ is really just equivalent to $\mathbf{f}$. The parameters in the $\mathbf{p}$ "vector" are the pseudoinverse authority matrix, $\mathbf{M}^{+}$, the commanded force and torque vector, $\mathbf{u}$, and the cost vector, $\mathbf{c}$. Note that it is very important that the constraints g and h are linear. The inequality constraint, g, is that the required forces from each thruster must be positive. This constraint removes the need to form thruster pairs later and makes the reconfiguration more robust. The equality constraint, h, ensures that only thruster forces that will produce the desired overall commanded system forces and torques are feasible in the optimization.

The selection of the initial point to start the optimization depends on which optimization technique is being used and whether that solution needs to be feasible or not. If the solution needs to be feasible, the nominal actuation mode process would need to be completed first to ensure that the constraints are being met. Then this non-fuel-optimal but feasible result can be used as the initial guess. If the optimization technique does not need a feasible solution to start, is suggested that the starting solution be the nominal mode calculation of the required forces without the correction for positive forces with thruster pairs. This approach removes the dependence on the thruster pair formulation and meets the equality constraint, which is notoriously hard and time consuming to find. This selection should improve the runtime; however, remember that starting the optimization with an infeasible solution will still increase the runtime.

A particular set of optimization techniques that works well for these problems is the active set algorithms, specifically the sequential linear-quadratic programming (SLQP) method. In the past, the Simplex method has also been shown to be successful in performing this optimization. The specific choice of optimization algorithm must

be able to handle equality and inequality constraints as well as infeasible initial points. For this thesis, an active set algorithm is used for the fuel-efficient mode. This method can handle linear constraints and small steps into the infeasible regions of the design space, which is important for this problem. This algorithm takes three general steps. First, it computes the gradients and Hessians. Second, it finds the steepest descent direction by solving the quadratic program. Third, it performs a search along a merit function in that direction to find the appropriate step size. It repeats this until the improvement in the objective function reaches the desired tolerance. This method is implemented in an option of MATLAB®'s `fmincon` function and for this thesis, is used as shown in the code in Appendix C.

The required thruster forces will be directly computed from the optimization algorithm. The only steps remaining are to multiply by the slope force as in Equation 3.33 to convert into required firing times, implement the saturation and minimum pulse width limits, and scale to preserve the direction. All of these steps are the same as discussed in in Section 3.3.1 in the nominal mode. Note that there is no requirement for determining thruster pairs or shifting firing times from one thruster to another in the pair. Enforcing the constraint, g in Equation 3.35, has relieved this thruster pair requirement.

Overall, there are a couple excellent benefits to the fuel-efficient actuation mode. First and foremost, the title characteristic is that all thruster firings will be completed to minimize the propellant consumption. The other benefit is that the optimization process removes the requirement to formulate thruster pairs. This makes the algorithm more robust to reconfiguration. If off-axis thrusters are used in the system, it becomes difficult, and in some cases impossible, to pair thrusters because some thrusters will not have a direct counterpart in the system. This layout is more likely to happen in an aggregative system than in a monolithic system, so using this fuel-efficient optimization technique enables more robust reconfiguration. One major disadvantage to the fuel-efficient actuation mode however is that it is very computationally expensive. The optimization algorithm requires at least an order of magnitude more operations than the pseudoinverse calculation and the solution of an

$Ax = b$ problem combined. Additionally, the maximum allowable forces and torques on the system will be less, because the algorithm is constrained to using only the efficient thrusters.

3.3.4 Agile Mode

The final actuation mode, the agile mode, is meant to utilize the full actuation capabilities of the aggregated spacecraft system. The level at which the nominal, precision and fuel-efficient modes saturate thrusters does not accurately reflect the full capabilities of the system. The agile mode attempts to enable the full firing time available for all thrusters. To achieve this maximum agility, while still preserving direction for both torque and force, is more complicated than what is implemented for the other actuation modes. The desired output is that all thrusters that produce a certain direction of force or torque will each have the same firing time and scale to match the desired force until they all reach the saturation limit at the same time. This behavior, where all thrusters are fired equally and ramped up together, is shown in Figure 3-12. Compare this figure to Figure 3-7 in Section 3.3.1 and Figure 3-11 in Section 3.3.3 to see how it differs from the nominal and fuel-efficient cases.

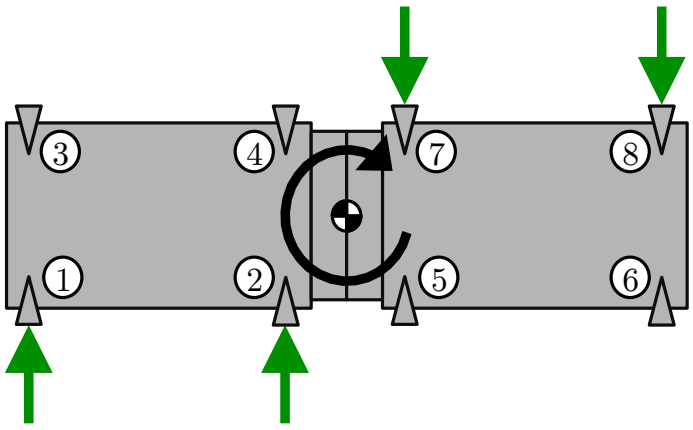


Figure 3-12: The representative thruster behavior of the 2-DOF aggregated spacecraft rotating in the agile actuation mode.

The process to achieve this agile behavior starts from the pseudoinverse authority matrix, $\mathbf{M}^{+}$, developed in Section 3.2.2 and involves a number of steps. The

MATLAB® code in Appendix C may be easier to follow and helps in getting a better understanding of the process. First, from $\mathbf{M}^+$, each thruster needs to be mapped to a pair with equal and opposite force and torque on the system. This process is the same thruster pair process as described in Section 3.3.1 in the nominal actuation mode.

The next step is to condense $\mathbf{M}^+$ to join thruster pairs together. In this step, $\mathbf{M}^+$ shrinks in half because the positive-force thruster in a pair absorbs the negative. Now $\mathbf{M}^+$ is in a formulation that allows both positive and negative thruster forces to be valid. These thruster pairs must be kept in memory, because they will be needed later to distinguish what a positive and negative required force means.

After that step, the $\mathbf{M}^+$ matrix needs to be condensed again based on what will be called torque groups for the remainder of this thesis. These torque groups are thruster pairs that have directionally equivalent forces and torques. Because all of the positive forces were taken in the first condensed $\mathbf{M}^+$ matrix, this grouping is more relevant to the torque produced by the pair and is thus known as a torque grouping. Checking that the forces and torques of a thruster pair are directionally equivalent reduces to verifying that the sign of every force and torque is identical. The objective of this torque grouping is to collect all thrusters that produce a specific torque and calculate a representative moment arm to be assigned to the group. This moment arm is the average of all thruster pairs in that torque group as can be seen by the representative torque group thruster shown in Figure 3-13.

After the torque groups are determined, the $\mathbf{M}^+$ matrix is ready to be condensed for a final time. The only difference between the columns in the current $\mathbf{M}^+$ matrix representing the members of a particular torque group should be the value in the row corresponding to the torque of the group. This value should be updated to reflect the average moment arm of the torque group. The forces should be identical and should not need updating. All of the columns can be merged into the same column in the condensed matrix. Again, this process is probably best understood by using the MATLAB® code in Appendix C.

In most situations this matrix will be a 6x6 matrix mapping the six commanded

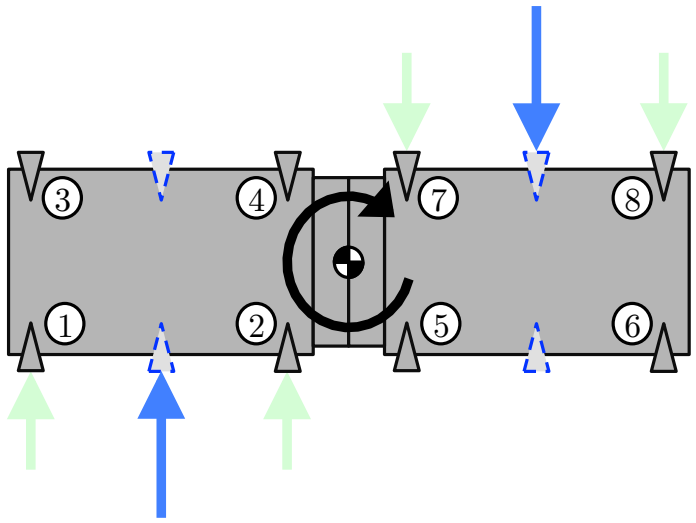


Figure 3-13: The creation of representative thrusters for a torque group in the 2-DOF aggregated spacecraft

forces and torques to the six resulting torque groups. The specific cases that will not result in a final 6x6 matrix are when the thrusters do not lie in the x, y or z plane of the center of mass and when the thrusters do not point in axis-aligned directions. In these two cases, the agile mode fails to utilize the full actuation capability of the system, but should still achieve better actuation that the nominal and fuel-efficient cases. This situation is an area of work that has not been explored through the efforts in this thesis and could be an area of future interest.

With this fully condensed matrix, a representative required thruster force for each torque group can be determined by the solution to the $Ax = b$ problem in the same manner as detailed in Section 3.3.1. The solution of this problem will be the required forces for each torque group, however, not the individual thruster forces. Thus, the final step is to distribute this representative force equally to all of the thruster pairs in the torque group. Once each thruster pair is assigned a specific force, the same process as detailed in Section 3.3.1 can be followed to get the final firing times to individual thrusters, to scale based on saturation, and to cut off based on the minimum pulse width.

In summary, the process takes the following steps:

1. Form pseudoinverse authority matrix, $\mathbf{M}^{+}$

2. Formulate thruster pairs

3. Condense $\mathbf{M}^+$ in half by combining thruster pairs

4. Find torque groups and representative moment arms

5. Condense $\mathbf{M}^+$ by combining torque groups

6. Solve $Ax = b$ problem to determine required forces from each torque group

7. Assign thruster firing times based on membership in torque groups

The agile actuation mode enables the full actuation capabilities of the aggregated spacecraft system. This characteristic can be very beneficial if maneuvers need to be completed in very short amounts of time. Space operations tend to be very slow due to the amount of human interaction in the process and the reliance on orbital mechanics to preserve fuel. If fully autonomous systems are implemented in the future, maneuvers have the potential to become much quicker. However, the consequence of improving the agility of the system is that it will not be as fuel-efficient. In addition the implementation of this algorithm is very complex and assumes that the system can be formulated into thruster pairs and torque groups. Thus, the algorithm is not as robust as the fuel-efficient actuation mode. Regardless, if the goal of aggregating a spacecraft system is to increase the actuation capabilities and speed of maneuvers, this agile actuation mode will prove incredibly useful.

3.4 Summary

The methodology of reconfiguring thruster selection logic upon the aggregation of spacecraft modules as discussed in this chapter is given as a generalized form that allows implementation on any system configuration. Assuming that the spacecraft is able to determine the appropriate physical parameters of the aggregated system, the methodology will work for any general spacecraft. This methodology will work with any number of thrusters, with any thruster locations and directions, with any compliment of thruster efficiencies and forces, with any center of mass location, and

with any number of modules. The only caveat is that the performance of the algorithm is dependent on the combination of system configuration and selected actuation mode. Even so, baseline performance is achievable for any spacecraft configuration.

The precise details of how to compute an aggregated business card, form the pseudoinverse authority matrix, and implement a specific actuation mode algorithm are described in the previous sections. In summary, the methodology is as follows after two spacecraft have docked or berthed:

1. Form aggregate business card for the combined system
2. Calculate the distance vectors from the center of mass to the thrusters
3. Form pseudo-inverse authority matrix, $\mathbf{M}^+$
4. Store the pseudo-inverse authority matrix, $\mathbf{M}^+$, in memory
5. For each commanded force and torque sent to the mixer by the controller:
    a. Access the pseudo-inverse authority matrix, $\mathbf{M}^+$, from memory
    b. Select desired actuation mode (nominal, precise, fuel-efficient, agile)
    c. Perform the associated thruster selection algorithm for this mode
    d. Command the thrusters to open for the calculated firing times
    e. Repeat for all commanded forces and torques until reconfiguration
6. Return to Step 1 upon spacecraft reconfiguration

Chapter 4

Thruster Selection Logic Simulation

4.1 Simulation Development

For verification and validation of the methods presented in Chapter 3, a simulation was developed to ensure that the reconfigurable mixer techniques could be implemented effectively and the actuation modes' performances could be compared. As the first requirement, the simulation must be able to check that the calculated thruster firing times are indeed producing the commanded forces and torques sent to the mixer. This verification will prove that the thruster selection logic is working as a baseline and that the algorithms will command the spacecraft appropriately. In addition, the performance of the mixer in terms of fuel consumption, minimum and maximum forces and torques produced, error, and computation time must be tracked. Bookkeeping these parameters will allow evaluation of the particular mixer's performance and comparison of the four actuation modes described in Chapter 3. Finally, the simulation must be able to assess the performance of the algorithms with several aggregative spacecraft systems that could have any number of thrusters in any location with respect to the center of mass.

One method of achieving these requirements would be to simulate the 6-DOF dynamics of a spacecraft system on orbit including a controller and estimator. Although

a high fidelity simulation of this nature would produce useful results and prove implementation of the thruster selection algorithms in a complete spacecraft ADC system, it will not directly assess the performance of the mixer. In this type of simulation, the effects of the system would be measured in terms of control performance most likely. This control performance more heavily relies on the controller used, the estimator and quite a bit of uncertainty employed in the model. The outputs would likely be performance of a position hold maneuver or performance to a step response. Again, although very useful, a higher fidelity, isolated test of the thruster selection algorithms is necessary and can actually be much simpler than implementing an entire dynamics simulation.

All that is needed to achieve the goals outlined in the first paragraph above is to develop a simulation that can take a vector of forces and torques as input and return the commanded firing times for a particular system. This process can be repeated for a profile of commanded force and torque vectors to assess the performance of the algorithm for a variety of control inputs. Of course, to check the performance of the algorithms on a variety of aggregative systems, the properties of the spacecraft being tested must be easily modifiable.

Therefore, a simulation as described was developed in MATLAB® that could easily interface with the reconfigurable mixer code in Appendix C. In the center of the simulation, a vector of commanded forces and torques is input into the reconfigurable mixer along with a business card (detailed in Section 3.1) of the system undergoing testing and a variable corresponding to which actuation mode is desired. The reconfigurable mixer returns the vector of firing times computed by the specified algorithm. This vector is summed to determine the total propellant consumption for that execution in a results tracking module. The vector is also left multiplied by the pseudoinverse authority matrix to compute the actual forces and torques applied to the system in a verification module. These applied forces and torques are compared to the commanded and an error is determined. This process is surrounded in two loops. The inner loop runs through a profile of commanded force and torque vectors and the outer through the four actuation modes. The performance of each of the ac-

tuation modes is tracked: propellant consumption; minimum and maximum x, y and z force applied; minimum and maximum x, y and z torque applied; the average error magnitude between commanded and applied vectors; and the average computation time. This simulation can be run for a specific spacecraft system by modifying the business card used. This process is described in Figure 4-1.

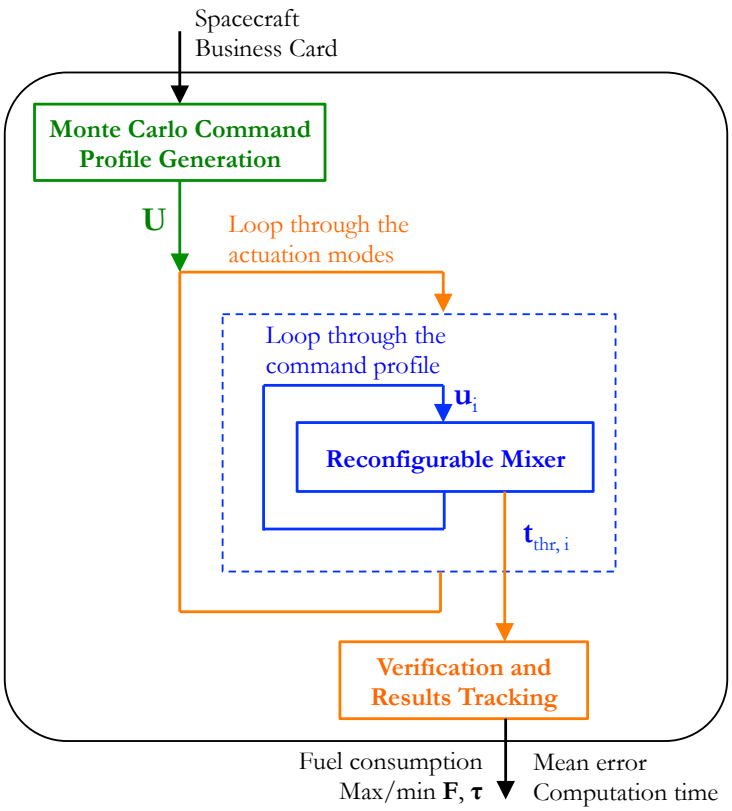


Figure 4-1: Flow diagram of the Monte Carlo simulation as implemented in MATLAB®

To simulate an accurate representation of the expected commanded forces and torques in on-orbit operations, a Monte Carlo approach was implemented. The exact behavior of an aggregative spacecraft system on orbit is not well know, but for the purpose of this simulation it is assumed that the controller will be following a set of waypoints that have been determined by a path planner. Typically, these waypoints will command the spacecraft to attempt to position hold at a particular location specified by the waypoint. This behavior is similar to sending step inputs to the control system and requesting the controller to command forces and torques to achieve the

new system state. Primarily these step inputs will be in one direction or along one axis. Rotating the spacecraft to an orientation that aligns the thrusters in a particular direction and then actuating in that direction will reduce the fuel consumption associated with firing opposing thruster against each other. Thus, a command profile is implemented in this simulation that excites all forces and torque initially, then individually excites the forces and torques of each axis, and finally has a position holding phase. This behavior is described for both 3-DOF and 6-DOF in Figure 4-2.

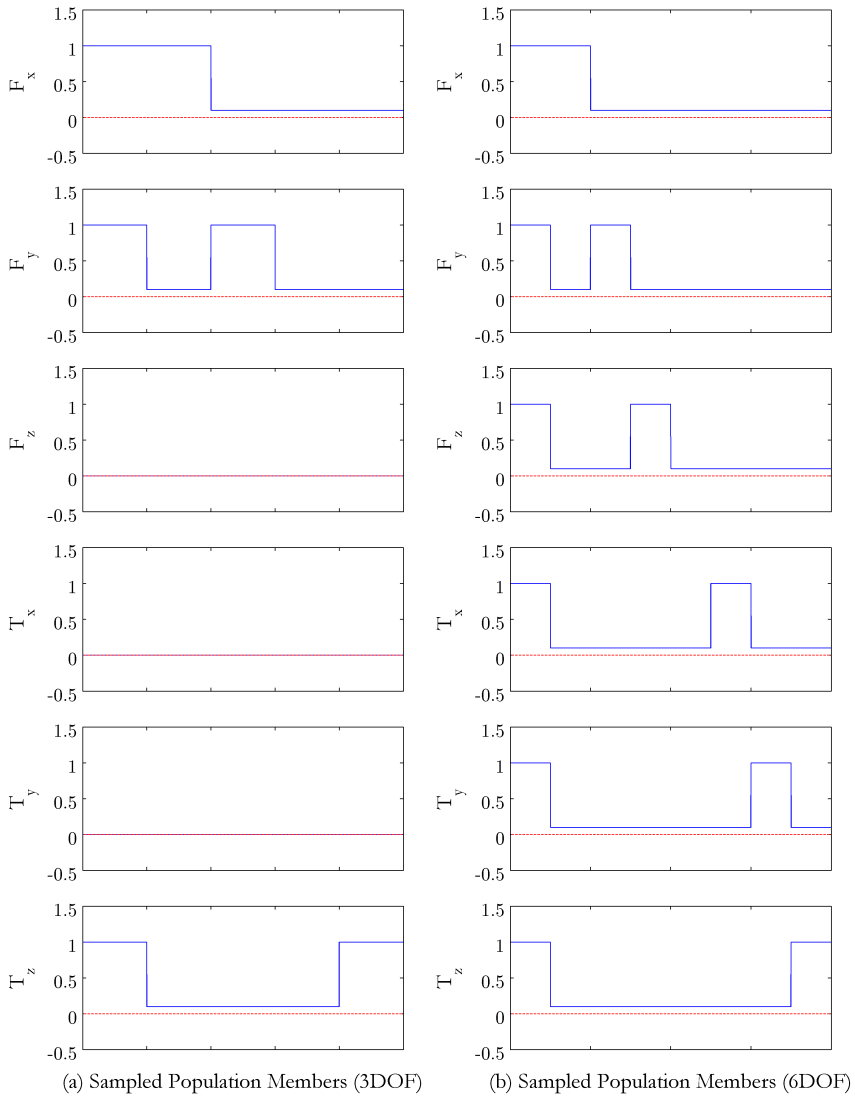


Figure 4-2: Scaling factors across Monte Carlo population to simulate the command profile

When a specific force or torque is excited—represented by a value of 1.0 in Figure 4-2—the Monte Carlo command generator selects from a normal distribution using the maximum possible force and torques as a reference for the standard deviation. This distribution is shown in the darker color in Figures 4-3 and 4-4. When a specific force or torque is not excited—represented by a value of 0.05 in Figure 4-2—the Monte Carlo command generator selects from a normal distribution using 5% of the maximum force and torques as reference for the standard deviation. This narrower distribution is shown in the lighter color in Figures 4-3 and 4-4.

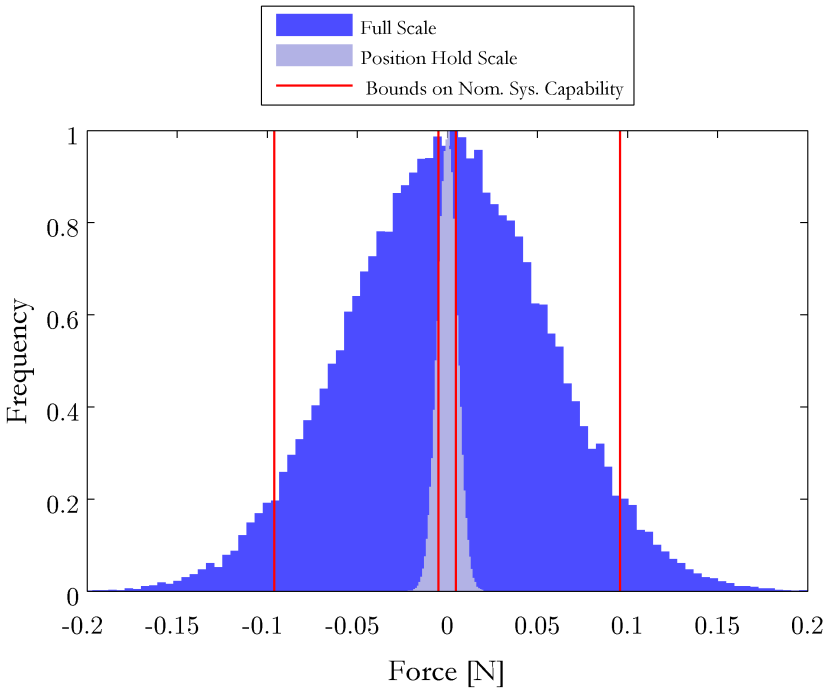


Figure 4-3: Sampled normal distributions used in the Monte Carlo force command generator for a single SPHERES satellite

In Figures 4-3 and 4-4, the vertical lines represent the constraints on the nominal system behavior. The outer lines show the maximum possible force and torque that can be produced by the system. The inner lines show the minimum force and torque that can be produced by the spacecraft. Therefore, the area between the inner and outer vertical lines represents the system capable commanded forces, while the area outside that region represents the commands that the system cannot achieve either

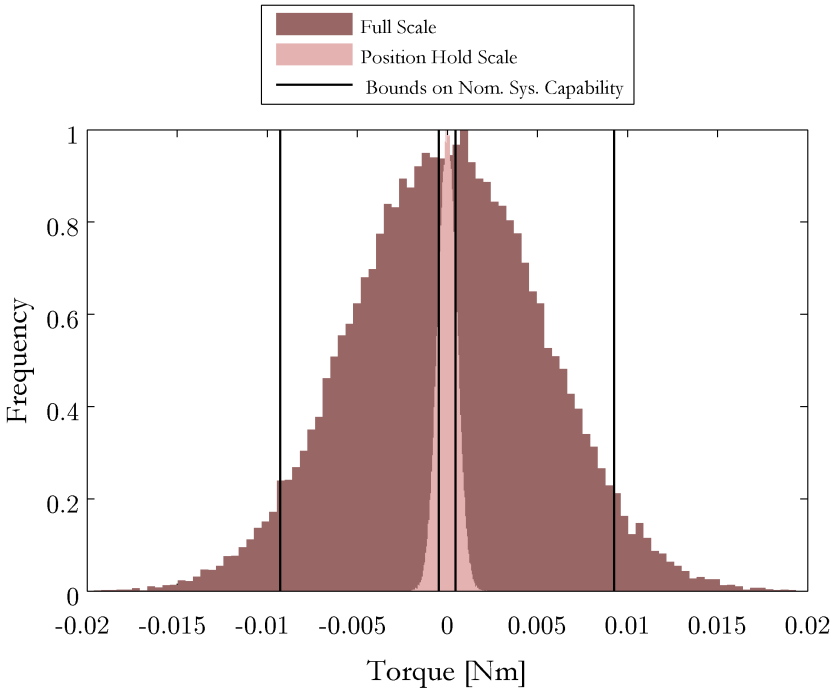


Figure 4-4: Sampled normal distributions used in the Monte Carlo torque command generator for a single SPHERES satellite

because of saturation or because of the thrusters' minimum pulse width. Note that Figures 4-3 and 4-4 represent the distributions used for a standard SPHERES satellite. The maximum effective force achievable on a 400ms duty cycle by the satellite is 0.098 N and the maximum torque is 0.0093 Nm. The minimum effective force producible is 0.0048 N and the torque is 0.00046 Nm.

This commanded force and torque distribution, $\mathbf{U}$ in Figure 4-1, is generated randomly from the same seed in MATLAB® to ensure that the performance is the same through multiple runs and comparisons can be made between runs on different spacecraft systems. The number of command vectors, $\mathbf{u}_i$ in Figure 4-1, can be selected based on the desired fidelity. This number should be very large for the Monte Carlo process to be effective. Typical values used through the testing were 100,000 or 1,000,000 iterations.

4.2 Simulation Results

The simulation in the previous section needs to be run on several spacecraft configurations to verify that the thruster selection algorithms from Chapter 3 are functional across many aggregative spacecraft system architectures. The computed firing times for a full profile of commanded forces and torques need to be accurate for the algorithms to be verified. This step is performed in the verification and results tracking module of the simulation in Figure 4-1. In addition, it would be interesting to compare the different actuation modes' performance on a variety of different spacecraft systems.

The simulation was run for six spacecraft configurations that relate to the different testbeds discussed in Section 1.3. First, a single SPHERES satellite in 3 DOF is run. This is compared to an aggregative system using two docked SPHERES satellites in 3 DOF. These two configurations are what could be tested on the MIT SSL's Flat Floor facility, as only 3-DOF dynamics are available for testing. In addition, to permit a full 6-DOF dynamics, the same single SPHERES satellite and two docked SPHERES satellites were run in the simulation in 6 DOF. This testing should more accurately show how a standard satellite would behave in micro-gravity. Confirmation of this testing could be performed during an ISS test session, but for the purpose of this thesis, only the simulation is performed. Each SPHERES satellite has 12 thrusters configured such that there can be 6 thruster pairs through the combination of the positive and negative direction thrusters across from each other. This SPHERES satellite system, alone, is fully controllable, but is not overdetermined, so there is no opportunity for the fuel-efficient and agile actuation modes to benefit. Regardless, it is important to verify the actuation modes still work when the authority matrix is not overdetermined. Thus, when two SPHERES satellites are docked, the authority matrix is overdetermined, because there are now 24 thrusters.

The other two configurations come from the 3-DOF ARMADAS testbed. One of the configurations is a single SWARM unit, containing a SPHERES satellite and the SWARM propulsion carriage. This configuration has 28 thrusters at a variety of

moment arms from the center of mass. By itself, the SWARM unit has an overdetermined authority matrix; thus, all actuation modes should operate uniquely. The second configuration is two SWARM units docked together. This configuration has 56 thrusters and therefore, with a large number of thrusters, should represent a situation where there are multiple spacecraft docked together. These two configurations were also tested with hardware in the nominal actuation mode as will be discussed in Chapter 5.

These configurations were run through the simulation as described in Section 4.1. The verification module showed that all of the actuation modes effectively produced the correct forces and torques on the system, so we know that their functionality has been confirmed. The remaining verification lies in the performance of the actuation modes. The following plots, tables and associated analysis will compare the performance of the different actuation modes across several spacecraft configurations. Specifically fuel-efficiency, agility, precision and computation time will be discussed. These are the characteristic properties of each actuation mode, so should result in interesting comparisons. Keep in mind that throughout this analysis, the profile as discussed in Section 4.1 is used. If the profile were to differ to be more conservative in motion or require much quicker responses, the results will likely be skewed. Instead of performing several analyses for different profiles, the more typical profile was utilized.

Table 4.1: Normalized propellant consumption across the different system configurations and actuation modes

	Fuel Consumption (normalized)			
	Nominal	Precision	Fuel-Efficient	Agile
SPHERE (3DOF)	1.000	2.330	1.000	1.000
Docked SPHERES (3DOF)	1.000	2.376	0.935	1.041
SPHERE (6DOF)	1.000	2.550	1.000	1.000
Docked SPHERES (6DOF)	1.000	2.519	0.957	0.944
SWARM (3DOF)	1.000	2.080	0.953	0.995
Docked SWARM (3DOF)	1.000	2.283	0.870	1.064

The fuel consumption across each actuation mode and each system configuration

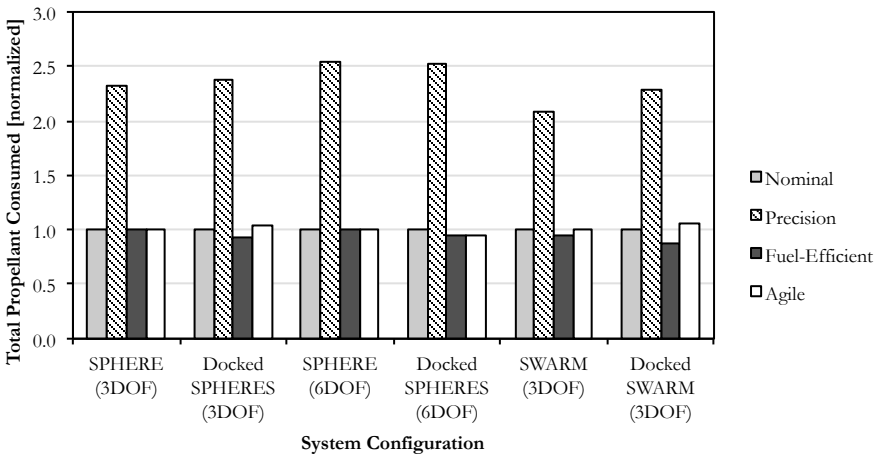


Figure 4-5: Normalized propellant consumption across the different system configurations and actuation modes

is shown in Table 4.1 and Figure 4-5 after normalizing around the nominal actuation mode. The first thing to note here is that for both the single SPHERES satellite cases, the nominal, fuel-efficient and agile actuation modes all have the same performance. This equal performance will be seen in all of the performance metrics except for computation time. The most obvious result from the fuel-efficiency metric is that the precision mode uses more than twice the fuel as the other modes. This fuel-inefficiency is a major deterrent to using the precision mode. The fuel-efficient mode on the other hand reduces the fuel consumption anywhere between 5 to 13% in the different systems. This amount may seem small, but this is essentially free fuel savings—computation time being the only expense—because the system achieves the same forces and torques the controller is commanding for less fuel. Furthermore, reducing the launch mass of propellant by 5 to 13% would greatly interest most launch providers and satellite manufacturers. The agile mode hovers around the same fuel efficiency as the nominal case, although in relevant cases predominately uses about 4 to 6% more fuel.

Displayed in Table 4.2 and Figure 4-6 is the mean error magnitude over the entire command profile. This error is normalized around the nominal case as the fuel consumption was. Again, the single SPHERES satellite cases have the same precision across the nominal, fuel-efficient and agile modes as expected. Interestingly, the

Table 4.2: Commanded vs. applied error magnitude across the different system configurations and actuation modes

	Error Vector Magnitude (normalized)			
	Nominal	Precision	Fuel-Efficient	Agile
SPHERE (3DOF)	1.000	0.316	1.000	1.000
Docked SPHERES (3DOF)	1.000	0.368	1.073	0.974
SPHERE (6DOF)	1.000	0.329	1.000	1.000
Docked SPHERES (6DOF)	1.000	0.398	1.000	0.980
SWARM (3DOF)	1.000	0.389	1.028	1.031
Docked SWARM (3DOF)	1.000	0.399	1.061	0.986

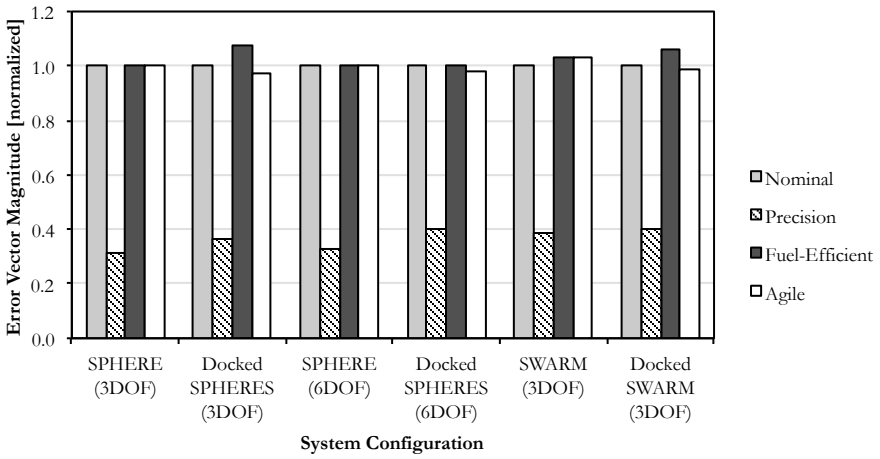


Figure 4-6: Commanded vs. applied error magnitude across the different system configurations and actuation modes

precision actuation mode performs anywhere from 2.5 to 3 times as precisely as the other actuation modes. Having one-third the error is mainly due to the number of commands that are issued below the minimum threshold for the thrusters to produce. Figures 4-3 and 4-4 give a good idea of what percentage of commands this would be. There are also slight differences between the precision in the fuel-efficient and agile actuation modes. The fuel-efficient mode tends to have higher error than nominal and the agile mode less error than nominal. This behavior is likely due to the error produced when saturating commands are issued. The fuel-efficient mode will saturate before the nominal and the agile mode will saturate later than the nominal. Thus, the agile mode should have less error as it more closely represents these over-saturated commands.

Table 4.3: Maximum force applied on each axis across the different system configurations and actuation modes

	Maximum Force Applied [N] (equal for all axes)			
	Nominal	Precision	Fuel-Efficient	Agile
SPHERE (3DOF)	0.096	0.096	0.096	0.096
Docked SPHERES (3DOF)	0.192	0.192	0.192	0.192
SPHERE (6DOF)	0.096	0.096	0.096	0.096
Docked SPHERES (6DOF)	0.192	0.192	0.192	0.192
SWARM (3DOF)	0.288	0.288	0.288	0.288
Docked SWARM (3DOF)	0.576	0.576	0.576	0.576

Table 4.4: Maximum torque applied on the z-axis across the different system configurations and actuation modes

	Maximum Torque Applied [Nm]			
	Nominal	Precision	Fuel-Efficient	Agile
SPHERE (3DOF)	0.009	0.009	0.009	0.009
Docked SPHERES (3DOF)	0.066	0.066	0.048	0.077
SPHERE (6DOF)	0.009	0.009	0.009	0.009
Docked SPHERES (6DOF)	0.066	0.066	0.048	0.077
SWARM (3DOF)	0.047	0.047	0.038	0.041
Docked SWARM (3DOF)	0.137	0.137	0.080	0.162

Figure 4-7 and Table 4.4 show the maximum achieved torque on the system throughout the full simulation. Table 4.3 shows the maximum achieved force on the system throughout the full simulation. There is no difference in commanded force output from any of the systems across the actuation modes in Table 4.3, however the maximum force achieved does show the benefit of aggregating spacecraft systems to collectively improve actuation capabilities. Both the single SPHERES satellite cases achieve very small maximum z-axis torques, but again are uniform over all actuation modes. Also, note that because the precision mode is implemented on top of the nominal mode, the torque behavior is identical. As desired, the agile mode produces up to 15% more z-axis torque than the nominal case. This behavior is very desirable if the quickest possible response to control inputs is needed by the system due to response time requirements. As a side effect of the fuel savings, the fuel-efficient mode achieves as much as a 30 to 40% decrease in maximum z-axis torque. The single SWARM unit scenario exposes a weak point in the agile actuation mode. If operating with multiple

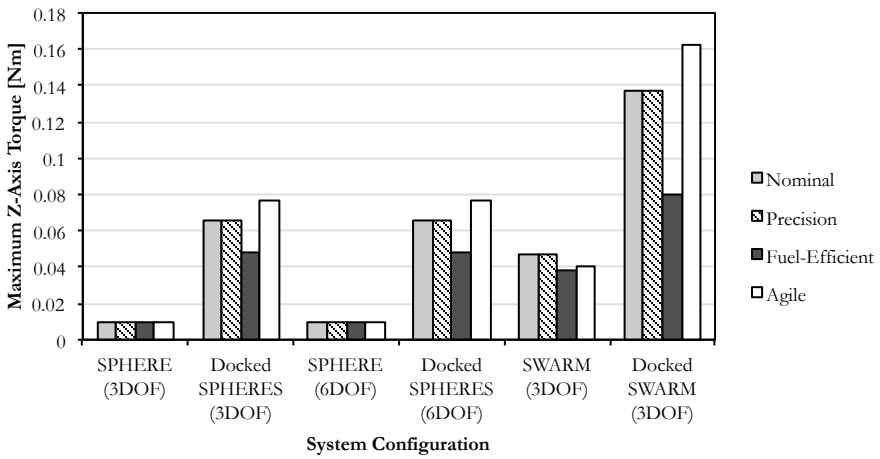


Figure 4-7: Maximum torque applied on the z-axis across the different system configurations and actuation modes

off-axis thrusters, as is the case with a SWARM unit, the agile mode can produce less than the desired torques on the system. This problem is alleviated in a 6-DOF environment as the torque groups are easier to form when there is more information about multiple force and torque axes available for the algorithm to group thrusters. Another interesting point that shows the benefits of aggregative spacecraft system is the dramatic increase in achievable maximum torques by combing two identical systems. As much as about a 400% increase was achieved in the single SWARM unit to two-docked SWARM units upgrade.

Table 4.5: Normalized average algorithm computation time across the different system configurations and actuation modes

	Average Computation Time (normalized)			
	Nominal	Precision	Fuel-Efficient	Agile
SPHERE (3DOF)	1.000	0.939	31.678	1.337
Docked SPHERES (3DOF)	1.000	0.971	41.997	1.667
SPHERE (6DOF)	1.000	1.049	30.855	1.457
Docked SPHERES (6DOF)	1.000	1.232	42.957	1.826
SWARM (3DOF)	1.000	1.045	50.353	1.848
Docked SWARM (3DOF)	1.000	0.997	60.772	1.681

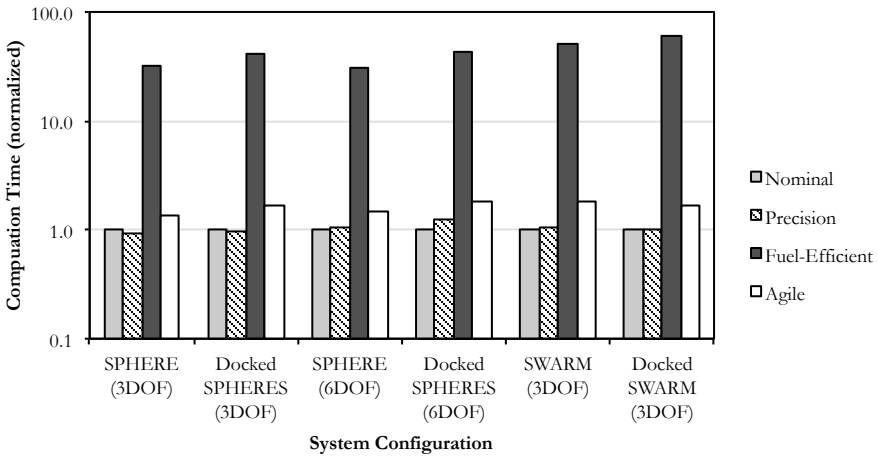


Figure 4-8: Normalized average algorithm computation time across the different system configurations and actuation modes

Figure 4-8 and Table 4.5 show the normalized results of the computation time required to complete these thruster selection algorithms. Note that Figure 4-8 has a logarithmic scale on the y-axis. Figure 4-9 and Table 4.6 show the non-normalized computation time results. Note that in Figure 4-9, there are two vertical axes—one for the nominal, precise and agile modes and one for the fuel-efficient mode alone. The first thing noticed in these figures and tables is that the fuel-efficient mode uses substantially more computation time. This additional computation time is due to the optimization problem being solved as opposed to a few simple matrix multiplications and a solution to $Ax = b$. Surprisingly though, this difference results in an increase of almost two orders or magnitude in computation time. As expected, the agile mode always takes more computation time because there are many more steps and lines of code in the agile mode calculation. Also, as expected, there is a slight increase in computation time when the precision mode is used, because of the small amount of additional code required to be added to the nominal mode's code to implement the precision mode. Figure 4-9 nicely shows how the computation time increases roughly proportional the number of thrusters on the spacecraft.

Each of the actuation modes excels for their specific metrics as desired and tailored to achieve. However, they also perform very weakly in other metrics, sometimes much more so than expected. The nominal mode is fairly average as expected. The

Table 4.6: Average algorithm computation time across the different system configurations and actuation modes

	Average Computation Time (ms)			
	Nominal	Precision	Fuel-Efficient	Agile
SPHERE (3DOF)	0.358	0.336	11.329	0.478
Docked SPHERES (3DOF)	0.505	0.490	21.214	0.842
SPHERE (6DOF)	0.341	0.358	10.528	0.497
Docked SPHERES (6DOF)	0.497	0.613	21.361	0.908
SWARM (3DOF)	0.519	0.542	26.112	0.958
Docked SWARM (3DOF)	1.117	1.114	67.893	1.878

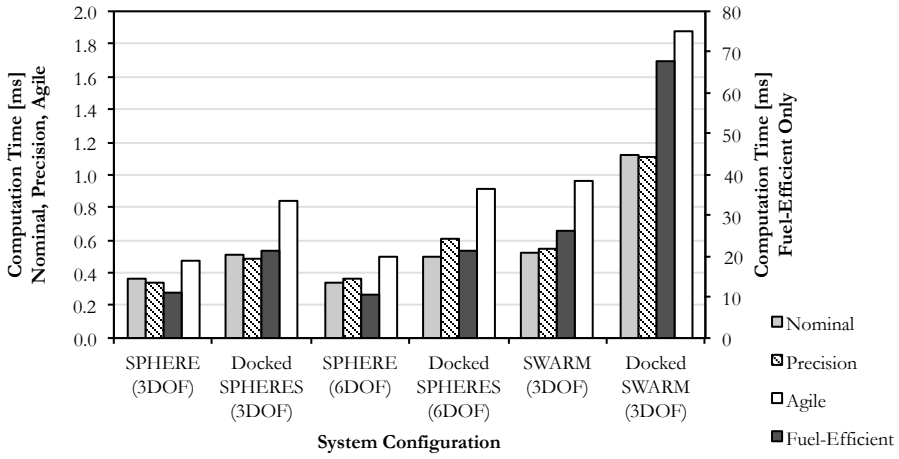


Figure 4-9: Average algorithm computation time across the different system configurations and actuation modes

precision mode did achieve a large improvement in the precision of the system, but drastically increased fuel consumption. The fuel-efficient mode saved a fair amount of propellant, yet computationally was extremely expensive and also reduced the total capable torque by a fairly large margin. Finally, the agile mode improved the agility of the system effectively, but worsened the other metrics slightly. Overall, the actuation mode should be tailored to the specific application and selected based on the desired characteristics of the system. The nominal mode seems to split the difference on a lot of performance metrics and thus could be the most desirable actuation mode for a general system.

4.3 Validation through the SPHERES Simulation

As discussed in Section 4.2, it was necessary to develop a specific simulation solely for the purpose of verifying and testing reconfigurable thruster selection algorithms. However, it is necessary to validate the simulation in some manner. Based on the spacecraft configurations used in Section 4.2, an efficient way to validate the Monte Carlo simulation would be to compare the results to the already developed, tested and validated SPHERES Simulation mainly completed by Jake Katz [17]. The SPHERES Simulation contains a very high fidelity dynamics model of the SPHERES satellites as well as communication interfaces, global estimation, time synching and pretty close to everything represented in the actual testbed on the ISS. Of particular interest to the thruster selection algorithm problem, the transient behavior of the thrusters opening is well modeled in the simulation, so that the minimum on-time behavior is well represented. The SPHERES Simulation is written in MATLAB® and Simulink®, although incorporates the SPHERES software through MATLAB® Executable (MEX) files and CMake compilation. Thus, the exact C code that is written to be bootloaded to the SPHERES satellites can be run through the SPHERES Simulation. The top-level Simulink® block diagram is shown in Figure 4-10. A screenshot of the visualization portion of the simulation is shown in Figure 4-11.

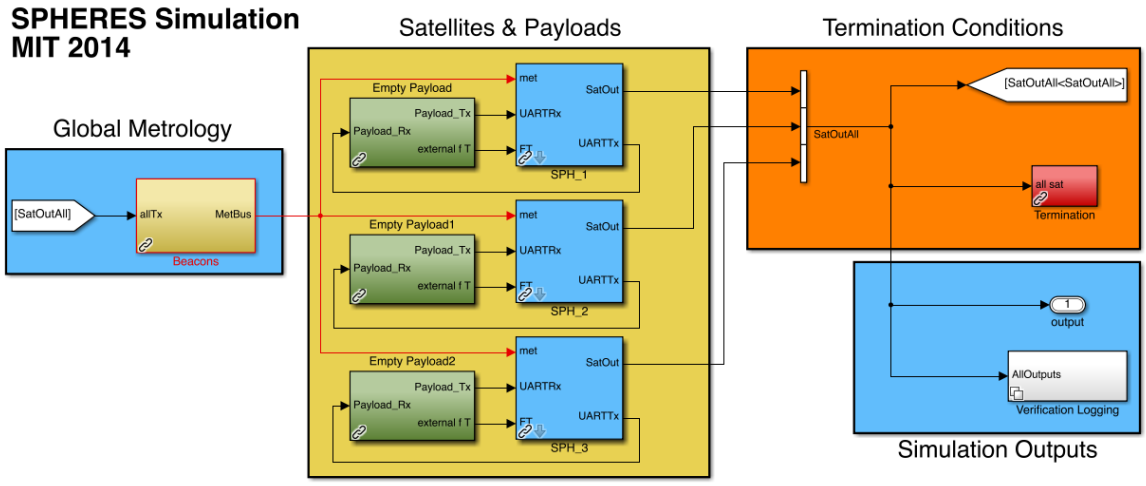


Figure 4-10: Top-level Simulink block diagram of the SPHERES Simulation

Unfortunately, at this point, a version of the simulation has not be developed to

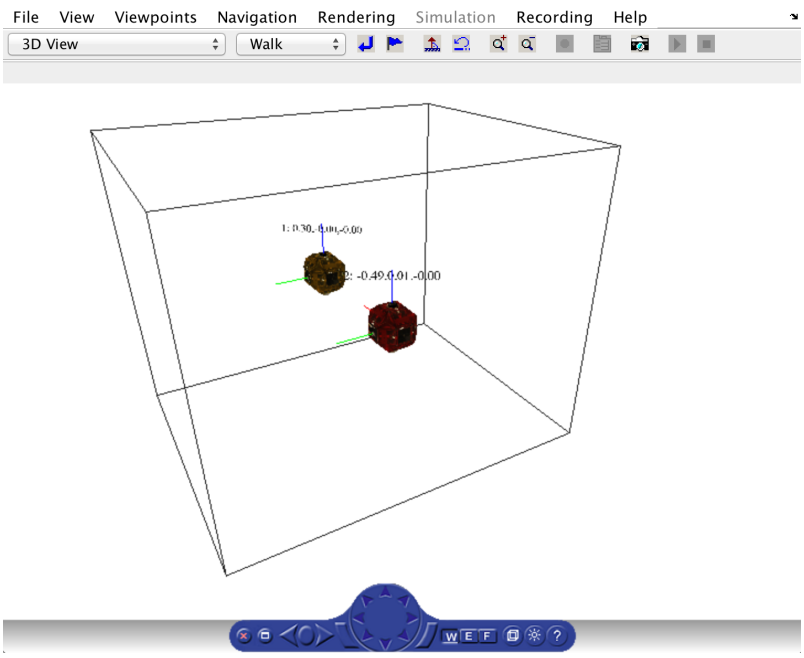


Figure 4-11: Screenshot of the SPHERES Simulation visualization

be able to control two docked SPHERES satellites. However, there remains room to validate the simulation in Section 4.2 as the precision mode can be implemented successfully and can show meaningful results with only one satellite. Among other critical pieces of data, the SPHERES Simulation keeps track of the propellant consumption and state vector of the satellite as it performs maneuvers in 6 DOF.

A SPHERES test project was written to compare fuel-efficiency and error magnitude results between the two simulations. This code includes the standard SPHERES proportional-derivate controller and the standard SPHERES Extended Kalman Filter. The code commands the SPHERES satellite to respond to a step function in the x-direction, a step function in the y-direction and a step function spin about the z-axis. On purpose, this behavior is similar to that shown in Figure 4-2 that was used as a basis for the Monte Carlo simulation. First, the authority matrix that is used in the standard SPHERES software was computed using the reconfigurable mixer code in Appendix C. This authority matrix matched exactly, so a new mixer was not written at this point. This standard mixer will be called the nominal mixer from this point forward. Next, a precise mixer was developed by adding the differential thruster firing technique to the end of the standard SPHERES mixer. The behavior

of the system while using both of these mixers will be compared.

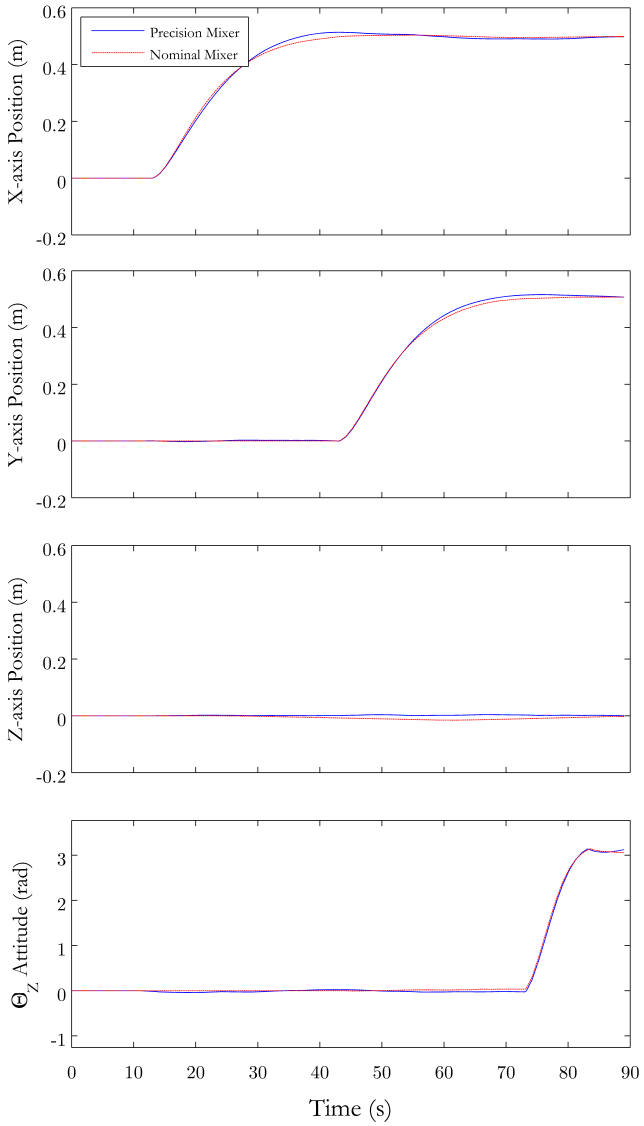


Figure 4-12: Truth position response of both the nominal and precision actuation modes to commanded maneuvers to a SPHERES satellite

The response of the state of the system is shown in Figure 4-12. One can see that the desired behavior of the system was achieved for both the nominal and precise mixer. Thus, adding the precision actuation mode to the SPHERES satellite still qualitatively achieves the same performance. To discuss the difference in behavior of the system, a zoomed in plot of the x-axis response to the step function is shown in Figure 4-13. One can see from this plot that the precise mixer has a quicker settling

time than the nominal mixer. This faster settling time is likely due to the fact that the precision actuation mode is capable of more finely producing forces when the commanded forces are quite small.

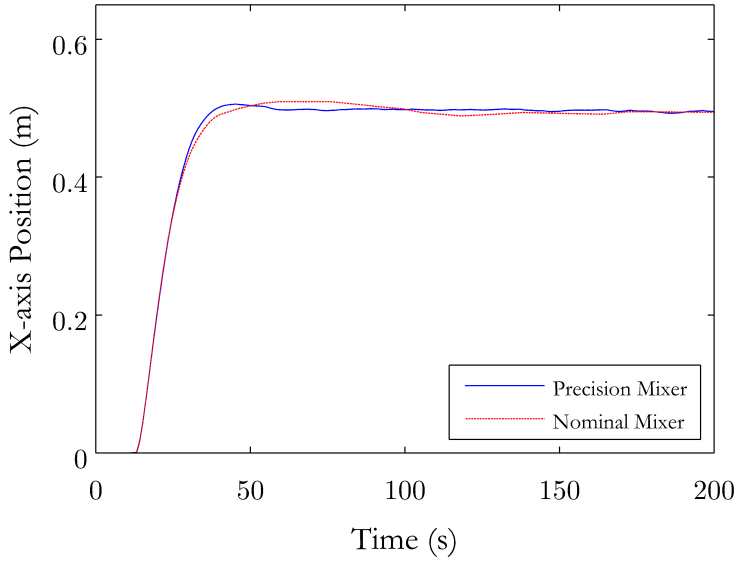


Figure 4-13: Truth position response of both the nominal and precision actuation modes to commanded x-axis position step

Probably the most significant result from the use of the SPHERES Simulation is shown in Figure 4-14. Figure 4-14 compares the precision of the two actuation modes in terms of the deadband in the position hold maneuver for each axis. One can see that the precision actuation mode has much finer control than the nominal mixer in all axes. The deadband is shown by the long sloped sections of the nominal mode compared to the short, almost noisy-looking sections of the precision mode. Of course, the noise of the environment and the estimator are also represented in this data, so it is not a perfect example of how much the deadband should improve, but it does show a very impressive result. Taking the Root Mean Square (RMS) error from the commanded position results in Table 4.8. Also, taking the bounds of the error as a pseudo-deadband measurement is shown in Table 4.7. This shows very similar results to the Monte Carlo simulation actually. As shown in Table 4.8, the RMS error is about 50% of the nominal case compared to the 30 to 40% in the Monte Carlo simulation. The deadband also shrinks to about 30 to 50% when using

the precision actuation mode. The differences between simulation results can likely be attributed to the noise from the estimator, because that seems to be the largest source and was not considered in the Monte Carlo simulation. This estimator noise could be increasing the perceived error in the SPHERES Simulation. Thus, as far as the error magnitude is concerned, the simulations seem to match and the Monte Carlo simulation seems to be validated.

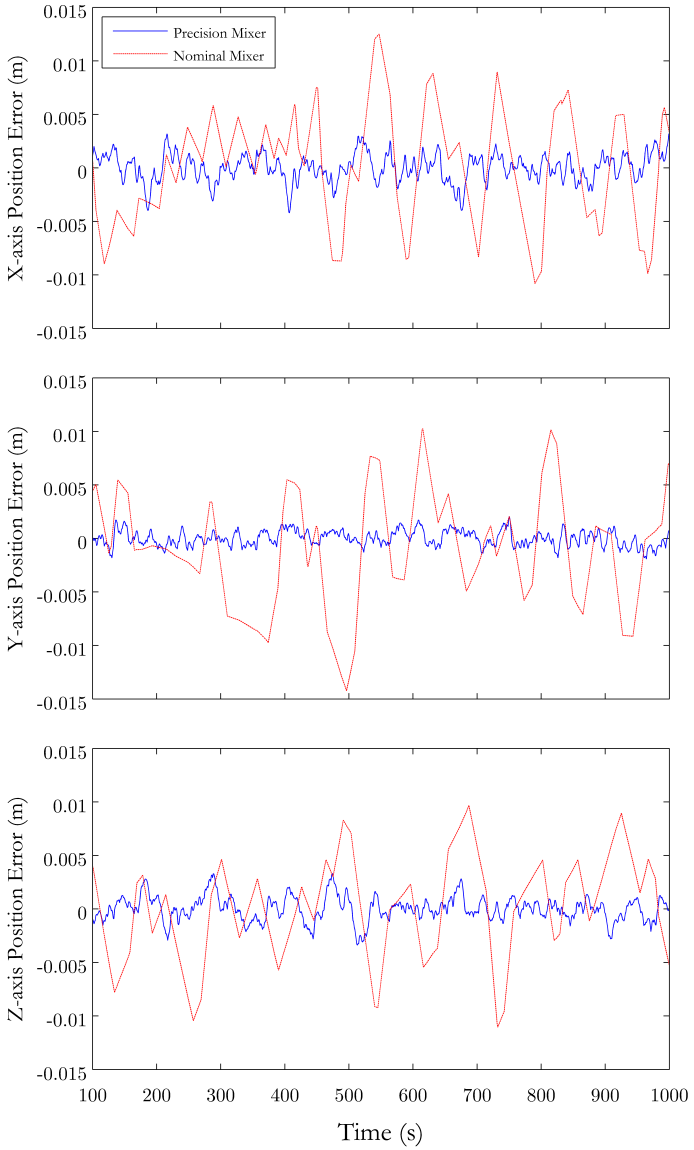


Figure 4-14: Truth position response of both the nominal and precision actuation modes to position hold commands to a SPHERES satellite

Table 4.7: Measured error bounds on the truth position during a position hold maneuver

	Measured Error Bounds (+/-)		
	X Position (m)	Y Position (m)	Z Position (m)
Nominal	0.0123	0.0117	0.0104
Precision	0.0068	0.0025	0.0033

Table 4.8: RMS error measured from truth position during a position hold maneuver

	RMS Error (+/-)					
	Position (m)			Velocity (m/s)		
	X	Y	Z	V_x	V_y	V_z
Nominal	0.0056	0.0053	0.0046	4.08E-04	3.50E-04	3.19E-04
Precision	0.0035	0.0035	0.0023	3.69E-04	2.64E-04	2.89E-04

Figure 4-15 shows the cumulative fuel consumption as time progresses through the test in the SPHERES Simulation. The rate that the precision actuation mode consumes fuel is roughly twice that of the nominal mode. The three large increases in propellant use for both of the curves correspond to the step-response maneuvers being performed. The other sections are the position holding portions. The precision actuation mode uses substantially more fuel during the position hold maneuvers than the nominal actuation mode. This result means that the forces and torques commanded due to the noise in the environment are small enough that the nominal actuation mode is not able to counteract it, but the precision mode is. The consumption during the first few seconds of the step maneuvers seems to be relatively similar as both mixers are applying the same thruster firing times.

Table 4.9: Normalized propellant consumption over three step function maneuvers

	Normalized Fuel Consumption			
	X Step	Y Step	Z Rotation	Total
Nominal	0.147	0.159	0.114	0.420
Precision	0.382	0.381	0.237	1.000

Table 4.9 compares the propellant consumption of the nominal and precision modes for each of the maneuvers. The precision mode uses about 2.38 times the

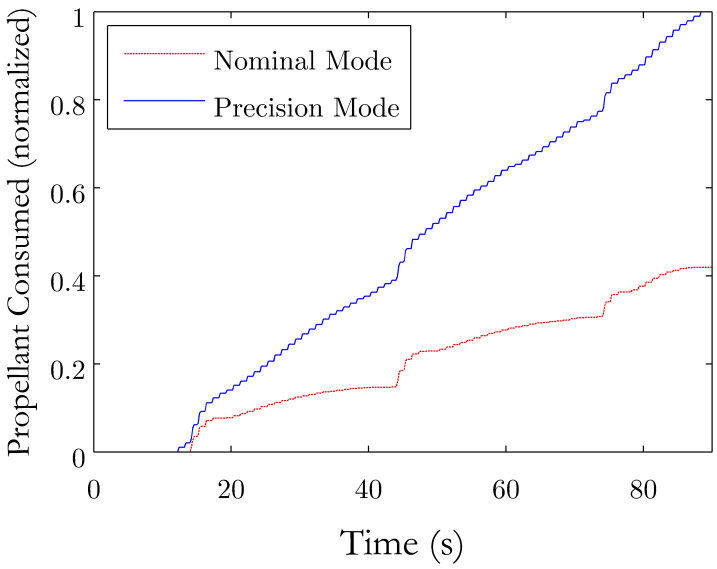


Figure 4-15: Propellant consumption throughout the full set of maneuvers for both the nominal and precision actuation modes

fuel as the nominal actuation mode. This closely matches the fuel-consumption results for the single SPHERE in 6 DOF case in the Monte Carlo simulation, which was 2.55 times the fuel consumption of the nominal case. Thus the simulation results for fuel efficiency also seem to match. This means that both the fuel-consumption and error reporting of the simulation have been validated against the SPHERES Simulation successfully. From this we must make the assumption that the other results from the Monte Carlo simulation are also valid, although there could be further validation to be completely confident.

4.4 Summary

This chapter discusses a Monte Carlo simulation that can be used to thoroughly test the effectiveness of thruster selection logic algorithms and compare the results from the different actuation modes. Briefly, the Monte Carlo simulation sends a profile of commanded forces and torques to the thruster selection algorithms and tracks the applied forces and torques that the algorithms output. This commanded profile is representative of that which would be commanded in an on-orbit assembly and

servicing scenario. The simulation was executed for several spacecraft configurations, with different numbers, locations and directions of thrusters. The performance with each of the actuation modes was tracked. The results were validated against the SPHERES Simulation for the nominal and precision modes and can be extended to the rest of the Monte Carlo simulation as the same principles are used throughout.

The results from the Monte Carlo simulation show trends in the expected directions for each of the actuation modes. The precision mode is able to improve the error of the system by more than a factor of two while only increasing the computation time slightly, but it uses about twice as much fuel when compared to the nominal mode. The fuel-efficient mode is able to reduce propellant consumption by as much as 13%, but cannot achieve full actuation capabilities and takes an order of magnitude more computation time. Finally, the agile mode is able to apply up to 15% higher torques on the system, but increases computation time moderately and consumes about 5% more fuel. There are several tradeoffs here, so the actuation mode should be chosen based on the desired system performance. The nominal actuation mode acts as a hybrid of the different modes and uses the least computation time. Therefore, the nominal mode may be useful as a general selection, when no area of high performance is specifically required.

Chapter 5

Hardware Testing

The hardware testing performed for this thesis, both in regards to the Phoenix project and the ARMADAS project, was accomplished with the help of Sternberg and McCarthy. Therefore, the theses of Sternberg [36] and McCarthy [24] also detail the concept of operations and results discussed in the next sections. Sternberg uses the results in the context of iterative and incremental testing in the purpose of reducing risk toward a robotic assembly and servicing mission. McCarthy specifically discusses the implementation of the RARC and DAC schemes on these testbeds and the relevant performance of those control algorithms. The takeaways from the hardware testing discussed in this thesis, however, are particularly drawn from the implementation and performance of reconfigurable thruster selection algorithms.

5.1 Phoenix Testing

5.1.1 Concept of Operations

The Phoenix testbed, as described in Section 1.3.3, is used to test a set of CONOPS relevant to the DARPA Phoenix program. The CONOPS of the Phoenix testing on the MIT Flat Floor Facility closely follows that shown in Figure 1-6. In the DARPA Phoenix demonstration mission, a larger servicer/tender spacecraft aims to detach and repurpose a functional communications antenna from a defunct satellite.

After removing the antenna, the servicer/tender spacecraft, shown in Figure 1-4, will place small propulsion spacecraft modules around the perimeter of the antenna at known locations. The servicer/tender will also centrally place a more computationally well-equipped module to issue commands to the propulsion modules in addition to performing any data transmission and receiving tasks related to the antenna. After the servicer/tender spacecraft has placed all of the components around the antenna, it will release the aggregated assembly to function as a self-sufficient communications satellite. This system is a perfect example of an aggregative spacecraft system as multiple spacecraft are jointly controlling a large space structure that could not be controlled by one of them individually.

There are several areas of these CONOPS that require significant research and development before implementation on orbit. Specifically, the Phoenix testing at MIT focuses on researching the joint control of the antenna aperture with multiple spacecraft. The SPHERES satellites and SWARM carriages are combined with a few additional structural elements to construct the Phoenix testbed described in further detail in Section 1.3.3. This testbed allows two-satellite control of a structure analogous to an antenna aperture. The test configuration is shown in Figure 5-1. The two maneuvers that are of interest in this testing are a pitch-slew maneuver starting and ending at rest and a rate-dampening maneuver starting at a given tip-off speed and ending at rest. Both maneuvers are expected to be directly traceable to the CONOPS of the DARPA Phoenix demonstration. As there is only one axis of rotation available in ground testing, the system will only be tested in 1 DOF. This degree of freedom represents the pitch rotation axis of the spacecraft.

It is important that these maneuvers are robust to uncertainty in the physical parameters of the system, as the repurposed antenna aperture may have changed mass properties through physical degradation on orbit during its previous years of operations. It is not likely that the manufacturing drawings and CAD will be precisely representative of the current antenna. To this end, the proof mass, M in Figure 5-1, can be positioned in multiple configurations. The idea is that the proof mass is positioned in an unknown location to the control algorithms as to simulate possible

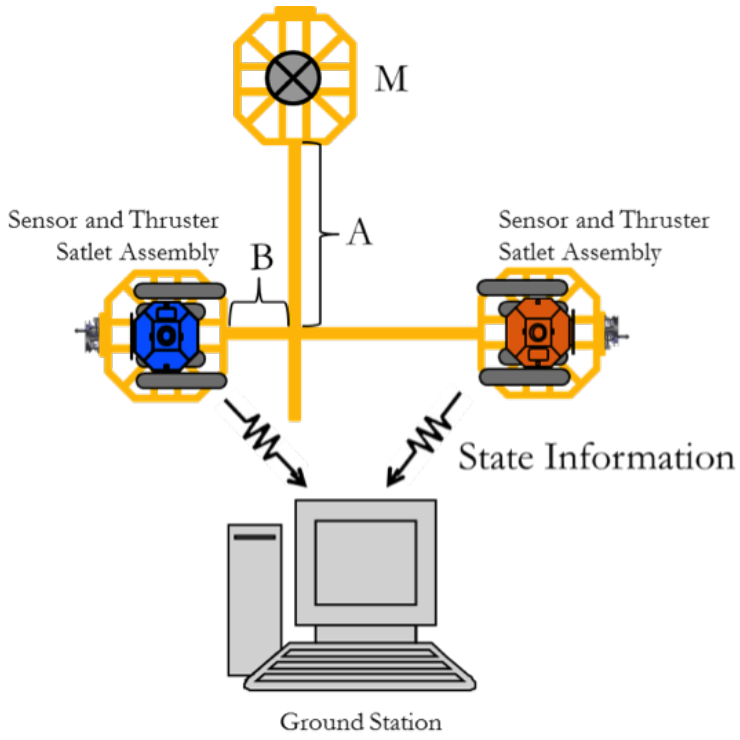


Figure 5-1: The CONOPS for the Phoenix testing on the MIT Flat Floor Facility [35]

uncertainty in the mass properties of the system. The system will be tested with the proof mass in multiple locations to ensure appropriate response. The representable inertia properties of the system are meant to be scaled versions of those of the on-orbit system.

In this CONOPS, it is considered that at the very least the spacecraft know which side of the center of mass of the system they are located. They do not know the precise distance to the center of mass, but have an initial guess that will allow for the correct forces and torques on the system. A Direct Adaptive Control (DAC) scheme is used to account for this uncertainty. In this method, a reference input is tracked and the commanded forces and torque are relatively small as to anticipate the reference input and conserve fuel. The error to the reference input is tracked and the control system adapts to the actuation properties of the spacecraft as time progresses. Thus, there is no need for a highly precise model of the system from the control systems standpoint.

There are 28 thrusters per spacecraft in this testbed, making a total of 56 thrusters.

These thrusters will each have different and uncertain locations with respect to the center of mass of the system. To account for this uncertainty only two possible baseline authority matrices are formed throughout the process. One matrix represents the case where the center of mass is far outside the thruster envelope—closer to the proof mass than to the satellites. The other configuration is when the center of mass is within the thruster envelope—closer to being centered between the two satellites. These two authority matrices are used with the nominal actuation mode, because there is a desire for both fuel-efficiency and maximum actuation torques. Based on the known configuration, the pseudoinverse authority matrices are computed with the assumed business cards for the associated configuration. The nominal actuation mode solves the $Ax = b$ problem with Cholesky factorization to determine the thruster firing times given the controller's commanded torques. The DAC algorithms should account for any minor errors between the applied and commanded torques that will be inherent in configurations with asymmetric centers of mass.

5.1.2 Results

Overall, the results from the Phoenix testing were mildly successful from the traceability viewpoint. The DARPA Phoenix program will benefit from the testing performed on the MIT Flat Floor, but the testing was not exhaustive enough to reduce any risk for their on-orbit demonstration. There were many problems that arose during the testing that prevented excellent performance. Nevertheless, the reconfigurable thruster selection logic used in this testing was verified to work effectively. The nominal actuation mode was implemented in hardware successfully and the appropriate thrusters fired when commanded. Thus, as a test of the reconfigurable thruster selection algorithms alone, the Phoenix testing was very valuable.

The foremost problem with the testing was the friction present on the epoxy floor. The air carriages did not float as effectively as desired on the surface and there was substantial friction present that assisted in slowing the spacecraft to a stop. This friction skewed the results to be slightly better for both the maneuvers. In the slew maneuver, the settling time was smaller than it likely would have been without the

friction, as the friction acted as a resistive force that trapped the air carriage after it slowed to a certain velocity. The effects of friction in the rate dampening maneuver are, of course, that the system stopped rotating sooner than if solely resisted by the thrusters. To counteract this friction problem, the tests were first run without a controller and with no thruster firings. With this data, the amount of friction on the floor could be characterized for incorporation into the final results. Generally, in an uncontrolled spin maneuver, the system would rotate for more than three or four total revolutions before it would hit the edge of the flat floor and the experiment had to be stopped. In the controlled rate-dampening maneuver, the spacecraft would stop in less than a full rotation. Thus, the DAC algorithms were successful, but the performance could not be precisely determined.

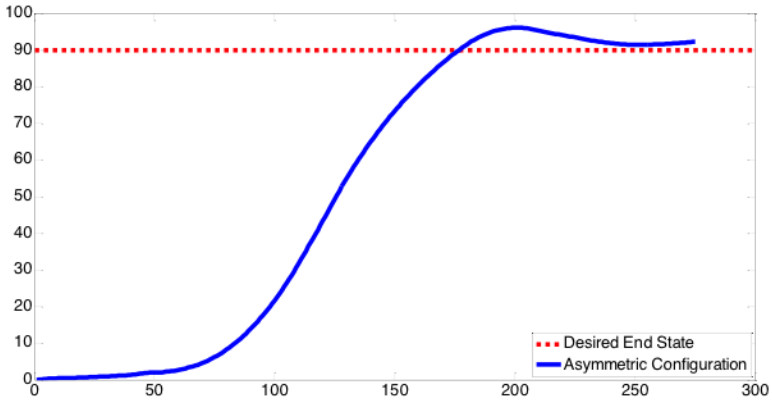


Figure 5-2: Response to a 90-degree z-axis rotation slew step command [35]

Figure 5-2 shows the response to the slew maneuver. The system is able to effectively perform a 90-degree pitch rotation as commanded. Figure 5-3 shows the response of the DAC algorithm to the rate-dampening maneuver in two proof mass configurations: one configuration with the proof mass fully extended and one with it fully contracted. Despite the friction and the uncertainty in the mass configuration, the system response still tracks the input relatively well.

The takeaway from this testing as relevant to this thesis is that a reconfigurable mixer was implemented in hardware successfully. The control algorithms were able to control the system effectively with these reconfigured mixers. At the very least, this

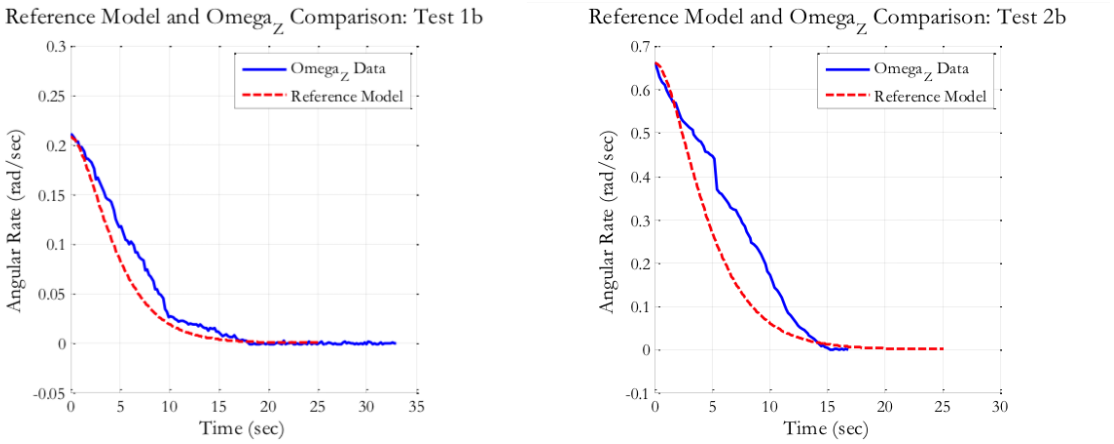


Figure 5-3: DAC response to rate-dampening maneuver with initial tip-off angular velocity in two configurations: (left) proof mass fully contracted (right) proof mass fully extended [35]

proves that the mixer can work in a configuration where the center of mass is not centered in the thruster envelope, as is the case with a SPHERES satellite. This testing suggests that the methods presented in this thesis are able to successfully reconfigure thruster selection logic in hardware to the level that it can interface effectively with control algorithms.

5.2 ARMADAS Testing

5.2.1 Concept of Operations

The ARMADAS testbed is detailed in Section 1.3.2 and consists of two SPHERES satellites and two SWARM carriages. The combination of a SWARM carriage and a SPHERES satellite, a SWARM unit represents a mobile spacecraft with its own sensing and propulsion systems. The CONOPS of the ARMADAS program is to demonstrate a scenario on the ground SPHERES facility at MIT that is traceable to a full, on-orbit servicing mission. The overall idea is that during on-orbit servicing of satellites, there will be times when multiple spacecraft must dock together and perform tasks while docked. Control algorithms need to be developed to update once the system has docked, or aggregated. The control algorithms must be able to control individual modules in addition to the docked system. Resource Aggregated Recon-

figurable Control (RARC) algorithms accomplish just this. Once the modules dock together, the gains of the controllers are updated through online model calculation using only information in the modules' business cards. The ARMADAS testbed aims to demonstrate that RARC algorithms can be used in on-orbit servicing and assembly missions. In addition, the ARMADAS project looks into model-based path planning and waypoint following, for a servicer spacecraft to navigate around a target satellite.

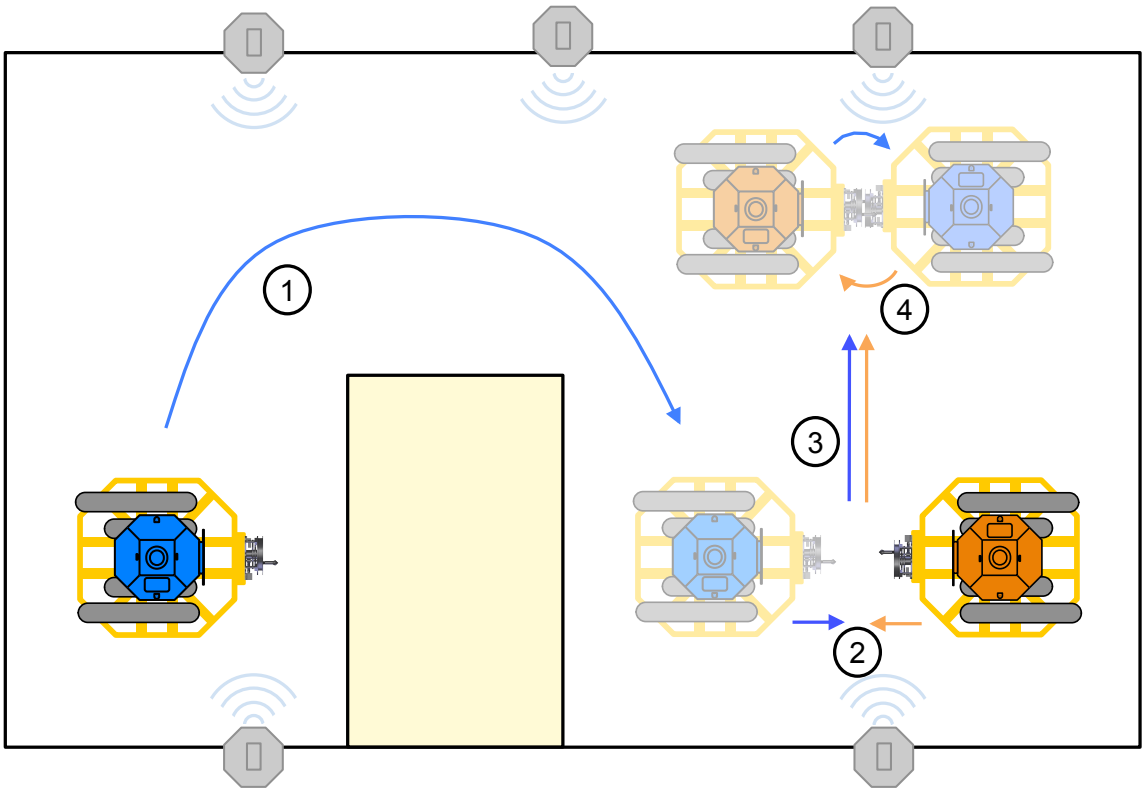
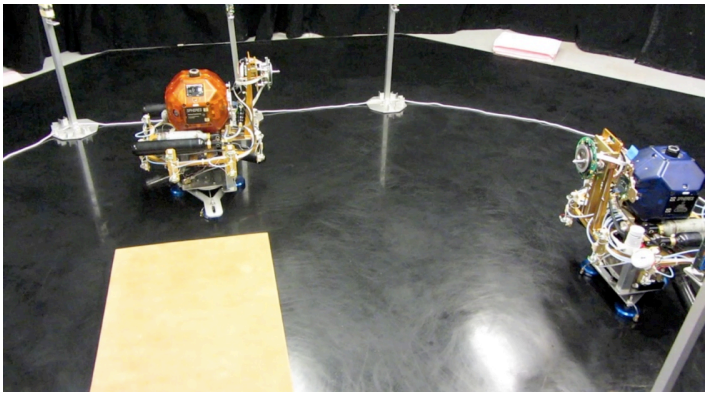


Figure 5-4: The CONOPS of the ARMADAS ground testing on the MIT Flat Floor Facility: Phase 1 through 4 [15]

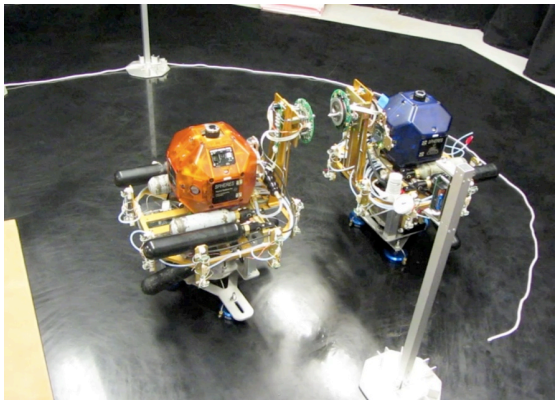
The full servicing scenario used in the ARMADAS testing is defined as pictured in Figure 5-4. In Phase 1, the spacecraft pictured on the left performs a path planning and waypoint following maneuver to navigate around an obstacle. Phase 1 is also pictured in Figure 5-5a. During this maneuver, the satellite is allocating risk along the trajectory and performing a chance-constrained optimization problem. This process is continuous, in a receding-horizon framework, so that the risk allocation is updated based on how much risk the satellite has already absorbed. The satellite is attempting to navigate around an obstacle to a rendezvous location where it can

dock with the secondary satellite. In Phase 2, both of the satellites execute a docking maneuver. Phase 2 is pictured in Figure 5-5b. In this maneuver, a series of software gates are passed if the satellites achieve a certain tolerance of position error. Upon passing all of the tolerance gates, the satellites thrust forward to firmly dock until the docking ports have captured each other. Once the docking ports fully capture and rigidly dock the modules together, the system reconfigures a business card to the aggregated state. From here, the new thruster selection logic is reconfigured and the RARC algorithms compute the new gains. The thruster selection algorithm that is used in this testing implements the nominal actuation mode. The pseudoinverse authority matrix was created as described in Section 3.2.2 and the nominal actuation mode as in Section 3.3.1. The process is very similar to the 2DOF example problem given throughout Chapter 3, although this involves the full, combined 56 thrusters of two docked SWARM units in 3 DOF. After the controller and the mixer have reconfigured, Phase 3 commences. The aggregated system translates as a single unit to a servicing location using the newly reconfigured control algorithm and thruster selection algorithm. After settling at the new location, the modules rotate together in Phase 4 as shown in Figure 5-5c. Finally, after a successful rotation, the modules undock and separate in the not-pictured Phase 5.

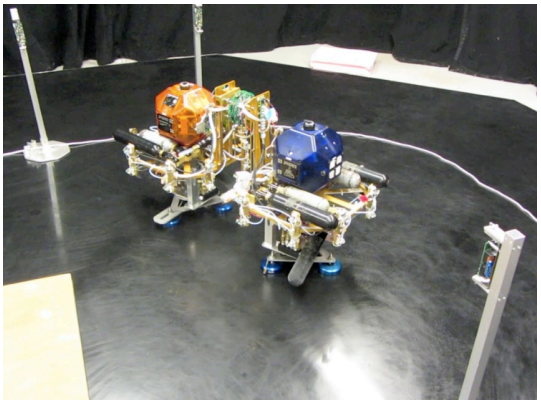
As applicable to this thesis, there are two instances of reconfiguring mixers from the standard SPHERES mixer throughout this process. The first reconfiguration is for the addition of the 16 extra thrusters on the SWARM propulsion carriage to the SPHERES satellite. The reconfiguration, which uses the nominal actuation mode, permits the satellite control of all 28 thrusters in the complete SWARM unit. The second reconfiguration occurs when the two modules dock together. Like the first reconfiguration, it also uses the nominal actuation mode and doubles the number of thrusters available for use to 56. All of these thruster selection logic reconfigurations are performed exactly as outlined in Chapter 3.



a.)



b.)



c.)

Figure 5-5: Pictures of ARMADAS testing: (a) Phase 1 (b) Phase 2 (c) Phase 4 [15]

5.2.2 Results

The ARMADAS testing produced substantial results in the risk-allocative path planning and RARC realms. This research should advance the field in so much that the algorithms are ready to be tested in a full 6 DOF rather than the 3-DOF testing performed on the ground SPHERES testbed. Qualitatively, the full CONOPS was achieved in a single, continuous run of the system without any human intervention. This achievement supports the concept that these systems will be fully operable autonomously once on orbit. Pictures from the full run are shown in Figure 5-5. This proves that the control algorithms and reconfigurable thruster selection algorithms can be implemented successfully in hardware and representative aggregative spacecraft systems.

Figures 5-6 and 5-7 show the results from the Phase 1 testing in ARMADAS.

Figure 5-6 shows eight attempts of the path planning and waypoint following maneuver. The center of mass of the single SWARM unit is shown as it travels around the obstacle to the rendezvous location. The module on the left, the servicer, knows where the obstacle is *a priori*. It plans the path to take appropriate risk to fill up an allocation. If the servicer took a wide turn around the first corner of the obstacle, there was substantial amount of margin left in the risk allocation by the time it attempts to round the second corner. Therefore, it attempts to cut the second corner closely to use up the remaining margin. The desired probability of success used in this experiment was about 90%, so having one of eight trajectories collide with the obstacle is representative of this parameter. These trajectories, and the trajectory in Figure 5-7, show that the reconfigured mixer worked effectively throughout Phase 1.

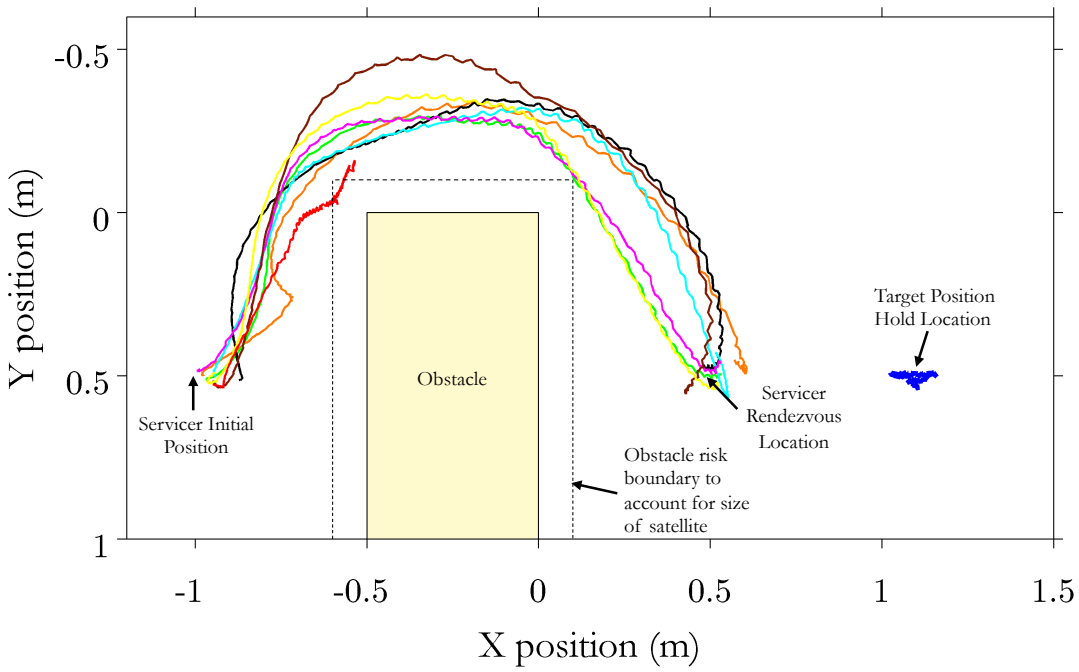


Figure 5-6: Multiple trials of the path planning and waypoint tracking maneuver in Phase 1 [15]

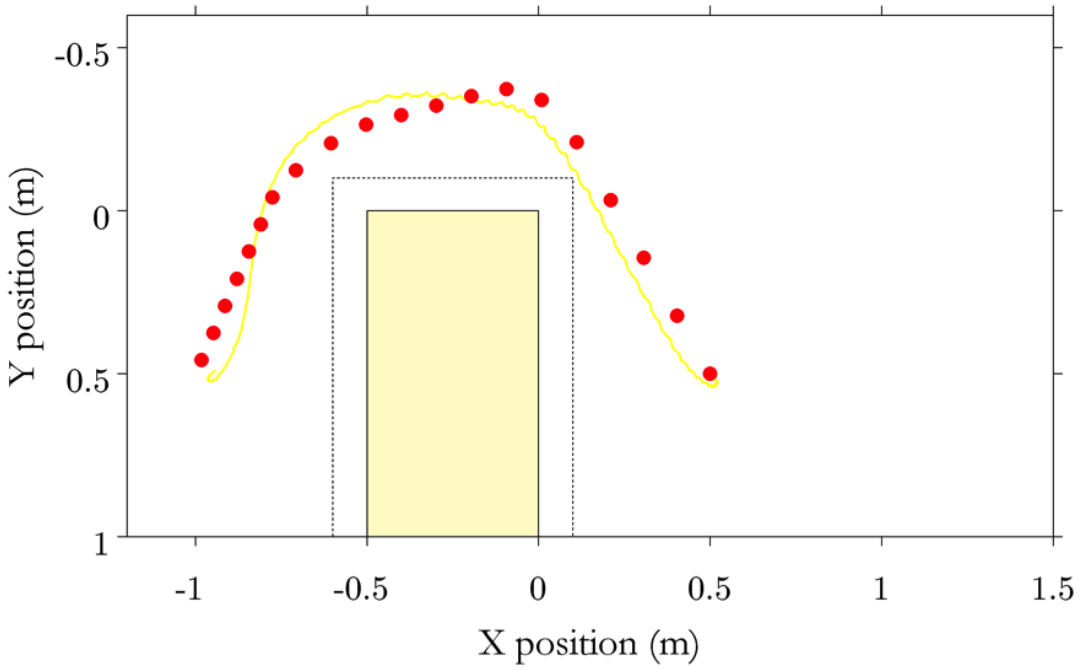


Figure 5-7: Waypoint tracking of the servicer satellite during Phase 1 [15]

Note that in Figure 5-7, the waypoints are given about two to three seconds ahead of the spacecraft's position, so at any point during the path, the spacecraft it attempting to control toward the waypoint that is a few positions ahead. The behavior that results is cutting corners a little tighter and lagging slightly behind the waypoints. Nevertheless, the satellite did follow the waypoints fairly tightly, only deviating by about 5 cm at certain points. Remember that the path is designed to adhere to a specific risk profile and optimizes the path of the satellite to achieve this probability of success. The algorithm is neither optimizing for fuel consumption nor for time as its main objective, although it does seem to compute a fairly short path that allows the satellite to pick up speed on the straight sections.

Figure 5-8 shows the response of the aggregated system in Phase 4, the 180-degree z-axis rotation maneuver. The behavior of the aggregated system is very repeatable as proven by the number of similar results in Figure 5-8. This rotation is a step function commanded by the guidance logic to the RARC controller. The RARC controller uses the system properties from the aggregated business card including

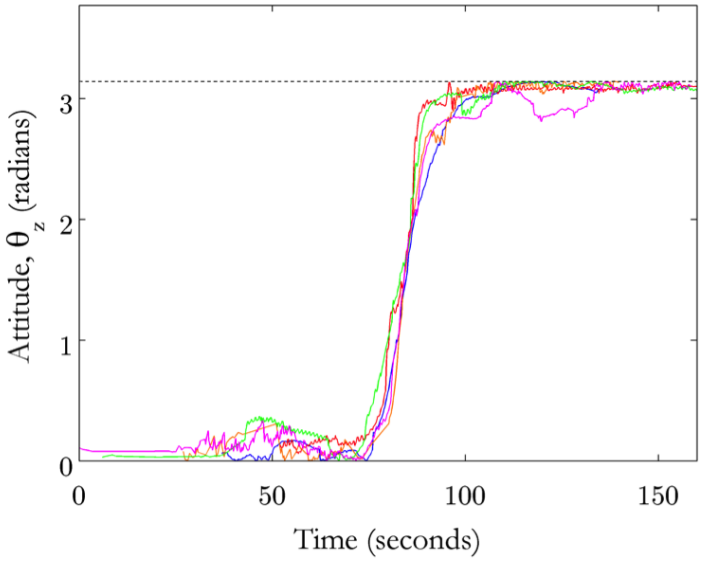


Figure 5-8: System response to the Phase 4 z-axis rotation maneuver [15]

the new mass properties, sensors and thrusters. At this point, the thruster selection logic has been reconfigured to account for all 56 thrusters. The responses show that the full aggregation has been successful, both in terms of the mixer and the control algorithms.

For thruster selection logic reconfiguration, the ARMADAS testbed is a perfect platform, because the reconfigurable mixers are designed for CONOPS exactly like these: with large number of thrusters on multiple modules, with reconfiguration online and in real time, and with aggregative system behavior. The SWARM units are the same as simulated in Section 4.2 in the Monte Carlo simulation. The single SWARM unit configuration is that used before Phase 3, while the two-docked SWARM units configuration is that used in Phase 3 and after. Testing the nominal actuation mode for these configurations both in the Monte Carlo simulation and in the hardware testing further validates the methods of Chapter 3.

Chapter 6

Conclusions and Future Work

6.1 Summary of Thesis

In summary, the work in this thesis proposes a methodology for reconfigurable thruster selection logic in the realm of aggregative spacecraft systems. The generalized form of the methods should allow implementation in a variety of systems without major modifications. The goal is that the algorithms are modular enough to allow reconfiguration of thruster selection logic on any combination of spacecraft modules given a method of determining the system properties. Through this thesis, the methodology of transferring system properties based on individual modules only knowing their own parameters (i.e., the business card model) proves to be very modular and incorporates low overhead computation costs. Building the reconfiguration algorithms around this architecture transfers the modularity afforded by the business card model to updating the thruster selection logic efficiently.

In Chapter 1, the interest in on-orbit assembly and servicing is motivated through a discussion of its relevance to large space structures, modular spacecraft design, and repair and refueling missions. Aggregative spacecraft systems are defined to be a set of interacting spacecraft modules that have the capability of docking or berthing with each other to perform tasks that none could individually achieve. Several testbeds based on the SPHERES facility are described. These testbeds are able to support experimentation with aggregative systems in a relevant environment for

close-proximity spacecraft dynamics. Finally, the research objective is formulated as developing a method to perform thruster selection in aggregative, thruster-controlled spacecraft systems through reconfigurable algorithms designed for specific actuation modes, which achieve desired precision, fuel-efficiency and agility goals.

In Chapter 2, a literature survey is discussed on the topics of aggregative control algorithms, system property identification upon reconfiguration, and thruster selection logic algorithms. Several control algorithms are discussed that could potentially solve the control problems associated with aggregative systems. Among these control algorithms are adaptive controllers, gain scheduling, online model calculation, and model switching. A few methods of determining system properties once modules have docked are look-up tables, system identification techniques and the business card method. To conclude the survey, the current state of the art in thruster selection logic is described including candidate optimal groupings, fuel optimization, and pseudoinverse methods. In summary, the work in the past has been performed on recovery in under-actuated systems, swapping mixers to known configurations, performing fuel optimization partially offline, and only operating in a single actuation mode. The work in this thesis is primarily for over-actuated systems, reconfiguring mixers in unknown configurations, fully online computation in real time, and a selection between several actuation modes.

In Chapter 3, the methodology of the reconfigurable thruster selection algorithms is described in detail. The process starts when two modules, each with their own propulsion system, dock together and attempt to control the thrusters of the aggregated system together. The modules swap business cards and the aggregated business card is updated to reflect the changes in mass, center of mass, inertia, number of thrusters, and the location and direction of those thrusters with respect to the center of mass. A pseudoinverse authority matrix is formed from this information in the business card. From this point, there is a choice between four actuation modes due to the overdetermined nature of the pseudoinverse authority matrix. The simplest, the nominal mode, uses the pseudoinverse authority matrix directly. The fuel-efficient mode solves a linear optimization problem. The precision mode uses

differential thruster firings to improve the resolution of applied forces. The fourth actuation mode, the agile mode performs a few manipulations to enable full thruster firing times on all thrusters.

In Chapter 4, the development and results from a Monte Carlo simulation of thruster selection logic algorithms are covered. The simulation produces a profile of controller-commanded forces and torques representative of an on-orbit assembly mission. This profile is fed through the reconfigurable mixer for all four actuation modes and for several spacecraft configurations. The results of this simulation show that compared to the nominal actuation mode the fuel-efficient mode can save as much as 13% of the total propellant consumed, the precision mode can decrease the error between commanded and applied forces by a factor of three, and the agile mode can increase the maximum achievable torque by about 15%. However, there are downsides to each of the actuation modes. The fuel-efficient mode takes an order of magnitude more computation time, the precision mode uses more than twice as much propellant, and the agile mode uses slightly more computation time and propellant. The results of the precision and nominal modes were validated through the SPHERES Simulation. In the precision mode, there is an improvement in the deadband of position hold maneuvers by about 30-50%.

In Chapter 5, two experiments implementing reconfigured thruster selection logic are examined. The Phoenix testing performed by MIT to support the DARPA Phoenix program shows how an aggregative system comprised of several small satellite modules positioned around the perimeter of an antenna aperture can collectively control the system. The ARMADAS testing used the SWARM units to perform a demonstration of a fully autonomous servicing scenario in the 3-DOF, ground testbed. Both of these projects successfully implemented the nominal actuation mode of the reconfigurable thruster selection logic. Through this testing it was shown that the algorithms discussed in Chapter 3 could be seamlessly integrated in the control systems of aggregative satellites.

6.2 Contributions

This section briefly lists the contributions presented in this thesis, as follows:

- Extension and implementation of the business card model from Mohan [26] to provide relevant parameters to the thruster selection logic problem
- Method to reconfigure any system's pseudoinverse authority matrix upon determination of the system's physical properties
- Formulation of four possible actuation modes that can be implemented in the thruster selection logic after computation of the pseudoinverse authority matrix
- The nominal actuation mode, which implements the pseudoinverse authority matrix directly and provides a time-efficient method to fuel-efficiently weight the usage of a thruster proportionally to the size of its moment arm about the center of mass
- The precise actuation mode, which uses differential thruster firing to improve the resolution of the thruster firing times from the thruster's minimum on-time to the flight computer clock period
- The agile actuation mode, which groups thruster pairs into common torque groups and ramps firing times up equally among all thrusters in a group to enable maximum torque capabilities of the spacecraft
- Thruster selection logic with the capability of switching the actuation mode in real time to adjust for system needs over time
- Verification and validation of these reconfigurable thruster selection algorithms through a Monte Carlo thruster selection logic simulation
- Validation and analysis of the nominal and precision actuation modes in a high fidelity, 6-DOF, close-proximity satellite dynamics simulation

- Hardware testing of the nominal actuation mode through the reconfiguration of the thruster selection logic upon the docking of two initially independent satellites

6.3 Recommendations for Future Work

This section lists some recommended areas either meant to be extensions of the work in this thesis or novel investigations:

- Specifically to further the verification and validation of the reconfigurable thruster selection algorithms presented in the thesis, there should be additional hardware testing performed. This hardware testing should implement each of the four actuation modes in a representative aggregative spacecraft system. The performance of the actuation modes should be compared with the results from this thesis and the simulation in Chapter 4 should be further validated. This hardware testing would require porting the reconfigurable mixer from MATLAB® to C for the purpose of running in a real-time embedded system.

- The SPHERES Simulation should be upgraded to simulate aggregative spacecraft dynamics including docking and contact dynamics. The simulation should allow the spacecraft to physically dock together, such that the thrusters from one spacecraft would act on the center of mass of the aggregated system rather than its own. The inertia and mass properties should also update in the simulation.

- The computational efficiency of the reconfigurable thruster selection logic should be investigated more thoroughly. There should be an in-depth comparison of different optimization techniques for the fuel-optimal actuation mode as implemented in hardware. Gradient-free and other heuristic methods should be investigated as potential for reducing computation time. A modified candidate optimal group technique could be implemented for aggregative spacecraft systems. The relative merits of computation of the authority matrix or solution of

the $Ax = b$ problem should be determined. A critical frequency of reconfiguration should be determined when it would be more beneficial to compute the inverse rather than solve the $Ax = b$ problem repeatedly.

- Other methods of determining the aggregated system's physical properties should be investigated. Sys ID techniques could be implemented to use the available sensors to determine precise mass properties of the system through response to known actuation. Sys ID techniques could also more precisely determine the effective force and torque an individual thruster produces on the system. These values could be directly input into the columns of the pseudoinverse authority matrix. Robust or adaptive controllers could be implemented to reduce the requirements on the knowledge of system parameters.

- Currently, there is not a method to automatically determine if thrusters are impinged upon docking two spacecraft together. Geometric analysis or Sys ID techniques could be implemented to determine whether certain thrusters should be included in the authority matrix or not.

- The reconfigurable thruster selection logic could be expanded into reconfigurable actuator selection logic to include more diverse actuators, such as reaction wheels, CMGs, magnet-torquers, or electromagnetic formation flight devices. This generalization would likely be at the same level as the thruster algorithms. The algorithm would choose which actuators to use based on current statuses, fuel/power-efficiency, maximum capable forces and torques, and finest control resolution.

- An uncertainty analysis should be performed on the on-orbit assembly and servicing problem. With this analysis, appropriate control algorithms and methods to deal with the uncertainty in mass properties of a system should be developed. Through these efforts, the reconfigurable thruster selection logic in the thesis could be expanded to be more robust to the presence of uncertainty.

Appendix A

Design of the SPHERES Halo

A.1 System Overview

As motivated in Chapter 1, there is currently a large interest in on-orbit assembly and servicing missions. To further develop the technologies used in these missions, risk-reducing research needs to be performed to prove mission viability. This risk-reduction must occur in a relevant environment, traceable to the mission CONOPS. A great opportunity exists in leveraging the SPHERES facility onboard the ISS to create a risk-reduction testbed for close-proximity spacecraft dynamics in a relevant microgravity environment. This new testbed is being jointly developed by DARPA, NASA Ames, MIT, and AFS through the InSPIRE-II program. To use SPHERES as the foundation of this testbed, several capabilities need to be developed.

First, to emulate a servicer/tender spacecraft as the DARPA Phoenix program uses, the testbed needs to have a method of docking or berthing to a target satellite. Thus, the SPHERES Docking Port, pictured in the top right of Figure A-1, was born. Furthermore, in a mission scenario, the servicer/tender needs to be able to use vision-based navigation and other sensors to assist in the process, while still having access to a docking port. The SPHERES satellites only have one electromechanical expansion port, the ExpV2. Thus, there is a need to break the ExpV2 out into several additional ports to enable the use of multiple peripheral devices simultaneously. The converged-upon design of this expansion device, named the Halo, is shown alone on

the left of Figure A-1 and fully integrated in the SPHERES facility on the right of Figure A-1.

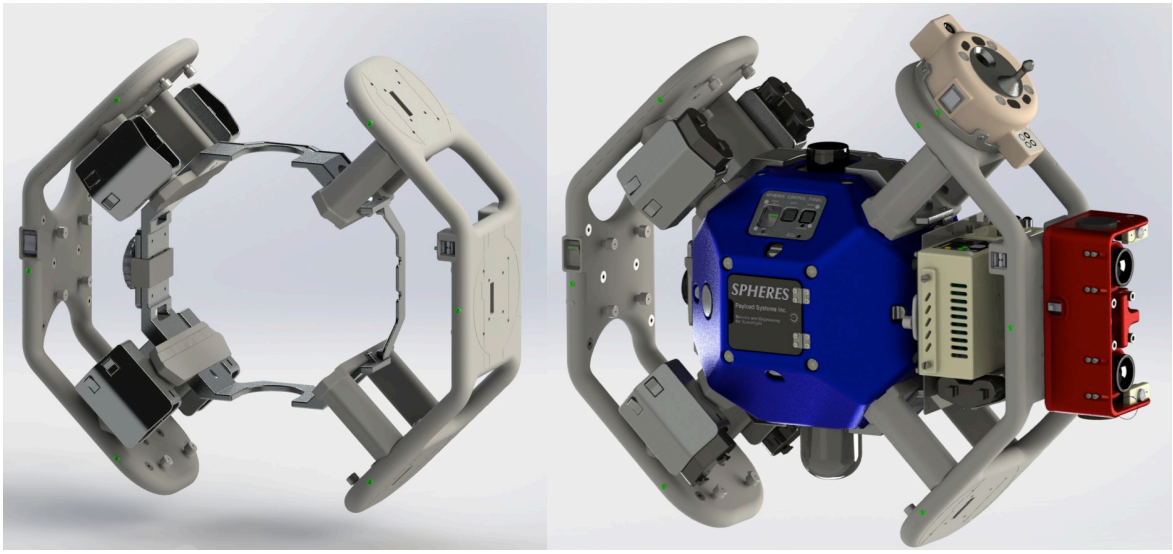


Figure A-1: CAD images of the Halo as of May 2014: (left) the Halo alone and (right) the integrated Halo with a SPHERES satellite, VERTIGO Avionics Stack, Optics Mount and Docking Port [23]

The Halo is assembled onto the SPHERES satellite to transform the ExpV2 into six external expansion ports around the perimeter of the SPHERES satellite. These electromechanical interfaces are known as Halo Ports. Each Halo Port has the same electrical and mechanical connection as the VERTIGO Avionics Stack. As is not the case with the VERTIGO Avionics Stack interface, however, the Halo Port has an additional new mechanical interface. This interface adds thumbscrews on the Halo side to prevent the need for peripheral devices to have thumbscrews or adapters. Currently, several peripheral devices have been successfully used on the Halo, including the VERTIGO Optics Mount, SPHERES Docking Port, an optical range finder, a thermographic camera, and CMGs.

The Halo Ports each support two USB connections and one Gigabit Ethernet connection. These data lines are routed through USB and Ethernet hubs respectively to the VERTIGO Avionics Stack. The VERTIGO Avionics Stack is a flight computer with a Linux operating environment that handles all communications between the SPHERES satellites and the peripherals and performs any heavy data processing

required. In addition, the Halo Ports each provide 5V, 1A and 11.1V, 1.9A power lines to the peripherals. The Halo power system contains four parallel Nikon rechargeable batteries identical to those used on the VERTIGO Avionics Stack.

As of the writing of this thesis, the Halo and SPHERES Docking Port are scheduled to launch to the ISS in October 2014. Prototypes haven been built by AFS, but the flight hardware is yet to come. Three of the five flight hardware versions of the Halo will be sent to the ISS. Six docking ports of the ten flight versions will be sent. Once on orbit, there are several test sessions scheduled to both checkout the hardware and perform science research relevant and traceable to on-orbit assembly and servicing CONOPS.

A more detailed discussion of the design of the Halo can be found in McCarthy [24] and the continuously updated Halo design document [23]. The work that is discussed in this thesis in Section A.2 was performed solely by the author. The work discussed in Section A.3 was performed in conjunction with McCarthy.

A.2 Structural Design

A major trade study was performed on the high-level structural design of the Halo to determine the best orientation, size, and port locations. Several design architectures were explored and evaluated against a set of metrics to assess their relative merits. The metrics include common properties such as mass, moments of inertia, shift in the center of mass, and number of parts. However, what drove the selection of the designs the most was the assessed difficulty in adhering to crew accessibility and SPHERES keep out zone requirements. The crew must be able to freely install batteries and CO_2 tanks into the SPHERES satellite without removing the Halo. The CO_2 pressure regulator knob and SPHERES user control panel must also be easily accessible by the crew. The pressure gauge must have a line of sight for inspection by the crew. In addition, the SPHERES keep out zone requirements relate mainly to the thruster plume impingement and the blockage of the ultrasound and infrared sensors. From this analysis, three front-running architectures emerged as shown on the left side of

Figure A-2.

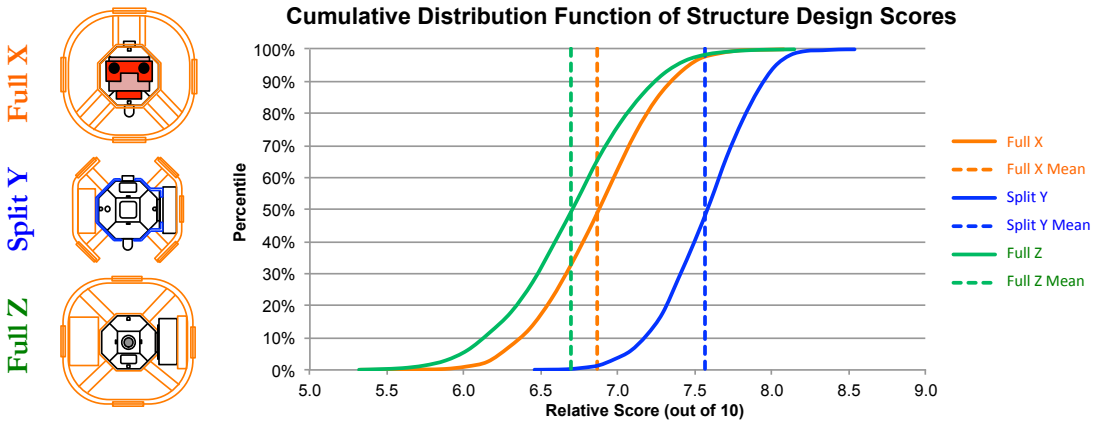


Figure A-2: Trade study results comparing three potential Halo structures [14]

A modified Pugh matrix was used to rank the designs based on scores in the different metric areas. To account for uncertainty in several of the calculated metrics, a Monte Carlo method was used. In this method, each variable was assigned a confidence level that dictated the width of a normal distribution from which a particular score would be drawn. The analysis was run for 10,000 iterations and the cumulative distribution functions of the design scores were compared in Figure A-2. Narrower distributions relate to more confidence in a design's score, while the mean of the distribution indicates the nominal score. Based on this analysis, riskier but better performing designs may not be chosen as there is a certain probability, that based on the uncertain computations performed, the actual score would be worse. Nevertheless, the best performing design did not leave much room for debate, leaving only a small portion of the distribution below the other two median scores. Out of this analysis, the Split Y design was chosen. As one can see in Figure A-3a, the Split Y design avoids each of the standard SPHERES keep out zones. Additionally, it is lower in mass due to the gap in the top and bottom, it has a smaller radius than the other designs, and it provides the best common work area in the field of view of the VERTIGO Optics Mount.

Figure A-3 shows the adherence of the Halo to the SPHERES keep out zones and crew accessibility requirements. In Figure A-3a, the yellow cones indicate the thruster

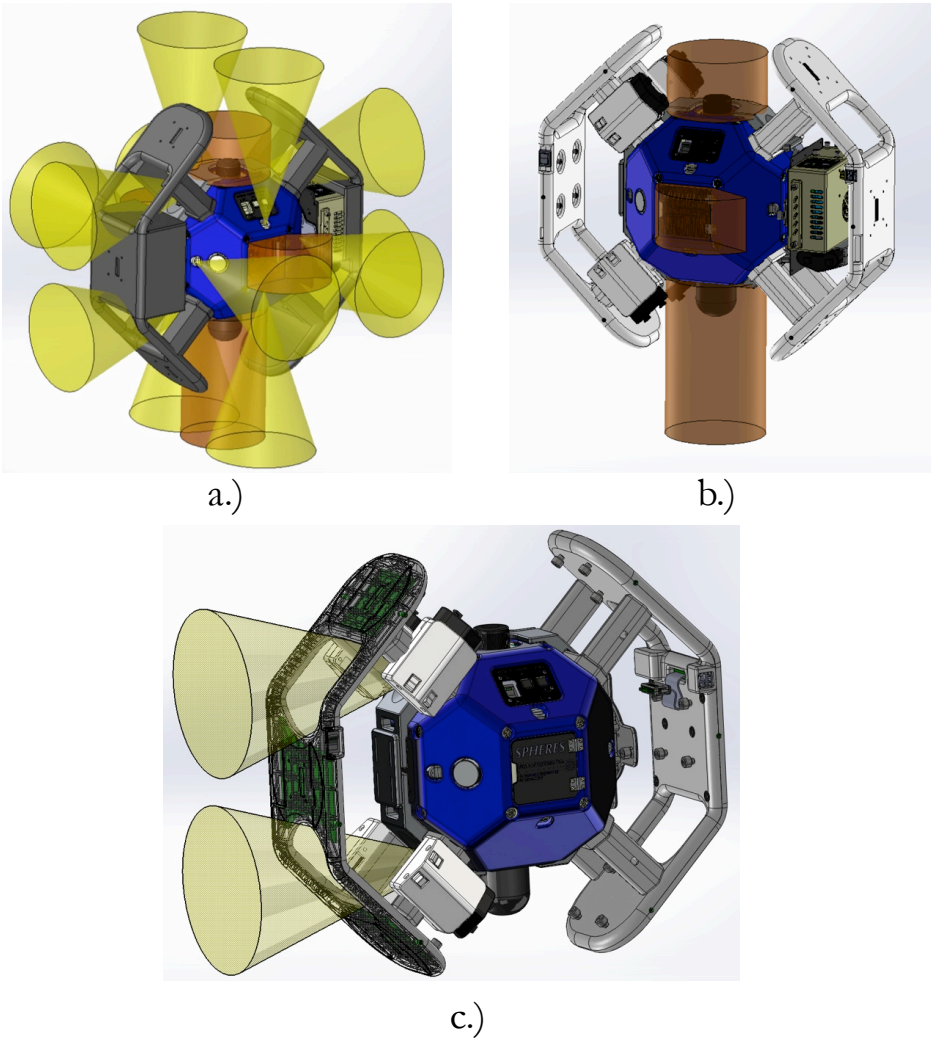


Figure A-3: Keep out zones and thruster plumes vs. the Halo structure [23] [14]

plumes from the SPHERES satellite. One can see that in this initial design of the Halo, the thruster plumes in the y and z-axes are completely unblocked. The x-axis thruster plumes flow through the hole in the structure but were unfortunately still partially blocked by the Halo Port. The crew accessibility requirements are shown by the orange volumes in Figure A-3a and A-3b. As the Halo structure was further developed, the thruster impingement was improved upon while still maintaining the crew accessibility requirements. In Figure A-3c, the back side of the Halo is shaped to curve around the thruster plume and prevent any unnecessary plume impingement. Unfortunately the front side of the Halo does not allow for any improvements in plume impingement, because the VERTIGO Avionics Stack blocks most of the plume rather

than the Halo itself. Having been tested successfully on orbit, the SPHERES system seems to actuation effectively with VERTIGO attached. Thus, there is a high level of confidence that the system will be effective on orbit. Note that there will be additional plume impingement when peripheral devices are installed and extend over the hole in the structure.

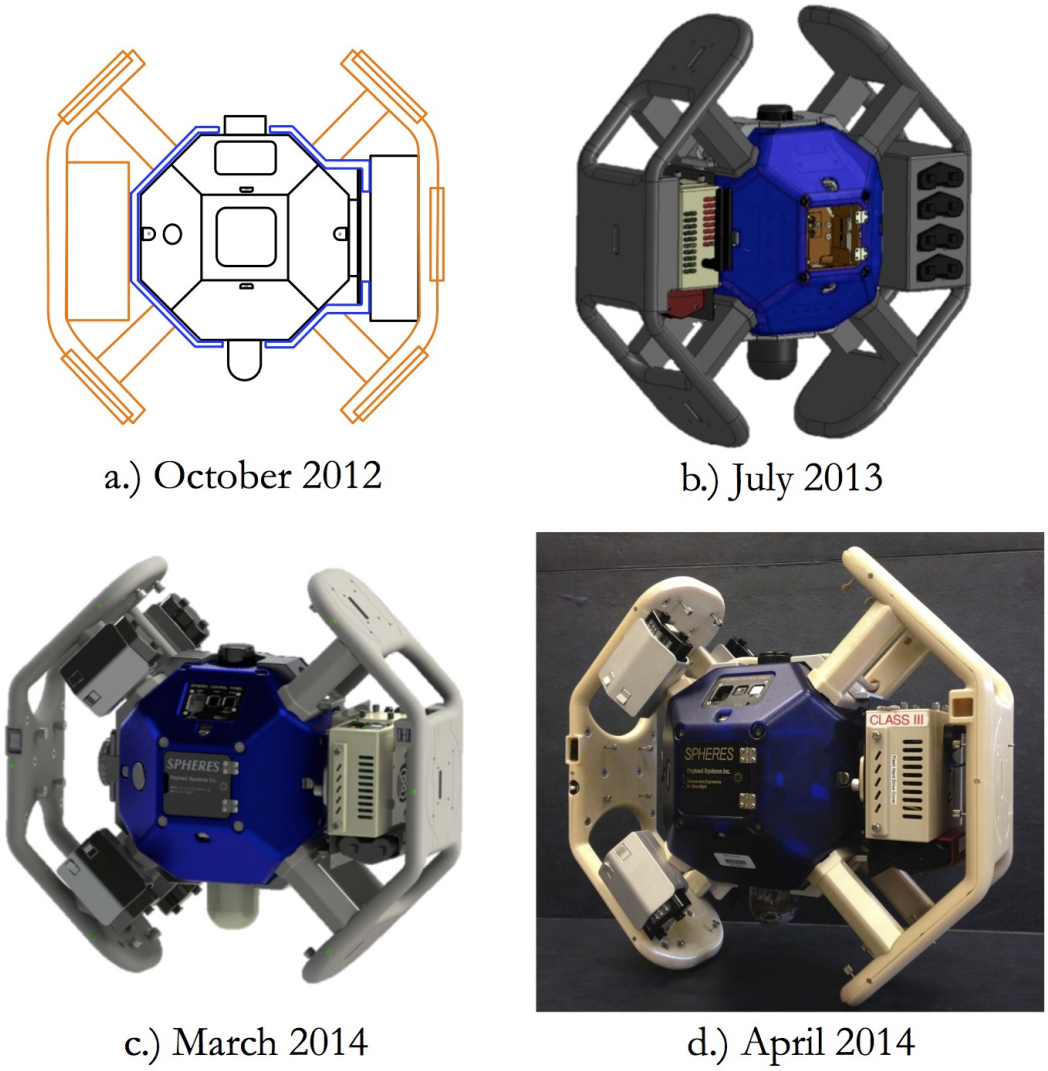


Figure A-4: Progression of the Halo structural design from initial concept (a) to prototype hardware (d) [23]

Figure A-4 shows the stages of development of the Halo from initial conceptual design in a graduate engineering class in 2012 to the CAD developed for the system requirements review in 2013 to the relatively final CAD completed by AFS in March

2014 to finally being 3D printed for the prototype in April 2014. The process will be the fastest the MIT SSL has completed flight hardware design and manufacture from concept to flight delivery. Keeping in mind that there was a period of inactivity between the graduate class and the start of the InSPIRE-II program, the total period from concept to flight hardware delivery will have taken approximately 14 months.

A.3 Software Architectural Design

Adding the Halo to the SPHERES and VERTIGO systems has several implications for how the software architecture should be updated. Particularly in the VERTIGO system, the software was built to only interact with a single, specific peripheral device: the Optics Mount. In the Halo software, there is a requirement for the software interfaces to be modular and for it to be easy to integrate new devices. Thus, the VERTIGO Linux software needs to be updated to a more general form that is capable of handling these new requirements. The Halo Linux software is this generalized architecture, hinging upon the HaloCore Application Programming Interface (API).

The Halo Linux software is outline in Figure A-5. The top-level API is called HaloCore, which consists of some common libraries and functions applicable to any system using VERTIGO and SPHERES. The responsibilities of HaloCore include parsing parameter files, setting thread priorities, initiating threads, running the main while loop for science, sending commands to the Halo motherboard, interfacing with the SPHERES satellite, and generic data storage, video streaming, and networking. More specific to the individual peripheral devices attached are the peripheral APIs. For example, the OpticsMountCore and DockingPortCore contain functions specific to communicating and storing data from these peripheral devices. Any data processing libraries or functions only relevant to a specific peripheral would be located in a peripheral API.

The class structure implemented in the Halo Linux software, shown in Figure A-6, is similar to the API structure. There is a generic HaloGSP class, which contains functions that all peripheral classes (e.g., OpticsMountGSP and DockingPortGSP)

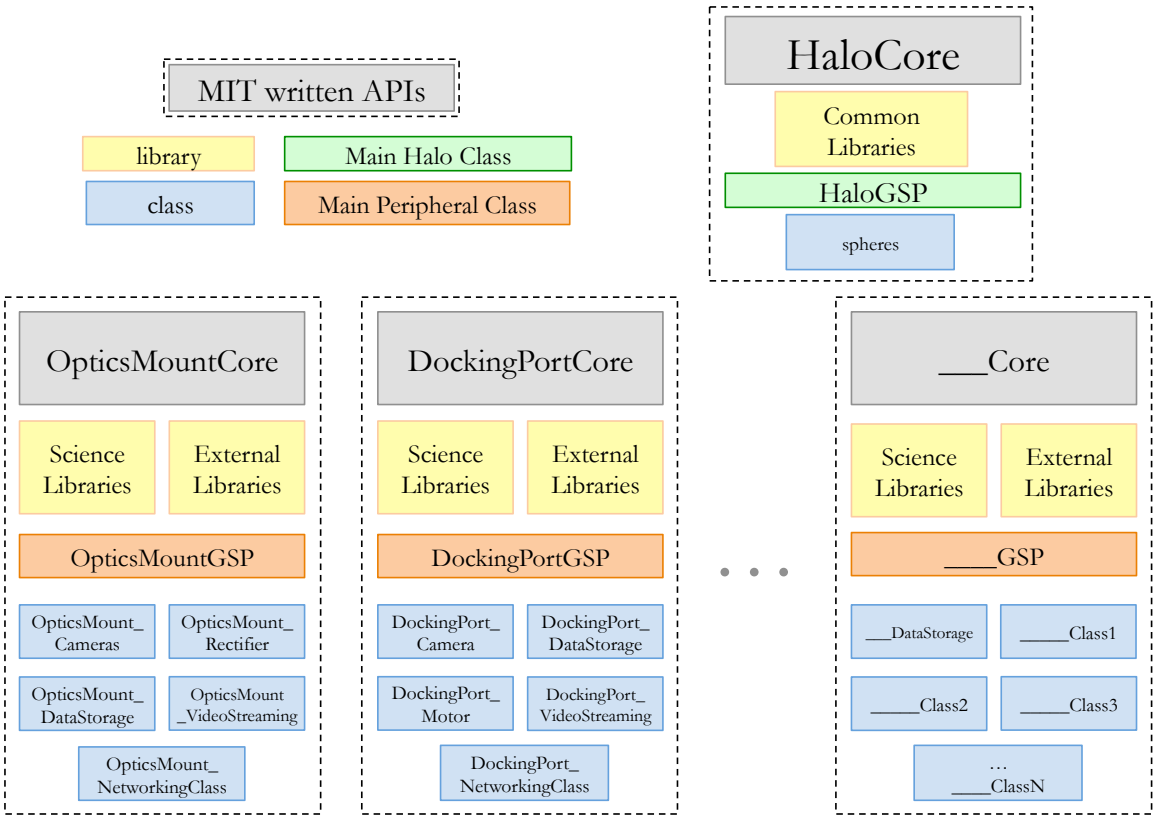


Figure A-5: Overview of the Halo Linux software and class structure [22]

inherit virtually in C++. A level below this is the guest scientist code that would run during an ISS test. This code is contained in the TestClass class and inherits functionality from all of the peripheral classes. The virtual inheritance of HaloGSP is important to note here because the peripheral classes all need to reference the same HaloGSP object and would interfere with each other if this were not the case.

Figure A-7 is a diagram depicting the interactions between software and hardware components in the Halo system. Dotted, terminated lines represent interfaces that are visible to the guest scientist developing code for a peripheral device. Solid lines show interfaces that cannot be modified by the guest scientist. There are two levels of hardware: the Halo itself and the external peripheral devices. There are also four levels of Linux Software: the drivers for peripheral devices, the daemon to interact with the ground computer and monitor the hardware, the Halo Linux software as described previously, and the guest scientist code in the far right column.

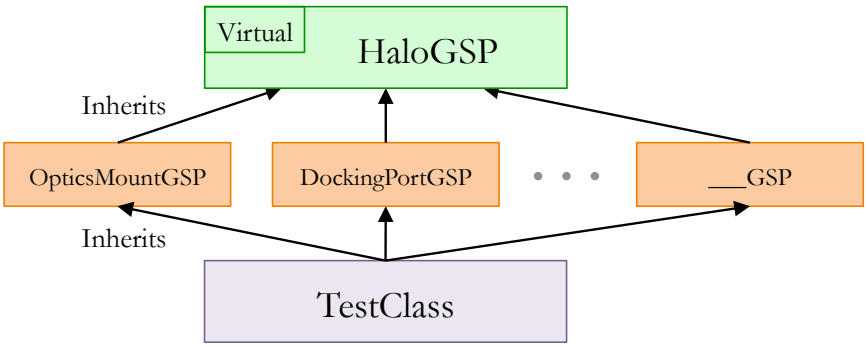


Figure A-6: Halo software architecture implementation and interactions [22]

Each of the guest scientist functions in the rightmost column of Figure A-7, are placed in Figure A-8 to show how they are called by the main function of HaloGSP. The guest scientist should define these functions in TestClass. First, the `runMain` function initializes by reading and parsing a parameter file and creating relevant threads. From here the guest scientist can create his own threads and perform any user specific initializations that could not be performed through the parameter file parsing. Finally, if desired, a background task thread is created with lowest priority. After these initializations, the main science while loop executes, calling the `GSrunMain` function, until the daemon terminates the test. This `GSrunMain` function has its own thread to contain all of the science and data processing code required by the guest scientist. The guest scientist should then shutdown the threads he created and perform any other required clean up procedures before the test eventually terminates.

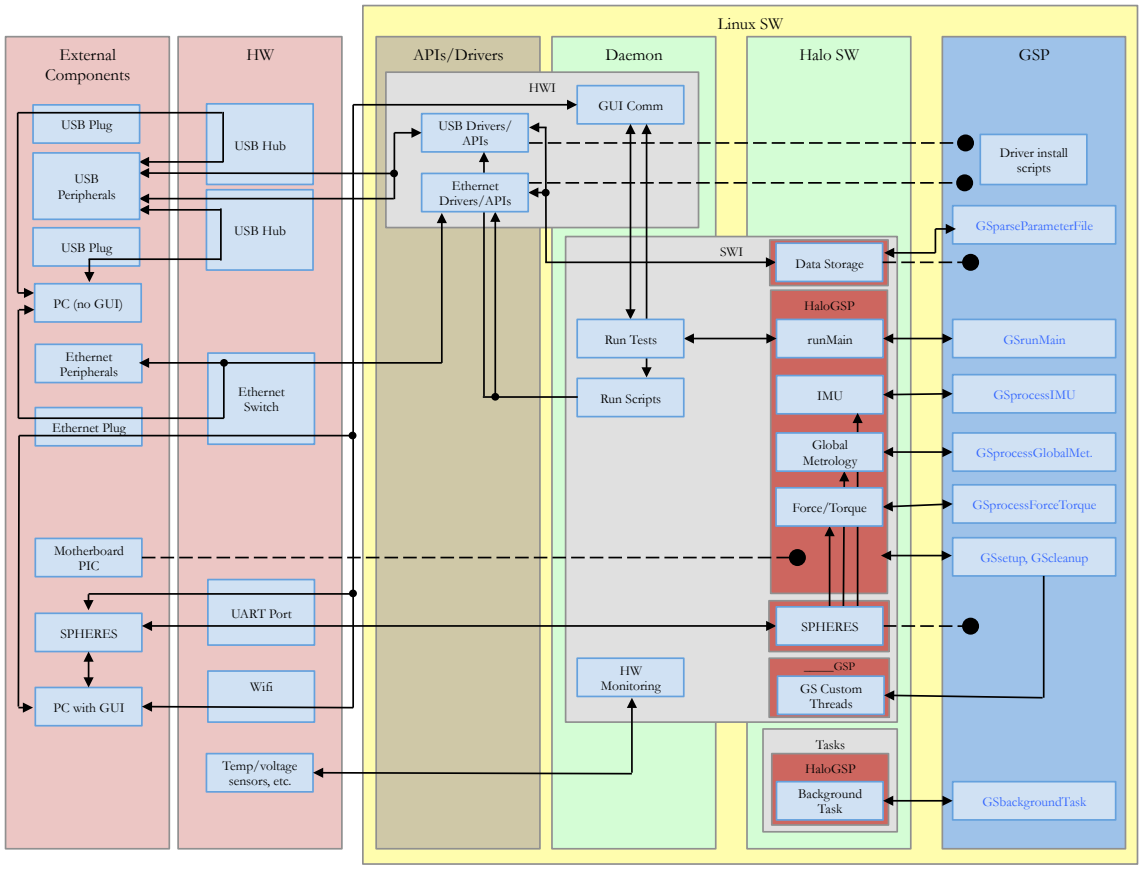


Figure A-7: Halo software architecture implementation and interactions [22]

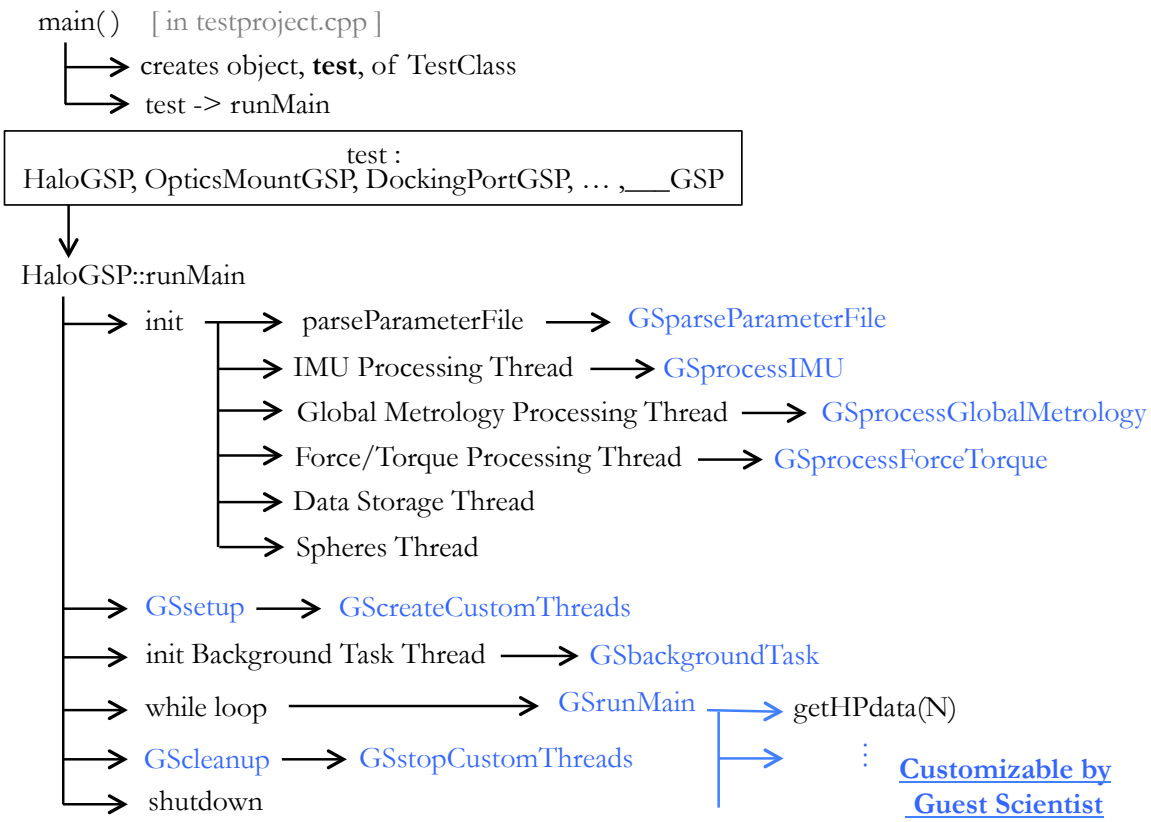


Figure A-8: Main loop of a HaloCore test project [22]

Appendix B

LQR Controller for SPHERES

B.1 Introduction and Motivation

This document details the design of a state-space controller for use in the SPHERES MATLAB®/Simulink® Simulation. The mission scenario will be that of on-orbit robotic servicing of satellites. An inspector satellite will "investigate" a static satellite in need of servicing. This investigation involves vision-based mapping and estimation of the target satellite's inertial properties and identification of a docking interface (although this part of the problem will not be covered by the project). The controller is able to control the position, attitude, velocity, and rate of a SPHERES satellite, using the existing SPHERES estimator to receive sensing data (position, attitude, velocity and rate). The satellite will track a reference path through waypoint tracking. This investigation will involve a circular, planar path around the target satellite with the investigator satellite always pointing its sensors (stereo cameras) toward the target satellite. After the sensing "orbits" are complete, the investigator satellite changes roles and will now become a servicing satellite that will attempt to dock to the target satellite. With the same controller, the servicing satellite will track a series of waypoints and proceed to "dock" to the target satellite. For the purpose of this project, this docking will be considered successful if the two satellites contact with a less than to be discussed velocity error and overshoot in attitude and position. The system will be operating in a 6-DOF, micro-gravity environment in the simulation

and will follow "$F = ma$" and "$\tau = I\alpha$" well enough that the system can be modeled under them.

Section B.7 contains the MATLAB® code used to design the controller. Section B.6 contains some additional plots for reference.

B.2 Developing the State-Space Plant

The plant for a free-floating satellite in a micro-gravity environment can be derived from Newton's Second Law, $F = ma$, and the corresponding angular version, $\tau = I\alpha$. It turns out that simply using these two equations is all that is necessary to adequately model the SPHERES satellite dynamics in micro-gravity. The state vector of the satellite, $\boldsymbol{x}$, is its positions, velocities, Euler-angle attitude and angular rates in an inertial frame (resulting in a twelve element vector). The actual SPHERES satellite will compute its attitude in quaternion form, although for the purposes of this project, the Euler angle representation will be used in the control law.

The six-element input vector, $\boldsymbol{u}$, consists of the external forces and torques in each axis acting on the system (i.e., the forces and torques created from the thruster firings).

See below for the formulation of the state-space representation of the open-loop system dynamics in the $\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x}+\mathbf{B}\boldsymbol{u}$ form. Note that in order to linearize the system, we must neglect the products of inertia from the inertia tensor, and thus are only left with the primary moments of inertia. Luckily, this is not a bad approximation, because the SPHERES satellite axes are very closely aligned with the principal axes, which makes the products of inertia close to zero in the first place. A nice side effect of this is that all of the states are decoupled from each other, so controlling attitude is separated from translation.

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \theta_x \\ \theta_y \\ \theta_z \\ \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \end{bmatrix} \qquad \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{\theta}_x \\ \dot{\theta}_y \\ \dot{\theta}_z \\ \ddot{\theta}_x \\ \ddot{\theta}_y \\ \ddot{\theta}_z \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \boldsymbol{u}$$

Once the loop is closed with negative feedback gains from $\mathbf{K}$, the output of the controller, also $\boldsymbol{u}$, will be the required forces and torques that the satellite will need to achieve to track a reference input. This output is translated from forces and torques to firing times of the necessary thrusters, through a form of jet-selection logic known

to SPHERES as the "mixer." The control engineer does not have to worry about controlling thrusters directly and can develop controllers with forces and torques as the outputs. The formulation so far is just as a regulator, although a reference input will be added in the next section.

$$\boldsymbol{u} = -\mathbf{K}\boldsymbol{x}$$

B.3 Developing an LQR Controller

Before designing the controller, the controllability of the plant should be examined. To do this, we can determine if the controllability matrix is full rank. A quick MATLAB® check, rank(ctrb(ss(A,B,C,D))), returns the rank of the controllability matrix to be 12, which is the equal to the number of rows, making it full rank. Thus the open-loop system is controllable, and we can apply any of several control strategies to the system.

Also, before diving into the design of the controller, we must consider what requirements we are going to place on the controller. The first thing to note is that the maximum force the SPHERES thrusters can execute is about 0.1 N each, creating an equivalent of about 0.2 N force in each axial direction and a torque of about 0.02 Nm. However, the SPHERES satellites operate on a 1 second duty cycle, where thrusters are only able to fire for 200 ms. The SPHERES mixer converts the commanded forces and torques into commanded impulses and attempts to match the total impulse that would be provided over the 1 second long control cycle. Since we're operating at a maximum 20% duty cycle, the maximum impulse to be commanded can be 0.04 Ns and 0.004 Nms. This translates to a maximum commanded force and torque of 0.04 N and 0.004 Nm. This must be considered upon implementing the controller (i.e., the required control input cannot be too large). Thus, this is our first requirement. The other requirements are derived from the allowable inspection and docking tolerances. Inspection has actually quite relaxed constraints, as all that is required is for the target satellite to stay within the field of view of the inspector satellite. This translates to overshoot and steady state error requirements in pointing to be less than

the $\pm 30^\circ$ field of view of the cameras. The tracking of the circular path around the target satellite is desired to be smooth, although there is no direct requirement on the tolerances allowed as long as it travels around the target. There is a benefit to following the design trajectory as far as camera focal length and depth computation are concerned, but much of this can be adjusted while in flight so it does not pose a large problem. The translational and attitude requirements are much stricter for the docking portion of testing. The geometry of the docking port requires that the controller be able to control a step response to within ± 1.5 cm and $\pm 2^\circ$ in all axes. All motion can be and is expected to be slow as the sensors inspecting the target satellite need computation time to process the data collected during the circumnavigation and the SPHERES satellites have very limited actuation capabilities. Thus an orbit around the target satellite should be on the order of 4 minutes or so and the docking step response is allowed to take as long as a minute or more.

LQR controllers are robust—guaranteeing a phase margin of $60^\circ$and gain margins of $(1/2, \infty)$—so choosing to design a controller with these methods should result in a controller that will be stable under the unavoidable sensor and actuator noise on the SPHERES testbed. To design the LQR controller, we must optimize the following cost function for our choice of state (Q) and control (R) cost matrices:

$$J_{LQR} = \int_0^\infty \left[\boldsymbol{x}(t)^T Q \boldsymbol{x}(t) + \boldsymbol{u}(t)^T R \boldsymbol{u}(t) \right] dt$$

Optimizing J_{LQR} is generally performed through using the Algebraic Riccati Equation to find the optimal K matrix described in the previous section, however for the purposes of this project, the MATLAB® lqr(...) function is used to compute the optimal K. Thus, designing the LQR controller is reduced to choosing appropriate Q and R cost matrices. This process was performed by following Bryson's Rules as general guidelines to construct the matrices and then weighting the control matrix so the control inputs will not saturate the SPHERES thrusters. Bryson's Rules attempt to normalize the signals by dividing the corresponding entry in the Q and R matrices by the maximum expected responses and control inputs. The maximum forces and torques for the system are discussed in the requirements paragraph above, and the

allowable errors are used for the maximum expected responses. Reasonable estimates for the remaining responses (velocity and rate) are derived from experience with the SPHERES testbed. Lastly, the weighting between the Q and R matrices was reserved as a design variable to control the magnitude of the commanded forces and torques to values that the SPHERES satellite can achieve without saturating. This weighting was iterated upon until the desired responses were obtained from simulation.

See Section B.7 for the MATLAB® code and output used to develop the controller.

The resulting controller was developed (parameters and Q and R matrices are detailed in the MATLAB® code in Section B.7):

$$\boldsymbol{u}(t) = \boldsymbol{r}(t) - \mathbf{K}\boldsymbol{x}(t)$$

with,

$$\mathbf{K} = \begin{bmatrix} 0.224 & 0 & 0 & 1.785 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.224 & 0 & 0 & 1.785 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.224 & 0 & 0 & 1.785 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0096 & 0 & 0 & 0.0234 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0096 & 0 & 0 & 0.0234 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0096 & 0 & 0 & 0.0217 \end{bmatrix}$$

resulting in a closed-loop system of the form:

$$\dot{\boldsymbol{x}}(t) = (\mathbf{A\text{-}BK})\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{r}(t)$$

Interestingly enough, the result of the LQR formulation (as an effect of the simple linearized model) was a set of purely proportional and derivative gains with extremely small gains corresponding to other states (much below float precision).

Please note that the SPHERES program has a well developed and tested Extended Kalman Filter (EKF) that will be used as the estimator for this project, so there will

be no discussion on the design of an estimator, and we will assume that the estimated state output from the EKF is sufficient.

B.4 Implementation in the SPHERES Simulation

The SPHERES team has developed a MATLAB®/Simulink® simulation that allows users to run C code through simulation before implementing on the hardware. This simulation is very high fidelity, modeling both the dynamics of the system as well as the internal electronics and sensors of the satellite. The simulated noise in the metrology/sensing system and in the actuators closely matches what has been seen on orbit. Thus implementing the LQR controller in the simulation is a large first step in validating the controller for use on satellites in a micro-gravity environment.

Implementing the LQR controller in the MATLAB® simulator (and eventually on the SPHERES satellites) requires that C code functions are written to multiply the control state errors by their associated gains in the K matrix to compute the desired forces and torques. These forces and torques then need to be fed into the SPHERES mixer and eventually sent as firing times to the thrusters. The maneuvers (i.e., waypoint tracking for the circumnavigation and step response for the docking) were coded in C and used in the control law. The circumnavigation waypoints are fed as the desired state into the controller and updated every second. The waypoints are paced so that the inspector satellite will complete one 360 degree orbit around the target satellite in four minutes and then position hold near the "sensed" docking port location. The "sensed" docking port location was hardcoded for the purposes of this project. The waypoints specify the [x y z] position of the satellite around the target satellite as well as the attitude [q0 q1 q2 q3] quaternion required to point the sensors at the target satellite at all times. The computation of the desired attitude is updated with the satellites current state as to always point to the target no matter if the translational controller is functioning properly or not. All of the waypoints include the control of multiple states of the satellite, so full control of all 12 states is continuously being performed.

B.5 Results from the SPHERES Simulation

Before we dive into talking about results, it is helpful to fully visualize the concept of operations that we are trying to control. As described earlier, there is an inspector satellite and a target. The inspector will complete a circular orbit around the target, always pointing its sensors at the target, then will perform a docking maneuver. These are distinct operations. The circumnavigation/inspection tests the controllers waypoint tracking performance, and the docking tests the step response. Keep this concept in mind while going through the results.

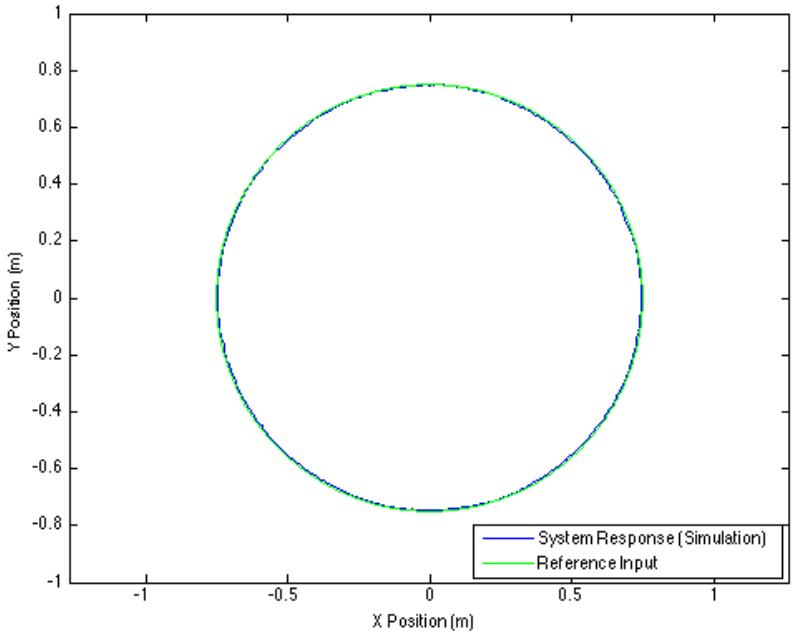


Figure B-1: Position tracking of a circular set of waypoints in a 4-minute, 0.75m orbit

The performance of the LQR controller was generally excellent for the selected concept of operations. It met all of the requirements set in both the circumnavigation segment and in the docking maneuver. Actually, normal SPHERES docking maneuvers track a reference glide slope into the docking position after passing through a series of tolerance gates, but this controller seems to have sufficient performance to simply use a step input to dock. See Figure B-1 to see how well the controller tracked the reference waypoints in the circular path around the target satellite at the origin.

This great performance likely has to do with the fact that the actual system closely matches our linearized model, as predicted in Section B.2.

Not considering the time aspect, the controller tracked the reference input to under a centimeter of accuracy (which is close to the 5mm noise in the sensing system). However, when we look at how quickly the controller responded to the reference input in Figures B-2 and B-3, we can see that the position controller lags the reference input by about 10 seconds and the attitude controller lags by about 5 seconds. This results in a consistently lower radius of the circle than desired by a mean of about 1.5 mm, which poses no problem for this application as that is lower than the pixel limited depth resolution of the stereo camera sensing system. Since the 4 minute orbit is fairly slow, the controller was tested on quicker orbit times of 2 minutes and 1 minute. The plots are shown in Section B.6, but generally the 10 second lag of the controller causes the radius of the followed circle to shrink. For the 1 minute orbit, the radius shrinks by 10cm, which is too large of an error for the camera system to handle, but the 2 minute case only shrinks by about 2 cm, which would add only a small amount of error to the vision system. The 5 second lag of the attitude is not because the LQR weighted the attitude to respond quicker than the position (it was weighted equally), but because the tracking algorithm uses an average of the current relative position of the two satellites and the commanded position. This halves the angle the inspector satellite would need to rotate to align with the target satellite. This averaging actually works very well in that the sensors are pointed at the target satellite almost perfectly at all times with no "perceived lag."

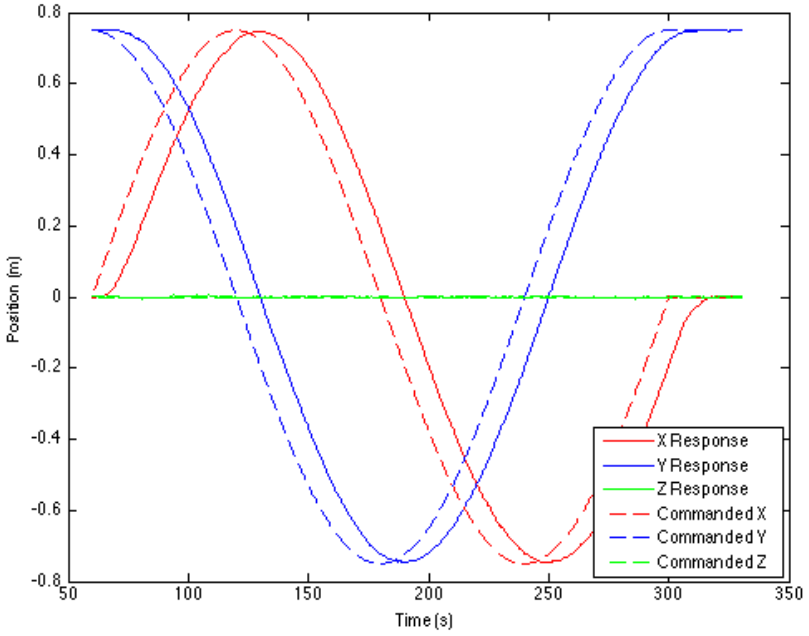


Figure B-2: Position tracking of a circular set of waypoints vs. time (in a 4-minute, 0.75m orbit)

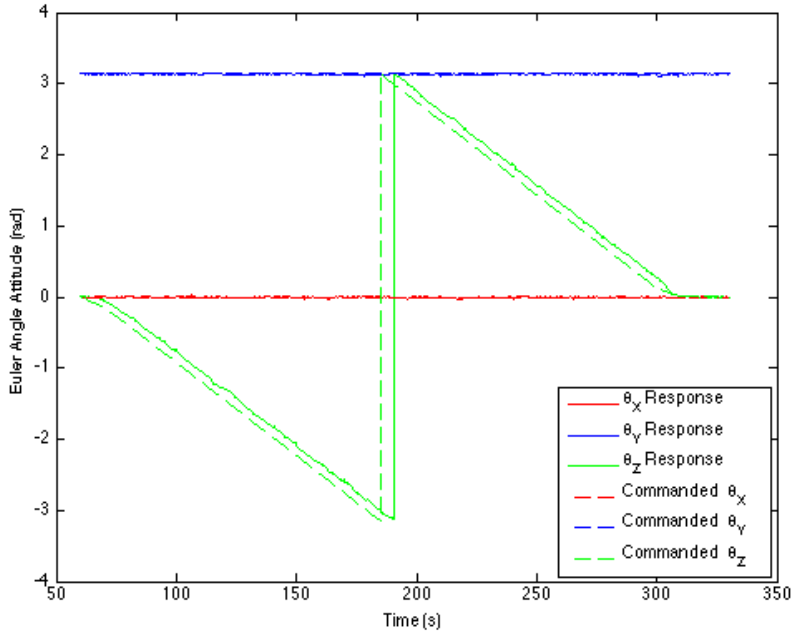


Figure B-3: Attitude tracking of commanded waypoints vs. time (in a 4-minute, 0.75m orbit)

The docking maneuver also performed very well as can be seen in Figure B-4. The rise time of the response was 13.6 seconds, with a max overshoot of 0.7% or 4 mm. There seems to be zero steady state error as well. This meets the requirements set for the docking tolerances. When compared to the standard SPHERES classical PD controller, the response of the LQR controller has a 3.2 second faster rise time and about an equivalent overshoot. The performance is better than the standard SPHERES controller, however, only because the fuel consumption is higher as seen in Figure B-5.

When comparing the controller overall to the standard SPHERES controller, we see very similar tracking performance as shown by Figures B-13 and B-14 in Section B.6 and somewhat quicker step response as seen in Figure B-4. However, in both cases the fuel consumption is greater for the LQR controller (Figure B-5 in this section and B-12 in Section B.6. Because fuel is a more valuable commodity than time on the International Space Station, this suggests that although the LQR controller has better performance in some sense, it is probably the best decision to continue using the standard SPHERES, classical PD controller in future test sessions.

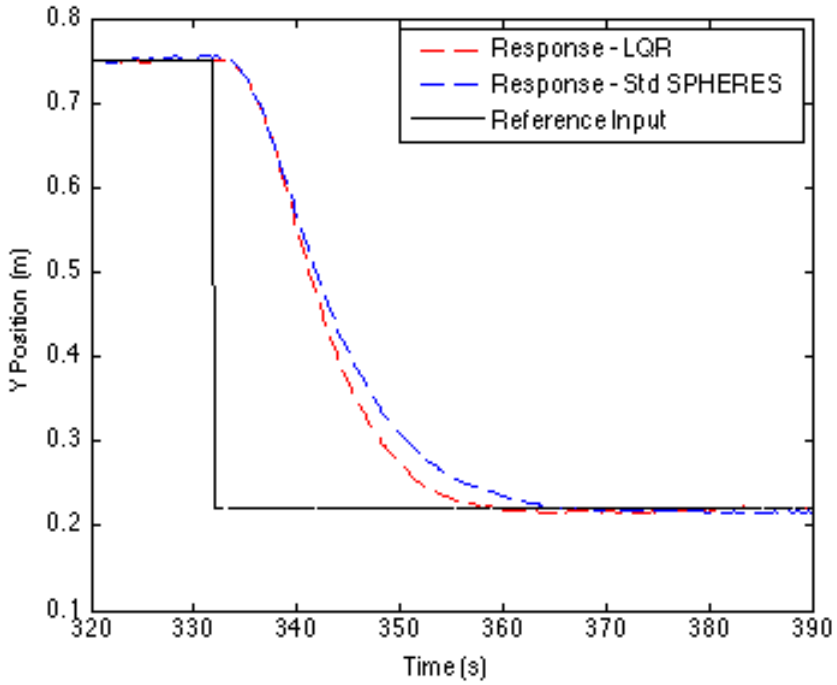


Figure B-4: Comparison of the step response of the LQR controller and the standard SPHERES controller during the 0.75m docking phase

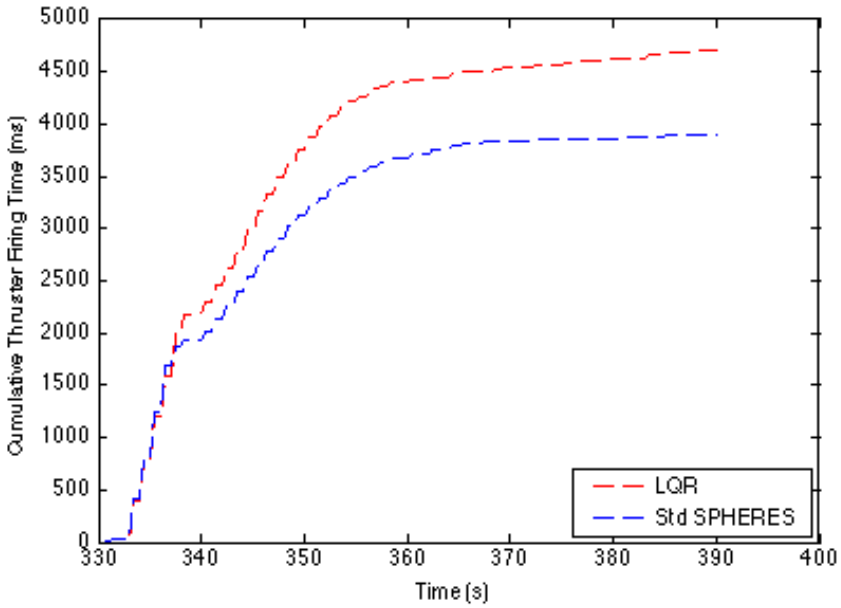


Figure B-5: Comparison of propellant usage between the standard SPHERES controller and the LQR controller during the docking step command of 0.75m

B.6 Additional Plots

This section includes a few additional plots that further detail some of the discussion points talked about in the main sections and are included for completeness.

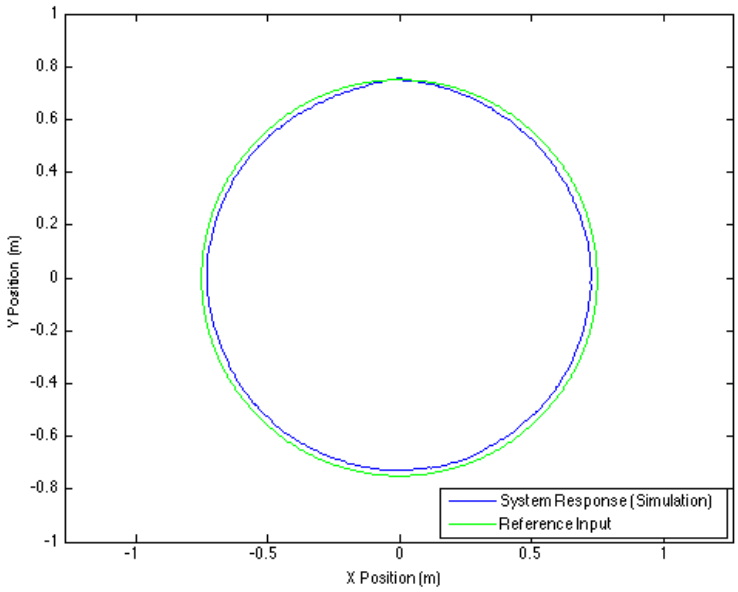


Figure B-6: Position tracking of a circular set of waypoints in a 2-minute, 0.75 m orbit around the target

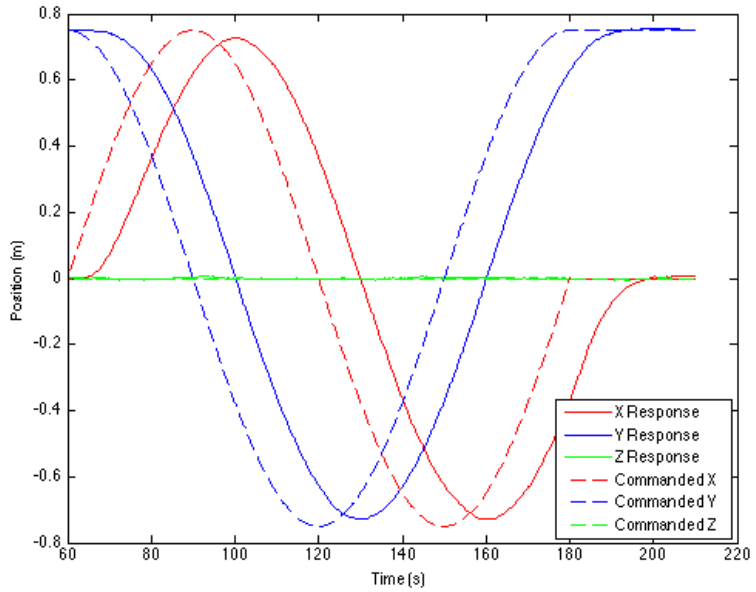


Figure B-7: Position tracking of a circular set of waypoints vs. time (in a 2-minute, 0.75 m orbit around the target)

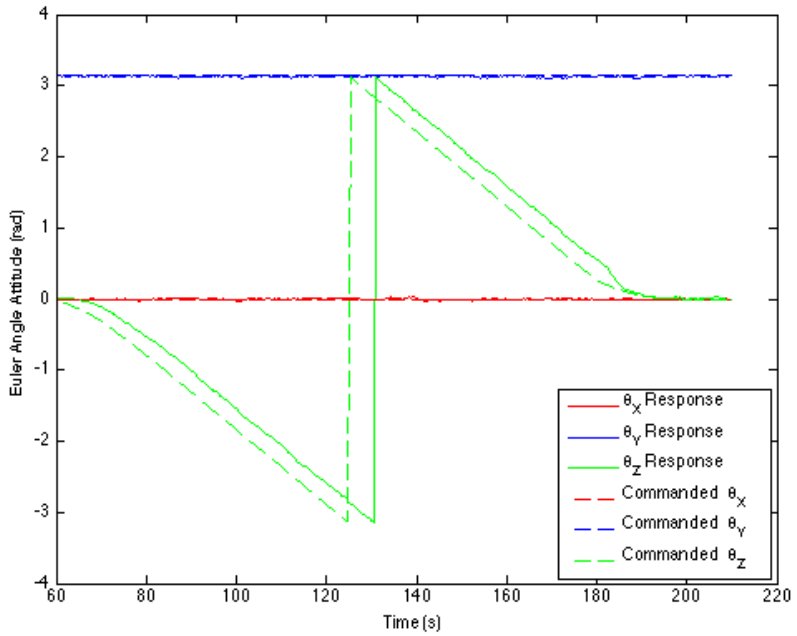


Figure B-8: Attitude tracking of commanded waypoints vs. time (in a 2-minute, 0.75 m orbit around the target)

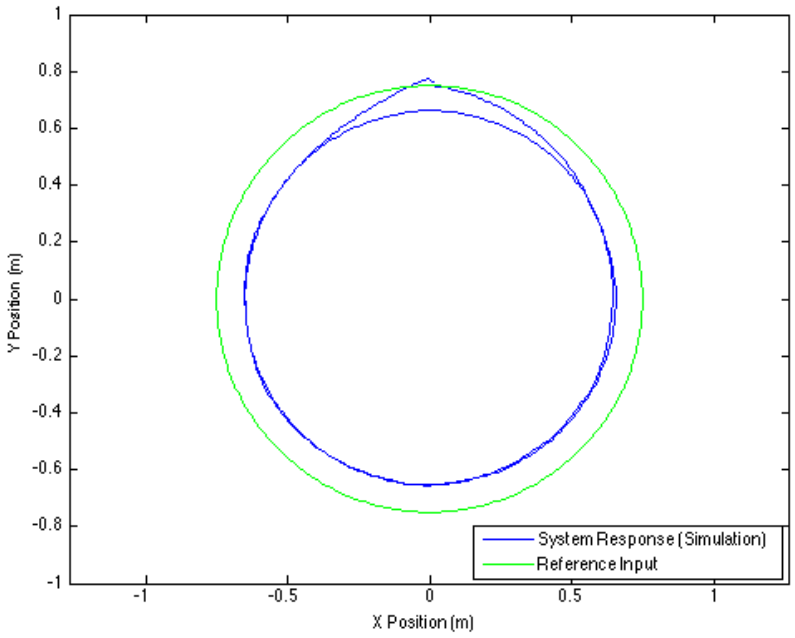


Figure B-9: Position tracking of a circular set of waypoints (in two 1-minute, 0.75 m orbits around the target)

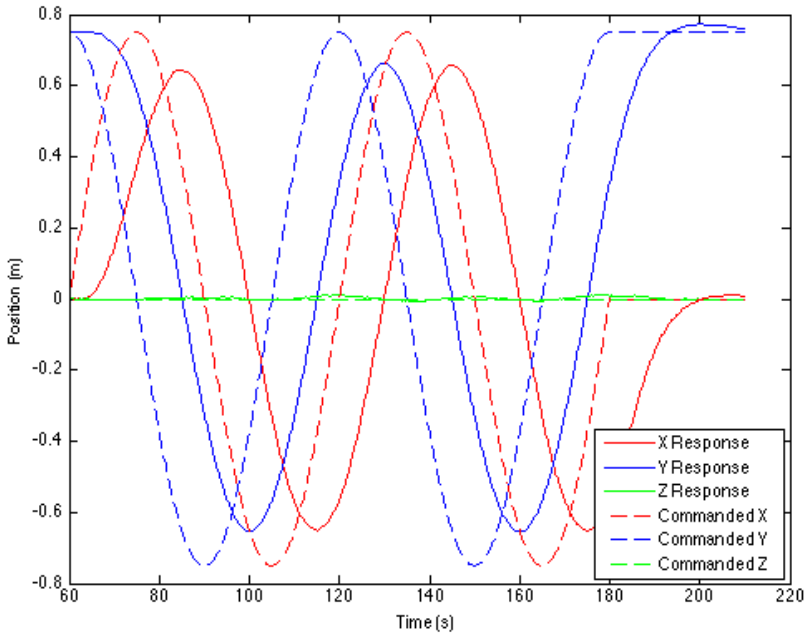


Figure B-10: Position tracking of a circular set of waypoints vs. time (in two 1-minute, 0.75 m orbits around the target)

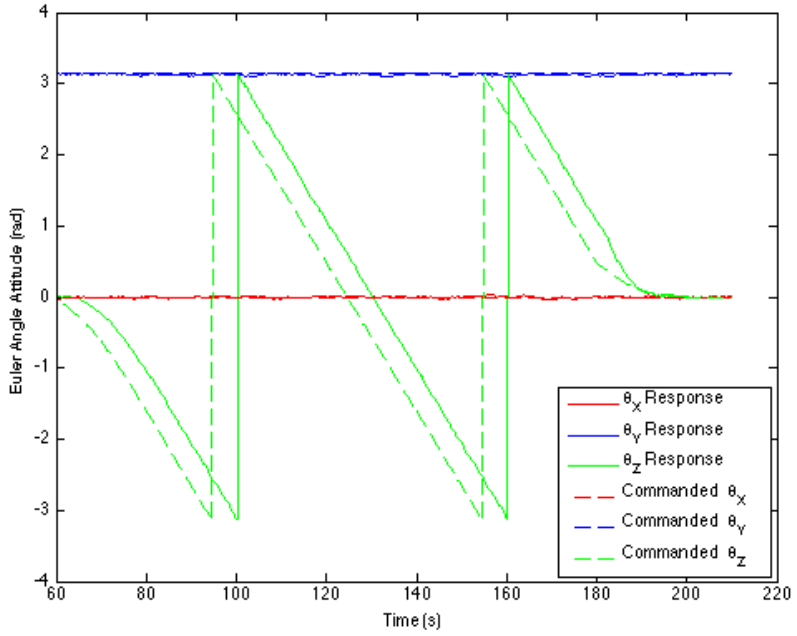


Figure B-11: Attitude tracking of commanded waypoints vs. time (in two 1-minute, 0.75 m orbits around the target)

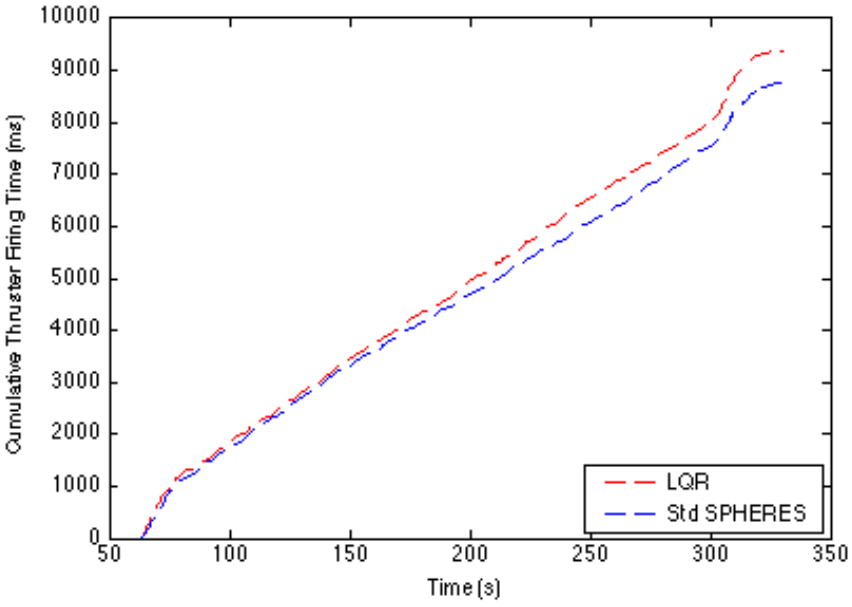


Figure B-12: Comparison of propellant usage between the standard SPHERES controller and the LQR controller during the 4-minute circumnavigation around the target satellite

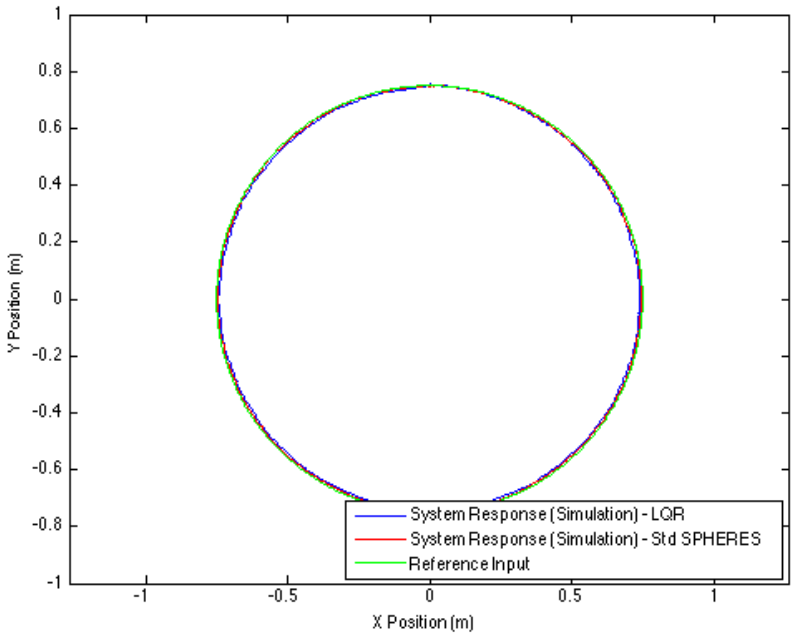


Figure B-13: Comparison of circular waypoint tracking between the standard and LQR controller during the 4-minute circumnavigation around the target satellite

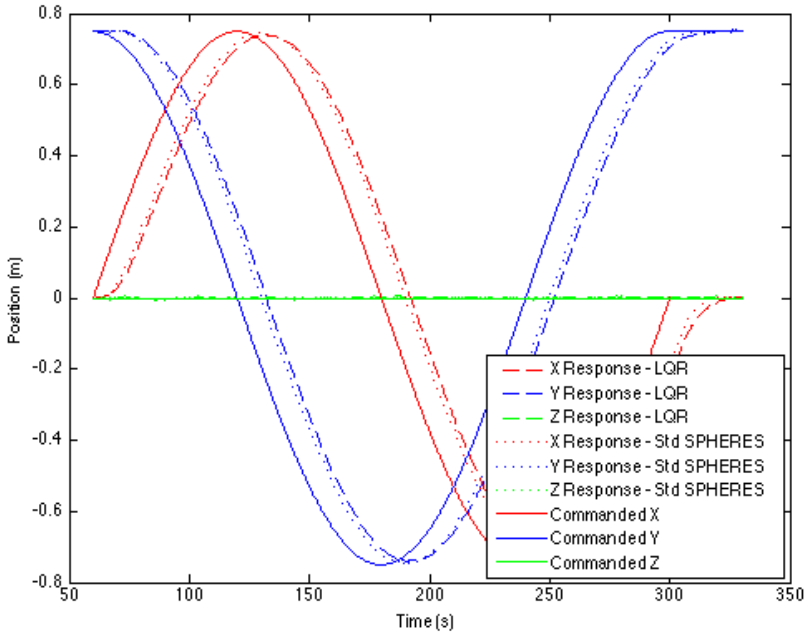


Figure B-14: Comparison of position tracking vs. time between the standard and LQR controller during the 4-minute circumnavigation around the target satellite

B.7 MATLAB® Code for Controller Development

../Code/AppB/spheres_lqr.m

```
%% spheres_lqr.m
% Chris Jewison
% jewisonc@mit.edu
% MATLAB Controller Development
% December 6, 2013

close all; clear all; clc;

%% Set up the problem, with constants and the plant dynamics
m = 4.3;          %satellite mass is 4.3 kg
Ixx = 0.02852;    % Inertia about x-axis
Iyy = 0.028317;   % Inertia about y-axis
Izz = 0.0245;     % Inertia about z-axis

A = zeros(12);
A(1,4) = 1;
A(2,5) = 1;
A(3,6) = 1;
A(7,10) = 1;
A(8,11) = 1;
A(9,12) = 1;

B = zeros(12,6);
B(4,1) = 1/m;
B(5,2) = 1/m;
B(6,3) = 1/m;
B(10,4:6) = [1/Ixx 0 0];
B(11,4:6) = [0 1/Iyy 0];
B(12,4:6) = [0 0 1/Izz];

%% Determine controllability
sys = ss(A,B,zeros(6,12),0);
Mc = ctrb(sys);
RANK_Mc = rank(Mc);
fprintf('Rank of the controllability matrix is: %d\n',RANK_Mc);

%% Set up Q and R matrices based on Bryson's Rules

max_angle = 2*pi/180; % 2 degree error req
max_rate = 1;   % 1 rad/s will saturate the gyros
max_pos = 0.015; % 1.5 cm error req
max_vel = 0.003;  % 0.3 cm/s to prevent large overshoot

Q = [sqrt(1/12)/max_pos^2   0 0 0 0 0 0 0 0 0 0 0;
     0 sqrt(1/12)/max_pos^2   0 0 0 0 0 0 0 0 0 0;
     0 0 sqrt(1/12)/max_pos^2   0 0 0 0 0 0 0 0 0;
     0 0 0 sqrt(1/12)/max_vel^2   0 0 0 0 0 0 0 0;
     0 0 0 0 sqrt(1/12)/max_vel^2   0 0 0 0 0 0 0;
     0 0 0 0 0 sqrt(1/12)/max_vel^2   0 0 0 0 0 0;
     0 0 0 0 0 0 sqrt(1/12)/max_angle^2 0 0 0 0 0;
     0 0 0 0 0 0 0 sqrt(1/12)/max_angle^2 0 0 0 0;
     0 0 0 0 0 0 0 0 sqrt(1/12)/max_angle^2 0 0 0;
     0 0 0 0 0 0 0 0 0 sqrt(1/12)/max_rate^2  0 0;
     0 0 0 0 0 0 0 0 0 0 sqrt(1/12)/max_rate^2  0;
     0 0 0 0 0 0 0 0 0 0 0 sqrt(1/12/max_rate^2)];

max_thrust = 0.04;  % 0.04 N saturates thrusters
max_torque = 0.004; % 0.004 Nm saturates thrusters
rho = 100;          % further penalize control to lower gains

R = rho*[sqrt(1/6)/max_thrust^2 0 0 0 0 0;
         0 sqrt(1/6)/max_thrust^2 0 0 0 0;
         0 0 sqrt(1/6)/max_thrust^2 0 0 0;
```

```
          0 0 0 sqrt(1/6)/max_torque^2 0 0;
          0 0 0 0 sqrt(1/6)/max_torque^2 0;
          0 0 0 0 0 sqrt(1/6)/max_torque^2];

%% Compute the K matrix through LQR and examine responses

K = lqr(A,B,Q,R);

%Copy text printed in this section directly into C code
fprintf(['float pos_p_x = %ff;\nfloat pos_p_y = %ff;\n'...
         'float pos_p_z = %ff;\nfloat vel_d_x = %ff;\n'...
         'float vel_d_y = %ff;\nfloat vel_d_z = %ff;\n'...
         'float att_p_x = %ff;\nfloat att_p_y = %ff;\n'...
         'float att_p_z = %ff;\nfloat rate_d_x = %ff;\n'...
         'float rate_d_y = %ff;\nfloat rate_d_z = %ff;\n'],...
         K(1,1),K(2,2),K(3,3),K(1,4),K(2,5),K(3,6),...
         K(4,7),K(5,8),K(6,9),K(4,10),K(5,11),K(6,12));

%% Create the closed loop system and examine step responses
Acl = A-B*K;
Bcl = B;
Ccl = eye(12);
Dcl = 0;

sys_cl = ss(Acl,Bcl,Ccl,Dcl);

step(sys_cl)
```

Appendix C

Reconfigurable Mixer Code

../Code/reconfigurable_mixer.m

```
function [ firing_times ] = reconfigurable_mixer(cmd, card, mode)
% %function [ firing_times ] = reconfigurable_mixer(cmd, card, mode)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 Function: reconfigurable_mixer                               %
%                                                                              %
%                 Author:   Chris Jewison, jewisonc@mit.edu                    %
%                           MIT Space Systems Laboratory                       %
%                 Date:     25 April 2014                                      %
%                                                                              %
% Description:                                                                 %
%   The function, reconfigurable_mixer, translates commanded forces and       %
%   torques from the controller into individual thruster firing times.        %
%   The mixer requires the required forces (x,y,z) and torques (x,y,z)        %
%   as input, as well as the business card of the system that specifies      %
%   the system properties (thruster locations and directions). The modes     %
%   input specifies whether the nominal, precision, fuel-efficient, or        %
%   agile actuation mode is desired from the mixer. The mixer works with      %
%   any number and location of thrusters, however every actuation mode        %
%   may not be compatible with certain configurations. Requires that          %
%   the system operates in six degrees of freedom.                            %
%                                                                              %
% Required files:                                                              %
%   reconfigurable_mixer.m                                                     %
%   merge_business_cards.m                                                     %
%   aggregate_cg.m                                                             %
%   aggregate_inertia.m                                                        %
%   aggregate_locations_and_directions.m                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%% Variable Definitions and Nomenclature
%   Complete definitions of variable in this function are listed below:
%
% INPUTS
%
% Variable      | Type          | Description
% --------------|---------------|------------------------------------------
% cmd           | 6x1 vector    | Vector of commanded forces and torques
%               |               |  [Fx; Fy; Fz; Tx; Ty; Tz]
% card          | struct        | Business card for system containing
%               |               |  fields m, rcg, rif, I, d_t, and r_t
%      m        | double        | Mass of module
%      rcg      | 3x1 vector    | Distance vector from the module's ref
```

```
%                |                |   point to its cg in its body frame
%       rif      |  3x1 vector    |  Distance vector from the module's ref
%                |                |   point to its active docking port
%       I        |  3x3 matrix    |  Inertia tensor of module
%       r_t      |  3xN matrix    |  Distance vectors from module's ref
%                |                |   point to each of its N thrusters
%       d_t      |  3xN matrix    |  Unit vectors in direction that each
%                |                |   of the module's N thrusters fire
%  mode          |  scalar        |  1 = nominal, 2 = precision,
%                |                |  3 = fuel-efficient, 4 = agile
%
%
%  OUTPUTS
%
%  Variable      |  Type          |  Description
%  --------------|----------------|-------------------------------------
%  firing_times  |  struct        |  Required firing times in ms for
%                |                |    individual thrusters with fields
%                |                |    on_times, off_times, length
%    on_times    |  Nx1 vector    |  On time for each thruster [ms]
%    off_times   |  Nx1 vector    |  Off times for each thruster [ms]
%    length      |  Nx1 vector    |  Total firing time required per
%                |                |  thruster [ms]

% Initializations for SPHERES system
duty_cycle = 0.4; % 40% duty cycle (nominal for ground, not ISS)
minPulseWidth  = 20; %ms
minControlWidth = 1; %ms
maxPulseWidth  = 1000*duty_cycle; %ms
halfWidth = ceil(maxPulseWidth/2);
thruster_force = 0.12*duty_cycle; % N
slopeForce     = maxPulseWidth/thruster_force; % ms/N
numThrusters  = size(card.d_t,2);
firing_times.length = zeros(numThrusters,1);
firing_times.on_time = zeros(numThrusters,1);
firing_times.off_time = zeros(numThrusters,1);


% Force that each individual thruster is able to produce on the system
force_thruster = card.d_t';

% Torque that each individual thruster produces on the system
torque_thruster = cross(card.r_t',card.d_t');

%Formulation of the inverse authority matrix
MixInv = [force_thruster torque_thruster];

%Pseudoinverse transformation to authority matrix
Mix = pinv(MixInv');

switch mode % Perform different algorithm based on acutation mode choice

    case {1,2}  % 1 = Nominal actuation mode
        % Nominal multiplication of authority matrix w/ thruster pairing

        %Determine the force required from each thruster to follow the
        %commanded forces and torques
        F_req = Mix*cmd;
        %Determine the firing time required from each thruster to follow
        %the commanded forces and torques
        t_req_temp = F_req*slopeForce;

        %Determine thruster pairs
        thruster_pairs = zeros(numThrusters,1); %initialize
        for ind1 = 1:numThrusters %for all thrusters
            if thruster_pairs(ind1)==0 %not paired already
                for ind2 = 1:numThrusters %for all other thrusters
```

```
                    if MixInv(ind1,:) == -MixInv(ind2,:) % opposite F/T
                        if isempty(find(thruster_pairs==ind2, 1)) ...
                                && (ind2 ~= ind1) %not already taken
                            thruster_pairs(ind1) = ind2;
                            thruster_pairs(ind2) = ind1;
                            break;
                        end
                    end
                end
            end
        end

        %Assign appropriate firing times to each pair (i.e., move all
        %negative times to the other thruster in the pair
        t_req = t_req_temp*0; %initialize
        for ind3 = 1:length(thruster_pairs) % for all thrusters
            if t_req_temp(ind3) ~= 0 && ...
                    t_req_temp(thruster_pairs(ind3)) ~= 0
                if t_req_temp(ind3) > 0 %if pos, the other is neg
                    t_req(ind3) = t_req_temp(ind3) - ...
                        t_req_temp(thruster_pairs(ind3));
                    t_req_temp(thruster_pairs(ind3)) = 0;
                    t_req_temp(ind3)=0;
                elseif t_req_temp(ind3) < 0 % if neg, the other is pos
                    t_req(thruster_pairs(ind3)) = -t_req_temp(ind3) ...
                        + t_req_temp(thruster_pairs(ind3));
                    t_req_temp(thruster_pairs(ind3)) = 0;
                    t_req_temp(ind3)=0;
                end
            end
        end


        if mode == 2  % 2 = Precision actuation mode
            %Decreasing the deadband to machine cycle vs. minPulseWidth

            %Scale required firing times to not exceed maxPulseWidth
            % while preserving direction for the other thrusters
            max_t_req = max(t_req);
            if max_t_req > maxPulseWidth
                scale = maxPulseWidth/max_t_req;
                t_req = t_req*scale;
            end

            % Determine which thruster need precise actuation
            under_thrustdeadband = t_req < minPulseWidth;
            over_controldeadband = t_req >= minControlWidth;
            precision_required = over_controldeadband & ...
                under_thrustdeadband;

            % Fire the opposing thruster in the pair to reduce deadband
            t_req(precision_required) = t_req(precision_required) + ...
                minPulseWidth;
            t_req(thruster_pairs(precision_required)) = ...
                t_req(thruster_pairs(precision_required)) ...
                + minPulseWidth;
        end

    case 3  % 3 = Fuel-efficient actuation mode
        % Solving a linear optimization problem to minimize prop use
        fo = Mix*cmd;
        %Set the linear inequality constraints (positive forces only)
        A = -eye(size(fo,1));
        B = zeros(size(fo,1),1);
        %Set the linear equality constraints (meet cmd)
        Aeq = MixInv';
        Beq = cmd;

```

```
180         %Create the objective function
181         c = ones(size(fo,1),1); %Cost vector
182         fueleff_obj = @(f) c'*f; %Objective function
183 
184         %Set optimization options
185         options = optimoptions('fmincon','Algorithm','active-set',...
186             'Display','off');
187 
188         %Fuel-optimally determine best thrusters to fire for how long
189         F_req = fmincon(fueleff_obj,fo,A,B,Aeq,Beq,[],[],[],options);
190         t_req = F_req*slopeForce;
191 
192 
193     case 4 % 4 = Agile actuation mode
194         %Enabling full firing of all thrusters to provide the max
195         %possible force/torque on the system [requires the system to be
196         %able to be formulated with 6 "thruster pairs"]
197 
198         % First determine the thruster pairs
199         thruster_pairs = zeros(numThrusters,1);
200         thruster_pair_num = zeros(numThrusters,1);
201         pair = 0;
202         for ind1 = 1:numThrusters
203             if thruster_pairs(ind1)==0
204                 for ind2 = 1:numThrusters
205                     if MixInv(ind1,:) == -MixInv(ind2,:)
206                         if isempty(find(thruster_pairs==ind2, 1)) ...
207                                 && (ind2 ~= ind1)
208                             thruster_pairs(ind1) = ind2;
209                             thruster_pairs(ind2) = ind1;
210                             pair = pair + 1;
211                             thruster_pair_num(ind1) = pair;
212                             thruster_pair_num(ind2) = pair;
213                             break;
214                         end
215                     end
216                 end
217             end
218         end
219 
220         %Create the condensed matrix by combining the pairs into single
221         %thrusters capable of pos and neg thrusting
222         num_pairs = numThrusters/2;
223         MixInv_condensed = zeros(num_pairs,6);
224         condensed = zeros(1,numThrusters);
225         pair = 0;
226         for ind3 = 1:numThrusters
227             if condensed(ind3)==0
228                 pair = pair + 1;
229                 condensed(ind3)=1;
230                 condensed(thruster_pairs(ind3))=1;
231                 if sum(MixInv(ind3,1:3)) > 0
232                     MixInv_condensed(pair,:) = MixInv(ind3,:);
233                 else
234                     MixInv_condensed(pair,:) = ...
235                         MixInv(thruster_pairs(ind3),:);
236                 end
237             end
238         end
239 
240         %Determine torque groups and combine them into fully reduced
241         %authority matrix (6x6)
242         if size(MixInv_condensed,1)~=6
243             MixInv_torques = ...
244                 full(spones(MixInv_condensed)).*sign(MixInv_condensed);
245             torque_groups = zeros(num_pairs,7);
246             group_num = 0;
247             members = zeros(num_pairs,1);
```

```
248             MixInv_grouped = zeros(6,6);
249             for ind4 = 1:num_pairs
250                 if torque_groups(ind4)==0
251                     group_num = group_num + 1;
252                     for ind5 = ind4:num_pairs
253                         if MixInv_torques(ind4,:) == MixInv_torques(ind5,:)
254                             %if members(group_num)>=num_pairs/6;
255                             %    break;
256                             %end
257                             torque_groups(ind5,1) = group_num;
258                             torque_groups(ind5,2:7) = MixInv_condensed(ind5,:);
259                             members(group_num) = members(group_num) + 1;
260                         end
261                     end
262                 end
263             end
264             num_pairs_in_group = zeros(group_num,1);
265             for ind8 = 1:group_num
266                 pairs_in_group = find(torque_groups(:,1)==ind8);
267                 num_pairs_in_group(ind8) = length(pairs_in_group);
268                 moment_arm = 0;
269                 for ind9 = 1:num_pairs_in_group(ind8)
270                     moment_arm = moment_arm + ...
271                         abs(norm(MixInv_condensed(pairs_in_group(ind9),4:6)));
272                 end
273                 %moment_arm = moment_arm/num_pairs_in_group;
274                 MixInv_grouped(ind8,:) = ...
275                   [num_pairs_in_group(ind8)*MixInv_torques(pairs_in_group(1),1:3) ...
276                                 moment_arm*MixInv_torques(pairs_in_group(1),4:6)];
277             end
278
279
280         else % Condensed mixer is already in a 6x6 form, so just use
281             % the condensed matrix and fake the torque groups
282             MixInv_grouped = MixInv_condensed;
283             torque_groups = [ [1;2;3;4;5;6] MixInv_condensed];
284         end
285
286         %Pseudoinverse transformation to authority matrix
287         Mix_grouped = pinv(MixInv_grouped');
288
289         %Determine the force required from each thruster to follow the
290         %commanded forces and torques
291         F_req = Mix_grouped*cmd;
292         %Determine the firing time required from each thruster to follow
293         %the commanded forces and torques
294         t_req_temp = F_req*slopeForce;
295
296
297         %Assign appropriate firing times to each pair (i.e., move all
298         %negative times to the other thruster in the pair
299         t_req = zeros(numThrusters,1); %initialize
300         for ind6 = 1:length(F_req) % for all thruster groups
301             if t_req_temp(ind6) ~= 0
302                 pair_nums = find(torque_groups(:,1)==ind6)';
303                 for ind7 = pair_nums
304                     thrust_num = find(thruster_pair_num==ind7,1);
305                     if sign(sum(MixInv(thrust_num,1:3))) == ...
306                             sign(t_req_temp(ind6))
307                         t_req(thrust_num) = ...
308                             abs(t_req_temp(ind6));
309                     else
310                         t_req(thruster_pairs(thrust_num)) = ...
311                             abs(t_req_temp(ind6));
312                     end
313                 end
314             end
315         end
```

```
316 end
317
318 %Scale required firing times to not exceed maxPulseWidth while
319 %preserving direction for the other thrusters
320 max_t_req = max(t_req);
321 if max_t_req > maxPulseWidth
322     scale = maxPulseWidth/max_t_req;
323     t_req = t_req*scale;
324 end
325
326 %Implement deadband
327 under_deadband = t_req < minPulseWidth;
328 t_req(under_deadband) = 0;
329 almost_full = 1.05*t_req > maxPulseWidth;
330 t_req(almost_full) = maxPulseWidth;
331
332 %Determine on and off times as centered pulses
333 firing_times.length = ceil(t_req);
334 firing_times.on_time = halfWidth - floor(t_req/2);
335 firing_times.off_time = halfWidth + ceil(t_req/2);
336 firing_times.on_time(under_deadband) = 0;
337 firing_times.off_time(under_deadband) = 0;
```

Bibliography

[1] Mark Baldesarra and David W. Miller. A Decision-Making Framework to Determine the Value of On-Orbit Servicing Compared to Replacement of Space Telescopes. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2007.

[2] David Barnhart, Lisa Hill, Margaret Turnbull, and Peter Will. Changing satellite morphology through cellularization. In *AIAA SPACE 2012 Conference and Exposition*, Pasadena, CA, 2012.

[3] David Barnhart and Brook Sullivan. Economics of repurposing in situ retired spacecraft components. In *AIAA SPACE 2012 Conference and Exposition*, Pasadena, CA, 2012.

[4] David Barnhart, Brook Sullivan, et al. Phoenix project status 2013. In *AIAA SPACE 2013 Conference and Exposition*, San Diego, CA, 2013.

[5] Edward V. Bergmann, S.R. Croopnick, J.J. Turkovich, and C.C. Work. An advanced spacecraft autopilot concept. *Journal of Guidance and Control*, 2(3):161, May-June 1979.

[6] Jovan D. Boskovic, Sai-Ming Li, and Raman K. Mehra. Reconfigurable flight control design using multiple switching controllers and on-line estimation of damage-related parameters. In *Proceedings of the 2000 IEEE International Conference on Control Applications*, Anchorage, AK, 2000.

[7] Allen Chen and David Miller. Propulsion system characterization for the spheres formation flight and docking testbed. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2002.

[8] DARPA. $http://www.darpa.mil/Our\_Work/TTO/Programs/Phoenix.aspx$, 2012. Accessed: 2013-01-15.

[9] John B. Davidson, Frederick J. Lallman, and W. Thomas Bundick. Integrated reconfigurable control allocation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2001.

[10] Nitin Dhayagude and Zhiqiang Gao. A novel approach to reconfigurable control system design. *AIAA Journal of Guidance, Control and Dynamics*, 19(4):963–966, 1996.

[11] Jonathan P. Gardner et al. The James Webb Space Telescope. *Space Science Reviews*, 123(4):485–606, 2006.

[12] Mark Ole Hilstad and David Miller. A multi-vehicle testbed and interface framework for the development and verification of separated spacecraft control algorithms. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2002.

[13] Matthew M. Jeffrey and Jonathan How. Closed-loop control of spacecraft formations with applications on spheres. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2008.

[14] Christopher Jewison and Bryan McCarthy. Halo System Requirements Review Presentation. Presented to DARPA for the InSPIRE-II program on September 26, 2013.

[15] Christopher Jewison, Bryan McCarthy, David Sternberg, Daniel Strawser, and Cheng Fang. Resource aggregated reconfigurable control and risk-allocative path planning for on-orbit servicing and assembly of satellites. In *AIAA Guidance, Navigation, and Control Conference*, National Harbor, MD, 2014.

[16] Jacob Katz and David W. Miller. Estimation and Control of Flexible Space Structures for Autonomous On-Orbit Assembly. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2009.

[17] Jacob G. Katz and Alvar Saenz-Otero. *Achieving Broad Access to Satellite Control Research with Zero Robotics*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2013.

[18] Edward A. LeMaster, David B. Schaechter, and Connie K. Carrington. Experimental demonstration of technologies for autonomous on-orbit robotic assembly. In *Space 2006*, San Jose, CA, 2006.

[19] Franck Martel. Optimal 6 axis command of a space vehicle with a precomputed thruster selection catalogue table. In *International Symposium on Space Flight Dynamics*, Munich, Germany, 2004.

[20] Frederick H. Martin and Richard H. Battin. Computer-controlled steering of the apollo spacecraft. *Journal of Spacecraft*, 5(4):400–407, April 1968.

[21] Peter S. Maybeck and Richard D. Stevens. Reconfigurable flight control via multiple model adaptive control methods. *IEEE Transactions on Aerospace and Electronic Systems*, 27(3):470–480, May 1991.

[22] Bryan McCarthy and Christopher Jewison. Halo Prototype Design Review Presentation. Presented to DARPA for the InSPIRE-II program on April 11, 2014.

[23] Bryan McCarthy and Roedolph Opperman. Halo Design Document. A living, breathing document with the most recent version located at http://ssl.mit.edu/svn/spheres/trunk/Missions/Halo/Documentation.

[24] Bryan McCarthy, Alvar Saenz-Otero, and David Miller. Development of a space-based robotic assembly and servicing testbed. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2014.

[25] MIT SSL SPHERES. ARMADAS Proposal. MIWG Proposal to USAF SMC, 2012.

[26] Swati Mohan and David Miller. *Quantitative Selection and Design of Model Generation Architectures for On-Orbit Autonomous Assembly*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2010.

[27] NASA SPHERES. $http://www.nasa.gov/images/content/179781main\_SPHERES12.jpg$, 2007. Accessed: 2013-04-07.

[28] Alexander G. Parlos and John W. Sunkel. Adaptive attitude stability and control for space station/orbiter berthing operations. In *AIAA Guidance, Navigation and Control Conference*, 1992.

[29] Christopher Masaru Pong and David Miller. Autonomous thruster failure recovery for underactuated spacecraft. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, September 2010.

[30] Marc Postman et al. Advanced Technology Large-Aperature Space Telescope (ATLAST). A NASA Astrophysics Strategic Mission Concept Study, Space Telescope Science Institute, May 2009.

[31] Santosh Ratan and Neil Evan Goodzeit. *Universal Thruster Selection Logic for Spacecraft Attitude Control, U.S. Patent No. 5,646,847*. U.S. Patent and Trademark Office, Washington, D.C., 1997.

[32] Lennon Rodgers and David W. Miller. Concepts and Technology Development for the Autonomous Assembly and Reconfiguration of Modular Space Systems. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, December 2005.

[33] Uday J. Shankar and Kidambi V. Raman. *Unified Spacecraft Attitude Control System, U.S. Patent No. 5,140,525*. U.S. Patent and Trademark Office, Washington, D.C., 1992.

[34] David Shoemaker. A survey of spacecraft jet selection logic algorithms. In *36th Annual American Astronautical Society Guidance and Control Conference*, Breckenridge, CO, 2013.

[35] David Sternberg, Bryan McCarthy, and Christopher Jewison. MIT Project Phoenix Testing. Testing report submitted to Aurora Flight Sciences on August 23, 2013.

[36] David C. Sternberg, David Miller, and Alvar Saenz-Otero. Development of an incremental and iterative risk reduction facility for robotic servicing and assembly missions. SM thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2014.

[37] STS-129. S129E009243. $http://spaceflight.nasa.gov/gallery/images/shuttle/sts-129/hires/s129e009243.jpg$, November 2009. Accessed: 2013-04-14.

[38] Brook R. Sullivan. *Technical and Economic Feasibility of Telerobotic On-Orbit Satellite Servicing*. PhD dissertation, University of Maryland, Department of Aeronautical and Astronautical Engineering, 2005.

[39] Steve Ulrich, Dustin Luke Hayhurst, Alvar Saenz-Otero, David W. Miller, and Itzhak Barkana. Simple adaptive control for spacecraft proximity operations. In *AIAA Guidance, Navigation, and Control Conference*, National Harbor, MD, 2014.

[40] W. S. Widnall, A. L Kosmala, F. H. Martin, R. H. Battin, D. G. Hoag, and R. R. Ragan. Apollo Guidance, Navigation and Control: Guidance system operations plan for manned CM Earth orbital and lunar missions using program Colossus I and program Colossus IA. Technical Report R-577, Massachusetts Institute of Technology Instrumentation Laboratory, Cambridge, MA, December 1968.

[41] Bong Wie. *Space Vehicle Dynamics and Controls*, chapter 7.6: Optimal Jet Selection. American Institute of Aeronautics and Astronautics, 2008.

[42] Edward Wilson, Chris Lages, and Robert Mah. On-line, gyro-based, mass-property identification for thruster-controlled spacecraft using recursive least squares. In *Proceedings of 45th IEEE International Midwest Symposium on Circuits and Systems*, Tulsa, OK, 2002.