MULTI-AGENT ROBOTIC SYSTEMS AND APPLICATIONS FOR SATELLITE MISSIONS

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAIʻI AT MĀNOA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTORATE OF PHILOSOPHY

IN

MECHANICAL ENGINEERING

MAY 2016

By

Miguel A. Nunes

Dissertation Committee:

Brian Bingham, Chairperson
Dilmurat Azimov
Luke Flynn
Trevor Sorensen
A Zachary Trimble

Keywords: Multi-Agent Robotic Systems, Satellite Attitude Control, Automated Rendezvous and Docking, Guidance Navigation and Control, Satellite Constellation

"*Each of us is here for a brief sojourn; for what purpose he knows not, though he senses it. But without deeper reflection one knows from daily life that one exists for other people.*"

Albert Einstein (1879 - 1955)

To my dear wife, Mara,

who offered me unconditional love and support throughout the course of this dissertation. Simply put I could not have done this without you!

To my beloved family,

that has always supported and motivated my education.

ACKNOWLEDGMENTS

Sometimes people say "I stand in the shoulder of giants". I fell this is specially true for the outcome of this dissertation. Being the first Ph.D. student at the Hawaii Space Flight Laboratory was no easy task, but it would just not have happened without the support of the following people.

First, I want to specially thank my committee members. Dr. Brian Bingham for guiding me through this academic exercise and for helping me to focus on the critical paths to get the Ph.D. done. I cannot be thankful enough for his patience, tremendous support and friendship. Dr. Dilmurat Azimov who has given me invaluable advice on special topics of GNC which made their way towards this dissertation. Dr. Luke Flynn who has given me the opportunity to work with the Hawaii Space Flight Laboratory and who secured the funds to support this research. Without Dr. Flynn I simply could not have done this Ph.D. His vision to develop a space program at the University of Hawaii have inspired me throughout these years. Dr. Trevor Sorensen who has given me a tremendous support and who originally designed COSMOS. Most of the work in this dissertation is based on COSMOS. Without it I could have not implemented the flight software and test it on the satellite testbed. Without Dr. Sorensen we would not have acquired the state of the art satellite attitude control testbed that was instrumental in this research. Dr. A Zachary Trimble for his willingness to accept some projects for his Senior Design Class which turned out to have a significant impact on this dissertation, specially for the magnetic torque rod and Helmholtz coils projects.

I also want to thank my HSFL team mates: Jeremy Chan, Amber Imai, Eric Pilger, Isaac Rodrigues, Lance Yoneshigue. Together we finished building HiakaSat, the first satellite to be launched from Hawaii. Of special mention regarding this dissertation. I'm specially thankful to Eric for the countless hours he has worked with me on the HiakaSat ADCS, the satellite testbed and the COSMOS software. Without Eric this work would not have been

done either. Jeremy, for the amazing help while I was building the satellite testbed and also for working on the control tools to operate the testbed. Isaac, for patiently proofreading this document.

Last but definitely not the least. I want to thank my loving, patient, supportive, sweet and dear wife, Mara. This work would also not have been possible without her. And finally my little and dear son Matias - who actually did not help much on this work, quite the contrary. But has changed my life for the better. I love you, man! My gratitude would not be complete without also thanking my Mamie, Papi, Mummi, Lisete, Carlos, and Dany, who gave so much for me and always supported my education. Finally, I want to thank my mum and dad. Even with the tremendous challenges we are facing as a family, I love them dearly.

ABSTRACT

A revolution in the space sector is happening. It is expected that in the next decade there will be more satellites launched than in the previous sixty years of space exploration. Major challenges are associated with this growth of space assets such as the autonomy and management of large groups of satellites, in particular with small satellites. There are two main objectives for this work. First, a flexible and distributed software architecture is presented to expand the possibilities of spacecraft autonomy and in particular autonomous motion in attitude and position. The approach taken is based on the concept of distributed software agents, also referred to as multi-agent robotic system. Agents are defined as software programs that are social, reactive and proactive to autonomously maximize the chances of achieving the set goals. Part of the work is to demonstrate that a multi-agent robotic system is a feasible approach for different problems of autonomy such as satellite attitude determination and control and autonomous rendezvous and docking. The second main objective is to develop a method to optimize multi-satellite configurations in space, also known as satellite constellations. This automated method generates new optimal mega-constellations designs for Earth observations and fast revisit times on large ground areas. The optimal satellite constellation can be used by researchers as the baseline for new missions.

The first contribution of this work is the development of a new multi-agent robotic system for distributing the attitude determination and control subsystem for HiakaSat. The multi-agent robotic system is implemented and tested on the satellite hardware-in-the-loop testbed that simulates a representative space environment. The results show that the newly proposed system for this particular case achieves an equivalent control performance when compared to the monolithic implementation. In terms on computational efficiency it is found that the multi-agent robotic system has a consistent lower CPU load of 0.29 ± 0.03 compared

to 0.35 $\pm$ 0.04 for the monolithic implementation, a 17.1 % reduction.

The second contribution of this work is the development of a multi-agent robotic system for the autonomous rendezvous and docking of multiple spacecraft. To compute the maneuvers guidance, navigation and control algorithms are implemented as part of the multi-agent robotic system. The navigation and control functions are implemented using existing algorithms, but one important contribution of this section is the introduction of a new six degrees of freedom guidance method which is part of the guidance, navigation and control architecture. This new method is an explicit solution to the guidance problem, and is particularly useful for real time guidance for attitude and position, as opposed to typical guidance methods which are based on numerical solutions, and therefore are computationally intensive. A simulation scenario is run for docking four CubeSats deployed radially from a launch vehicle. Considering fully actuated CubeSats, the simulations show docking maneuvers that are successfully completed within 25 minutes which is approximately 30% of a full orbital period in low earth orbit.

The final section investigates the problem of optimization of satellite constellations for fast revisit time, and introduces a new method to generate different constellation configurations that are evaluated with a genetic algorithm. Two case studies are presented. The first is the optimization of a constellation for rapid coverage of the oceans of the globe in 24 hours or less. Results show that for an 80 km sensor swath width 50 satellites are required to cover the oceans with a 24 hour revisit time. The second constellation configuration study focuses on the optimization for the rapid coverage of the North Atlantic Tracks for air traffic monitoring in 3 hours or less. The results show that for a fixed swath width of 160 km and for a 3 hour revisit time 52 satellites are required.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The first main motivation for this research is the development of technologies to improve the autonomy of future space missions, in particular small satellite missions. The National Aeronautics and Space Administration (NASA) Technology Roadmap for Robotics and Autonomous Systems (TA4)[1], which defines the pathways for this sector of technology research and development for the next 20 years (2015-2035), emphasizes that "*autonomous systems will reduce the cognitive load on humans given the abundance of information that has to be reasoned upon in a timely fashion*" and "*smarter and more agile space robots will be better equipped to sense and react to anomalies onboard, making them less dependent on the ground crew.*" There is a broad spectrum of future missions that will require full autonomy. One particular example are Earth observation satellites, specially if they are part of a coordinated constellation and quick response times are required to observe a site on the ground. Other examples of missions that will require a high level of autonomy are, large space structures assembly missions, satellite inspections and satellite on-orbit servicing missions. Small satellites will be critical in the future development of all these categories. One of the key aspects behind this thesis is the enablement of small satellites missions. The first section is focused on the development of autonomy for a small satellite. The second section is focused on the cooperative coordination of small satellites, and the last section is concerned with satellite constellations using small satellites.

The other motivation for this work is the advent of space missions using a large number of satellites in coordination for applications that cannot be done using monolithic satellites. In the last decade there has been a growing interest in distributed space architectures mostly because of the development of small satellites. The reduced size and cost with increased

performance of such platforms are opening new possibilities for the scientific, commercial and defense space. Figure 1.1 shows the number of currently operational small satellites (< 180 kg) compared to the number of operational large satellites (≥ 180 kg). This figure clearly shows that in 2014 the number of small satellites surpassed the existing number of large satellites that are still in operation[1]. This growing trend for small satellites is expected to continue.

Related to the growing interest in small satellite technologies, several other missions have motivated the interest to distribute space assets for the future of space exploration. Notable examples are the DARPA Orbital Express mission launched in 2007, the PRISMA mission in 2010 and TanDEM-X also launched in 2010. These missions were flown and demonstrated the applicability of distributed and coordinated space architectures. Other related mission concepts have been discussed such as DARPA's System F6 (Future Fast, Flexible, Free-Flying, Fractionated Spacecraft united by Information eXchange)[2]. The F6 program was proposed to demonstrate that fractionated satellite capabilities can lower the cost and increase responsiveness and effectiveness of space missions using small and fractioned satellite subsystems[3]. It is simpler to maintain and replace a small satellite than a large monolithic one. Although the fractionated spacecraft concept is very promising it has not been fully demonstrated in operation. In 2015 SpaceX and OneWeb also announced the deployment of large satellite constellations for broadband Internet around the globe in the next decade. These new ventures are expected to launch more satellites in the next 10 years than were ever launched in the past 60 years, since the beginning of space exploration. All these factors contribute to the main motivation of this work to enable technologies for the study and

[1]The number of satellites is calculated with the existing information of mass for satellites from the USC satellite database `http://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database`. There are in fact more satellites launched per year but no mass information is reported and or the satellites are not operational (decommissioned, reentered the atmosphere, etc.).

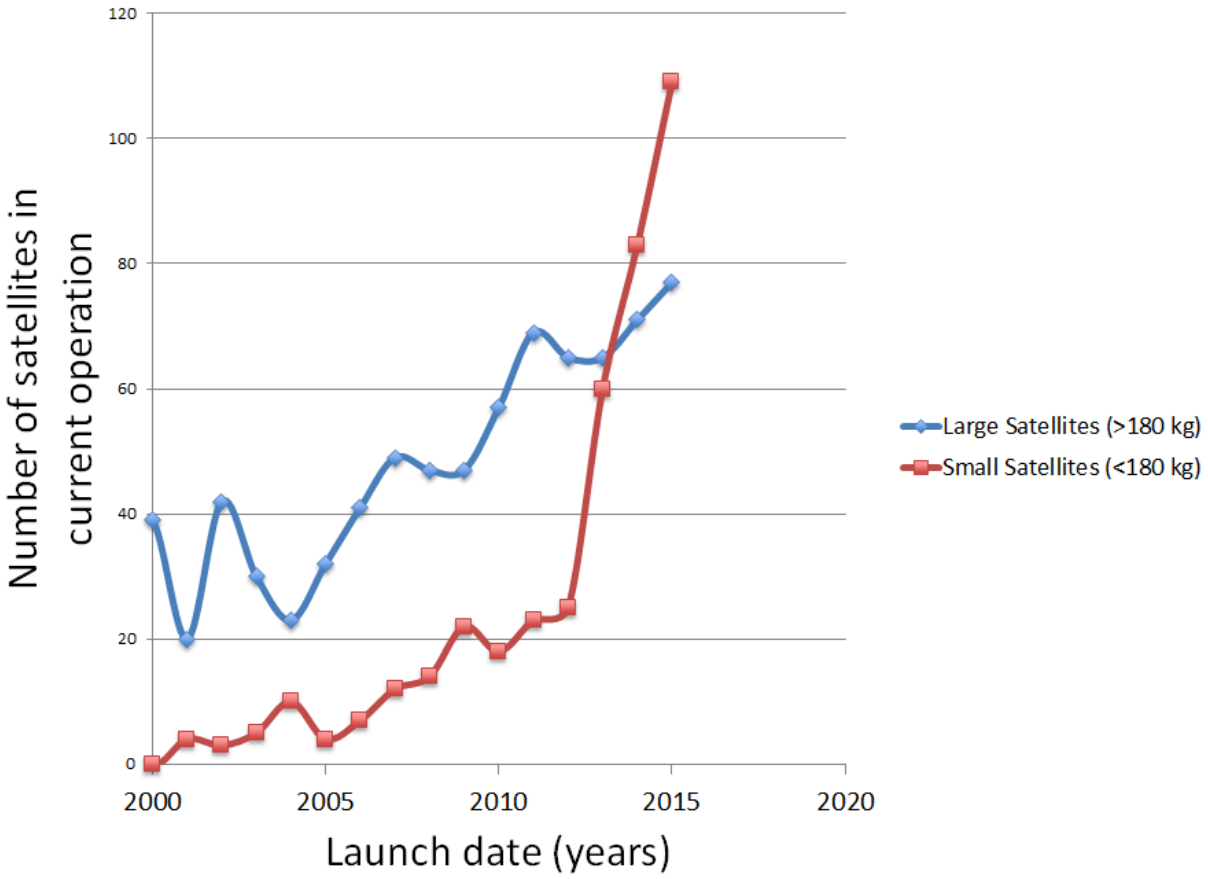

Figure 1.1: Number of currently operational small satellites ($<$ 180kg) compared to the number of operational large satellites ($\geq$ 180 kg) since the year 2000.

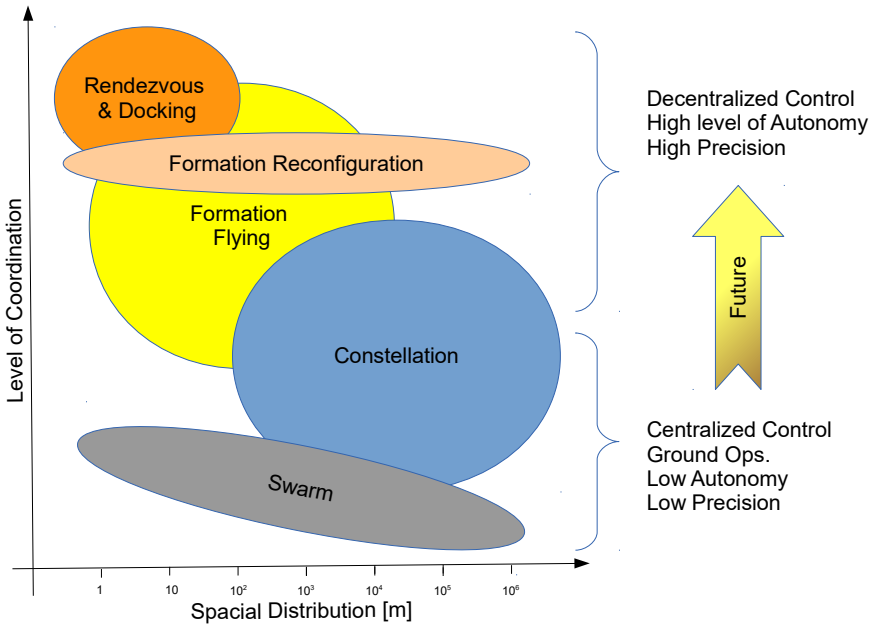

Figure 1.2: Relation of spacial distribution, level of coordination and sophistication for future distributed satellite architectures

deployment of small and distributed space architectures.

Figure 1.2 shows in a conceptual representation the evolution of distributed space architectures. In particular, the main goals for such missions are augmenting autonomy and increasing the level of coordination and precision flying. This work is a precursor for enabling autonomous distributed space architectures, and proposes that multi-agent robotic system (MARS) may be one of the fundamental aspects to enable future autonomous and distributed architectures in space. This is done by showing that systems operated by multiple autonomous agents, also known as MARS, are able to expand the autonomous capabilities and interact with the surrounding environment in a very robust way - which is necessary when operating in space. The goal is to enable advanced and autonomous missions with limited ground support. This work is related to the DARPA F6 concept because a MARS is one of the software aspect that can enable a mission such as the F6. The F6 concept fo-

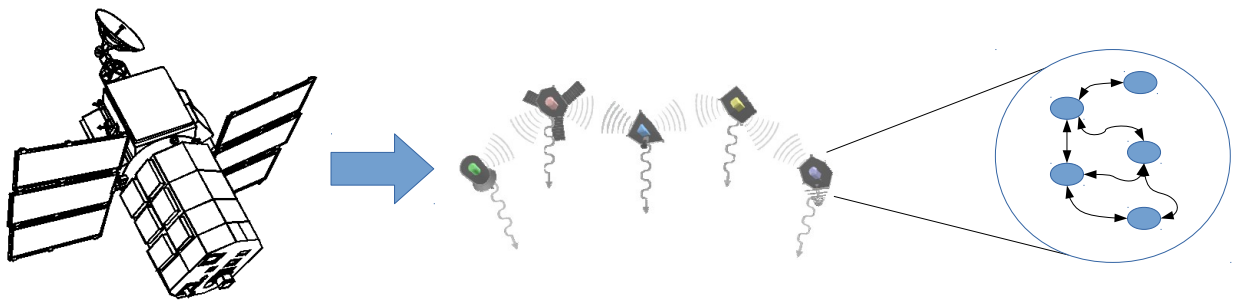

Figure 1.3: Left to right: large monolithic spacecraft is fractionated into a cluster of smaller satellites. A multi agent robotic system operates the each satellite.

cuses on separating the main functionality or subsystems of a large space asset into smaller hardware systems that are interconnected. The work on this thesis is particularly focused on the software distribution aspect using the agent paradigm, not necessarily the system fractionation. But a significant effort was made in this work to develop a MARS that is flexible and scalable enough to operate one satellite or multiple satellites. Figure 1.3 shows the work developed in this thesis in perspective with the DARPA F6 concept. DARPA's concept is to fractionate a large spacecraft into smaller units (left side of figure). Each of these small satellites can be operated by a MARS that operates each spacecraft, but can also be expanded to operate the whole cluster (right side of figure). Also MARS in the future can be used to autonomously manage large satellite constellations such as the mega-satellite constellation proposed by OneWeb and SpaceX. Ground operations of such large constellations will eventually not be scalable since it is not realistic to have one-thousand people and/or ground stations to control one-thousand satellites. This work shows that the software agent paradigm is a viable solution for the future of autonomy and distributed architectures in space.

There are two future applications in particular that motivate this work. The first is the assembly of large telescopes in space such as depicted in Figure 1.4. One of the current

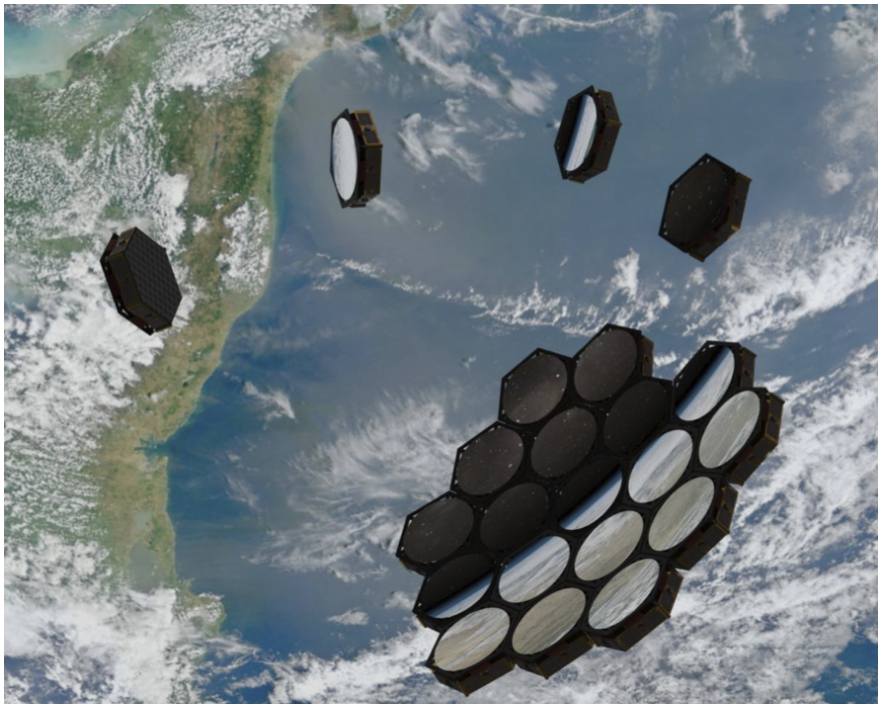

Figure 1.4: Artist rendering of space telescope assembly. Image Credit AAReST project.

concepts in development is the Autonomous Assembly of a Reconfigurable Space Telescope (AAReST)[4] as a collaborative effort between the California Institute of Technology, the academic branch of NASA's Jet Propulsion Laboratory, and the University of Surrey. A second application is the concept for building a large solar energy collector in space such as depicted in Figure 1.5. The Japan Aerospace Exploration Agency (JAXA) recently proposed that a solar farm can be built within the next 25 years, this would be a 1-gigawatt commercial system. NASA also released a report in 2012 to demonstrate the feasibility of a large solar powered satellite delivering power using phased arrays [5]. Such structures will be several kilometers long, and it is not possible to assemble these large structures on the ground and put them in space using rockets. One solution is to launch and autonomously assemble the small individual pieces in space to effectively build such a large solar farm in a scalable way.

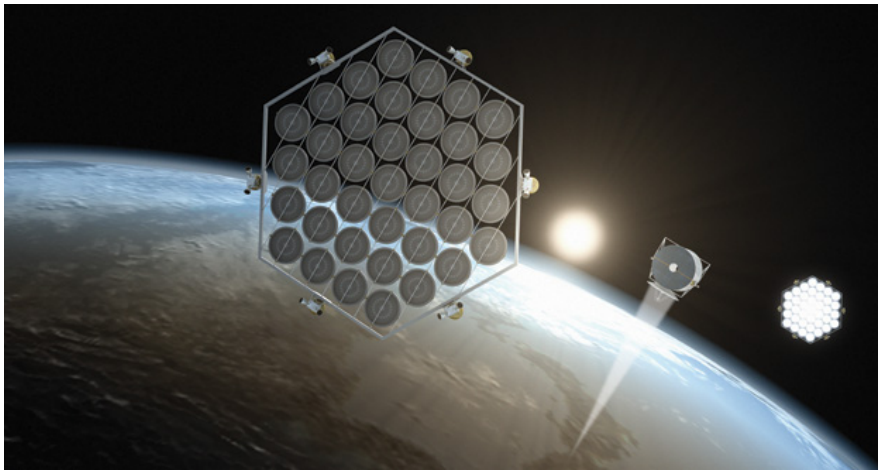

Figure 1.5: Illustration of space solar collector. Image Credit John MacNeill.

The following is a summary of the main contritions of this work. In Chapter 2 a new MARS is developed for distributing the attitude determination and control subsystem (ADCS) for HiakaSat, a small satellite built at the Hawaii Space Flight Laboratory (HSFL). This is a first step to show that MARS is viable solution for the attitude control and autonomy of a single spacecraft. Two small satellite ADCS implementations are compared: the new MARS and a typical monolithic implementation. Using hardware-in-the-loop tests it is shown that the MARS is capable of implementing the same ADCS algorithms with no control degradation, the control performance is not significantly affected on the pointing accuracy and detumble rate. But there are added benefits by using the MARS such as reduced CPU load, improved tolerance to subsystem faults, modularity and scalability. This is a significant improvement over the traditional monolithic implementation for ADCS because of the benefits introduced by MARS. As part of the ADCS implementation a new orbital propagator is introduced, the "pseudo-ECI" propagator, which addresses the "lost-in-space" problem after separation, and therefore reduces the risk for mission success. In Chapter 3, a new MARS is presented for controlling multiple satellites to move cooperatively. Also a six degrees of freedom (6DoF) guidance, navigation and control (GNC) method is introduced

based on a new 6DoF guidance algorithm for docking the satellites. This is a 6DoF extension of the three degrees of freedom (3DoF) E-Guidance method used in the Apollo era. The new E-Guidance method has advantages over existing approaches particularly because of the explicit guidance solution that is computationally efficient, and can be implemented in embedded computers to compute the guidance solution in real time. As a proof-of-concept for the MARS and GNC algorithms, an application for the autonomous rendezvous and docking (ARD) of four CubeSats is shown. In Chapter 4 a new algorithm, based on GA, is developed to find the optimal configuration of large satellite constellations to cover ground areas in a fast revisit time. This is particularly useful for Earth observing applications. Two case studies are presented to illustrate the utility of this method: 1) a constellation for global ocean observation with a 24 hour revisit time or less 2) a constellation design for North Atlantic Track observations with a 3 hour revisit time or less. In summary, one of the main contributions of this dissertation is to show that MARS can be used to solve the autonomy problem for satellites in particular the ADCS problem and the ARD for multiple satellites. The second main contribution is to show that satellite constellations can be significantly optimized in terms of numbers of satellites and revisit time for large areas of the globe.

This work is split into three main sections. Each chapter has its own introduction and conclusion with a more detailed description of the approach taken, therefore only a brief introduction is given in the next paragraphs for context. Chapter 2 describes the development of a MARS based on the Comprehensive Open-architecture Solution for Mission Operations Systems (COSMOS) for the decentralized attitude control problem with focus on HiakaSat, a small satellite for Earth observations developed at the HSFL. The main objective of this section is to develop a MARS to augment the possibilities for autonomy for a satellite ADCS. The MARS definition is given within the context of this work and implemented through several software agents which are designed to simplify and distribute the ADCS. Each agent is

focused on a particular aspect of the ADCS, for example, some agents act as drivers to the hardware while other agents execute the navigation and control algorithms of the ADCS. Each agent is capable of communicating with the other agents and reacting to the incoming data and external commands. Also, failure detection isolation and recovery (FDIR) mechanisms are included in each agent to enhance the robustness of the system. A computational control architecture is presented with the mathematical formalism needed for real time onboard execution. The proposed control and navigation algorithms are also validated using numerical simulations. The complete ADCS software and hardware is tested and validated on the HiakaSat testbed which uses a 3DoF air bearing as a representative platform to simulate the space environment. One other important aspect of this study is the comparison of a typical centralized/monolithic ADCS with the proposed MARS implementation. The goal of this comparison is to demonstrate that there are no significant disadvantages in MARS approach, but instead there are multiple benefits such as the decentralization of functionality, robustness to failures, computational efficiency, modularity, plug-and-play capabilities and scalability.

Chapter 3 is inspired by the possibility to build large space structures in the future, such as large space telescopes or solar farms. The main objective is to demonstrate that it is possible to autonomously dock multiple small satellites as building blocks to these large space structures. This work introduces a new MARS for the implementation of a distributed but coordinated system for ARD. In particular a new 6DoF E-Guidance method for position and attitude is developed based on the 3DoF E-Guidance method used during the Apollo era for the lunar module landings. The new E-guidance method is made part of a complete 6DoF GNC architecture for controlling each satellite. The GNC algorithms are implemented in software libraries as part of the MARS for realistic experimentation of the architecture for autonomous coordination of the satellites. To demonstrate the applicability of the proposed

architecture for control multiple spacecraft, a simulation scenario is run for docking four CubeSats separated radially and deployed from a launcher with a typical separation velocity.

Chapter 4 addresses a slightly different problem from the previous sections, but the topics are related to the distributed nature of future space architectures such as the large satellite constellations. This is motivated by the recent interest in the commercial, private and research communities to launch satellite constellations with a large number of satellites for fast revisit times of different regions of the Earth. The problem is non-trivial because of the large number of possible combinations for the satellite configurations, the sensor configurations, and other requirements such as daylight data acquisition. These large satellite constellation designs require powerful optimization tools to minimize the number of satellites while maximizing the coverage. A new constellation generator algorithm is implemented using Mathworks MATLAB® and the Systems Tool Kit (STK) that produces valid satellite configurations. The optimization method is based on GA that finds an optimal configuration by evaluating the performance of different satellite constellations. Two case studies are presented. The first case is the optimization of a satellite constellation for rapid coverage of the global ocean in 24 hours, 12 hours and 4 hours revisit time. Motivations for such a study are the surveillance of maritime piracy or the study of plankton across the globe. The second constellation optimization study focuses on the optimization for the rapid coverage for the North Atlantic Tracks which are the main air traffic airways that connect the U.S and Europe. The motivation for this study is the air traffic monitoring between Europe and North America, particularly in disruptive situations like the Icelandic volcanic eruptions in 2010.

Chapter 5 concludes the thesis by summarizing the most important results and conclusions, listing the main contributions of this work and finally by addressing possible recom-

mendations for future work.

CHAPTER 2
MULTI-AGENT ROBOTIC SYSTEM FOR SATELLITE ATTITUDE CONTROL

2.1 Introduction

Space missions commonly rely on robotic systems that are controlled from the ground with human assistance for decision making, mission planning and operations[1], but the tendency is to increase autonomy for these missions by relying on onboard autonomy. Also, current software architectures for space missions primarily rely on centralized/monolithic control systems as demonstrated by Table 2.1 which shows a survey of recent satellite missions. Centralized software is not a sustainable paradigm for the complex missions envisioned for the future because of the unscalable monolithic complexity that limits the development of the satellite autonomy. This work addresses the autonomy problem of a spacecraft by developing a distributed software architecture - a multi-agent robotic system - for the attitude determination and control subsystem. The MARS is implemented and tested in a small satellite hardware testbed.

Satellite Name	Buit by	Purpose	Launch date	ADCS	ADCS software	ADCS CPU	CPU MHz	DMIPS
Dove 2b-9	Planet Labs	Earth Observation	2015-10-07	MAI-200	monolithic	NXP LPC1768	100	125
LAPAN A2	Indonesian Space Agency	Earth Observation	2015-09-24	custom	monolithic	SH7145 RISC	50	65
TET-1	DLR	Earth Observation	2012-07-22	custom	monolithic	LEON-3	125	20
SPOT-7	EADS Astrium	Earth Observation	2014-06-30	custom	monolithic	LEON-3-FT/SCOC	80	74
QB50P1	von Karman Institute	Tech demo	2014-06-19	Cube ADCS	monolithic	ARM Cortex-M3	48	60
MicroMAS	MIT	Earth Science	2014-07-13	MAI-400	monolithic	NXP LPC1768	100	125
TechDemoSat-1	SSTL	Tech demo	2014-07-08	SSTL-150 bus	monolithic	OBC750	733	500

Table 2.1: Survey of recent satellite ADCS implementations.

2.1.1 Motivation

The NASA roadmaps are a guideline of the eminent technologies needed for the space sector during the next 20 years. The latest NASA roadmap for technology area 4 "Robotics and Autonomous Systems" (2015-2035)[1] hinges on the importance of agents and multi-agent robotic system precisely to address the problem of autonomy and mission complexity. Future space missions will rely more on systems capable of solving complex problems onboard without human intervention. This fact is specifically important for missions where communication with ground is limited, disrupted or even impossible. One other important factor is related to the fast changing system dynamics making remote operations simply not possible. The evolving system complexity factor is also generally true for any robotic system, but in space there are added difficulties such as accessibility, communication delays, harsh environments, etc. Also, modern robotic systems will increasingly depend on more than one vehicle to support the mission - these will be mobile teams of autonomous spacecraft working cooperatively towards common goals. Software is the key element to enable such complex robotic spacecraft missions.

The development of sophisticated software architectures that are fully autonomous, distributed and that work tightly with the hardware of the spacecraft will become ever more important. The software will have to understand and interpret the unpredictable environment and changing conditions to appropriate actions. This chapter proposes that a distributed and modular architecture, such as a MARS, can be effectively applied to satellite ADCS and improve the autonomy of the system. This is also a step towards demonstrating that MARS is generalizable for other satellite systems to improve space mission autonomy.

2.1.2 Background

The idea of distributing complex algorithms and functionality among multiple cooperative software agents was originally introduced by the Artificial Intelligence (AI) community[6]. In recent years the concept has been evolving among various research areas, notable examples are the robotics and control communities[7]. There are two main schools of thought regarding MARS. The first, based on the traditional AI community, is focused on software based agents that reside in a network environment[8][9][10] rather than a specific physical environment. The second approach is mostly used by the robotics and control communities that consider agents as separate robots[11][12][13] where each agent is a robot reacting in a physical environment. Many times there is an overlap between the two approaches. The approach taken in this work is the closely related to the former. Defining MARS based on software agents that reside in a computer network system, as it is withing the AI community, but with the difference that the agents also interface directly with the robot hardware, in this case the satellite hardware. The term MAS and MARS are closely related but in this work the acronym MARS is used to emphasize the close relation with the robot hardware.

The following work is focused on distributing the software of a satellite ADCS using the newly developed MARS. The attitude stabilization problem of a rigid body spacecraft has been solved[14] and implemented[15] for past missions but typically in monolithic implementations. Recent examples of satellites that implement monolithic ADCS are listed in Table 2.1. But the attitude control systems will keep pushing the requirements in terms of autonomy[16], pointing precision and stability[17], disturbance rejection and robustness[18], fast slew maneuvering and others. To address these new ADCS problems the monolithic paradigm is not sustainable because of the inherent complexity and centralized control which can limit the evolution of autonomous algorithms, is not modular, scalable and also can lead more easily to a complete system failure. MARS on the other hand can be made to im-

plement the same ADCS algorithms, but in a distributed manner with "intelligent" agents. This enables systems that can be more autonomous, robust, scalable and can ultimately improve the precision and responsiveness of the control.

Typically the ADCS software is implemented in a system with limited computing power[19]. One of the consequences of such a limitation is that the software cannot grow in complexity to easily enable more autonomy. Also, typical ADCS computers are based on micro-controllers without multi-threading ability which forces the monolithic implementation. The ADCS software then grows in monolithic complexity until it reaches the maximum level of instructions to implement the required functionality in memory limited micro-controllers. Monolithic ADCS can take years to develop because of the inherent complexity, but also because these are custom made for specific mission requirements. This centralized integration can lead to complete system failures when radiation events upset the software system. By distributing the software into different memory spaces in the computer, as is the case of MARS, the probability of system failures is minimized. With the advent of new computer capabilities for space, it becomes possible to distribute computing resources within the same computer core or even across multiple computers using a MARS. The advantages of a MARS to traditional monolithic systems start with the elimination of the monolithic complexity but continue on the scalable development for system autonomy, system robustness against failures, increased reliability, added flexibility and modularity, and also on a more practical aspect it reduces the development time and cost of software development. This work pushes forward the ability to implement the aforementioned future ADCS requirements using a newly developed MARS. This solution allows one to break down the complexity of large and monolithic ADCS in a scalable way using separate modules or agents. Special focus is given to the computational efficiency, the distributions of tasks, and robustness to failures for an ADCS of a small satellite.

2.1.3 Related Work

Large space missions typically have large teams and resources to develop the complex systems required for the mission, including software to implement autonomy - some example missions are listed next. On the other hand, small satellite missions have limited resources and the software development is many times an afterthought. What this work brings to the small satellite community is the ability to develop complex software autonomy using the proposed MARS based on COSMOS. This enables small teams to also develop sophisticated missions taking advantage of the embedded technological advancements including the advancements in software such as the one developed for this thesis. For the context of system level autonomy the following list of sample missions is presented to show the evolution of large spacecraft systems that have implemented some level of autonomy. In 1994 the Clementine spacecraft experimented with autonomous operations during its lunar orbit[20], followed shortly that same year by an autonomous operations experiment on the US Air Force Technology for Autonomous Operational Survivability (TAOS) satellite whose purpose was to demonstrate autonomous operation and space navigation systems to reduce satellite ground support needs. In 1998 Deep Space 1 (DS1) became the first spacecraft to use an autonomous navigation system and autonomous remote agent. The autonomous navigation, or Autonav[21] system was developed by NASA's Jet Propulsion Laboratory (JPL) and had the purpose of determining its position autonomously without the intervention of ground control operators. The Autonomous Remote Agent Experiment (ARAX)[22], also developed at JPL and NASA Ames Research Center, was the first onboard artificial intelligence system to control a spacecraft without human intervention. ARAX was an intelligent and self-repairing software application with a planner/scheduler, smart executive, mode identification and recovery. This agent experiment was successfully tested during the DS1 mission. The work became a precursor for the future of space exploration with autonomous control and operations. These technologies have allowed spacecraft to operate in

deep space with minimal human intervention and greater reliability and robustness to failure. Since then the technology from DS1 has been used for the Phoenix Mars Lander, the Mars Science Laboratory and the Earth Observing 1 (EO-1) satellite.

2.1.4 Contributions

This work presents an autonomous onboard ADCS using the MARS paradigm. The MARS is based on the COSMOS core software framework also developed at the HSFL. Part of the motivation for the new ADCS is to reduce the complexity of the software elements and increase the robustness of the ADCS system by implementing different agents that operate each ADCS hardware and software components. The implementation also adds one agent for the navigation solution and another for the control algorithms. One other major advantage of using different agents is that these can be deployed virtually in any platform. For development this is a very practical situation where the agents that directly connect to the hardware can live in the satellite platform but the other agents can be deployed in any other computer, which could have more computing resources, making the development and testing of the software very flexible. This allows the software developers to focus on particular aspects of the control architecture rather than the whole system level software. As far as the author is aware the MARS approach has never been applied to a complete satellite ADCS. This particular MARS based ADCS was developed for the HSFL satellites[23, 24] and proper definitions for the MARS are given in this work. A description of the software framework of the MARS and a definition of the various agents that compose the distributed satellite ADCS is also given.

The main contributions of this work is the implementation of a MARS framework for autonomous onboard satellite ADCS and its comparison with a typical monolithic ADCS.

One other contribution of this work is the introduction of a "pseudo-ECI" orbital propagator to resolve the problem of "lost-in-space" after the satellite is deployed. This is important for the satellite to be able to point at the Earth once deployed in the case of Global Positioning System (GPS) failure. The design and experimental validation of the MARS are part of the contributions. The work is focused on the implementation of the MARS and the collection of experimental results in a realistic setting using an ADCS testbed developed at the HSFL.

2.1.5 Outline

The work is organized as follows. Sec. 2.2 introduces the MARS and relevant concepts which are the basis of this work. In Sec. 2.3 a case study is presented based on the HiakaSat satellite developed at HSFL. Sec. 2.4 described the satellite attitude control architecture along with the various agents that implement the complete control system. Sec. 2.5 presents the results of the software simulations to validate the control algorithms and also the hardware-in-the-loop tests to validate the MARS based ADCS using satellite testbed. Finally in Sec. 2.6 a summary is presented with the conclusion of the work done and future research directions are proposed.

2.2 Multiagent Systems

The current concepts of "agent" and "MAS" have emerged from the field of distributed artificial intelligence during the 1970s[6], but only in 1990s has the "agent" (some would say the "intelligent agent") paradigm solidified by merging the existing AI concepts and computer science[25]. The original agent concept has continued to evolve and has branched into multiple fields of research. Russell[8] now defines modern AI "*as the study of agents that receive percepts from the environment and perform actions*". This concept has recently become

ubiquitous in AI but other fields are following the same steps such as computer science[26], robotics[27], control[28], decision theory and economics[25], power engineering[29], sensor networks[26] and many others. Most of these fields define an agent as an entity (computer program, robot, etc.) that is executed autonomously and is capable of reacting to the environment in which it is contained on behalf of another agency[26]. Most definitions found in the literature are broad and generic. Gerhard Weiss[30] suggests that "*an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to achieve its delegated objectives*". This definition is focused on a software entity and does not necessarily define an intelligent agent. In decision theory and economics an "agent" has been referred to as a "rational agent" which is one that acts in a way that achieves the best expected outcome[8]. When this concept was combined to the computer science definition of objects then the agent concept became an "intelligent agent"[25]. This definition is the modern paradigm of an "agent" when applied in the sense of computing.

Considering the previous definitions and bringing the focus to the present work of a MAS for autonomous onboard satellite attitude control we propose that **an intelligent agent is a computer system that is capable of sensing and acting on its environment (reactive), can interact with other agents (social) and can autonomously decide which actions are necessary to maximize the chances of achieving the control goal (proactive)**. With this in mind, simple programs can be "intelligent agents" if these are problem-solving oriented and maximize the chance of success of the system objectives. Any control system can be said to be an agent if it meets the proposed definition. Other fields have started to define agents as fully functional complex machines (robots) that behave intelligently. In essence agents enable autonomy with purpose. But for the purpose of this work we define a control agent as a software program designed to execute tasks in a reactive,

social and proactive way[9].

An agent may contain a control thread but, as an abstraction of the autonomy concept, it is flexible and reactive, not hardwired, so that it can adapt to the surrounding environment. The agent may decide to choose between different control threads in reaction to the environment, it may also automatically learn which control loops are better or even improve the gains on the loops. In the area of robotics an agent can be thought of as a generalization of a control program. The important concepts to retain regarding "robotic agents" is that they must behave in a reactive, social and proactive way.

Similar to the "agent" concept, a **MAS** comes from the field of distributed artificial intelligence. A MAS is defined as a system with multiple intelligent agents that interact with each other. Most commonly a MAS is referring to multiple software agents, but researchers are finding that the MAS concept can be attached to multi-robotic systems[12]. The concept of a MAS has evolved and has also branched into multiple fields of research such as cooperative robots[11], computer games and movies[31], computer vision[32] and data mining[33]. So much that there are even different communities at odds over what the purposes are for MAS and the corresponding definitions[12]. For these reasons finding a precise definition of a MAS is not trivial because it depends on the field or research. Even so it is possible to summarize some of the important concepts that are common to MAS literature. In the context of this work a ***MAS is as a collection of Agents that interact with each other in the pursuit of common goals or tasks***. This is done in such a way that only by the action of the group is the task achieved, no single agent can complete the overall goal of the system. A MAS must also be a framework or infrastructure to enable the communication between agents and also specify the interaction protocols.

Because of the traditional limited computing power used in space applications[34, 19] the systems are forcibly monolithic. Figure 2.1 shows the evolution of the computing throughput of space qualified processors since the 1990's compared to commercial processors[35]. The space qualified processors lag typically 10 years behind the commercial counterparts and their proliferation into the space industry takes usually another 10 years. This means that the average computer in operational satellites lags typically 20 years when compared to commercially available computers on the ground. Monolithic systems, as opposed to MAS, combine all the required functionality into a centralized and tightly coupled architecture for data collection, data processing and decision making. For these reasons standard in satellite ADCS have been typically implemented as monolithic systems. But today with the advent of more powerful space qualified embedded computers it is becoming possible to deploy a MAS into an embedded system with a very small footprint, enabling the dynamic management of data and hardware with more advanced software, in this work the interest is in software agents. With a MAS different components can be added "on the fly" on one or more processors and the data processing can also be easily distributed. The task to efficiently control a spacecraft becomes very complex when there is a large number of data collection channels and there is also a need to increase robustness of the control program execution. The growing sophistication of navigation and control algorithms also increases the software complexity. In a monolithic system if the control program fails for some reason the whole control mechanism fails. But in a MAS parts of the control mechanism may fail without causing the entire system to fail. With the advent of multiple devices interconnected with each other, of multi cores computers and multi-vehicles missions it will not be possible to only perpetuate the monolithic system model because of the limitations previously mentioned. Table 2.2 summarizes the advantages and disadvantages of monolithic systems vs. MAS.

A MAS in the context of this work will borrow the characteristics introduced: it is a

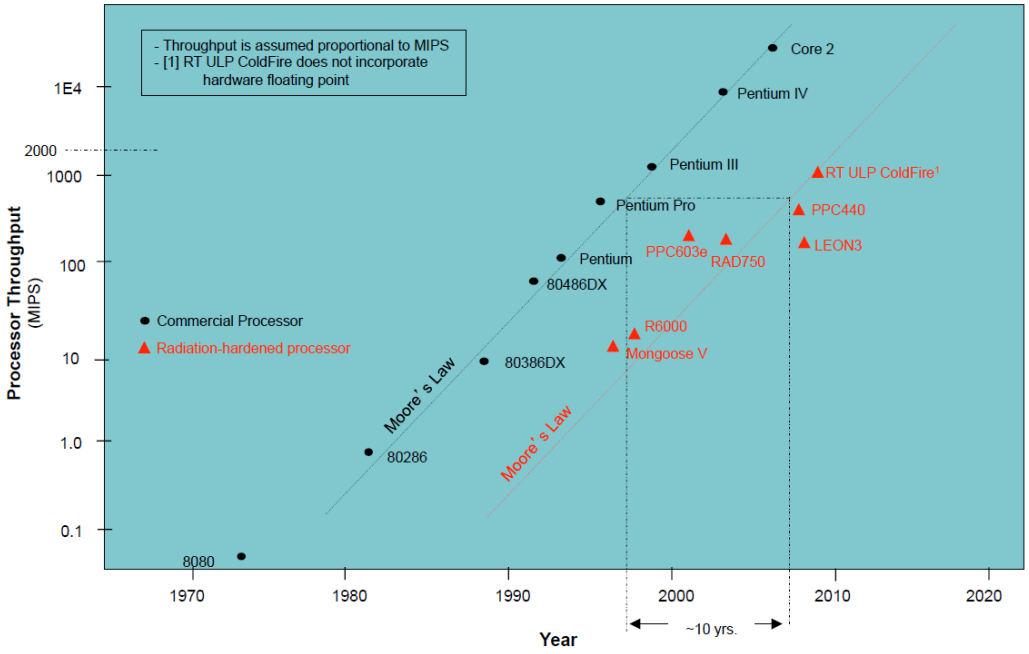

Figure 2.1: Evolution of computing throughput of space qualified processors vs. commercial processors. Image credit RHESE Project overview slides.

Function	**Monolithic System**	**Multiagent System**
software development	monolithic, centralized	modular, distributed, scalable
add/remove components dynamically	fixed, not possible	flexible
data processing	centralized	centralized or distributed
geographic distribution	centralized	centralized or distributed
robustness to failures	whole system fails	partial system failure
plug-n-play	fixed, not possible	flexible to implement
scalability	limited to single units	scalable, limit is data bus bandwidth

Table 2.2: Qualitative comparison of monolithic systems with multi-agent system

software framework that enables two or more agents to socialize and react autonomously for the purpose of controlling a satellite. Each agent has its own localized goals, but works toward an overall arching control solution of the satellite. Particularly for this work we will focus on a MAS for autonomous onboard attitude control. These agents run inside the main onboard computer for the satellite. Future work will expand the MAS concept to deploy multiple navigation, attitude control and other agents to control a distributed architecture of satellites.

Because the concept of MAS is plural (can be contained inside a single processor board and can be also generalized to be used in multiple boards/satellites) we define a **MAS-bracket** as a group of agents that run on the same board with the purpose of controlling a single vehicle. A group of brackets is **MAS-cluster** that controls a multi-vehicle architecture.

In recent years the robotics community has started to develop its own versions of MAS frameworks that enable researchers to work with multi robotic scenarios where the hardware is an important key factor. Some researchers have proposed to differentiate the pure MAS AI frameworks from the robotic MAS frameworks[27]. For this work we use the following nomenclature, general purpose MAS are simply referred to *MAS Frameworks* and the robot-focused MAS are *MARS Frameworks*. Other common nomenclatures for MARS Frameworks found in the literature are "Robotic Middleware" [36], "Robotic Software Systems"[37] or even "Robotic Software Frameworks"[27]. Table 2.3 summarizes the main features of most important existing MARS Frameworks along with COSMOS.

The following is a brief summary of the different MARS frameworks presented in Table 2.3 highlighting the differences, disadvantages and advantages compared with COSMOS.

MARS Name	Community	OS	Licence	MAS Standard	Open Source	Dev. Language	Drivers	Control Real-time
COSMOS	Satellites, UAVs	Linux, Mac, Win	LGPL	no	yes	C/C++	yes	partial
Mobile-C	Embedded, real-time	Linux, Mac, Win	GPL	FIPA MASIF	yes	C/C++	no	yes
YARP	Humanoid Robots	Linux, Mac, Win	LGPL	no	yes	C/C++	yes	yes
ROS	Robots in general	Linux	BSD	no	yes	C++,	yes	no
OROCOS	Robotics research	Linux, Mac, Win	LGPL	no	yes	C++	yes	yes
OpenRDK	Heterogeneous robots	Linux, Mac	GPL	no	yes	C++	yes	no

Table 2.3: Multiagent Systems Middleware packages

Mobile-C is a Multi-Agent platform for mobile C/C++ agents. It is implemented as a library so it can be easily integrated into applications to handle mobile agents. A mobile agent has the same properties as described previously with the advantage of mobility. This means that the agent, computer software and its data, is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer. COSMOS is not a mobile agent framework since in principle the agents are constrained to a physical embedded computer in space. On the other hand one of the advantages of COSMOS over Mobile-C for spacecraft is the ability to send requests, so instead of moving the agent, agents can request other agents to act on their behalf.

Yet Another Robot Platform, or YARP, is a very mature C++ framework specifically focused on humanoid robotic systems development. This framework was originally developed at the University of Genoa and The Massachusetts Institute of Technology but it has since then been adapted by other industry groups. It allows to easily deploy a collection of programs communicating in a variety of peer-to-peer connection types such as tcp, udp, multicast, local, XML/RPC and MPI. Even thought YARP is an excellent framework to develop a MARS based application it is a low level framework with only the essential mechanism for communications and would requires some effort to implement the typical agent like qualities. COSMOS is primarily focused on UDP type messages broadcasted over the network so compared to YARS it is limited in this way. Nevertheless COSMOS provides a comprehensive functionality to deploy agents with less than 10 lines of code.

ROS is the Robot Operating System. This is a very generalist framework that can easily be integrated with other technologies. ROS started at Stanford University and has quickly gained a strong reputation among the robotics community. This is a hybrid P2P distributed architecture but with a centralized master node that must be known by each client. The master node is responsible for managing the lookup and naming services and the communication mechanism is build upon a TCP transport layer with some interaction using XML-RPC/TCP. One major difference between COSMOS and ROS is the fact that most of the communication is send over UDP in ASCII format using JSON. This makes debugging of the raw messages a simple process compared to ROS and other frameworks.

Open Robot Control Software (OROCOS) surfaced from the European Robotics Research Network. This is a specialized framework for industrial robotic systems in real time. Some of the main focus of this framework is dedicated to robot hardware interfaces and drivers for real time applications. It is compliant with hard real time operating systems such as RTAI, Xenomai, QNX and WxWorks. OROCOS is also a hybrid P2P architecture of various nodes that can communicate over the network. COSMOS has not yet been tested in hard real time operating systems which is one of the main diferences with OROCOS. Nonetheless COSMOS is also very hardware oriented providing a number of drivers for sensors and actuators for satellites making it very easy to implement a COSMOS based software into a new satellite.

OpenRDK is focused on quick development of robotic systems for small research projects. It also has a hybrid P2P distributed architecture (agents) based on UDP or TCP but also depending heavily of shared memory and threads (components). This framework also has a set of console commands to control, monitor and logging every aspect of the system. One of the similarities with COSMOS is the option of sending messages using an ASCII based

format or binary. In the case of OpenRDK the ASCII format is XML and in COSMOS is JSON. One major difference with COSMOS is that OpenRDK is not cross compatible with the major operating systems while COSMOS has been designed to execute in most operating systems, from desktop OS such as Windows, Linux and Mac as well as embedded OS such as Linux Yocto, Linaro, etc.

One major difference that separates COSMOS from all the remaining frameworks is the supporting infrastructure for the space community such as orbital propagator algorithms, fitting algorithms, attitude control libraries and satellite hardware drivers. The other frameworks listed, except ROS, are serving also a specific community so while trying to bring a MARS from one community to another may be possible it is also a challenge because many of the available algorithms and tools are not relevant. This is essentially the main reason why COSMOS was developed in the first place, to address the needs of the small satellite community such as universities and research laboratories to avoid reinventing a distributed software framework for every new project.

In this work we use the COSMOS[38] to develop the satellite ADCS software. COSMOS-core is a robotic system middleware, and in essence a MARS, that contains an agent library for distributed architectures and also for distributed message passing within multiple nodes. COSMOS has been designed to operate multiple satellites [39] and also used for different mission design studies such as GNC for multiple satellites [40]. COSMOS was also successfully implemented as the flight software of HiakaSat. COSMOS is a C++ software framework initially developed for satellite mission operations, but it has evolved to become an ecosystem for flight software and is now being used to operate different kinds of vehicles or nodes, not necessarily just satellites. The software has been primarily developed by the HSFL and it is open to collaborators.

Each COSMOS agent runs a *Main Thread* on a infinite loop in a threaded environment. This thread is responsible for the agent communication process and for executing the requests that have been implemented. One of the default requests is to shutdown the agent so it is possible to remotely terminate an agent. To start an agent automatically a daemon or script process must be in place. Agents make their presence known through a COSMOS *Heartbeat*. Any client wishing to communicate with an agent listens to the COSMOS multicast address until it either receives a Heartbeat from the agent it is waiting for or times out. Once it acquires the desired agent, it sends a request to the IP address and port taken from the Heartbeat. It then waits for a response on that same IP address and Port until it receives one or times out. All requests and responses are in plain ASCII. The COSMOS *Namespace* has a flat mapping of names to single values. Each name represents a string or a number. The meaning of each entry is predefined such as the units and data type of each value. The naming convention for message passing uses digits embedded within the name to represent the entry of an equivalent array. Each member of the Namespace is then mapped to a unique location in memory; the memory locations being drawn from a globally accessible storage space. This Namespace map is stored as a simple list of entries, allowing either forwards or backwards translation between COSMOS name and memory storage.

The communication mechanism in COSMOS is asynchronous message passing using the UDP broadcasting transport. COSMOS implements a naming service to enable different agents to locate each other. It also allows pushing mechanisms (the Namespaces). It has a lookup service that allows it to find agents that provide other services and requests. As far as the author is aware COSMOS is the only MAS openly available that is focused on satellite middleware.

2.3 Case Study: HiakaSat

HiakaSat is the first satellite designed and built by HSFL for the ORS-4 launch using the Super Strypi launch vehicle. HiakaSat is a 55kg micro-satellite designed for remote sensing with its primary payload as the Space Ultra Compact Hyperspectral Imager (SUCHI) also designed and developed in-house[41]. SUCHI was funded by a DAPRA project and it is a long wave infrared hyper-spectral imaging system. The satellite also includes two color cameras with both wide and narrow field of view lenses. These three imagers are used together to provide remote sensing data across a wide range of wavelengths. Unfortunately the Super Strypi failed to reach orbit on its debut flight on November 3, 2015[42]. Since this work is based on the development of the flight software to control HiakaSat and because the satellite did not reach orbit, the focus is directed towards the experimental validation of the MAS software on the HiakaSat testbed developed for the satellite. Although this work is focused on HiakaSat the concepts and results are applicable to other future satellites. Figure 2.2 shows the satellite before delivery for the launch.

2.3.1 On Board Computer Subsystem

The central system of the satellite is the on-board computer (OBC), which is the main flight computer running the software agents described in this work. All the sensors, actuators, payloads are connected to the OBC using standard bus protocols such as RS232, RS422 and Ethernet. The OBC consists of one single core Texas Instruments DaVinci DM3730 processor with an ARM Cortex-A8 architecture. It runs at 800 MHz with a possible max clock speed of 1GHz with RAM of 512 MB LPDDR (PoP) and 512 MB of Flash memory. The operating system is a Linux Linaro based image tailored and optimized for space missions. Most satellite processors have very low computing power and every process is made

Figure 2.2: HiakaSat, HSFL engineers working on final preparations before delivery

Figure 2.3: HiakaSat On Board Computer box. Figure on the left shows the 3D model and the figure on the right shows the flight unit operational before final integation with the satellite.

to minimize the load on the processor instead of maximizing the functionality of the system. Most modern satellite computer processors have less than 100 Dhrystone MIPS as shown by Ginosar[19]. These figures are rapidly changing but general dissemination in the space sector is typically slow (most computers currently used in space are using technology that was made available to the public approximately 20 years ago[19]). With more capable processors coming into the space industry a strong case is made for the utilization of MAS. In the case of the HiakaSat satellite it has a processor capable of 2000 DMIPS, which is 20 times more processing power than the average satellite processor existing today for most satellites allowing us to test the MAS and decouple the control processes, or in this case the ADCS agents.

2.3.2 ADCS

The objective of this work is to develop an attitude control architecture for satellite dynamics for HiakaSat that is a magnetically actuated satellite using magnetic torque rods as the primary control actuators based on a MAS concept. The components for the HiakaSat ADCS are shown in Figure 2.4. This control architecture must reject external disturbances

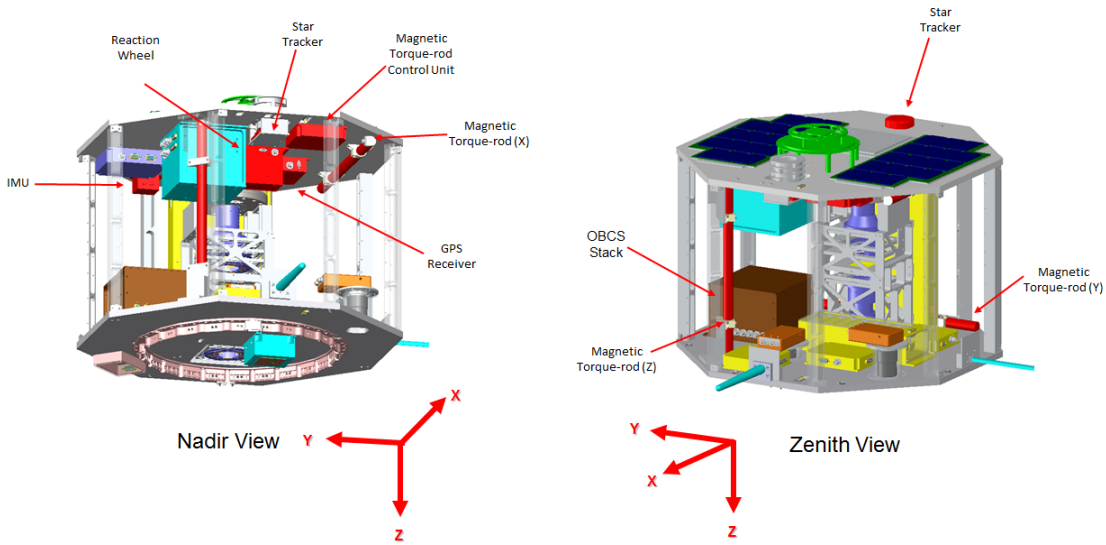

Figure 2.4: HiakaSat ADCS components

(environmental torques) efficiently that come from various sources but primarily from the interaction of gravity on the satellite mass (gravity gradient) and the uncertainty of the precise location of the center of mass (specially if the center of mass is determined using CAD models instead of actual measurements using mass properties instruments). Other external torques exist such as the solar radiation pressure and atmospheric drag but are not as significant.

To fulfill its main mission requirements the satellite must be three-axis stabilized and capable of autonomous pointing with an accuracy of 5 angular degrees or better. This capability must be maintained during eclipse time or sunlight. The operational modes of the attitude control system are listed in Table 2.4. The MAS automatically determines in which mode the satellite must be operating and changes modes according to the current status. While all modes must be supported, the detumble and nominal modes are mission critical. The nominal mode forces the spacecraft to orient itself with the orbital frame where the Z-axis points towards the geocenter, this is the local-horizontal local-vertical (LVLH) frame.

Mode	Description
Drift	Attitude control disabled to allow tumbling
Detumble	Reduce angular rotation and hold to random inertial attitude
Nominal	local-horizontal local-vertical (LVLH) hold attitude mode with nadir facing Earth
Special Pointing	e.g. Moon-pointing, inertial pointing
Off	no attitude estimation or control is performed

Table 2.4: Typical satellite ADCS modes

From this mode the spacecraft must have the flexibility to point to any attitude if required, such as pointing to the Moon or some other point of interest. The attitude data collected from the attitude sensors are processed by the control agents implemented on the main flight computer to determine spacecraft orientation. The attitude determination hardware is commercial off-the-shelf (COTS) - a Sinclair Interplanetary Star Tracker ST-16 and an Inertial Measurement Unit VectorNav VN-100 with a three axis rate gyro and three axis magnetometer sensor. The orbit determination data are processed using a Novatel OEMV-1 GPS sensor. The control authority is implemented using three magnetic torque rods (MTRs) developed at the HSFL. Each one aligned with the Cartesian directions of the satellite body frame. There is one agent to operate each of these hardware components individually.

Sensors

Star Tracker (ST) The start tracker selected is the first generation start tracker (ST-16) from Sinclair Interplanetary as shown in Figure 2.5. The sensor provides an attitude quaternion and angular rates at 2 Hz. The accuracy is better than 10 arcsecond cross-boresight (RMS) and less than 74 arcsecond around boresight (RMS). The maximum slew rate is 3 degrees per second. The attitude results collected by the star tracker are given

Figure 2.5: First Generation Star Tracker (ST-16) from Sinclair Interplanetary. Image credit Sinclair Interplanetary

in the sensor frame and must be converted to the body frame using the above formulation. The data is sent in the form of a four-element quaternion vector representing the rotation from the inertial reference frame into the frame of the sensor. The inertial frame used by the sensor is the Earth-Centred Inertial J2000 system while the star positions are not corrected for parallax. This permits the simplification of not having the data depend on time while loosing some precision. The rotation matrix from the inertial frame F_I to the sensor body frame F_B can be found from:

$$R_{B\leftarrow I} = \begin{bmatrix} 1-2q_y^2-2q_z^2 & 2(q_xq_y-q_zq_w) & 2(q_xq_z+q_yq_w) \\ 2(q_xq_y+q_zq_w) & 1-2q_x^2-2q_z^2 & 2(q_yq_z-q_xq_w) \\ 2(q_xq_z-q_yq_w) & 2(q_yq_z+q_xq_w) & 1-2q_x^2-2q_y^2 \end{bmatrix} \tag{2.1}$$

This attitude collected on the star tracker frame will have to be converted to the body frame. This is only an extra static transformation since the sensor frame is fixed with respect to the body frame. The angular velocity of the sensor is given with respect to the ECI in

the sensor frame. The following formulas show the conversion between the raw sensor data into the body attitude:

$$q_{B\leftarrow I} = q_{B\leftarrow S} q_{S\leftarrow I} \tag{2.2}$$

$$\omega_B = q^* \omega_S q \tag{2.3}$$

The star tracker body frame X direction is aligned with the satellite -X direction. The sensor Z direction is aligned with the outward boresight and is aligned with the satellite -Z direction. The sensor Y axis is given by the triad of the Z and X axis ($Y = Z \times X$). An orthonormal basis for the star tracker sensor and for the satellite body frame are defined. The basis is defined with three orthonormal vectors $\hat{I}, \hat{J}, \hat{K}$. $F_B = \hat{I}_B, \hat{J}_B, \hat{K}_B$ is the basis for the body frame, in this case it will be the identity basis. $F_S = \hat{I}_S, \hat{J}_S, \hat{K}_S$ is the basis for the sensor frame.

$$\begin{aligned} \hat{I}_S &= [-1, 0, 0] \\ \hat{J}_S &= [0, 1, 0] \\ \hat{K}_S &= [0, 0, -1] \end{aligned} \tag{2.4}$$

The rotation matrix that brings a vector from the sensor to the body frame is computed by a direction cosine matrix formed by the given basis. The direction cosines of an Euclidean vector are the cosines of the angles between the vector and the three coordinate axes. Equivalently, they are the contributions of each component of the basis to a unit vector in that direction. The direction cosine matrix in cause is given by:

$$\mathbf{R}_{B\leftarrow S} = \begin{bmatrix} \hat{I}_S.\hat{I}_B & \hat{J}_S.\hat{I}_B & \hat{K}_S.\hat{I}_B \\ \hat{I}_S.\hat{J}_B & \hat{J}_S.\hat{J}_B & \hat{K}_S.\hat{J}_B \\ \hat{I}_S.\hat{K}_B & \hat{J}_S.\hat{K}_B & \hat{K}_S.\hat{K}_B \end{bmatrix}^T \tag{2.5}$$

equivalently the computation of the quaternion that represents this rotation is given by

$$q_w = \frac{1}{2}\sqrt{1 + R_{11} + R_{22} + R_{33}} \tag{2.6}$$

$$q_x = \frac{1}{4q_w}(R_{32} - R_{23}) \tag{2.7}$$

$$q_y = \frac{1}{4q_w}(R_{13} - R_{31}) \tag{2.8}$$

$$q_z = \frac{1}{4q_w}(R_{21} - R_{12}) \tag{2.9}$$

Inertial Measurement Unit (IMU) The inertial measurement unit (IMU) selected is the The VN-100 Rugged from VectorNav Technologies. The physical unit is shown in Figure 2.5. This is a 3-axis miniature, high-performance IMU enclosed in a aluminum package. It contains 3-axis accelerometers, 3-axis gyros and 3-axis magnetic sensors, with a 32-bit processor.

The IMU sensor is also not aligned with the satellite main body frame because its physical location constrained its orientation. The sensor X direction is aligned with the satellite Y axis. The sensor Y axis is aligned with the satellite X axis. The sensor Z axis is aligned with the -Z axis. We define a orthonormal basis for the IMU and for the satellite body frame. The basis is defined with three orthonormal vectors $\hat{I}, \hat{J}, \hat{K}$. $F_B = \hat{I}_B, \hat{J}_B, \hat{K}_B$ is the basis for the body frame, in this case it will be the identity basis. $F_S = \hat{I}_S, \hat{J}_S, \hat{K}_S$ is the basis for the sensor frame.

Figure 2.6: VN-100 Rugged Inertial Measurement Unit from VectorNav Technologies. Image credit VectorNav Technologies.

$$\begin{aligned} \hat{I}_S &= [0, 1, 0] \\ \hat{J}_S &= [1, 0, 0] \\ \hat{K}_S &= [0, 0, -1] \end{aligned} \tag{2.10}$$

The rotation matrix that brings a vector from the sensor to the body frame is computed by a direction cosine matrix formed by the given basis. The direction cosine matrix is given in the same way as the star tracker sensor in Equation 2.5.

The IMU agent also applies the calibration bias for the gyro. This bias is computed by collecting data from the IMU for long periods of time and fitting the best estimate of the bias for that data.

$$\omega_B = R_{B \leftarrow S}(\omega_S + \omega_{bias}) \tag{2.11}$$

Figure 2.7: OEMV-1 Single-Frequency GPS Receiver from Novatel. Image credit Novatel Technologies.

The IMU agent also applies the calibration for the magnetometer measurements. This calibration is computed by collecting data from the IMU moving the sensor in every possible direction and computing the offsets and scaling factors to reduce the data ellipsoid into a sphere.

$$B_B = R_{B \leftarrow S}(B_S + B_{bias})B_{scale} \tag{2.12}$$

Global Positioning System (GPS) sensor The GPS receiver selected is the OEMV-1 sensor from Novatel. The OEMV-1 is a single-frequency GNSS receiver for the L1 and L-Band frequency. The physical unit is shown in Figure 2.7.

Actuators

Magnetic Torque Rods (MTR) The Magnetic Torque rods used in this project were developed by students mentored by faculty in the engineering department and HSFL engineers. The physical model is shown in figure 2.8. There are three torque rods, one for each axis of the satellite. Every MTR is made of a magnetic core with high permeability and a coil of copper wire around it, such as in a solenoid. Each torque rod is approximately

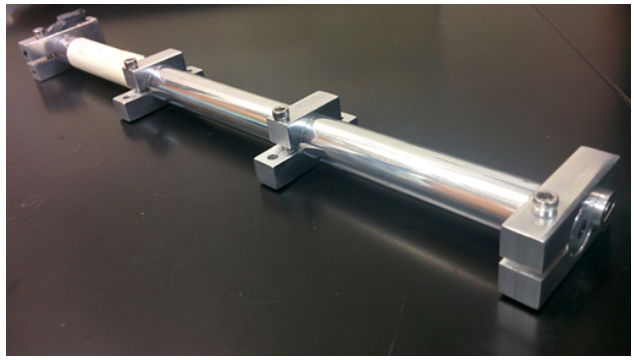

Figure 2.8: MTR developed by HSFL students

349 mm in length with a 0.677 kg mass. The operating voltage is 28V and each rod has a designed 300 Ohm resistance and maximum magnetic moment of 13 Am^2 at 100 mA.

The MTR produce a torque from the interaction of their magnetic dipole $\boldsymbol{M}$, that is generated when powered on, and the local magnetic field of the Earth $\boldsymbol{B}$:

$$\boldsymbol{\tau}_{ctr} = \boldsymbol{M} \times \boldsymbol{B} \tag{2.13}$$

It is common for LEO satellites to use MTRs, but they are most commonly used in combination with reaction wheels and used as a secondary actuator set for desaturating the reaction or momentum wheels. In this particular case the MTR are used as the primary control actuators. MTR can be used as primary actuators for low enough orbits because the intensity of the Earth's magnetic field B is strong enough to produce a control moment when interacting with the magnetic moment produced by the torque rods. For a 500 km orbit the magnetic field varies from approximately 0.20 to 0.55 Gauss. Figure 2.9 shows the typical magnetic field variation for the described orbit over a few orbits. This is computed using the latest World Magnetic Model released in 2015. This moment or torque produces a change in the angular momentum of the satellite. By precisely controlling the moment produced

by the three MTR the attitude of the satellite can be precisely guided. The satellite has three MTR placed in a orthogonal configuration in each axis of the Cartesian body frame. To make the spacecraft rotate at one revolution per orbit (LVLH tracking) from a stable attitude the spacecraft requires a constant control moment of approximately 0.0015 $mN.m$ during 30 minutes. Considering the fluctuation of the magnetic filed as shown in Figure 2.9 and using the torque rods at 50% power (6.5 Am^2) each rod can produce a control moment of 0.13 $mN.m$ to 0.36 $mN.m$ assuming the magnetic field is perpendicular to the moment. This means that in ideal conditions there is plenty of control moment, in excess of 80 times more, to maneuver the spacecraft to one revolution per orbit. Although the torque rods can produce enough torque when the Earth's magnetic field is co-aligned with the desired torque no torque can be produced. But on a cyclic basis during the orbit it is possible to generate sufficient torque to control the spacecraft as demonstrated in the software simulations. In practice the magnetic torque rods only are allowed to produced a moment when the magnetic field is almost perpendicular to the moment produced.

2.3.3 Flight Software

The flight software (FSW) for HiakaSat autonomously monitors and maintains the spacecraft state of health (SOH), controls the operation of the spacecraft, monitors the SOH of the payloads, performs the algorithms for the ADCS and controls the payloads. The FSW will always be recoverable from loss of power or function without ground support. It is also designed so it can be reloaded or modified from the ground without endangering the spacecraft SOH or mission. FSW modes are handled as a state machine amongst the persistent agents and all other commands are handled as independent programs. These programs either request changes in state from the processes, or directly perform simple functions themselves. The executive agent oversees the launching of all subprocesses and programs and it is the

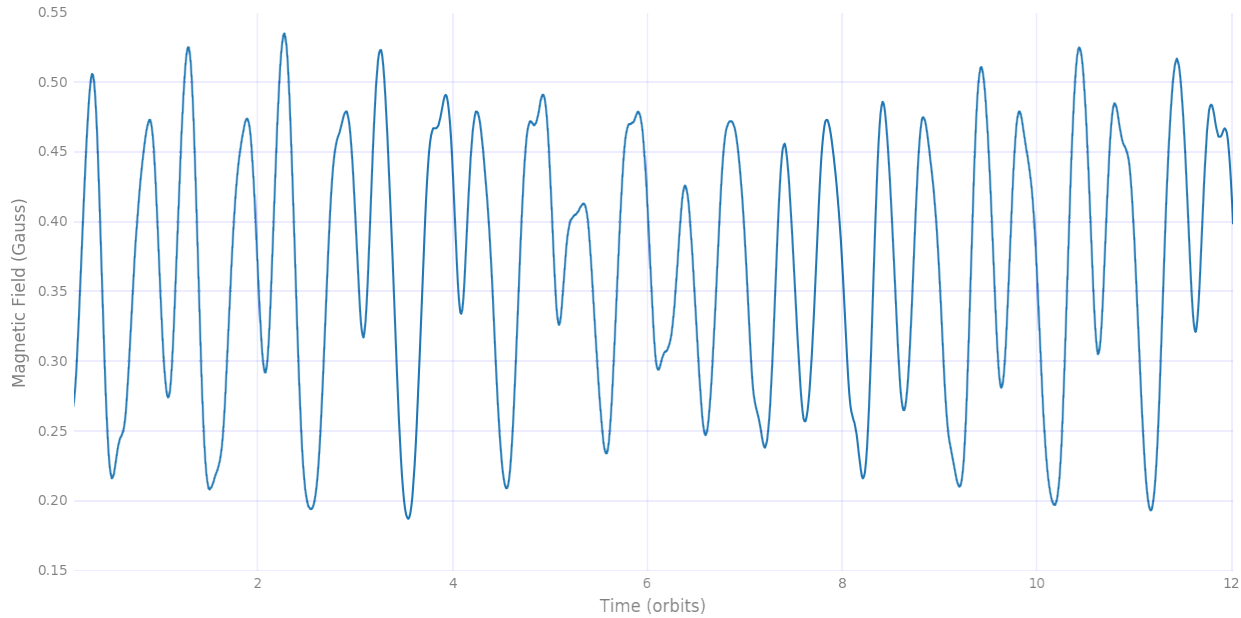

Figure 2.9: Total magnetic field over several orbits

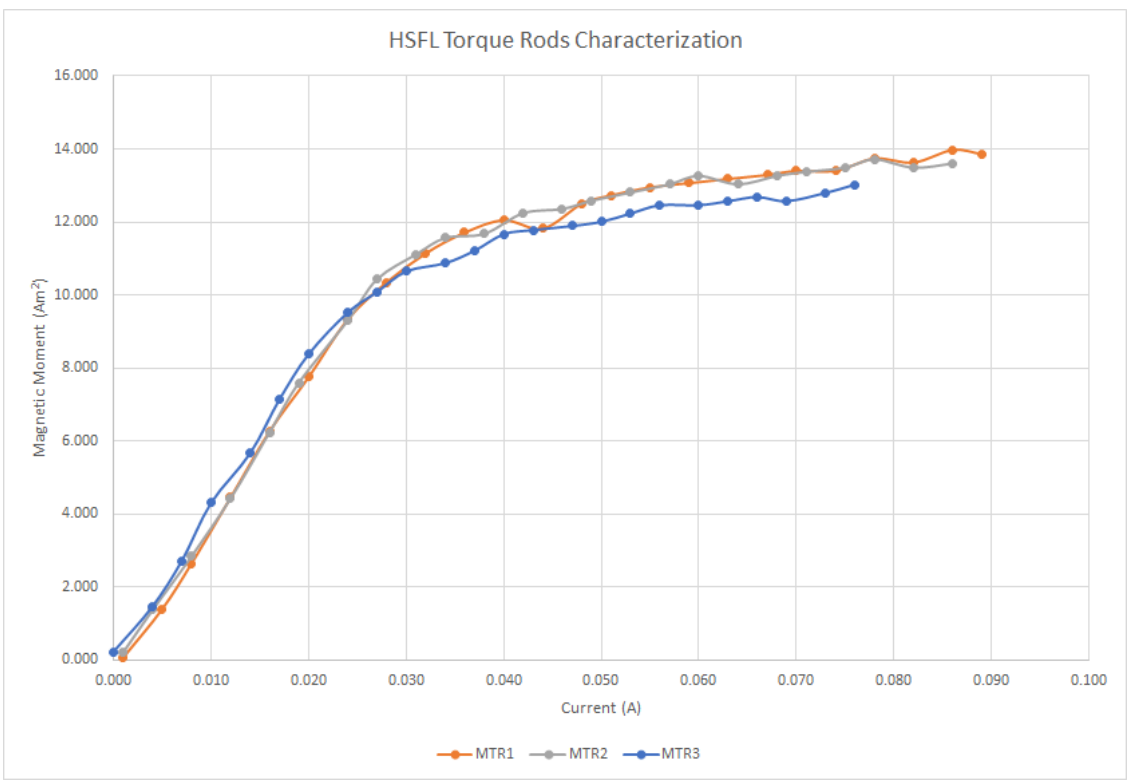

Figure 2.10: HSFL MTR characterization curves

primary path of communications into the FSW. All the FSW for HiakaSat including the satellite testbed is based on the COSMOS-core libraries which provide the agent functionality and many of the physical models, mathematical operations and drivers for the hardware components.

The COSMOS agents have been described previously in sight of other projects[38, 39, 43, 40] but in this section focus is given to the description of the agents that operate in the HiakaSat FSW and the satellite testbed. Starting with an overview of the agent architecture and then through the work more detailed descriptions will be added for each particular agents that is part of the ADCS.

Main thread. Each COSMOS agent runs on a infinite loop with multiple threads in parallel. Threads allow multiple functions to execute concurrently. This main thread is responsible for the agent communication process and for executing the requests that have been implemented. One of the default requests is to shutdown the agent so it is possible to remotely terminate an agent and start a new one when necessary. To start an agent automatically another agent or script must take the initiative to start the process.

Heartbeat. Each agent makes its presence known through the network of other agents using a broadcast message called the COSMOS Heartbeat. Any other agent wishing to communicate can listen to the COSMOS multicast address until it either receives a Heartbeat from the agent it is waiting for, or it times out. Agents can also received Heartbeats from different sources, even unexpected agents as long as they transmit with the same Heartbeat format. This makes the multi agent system flexible to changes. Once it acquires the desired Heartbeat, it sends a *Request* to the IP address and port taken from the Heartbeat. It then waits for a *Response* on that same IP address and port until it receives one, or times out.

All Requests and Responses are implemented in plain ASCII instead of binary to facilitate the inspection of messages.

Namespace. The COSMOS Namespace contains all the possible data structures necessary to operate a node, in this case the satellite. The Namespace is accessed inside the agents by the COSMOS Structure that is created at run time. Each entry on the Namespace represents a string or a number. The meaning of each entry is predefined such as the units and data type of each value. Examples of Namespace entries are "agent_addr" that contains the agent IP address, or "device_imu_omega_000" which contains the angular speed for the IMU and "node_loc_pos_eci" that contains the earth centered inertial (ECI) location information - the position in this case - for the current node. Each member of the Namespace is then mapped to a unique location in memory; the memory locations being drawn from a globally accessible storage space. This Namespace Map is stored as a simple list of entries, allowing either forwards or backwards translation between COSMOS Name and memory storage. The Namespace field and data are represented using JavaScript Object Notation (JSON).

Requests. One particular aspect of the COSMOS agents is that any operator or agent can send a Request to an agent to change its behavior or execute a particular function. This makes the agents very reactive to the environment they live in. Each agent has its default requests once started, see Table 2.5 for a list. Additionally the user can implement specific requests according to the functional needs for the agent.

The concept of agents and MAS was previously introduced but to summarize a software agent is a persistent program that is capable of acting in an autonomous and independent way to achieve its goals. Software agents have three main characteristics:

- **reactivity**: capable of receiving information from the environment and act on that

agent request	description
help	list of available requests for this agent
shutdown	request to shutdown this agent
idle	request to transition this agent to idle state
run	request to transition this agent to run state
status	request the status of this agent
getvalue {"name":value}	get specified value(s) from agent
setvalue {"name":value}	set specified value(s) in agent
listnames	list the Namespace of the agent
forward nbytes packet	broadcast JSON packet to the default SEND port on local network
echo utc crc nbytes bytes	echo array of nbytes bytes, sent at time utc, with CRC crc.

Table 2.5: Default COSMOS agent Requests

same environment using the received information in accordance to objectives/goals.

- **sociability**: can communicate with other agents by exchanging relevant information that is understood by each agent.

- **proactivity**: can decide and act autonomously (no external operation) to maximize the chances of achieving the overall goals.

In this sense the COSMOS agents developed for the HiakaSat fit the above criteria because these are:

- **reactive**: each agent can receive information from the space environment using the sensor drivers and/or from other agents and will send the control information to the actuators to move the satellite in space. They are also reactive because each agent has Requests available that permit any other agent to request specific behavior and/or data to other agents.

- **social**: every agent in the FSW can communicate with each other agent by exchanging relevant information using the regular SOH messages and others messages as requested.

Each agent sends are receives the messages using the same "language", in this case JSON so every agent can easily communicate.

- **proactive**: each agent monitors its behavior and evaluates its performance to determine if it is acting according to the predefined objectives. Each agent implements a state machine and different functionality for logical reasoning and decides what is the best mode to operate.

Because the agents implemented for the FSW interact directly with the satellite hardware a slightly different nomenclature can be used to differentiate this system from a purely MAS that does not interact with hardware. The MARS can be said to be a particular implementation of a MAS for robotic systems, such as a satellite. This nuance may be important to differentiate the pure AI MAS with the recent developments in the robotics community to increase the autonomy for make modern robots. Figure 2.11 shows how a MARS fits in the current context of AI and robotics.

An important advantage of the MARS is that these are typically compatible in different operating systems and also can be directly compiled for embedded processors. Most MARS are developed in C/C++ code while traditionally MAS are typically implemented in Java which will not perform as efficiently as C/C++ code since it is known to be very efficient when dealing with hardware such as drivers, networking, etc. Memory management is another reason why C/C++ will run faster on embedded processors. Also Java based MAS have been tested towards the development of distributed satellite systems but with some challenges[44].

With the COSMOS agents, hybrid peer-to-peer (P2P) distributed architectures can be developed where each computing platform can have one or more agents running. For this project we are focused on one computing platform running multiple agents. Each agent is

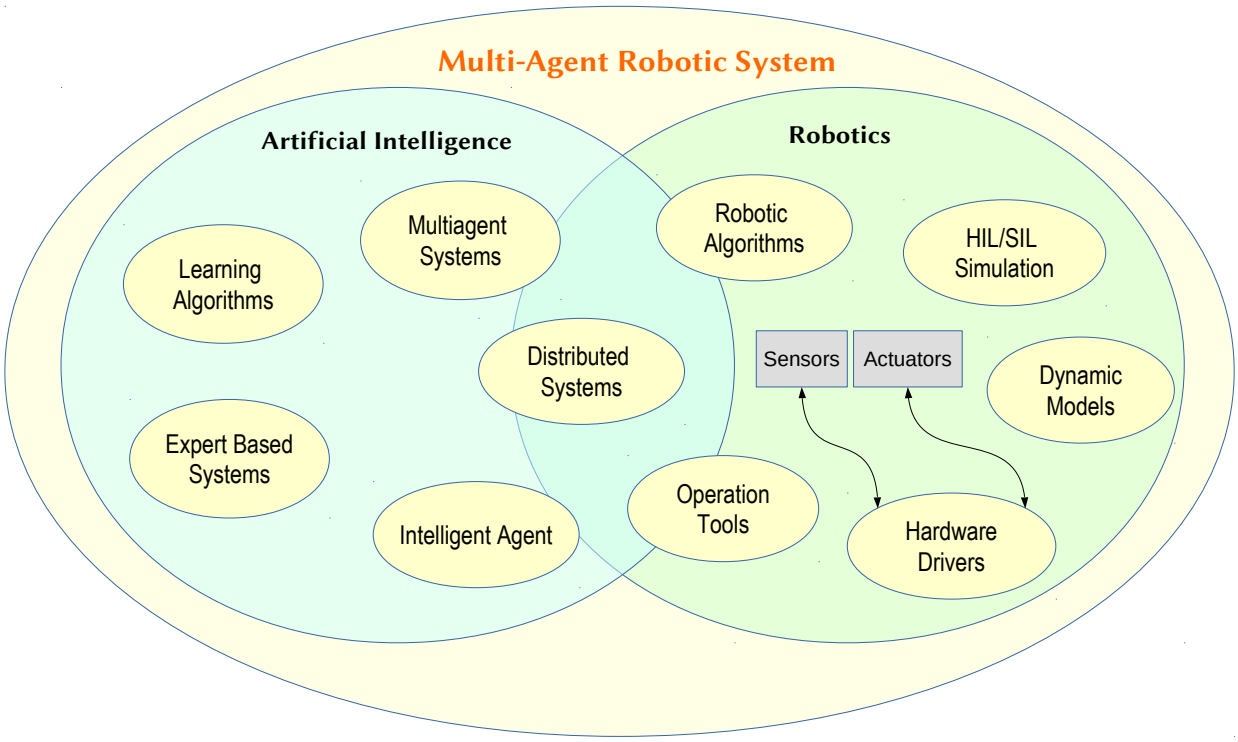

Figure 2.11: Context of Multi-Agent Robotic Systems

setup with its own unique name that is broadcasted across the node network. Each COSMOS agent has its own Namespace that can be shared across the other agents along with its SOH messages that are broadcasted at a regular interval. SOH messages send information that is a subset of the available Namespace with the most relevant information about the system, agent or node. The user can configure the SOH message format to add more information that is important. Every agent can collect data from other agents in the same network. If necessary, the agents can forward the information to another network as long as there is a network path available. This enables the creation of truly distributed MAS across the globe. This is specially useful for satellite missions where a satellite is the space node and the ground stations are the ground nodes. Since these are all physically distributed across large distances the COSMOS agents can bridge the physical gap by networking the information between them.

2.4 Satellite Attitude Control Implementation

The satellite control architecture is composed of multiple software agents that break down the complexity of the system by focusing on specific aspects such as driver operation or higher level logic and state machine implementation:

- **agent control**, where the main attitude control logic is implemented along with the state machine for the various control modes
- **agent navigation**, where the data fusion and attitude estimation happens with decision making on what control modes the spacecraft needs to be in
- **agent star tracker**, for the data collection and operation of the start tracker hardware and proper hardware monitoring and failure management, it also implements the sensor

calibration and conversion of the data represented in the sensor frame into the body frame

- **agent imu**, for the data collection and operation of the inertial measurement unit and proper hardware monitoring and failure management, it also implements the sensor calibration and conversion of the data represented in the sensor frame into the body frame

- **agent gps**, for the data collection and operation of the global positioning unit and proper hardware monitoring and failure management, it also implements the sensor calibration and conversion of the data represented in the sensor frame into the body frame

- **agent tcu**, for the operation of the magnetic torque rods and proper hardware monitoring failure management, it also implements the torque rods calibration

The ADCS agents architecture is seen in Figure 2.12. The other two agents that are important for the operation of this control scheme are the *executive agent* and the *state-of-health (SOH) agent*. The exec agent is responsible for commanding all the other agents operating in the satellite, including the navigation and control agents. The SOH agent collects the most important information from all the agents in the satellite, including the control agents for logging and forwarding for real time operations. Finally on the hardware side the sensors and actuators permit the closing of the control loop.

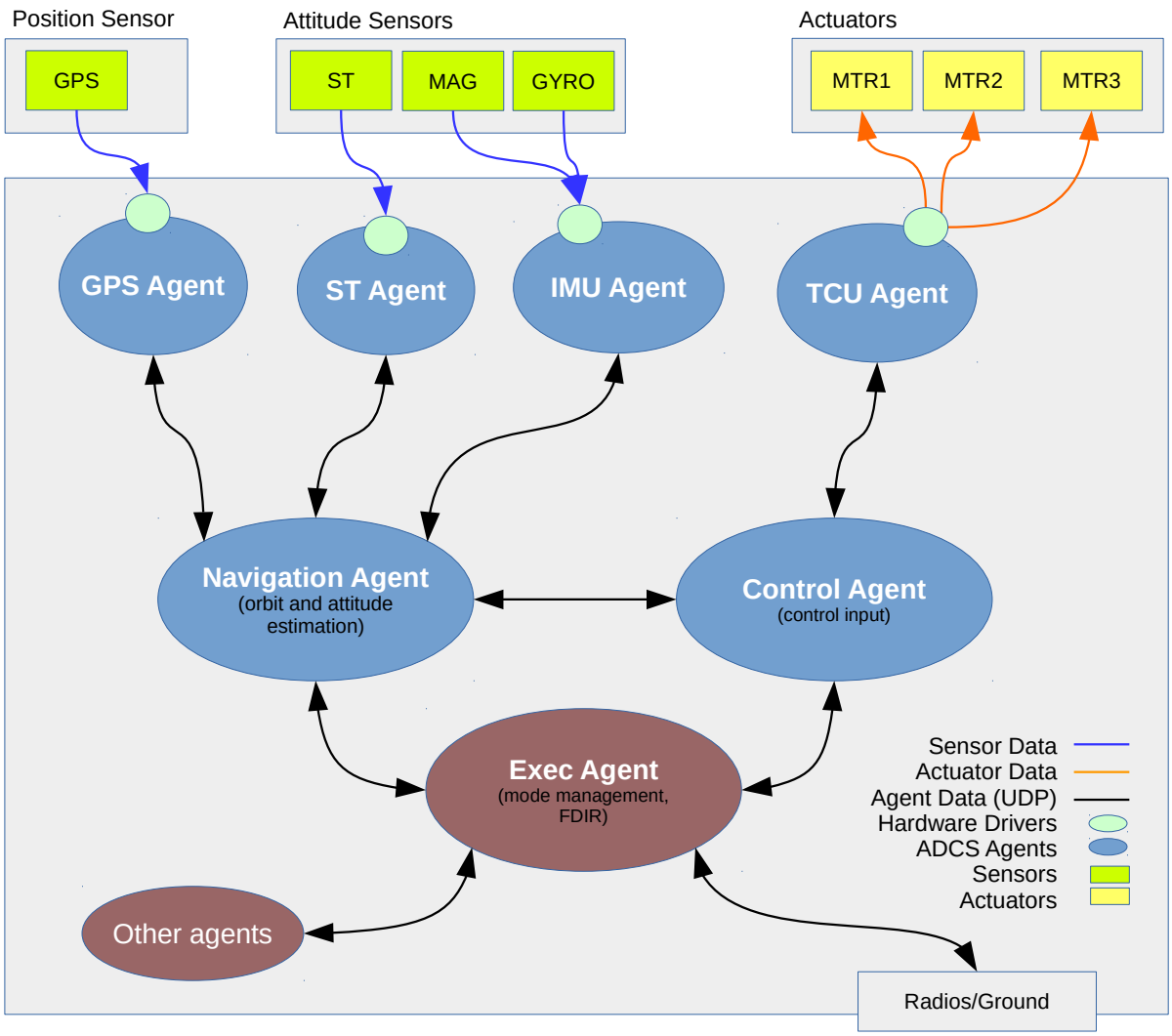

Figure 2.12: Satellite control MAS architecture

2.4.1 Attitude Dynamics

Euler's equation of the rigid body dynamics is the mathematical foundation that models the evolution of the attitude dynamics of the spacecraft:

$$\mathbb{I}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times \mathbb{I}\boldsymbol{\omega} + \boldsymbol{\tau} \tag{2.14}$$

The state vector ($\boldsymbol{x}$) of interest to this problem combines the attitude represented by a quaternion $\boldsymbol{q}$ and the attitude rate represented by the angular velocity vector $\boldsymbol{\omega}$, so $\boldsymbol{x} = [\boldsymbol{q}, \boldsymbol{\omega}]$. The angular velocity is explicitly expressed as $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]$. A quaternion is defined as a vector in four dimensional algebra $\boldsymbol{q} \in \mathbb{R}^4 \equiv [\boldsymbol{q}^T \quad q_w]^T$ where $\boldsymbol{q} = [q_x \quad q_y \quad q_z]^T$ and $\boldsymbol{q}^T\boldsymbol{q} = 1$. The vector part of the quaternion depends on the Euler axis of rotation (the generalized/principal axis that can represent the direction of any rotation) and the quaternion scalar only depends on the amount of rotation. The major advantage of using quaternions over the Euler angles is that with quaternions we prevent singularities in the kinematics equations. The attitude kinematics and dynamics for a rigid body are represented here using the quaternion formulation $\boldsymbol{q}$ with scalar in the last position of the vector (as opposed to the first position). It would be possible to arrange equivalent formulations for other attitude representations such as Euler angles, Rodrigues Parameters or even the Rotation Matrices. The attitude model considers a rigid body with internal control moments and a potential function that depends only on the attitude.

The attitude kinematics using the quaternion representation with scalar in the last entry

of the quaternion vector are given by the following equation:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\begin{pmatrix} 0 & -\omega_z & -\omega_y & -\omega_x \\ \omega_z & 0 & \omega_x & -\omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ \omega_x & \omega_y & -\omega_z & 0 \end{pmatrix}\boldsymbol{q} \tag{2.15}$$

The torque in Euler's equation can be expanded in two main categories: 1) external torque, $\boldsymbol{\tau}_{ext}$, (from gravity gradient, solar radiation pressure, etc.) and 2) internal torque also better known as control torque, $\boldsymbol{\tau}_{ctr}$, (from reaction wheels, magnetic torque rods, etc.). This way $\boldsymbol{\tau} = \boldsymbol{\tau}_{ext} + \boldsymbol{\tau}_{ctr}$. The gravity gradient torque can be modeled as

$$\boldsymbol{\tau}_g = 3\left(\frac{\mu_{Earth}}{r_{sat}^3}\right)\boldsymbol{z_o} \times \mathbb{I}\boldsymbol{z_o} \tag{2.16}$$

Using only the gravity gradient torque we can now write Euler's equation more explicitly:

$$\dot{\boldsymbol{\omega}} = \mathbb{I}^{-1}\left(-\boldsymbol{\omega} \times \mathbb{I}\boldsymbol{\omega} + 3\left(\frac{\mu_{Earth}}{r_{sat}^3}\right)\boldsymbol{z_o} \times \mathbb{I}\boldsymbol{z_o} + \boldsymbol{\tau}_{ctr}\right) \tag{2.17}$$

In the Euler dynamics equation the following are constants: the inertia tensor $\mathbb{I}$, Earth's gravitational parameter μ_{Earth}, and in some cases for simplicity one can assume that the satellite orbit is circular and so the satellite orbital radius r_{sat} is also constant. The values used for these constants are given in Table 2.6. The variables are the angular velocity of the satellite $\boldsymbol{\omega}$, the z-axis vector of orbital frame $\boldsymbol{z_o}$ and the control torque vector $\boldsymbol{\tau}_{ctr}$.

In the most complete simulations we use a more complete representation of the external disturbances in Euler's equation 2.14 the torque $\boldsymbol{\tau}$ is in fact a combination of multiple torques such as the control torque $\boldsymbol{\tau}_{ctr}$, gravitational torque $\boldsymbol{\tau}_G$, atmospheric torque $\boldsymbol{\tau}_A$ and

$\mu_{Earth} = GM = 398\,600.4418\,\mathrm{km}^3\,\mathrm{s}^{-2}$	Earth's gravitational parameter
$r_{Earth} = 6378.14\,\mathrm{km}$	Earth's radius
$r_{sat} = r_{Earth} + h_{sat}$	Satellite radius
h_{sat}	Satellite altitude for circular orbit

Table 2.6: Some physical parameters for the Earth and satellite

radiational torque $\boldsymbol{\tau}_R$:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{ctr} + \boldsymbol{\tau}_G + \boldsymbol{\tau}_A + \boldsymbol{\tau}_R \tag{2.18}$$

$$\tag{2.19}$$

The satellite attitude dynamics are mostly affected by the three MTRs. The magnetic actuation model requires the knowledge of the local magnetic field in the satellite orbit and this information is captured by the magnetometer sensor inside the satellite or inferred by the satellite position in orbit using the most recent World Magnetic Model (WMM). The satellite attitude sensors collect the information in the local sensor frame and then the agent must convert the data into to the body frame for the control loop to decide the best control strategy. The satellite attitude is expressed using the quaternion which represents the rotation of a given frame into the satellite body frame. A definition of the reference frames used to represent the attitude dynamics is given as follows.

Body Frame. The body frame represented by the symbol F_B is aligned with the principal axis of inertia of the satellite with the origin at the center of mass. The axes about which the moment of inertia tensor is a diagonal. This is a Cartesian frame and the principal directions are arbitrarily assigned but defined as represented in Figure 2.13 where the X direction if going to be the nominal direction of travel. The Z axis is directed towards the

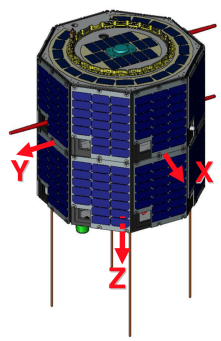

Figure 2.13: Satellite body frame

boresight of the main science payload and the Y axis is determined by the triad of the right handed Cartesian frame.

Orbit Frame. The orbit frame is represented by F_O is also known as LVLH frame and moves as has its origin with the center of mass of the satellite. The Z-axis is directed towards the geocenter and the Y-axis is normal to the orbital plane, this is in fact the cross product of the satellite position vector with the velocity vector, and +X is given by the triad (it coincides with the direction of travel for circular orbits). This frame has angular rate $\boldsymbol{\omega}_O$ in the orbital plane around the Earth's center.

Inertial Frame. One of the most important frames is the Inertial frame where the attitude dynamics are properly represented without fictional forces. There are various definitions for inertial frames, for this work we use the Geocentric Celestial Reference Frame (GCRF) that is the Earth-centered counterpart of the International Celestial Reference Frame (ICRF). This is one of the ECI frames. In this work we will interchangeably use GCRF and ECI frames. This frame is represented by the symbol F_I and has its origin at the Earth's center. The X-axis is directed to the vernal equinox (the first point of Aries), the Z-axis is directed towards the celestial north pole and finally the Y-axis completes the right handed axes system. Sometimes we also refer to the Inertial frame as the ICRF that is aligned with the axes

of the ICRF but with origin at the center of the Earth. This is an adaptation of the true definition of the ICRF since in reality its origin is at the barycenter of the Solar System.

Frame Transformations. We require proper definitions of frame transformations for the satellite equations of motion and to convert the various vectors from the sensors and the control input between the different frames. We use Euclidean linear frame transformations, more specifically a rotation transformation represented by a rotation matrix. (Euclidean transformations preserve the length and angle measures and can be translation, rotation and reflection.) The interpretation of a rotation matrix is easily subject to ambiguities. To properly define the meaning of a rotation matrix we use the following terminology. When the coordinates of a point change due to the rotation of the coordinate system we use the term *passive transformation*, and *active transformation* when the coordinate system is fixed but the point rotates. We sometimes use an abbreviated terminology, *rotation* means by default an active transformation and *transform* is a passive transformation.

To represent a vector in Frame F_b actively transformed to F_a we use the rotation matrix $R_{a\leftarrow b}$. All proper rotation matrices are orthogonal meaning that $R^T_{b\leftarrow a}R_{b\leftarrow a} = I$ and $R_{a\leftarrow b} = R^T_{b\leftarrow a}$. A rotation matrix can be simplified and equivalently represented by a quaternion. The rotation matrix R representing an active transformation can be written using the quaternion elements:

$$R = (q_w^2 - \boldsymbol{q}^T\boldsymbol{q})I + 2\boldsymbol{q}\boldsymbol{q}^T + 2q_w\boldsymbol{Q} \tag{2.20}$$

where

$$Q = \begin{pmatrix} 0 & q_z & -q_y \\ -q_z & 0 & q_x \\ q_y & -q_x & 0 \end{pmatrix} \tag{2.21}$$

2.4.2 Navigation

The satellite navigation consists of determining the state estimate for the satellite attitude and position (orbit determination). This information is sent to the controller so that the algorithm can compute the control input to correctly point the satellite. The attitude control must know the orbital position of the satellite to maintain nominal pointing (tracking LVLH frame) because without this information the satellite does not know where the center of the Earth is with respect to the satellite frame and nominal operation is not possible. The navigation algorithms are implemented in the navigation agent that after calculating the satellite state will broadcasts that information which in turn will be collected by the control agent.

Orbit Determination

One of the goals of the navigation algorithm is to determine the orbital state of the satellite. The orbital state is a vector composed of the position and velocity at current time $\boldsymbol{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]$. This state is determined either by collecting GPS data[45] or by using an on-board orbital propagator in the case when GPS information is not available[46]. Without GPS data the onboard orbit propagator requires an initial state estimate that can be used for extrapolation. Such estimate usually is computed by ground-tracking stations using laser, radar, Doppler among other methods. Once computed the state estimate is uploaded

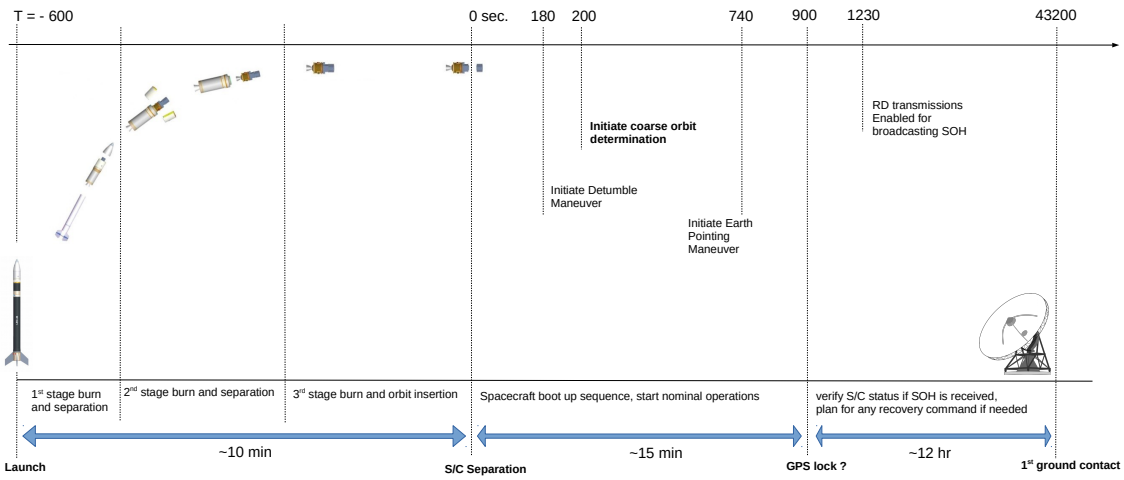

Figure 2.14: Initial orbit sequence for HiakaSat

to the satellite, usually in the form of a two-line-elements file, during the first contact with the satellite. This position state estimate is frequently computed and updated during the satellite mission lifetime. Once the satellite is deployed the process of computing a state estimate and sending the information to the satellite can take several hours. During this time the satellite is "blind" with respect to its orbital information. In the case of HiakaSat, the satellite only has access to the ground station 12 hours after deployment as shown in time sequence in Figure 2.14.

Typically there are three main stages for orbital state determination. The first is when the satellite is separated from the launch vehicle and the system boots for the first time in orbit. This is considered a cold start for GPS receiver because there is no knowledge of the GPS satellite constellation (also known as almanac) at this point. The orbital state from the GPS is undetermined until a first fix is obtained. The time to acquire a GPS fix is known as time to first fix (TTFF) and for space borne GPS receivers it is typically in the order of several minutes. The "blind" orbital state situation can be mitigated using with a

dead reckoning orbital propagator if an initial orbit insertion state is known a priori. The second stage for the orbit state determination occurs when the GPS has acquired a lock and is nominally available. Then the orbital state is directly used or mixed with an orbital propagator to improve the position accuracy. Finally there will be situations when the GPS information is temporarily unavailable (for a few seconds or minutes) during nominal operations. This situation is resolved using a standard orbital propagator such as the "Gauss Jackson" propagator[47]. For these propagators the current absolute time information must available and the computer must be capable of running the algorithm in real time without affecting the remaining processes.

In this section the first scenario is addressed to improve the chances of mission success for HiakaSat once separated from the launch vehicle. If the GPS receiver fails during the first orbit the satellite may not be able to point to the Earth and so it reduces the chances for a ground station contact. This effect drastically reduces the chances to recover the satellite in the eventuality that a command from the ground must be sent to the satellite to recover from any other possible failure. But a dead reckoning propagator would allow the satellite to know its approximate position with respect to the Earth after a specified elapsed time from deployment and by knowing the position state the chances of mission success increase. The satellite can more easily point the antennas to the ground and also increase its beacon activity when it is close to the primary ground stations. One example of a failed mission immediately after deployment because of a missing command is the NASA Spartan 201 mission. After its release from the Space Shuttle Columbia the satellite failed to initiate its pointing maneuver because of an onboard anomaly[48].

One of the challenges of acquiring a GPS signal during satellite deployment is due to the fact that the satellite must have a stable attitude so that its antenna is pointing to-

wards the GPS constellation for proper signal reception. But the satellite will be typically tumbling in the initial section of the orbit until the detumble maneuver is complete. When the GPS receiver antenna is pointing towards the Earth it will not receive any signals from the GPS constellation, so no lock is possible. If the tumbling continues it may be that the GPS receiver will never get a proper lock because of the speed that the satellite is traveling and the intermittent signal acquisition. Typical TTFF for a GPS receiver in space and in ideal conditions (antenna always pointing at GPS satellites) varies between 10 to 20 minutes because of cold start conditions. The main reason being that the GPS almanac needs to be downloaded (see for example pg. 312 of [49]). During this time the satellite has traveled thousands of kilometers and warm-up algorithm for the GPS receiver may fail[50]. If possible it is important to feed the GPS with an estimated state vector, which will speed the lock acquisition of the GPS.

There are mainly two method for low earth orbit (LEO) orbit determination[51]. The conventional method, which has been used since the beginning of the space exploration era, uses ground stations to determine the orbital state of the satellite using two way ranging with radars, lasers or similar measurement methods. Today, such methods have a typical accuracy of 5-1000 meters and have a required data duration of 1-2 days[52] depending on the method used. By using ground observation methods only it may take from a few hours up to a few days to upload the orbit information. The second option uses GPS based lock acquisition with a typical accuracy of 5 to 20 meters[45] and the position information may be available sometime between 10 to 30 minutes, depending on the GPS receiver[53]. Both methods do not provide the orbit state information immediately after launch, with the best case being the GPS receiver if no failures occur. To overcome the time lag limitation and of possible failure of the initial GPS lock, a dead reckoning propagator is implemented that uses the best orbital deployment estimate determined before launch. To the best knowl-

edge of the author this approach has not yet been developed because most missions can afford to have multiple tracking stations and redundant GPS units that increase the chances to quickly acquire the orbit state information. But this problem affects particularly small satellites because of limited resources such as the availability of ground tracking stations and more reliable and redundant GPS receivers.

The dead reckoning orbital propagator is entitled "pseudo-ECI" propagator because the state is propagated using an alternative ECI reference frame determined at the initial orbital insertion point. The "pseudo-ECI" propagator is based on the Keplerian equations. The initial estimate of the orbital state may not be very precise but the information can prove to be invaluable to approximately point the satellite at the Earth during the initial phase of the mission. If the satellite does not know its current orbital state, the control algorithm for nadir pointing will fail and ultimately the mission may fail since the satellite cannot reliably point to the Earth and ground station contact is not possible. This situation is sometimes called "lost in space".

The "pseudo-ECI" orbital propagator (pECI) is described in Algorithm 1. One critical aspect of this propagator is to measure mission elapsed time which can be taken care of using the OBC clock, the other is to have the best possible estimate of the orbital state insertion, position and velocity loaded on the satellite prior to launch. The pECI is developed taking advantage of the orbital propagation methods that assume an inertial frame by assuming the pECI frame is coincident with the earth centered earth Fixed (ECEF) frame at the time of deployment. The algorithm outputs the geocentric position using the pseudo Earth Centered Inertial frame computed by 2. The orbital state is computed using Algorithm 3 which is then converted to geocentric position using 4. As time evolves the ECEF frame will rotate at the Earth daily rotation rate but the pECI will keep its fixed orientation in space,

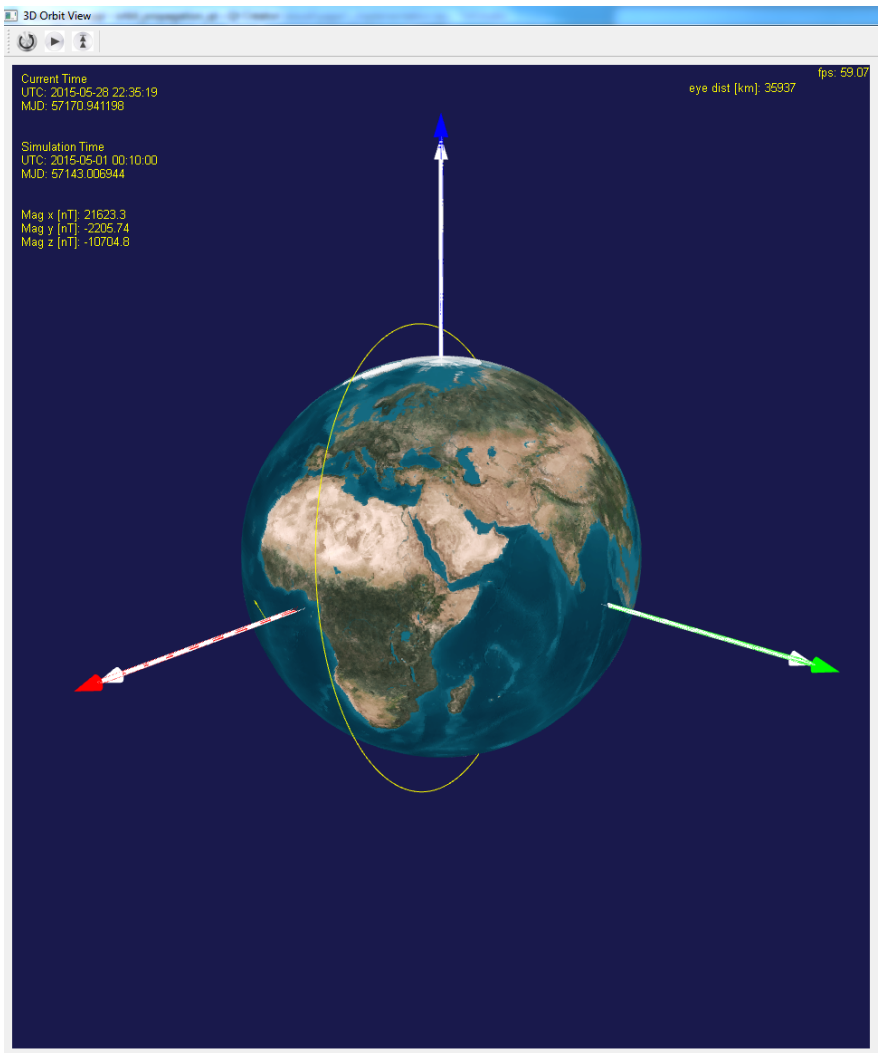

Figure 2.15: Demonstration orbit computed by the "pseudo-ECI" propagator

hence the reason for the term "pseudo-ECI". It is important to note that this is only an approximate inertial frame and this method will only work for a few orbits after deployment and gradually the precision will be lost. This pECI propagation does not take into account the Earth's perturbations or other celestial bodies perturbations that are reflected on the Nutation and Precession.

Algorithm 1 Compute geocentric position using pseudo Inertial Frame

```
procedure computeGeocentric
    if t initial then
        computePseudoEci
    end if
    loop agent
        computeOrbitalState
        convertPseudoEciToGeocentric
    end loop
end procedure
```

Algorithm 2 Compute pseudo ECI

```
function computePseudoECI(r_ecef, v_ecef)    ▷ input: position and velocity in ECEF frame
    offset = [0, Ω_earth ‖r_ecef‖ , 0]    ▷ Compute velocity offset for pECI frame
    dot = i_1 * r_1 + i_2 * r_2    ▷ dot product
    det = i_1 * r_2 − i_2 * r_1    ▷ determinant
    α = atan2(det, dot)    ▷ angle that separates pECI from ECEF
    M = AngleAxis(α, UnitZ() );    ▷ Rotation matrix that represents the new pECI
    v_peci = M × offset + v_ecef    ▷ intial velocity in pECI
    r_peci = r_ecef
    Return v_peci, r_peci    ▷ return values are stored in class
end function
```

Attitude Determination

For the navigation solutions a nonlinear extension to the Kalman filter is used, the Extended Kalman Filter (EKF). This version of the Kalman Filter linearizes about an estimate of the current mean and covariance for better estimates of a nonlinear model.

Algorithm 3 Compute Orbital State Vector after ΔT

```
function computeOrbitalState(r_0, v_0, ΔT)
    keplerian = rv2keplerian(r_0, v_0)   ▷ Compute keplerian elements from r and v vectors
    n = sqrt(ν / a^3)                    ▷ mean motion
    if e ≠ 0 then
        E_0 = nu2anomaly(e, nu)
    end if
    if e ≤ 1.0 then
        M_0 = E_0 − e × sin(E_0)
        M = M_0 + n * dt                 ▷ compute Mean anomaly
    end if
    E = eccentricAnomaly(M, e)           ▷ compute eccentric anomaly E
    if e ≠ 0 then
        nu = anomaly2nu(e, E)            ▷ find true anomaly
    end if
    r,v = keplerian2rv(k)                ▷ convert keplerian vector to cartesian
    Return r, v                          ▷ return new state vector
end function
```

Algorithm 4 convert state vector from Pseudo ECI To Geocentric Frame

```
function convertPseudoEciToGeocentric(ΔT, r_ECI)
    α = ω_⊕ ΔT                           ▷ angle between ECI and ECEF
    M = AngleAxis(−α, UnitZ())           ▷ Rotation matrix that represents the new pECI
    r_ecef = M r_eci                     ▷ new position vector in ECEF
    alt = ‖r_ecef‖ − r_⊕                 ▷ altitude
    lat = −[arccos(y_ecef / ‖r_ecef‖) − π/2]   ▷ latitude
    lon = arctan2(y_ecef, x_ecef)        ▷ longitude
    Return alt, lat, lon                 ▷ return state vector in geocentric coordinates
end function
```

Consider the discrete non-linear system dynamics and the observation model with Gaussian noise:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= f\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{y}_k &= h\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k \end{aligned} \tag{2.22}$$

The state vector x is defined by the attitude quaternion and angular velocity vectors $\boldsymbol{x} = [q_0, q_1, q_2, q_3, \omega_x, \omega_y, \omega_z]$. The initial state vector is defined by a random attitude and angular velocity $\boldsymbol{x}_0$ with known mean and covariance. The sensor measurement state vector $\boldsymbol{y}$ is given by the direct measurements of the star tracker and gyroscope $\boldsymbol{y} = [q_0, q_1, q_2, q_3, \omega_x, \omega_y, \omega_z]$ with known sensor noise.

The following equations summarize the EKF steps. The state transition and observation matrices are defined to be the following Jacobians:

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}} \tag{2.23}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{2.24}$$

With the Jacobian matrices the state and covariance estimates must be computed:

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}) \tag{2.25}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^{\top} + \boldsymbol{Q}_{k-1} \tag{2.26}$$

To compute the innovation and innovation covariance we have:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{2.27}$$

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top + \boldsymbol{R}_k \tag{2.28}$$

With the previous steps it is now possible to compute the Kalman gain:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1} \tag{2.29}$$

Finally we can compute the updated state and covariance estimates:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k \tag{2.30}$$

$$\boldsymbol{P}_{k|k} = (\boldsymbol{1} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1} \tag{2.31}$$

The EKF is an iterative prediction process where at each step we calculate the state estimate x, the Kalman gain K and the process covariance matrix P.

2.4.3 Control

Detumble Maneuver

When the satellite is separated from the launch vehicle it has a residual angular velocity that must be dissipated. The process of dissipation of the angular velocity is the detumbling of the spacecraft. The detumble maneuver is the first critical maneuver of the satellite and is the first action started by the control agent after the satellite is separated. This maneuver reduces the angular rate of the satellite to almost zero in the inertial frame reference. It is only after this maneuver that the control agent checks its status and determine if it is possible to transition into nominal pointing mode, which is nadir pointing (satellite body z

axis points towards Earth). It is assumed that there can be a 5 deg/sec angular rotation ($\omega \leq [5, 5, 5] deg/s$)in each axis when the satellite is deployed, these are typical maximum values for launch vehicle separation for satellites at orbit insertion. The satellite must detumble in minimum time so that operations may start as soon as possible. The following is a brief description of the derivation of the optimal control for a minimum time detumble maneuver. For this particular problem of time-optimal control one needs to drive the spacecraft initial angular velocity to zero in minimum time: $\boldsymbol{\omega}_0 = \boldsymbol{\omega}(0) \rightarrow \boldsymbol{\omega}(t_f) = \mathbf{0}$.

The problem statement to detumble the spacecraft in minimum time is as follows. Consider the spacecraft dynamical model described by Euler's equation of the rigid body rotation

$$\dot{\boldsymbol{\omega}} = -\mathbb{I}^{-1}(\boldsymbol{\omega} \times \mathbb{I}\boldsymbol{\omega}) + \mathbb{I}^{-1}\boldsymbol{\tau} \tag{2.32}$$

where $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T \in \mathbb{R}^3$ is the state to control (angular speed), $\mathbb{I}$ is the inertia tensor and $\boldsymbol{\tau} = [\tau_1, \tau_2, \tau_3]^T \in \mathbb{R}^3$ is the control input. To simplify the following derivations the cross term of the Euler's equation is defined as

$$\Omega(\boldsymbol{\omega}) = -\mathbb{I}^{-1}(\boldsymbol{\omega} \times \mathbb{I}\boldsymbol{\omega}). \tag{2.33}$$

The boundary conditions are dictated by the initial angular rate of the separation of the launch vehicle represented by $\boldsymbol{\omega}(0) = \boldsymbol{\omega}_0 = [\omega_{1_0}, \omega_{2_0}, \omega_{3_0}]$ and the final condition is $\boldsymbol{\omega}(t_f) = \boldsymbol{\omega}_f = [0, 0, 0]$. For the minimal time optimization the following cost function is used that is subject to the zero angular velocity condition $\boldsymbol{\omega}(t_f) = \mathbf{0}$ combined by the fact that the control torque is bounded by $\|\boldsymbol{\tau}_0\| \leq \tau_{max}$. The transversality condition[54] is $\lambda(t_f) = -1$.

$$J = \int_0^{t_f} 1 dt \tag{2.34}$$

The Hamiltonian system is defined by the following equation:

$$H = 1 + \boldsymbol{\lambda}^T \Omega\left(\boldsymbol{\omega}\right) + \mathbb{I}^{-1} \boldsymbol{\lambda}^T \boldsymbol{\tau} \tag{2.35}$$

The costates vector is represented by $\boldsymbol{\lambda} \in \mathbb{R}^3$, then by the optimal control the costates must satisfy the following equations:

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{\omega}} = -\frac{\partial \Omega^T}{\partial \boldsymbol{\omega}} \boldsymbol{\lambda} \tag{2.36}$$

The derivation of the following equation is extensive but the final form is

$$\frac{\partial \Omega^T}{\partial \boldsymbol{\omega}} = \mathbb{I}\left[\boldsymbol{\omega}\times\right]\mathbb{I}^{-1} - \left[\mathbb{I}\boldsymbol{\omega}\times\right]\mathbb{I}^{-1} \tag{2.37}$$

by solving the optimal control problem the solution is actually elegant and robust[55]:

$$\boldsymbol{\tau}^* = -\frac{\mathbb{I}\boldsymbol{\omega}^*}{\|\mathbb{I}\boldsymbol{\omega}^*\|}\tau_{max} \quad \forall t \in [0, t_f) \tag{2.38}$$

It is this control solution that is implemented to compute the desired torque for the attitude control agent during the detumble maneuver. One must note that $\boldsymbol{\tau}^*$ must be converted to a feasible torque that the three torque rods can produce. To find the feasible torque, the magnetic moment $\boldsymbol{M}$ must be found which the torque rods can produce given the instantaneous magnetic field $\boldsymbol{B}$. The best possible moment resides on a plane that is perpendicular to the local geomagnetic field vector in the body frame. This relation can be

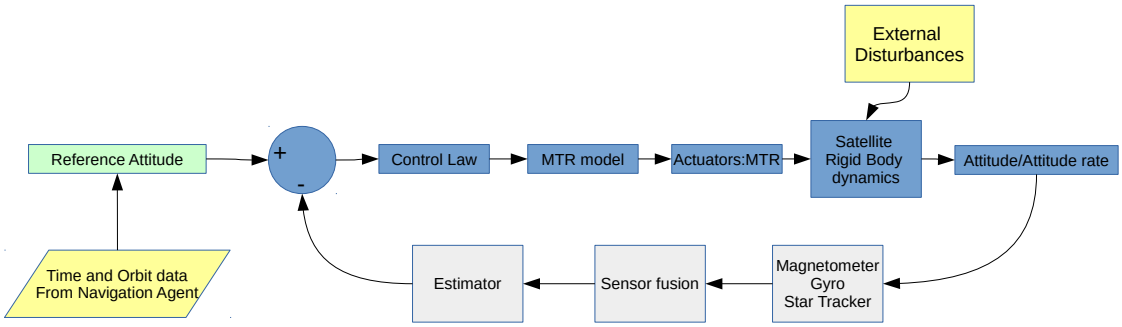

Figure 2.16: ADCS functional block diagram

computed by

$$\boldsymbol{M} = \frac{\boldsymbol{B} \times \boldsymbol{\tau}^*}{|\boldsymbol{B}|^2} \tag{2.39}$$

Pointing Maneuver

The control agent also contains the algorithm to compute the desired control moment for pointing the satellite to any desired orientation (pointing mode). The attitude control functional block diagram is shown in Figure 2.16 for the pointing tracking maneuver.

The attitude control scheme is based on feedback of trajectory tracking errors. The control law is designed for global tracking of the desired attitude and angular velocity trajectories. The control authority is singular when the desired control vector is in the direction of the local geomagnetic field, this is one of the major difficulties when dealing with magnetic torque rods precisely because of the cross product in eq 2.13. To maintain the nominal attitude mode a pitch rate of approximately 0.063 deg/s is required, which is overcome by the control actuators.

The control input is computed by a PD-type control law that balances the proportional difference from the desired attitude with the current attitude (attitude error q_e) and the difference from the first order attitude derivative (attitude rate error $\dot{q}_e$)

$$\boldsymbol{\tau}_{ctr} = -k_p \boldsymbol{q}_e - k_d \dot{\boldsymbol{q}}_e \tag{2.40}$$

This control input is then sent to the torque-rod control unit (TCU) agent that expects a moment in units of Am^2.

2.4.4 ADCS Agents

Agent: Control

The main goals of the attitude control agent is to collect the state information from the navigation agent, determine the appropriate control mode and compute an efficient control input for the actuators. Figure 2.17 shows the diagram of the agent control functionality. The control is computed using the control algorithms described in section 2.4.3 with the main functions to detumble and point the satellite.

When initialized the control agent searches for the other agents required for proper control operations. These are agent Navigation, TCU, IMU, star tracker (ST) and GPS. If a specific agent is not found a request is sent to start it. Once all the agents have started there is a regular check at every minute on each agent to determine connectivity. Next all the necessary agent *Requests* are added to the existing default agent Requests. The new requests are listed in Table 2.8. The next step at the initialization of the agent is the addition of specific Namespace data fields to the SOH. Finally two new threads are initialized, one to filter (poll) specific heartbeat data sent by the Navigation Agent and another to send

agent function	description
Reactive	collect state estimate, compute control input, send control input to other agents (specifically agent TCU)
Social	send and receive SOH messages and/or requested data
Proactive	state machine with decision process to maintain control objectives

Table 2.7: Description of control agent qualities

Requests to Agent TCU with the most updated control input.

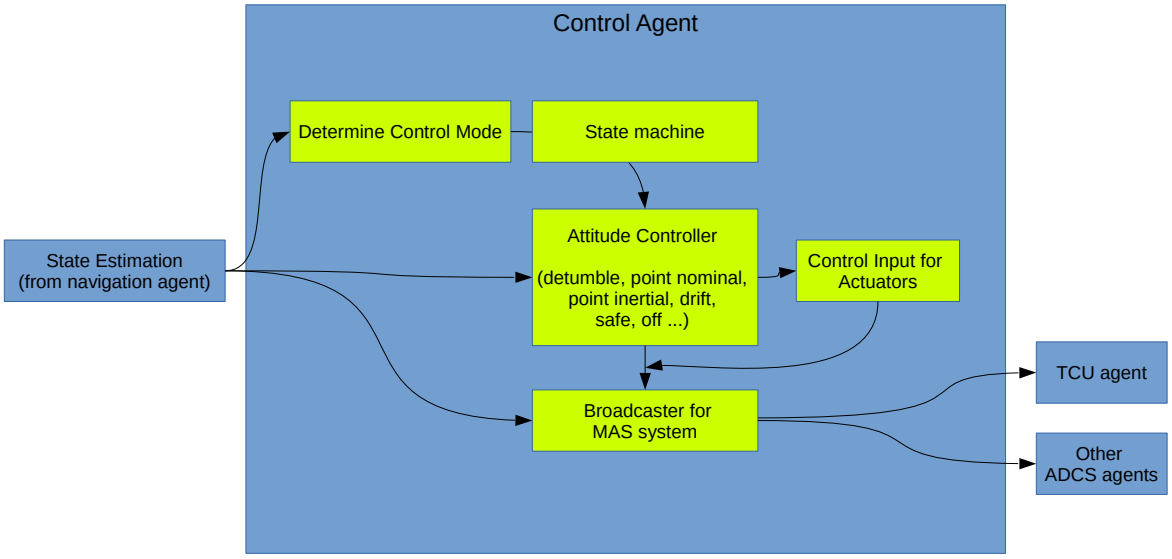

Figure 2.17: Attitude Control Agent Architecture

Agent: Navigation

The main purpose of the navigation agent is to collect relevant data from the other agents that act in behalf of the attitude (star tracker agent and IMU agent) and position sensors

agent request	description
desired attitude	set desired target attitude and velocity
mode	change control mode (detumble bdot, detumble omega, point inertial, point lvlh, pd, off, auto, manual, speedup).
log	start logging all the data
debug	turn on/off debug messages (gps, gps raw, st, st raw, imu, control, estimator, point, general, log, lsfit, off)

Table 2.8: List of control agent requests

(GPS agent). The navigation agent filters the incoming data to provide a best state estimate (position and attitude) and then forwards that information to the control agent. The navigation agent also includes the fault tolerant algorithm to maintain performance in the event of sensor failures. Typically the navigation functions for satellite ADCS reside within the monolithic program in a closed control loop scheme tightly coupled with the control algorithms, but by using the MAS the monolithic paradigm is shifted towards a more modular solution and the software is more easily upgradeable and maintained. One possible drawback is that there needs to be a mechanism to handle unexpected delays in the communication of the incoming sensor data. For this particular navigation agent the mechanisms developed to account for these uncertainties on the incoming data is an extrapolation algorithm using a least squares fitting for each data set that is evaluated at specific intervals when the complete state estimate is computed.

If for some reason there is a radiation hit on the memory space where the navigation agent is running the other agents can shut down the navigation agent and initiate a new navigation agent running in another memory space that is intact. This event will impact the production of new state estimates but will not necessarily take down the whole ADCS system. For instance the control agent detects that there is a failure on the navigation agent

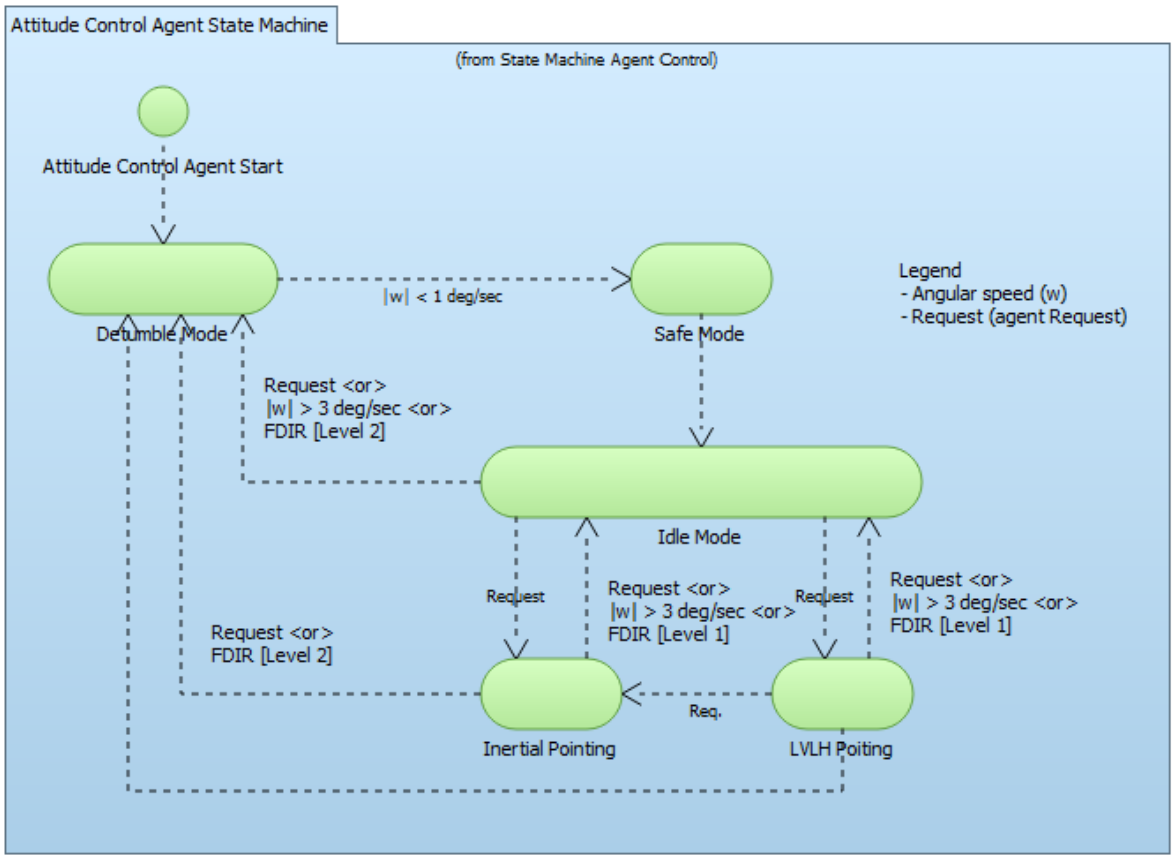

Figure 2.18: Attitude Control Agent State Machine

and stops the control action. The executive agent (not part of the ADCS will then take responsibility to restart the navigation). This is one of the aspects of flight software robustness brought by the MAS.

Table 2.9 lists the navigation agent qualities that make it part of the MAS. Table 2.10 lists the navigation agent requests implemented. Figure 2.20 shows the state machine for the navigation agent.

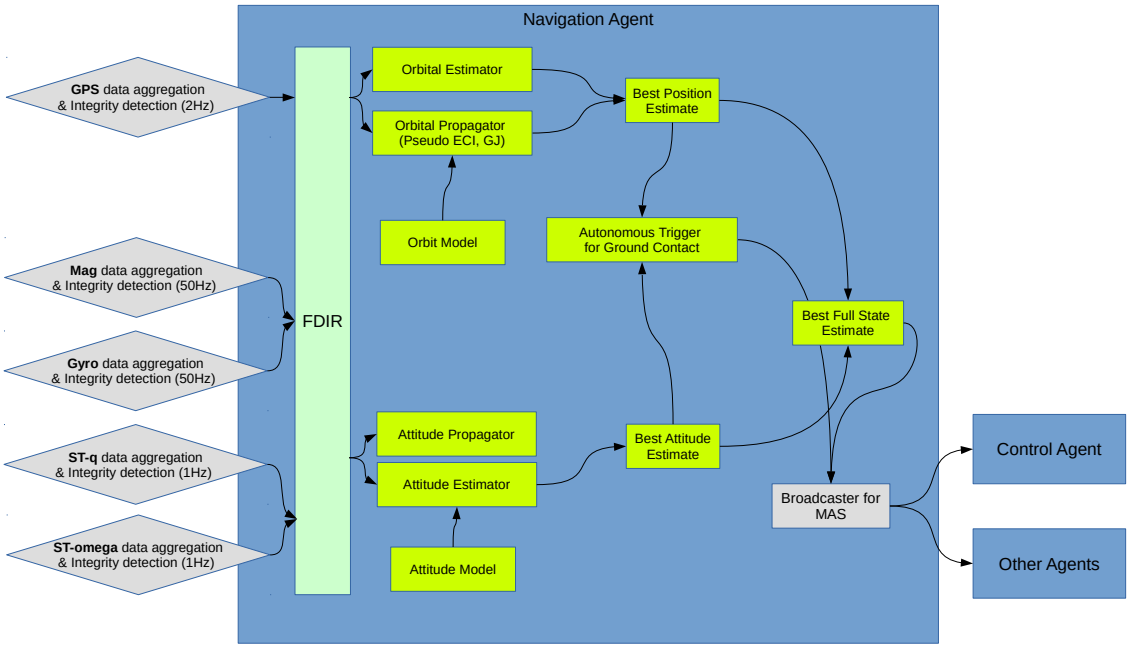

Figure 2.19: Navigation Agent Architecture

agent function	description
Reactive	collect sensor data from other agents, compute state estimate, send state estimate to other agents (specifically agent control)
Social	send and receive SOH messages and/or requested data
Proactive	state machine with decision process to maintain navigation modes/objectives

Table 2.9: Description of navigation agent qualities

agent request	description
mode	set estimator mode (0 = NONE, 1 = LSFIT, 2 = EKF).
debug	turn on/off debug messages (gps, gps_raw, st, st_raw, imu, estimator, general, log, lsfit, off)

Table 2.10: List of navigation agent requests

Agent: Star Tracker

The *star tracker agent* is described in this section. Its main purpose is to handle the connection with the star tracker hardware and collect the attitude state produced by the star tracker and forward that information to the other control agents, specially the navigation agent. The other main purpose of this agent is to monitor the health of the star tracker hardware and report that to the system. The *star tracker agent* also takes care of the calibration and the frame conversion.

Table 2.11 lists the star tracker agent qualities that make it part of the MAS. Table 2.12 lists the star tracker agent requests implemented. Figure 2.21 shows the state machine for the star tracker agent.

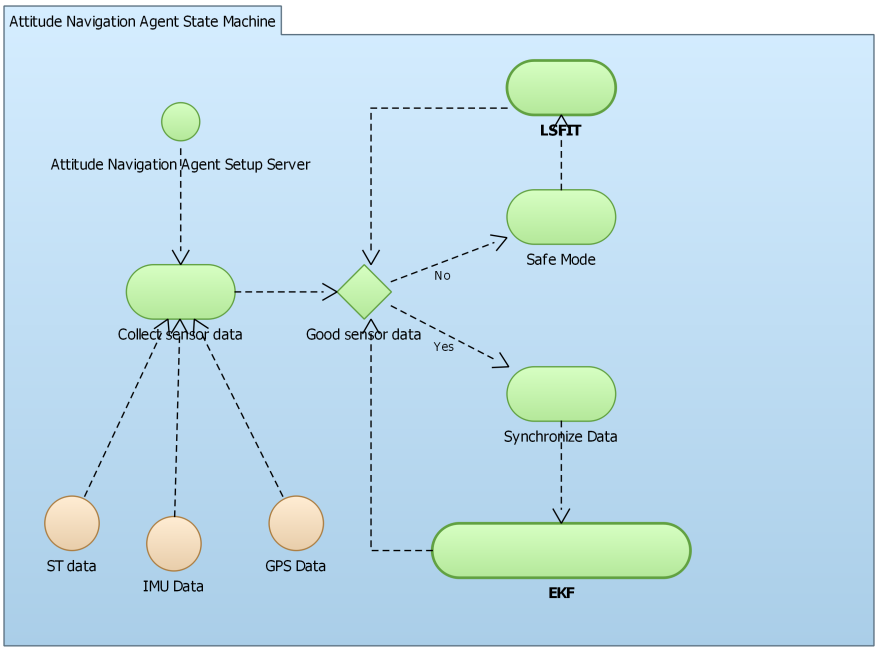

Figure 2.20: Navigation agent state machine

Agent: Inertial Measurement Unit

The main goal for the IMU agent is to collect the magnetic field B and angular rate ω from the IMU hardware and forward that information to the other agents, especially the navigation agent. The other main purpose of this agent is to monitor the health of the IMU hardware and report that to the rest of the system. The IMU agent also takes care of the calibration and the frame conversion for the sensor data.

Table 2.13 lists the IMU agent qualities that make it part of the MAS. Table 2.14 lists the IMU agent requests implemented. Figure 2.22 shows the state machine for the IMU agent.

Agent: Global Positioning System

The main purpose of the GPS agent is to collect the position information from the GPS hardware and forward that information to the other control agents, especially the navigation agent. The other main purpose of this agent is to monitor the health of the GPS hardware

agent function	description
Reactive	collect star tracker sensor data, forward the data to other agents (specifically agent navigation)
Social	send and receive SOH messages and/or requested data
Proactive	state machine with decision process to maintain operation modes/objectives

Table 2.11: Description of star tracker agent qualities

agent request	description
set mode	set ST operation mode (0 = ST closed, 1 = ST open, 2 = ST simulated).
debug	turn on/off debug messages
debug raw	turn on/off raw data from the ST
q	request quaternion
omega	request angular velocity (ω)
att	request attitude state (quaternion q, omega ω)

Table 2.12: List of star tracker agent requests

and report that to the rest of the system.

Table 2.15 lists the GPS agent qualities that make it part of the MAS. Table 2.16 lists the GPS agent requests implemented. Figure 2.23 shows the state machine for the GPS agent.

Agent: Torquerod Control Unit

The TCU agent is described in this section. Its main purpose is to control the magnetic torque rods using the control input from the control agent. The other main purpose of this agent is to monitor the health of the TCU hardware and report that to the rest of the system.

Table 2.17 lists the TCU agent qualities that make it part of the MAS. Table 2.18 lists

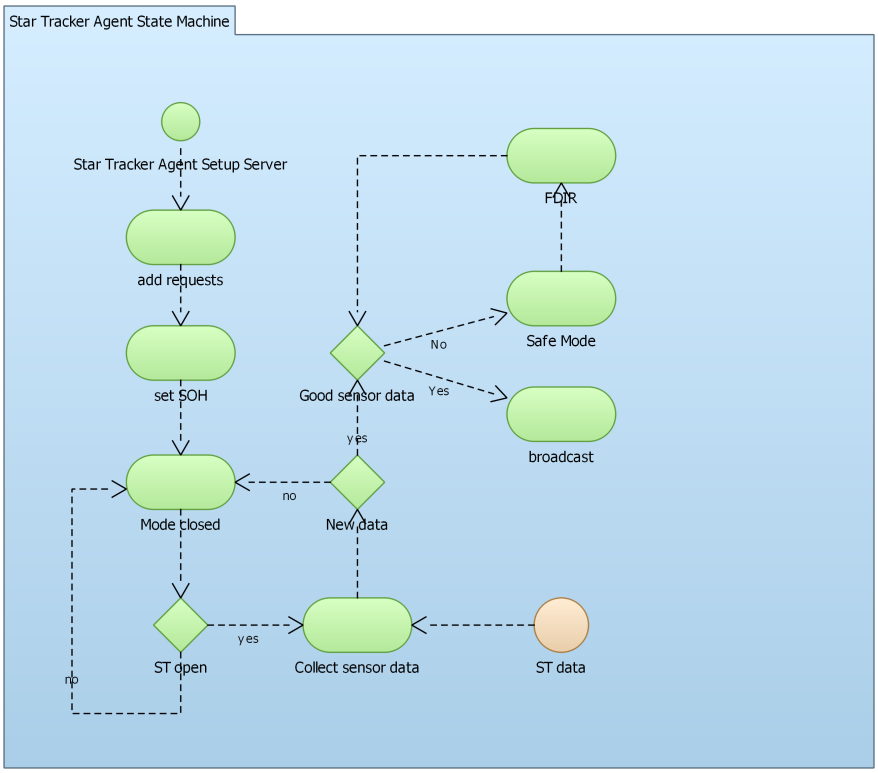

Figure 2.21: Star tracker agent state machine

the TCU agent requests implemented. Figure 2.24 shows the state machine for the TCU agent.

2.5 Results

In this section some of the results obtained from the software simulations are presented and also from the hardware-in-the-loop (HIL) tests for the HSFL satellite using the MARS control architecture. The satellite has Inertia tensor $\mathbb{I} = diag\{2.5448, 2.4444, 2.6052\}\, kgm^2$ and a mass of 55 kg. The satellite orbit is slightly elliptical with 415 km of perigee and 490 km of apogee on a near sun-synchronous orbit (polar) with inclination i of 94.7 degrees. The MTRs are capable of producing a maximum magnetic moment of 13 Am^2, measured during actuator characterization. The attitude requirements impose that for nominal operations

agent function	description
Reactive	collect IMU sensor data, forward the data to other agents (specifically agent navigation)
Social	send and receive SOH messages and/or requested data
Proactive	state machine with decision process to maintain operation modes/objectives

Table 2.13: Description of IMU agent qualities

agent request	description
set mode	set IMU operation mode (0 = closed, 1 = open, 2 = simulated).
debug	turn on/off debug messages
debug raw	turn on/off raw data from the IMU
mag	request magnetic field
omega	request angular velocity (ω)
all	request IMU state (magnetic field B, omega ω)
omega offset	add offset to omega data in deg/sec
mag offset	add offset to mag data in uT

Table 2.14: List of IMU agent requests

the attitude of the satellite is 5 deg or better for pointing accuracy with a residual angular velocity norm of 1 deg/sec or less. The attitude is represented in quaternion space and the controller agent uses the quaternion error to compute the torque rods inputs. For the numerical simulations the assumption that the mass in the satellite is properly distributed is used in such a way that the principal body axes coincide with the body frame of the satellite - this is why a diagonal inertia tensor is used.

2.5.1 Software Simulation

The control algorithm is tested within a 6DoF simulation for position and attitude dynamics implemented with COSMOS to demonstrate the ability of the attitude controller to detumble

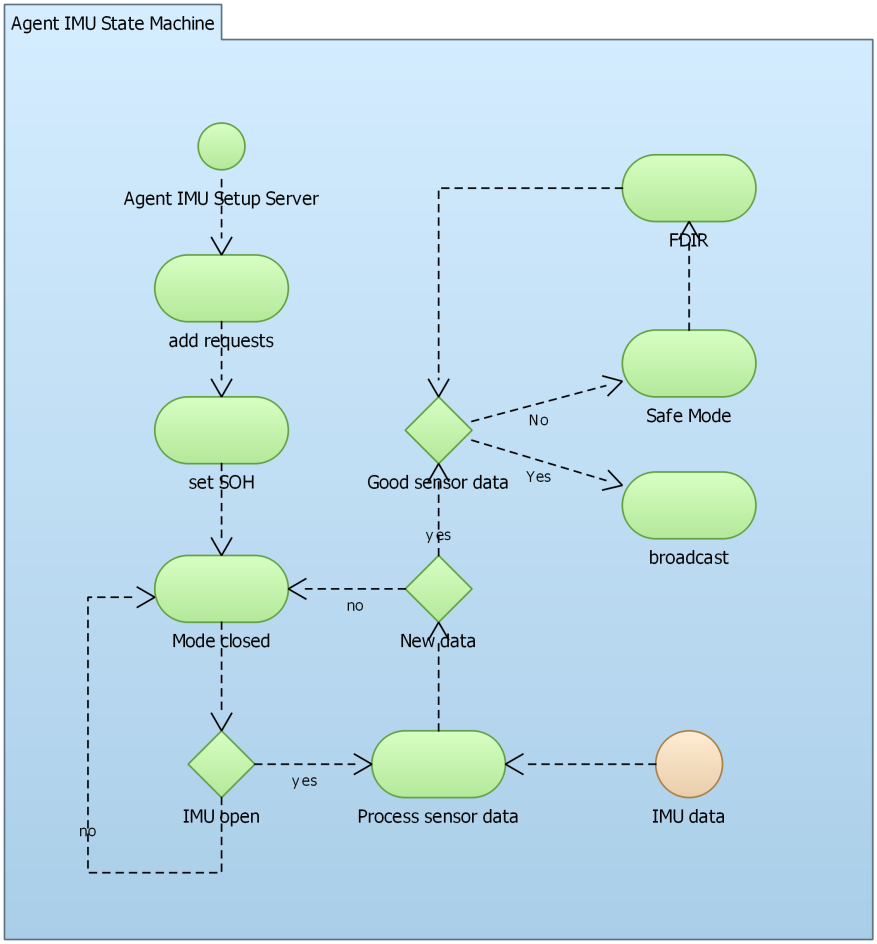

Figure 2.22: IMU agent state machine

and point the spacecraft using a realistic orbit and physical properties of the satellite. The satellite attitude and position dynamics are propagated with an Gauss Jackson method[47] that is a numerical integration method with a multi-step predictor-corrector for second-order ordinary differential equations. The maneuver starts with initial conditions $\boldsymbol{q} = [0, 0, 0, 1]$ and $\boldsymbol{\omega} = [10, -7, 5]\,^{\circ}\,\mathrm{s}^{-1}$. These initial conditions are a possible worst case scenario because the initial angular velocity is expected to be 5 deg/sec or less after separation. The detumble maneuver is actuated using the optimal control law described previously while the pointing maneuver uses the PD-control scheme. Figure 2.25 and 2.26 show the satellite detumble and attitude pointing maneuvers. The first plot on the top of Figure 2.25 represents the three

agent function	description
Reactive	collect GPS sensor data, forward the data to other agents (specifically agent navigation)
Social	send and receive SOH messages and/or requested data
Proactive	state machine with decision process to maintain operation modes/objectives

Table 2.15: Description of GPS agent qualities

agent request	description
set mode	set IMU operation mode (0 = closed, 1 = open, 2 = simulated).
debug	turn on/off debug messages (state, general, settime, gpstime, bestxyz)
debug raw	turn on/off raw data from the GPS
state	request GPS data
reset	reset the GPS

Table 2.16: List of GPS agent requests

components of the angular velocity, including the Euclidean norm of the vector $\|\boldsymbol{\omega}\|$. It is clearly seen that $\|\boldsymbol{\omega}\|$ decays steeply for the first 10% of the first orbit, which corresponds to approximately 10 minutes, and asymptotically goes to zero to complete the detumble maneuver by the end of the second orbit. The complete detumble maneuver takes approximately 2.2 orbits or approximately 200 minutes. The second plot represents the satellite attitude in quaternion space. For the first part of the orbit the attitude is changing drastically, but it slowly comes to track the LVLH attitude by the end of the maneuver and then maintains the required pointing. Note that the attitude continues to change after detumble maneuver because the satellite keeps orbiting the Earth and the LVLH changes at every instant in the orbit.

In Figure 2.26, the first plot on the top represents the components for the MTR moments. The saturation limits for the MTR are not reached in this particular case. There is a

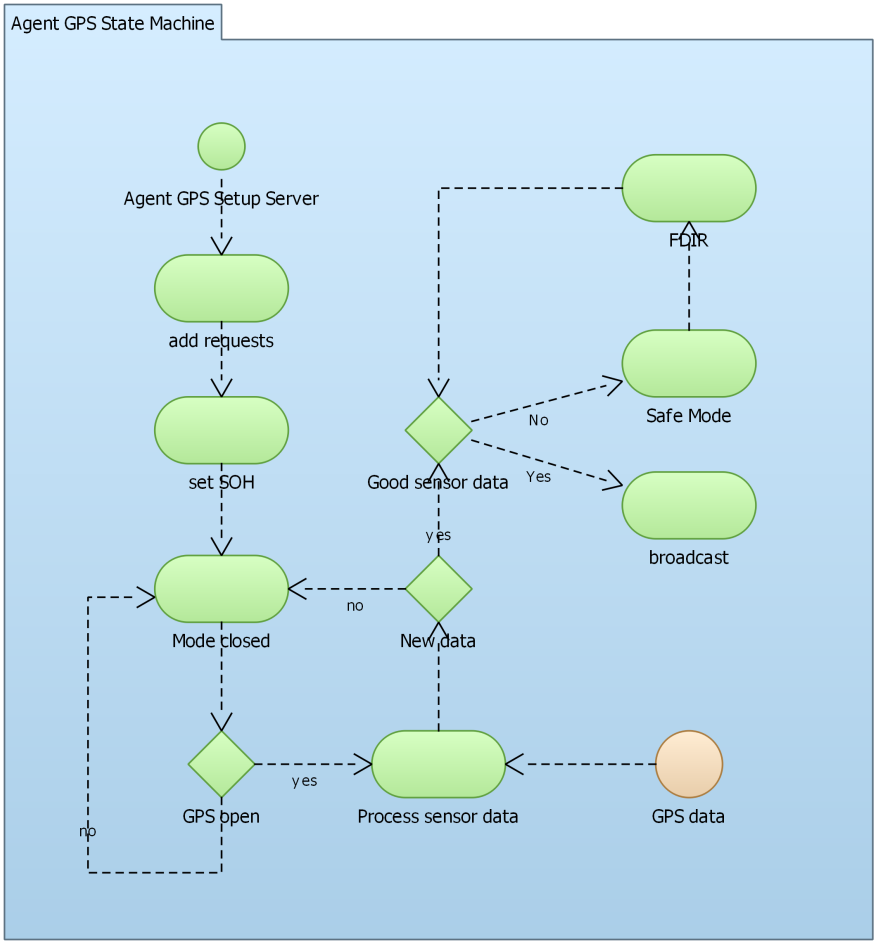

Figure 2.23: IMU agent state machine

clear distinct transition between the detumble mode and the pointing mode can be seen just before the 1.5-orbit mark. The second plot shows the total torque produced by the satellite over the same period. Correspondingly in this plot it can also be seen the transition from the detumble maneuver to the pointing maneuver just before the 1.5-orbit mark. These results show that for this particular simulation the detumbling and LVLH attitude tracking can be achieved well within a few typical orbits in the worst case scenario. This particular orbit has a period of 93.6 minutes.

Another set of software simulations was run to validate the attitude dynamics model

agent function	description
Reactive	collect control data, drive the magnetic torque rods, turn rods off per agent control request
Social	send and receive SOH messages and/or requested data
Proactive	state machine with decision process to maintain operation modes/objectives

Table 2.17: Description of TCU agent qualities

against the physical model for the satellite testbed using the air bearing. Figure 2.27 shows the angular rates for the test condition of the satellite testbed starting with 5 deg/sec around the z-axis. The detumble maneuver takes approximately 5.8 minutes which is very close to the average detumble time during the tests in the satellite testbed, see Table 2.23. The pointing maneuver also behaves in a very similar way. Compare with figures in Appendix C.1. Figure 2.28 shows the control inputs for the same maneuver. The pointing maneuver starts at approximately 500 seconds. The simulation time starts at 200 seconds to replicate the boot up time for the flight computer in the satellite testbed runs.

2.5.2 Hardware in the Loop Tests

The satellite testbed replicates the satellite hardware and its environment as close as possible to realistic conditions experienced in space. Most of the testbed hardware is the same as used in the satellite with the exception of the star tracker and TCU. But these are replicated to behave exactly as the real unit. The star tracker simulator uses a motion tracking camera to collect the attitude and transports the data using the same protocol as the flight unit. The flight computer then reads the data exactly as it would from the start tracker unit using the same data interface (RS422). This becomes a true plug and play testing capability. Equivalently the MTR were developed in house and they are commanded using the TCU

agent request	description
set mode	set operation mode (0 = closed, 1 = open, 2 = simulated).
debug	turn on/off debug messages
debug raw	turn on/off raw data for the TCU
state	set TCU state
reset	reset the TCU
ix	set Current for Torque Rod 0 (X-Axis)
iy	set Current for Torque Rod 1 (Y-Axis)
iz	set Current for Torque Rod 2 (Z-Axis)
ixyz	set Current for Torque Rod 0,1,2 (XYZ-Axis)
on	turn on TCU
off	turn off TCU
tlm	report latest agent status and TCU telemetry
momxyz	Set amps to effect requested magnetic moment in body frame

Table 2.18: List of TCU agent requests

that receives the commands exactly as the flight unit does. The flight version of the MTRs can produce twice the magnetic moment compared to the testbed MTRs. This work only shows the results for the testbed version of the MTRs because if the control is validated with these then the flight units can certainly accomplish the same controlling performance. The software is the exact same software as installed in the flight computer. This enables realistic testing of the hardware and software while mission operations scenarios can be also tested and executed before sending the commands the satellite in orbit. The main limitation at this point are the batteries that drain after 3 hours of operations. Figure 2.29 shows the satellite testbed with the Helmholtz chamber surrounding the satellite hardware.

The testbed dynamics engine is developed in COSMOS which provides the models for a simulated space environment to the testbed to allow a more realistic operation of the platform. The dynamics engine also controls the different hardware and software configurations in the satellite system simulator and allows the tuning and mixing of signals and interrupts,

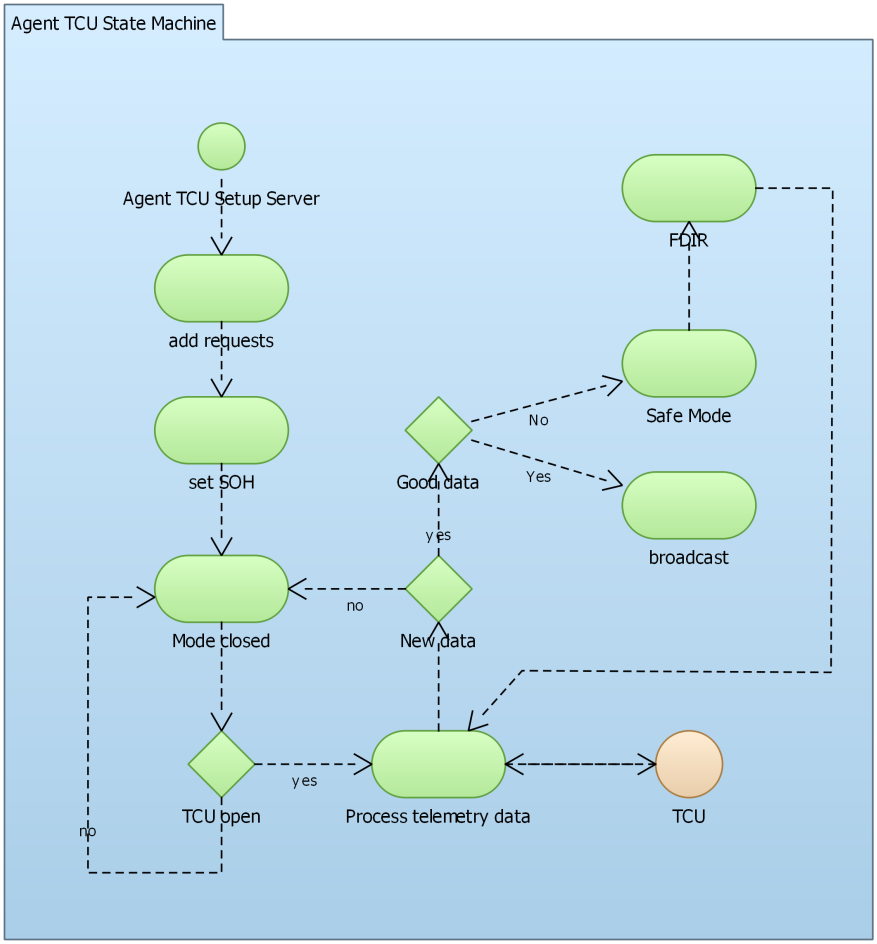

Figure 2.24: TCU agent state machine

adding noise and possible failure modes. All this is done controlled by an user interface or an automated scripting sequence. The satellite testbed platform[56] is supported by an air-bearing which is one of the most important components of the satellite testbed and has a very low disturbance torque of less than 10^{-5} N.m. This is roughly equivalent to disturbance torques experienced in space for LEO satellites. The platform is capable of rotating 360 deg. about the vertical axis and approximately $\pm$ 20 deg. in the roll and pitch axis. This allows for essentially 3DoF rotation capability as experienced in space for attitude dynamics. The center of gravity (CoG) of the platform including the ADCS system is calibrated using an integrated linear motor mass system installed on the platform. The platform also has an

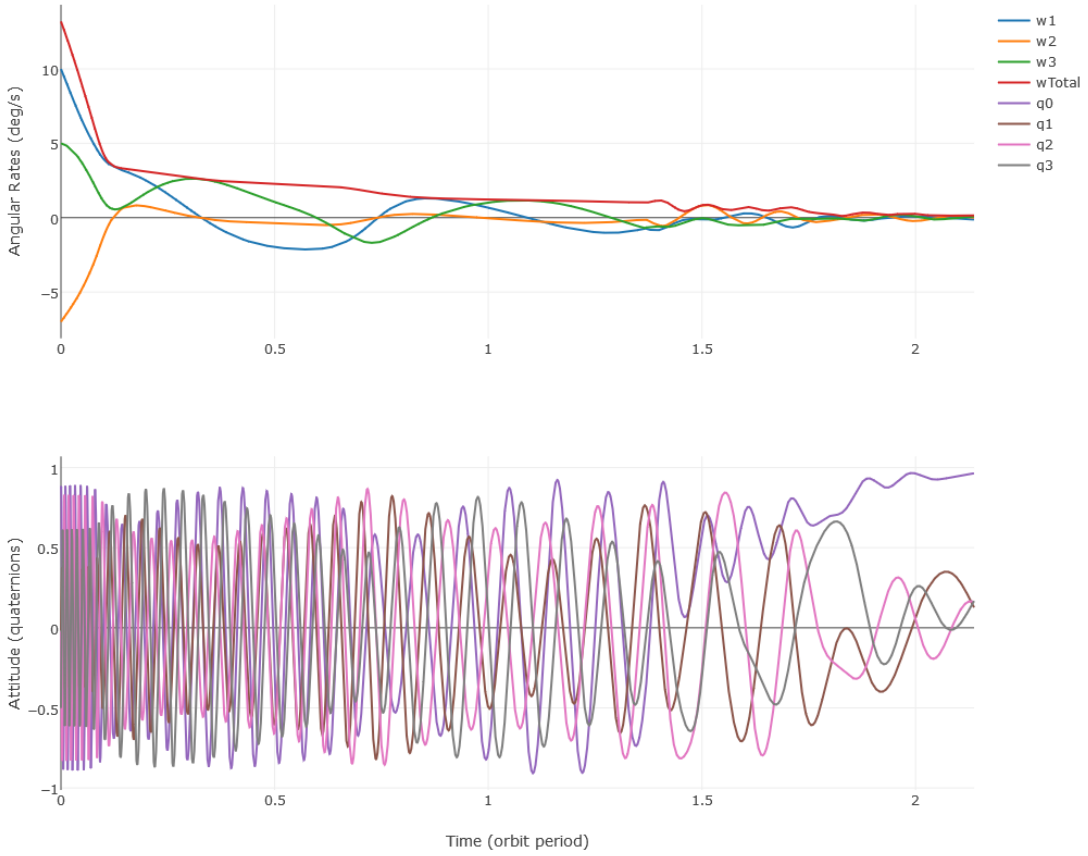

Figure 2.25: Angular rate and attitude for satellite detumble and pointing maneuver from worst case scenario

integrated 24 V DC (unregulated), 400 Wh power source for untethered operation of the system under test for long periods of time.

Monolithic vs MARS ADCS

In this section the proposed MARS based ADCS is compared with a traditional monolithic ADCS. Both implementations use the COSMOS core libraries for the agent functionality. The estimation and control algorithms are implemented within each agent and deployed

Figure 2.26: Magnetic torque rods moment and total torque for satellite detumble and pointing maneuver from worst case scenario

into the flight computer. The monolithic and MARS implementations use C/C++ and the same cross toolchain for embedded ARM computers (gcc-linaro-4.8-arm-linux-gnueabihf). The core functions for the monolithic software are implemented in 2468 lines of code while the MARS software is implemented using 4931 lines of code, which is essentially two times more[1]. The main reason for having more lines of code (LOC) on the MARS is because the agents add more functionality such as the "requests", the "state of health" messages, and also have more sophisticated state machines, all implemented in different threads. This naturally

[1]The estimate was done using the program "Count Lines of Code" available at https://github.com/AlDanial/cloc. Retrieved February 2016

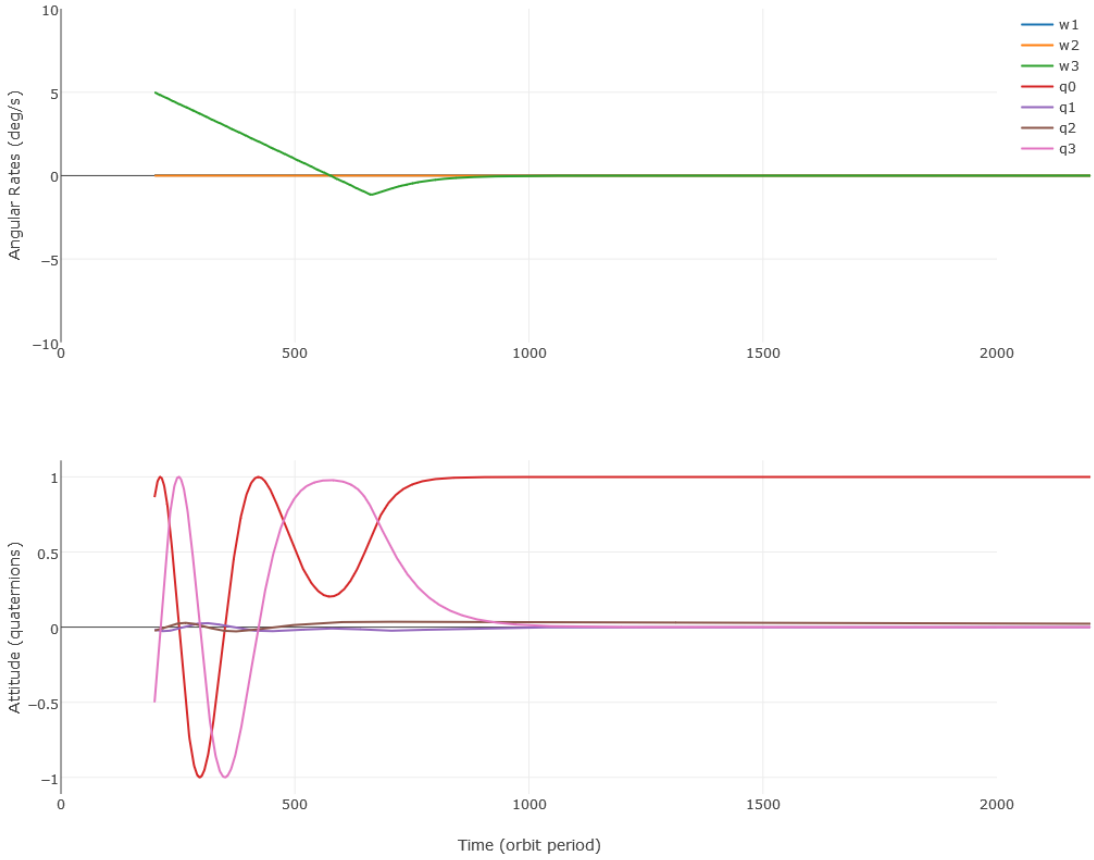

Figure 2.27: Angular rate and attitude for satellite testbed detumble and pointing maneuver simulation

increases the LOC for the MARS software. Table 2.19 shows the LOC breakdown between the different implementations. These lines of code are based on the COSMOS core libraries which currently have more than 100,000 lines of code. The complete set of lines of code for the spacecraft, including the ADCS, is approximately 50,000. Both ADCS implementations have the same sensor data, algorithmic estimation and control structure. The main difference lies in the fact that on the monolithic implementation the instructions are run in sequence, as it would typically be implemented in an embedded micro-controller, while on the MARS the instructions are fully distributed and operate in different threads and state machines in

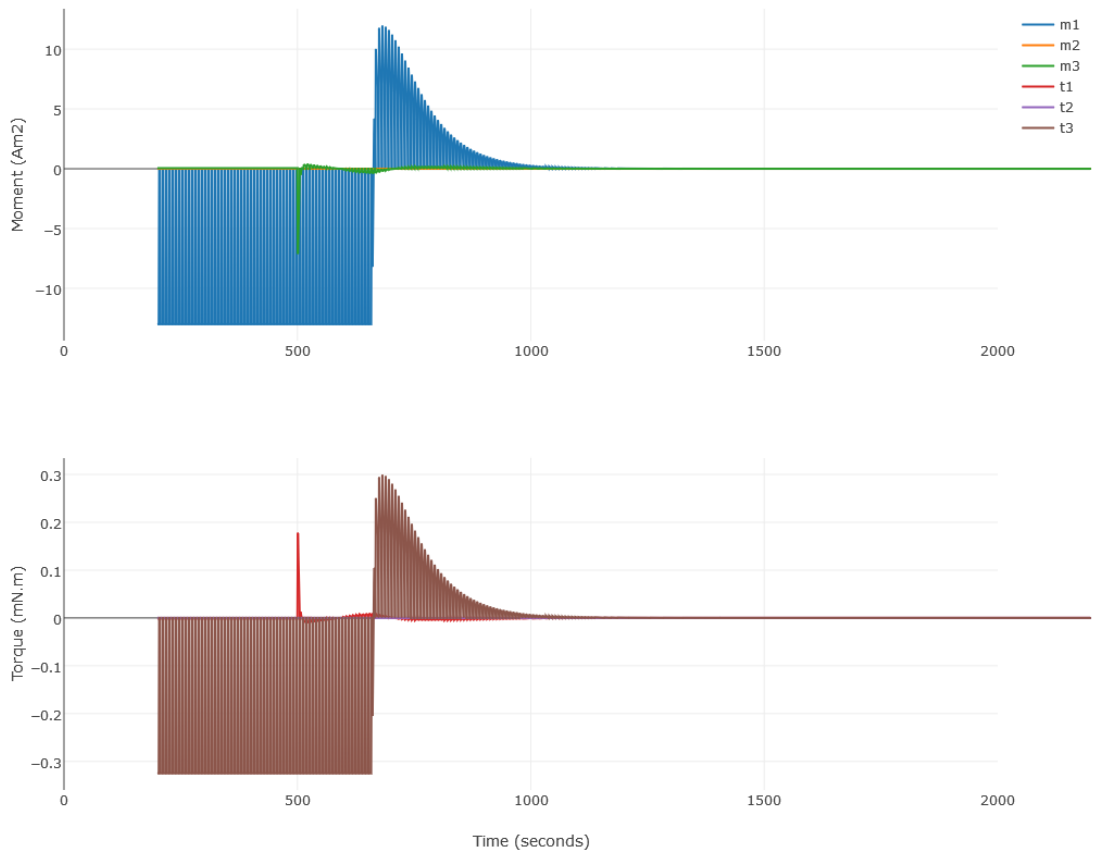

Figure 2.28: Magnetic torque rods moment and torque for satellite testbed detumble and pointing maneuver simulation

every agent. The performance parameters that will be compared are:

- **CPU percentage for each process**. This metric is the CPU time consumed by a particular running process over a specific period of time, for this particular work it is averaged against two seconds.

- **average CPU load of all processes running**. The CPU load is a measure of computational throttle of the system and the one represented here is the load average of the last minute. When the CPU load is less than one the CPU is capable of performing

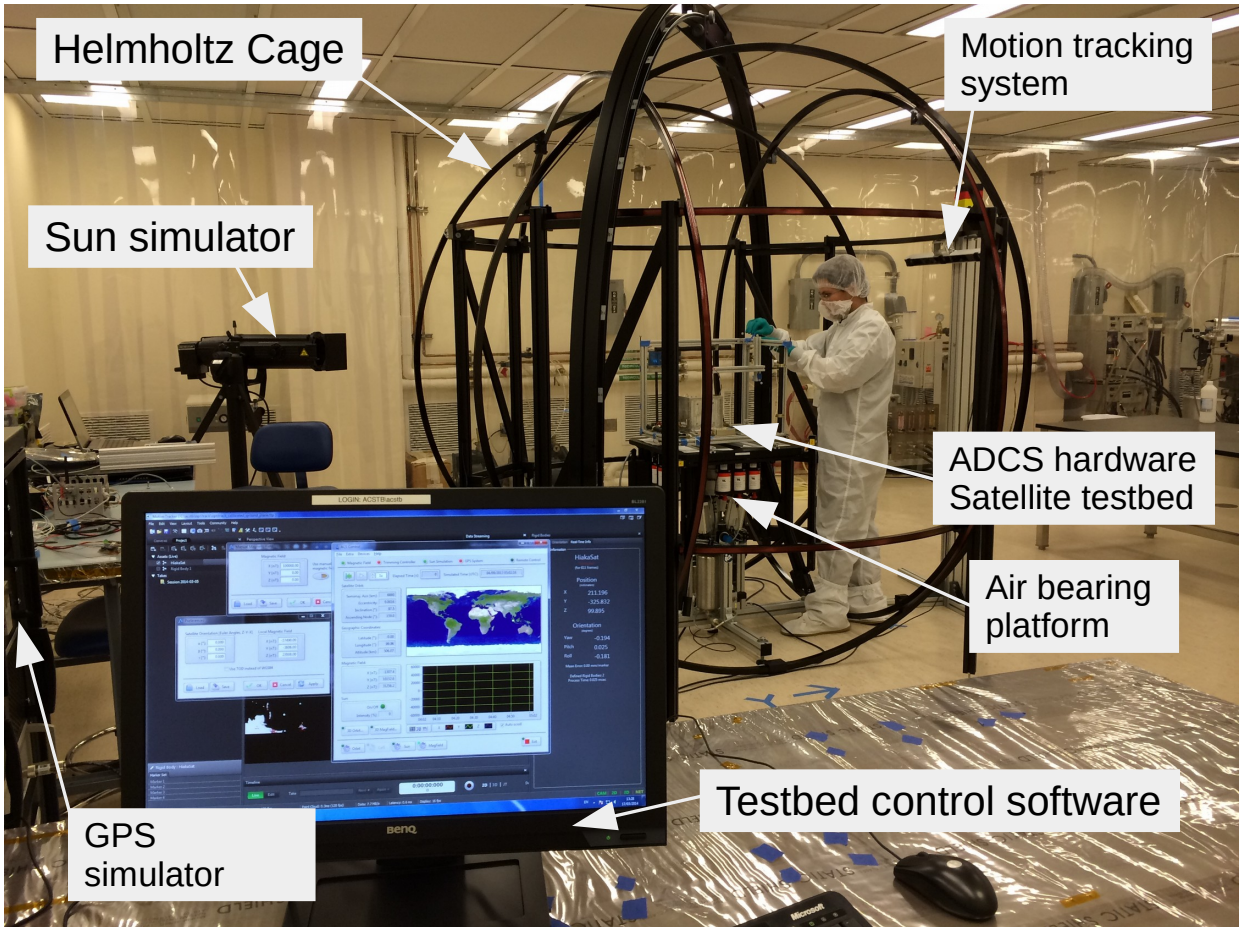

Figure 2.29: HSFL Satellite ADCS testbed

code	LOC (mono)	LOC (MARS)
libraries ADCS	1166	1166
monolithic ADCS	1302	
agent control		611
agent navigation		804
agent imu		453
agent st		514
agent gps		509
agent tcu		874
total	2468	4931

Table 2.19: Comparison of lines of code for monolithic implementation and MARS

all the scheduled tasks. When the load is above one it means the CPU has to backlog certain tasks for when it has the time to process it. This situation significantly impacts the real-time control process because the of the unpredictable lag in the system.

- **attitude**. The satellite attitude is represented in the quaternion space, when close to identity it is pointing in the set target attitude $q_{target} = [0, 0, 0, 1]$.

- **attitude rate (ω)**. The attitude rate or angular velocity is an important measure for stability of the system. When the satellite is stable the angular velocity is approximately null.

Figure 2.30 shows the data collected during one typical set of tests for the monolithic implementation (left column) and the MARS ADCS (right column). These particular tests run for 700 sec. At the beginning of the run the computer is booting for approximately 60 seconds. The final sequence of the boot process runs specific scripts to load the first set of agents and other programs. Only 120 seconds after the boot process is complete is that the ADCS agents are initiated, the main reason is to de-conflict the ADCS agents and other processes that may still be initiated. These tests show the control performance during the attitude maneuvers to transition the satellite from a "blind state" to a stable attitude, as is

happens on a typical satellite deployment.

The first plot on the top shows the CPU percentage used for each of the agents during the test duration. The plot shows that the ADCS agents only start at approximately 180 seconds as expected (60 seconds for the boot process plus 120 seconds of wait). This is why there is no data before 180 seconds since the agents themselves report the data that is shown. The ADCS process on the monolithic implementation has a typical percent usage of 28.97 %. On the MARS implementation every agent operates at less than 10% of CPU and summing all of their combined CPU percentage the total is 30.09 % which is very close to the same amount of % CPU consumed by the monolithic implementation. The difference is 1.12 %. A priori this is a surprising result because these are completely different software implementations and a higher total % CPU use for the MARS implementation could have been expected. On the other hand this result is in itself consistent and adds a level of confidence because both approaches use exactly the same algorithms with the exception of the agent server, communication overload and state machines for the MARS implementation. This proves that both implementations not only consume the same amount of CPU, and this result can be possibly extrapolated if more agents are implemented, but more importantly that the MARS does not incur in any CPU overload because of the "agent" sections. On this particular implementation of the MARS every agent runs on the same processor core and there seems that there's no advantage compared with the monolithic implementation. But if the ADCS where to be distributed across multiple CPUs the monolithic implementation would not be possible while the MARS implementation would not require any change. The system would work "off-the-shelf". The ability to distribute the ADCS functions across multiple CPUs is a major advantage of MARS based ADCS.

The second plot shows the CPU load used by all the processes running on the embedded computer. During the initial 200 seconds of the run the processor is booting and that is the reason for the transient CPU load seen in the plot. For these two particular cases the CPU load average for the whole duration of the test is represented by the dashed line. The average CPU load on the monolithic implementation is 0.56 compared to 0.43 on the MARS implementation. This result is slightly surprising because it would be expected than more processes running (agents) would incur a higher load on the CPU. This result is analyzed in more detail later in the results section with more tests. This indicates that a MARS is computationally efficient. The third plot shows the attitude quaternion. The monolithic implementation takes approximately 300 seconds to acquire the target set attitude and the MARS takes approximately 200 seconds. The time difference is not particularly relevant because both tests start at slightly different angular velocities but more importantly is that the transient period is in the same order of magnitude. The last plot shows the angular velocity. The test on the monolithic implementation starts with an angular rate of approximately 7 deg/sec while the MARS implementation starts at approximately 5 deg/sec.

To gain further insight into the different performance parameters, four separate tests were executed for both the monolithic implementation and the MARS. The following tables summarize the results for each test and the corresponding plots can be seen on the Appendix C.1 and C.2.

Table 2.20 lists the average CPU % and standard deviation of the monolithic ADCS tests. The average values are taken considering the whole duration of the tests. The total average of the ADCS process consumes 28.99 % $\pm$ 0.12 of CPU. Table 2.21 lists the average CPU % and standard deviation taken by each agent process for the MARS. Every process consumes less than 10 % of CPU and all except the TCU agent have a very small variability

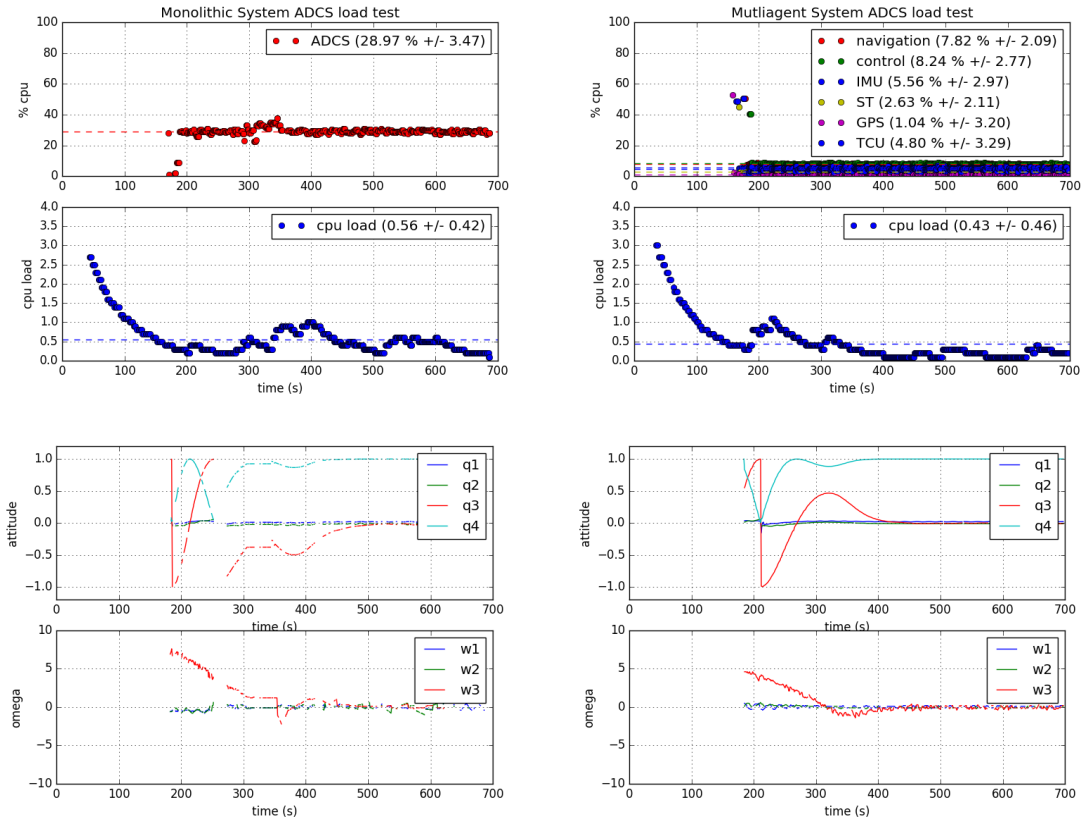

Figure 2.30: Typical performance plots for detumble and point maneuver for the monolithic architecture (left column) with the MARS (right column).

run	ADCS
1	28.93 ± 2.07
2	29.19 ± 2.3
3	28.88 ± 1.86
4	28.97 ± 1.82
mean	**28.99 ± 0.12**

Table 2.20: CPU % mean use for monolithic ADCS runs

agent	run 1	run 2	run 3	run 4	mean
navigation	7.67 ± 0.41	7.68 ± 0.44	7.67 ± 0.44	7.64 ± 0.42	**7.67** ± 0.02
control	7.94 ± 0.47	7.93 ± 0.46	7.69 ± 0.55	7.9 ± 0.43	**7.87** ± 0.10
imu	5.16 ± 0.41	5.17 ± 0.44	5.14 ± 0.44	5.17 ± 0.4	**5.16** ± 0.01
st	2.45 ± 0.7	2.47 ± 0.73	2.48 ± 0.7	2.47 ± 0.65	**2.47** ± 0.01
gps	0.84 ± 0.69	0.85 ± 0.7	0.84 ± 0.63	0.84 ± 0.7	**0.84** ± 0.00
tcu	4.59 ± 0.91	5.1 ± 1.58	6.73 ± 2.08	4.59 ± 0.88	**5.25** ± 0.88
total CPU %	28.65	29.20	30.55	28.61	**29.25** ± 0.79

Table 2.21: CPU % mean use for MARS tests

on the CPU % use. This is due to the fact that the TCU agent has a transient state every few seconds to turn off the actuators for the magnetometer reading, therefore the running process changes from "idle" to more active very frequently. The average total CPU % for the MARS tests is 29.25 % ± 0.79. This result is very close to the monolithic tests, which increases the confidence on the fact that the algorithms running on both tests are the same, but also indicates that there is no significant overhead regarding the networking functions running in the agents. The CPU % difference between the monolithic and MARS implementation for the ADCS is 0.9 % which is negligible in practice.

Table 2.22 shows the results for the CPU load for the monolithic tests on the let side columns. The average CPU load of all the tests is 0.35 ± 0.04. For this case there is very little variability on the load incurred by the ADCS process. The right side columns show the results for the CPU load during the MARS tests. The average CPU load of all the tests is

run	CPU load (mono)	run time	run	CPU load (MARS)	run time
1	0.35 ± 0.17	1000	1	0.26 ± 0.13	1000
2	0.31 ± 0.17	1000	2	0.30 ± 0.18	1000
3	0.32 ± 0.21	1000	3	0.33 ± 0.25	1000
4	0.41 ± 0.17	1000	4	0.25 ± 0.14	1000
mean	**0.35 ± 0.04**		mean	**0.29 ± 0.03**	

Table 2.22: CPU load for monolithic tests (left). CPU load for MARS tests (right).

0.29 ± 0.03. It is interesting to note that the load average for the MARS tests is lower than the one obtained for the monolithic runs. The CPU load difference is 0.06, the MARS implementation requires 17.1 % less CPU load compared to the monolithic implementation. This may be a surprising fact but it confirms that, at least for this particular implementation, distributing the computational functions using a MARS does not incur in more overhead for the CPU, instead it is actually more efficient according to the executed runs. One of the direct benefits of the MARS implementation is that more CPU time is left, even if the difference is slight, for other tasks compared to the monolithic implementation.

Table 2.23 contains the results for the detumble maneuvers during the monolithic tests (left columns) and the MARS test (right columns). The tables list the initial angular rate (ω_0) in degrees per second and the detumble maneuver time in minutes. The initial angular rate is set by using a control command to accelerate the satellite testbed platform. This is done using a constant torque command. Once a certain angular rate is reached the computer is automatically rebooted to simulate the release sequence from the launch vehicle. The average detumble maneuver time for the monolithic tests is 4.2 ± 0.4 min. while the MARS average is 5.5 ± 0.3 sec. This difference is due to the fact that the monolithic implementation has a control duty cycle that is faster than the MARS implementation. The most important aspect to draw from these results is that the maneuver times are within the same order of magnitude and approximately 5% of the orbit period, which is sufficient to initiate nominal

run	ω_0 (deg/sec)	maneuver (min.)	run	ω_0 (deg/sec)	maneuver (min.)
1	5.1	4.4	1	5.0	5.3
2	4.9	4.7	2	5.9	5.8
3	4.9	3.7	3	4.9	5.6
4	5.1	3.8	4	5.1	5.2
mean	**5.0 ± 0.1**	**4.2 ± 0.4**	mean	**5.0 ± 0.1**	**5.5 ± 0.3**

Table 2.23: Detumble performance for monolithic tests (left) and for MARS tests (right).

run	mono (°)	MARS (°)
1	1.34 ± 0.61	1.37 ± 0.29
2	2.20 ± 0.99	1.54 ± 0.53
3	2.44 ± 0.98	1.91 ± 0.49
4	2.42 ± 0.96	1.85 ± 0.42
mean	**2.60 ± 1.03**	**1.67 ± 0.22**

Table 2.24: Attitude pointing performance for monolithic tests (left) and for MARS tests (right).

operations.

Table 2.24 contains the results for the pointing performance during the monolithic tests (left columns) and the MARS test (right columns). The table lists the average pointing error in degrees for both cases. The average pointing error for the monolithic tests is 2.6° ± 1.03 while the MARS average is 1.67° ± 0.22 sec. The MARS implementation is 20.6 % more accurate. The most important aspect to draw from this results is that the pointing performance for both implementations is within the the set requirements ($\leq$ 5 deg.). Also the MARS is very capable to maintain the pointing precision.

Failure Detection Isolation and Recovery

Figure 2.31 represent the recovery after a failure for the *star tracker agent* software. An error was purposely injected on the star tracker agent (such as a radiation hit on the software).

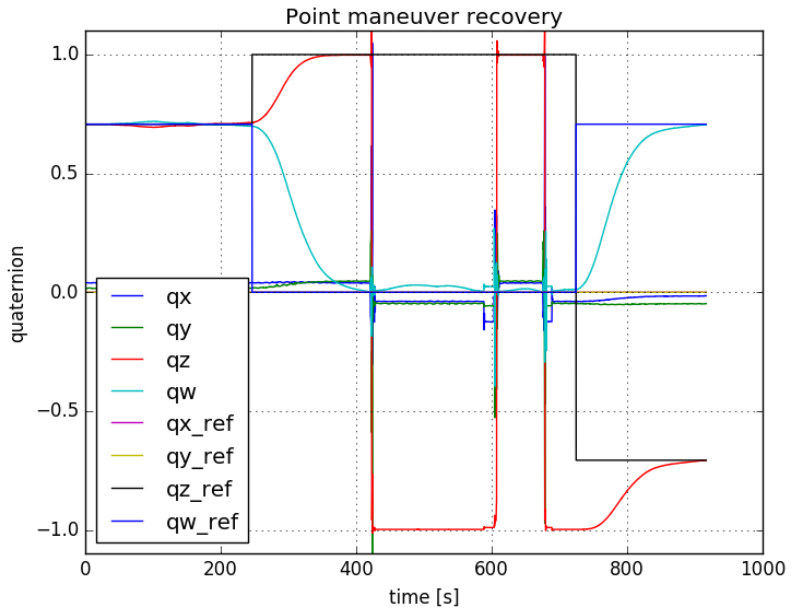

Figure 2.31: Attitude pointing maneuver recovery using a MARS architecture

The anomaly is detected and the star tracker agent is shut down and another copy of the star tracker agent is initialized automatically in another memory space, maximizing therefore the possibility for recovery. The system recovers from the anomaly and the pointing maneuver continues as expected. The recovery process is represented in the flow diagram in Figure 2.32. This recovery would be harder to achieve with a monolithic system because the anomaly would reside in the centralized control program and this would have to be re-initiated. The event would bring the control effort to a halt and the satellite would be drifting for as long as the recovery of the monolithic system would take. This shows one more advantage of a MARS, specifically in terms of robustness of software failures.

2.6 Conclusion

In this work a MAS software is developed for a satellite ADCS and compared to a typical monolithic implementation of the ADCS. It is found that the MAS performs equivalently, or

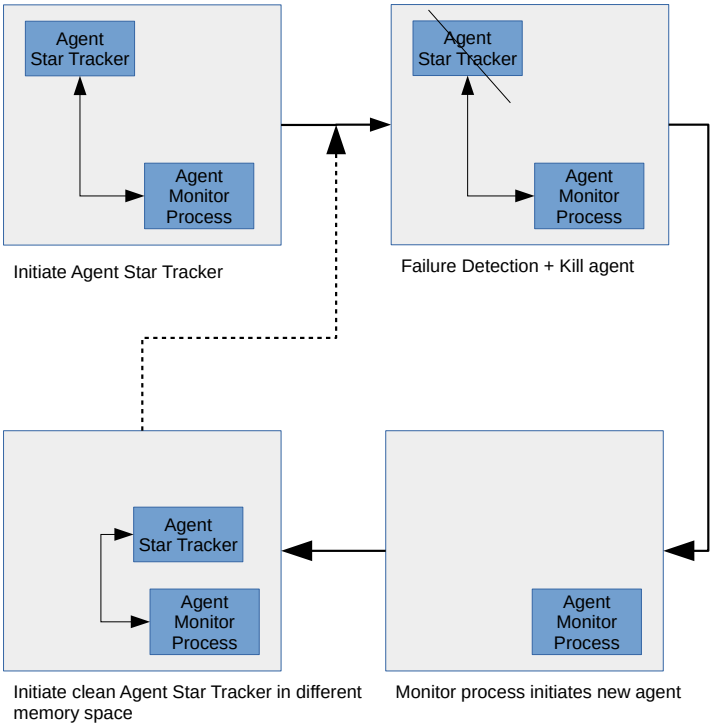

Figure 2.32: Flow diagram with agent star tracker recovery example

better, than the monolithic implementation in the tested scenarios based on actual operations of the HSFL HiakaSat. The MAS is implemented on a satellite onboard computer and tested on a satellite testbed at the HSFL. The satellite control authority is under-actuated and consists of only three MTR that are orthogonally placed in the satellite bus and the MAS software consists of six independent agents that monitor and control the ADCS system.

Numerical results are presented which show that the control system can successfully keep the satellite within the desired requirement for stability, less than 1 deg/sec of angular velocity and pointing precision of less than 5 deg. The simulation is implemented using 6DoF system dynamics using the COSMOS framework and it shows that the control and estimation logic can meet the requirements starting from a worst case scenario with initial angular velocity much higher than specified by the launch vehicle. The detumble maneuver is executed in 30% of the orbit and a LVLH attitude tracking mode is started thereafter.

A HIL test is done using the HSFL satellite testbed using satellite hardware and software to replicate the space conditions as closely as possible. Six software agents are implemented where each takes a logical separate function within the ADCS. The higher level agents are the control and navigation agents which process the sensor data and implement the higher level control autonomy function. The other agents deal mostly with the sensors and actuators and respective failures. Each of these agents have a failure detection mechanism to inform the remainder of the MAS for taking appropriate action. Finally the TCU agent manages the control commands to the MTR. The first important result is that the MAS is computationally more efficient than the monolithic solution. Also the results show that the distributed ADCS based of the MAS can operate with equivalent control performance to a monolithic ADCS. A simple test case of sensor failure and recovery was demonstrated by injecting a failure mode into the star tracker and let the MAS determine the failure, shutdown the agent

in cause and restart the same agent as a tentative to recover without compromising the rest of the ADCS. This demonstrates that a MAS can be quite robust while a monolithic system will impair the whole ADCS to recover and restart. These important results (computational efficiency, control performance, failure detection and recovery) are a strong indication that the use of a MAS for ADCS is a sound realization.

This work also introduced a "pseudo-ECI" orbital propagator for initial orbit determination when the satellite is separated and does not have any sensory information such as the GPS. This improves the autonomy of the satellite by enabling it to initiate the nominal operations and track LVLH in case there is no GPS lock or ground contact.

This work focused on developing a baseline for a MAS but the advantages of such a system were not explored in detail such as different levels of resilience to failures or higher level automation and/or intelligence. One interesting venue of research for such a MAS-based ADCS is the search for adaptive methods that improve the system performance over time. Also for future work the study of onboard scalability for MAS is important, specially if the ADCS has a large number of sensors and/or complex logic or must be in coordination with other spacecraft in a multi-satellite mission.

CHAPTER 3
MULTI-AGENT ROBOTIC SYSTEM FOR DOCKING MULTIPLE SATELLITES

3.1 Introduction

The first docking of two spacecrafts was performed in 1966 during the Gemini program in preparation for the Apollo lunar missions[57], and was human controlled. Gemini 8 was commanded by Neil Armstrong while he docked the spacecraft with the unmanned Agena Target Vehicle. The first fully automated docking between two spacecrafts was just the year after in 1967 by two Soviet spacecrafts: Kosmos 186 and Kosmos 188. Since then there have been other successful examples of docking missions, but they mostly fall in the category of large spacecraft and human space exploration such as ESA's Automated Transfer Vehicle[58]. Recently, interest has increased on the prospects of autonomously docking small satellites for robotic space exploration[59]. With the ongoing effort to reduce the cost of small satellites and increase their capabilities, it is expected that multiple satellite missions will become more frequent and in particular new missions that require the ARD of multiple small satellites. This work addresses the ARD problem for multiple satellites by developing a new GNC architecture using a multi-agent robotic system for efficient communication and coordination between the various satellites.

3.1.1 Motivation

Multiple small satellites can be used as building blocks to assemble large space telescopes, large solar arrays, satellite servicing stations, etc. The goal for this work is to develop a MARS framework and a GNC architecture to enable ARD of multiple small satellites. This is the foundational work to enable realistic simulation scenarios for future mission designs

and analysis using multiple satellites. In this work a simulation is run for docking four CubeSats as conceptually represented in Figure 3.1. The trajectory represented in the figure is the virtual-target orbit for the center of mass of the combined satellites. Each satellite is radially separated from the launcher and then are autonomously docked around a virtual reference orbit. The method however can be applicable to any size of spacecraft. Focus is given to small satellites because of the rise of upcoming docking missions using small satellites but the methods developed can be used in any kind of spacecraft.

3.1.2 Related Work

Until 2015 there had not yet been a space mission to demonstrate the feasibility of docking small satellites other than the experimental Synchronized Position Hold Engage and Reorient Experimental Satellite (SPHERES) project inside the International Space Station (ISS). This is expected to change in 2016 with the NASA Cubesat Proximity Operations Demonstration (CPOD) mission. Refer to Table 3.1 to see the list of the docking missions previously launched and the new missions planned. Every docking mission that has previously flown was using very large satellites where at least one has a typical mass of several thousands of kilograms. A small satellite is considered to have a dry mass of 180 kg or less[60]. This one of the main reasons why the CPOD mission is so unique, because it will demonstrate ARD for small satellites for the first time in space history. The SPHERES project is the closest to such a technological experiment for testing formation flying and ARD technologies but it is an experimental platform of three small satellites onboard the ISS[61].

The first mission to test ARD for small satellites that will be flying outside of a controlled environment such as the ISS is the CPOD[62] which is a technology demonstration for proximity and ARD mission using two CubeSats. This is a proof of concept originally

Mission	Year	Chaser Sat	Mass (kg)	Target Sat	Mass (kg)	Status	Country
Cosmos	1967	Cosmos-186	6 530	Cosmos-188	6 530	Success	Russia
ETS-VII	1998	Hikoboshi	2 500	Orihime	400	Success	Japan
Orbital Express	2007	ASTRO	1 089	NEXTSat	227	Success	USA
ATV	2008	ATV	20 750	ISS	419 455	Success	ESA
CPOD	2016	CPOD-A	5	CPOD-B	5	Pending	USA

Table 3.1: List of autonomous docking missions

expected to launch in 2015 but it is now manifested to launch in 2016. Each of the previously described missions use only two satellites but for the future other mission concepts are being proposed to dock multiple satellites. The AAReST is one of the mission concepts using multiple CubeSats to build a telescope in space with a preliminary phase launching two CubeSats in 2016[4] but is expected to be able to perform ARD for multiple satellites in the near future. The AAReST mission is a partnership between Surrey Space Center and Caltech/JPL. One other notable related concept of modular and multiple spacecraft is the "satlet" that is being developed by Defense Advanced Research Projects Agency (DARPA) with the Phoenix project[63] in partnership with NovaWurks. eXCITe is the first "satlet" experiment also to be launched in 2016. This initial experiment does not carry "satlets" for ARD but it is envisioned that in the future "satlets" will be capable of performing docking maneuvers.

Other works have looked into the docking problem for multiple satellites. Kobilarov[64] expands on a maneuver mechanism using a cross-entropy motion planning method that searches for the optimal trajectories. Phillips[65] develops a 6DoF rendezvous and docking method with a gradient descent approach using impulsive control. The approach finds a minimum cost path while avoiding obstacles. Boyarko[66] develops an optimal method for a rendezvous trajectory of minimum-time and energy with base on the Pontryagin minimum principle. McCamish[67] also develops an autonomous distributed control algorithm

for docking maneuver based on the Linear Quadratic Regulator and a potential function method for collision avoidance. These works are based on numerical methods for trajectory planning and thus the solutions for 6DoF are not easily computable in real time when using embedded processors. In this work a different guidance method is proposed that is not based on numerical optimization approaches but is based on a newly developed 6DoF guidance scheme as an explicit guidance law. The guidance solution can be computed in real-time and is implementable on a embedded computer as part of the 6DoF GNC algorithms which are implemented in a MARS for realistic experimentation of the system.

The MARS is developed using COSMOS is intended to be a unified framework for flight software and mission operations[38, 39]. COSMOS is being designed to support the control and operations of multiple satellites and as a result of this work a new GNC module is developed. One of the goals of the COSMOS project is to be easily adaptable to new architectures and easily scalable. Such is the case of a multi satellite mission. The development of the COSMOS framework to operate and control a large number of satellites also enables simulation scenarios to be replicated in accelerated or real time for design studies, operations training and research.

3.1.3 Contributions

The main contributions of this work are:

1. the development of a new 6DoF guidance method for full state guidance, position and attitude, by extending the 3DoF E-guidance method.

2. the development of a new MARS for this particular 6DoF GNC architecture that can realistically implement the algorithms to autonomously monitor and control various

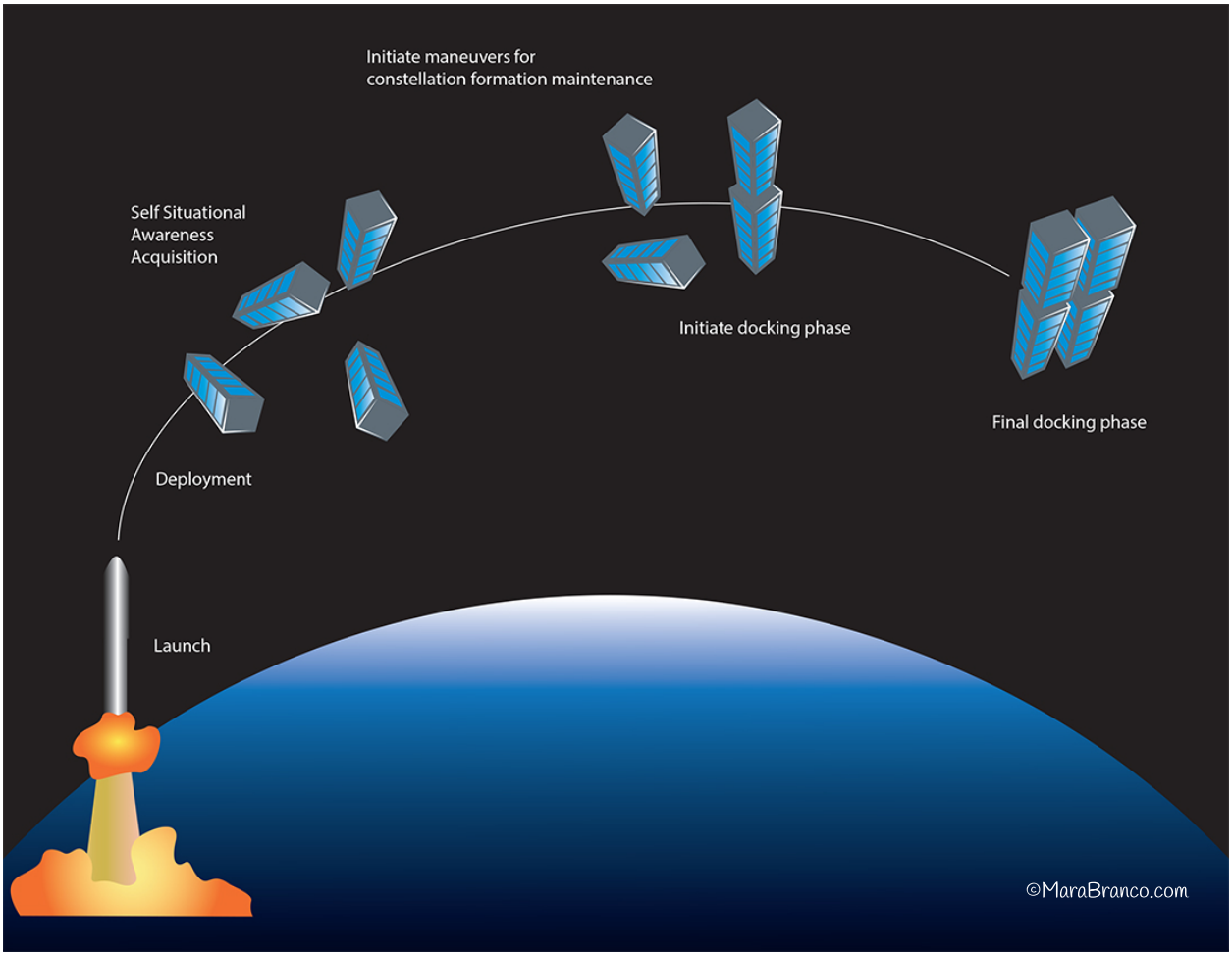

Figure 3.1: Conceptual mission docking four CubeSats.

satellites for ARD.

3. demonstrate that the 6DoF GNC architecture and the MARS are a feasible solution for ARD by simulating the rendezvous and docking of four CubeSats after a typical launch vehicle separation.

3.1.4 Outline

The work is organized as follows. Section 3.2 formulates the docking problem in terms of the system dynamics for satellites in LEO. Section 3.3 explains the MAS based on the COSMOS

software to enable a multiple satellite mission. The GNC architecture is structured in section 3.4 and the derivation of the new 6DoF guidance method is presented. Section 3.5 describes some of the software tools developed to visualize the ARD process. Finally on Sec. 3.6 a simulation scenario is demonstrated for ARD of four CubeSats.

3.2 Docking Problem Formulation

A simplified model for docking two LEO spacecrafts, the interceptor-satellite and target-satellite, is considered. The CubeSats compute their maneuvers by knowing the desired target position and by forcing a soft approach so that the velocity is zero at the desired target. The interceptor-satellite is in the vicinity of the target-spacecraft as it would be the case after a successful orbital rendezvous. A GPS sensor is used to provide the position and velocity information of the interceptor and target spacecraft. It is also assumed that a perfect knowledge of the satellite attitude is known. Every spacecraft can broadcast its current position in real time so that every other spacecraft can compute the probability of collision and recompute its trajectory as necessary. Also each spacecraft is modeled as a fully actuated rigid body where the state vector is the combination of position $\boldsymbol{r} = [x, y, z] \in \mathbb{R}^3$, velocity $\dot{\boldsymbol{r}} = [\dot{x}, \dot{y}, \dot{z}] \in \mathbb{R}^3$, attitude quaternion $\boldsymbol{q} = \left[q_v^T, q_w\right]^T$ with $q_w \in \mathbb{R}, q_v \in \mathbb{R}^3$ and $\|q\|_2 = 1$ for unit quaternion, and the body fixed angular velocity $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z] \in \mathbb{R}^3$. The satellite has the inertia tensor made of the principal moments of inertia $\mathbb{I} = diag(I_x, I_y, I_z)$.

The following assumptions are applied to the solution of this rendezvous and docking problem. The trajectory control is 6-dimensional, the relative motion is modeled using Hill's equations, the GPS sensor is used for relative position information, the attitude for each spacecraft is assumed to be fully known and the new 6DoF E-guidance method is used for

trajectory and attitude planning. The complete state vector is $[\boldsymbol{r}, \dot{\boldsymbol{r}}, \boldsymbol{q}, \boldsymbol{\omega}]$. The actuation of the spacecraft has generic control inputs $\boldsymbol{u} \in U \subset \mathbb{R}^6$ where for this work U is not yet bound, $f \in \mathbb{R}^3$ is the generic translational force and $\tau \in \mathbb{R}^3$ is the generic torque acting on the satellite body.

Relative Formation dynamics is used to formulate the motion of a target satellite in Low Earth Orbit (LEO). An orbital frame centered in the satellite that is pointing to the Earth (Local-Vertical Local-Horizontal) is used. The radial direction pointing toward the Earth determines the $\vec{x}$-axis, the $\vec{z}$-axis points towards the direction of the angular momentum vector and the $\vec{y}$-axis is determined by the cross product of $\vec{z} \times \vec{x}$. The most important forces and torques experienced in LEO come from the relative dynamics and the gravity gradient. The Clohessy-Wiltshire equations[57], also known as the Hill-Clohessy-Wiltshire (HCW) equations, describe the relative motion dynamics of two satellites in near circular orbits. The Hill's equations for the target frame described above take the form

$$\begin{aligned} \ddot{x} - 2\omega_o \dot{y} - 3\omega_o^2 x &= u_x \\ \ddot{y} + 2\omega_o \dot{x} &= u_y \\ \ddot{z} - \omega_o^2 z &= u_z. \end{aligned} \tag{3.1}$$

The variables x, y, z form the relative coordinate position of the interceptor satellite with respect to the target satellite, and u_x, u_y, u_z are the translational control inputs, $\omega_o^2 = \left(\frac{\mu_{Earth}}{r_{sat}^3}\right) > 0$ is the orbital angular rate constant.

For LEO orbits the relative motion is affected by the Earth's oblateness factor (J2) and atmospheric drag, but during the short time window of a docking maneuver these effects are negligible. The analytical solution to Hill's equations is known for circular orbits without

these effects [57] as

$$
\begin{aligned}
x(t) =& \left(\frac{2\dot{y_0}}{\omega_o} - 3x_0\right) cos(\omega_o t) + \frac{\dot{x_0}}{\omega_o} sin(\omega_o t) + \left(4x_0 - \frac{2\dot{y_0}}{\omega_o}\right) \\
&+ \frac{2}{\omega_o^2} u_y (sin(\omega_o t) - \omega_o t) + \frac{u_x}{\omega_o^2}(1 - cos(\omega_o t)) \\
y(t) =& \left(\frac{4\dot{y_0}}{\omega_o} - 6x_0\right) sin(\omega_o t) - \frac{2\dot{x_0}}{\omega_o} cos(\omega_o t) + (6\omega_o x_0 - 3\dot{y_0})\, t \\
&+ \left(y_0 + \frac{2\dot{x_0}}{\omega_o}\right) + \frac{2}{\omega_o^2} u_x \left(\omega_o t - sin(\omega_o t)\right) + u_y \left(\frac{4}{\omega_o^2}(1 - cos(\omega_o t)) - \frac{3}{2}t^2\right) \\
z(t) =& z_0 cos(\omega_o t) + \frac{\dot{z_0}}{\omega_o} sin(\omega_o t) + \frac{u_z}{\omega_o^2}(1 - cos(\omega_o t)).
\end{aligned}
\tag{3.2}
$$

The discrete state-transition matrix based of the Hill's equations is used to obtain the solution of the linearized dynamical system of the interceptor satellite assuming the state vector $\boldsymbol{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]$.

$$
A = \begin{bmatrix}
1 & 0 & 0 & \Delta t & 0 & 0 \\
0 & 1 & 0 & 0 & \Delta t & 0 \\
0 & 0 & 1 & 0 & 0 & \Delta t \\
3\omega_o^2 \Delta t & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & -2\omega_o \Delta t & 1 & 0 \\
0 & 0 & -\omega_o^2 \Delta t & 0 & 0 & 1
\end{bmatrix}, B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
$$

The linearized state propagation is given by

$$
\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k. \tag{3.3}
$$

The attitude kinematics and dynamics for a rigid body are represented here using the quaternion formulation $\boldsymbol{q}$ with scalar in the first position of the vector. It is possible to

arrange equivalent formulations for other attitude representations such as Euler angles, Rodrigues parameters or rotation matrices. The attitude model considers a rigid body with internal control moments and a potential function that depends only on the attitude.

The attitude kinematics using the quaternion representation with scalar first are given by the following equation:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\begin{pmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{pmatrix}\boldsymbol{q} \tag{3.4}$$

Euler's equation gives the attitude dynamics for each individual satellite as

$$\mathbb{I}\dot{\boldsymbol{\omega}} = \mathbb{I}\boldsymbol{\omega} \times \boldsymbol{\omega} + \boldsymbol{\tau}. \tag{3.5}$$

The torque in Euler's equation can be expanded in two main categories: 1) external torque, $\boldsymbol{\tau}_{ext}$, (from gravity gradient, solar radiation pressure, etc.) and 2) internal torque, also better known as control torque, $\boldsymbol{\tau}_{ctr}$, (from reaction wheels, magnetic torque rods and other actuators). This way $\boldsymbol{\tau} = \boldsymbol{\tau}_{ext} + \boldsymbol{\tau}_{ctr}$. For simplicity only the gravity gradient torque is used as an external torque and all the remaining environmental torques are negligible. The gravity gradient torque can be modeled as

$$\boldsymbol{\tau}_g = 3\omega_o^2 \boldsymbol{z}_o \times \mathbb{I}\boldsymbol{z}_o. \tag{3.6}$$

where $\boldsymbol{z}_o = \boldsymbol{e}_{z_o} = (0, 0, 1)$ is the z-axis vector of the orbital frame. Euler's equation more

explicitly are

$$\dot{\boldsymbol{\omega}} = \mathbb{I}^{-1}\left(\mathbb{I}\boldsymbol{\omega} \times \boldsymbol{\omega} + 3\left(\frac{\mu_{Earth}}{r_{sat}^3}\right)\boldsymbol{z}_o \times \mathbb{I}\boldsymbol{z}_o + \boldsymbol{\tau}_{ctr}\right). \tag{3.7}$$

In the Euler dynamics equation the following are constants: the inertia tensor $\mathbb{I}$, Earth's gravitational parameter μ_{Earth}, and for simplicity the assumption is that the satellite orbit is circular and so the satellite orbital radius r_{sat} is also constant. The variables are the angular velocity of the satellite $\boldsymbol{\omega}$, the z-axis vector of orbital frame $\boldsymbol{z}_o$ and the control torque vector $\boldsymbol{\tau}_{ctr}$.

3.3 Multi-Agent Robotic System

For successful docking each spacecraft must be capable to control autonomously its position and attitude. The other main function of each spacecraft is to share the state information and cooperate with the other spacecraft for collision avoidance. The MARS is responsible for the aforementioned functions. Each spacecraft has a MARS bracket implemented to execute the GNC algorithms to perform the autonomous docking. Figure 3.2 shows the MARS bracket diagram that is implemented in each spacecraft in context with the spacecraft hardware and the space environment that affects the motion dynamics. The MARS cluster combines the various MARS brackets that are implemented in each spacecraft. Figure 3.3 shows the conceptual diagram of the MARS cluster.

For this particular project each satellite knows its exact state and relays that information to its neighbors. The MARS system communicates the satellite states through active message passing instead of relying on active observation of the neighbor satellites. This

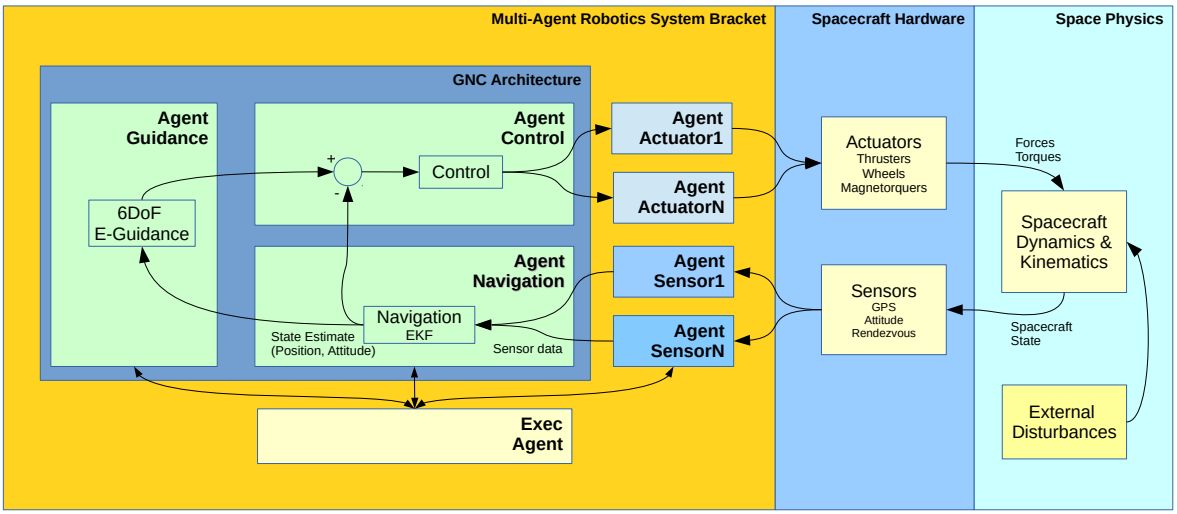

Figure 3.2: MARS bracket with the embedded GNC architecure

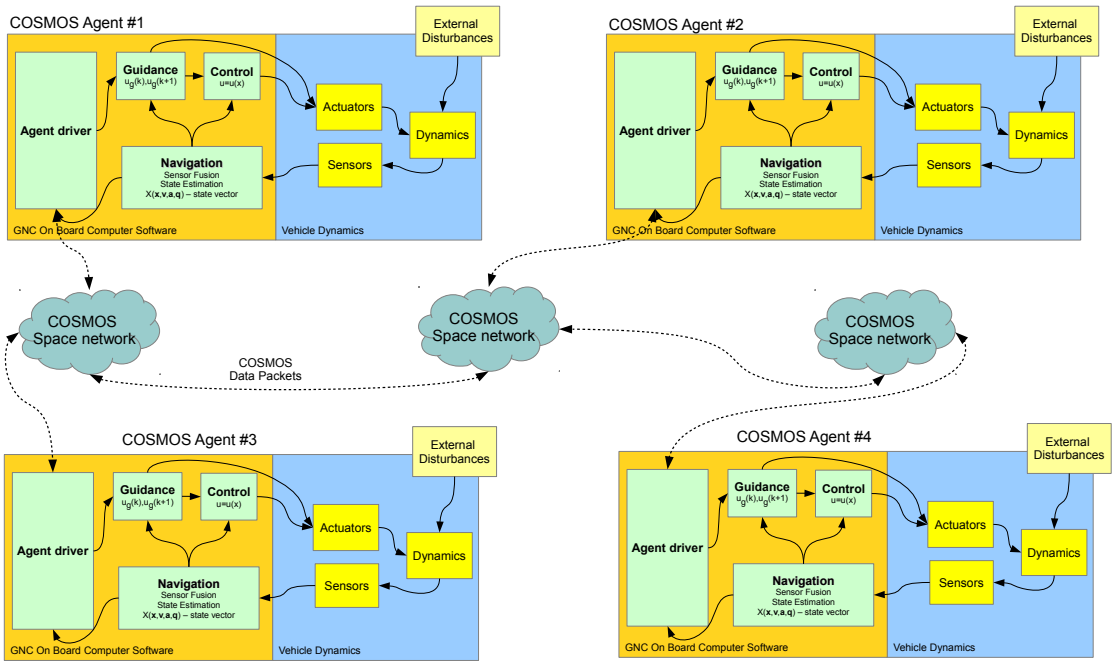

Figure 3.3: MARS cluster of the four satellites.

message passing is accomplished with the agents that exist in each satellite and are able to communicate by sharing the required information. The communication is done using the agents that share data with IP based packets with the UDP protocol over a wireless network.

The MARS is implemented using the COSMOS core libraries which enables a generic and modular approach for software development. Each satellite is considered to be a COSMOS "node". Each node that lives in the same network can talk to every other node using the COSMOS framework. For this particular work there are four COSMOS nodes that share state information for the cooperative control.

3.4 GNC Architecture

This section describes the GNC architecture including the newly developed 6DoF guidance method. The GNC algorithms are implemented into a new software library which is built upon the COSMOS core libraries. One main advantage of developing the GNC software within the COSMOS framework is that the simulating environment such as orbit propagation and disturbance models becomes automatically available. Maybe more importantly the software becomes a part of the COSMOS flight software that can be quickly leveraged for future missions reducing the development time significantly.

Figure 3.4 shows the typical block diagram of the GNC loop. The primary function of the GNC is to collect the data from the various sensors for position and attitude measurements, decide what is the best state trajectory to the target, and produce the control commands for the actuators to maneuver the satellite. The navigation function is to produce the best 6DoF state estimate by knowing the model dynamics and error noise statistics. Then the best state estimate is sent to the guidance and control functions. The guidance function

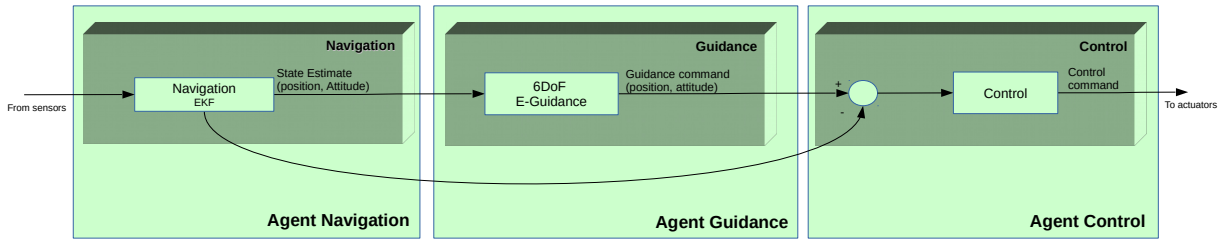

Figure 3.4: GNC architecture in context with the MARS

defines the state trajectory also known as the reference states for the control of the position, velocity, attitude and angular rates. The control function computes the force and torque commands to maintain the satellite in the correct state trajectory. In the next sections a more detailed description is given for each of the GNC functions.

3.4.1 Guidance

The guidance function computes the state reference to maintain the satellite in the correct 6DoF trajectory which includes the position and attitude. The guidance function presented in this work is based of the E-Guidance method that is an explicit method for computing thrust guidance commands given initial and final conditions (boundary value problem)[68]. This method was originally developed for the Apollo Guidance and Navigation problems as an explicit method but it is now expanded for spacecraft rendezvous and docking as a real time onboard guidance law. Part of the E-guidance scheme is the E-matrix that relates the desired final state (final boundary conditions) with the current state (current boundary conditions), this creates a mechanism to compute the guidance coefficients for efficient maneuvering of spacecraft. The E-matrix coefficients determine the required force and torque required to maneuver the spacecraft. The derivation of the 3DoF E-Guidance equations for position is presented in Appendix E. One important difference of the standard E-Guidance equations is that these are derived for the guidance of a vehicle moving on a

central force field and for the ARD application the Hill's frame is an accelerating frame so the E-Guidance equations require the addition of fictional forces to take into account those non-inertial effects. The required acceleration $\ddot{x}$ is then the sum of the thrust acceleration and the fictitious forces given on the Hill's frame. For the x-direction the total acceleration becomes

$$a_{T_x}(t) + a_{Hills} = c_1 + c_2(T - t) \tag{3.8}$$

or

$$a_{T_x}(t) = c_1 + c_2(T - t) - a_{Hills} \tag{3.9}$$

The derivation of the E-Guidance equations for the attitude follows. At some instant $t = t_i$ the navigation agent provides the guidance agent the current attitude state of the spacecraft (attitude $\boldsymbol{q}$, attitude rate $\boldsymbol{\omega}$)

$$\begin{aligned} \boldsymbol{q}(t) &= [q_x(t) \quad q_y(t) \quad q_z(t) \quad q_w(t)] \\ \boldsymbol{\omega}(t) &= [\omega_x(t) \quad \omega_y(t) \quad \omega_z(t)] \end{aligned} \tag{3.10}$$

and the desired attitude target state is

$$\begin{aligned} \boldsymbol{q}(T) &= [q_{x_T} \quad q_{y_T} \quad q_{z_T} \quad q_{w_T}] \\ \boldsymbol{\omega}(T) &= [\omega_{x_T} \quad \omega_{y_T} \quad \omega_{z_T}] \end{aligned} \tag{3.11}$$

Typically the final angular velocity is set to zero

$$\boldsymbol{v}(T) = [0 \quad 0 \quad 0]$$
$$\boldsymbol{\omega}(T) = [0 \quad 0 \quad 0] \tag{3.12}$$

The guidance law will command the spacecraft to maneuver from the current boundary conditions (3.10) to the desired boundary conditions (3.11). The problem in essence is to compute the angular acceleration (torque) given the rendezvous two-point boundary-value conditions for $t_o \leq t \leq T$ such that at $t = T$ the spacecraft attitude meets its target attitude. The energy spent on this maneuver can be evaluated using the angular acceleration by $\int_{t_i}^{T} \sqrt{\boldsymbol{\alpha}_T.\boldsymbol{\alpha}_T} dt$ and this parameter is to be minimized. The derivation of the E-Guidance for the attitude dimensions can be done for each dimension independently. The following derivation takes into account one of the angular positions and its derivatives. θ will be used to represent the angular position.

By integrating the total acceleration from current time $t = t_i$ to a generic time t_f the equations is

$$\dot{\theta}(t) - \dot{\theta}(t_i) = \int_{t_i}^{t} \ddot{\theta}(s) ds \tag{3.13}$$

or equivalently

$$\dot{\theta}_T - \dot{\theta}_0 = \int_{t_i}^{T} \ddot{\theta}(s) ds \tag{3.14}$$

and integrating both sides

$$\theta_T - \theta_0 - \dot{\theta}(t_i) T_{go} = \int_{t_i}^{T} \left[\int_{t_i}^{t} \ddot{\theta}(s) ds \right] dt \tag{3.15}$$

where $T_{go} = T - t_i$. The two integral equations on the acceleration term will be transformed into a pair of simultaneous linear algebraic equations with two unknowns. There is a infinite number of possible solutions but one can limit the factors and propose the following

form for the solution as

$$\ddot{\theta}(t) = c_1 p_1(t) + c_2 p_2(t) \tag{3.16}$$

where p_1 and p_2 are linearly independent and c_1 and c_2 are coefficients chosen to satisfy Equations E.7 and E.8. Substituting the equations we have

$$\dot{\theta}_T - \dot{\theta}_0 = f_{11} c_1 + f_{12} c_2 \tag{3.17}$$

$$\theta_T - \theta_0 - \dot{\theta}(t_i) T_{go} = f_{21} c_1 + f_{22} c_2 \tag{3.18}$$

where

$$f_{11} = \int_{t_i}^{T} p_1(t) dt \tag{3.19}$$

$$f_{12} = \int_{t_i}^{T} p_2(t) dt \tag{3.20}$$

$$f_{21} = \int_{t_i}^{T} \left[\int_{t_i}^{t} p_1(s) ds \right] dt \tag{3.21}$$

$$f_{22} = \int_{t_i}^{T} \left[\int_{t_i}^{t} p_2(s) ds \right] dt \tag{3.22}$$

if p_1 and p_2 are integrable then the f functions are algebraic and the solution for c_1 and c_2 is found by solving the equation

$$\begin{bmatrix} \dot{\theta}_T - \dot{\theta}_0 \\ \theta_T - (\theta_0 + \dot{\theta}_0 T_{go}) \end{bmatrix} = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \tag{3.23}$$

or equivalently

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} \begin{bmatrix} \dot{\theta}_T - \dot{\theta}_0 \\ \theta_T - (\theta_0 + \dot{\theta}_0 T_{go}) \end{bmatrix} \tag{3.24}$$

The E-matrix is the inverse of the F-matrix

$$e_{11} = f_{22}/\Delta \tag{3.25}$$

$$e_{12} = -f_{12}/\Delta \tag{3.26}$$

$$e_{21} = -f_{21}/\Delta \tag{3.27}$$

$$e_{22} = f_{11}/\Delta \tag{3.28}$$

and $\Delta = f_{11}f_{22} - f_{12}f_{21}$. Now assuming $p_1(t) = 1$ and $p_2(t) = T - t$ and the E-matrix for this particular solution is

$$E = \begin{bmatrix} 4.0/T_{go} & -6.0/T_{go}^2 \\ -6.0/T_{go}^2 & 12.0/T_{go}^3 \end{bmatrix}. \tag{3.29}$$

so then c_1 and c_2 are given by

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 4.0/T_{go} & -6.0/T_{go}^2 \\ -6.0/T_{go}^2 & 12.0/T_{go}^3 \end{bmatrix} \begin{bmatrix} \dot{\theta}_T - \dot{\theta}_0 \\ \theta_T - (\theta_0 + \dot{\theta}_0 T_{go}) \end{bmatrix} \tag{3.30}$$

and the solution for the dynamics is

$$\ddot{\theta}(t) = c_1 + c_2(T - t) \tag{3.31}$$

the required angular acceleration $\ddot{\theta}$ is then

$$\alpha_{T_x}(t) = c_1 + c_2(T - t) \tag{3.32}$$

or

$$\alpha_{T_x}(t) = c_1 + c_2(T - t) \tag{3.33}$$

The newly derived E-guidance formulation is expressed by the E-Matrix that is a scaling factor matrix and the P-vector is an equivalent error-to-target vector that accounts for the difference from current position state to the desired target state in each direction i. For the position guidance we have:

$$P_i = \begin{bmatrix} \dot{\boldsymbol{r}}_{i_T} - \dot{\boldsymbol{r}}_i \\ \boldsymbol{r}_{i_T} - (\boldsymbol{r}_i + \dot{\boldsymbol{r}}_i T_{go}) \end{bmatrix} i = x, y, z \tag{3.34}$$

For the attitude guidance the angular difference is represented by the quaternion vector (q_x, q_y, q_z) and its equivalent derivative is the angular rate vector ω. This takes a similar

vector for the angular difference, in this case Q:

$$Q_i = \begin{bmatrix} \boldsymbol{\omega}_{i_T} - \boldsymbol{\omega}_i \\ \boldsymbol{q}_{i_T} - (\boldsymbol{q}_i + \boldsymbol{\omega}_i T_{go}) \end{bmatrix} i = x, y, z \tag{3.35}$$

and the C-matrix, which is the product of the E-matrix and P-vector is given as

$$C_i = EP_i, \quad i = x, y, z, q_x, q_y, q_z. \tag{3.36}$$

Finally the control forces and torques are computed from the C-vector coefficients

$$f_x = (C_{x_1} p_1 + C_{x_2} p_2) - 2\omega\dot{y} - 3\omega^2 x \tag{3.37}$$

$$f_y = (C_{y_1} p_1 + C_{y_2} p_2) - 2\omega\dot{x} \tag{3.38}$$

$$f_z = (C_{z_1} p_1 + C_{z_2} p_2) + \omega^2 \dot{x} \tag{3.39}$$

$$\tau_x = (C_{q_{x_1}} p_1 + C_{q_{x_2}} p_2) \tag{3.40}$$

$$\tau_y = (C_{q_{y_1}} p_1 + C_{q_{y_2}} p_2) \tag{3.41}$$

$$\tau_z = (C_{q_{z_1}} p_1 + C_{q_{z_2}} p_2) \tag{3.42}$$

The acceleration factor in this problem is changed from the original formulation of the E-guidance described in [68] because the reference frame is changed and there are fictitious forces that must be accounted for in the Hill's frame to maintain the correct form.

3.4.2 Navigation

For the navigation solution the Extended Kalman Filter (EKF) is used because of the nonlinear attitude dynamics of the satellite. The EKF is a nonlinear extension to the Kalman

filter. The EKF linearizes about an estimate of the current mean and covariance for better estimates of the nonlinear model.

Consider the discrete non-linear system dynamics and the observation model with Gaussian noise:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= f\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{y}_k &= h\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k \end{aligned} \tag{3.43}$$

The state vector x is defined by the position and velocity vectors $\boldsymbol{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]$. The initial state vector is defined by a random position and velocity x_0 with known mean $\mu_0 = E[x_0]$ and covariance $P_0 = E[(x_0 - \mu_0)(x_0 - \mu_0)^T]$. The sensor measurement state vector y is given by the direct GPS measurements $y = [x, y, z]$ with known error noise. In the EKF a linearization is made about the most recent state estimate predicted. The following equations summarize the EKF steps. The state transition and observation matrices are defined to be the following Jacobians:

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}} \tag{3.44}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{3.45}$$

With the Jacobian matrices the state and covariance estimates must be computed:

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}) \tag{3.46}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^{\top} + \boldsymbol{Q}_{k-1} \tag{3.47}$$

To compute the innovation and innovation covariance we have:

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{3.48}$$

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top + \boldsymbol{R}_k \tag{3.49}$$

With the previous steps it is now possible to compute the Kalman gain:

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1} \tag{3.50}$$

Finally we can compute the updated state and covariance estimates:

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k \tag{3.51}$$

$$\boldsymbol{P}_{k|k} = (\boldsymbol{1} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1} \tag{3.52}$$

The EKF is an iterative prediction process where at each step we calculate the state estimate x, the Kalman gain K and the process covariance matrix P. This process gives a good navigation estimate for the Hill's equations given the known dynamical model of each spacecraft and its thrust control input and sensor informations.

The Measurement Model is based of the GPS sensor which gives direct measurement over the relative position of the target with respect to the interceptor. Then the linearized sensor matrix is:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The sensor is a GPS that measures directly the position of the interceptor and the target, the difference in these two positions gives the relative position which is directly used in the

model. The sensor has a standard deviation for each axis of:

- $\sigma_z = 6$ m
- $z_{meas,k} = z_k + v_k$ where $v_k \sim \mathcal{N}(0, \sigma_z^2)$ and z_k is the true state

For the process covariance high values are given because of the initial uncertainty

$$P_0 = \begin{bmatrix} 10^3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^3 \end{bmatrix}$$

and the process noise matrix is

$$Q = \begin{bmatrix} 10^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^2 \end{bmatrix}$$

Finally the sensor noise matrix is defined as

$$R = \begin{bmatrix} 10^2 & 0 & 0 \\ 0 & 10^2 & 0 \\ 0 & 0 & 10^2 \end{bmatrix}$$

3.4.3 Control

After the current state of the satellite is estimated it is necessary to compute the automated maneuver to approach the desired state in an optimal way. Using Linear Quadratic Regulator (LQR) is one way of doing this because it defines an optimal feedback control for a linear system such as for the Hill's equations for the simplified case are. From the state space model

$$\dot{X} = AX(t) + Bu(t) \tag{3.53}$$

$$Y = CX(t) + Du(t) \tag{3.54}$$

and for the continuous case

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 3\omega^2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -2\omega & 1 & 0 \\ 0 & 0 & -\omega^2 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = I_{6,6}, D = 0$$

The algebraic Ricatti equation can be solved to find the solution to this optimal problem:

$$u(t) = -K_{LQR}(X(t) - X_{desired}) \tag{3.55}$$

By having u we can determine the final time at which the interceptor gets into the desired state T that is then plugged into the guidance equations.

In the case of this project the interest is to minimize the fuel or energy expenditure, and not so much the time. Using the cost representing energy expenditure we have:

$$J = min_{u_0,\ldots,u_{k-1}} \sum_{i=0}^{k-1} \|u_i\| \tag{3.56}$$

3.5 Tools for ARD Simulations

3D visualization is an important aspect of complex simulations such as ARD. Using the COSMOS framework and the open Qt framework a number of Tools have been developed to facilitate the execution and visualization of the simulation results.

Mission Operations Support Tool (MOST), see Figure 3.5, is one of the primary COSMOS Tools that is used to visualize the overall satellite behavior. MOST, while providing some capability for communicating, serves the primary purpose of analyzing incoming telemetry to provide visualization of the current SOH of the satellite or "node", as well as its context with respect to other vehicles. Any data variable in the COSMOS namespace can be configured to be displayed graphically using the strip charts in MOST. Particular instances of MOST can be tuned to particular missions, taking advantage of specialized subsystem screens. The entire operation of MOST is time-based, allowing for either real time viewing, or playback at variable speeds. MOST can work in various modes that allow use with both real-time as well as archived actual or simulated data, and is capable of extrapolating date based on spacecraft and system models.

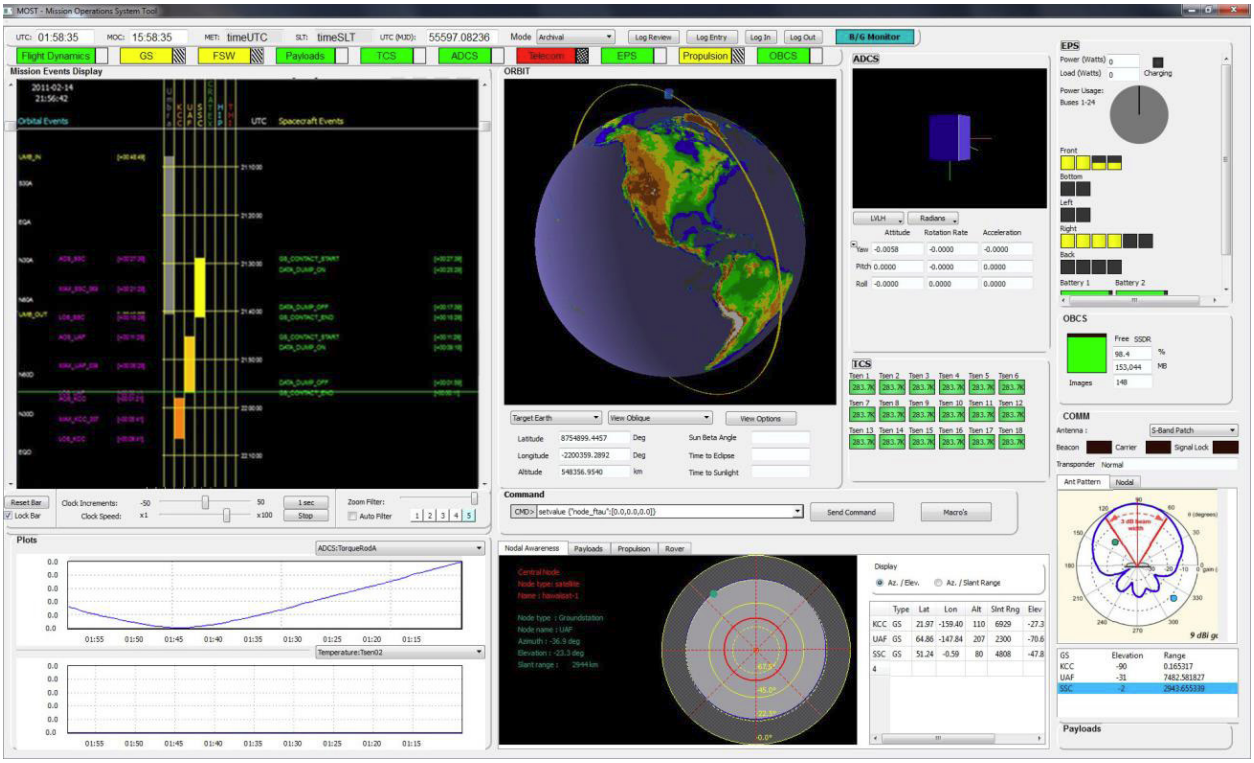

Figure 3.5: MOST monitoring one satellite in orbit.

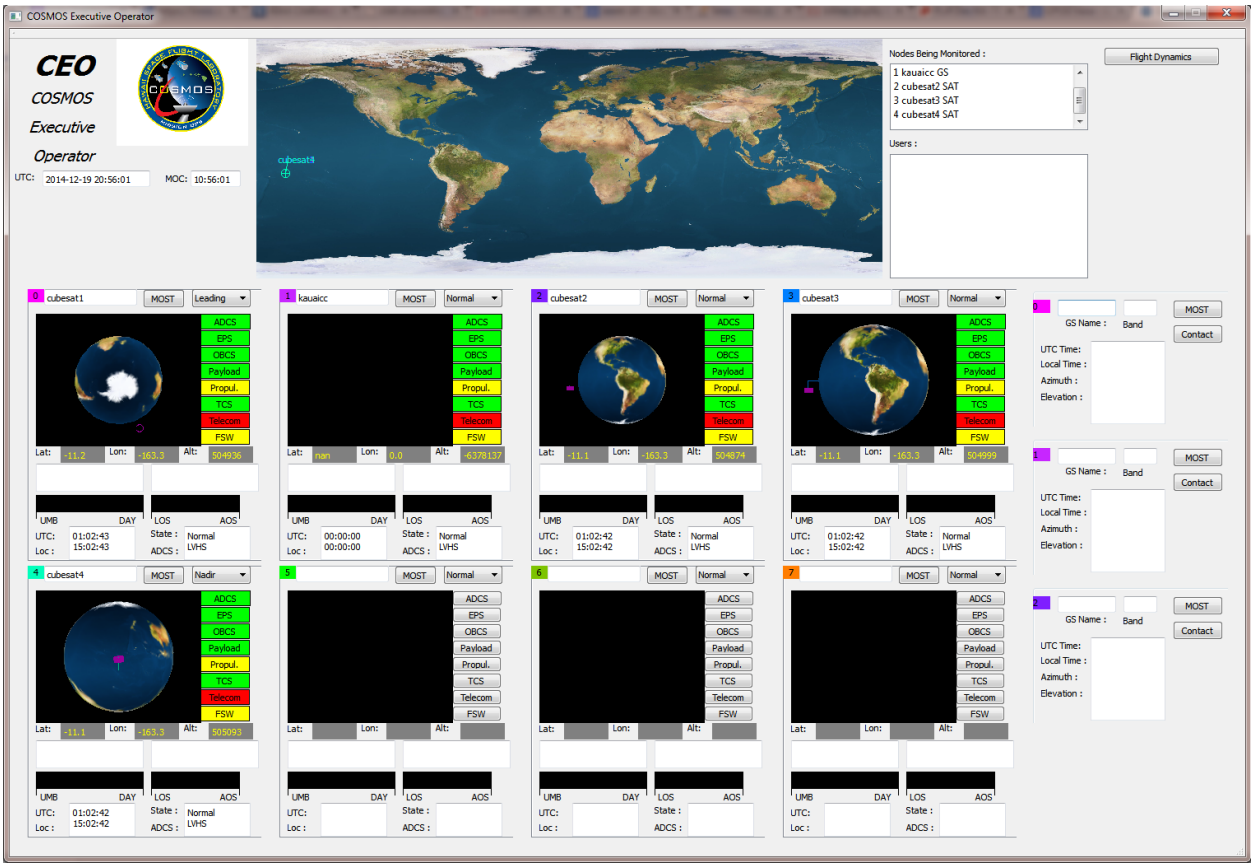

Figure 3.6: The CEO monitoring four CubeSats in cooperative motion.

While MOST is designed to operate a single vehicle, with some indication of the surrounding context, such as other satellites nodes, the COSMOS Executive Operator (CEO), shown in Figure 3.6, is responsible for the multi satellite context. CEO monitors all satellite nodes in a given sphere of operation, displaying a summary of their , and providing access to their various tools (such as MOST, and other COSMOS Tools for more unique nodes.) Through a combination of CEO and MOST, a complete picture can be derived of the current state of the satellite cluster. Figure 3.6 shows a simulation of four satellites in cooperative motion for docking. The CEO overviews the whole set of satellites and can control either the whole set or by bringing MOST up it can monitor and control individual satellites.

Any standard program can collect data from the COSMOS network by using the COSMOS agents paradigm, and through use of the special functions provided by the COSMOS libraries. At the basic level, a COSMOS aware program will be able to sense the data being passed by other COSMOS agents. Using the COSMOS libraries, this COSMOS "client" will be able to gather data, decode and convert to other data formats as needed. For extended functionality, it will be able to communicate with other COSMOS agents also requesting them to perform specialized functions such as adding noise to a particular system.

The COSMOS 3D docking viewer program was developed as a COSMOS client to visualize ARD mission and specifically visualize the performance of the docking maneuvers. Figures 3.7 and 3.8 show the 3D docking viewer at two different stages of an example docking sequence. The COSMOS 3D docking viewer program allows the mission engineer to run multiple simulations from given initial conditions and then replay the simulation in different ways: real time, faster than real time, reverse time or show the state of all the satellites at a specific time with a slider. The docking viewer shows the 3D maneuvers of each satellite and also shows the preselected performance plots on the right side. Each satellite is drawn to scale. It is also possible to select which performance plots must appear of the right hand side of the program.

3.6 Simulations

The simulation scenario consists of four CubeSats that are separated from the rocket with a typical separation velocity of 1 m/s radially. Each CubeSat is separated by 90 degrees around the spin axis of the launch vehicle. The scenario considers simplified conditions compared to a realistic launch since the objective is to test the algorithms and proposed method. The CubeSats are considered to be fully actuated with six thrusters for position control and

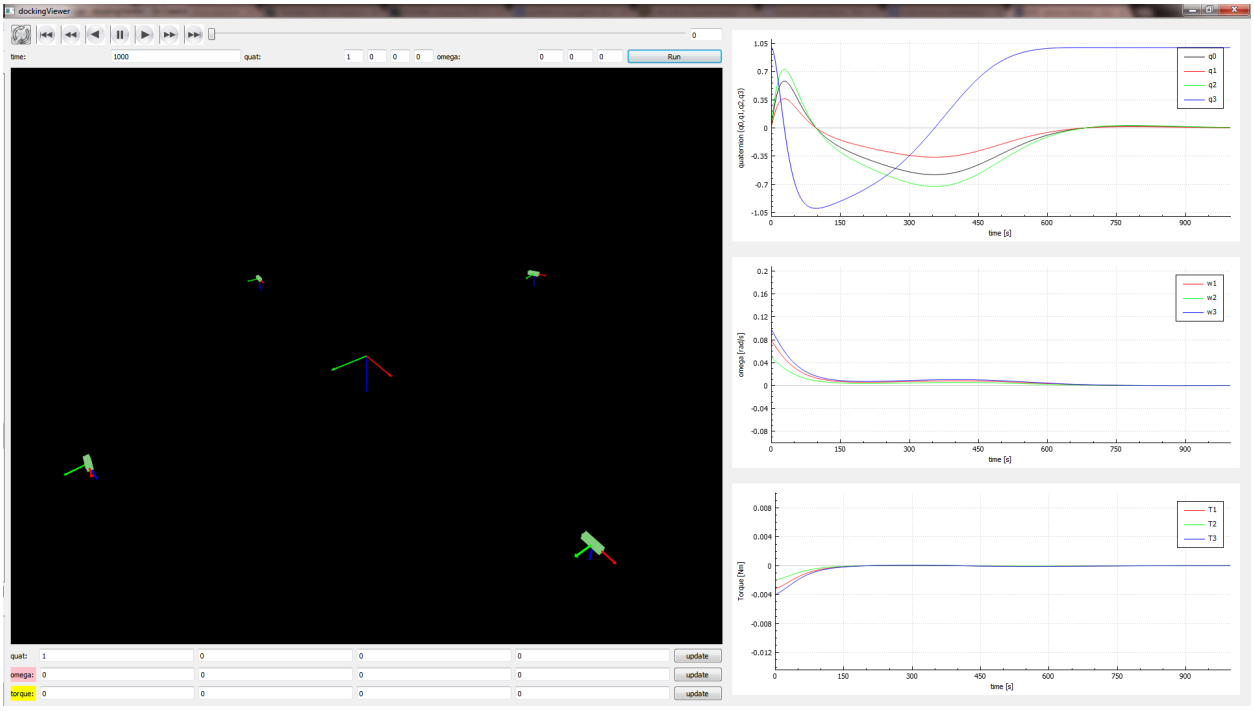

Figure 3.7: COSMOS 3D Docking Viewer with four CubeSats at their initial configuration.

three reaction wheels for attitude control.

The docking maneuver requires the CubeSats to navigate from any given initial condition and arrive with optimal performance to a target position and designated velocity. In this case the designated velocity is zero. The docking sequence is chosen according to Figure 3.9 where each arrow represents the direction of the CubeSat trajectory so that the probability of collisions between each CubeSat is minimized. CubeSat-1 is ejected towards the $+X$ axis and its target position is aligned with the same axis. The equivalent configuration happens for the other three CubeSats.

On Figure 3.10 the separation distance from the target position for each of the four CubeSats can be seen. As expected from the GNC algorithm each CubeSat takes a different time

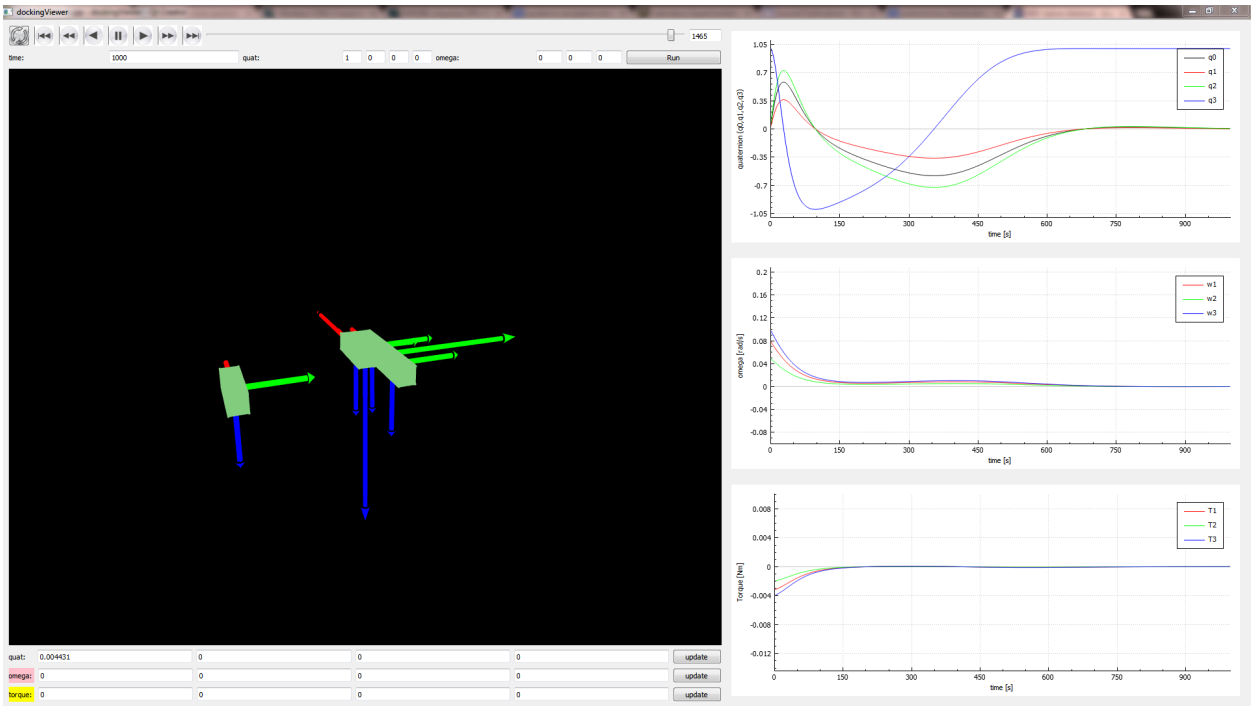

Figure 3.8: COSMOS 3D Docking Viewer with four CubeSats at the end of the simulation.

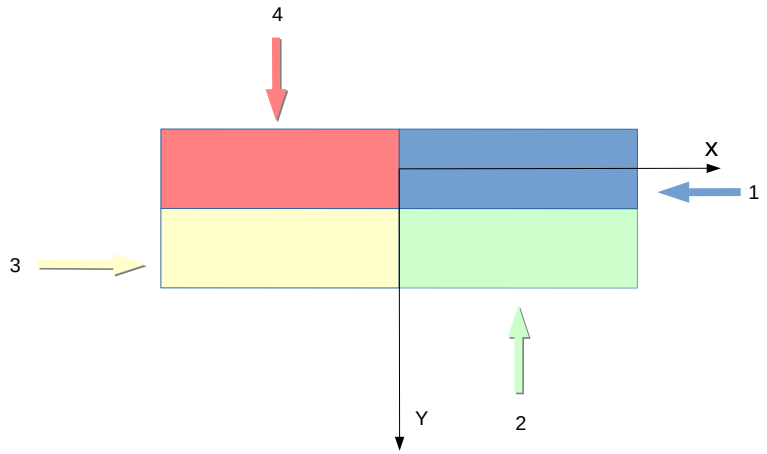

Figure 3.9: The diagram shows the docking sequence for this particular simulation of four CubeSats.

to return to the target position. A fixed trip duration was chosen a priori for each CubeSat, for CubeSat-1 $T_{go_1} = 1000s$, for CubeSat-2 $T_{go_2} = 1100s$, for CubeSat-3 $T_{go_3} = 1200s$, for CubeSat-4 $T_{go_4} = 1500s$ and Figure 3.10 shows that the time-to-go is achieved and the four CubeSats are all docked after 1500 seconds. For this particular problem the docking maneuver has a U-shape where each CubeSat initially navigates away from the target, reaches a maximum distance and stops the escape movement and then slowly comes back to reach the designated target with zero-velocity state.

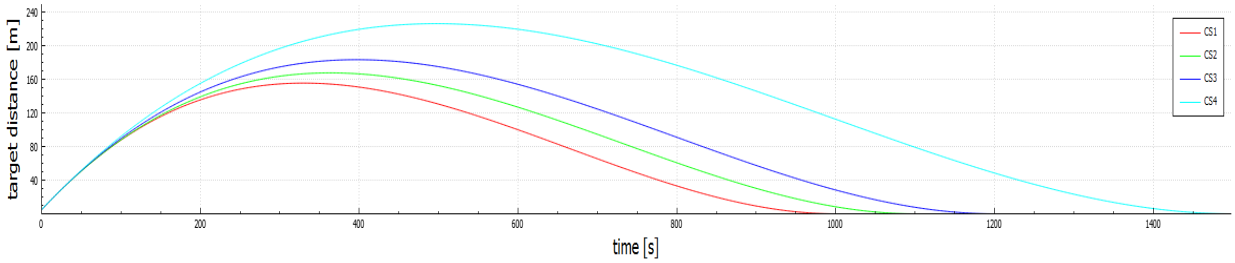

Figure 3.10: Virtual target distance during the docking maneuvers for the four CubeSats.

Equivalently Figure 3.11 shows the evolution of the state position and velocity of CubeSat-1. The other CubeSats have similar performance curves. Each maneuver exhibits the expected behavior since the launcher vehicle needs to expel the CubeSats from the payload stack with the prescribed exit velocity and then have to come back to the target position which in this case is the origin of the orbital frame. The target position can be easily changed as necessary. The final state of the plot shows that the CubeSat reached the target position with zero velocity as expected.

Figure 3.12 shows the attitude performance for CubeSat 1 and 2. The performance of each CubeSat is represented by three vertical-plots. The plot on the top shows the evolution of the attitude quaternion and at the end of the maneuver the plot shows that the attitude

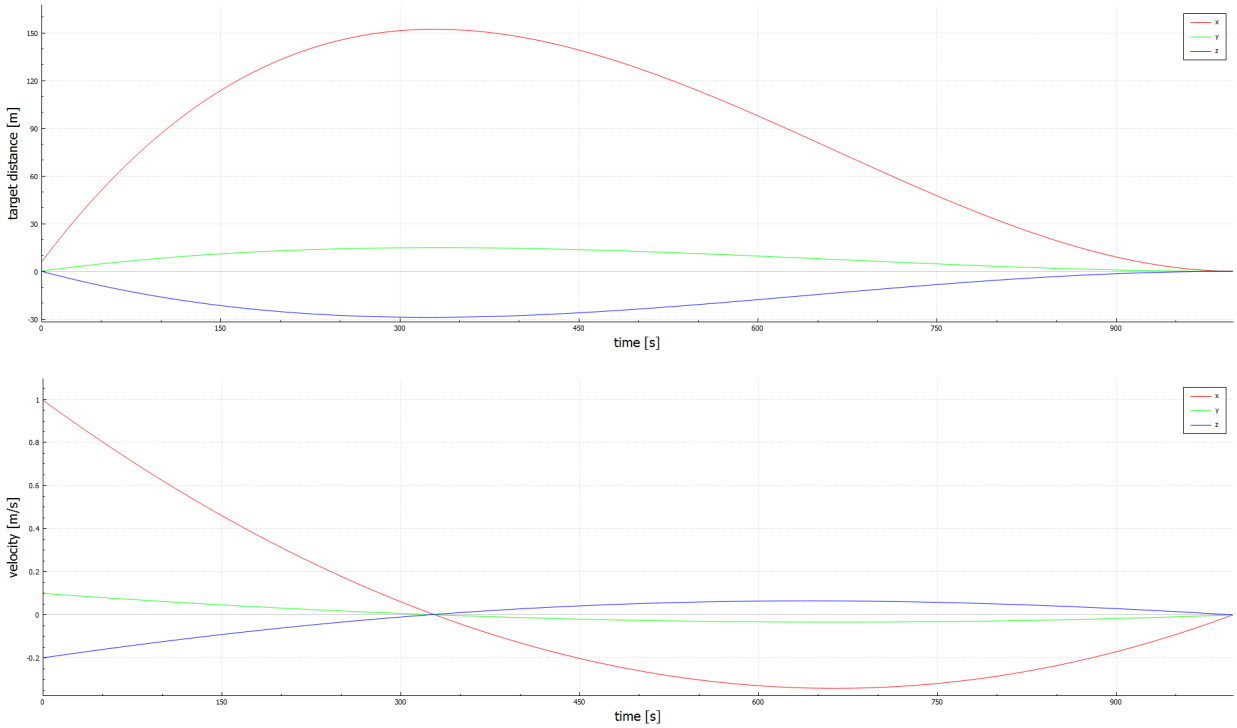

Figure 3.11: Position and velocity for the simulated ARD of CubeSat-1 with target satellite using E-guidance

for the CubeSat is the identity quaternion which is the target attitude. The plot in the middle of the column shows the angular velocity for the CubeSat and at the desired target the angular velocity is zero, as expected. Finally the third plot at the bottom of the column shows the torque used to perform the attitude maneuvers. Similarly on Figure 3.13 the attitude performance for CubeSat 3 and 4 can be seen. These plots show that all of the four CubeSats return to the predefined target with the desired attitude and so a docking maneuver is possible.

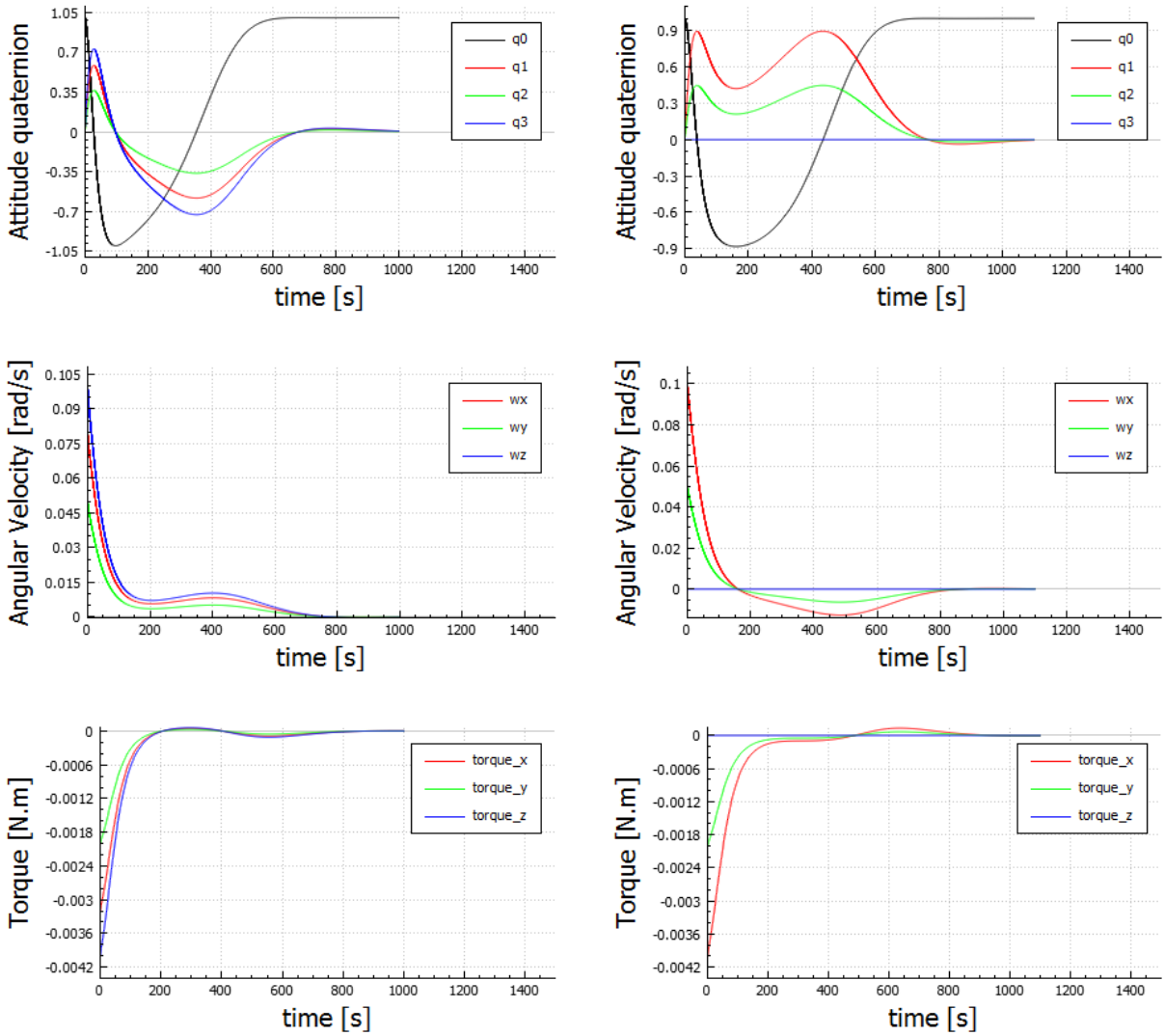

Figure 3.12: Docking maneuvers performance of CubeSat 1 and 2 with attitude, angular velocity and torque

3.7 Conclusion

The problem of ARD for multiple satellites is motivated by the emergence of small satellite technologies that will enable multi satellite missions in the near future such as the assembly of large space structures in space. In this work an extended E-guidance method is introduced

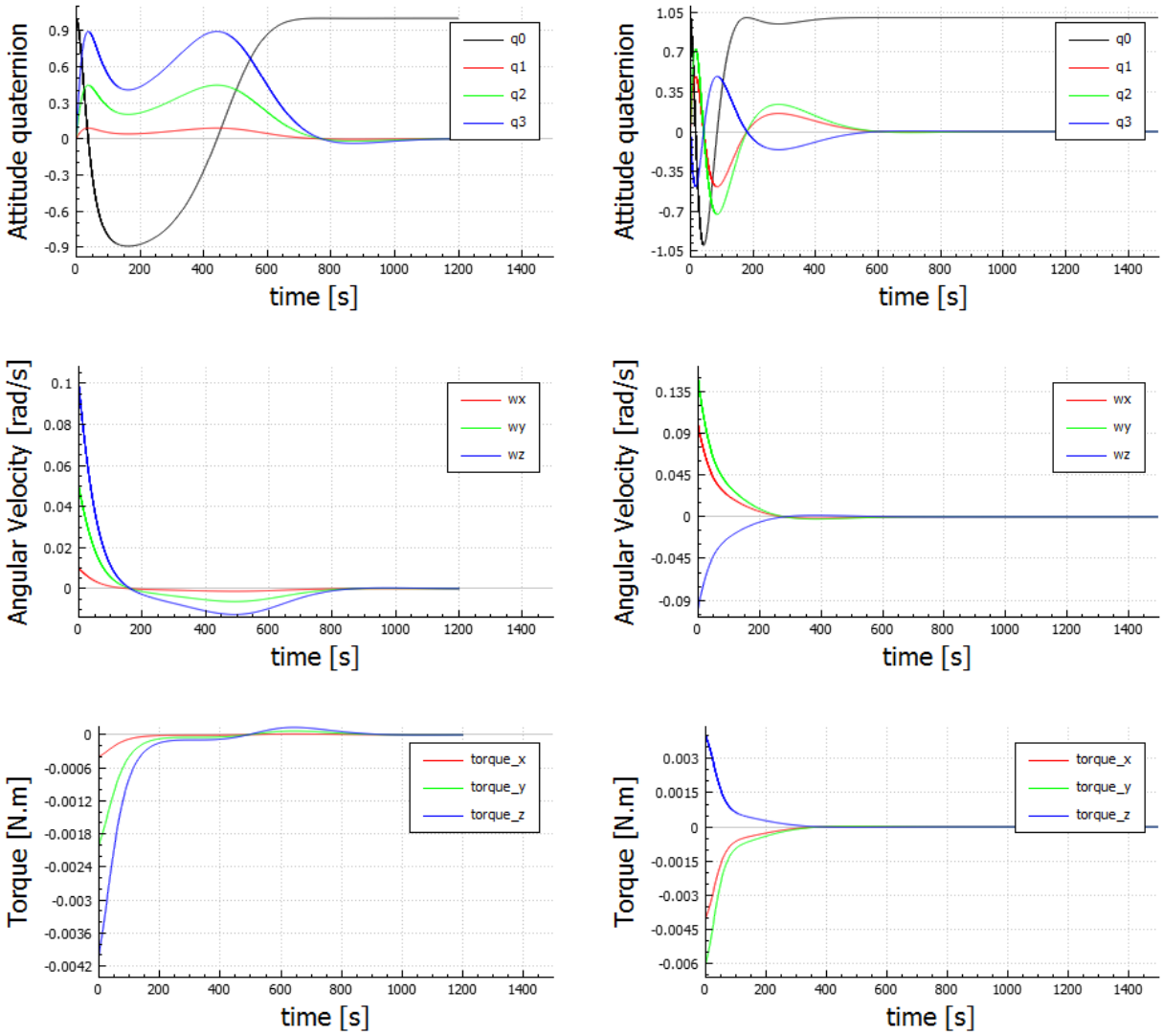

Figure 3.13: Docking maneuvers performance of CubeSat 3 and 4 with attitude, angular velocity and torque

for attitude guidance of a complete 6DoF real-time GNC architecture for ARD of multiple CubeSats that are fully actuated. The 6DoF GNC architecture is integrated into a MARS using the COSMOS software enabling the intercommunication between each satellite. A case study is presented for ARD of four CubeSats deployed from a payload stack of a rocket with successful maneuvering to desired target under 1500 seconds or 25 minutes which is

approximately 30% of a full orbital period in LEO. For this particular case the assembly of structures in space is possible within one orbit. Future work will look into underactuated CubeSats and a collision avoidance mechanisms for docking multiple CubeSats using the MAS.

CHAPTER 4
SATELLITE CONSTELLATIONS OPTIMIZATION

4.1 Introduction

Today's embedded computers are smaller and more capable than ever before. This is the case not only in terms of computational power but also in increased power efficiency. These advances are rapidly increasing the capabilities of small satellites making them more capable. Because of these technology advances the use of small satellites is becoming increasingly common. These capabilities are also driving down the satellite production costs. Recently a few commercial companies (SpaceX, OneWeb, Siemens) announced the launch of large satellite constellations using small satellites for Internet connectivity around the globe. These ventures would not be possible a few years ago but with the advances of embedded technology they are now. For these reasons, new designs of constellations of small and cost-effective satellites are currently being sought for future space missions.

This project is based on the hypothesis that GA optimization techniques can be effective in the design formation of satellite constellations when mission objectives and specific requirements are given. Satellite constellation design usually starts with human reasoning based on the design problem but a non-optimal solution is obtained since the design space is extremely large and all possibilities cannot be accounted for. The optimization method used in this work was based on GA because the design of a satellite constellation for the given missions requirement is not formulated with analytical solutions and is also a complex problem that involves the combination of a large amount of variables and parameters that leads to an very large number of solutions[69]. Some of the design variables are the orbital parameters of each satellite, the number of orbital planes, the spacing in between each

satellite, the total number of satellites, and others. This work proposes that by employing GA based optimization techniques to find solutions for the configuration of large satellite constellations we can not only minimize the number of satellites required for the mission but we can also find solutions that would be practically impossible for a human to devise using standard constellation configuration techniques.

GAs enable a search mechanism with specified constraints to place the satellites and evaluate their configuration with a fitness function. A GA is a search heuristic that generates an insightful and more useful solution at each iteration based on a fitness value of the previous generation. GAs are typically used to find the optimal solution of problems that are not well defined or difficult to model like discontinuous sets, highly non-linear functions, stochastic or even with undefined variables. These are also used to find solutions for problems that are difficult to solve with traditional optimization algorithms. The GA algorithm starts with a population of strings (also called chromosomes or the genotype of the genome) each one carrying a genotypic content. These encode the candidate solutions (also called individuals or phenotypes). The genotype has the primitive parameters (the genes) that determine the individuals' layout and topology in the context of the methodology. Figure 4.1 exemplifies the concept of the genotype content and its representation through the phenotype which also refers to the geometric and physical parameters for the application object.

In this section a method for satellite constellation optimization is proposed based on a new constellation configuration generator and also using a GA that evaluates the performance of the generated satellite constellations by the interaction of Mathworks MATLAB® (MATLAB) with STK. This project was done to leverage a general method to unveil different constellation designs which can incorporate given constraints and optimize the number and overall distribution of the satellites so it may enable new space missions that may have

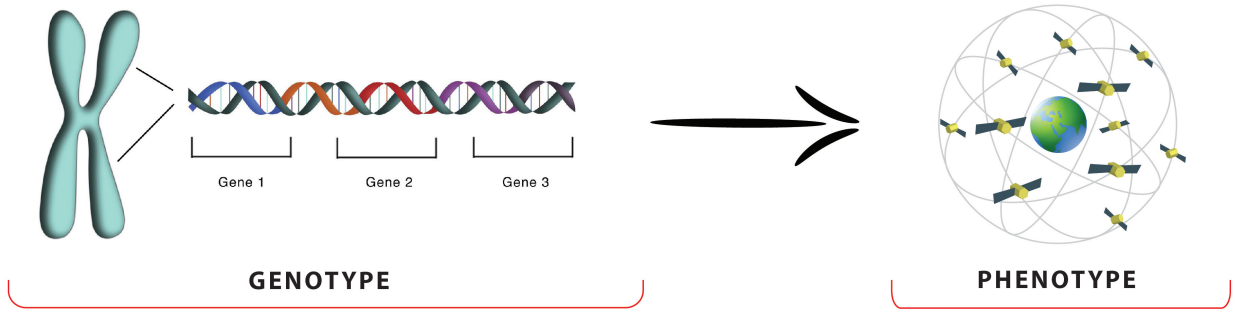

Figure 4.1: Example of a chromosome and its genes.

never been thought of before because of unknown solutions or other limitations such as the previous prohibitive cost of a large number of satellites.

4.1.1 Motivation

There are two main satellite constellation concepts that drive this work: 1) what is the optimal configuration of a satellite constellation to observe the surface of all the oceans of the globe within one day or less of revisit time? and 2) what is the optimal satellite constellation to observe a volcano with a revisit time of three hours or less? The Landsat program has a similar mandate to cover the globe in as little passes as possible. Landsat's 7 main goal is to collect photos of the globe to provide up-to-date and cloud-free images, but it takes 16 days to completely revisit the globe. Using multiple satellites at a LEO the revisit time may be reduced significantly but an optimal configuration must be determined to minimize the number of satellites and maximize the revisit frequency.

For the case of the ocean observation constellation, optimal results are presented for the satellite constellation that revisits every point in the oceans for complete coverage in every 24, 12 and 4 hours. These results are combined into a general equation providing a benchmark for future studies that require a complete coverage of the oceans within a specified revisit

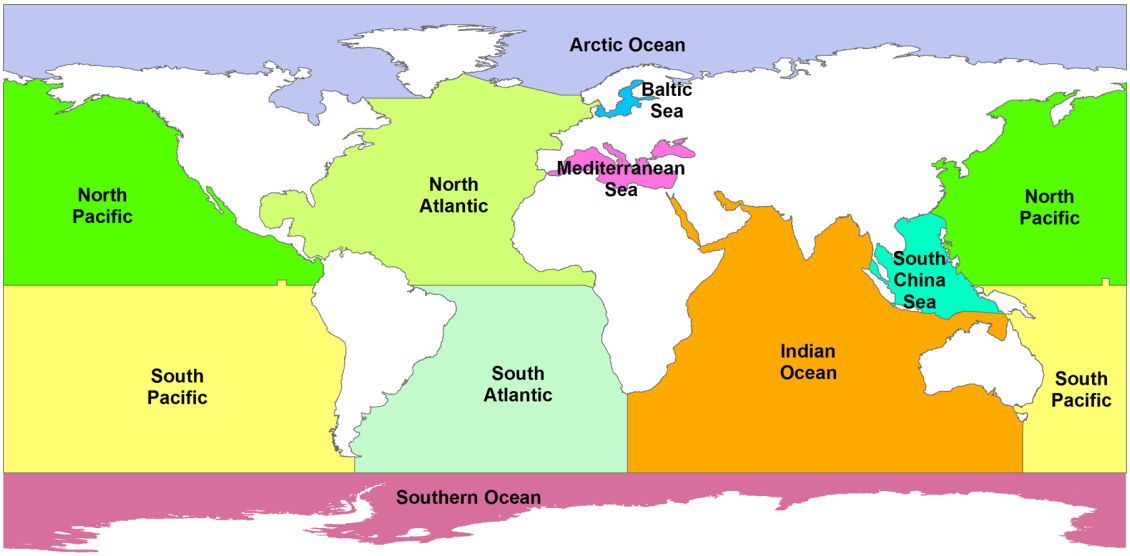

Figure 4.2: The different oceans of the world. Image credit: NOAA

time, with a daytime lighting constraint. This work does not address the cloud coverage aspect that could also be critically important. There are various motivators for collecting near real time data for the oceans of the globe such as monitoring the temperatures, the currents, the volumes, the plancton and even for maritime piracy. Figure 4.2 shows the vast area that the global ocean covers[1].

The second concept of using a satellite constellation that serves to monitor in near-real time the ash clouds from volcanic eruptions can lead to more efficient airline traffic. The cloud of ashes from the volcanic eruptions can lead to the closure of airports because it poses a serious danger to aircraft engines. The airspace is closed for many hours, even for days or weeks. This was the case in Central Europe, UK and Scandinavia in 2010 when the Eyjafjallajökull volcano eruptions forced the airplanes to stay on the ground. Figure 4.3 shows the 2010 Eyjafjallajökull volcano eruption in Iceland. The picture on the left was taken by Envisat's Medium Resolution Imaging Spectrometer (MERIS) by April 15th 2010

[1]Image retrieved from `http://www.ngdc.noaa.gov/mgg/global/etopo1_ocean_volumes.html`

and it shows the volcanic ash clouds sweeping across the UK that is more than 1000 km from the eruption in Iceland. The picture on the right shows a plume from the Chaiten volcano, Chile, seen from MODIS on the Terra satellite in 2008 stretching across Patagonia from the Pacific to the Atlantic Ocean. The 2010 eruptions consequently made the airlines loose $250 million per day[2]. This amount accumulated up to $1.7 billion in lost revenue during the eruption period. This started a heated debate between the airlines, governments, and other agencies responsible for the safety of the airspace about whether or not it was safe for airplanes to fly and if so where was the limit drawn. These agencies depend on the volcanic ash aviation warning systems that has been developed over the years to support these unpredictable scenarios. The International Civil Aviation Organization (ICAO) along with the World Meteorological Organization (WMO) created the International Airways Volcano Watch (IAVW). The IAVW is the source for disseminating volcano ash warnings through the nine Volcanic Ash Advisory Centers (VAAC) around the world and from these to the aviation sector. There are other volcanic monitoring systems that could also help in similar situations such as the MODVOLC[3] run by the Hawaii Institute of Geophysics and Planetology which ingests MODIS Terra and Aqua satellites data to show global volcanic hot areas and possible eruptions. There are also models that can be implemented for tracking ash plumes. One such example is the Puff model developed at the University of Alaska Fairbanks[4] using the Advanced Very High Resolution Radiometer (AVHRR) data[5]. The main problem with volcanic ash tracking still lies in the fact that the data collection is limited since there is currently no way to completely survey the volcanic ash volume in the atmosphere in near-real time and with enough precision[70]. Because of this the agencies tend to be overcautious for safety reasons, but this could potentially prohibit airplanes to take off while it would be

[2]http://www.telegraph.co.uk/travel/travelnews/7609356/Volcanic-ash-cloud-European-flights-set-to-resume.html

[3]`http://modis.higp.hawaii.edu/`

[4]http://pafc.arh.noaa.gov/puffweb2/puffweb.php

[5]`http://noaasis.noaa.gov/NOAASIS/ml/avhrr.html`

perfectly safe to fly if more accurate ash clouds progression reports existed. This problem can be mitigated if a satellite constellation is designed to revisit the ash cloud volume in a fast revisit time.

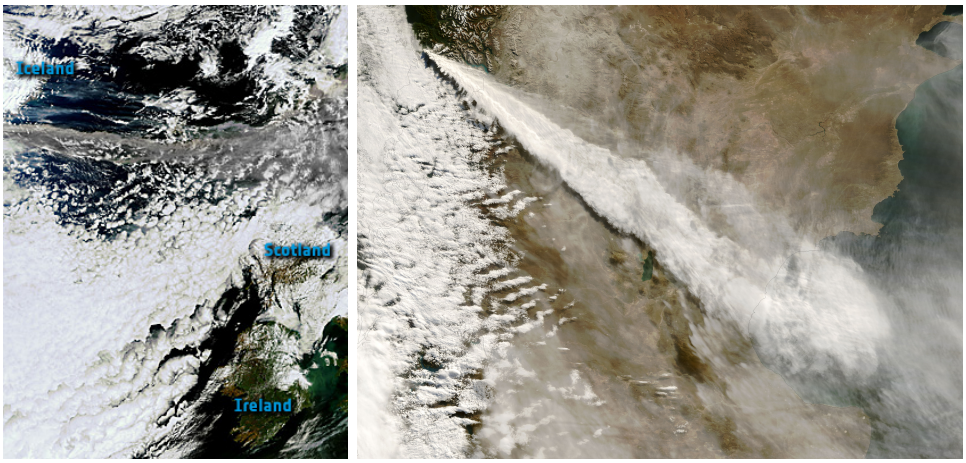

Figure 4.3: Eyjafjallajökull volcano eruption in 2010 (left). Chaiten volcano eruption in 2008 (right).

This work presents results for the optimization of a satellite constellation that completely covers the oceans in a revisit time of 24, 12 and 4 hours. Another satellite constellation is studied to cover in 4 hours or less the North Atlantic Tracks where most of the air traffic flows between the USA and Europe. The ultimate goal of this research is to provide a general method that searches for the near optimal solution of the multi-domain problem of satellite constellations configurations. By providing such a method and a catalog of case studies, future satellite constellation missions can be evaluated and benchmarked against the results here presented.

4.1.2 Contributions

The most important contribution introduced with this work are: 1) a new satellite constellation configuration generation algorithm for Earth observation satellites 2) the formulation of the satellite constellation using a genotype for a GA s optimization problem and 3) the validation of these methods with benchmark results of the optimization of two study cases of optimal satellite constellations configuration.

The results can be used as a reference for future satellite constellation missions that will need to cover large regions on Earth and have similar requirements. Table G in Appendix G shows the most important satellite constellations available today or that are currently being designed. Most of the existing satellite constellations are focused on the telecommunications sector rather than for Earth observations but this is due to change in the near future. Of the Earth observation constellations only the RapidEye has similar characteristics to the current study but the revisit time is within the 3-4 days, which is not within the requirements of this study (less than one day). This shows the uniqueness of the proposed satellite constellations. The uniqueness in this study is based on the fact that the remote sensing observation is focused on a very specific area target (the oceans), on a very rapid revisit time with a daylight only coverage constraint. Finally, another important contribution brought forward in this work is the software and methodology to optimize any kind of Earth observing satellite constellations. The software that was developed is modular and allows the user to set different optimization parameters subject to the most common constraints.

4.1.3 Related Work

Satellite constellation design studies were started in the late 1950's to understand the feasibility of multiple satellites for communication purposes[71, 72]. One important pioneer of

satellite constellation design was J. G. Walker because he extended a geometrical approach towards constellation design in the early 1970's [73]. The classical constellation design formulation has been used from the 1950's up until the 1990's. But the current modern approaches for constellation optimization vary in greatly from closed form formulations to computer intensive optimization algorithms[74].

There have been several approaches to satellite constellation optimization using GA. These vary greatly in the satellite constellation requirements, configuration generation and mission objectives. Pegher[75] uses a GA to optimize coverage and revisit time for military satellite constellations. This problem uses a fixed number of satellites of no more than 25 for a radar sensor constellation. Xie[69] develops a GA based optimization for a LEO constellation also using a radar sensor. A Walker delta constellation method is used for the constellation configuration generation. Ely [76] develops GA for zonal continuous coverage with a maximum number of 40 satellites for sensors with large aperture. These studies do not add requirements such as daylight optimized configurations or variable sensor visibility that are considered in this work. Also the new method for geometric distribution of the constellation introduced with this project is adapted for the large coverage and short revisit time satellite configuration problem by allowing more configuration patterns than the typical Walker delta constellation configuration that is used in the previous studies. Most problems addressed previously are also different from this research problem because the coverage areas considered in this work is significantly larger while using smaller satellites, with smaller optics, given the recent technology improvements. Also the introduction of mission constraints where mostly non-existent in the other problems introduced in the literature. These two factors play a major role in the new methodology introduced. In fact, the applicability of the method developed to solve the current design problem with the large coverage area with predefined constraints makes it a general enough method applicable to new studies of a

number of other possible satellite constellations (like the monitoring of all volcanic activity on Earth in near real time). From the literature review it was found that the application of the other GA based methods was not generally applicable to this work because of the issues discussed above.

4.2 Problem Definition

A group of satellites working towards a common purpose is called a satellite constellation. The main purpose of this study is to find satellite constellations that cover a specific region or the whole globe. The objective of a single satellites in a constellation is often the collection/transmission of data for a particular region on Earth by using a radar, imagers or some other kind of sensing device. The interest of a satellite constellation is to expand and group the capabilities of single satellites and so cover at once a large area of interest on Earth for nadir pointing sensors/transmitters. The goal of designing a satellite constellation is to meet the mission requirements and find the minimum number of satellites necessary for continuous coverage, or coverage with few discontinuities. This is referred as area coverage and can be represented in the total area covered or in terms of percentage of the desired area to be covered. This is the first metric used in the optimization problem. Another important metric of interest for this work if a particular area is fully covered within the expected revisit time. During the past decade new constellation design studies have explored these two metrics[77].

To define each satellite orbit we need six orbital parameters (the Keplerian Elements), and thus a constellation of N satellites will require $N * 6$ parameters. But because every constellation mission has different imposed requirements and constraints, the total number

of parameters increases non-linearly totaling more than $N * 6$. This type of problem can be categorized as a Mixed Integer Nonlinear Optimization Problem (MINLP). Mixed Integer Nonlinear Problems refer to mathematical problem sets with a mix of continuous and discrete variables. MINLPs are used to formulate problems where the discrete system structure and parameters are to be optimized simultaneously [78]. The problem of satellite constellation optimization is complicated by adding constraints on the orbits of the satellites and on their operation requirements.

Coverage is usually defined as the accumulated region on the Earth's surface that is in direct line of sight with the satellite. The accumulated satellite coverage footprints forms the coverage geometry. Most classical satellite constellation design methods are based on a coverage geometry that focuses on nadir pointing for surface coverage with push broom type sensors. This assumption, which is accurate for electromagnetic sensing and imaging, is based of the common coverage geometry shown in Figure 4.4. In this type of coverage geometry, all the parameters can be obtained analytically from planar and spherical geometry calculations. This is derived directly from the geometry of Figure 4.4 using the Law of Sines, where θ is the elevation angle, r_e is the radius of the Earth, r_{sat} is the orbital radius, h is the orbital altitude and β is the Earth central angle of the coverage footprint. The analytic representation allows coverage to be completely determined with respect to the arrangement of the satellites.

Coverage performance can also be computed numerically using a set of grid points arranged around the surface of Earth [79]. This method is often substituted for pure analytic coverage determination. In these cases, the grid of points and the time step for analysis must be carefully defined.

There has been various attempts to classify constellation arrangements [80]. The most

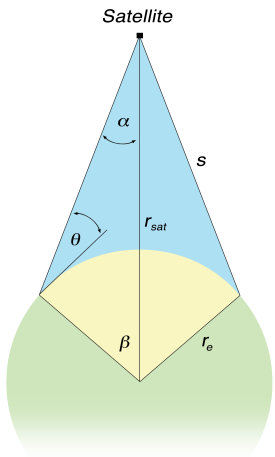

Figure 4.4: Satellite field of view geometry θ is the elevation angle, r_e, is the radius of the Earth, r_{sat} is the orbital radius, and β is the Earth central angle of the coverage footprint.

common arrangements are defined by sensor coverage geometry and desired area coverage, this way a coverage definition can be created. When satellites are arranged with common orbital characteristics (e.g. inclination and altitude) and other design conditions (e.g. coverage area lighting constraint), that arrangement is made a constellation type because it defines the basic structure of a satellite arrangement. Still, there are infinite possible combinations within a constellation type each of which will have a unique constellation configuration and correspondingly a unique behavior that will affect the mission for better or worse. The possible variations come from design parameters like the change in inclination, or the total number of satellites per orbital plane. Within a constellation type there can different coverage methods for arranging the satellites.

4.3 Problem Formulation

This section explains the most important components of the methods developed and used on the research presented in this work. It also presents the new tools that support the presented methodology for the optimization of satellite constellations with multiple mission

constraints.

4.3.1 Satellite Constellation Models

There are various ways of defining a satellite constellation. For this project a generalization of the Satellite Walker Constellation was developed as the baseline for the geometric formulation of the satellite constellation.

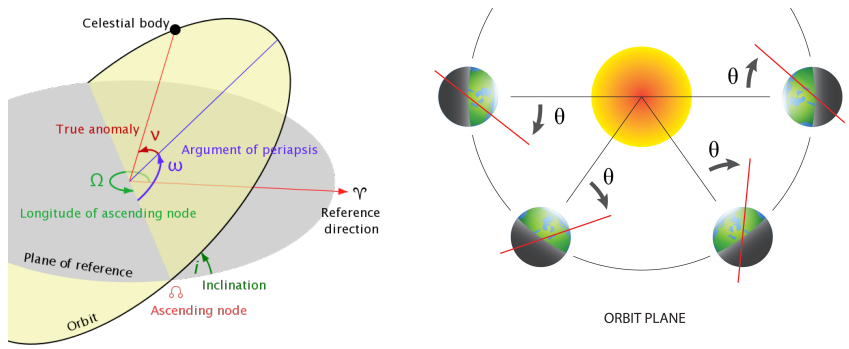

Figure 4.5: (left) Diagram representing the orbital parameters of satellite. Image source: Wikipedia. (right) Diagram representing the orientation of a Sun-synchronous orbit in different points of the year.

The formulation generates the constellation of satellites based of one reference satellite from where the remaining satellites will be defined. This permits a more consistent approach to the search formulation and it is also in line with a realistic launch of multiple satellites. Nonetheless, this requirement could be waived. This reference satellite is in a Sun-synchronous Orbit (SSO) so that each satellite looks at the Earth's surface at the same local mean solar time after each orbit. A Sun-synchronous or heliosynchronous orbit provides an interesting orbital configuration for Earth observation satellites because the orbit ground path retains its local mean solar time, meaning that every time a satellite is looking at the same location on the ground it retains the same lighting conditions (Figure 4.5), not counting seasonal variations.

Constellation Configuration Formulation

The constellation formulation that is presented is a unique aspect of this work. It is formulated after various experiments to create a systematic way to generate a satellite distribution that is not random but it is also not too limited or constrained (as is the case of the simple Satellite Walker Constellation). The various parameters that define the constellation distribution are defined in Table 4.1 and in the following list:

- OP, number of Orbital Planes within a group of satellites.
- SOP, number of Satellites per Orbital Plane: this is the number of satellites to be inserted in each plane of the constellation.
- SG, number of satellite groups where the distribution of satellites will be determined by the two previous parameters.
- ΔRAAN, Right Ascension of the Ascending Node plane separation: the angle that separates each orbital plane within a group.
- ΔMA, Mean Anomaly separation: the angle that separates successive spacecraft within an orbital plane.
- ΔRAAN_GROUP: Right Ascension of the Ascending Node group separation: the angle that separates the first satellites in each cluster of satellites.

Sensor Model

The sensor type used in each satellite of the constellation is rectangular and it models the field of view (FOV) of the instrument on board that takes the pictures. This is a type of push broom sensor that is defined according to specified vertical and horizontal half-angles. These angles are computed in the software given the current altitude of the satellite according to Equations (4.1) and (4.2). The Horizontal Half-Angle (α_H) is represented by the angle from the Z-direction (boresight) to the edge of the sensor in the XZ plane given in the sensor's

coordinate system. Similarly the Vertical Half-angle (α_V) is represented by the angle from the Z-direction (boresight) to the edge of the sensor in the YZ plane given in the sensor's coordinate system. This framework assumes that the satellite to which the sensor is attached has its Z axis pointed toward nadir and X axis pointed to the velocity vector. The diagram in Figure 4.6 represents the structure of the rectangular sensor.

$$\alpha_H = \arctan\left(\frac{\text{swath width}/2}{altitude}\right) \times \left(\tfrac{180}{\pi}\right) \tag{4.1}$$

$$\alpha_V = \arctan\left(\frac{\text{swath height}/2}{altitude}\right) \times \left(\tfrac{180}{\pi}\right) \tag{4.2}$$

4.3.2 Genetic Algorithm for Constellation Optimization

This section describes the use of the GA in this work as a search tool for the satellite constellation optimization. This process encodes the constellation grammar and searches for the individual with the best fitness. The GA starts with a population of chromosomes each one carrying a genotypic content. An agglomerate of chromosomes creates a gene that has the primitive parameters that determine the individuals' layout and topology in the context of the satellite constellation configuration. It is the phenotype that translates the raw information in the genotype to the actual satellite constellation model in physical terms. The evaluation of the phenotype is carried over using the coverage analysis done on the constellation model (the phenotype). The final step of the evaluation is to compute the fitness of the individual. The genetic algorithm then uses the fitness values for the current population and selects the best individuals and it then advances to the next generation by selection, mutation and crossover operators. The genome is the entirety of an individual's hereditary information which includes the genes that define it. In this work the genome is composed of three main classes of genes. These genes affect directly the satellite constellation.

In this study, the genome is a combination of three genes that compose the axiom, the production rules and the geometric-physical parameters that define the constellation to be analyzed. Table 4.1 has a description of the gene phenotype parameters. For this particular project, the constellation generation gene is represented by a random vector of numbers ranging from 0 to 100. The gene then generates the complete phenotype configuration in Equation (4.3).

$$\text{Gene Phenotype} = [\text{OP}; \text{SOP}; \text{SG}; \Delta\text{MA}; \Delta\text{RAAN\_GROUP}; \Delta\text{RAAN}; \text{SW}] \tag{4.3}$$

The GA provides a random search for the satellite constellation configuration and it may appear that the solutions are tailored when indeed they are not. Genetic Algorithms are known to solve non-integer problems in a simple way and also they do not require gradient information for the search. GA s also tend to be slow on the search for the optimal solution. This can be improved using insightful boundaries on the search domain. Some of these bounds can be found in the case studies description of this work. But as an example, the inclination of the orbit can be limited between 80 and 100 deg. when near-polar orbits constellations are desirable. Another improvement that can be made is to limit the scope of the problem and available variables. For complete Earth coverage or even for specific area asymmetric constellations often provide better results. [81, 82]

parameter	genotype	phenotype
# Orbital Planes (OP)	$[0 \ldots 100]$	$[2 \ldots 25]$
# Sat./Orbital Plane (SOP)	$[0 \ldots 100]$	$[1 \ldots 6]$
# Sat. Groups (SG)	$[0 \ldots 100]$	$[1 \ldots 5]$
Δ Mean Anomaly (MA)	$[0 \ldots 100]$	$[0 \ldots 360]^o$
Δ RAAN_GROUP	$[0 \ldots 100]$	$[0 \ldots 360]^o$
Δ RAAN	$[0 \ldots 100]$	$[0 \ldots 10]^o$
Swath Width (SW)	$[0 \ldots 100]$	$[80 \ldots 160]$ km

Table 4.1: Example of a chromosome and its de-codification

Searching for the optimal constellation using a systematic approach is the main objective

for this work. The optimization problem is formulated by the minimization represented in Equation (4.4), applied to the fitness function $f(x)$, Equation (4.5). Where x is the design parameter vector, Ω the parameter space and $f(x)$ the vector objective function. $p(x)$ is the penalization function and λ_i is the Lagrange multiplier that corresponds to the penalization function p_i.

$$optim = min\left[\frac{\text{parameter}}{\text{parameter benchmark}} + \sum_{x\in\Omega} f(x)\right] \tag{4.4}$$

$$f(x) = \lambda_i \times p(x)_i \tag{4.5}$$

The requirements for the ocean coverage constellation have a significant variance in the revisit time for the global ocean: 24, 12 and 4 hrs. This is by itself a complex problem to solve considering the revisit times and focus on the search on water instead of land. By adding the eclipse exclusion constraint (i.e., image only in daylight) the constellation design becomes a more complex problem to solve. Human intuition may help to solve this problem but the search space is so vast (for the constellation configuration) that human intuition alone will not be able to assess all possible situations. For this reason an appropriate computational method must be used for this optimal constellation design search. A constellation generator algorithm was developed using MATLAB and STK. The algorithm generates N satellites in a specific configuration given a reference satellite and the input parameters. The reference satellite was chosen to be inserted in a near Sun-synchronous orbit, being launched from Hawaii. The remaining satellites are launched from the same place but differ in the orbital planes with a delta-Right Ascension of the Ascending (ΔRAAN) node from the reference satellite. The algorithm can place multiple satellites in the same orbital plane. The satellites in the constellation can be separated in groups where each group is separated by another ΔRAAN. The example in Figure 4.6 shows eight satellites separated in two groups. Each group has four satellites and two orbital planes; each orbital plane has two satellites

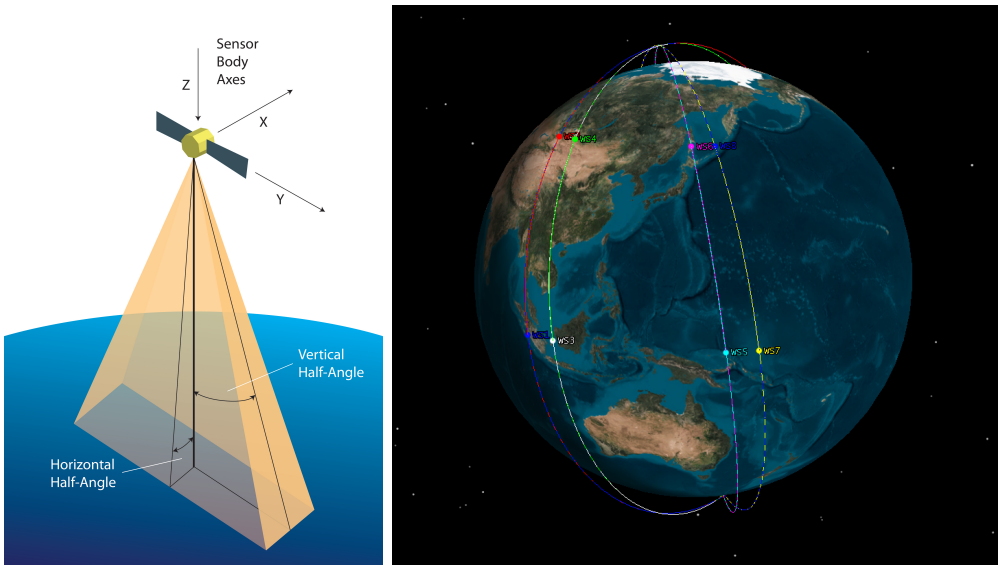

Figure 4.6: (left) Rectangular Sensor Pattern. (right) Example of satellite groups separation in STK.

separated by 45 degrees. Each orbital plane in the same group is separated by 7 degrees.

The search process for the best constellation configuration is implemented using the GA and Direct Search Toolbox in MATLAB. At every iteration a new satellite constellation is generated by interpreting the genotype and then evaluated using the fitness function. For each generation the individual with lower fitness value is kept since its the most optimized satellite constellation. The final step is to compute the fitness value given the parameters for the current constellation. This information is used in the GA to select the best individuals from the population generated. The fitness value takes into account "how good" the given constellation is. Because this is a single objective optimization, where the single objective is to minimize the number of satellites while keeping the constraints, the fitness could be merely the number of satellites. Nevertheless, this would not take into account the other mission requirements like the coverage area and revisit time. It is not necessary to penalize the objective fitness function if the requirements are kept. There is a penalization domain if the satellite constellation configuration is found to be outside the required/desired boundaries.

Finally the fitness value is computed as given by Equation (4.6). The lower this value, the fitter/more optimal the individual is. The penalization is given in Equation (4.7).

$$
\begin{aligned}
fitness = & \frac{\text{\# of satellites}}{\text{desired \# satellites}} \\
& + \lambda_1 \times \text{penalization(desired \# satellites)} \\
& + \lambda_2 \times \text{penalization(desired coverage)} \\
& + \lambda_3 \times \text{penalization(desired revisit time);} \qquad (4.6)
\end{aligned}
$$

$$
penalization = e^{max[0, x_{In} - x_{Lim}]} - 1 - max[0, x_{In} - x_{Lim}] \qquad (4.7)
$$

4.4 Results

This section shows the results of the study made for the earth observation constellations: Case 1) world-wide ocean coverage in 4, 12 and 24 hours revisit-time; and Case 2) North Atlantic Tracks coverage in 1, 2 and 3 hours revisit-time.

4.4.1 Case 1: World-Wide Ocean Coverage

The goal of this study is to obtain satellite constellations that have revisit times of 24hrs, 12hrs, 4hrs in any point of the global oceans. It is important to note that the constellation is attempting to image the Earth's oceans during daytime, and not just a particular point on the Earth. This is by itself a complex problem to solve considering the revisit times and focus on the search on water instead of land. This corresponds to a sunlit ocean coverage time. Since the instruments only operate in daylight, there is a maximum 12 hour window to revisit a given area every 24 hours. By adding the eclipse exclusion constraint the constellation design even becomes a more complex problem to solve. The generated constellations

Orbital Parameter	**Value**	**Obs**
Semi Major Axis	6878.14	Earth R + altitude
Altitude	500 km	-
Eccentricity	0	circular orbit
Inclination	97.4065 deg	Sun synchronous
Arg. Perigee	0	-
RAAN	12.9526 deg	5:00 pm descending
Mean Anomaly	0 deg	-

Table 4.2: Orbital Parameters that generate the reference satellite for the constellation generator.

in this work are made possible to launch from the Pacific Missile Range Facility (PMRF), Hawaii, given some launch azimuth constraints and orbital insertion altitudes limited to 600 km. The reference satellite orbital parameters for this project have been chosen as given in Table 4.2. These parameters guarantee a circular SSO at 500 km of altitude from a PMRF launch.

The constellation optimization solver uses the fitness function given in Equation (4.6). The $\lambda's$ chosen were: $\lambda_1 = 10^{-6}$, $\lambda_2 = 10^{-3}$, $\lambda_3 = 10$. The reason for these numbers lies in the fact that more weight/importance was given to the bad performance of the revisit time parameter. If the given satellite constellation was above the desired revisit time and bellow the desired coverage it would be more penalized than if just below the desired coverage. The same penalization formulation is applied for the excess of number of satellites.

For the 24 hours revisit case with variable swath width the desired number of satellites was set to 60 and the desired percent coverage was set to 100%. For the 12 hours revisit case with fixed swath width of 80 km the desired number of satellites was set to 120 and the desired percent coverage was set to 100%. For the 4 hours revisit case with fixed swath width of 80 km the desired number of satellites was set to 80 and the desired percent coverage was set to 100%.

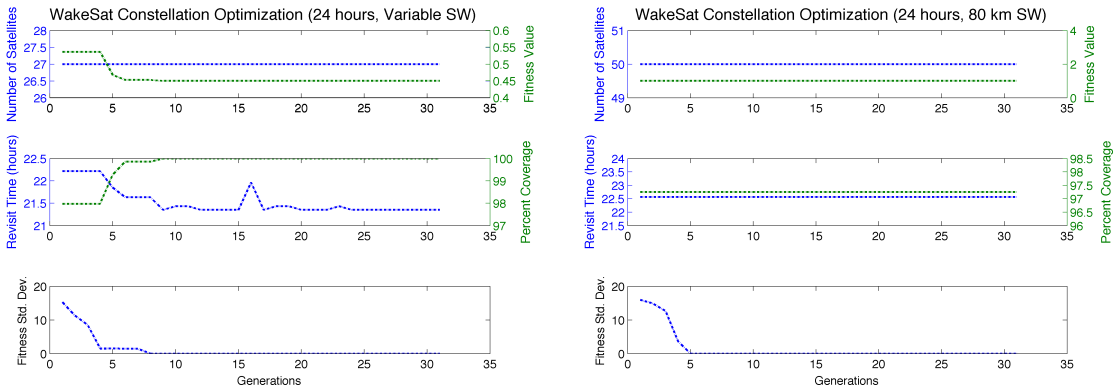

Figure 4.7: GA runs for 24 hours revisit time with *variable* swath width. GA runs for 24 hours revisit time with *fixed* swath width of 80 km (right).

The first optimization case is shown in Figure 4.7 on the left plots. The number of satellites and revisit time is in blue. The fitness and regional percentage coverage are represented by the green lines. The third plot shows the standard deviation for the 10 best individuals in each generation. This run has a variable swath width from 80 km to 160 km for a 24 hours revisit time. At the end of the 30 generations the swath with was 160 km, which is expected because the same sensor provides more coverage. This case holds 27 satellites for a complete global coverage of the oceans in 24 hours. The following case, plots on the right in Figure 4.7, shows the optimization trial for the 24 hours revisit time for a fixed swath width of 80 km. 50 satellites were obtained for this case.

The left side of Figure 4.8 shows the GA optimization runs for the 12 hours revisit time with a variable swath width. The number of satellites and revisit time is shown in blue. The fitness and regional percentage coverage in green. The third plot shows the standard deviation for the 10 best individuals in each generation. Again, at the end of the 30 generations the swath with was again 160 km as expected. This case holds 108 satellites for a complete global coverage of the oceans in 12 hours. Figure 4.8 (right) shows the outcome of the GA

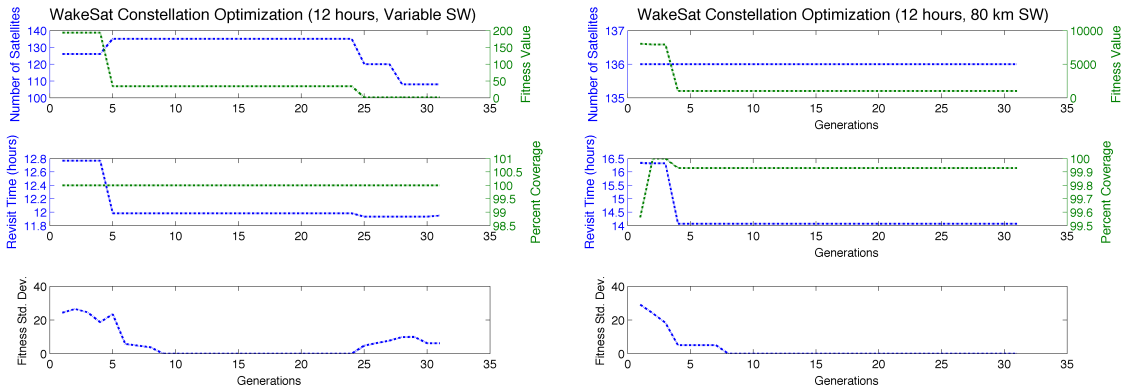

Figure 4.8: GA runs for 12 hours revisit time with *variable* swath width (left). GA runs for 12 hours revisit time with *fixed* swath width of 80 km (right).

for the 12 hours revisit time with a fixed swath width of 80 km. It can be seen that the best number of satellites per population does not change along the generations and the big difference resides in the revisit time that gets closer to 12 hours. In fact it does not reach the 12 hours but stays close to 14 hours. This is the reason why the fitness value is still significantly high since the desirable revisit time is 12 hours.

An empirical formulation that relates the revisit time to the number of satellites is expressed in Equation (4.9). For this purpose a "pareto-front" was drawn in the search space for the 12 and 24 hours revisit time cases analyzed by the GA runs. This "pareto" was obtained from the GA runs and expressed in Figures 4.9 and 4.10.

It is interesting to note that there is a concentration around the 12 hours (red dots) and another in the 24 hours range (blue dots). This was expected because of the requirements imposed by the fitness function that limits the number of hours and the number of satellites.

Finally, for the 4 hours revisit case we note that the groups of satellites must be optimally separated and configured in a way that a segment of the Earth that is covered by

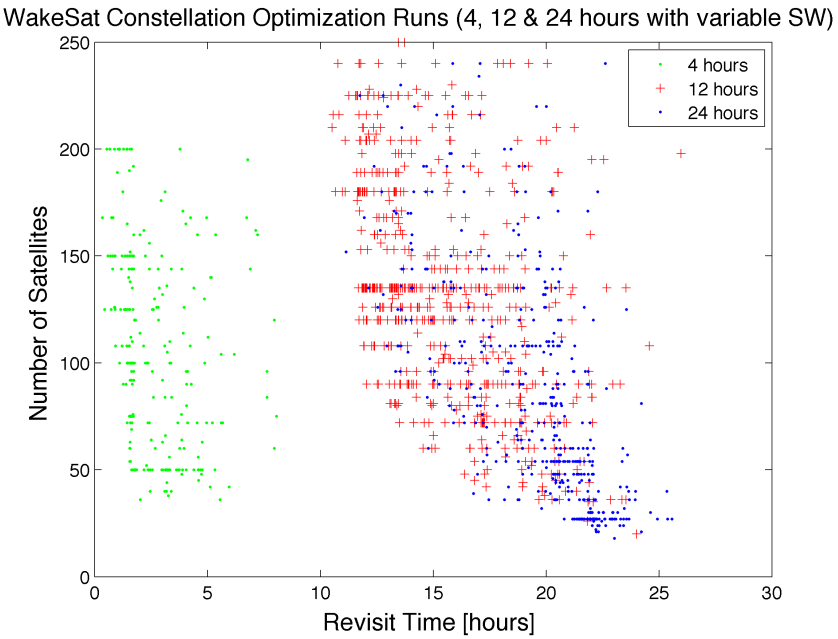

Figure 4.9: Search space of the GA runs for global ocean coverage in 4 (green), 12 (red) and 24 (blue) hours revisit time.

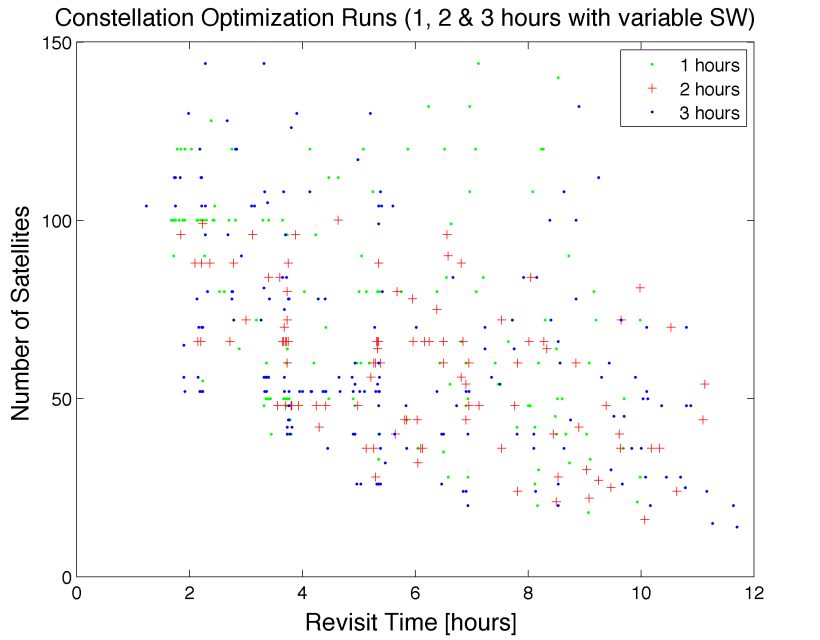

Figure 4.10: Search space of the Genetic Algorithm Runs for North Atlantic Tracks in 1 (green), 2 (red) and 3 (blue) hours revisit time

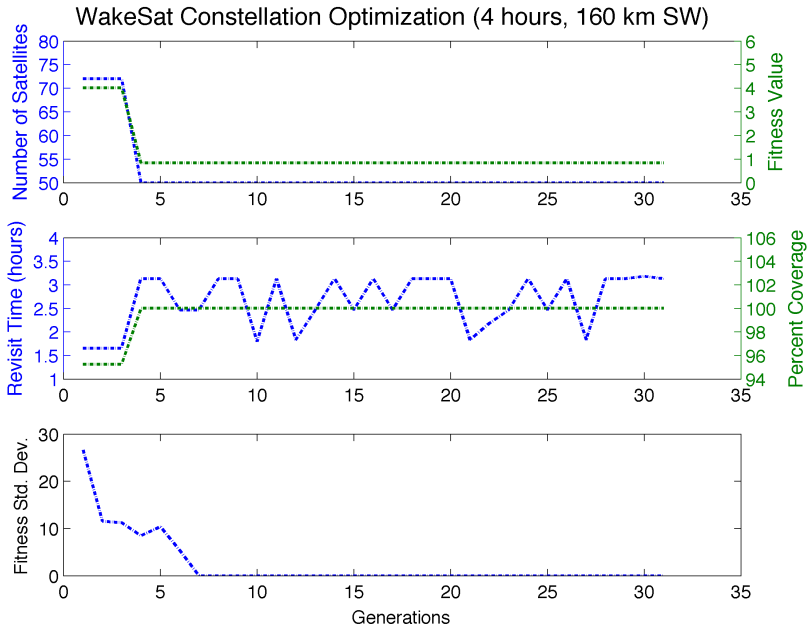

Figure 4.11: Genetic Algorithm runs for 4 hours revisit time with 160 km swath width.

Group 1 will be revisited 4 hours later by Group 2. Figure 4.11 and Figure 4.12 show the optimization results for this case. On Figure 4.11 the number of satellites and revisit time is in blue. The fitness and reginal percentage coverage is in green. The third plot shows the standard deviation for the 10 best individuals in each generation. On Figure 4.12 the blue areas indicate that the area is revisited in 4 hours or less and red indicates that the area is revisited in more than 4 hours. The GA run was somewhat different for the 12 hours and 24 hours revisit time cases because of the moving area target because of the daylight constraint. For this reason the GA run was done over four smaller coverage areas that were spaces inside a 12 hour coverage window. In this way the GA was run over a sample coverage area and a valid result for the global ocean had to be extrapolated. It is for this reason that we find 50 satellites in the GA solution and 80 satellites in the real constellation deployment solution. The limit number of satellites for the 4 hours revisit time was 120. Figure 4.9 shows a clear concentration of green dots in this area.

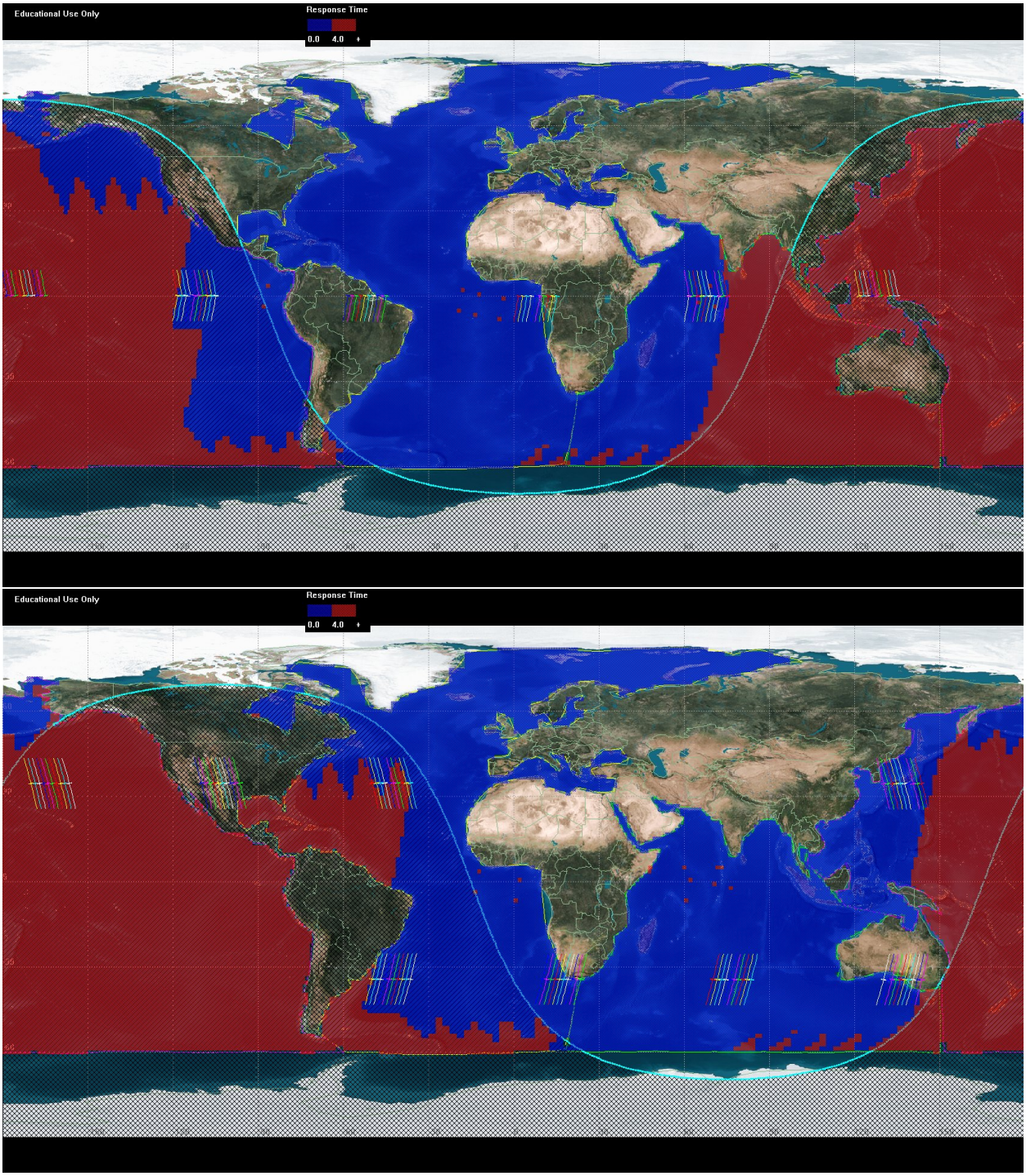

Figure 4.12: Snapshots of the 4 hours optimization constellation at different times.

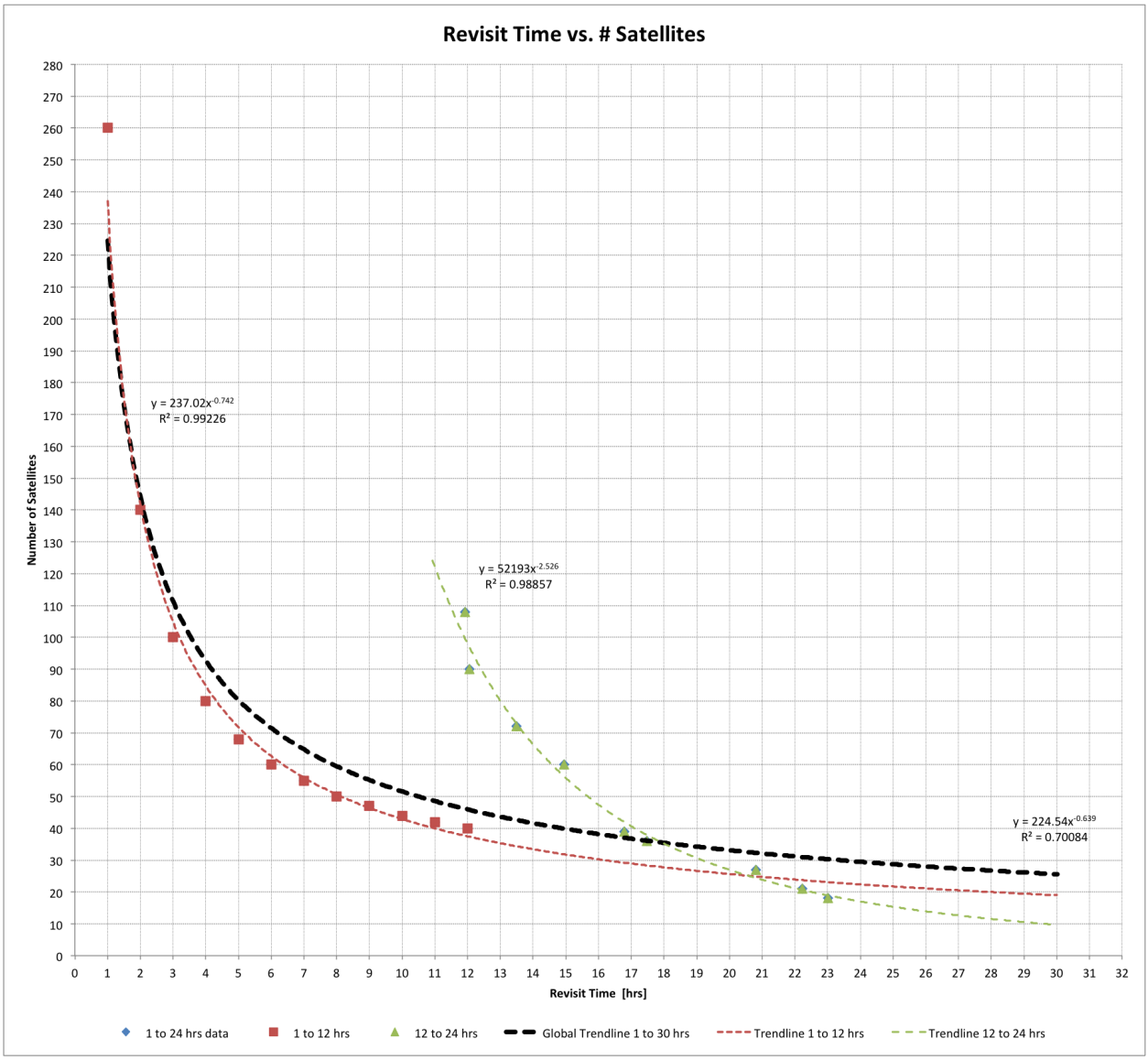

Figure 4.13: 'Pareto front' showing the satellite constellation # for 1 hours to 24 hours revisit time. Extended predictions are for 30 hours.

Equation (4.8) summarizes the prediction of the number of satellites with regard to the desired revisit time when the revisit time is in between 0 and 12 hours. Equation (4.9) summarizes the prediction of the number of satellites when the revisit time is in between 12 and 24 hours.

$$\#\text{ satellites} = \left[\frac{180}{\textbf{revisit time} \times \frac{360}{24}} + 1\right] \times 20 \tag{4.8}$$

$$\#\text{ satellites} = 52193 \times (\textbf{revisit time})^{-2.526} \tag{4.9}$$

Run	Indiv.	Gen.	Run Time	min Fitness	# Satellites
24 hrs 80km SW	50	35	24h	1.0069	50
24 hrs variable SW	50	35	82h	0.4500	27
12 hrs 80km SW	50	35	NA	986.5078	136
12 hrs variable SW	50	35	140h	0.9	108
4 hrs 160 km SW	50	35	50h	0.8333	50

Table 4.3: Different optimization runs with the GA based on the methodology for constellation optimization for the global revisit of the oceans.

4.4.2 Case 2: North Atlantic Tracks Coverage

The second satellite constellation optimization case, North Atlantic Tracks coverage, had a fixed swath width of 160 km. The intent was to revisit the region in 1, 2 and 3 hours of revisit time. Figure 4.14 shows the optimization trial for the 1 and 2 hours revisit time. The number of satellites and revisit time is in blue. The fitness and regional percentage coverage is in green. The third plot shows the fitness standard deviation for the 10 best individuals in each generation. Figure 4.14 shows that the number of satellites for the 1 hour revisit time is at least 100 satellites. For 2 hours revisit-time 66 satellites are necessary for a complete coverage of the North Atlantic Tracks. This is a significant trade-off that must be considered by space mission drivers, increasing the number of satellites from

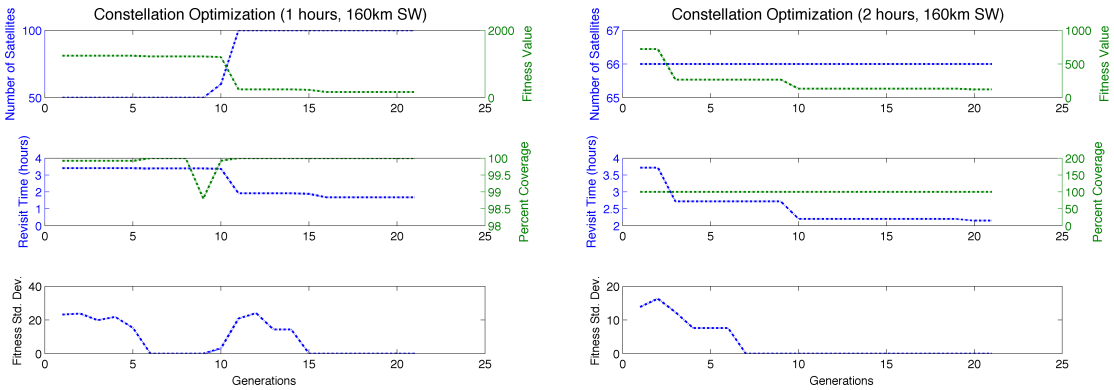

Figure 4.14: GA runs for 1 hour revisit time with fixed swath width of 160 km. GA runs for 2 hours revisit time with fixed swath width of 160 km.

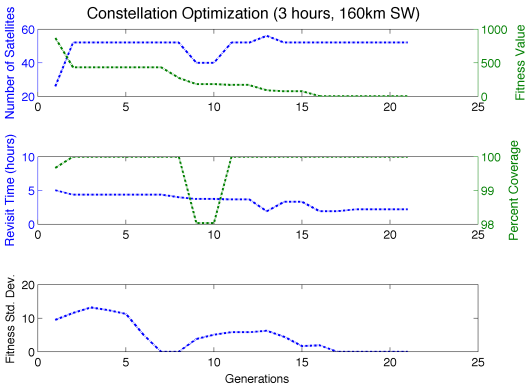

Figure 4.15: GA runs for 3 hour revisit time with fixed swath width of 160 km.

66 to 100 will provide a constellation with a shorter revisit time of 1 hour, instead of 2 hours.

Figure 4.15 shows the outcome of the GA for the 3 hours revisit time with a fixed swath width of 160 km. The number of satellites and revisit time is in blue. The fitness and regional percentage coverage is in green. The third plot shows the fitness standard deviation for the 10 best individuals in each generation. It can be seen that the optimal number of satellites is 52. This number does not change significantly along the generations, but the important factor to be considered is that the revisit time drops from 5 to 3 hours as desired.

Run	Indiv.	Gen.	# Satellites
1 hrs 160km SW	50	35	100
2 hrs 160km SW	50	35	66
3 hrs 160km SW	50	35	52

Table 4.4: Optimization runs for the North Atlantic Track satellite constellation coverage.

4.5 Conclusion

This project leverages a general method to optimize Earth observation satellite constellation taking into account specific design requirements. This method can make possible new space missions that may have never been considered. The work takes a practical approach to the final constellation configuration and provides a benchmark to be used in new ways of satellite constellation optimization. It improves on the empirical results and improves remarkably the "human-inferred-configuration". The results of the benchmarks obtained for the two study cases are summarized in Figure 4.13. This work also explores the optimal design for satellite constellation given some constraints, like the limited availability of a satellite sensor during the illuminated part of the orbit. The solution paradigm is inspired by the constellation generation method and robustness of the GA that searches for optimal satellite configurations.

The number of satellites of the constellations for the given revisit times (24, 12 and 4 hours) was totally unknown and hard to predict before the study of the constellation optimization was done. Satellite constellations predictions were very rough and were estimated empirically with a great amount of variability. For the 24 hours revisit time satellite constellations ranged from 60 to 150 satellites. For 12 hours the number of satellites varied from 100 to 200 satellites. For the 4 hours revisit case no real estimates were done because the extreme number of satellites (more than 200) was not an acceptable figure.

Results are presented for the optimized satellite constellations that revisit every point in

the oceans for complete coverage in every 24 hours, 12 hours and 4 hours. These optimized results are combined into a general equation to provide a benchmark. Another result is presented for the coverage of the North Atlantic Tracks using a constellation of small satellites by revisiting this area in 1, 2 and 3 hours. By providing a satellite imaging system for the North Atlantic Tracks, with a responsive revisit time over the ash clouds, the responsible agencies can alleviate the stresses under the aviation sector by enabling more air-routes. This responsive tracking information could potentially save millions of dollars. From the results of this research it is shown that even a constellation of 100 small satellites could enable a 1 hour revisit time monitoring frequency. It is estimated that up to $300 Million are required to launch a constellation of 100 small satellites. This cost is almost covered with the $250 Million-per-day-loss to the operating airlines during the Eyjafjallajökull eruptions.

The proposed method to generate satellite constellation with multiple constraints was tested and validated, making it possible to create a systematic approach to search for the best satellite constellation configurations. This work enables future Earth observation missions to be more seriously considered.

CHAPTER 5
CONCLUSION

This chapter summarizes the thesis and lists the most important contributions of the work and presents future research directions.

5.1 Summary

The main objective of this thesis is to introduce new methods to enable the development of autonomous and distributed space architectures that are becoming critically important for future robotic space missions. Three specific problems were addressed in this work which include the control and autonomy for the HiakaSat by the development of a new MARS for the satellite ADCS, the demonstration of multiple satellites docking by a newly developed 6DoF GNC architecture also using a MARS, and finally the optimization of a constellation of a large number of satellites to cover large regions of the Earth in a fast revisit time using a newly developed algorithm. A brief summary of each section follows.

In Chapter 2 the MARS is introduced along with the satellite attitude control architecture for the HiakaSat. The satellite is replicated in the laboratory using an engineering model running with the satellite testbed built for testing the HiakaSat system. The exact same software simulation can be uploaded to the satellite testbed and run with the hardware-in-the-loop. The MARS is implemented for the ADCS using COSMOS and deployed on the HiakaSat flight computer connected to the ADCS sensors and actuators. The MARS consists of six independent agents that monitor and control the ADCS and three MTRs that are orthogonally placed in the satellite for control authority. The fact that HiakaSat only uses three MTRs makes it an under-actuated satellite and therefore harder to maneuver because

of the inherent control singularity in the magnetic field direction. This singularity problem is addressed using the control algorithms and numerical results are presented to show that the control system can successfully detumble and point the satellite under the given pointing precision requirement of 5 degrees or better. A typical monolithic ADCS is also implemented and compared with the distributed ADCS (MARS). For computational efficiency it is found that the distributed ADCS has a consistent lower CPU load of 0.29 ± 0.03 compared to 0.35 ± 0.04 for the monolithic implementation, a 17 % reduction. As it is demonstrated by the results, a MARS can achieve equivalent control performance and overcome some of the deficiencies of monolithic systems such as computational load, software system robustness, failure detection and recovery with minimal effects on the system behavior. By using a MARS the foundation of much more complex distributed control structures is in place which can be deployed on a single processor or across multiple ones.

In Chapter 3 a MARS is developed based on COSMOS to demonstrate the autonomous docking for multiple satellites. An essential element of the cooperative control for satellites is the use of agents that are the connecting element between each satellite for managing the autonomy and communicate the state information with other satellites. The MARS is developed using the same elements for the flight software making this implementation a step closer to a realistic implementation for future possible missions. Based on the E-Guidance method, a new 6DoF guidance method is introduced as part of a real-time 6DoF GNC framework for docking multiple fully actuated satellites. The 6DoF GNC computes in real time the controls inputs for the coupled motion in position and attitude of fully actuated spacecraft considering a proper rigid body dynamics and orbital mechanics. A case study is presented for ARD of four cubesats deployed from a typical launcher. The simulated docking maneuvers are completed within 25 minutes, which is approximately 30% of a full orbital period in LEO.

In Chapter 4 a general method to optimize Earth observation satellite constellations is introduced by taking into account specific design requirements such as swath width, orbital parameters and area target revisit times. The work takes a practical approach to the final constellation configuration and provides a benchmark to be used in new ways of satellite constellation optimization. This work also explores the optimal design for satellite constellation given constraints, like the limited availability of a satellite sensor during the illuminated part of the orbit. Results are presented for two case studies, the first is the coverage with fast revisit time, 24 hours or less, of the Earth's oceans. The second is the coverage of the North Atlantic Tracks with fast revisit time, 3 hours or less. The first case results show that for a swath width of 80 km a number of 50 satellites is required to cover the oceans in a 24 hour revisit time. For a swath width of 160 km the same number of satellites can cover the whole region in 4 hours. For the second case the results show that for a fixed swath width of 160 km. For the 1 hour revisit time at least 100 satellites are required. For the 2 hours revisit-time 66 satellites are necessary for a complete coverage and for the 3 hour revisit time 52 satellites are required.

In conclusion, small satellites ($\leq$ 180 kg) are being used as a catalyst for advanced space missions. Because of the accelerated use of small satellites, combined with recent advances in space technology, more missions are being proposed that push the boundaries of what has been the standard practice in the space industry. This dissertation in particular addresses the need for a high level of autonomy and coordination for large numbers of satellites. A significant effort was dedicated to the development of a MARS, and the experimental validation on a satellite testbed showed that this is a viable solution for the operation of satellites that require a high level of autonomy. This system is flexible and scalable to enable autonomous operations of one satellite or multiple satellites. In the future it is expected that MARS can be used to autonomously manage large satellite constellations such as the mega-

satellite constellation currently being proposed. This work also showed that the configuration of satellite constellations can be significantly optimized. With 100 satellites or less, a very capable constellation can be launched to revisit large portions of the Earth in just a few hours.

5.2 Contributions

A list of the most important contributions of this work are listed below by major sections:

Chapter 2

- a new distributed software architecture, a MARS, was developed for a satellite ADCS to control and improve autonomy for a spacecraft. The MARS was tested on the HiakaSat satellite testbed to verify if the implementation meets the control requirements.
- the MARS was compared with a monolithic implementation of the ADCS to show that the control performance is equivalent. Therefore using MARS for ADCS is a viable solution.
- the operation of the satellite ADCS was successfully demonstrated on the 3DoF air bearing testbed validating the typical deployment sequence of HiakaSat, detumble maneuver and pointing maneuvers for entering nominal operation modes.
- a "pseudo-ECI" propagator was developed to propagate the orbital state of the satellite after deployment in case the GPS does not acquire a lock.

Specific Project contributions

- the complete ADCS for HiakaSat was designed and developed. This includes the selection of sensors and actuators, the software drivers as well as the development of the estimation and control software for the different modes of operation.

- a satellite testbed was designed and developed for testing the HiakaSat engineering model in a representative environment. Strong focus was given to make the testbed represent space as realistically as possible.
- as part of the development of the MARS many COSMOS core libraries were created and/or improved. This expanded COSMOS and also validated its applicability as a flight software, testbed software and mission operations software.
- developed a new graphics user interface (GUI) tool "Satellite Attitude Viewer 3D" to run the simulation and visualize the attitude of the satellites.
- developed a new software tool to simulate the satellite attitude "Satellite Engine".

Chapter 3

- derived the 3DoF E-Guidance for position for ARD using the Hill's accelerating frame.
- derived the 3DoF E-Guidance for attitude using a quaternion based representation.
- developed a new 6DoF guidance method for position and attitude by extending the existing new 3DoF E-Guidance derivations.
- developed a 6DoF GNC architecture using the new 6DoF guidance for satellite ARD.
- developed a MARS using the 6DoF GNC algorithms for ARD for docking multiple satellites.
- introduced a new experimental mission scenario for docking four CubeSats deployed radially by a launcher, in a typical deployment configuration.
- the resulting software is made into a new GNC software module for COSMOS.

Specific Project contributions

- developed a new GUI tool "3D Docking Viewer" to run the simulation, visualize and analyze the docking of multiple satellites.

Chapter 4

- developed a new algorithm to generate satellite constellation configurations.
- developed a new generic and flexible satellite constellation optimization method based on GA using MATLAB and STK.
- obtained benchmark results for an optimal satellite constellation for fast revisit time of the global Ocean, in 24 hours or less.
- obtained benchmark results for an optimal satellites constellation for fast revisit time to cover the North Atlantic Tracks, in 3 hours or less.
- created an empirical benchmark curve for constellations with a large number of satellite to be used in future design studies.

5.3 Future work

The following points highlight future research aspects to carry the present work forward.

Chapter 2

This work started to explore the potential in MARS, but it only touched the subject in a superficial level. One of the developments that was initialized, but not completed due to time constraints, was the definition of a quantitative method to evaluate a MAS. This was motivated by the fact that in the MAS literature the narrative is always qualitative when trying to define MASs. By transforming the narrative to a more quantitative way using the

proposed MARS Quotient (see Appendix D) then not only different MAS can be compared but also a better understanding is gained about a specific MAS. For this particular work of a MARS for ADCS it would simply mean to implement the Agent Quotient for all the six agents and analyze the system in light of the results.

The work can also be further expanded by adding different attitude controllers and estimators within the MARS, and by implementing learning algorithms the system could learn what type of algorithm combination is best for different modes and mission scenarios. This evaluation can be quantified using the proposed MARS Quotient while deploying the software on the onboard computer and running a complete mission scenario with realistic inputs. Since the MARS is embedded in a dynamic environment and each agent is interacting with one or more agents valid (or optimal) policies must be found dynamically as well. Policies such as data transfer frequency or what control algorithm must be used are just two examples that can be applicable to the problem of attitude control.

To extend the usability of the satellite testbed platform it would be important to develop a wireless power mechanism to enable continuous operations of the testbed. One of the problems found while performing the MARS characterization tests is that the testbed battery would drain in approximately four hours. This limits the operations testing for long periods of time as it would be necessary to fully validate the satellite system autonomy.

Chapter 3

Chapter 3 introduced a new MARS and a new GNC method for docking various satellites. Future work should focus on looking at the same mechanism to dock a larger number of satellites in the order of dozens, hundreds and thousands. Also different under-actuated control architectures should be investigated to look into the applicability of such docking algorithms

using small satellites and find more efficient ways to maneuver the satellites. Finally looking into more sophisticated collision avoidance mechanisms for performing safe maneuver while docking multiple CubeSats might be a very interesting research effort.

Chapter 4

In Chapter 4 a method for constellation optimization is presented. This work can be further expanded by optimizing not only for a region of the globe (spacial coverage) but also for a periodic revisit to collect data along different times of the day (temporal coverage). Something to explore is to develop an equivalent optimization algorithm but instead of using a GA based optimization use simulated annealing for large search spaces of multi-objective, multi-constrained problems since these methods tend to be faster at finding the solution compared to typical GA.

Also a long term stability study for the satellite constellation should be done to evaluate the required thrust/energy of each satellite to maintain its orbit configuration, specially to understand the J2 effects, and what type of on-board propulsion system should be used to maintain the configuration. Part of this effort should be to analyze the sensitivity of the different parameters in the gene pool and gain insights into new optimization gains. One other related study is to find satellite constellation configurations that degrade in such a way that the overall mission requirements are not significantly affected for long periods of time (months, years). Finding such solutions would simplify and reduce the cost of the overall mission because no propulsion would be required.

Finally, expanding the MARS developed in the previous chapters and applying it to a satellite constellation management mission would be very important to demonstrate the scalability and applicability of such concepts. A number of satellite constellation missions can be

realistically simulated in a computer cluster, to create a satellite constellation simulator, and therefore test the MARS to autonomously operate large constellations in different mission scenarios.

APPENDIX A
LIST OF ACRONYMS

3DoF three degrees of freedom.

6DoF six degrees of freedom.

AAReST Autonomous Assembly of a Reconfigurable Space Telescope.

ADC attitude determination and control.

ADCS attitude determination and control subsystem.

AI Artificial Intelligence.

ARAX Autonomous Remote Agent Experiment.

ARD autonomous rendezvous and docking.

CEO COSMOS Executive Operator.

COSMOS Comprehensive Open-architecture Solution for Mission Operations Systems.

COTS commercial off-the-shelf.

CPOD Cubesat Proximity Operations Demonstration.

CRS complex robotic spacecraft.

DARPA Defense Advanced Research Projects Agency.

ECEF earth centered earth Fixed.

ECI earth centered inertial.

EKF Extended Kalman Filter.

FDIR failure detection isolation and recovery.

FOV field of view.

FSW flight software.

GA genetic algorithm.

GCRF Geocentric Celestial Reference Frame.

GNC guidance, navigation and control.

GPS Global Positioning System.

GUI graphics user interface.

HIL hardware-in-the-loop.

HSFL Hawaii Space Flight Laboratory.

ICRF International Celestial Reference Frame.

IMU inertial measurement unit.

ISS International Space Station.

JAXA Japan Aerospace Exploration Agency.

JPL Jet Propulsion Laboratory.

JSON JavaScript Object Notation.

LEO low earth orbit.

LOC lines of code.

LVLH local-horizontal local-vertical.

MARS multi-agent robotic system.

MAS multi-agent system.

MASQ multi-agent system quotient.

MATLAB Mathworks MATLAB®.

MOST Mission Operations Support Tool.

MTR magnetic torque rod.

NASA National Aeronautics and Space Administration.

OBC on-board computer.

RHESE Radiation Hardened Electronics for Space Environments.

SOH state of health.

SPHERES Synchronized Position Hold Engage and Reorient Experimental Satellite.

ST star tracker.

STK Systems Tool Kit.

SUCHI Space Ultra Compact Hyperspectral Imager.

TCU torque-rod control unit.

TTFF time to first fix.

WMM World Magnetic Model.

APPENDIX B
NOMENCLATURE

$\boldsymbol{x}$	=	state vector
$\boldsymbol{r}$	=	linear position vector $(x, y, z) \in \mathbb{R}^3$ (m)
$\dot{\boldsymbol{r}}$	=	linear velocity vector $(\dot{x}, \dot{y}, \dot{z}) \in \mathbb{R}^3$ (m/s)
$\ddot{\boldsymbol{r}}$	=	linear acceleration vector $(\dot{x}, \dot{y}, \dot{z}) \in \mathbb{R}^3$ (m/s)
$\boldsymbol{q}$	=	angular position/attitude vector in quaternion space $[q_x, q_y, q_z, q_w] \in \mathbb{R}^4$
$\boldsymbol{\omega}$	=	angular velocity vector $(\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ (rad/s)
$\boldsymbol{\alpha}$	=	angular acceleration vector $(\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ (rad/s^2)
m	=	satellite mass
$\mathbb{I}$	=	inertia tensor $diag(I_x, I_y, I_z)$ (kg.m^2)
$\boldsymbol{f}_{ctr}$	=	control forces $\in \mathbb{R}^3$ (N)
$\boldsymbol{f}_{ext}$	=	external forces (solar radiation, atmospheric drag, etc.) $\in \mathbb{R}^3$ (N)
$\boldsymbol{\tau}_{ctr}$	=	control torque $\in \mathbb{R}^3$ (N.m)
$\boldsymbol{\tau}_{ext}$	=	external torques (solar radiation torque, gravity gradient, etc.) $\in \mathbb{R}^3$ (N.m)
ω_o	=	orbit angular rate ≥ 0 (rad/s)
μ_{Earth}	=	Earth gravitational parameter 398600.4415 km^3/s^2
r_{sat}	=	satellite orbit radius
h	=	orbit altitude
v	=	orbit velocity
P	=	orbit period
i	=	orbit inclination
e	=	orbit eccentricity
$\boldsymbol{\tau}_g$	=	gravity gradient

F_B	=	body frame
F_O	=	orbital frame = LVLH
F_I	=	inertial frame
$R_{B \leftarrow A}$	=	rotation matrix, transform vector in frame A to frame B $\in SO(3)$
M	=	MTR dipole moment
B	=	magnetic field of Earth
$\boldsymbol{g}$	=	gravitational acceleration vector (m/s^2)
$\boldsymbol{a}_T$	=	thrust acceleration vector (m/s^2)
T_{go}	=	time to go (s)
$C(c_{ij})$	=	C-Matrix for E-Guidance
$E(e_{ij})$	=	E-Matrix for E-Guidance
P	=	P-vector for E-Guidance

APPENDIX C
RESULTS FOR MAS ADCS VS. MONOLITHIC ADCS

C.1 Monolithic runs

The following figures show the results for the Monolithic tests running for 2000 seconds. The purpose was to determine the average CPU load during steady state and also to compare the control performance.

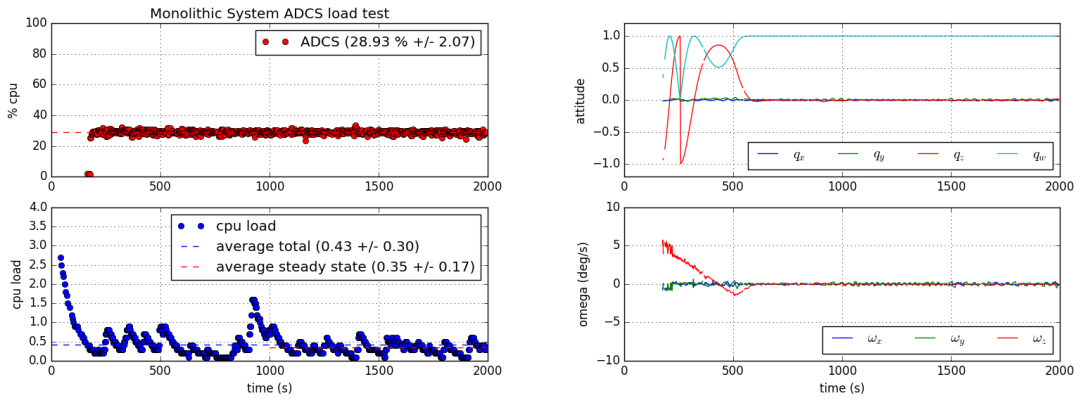

Figure C.1: Run 1. Performance plots for detumble and point maneuver for the Monolithic architecture.

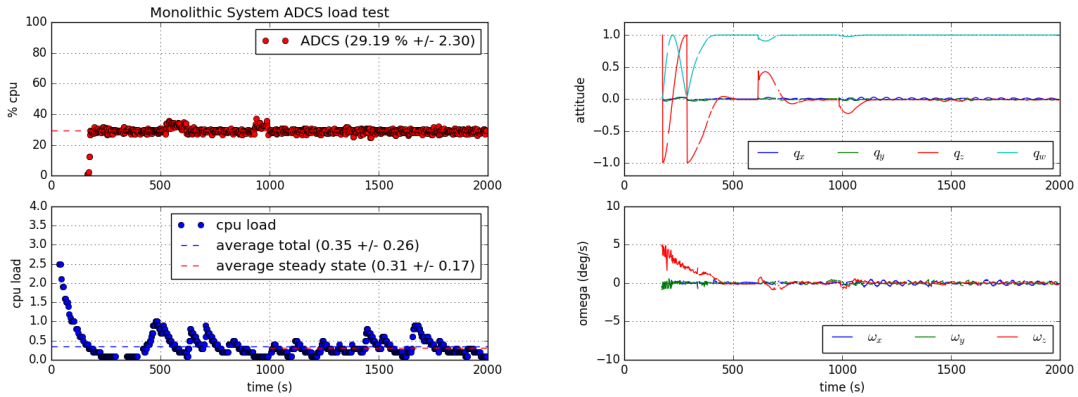

Figure C.2: Run 2. Performance plots for detumble and point maneuver for the Monolithic architecture.

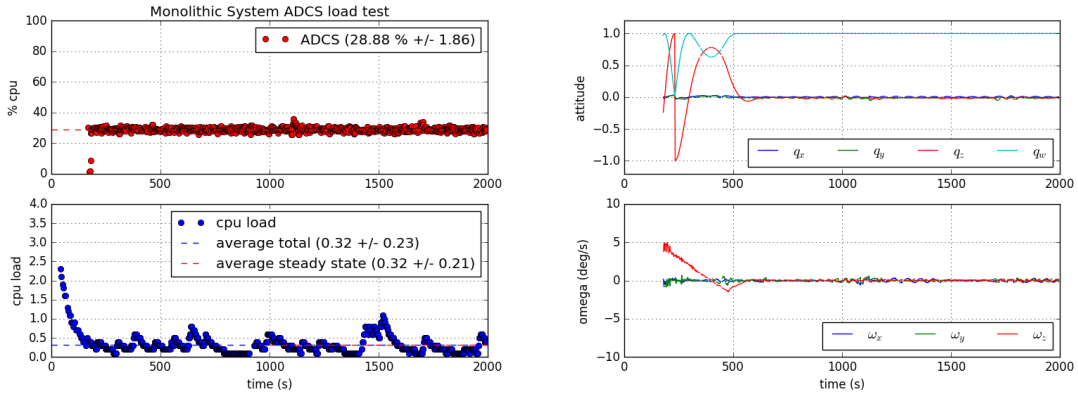

Figure C.3: Run 3. Performance plots for detumble and point maneuver for the Monolithic architecture.

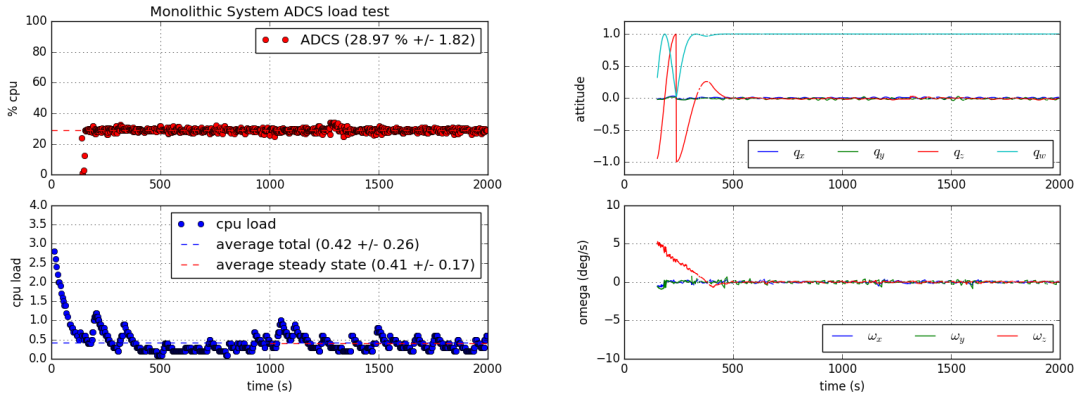

Figure C.4: Run 4. Performance plots for detumble and point maneuver for the Monolithic architecture.

C.2 Multi-Agent System runs

The following figures show the results for the MAS tests running for 2000 seconds. The purpose was to determine the average CPU load during steady state and also to compare the control performance.

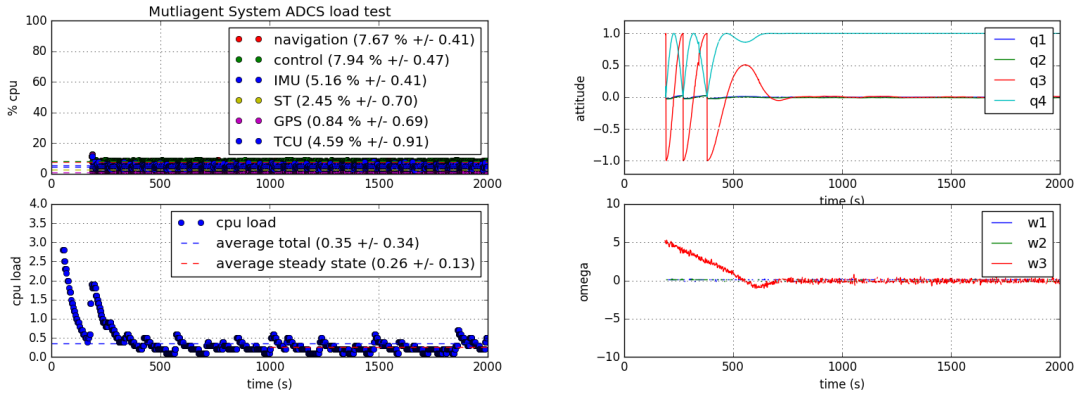

Figure C.5: Run 1. Performance plots for detumble and point maneuver for the MAS architecture.

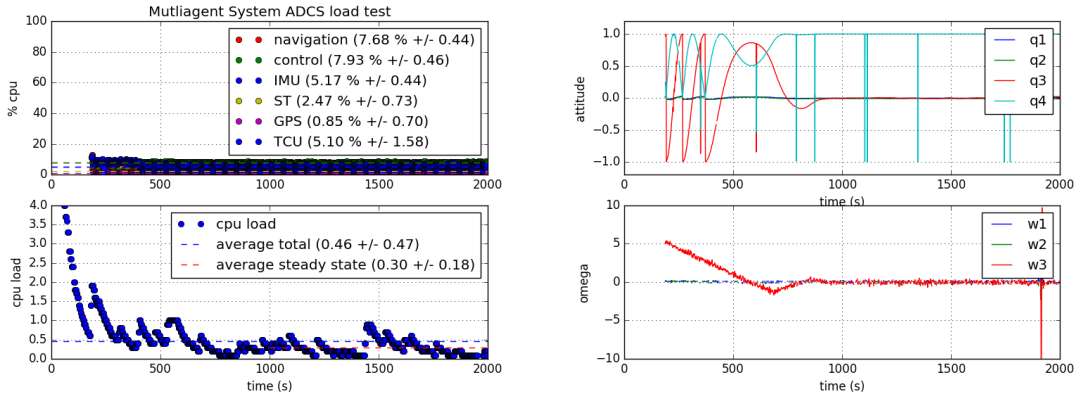

Figure C.6: Run 2. Performance plots for detumble and point maneuver for the MAS architecture.

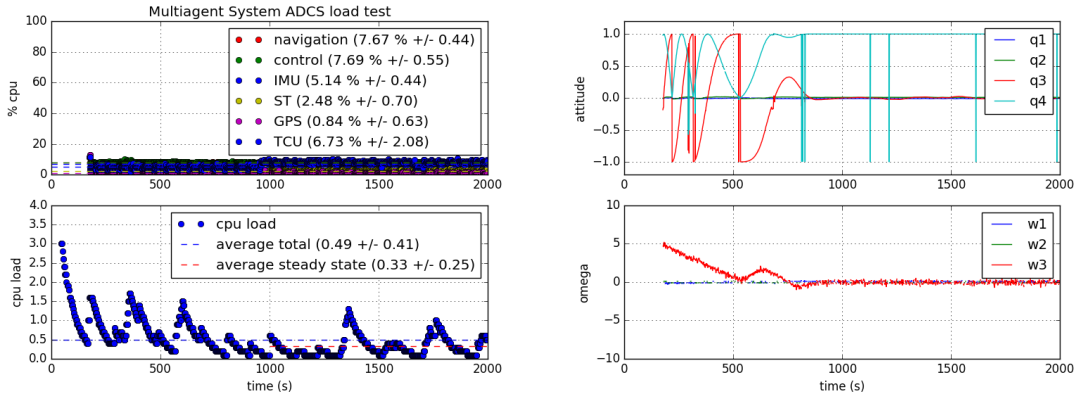

Figure C.7: Run 3. Performance plots for detumble and point maneuver for the MAS architecture.

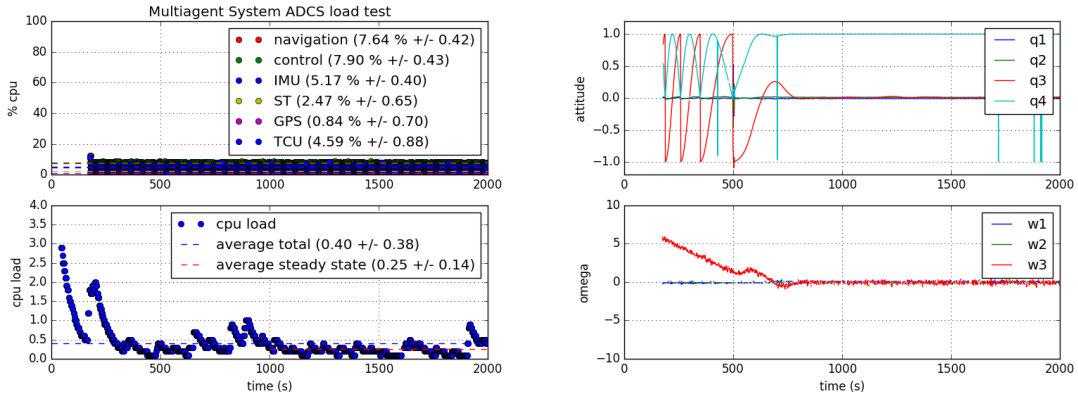

Figure C.8: Run 4. Performance plots for detumble and point maneuver for the MAS architecture.

APPENDIX D
AGENT QUOTIENT

Much of the literature about agents and MASs deals with the concepts in a very generic way, not very precise and totally adapted to the community to which it is presented. Though this is sufficient for the most part, the more agents concepts exist and the more cross-disciplinary they become the more important it will be to precisely define if a system can be truly an agent or MAS or only a hybrid or special form such as traditional control system. For this reason we propose an agent quotient (AQ), a metric that can evaluate an agent in the design phase and also during run time. This metric can set benchmarks to determine if something that is called an agent can be truly be referred to an agent and to which degree. Extrapolating the idea we also propose the concept of a multi-agent system quotient (MASQ) that combines all AQs from the different agents and provides a metric for the whole system. A MASQ can also be extremely useful to determine the performance of a given system and compare that measure against the expected/designed MASQ.

A perfect agent will have a quotient of 1 (or 100 %) and similarly a perfect MAS will also have a quotient of 1 (or 100 %). In reality this is never possible but the metric can for example determine the threshold of when a running system transitions form a simple system (AQ < 0.5) to a more complex one (AQ > 0.5). We propose that a system is an agent system when the levels of autonomy, sociability, intelligence and sagacity are positive. The higher these metrics are the more sophisticated the agent system is. In this section we present a tentative formulation to evaluate the AQ and MASQ for a running system.

Typical control systems tend to focus on the following essential aspects: responsiveness (Re), controllability/observability(Co) and robustness(Ro). For the agent system we propose

to keep the essential aspects of the control system, but add a few more that will enclose a better representation for a more sophisticated system: autonomy(Au), sociability(So), intelligence(In) and finally sagacity(Sa). These aspects draw inspiration from true human agents. In fact, for a human to be considered sagacious or wise, all these aspects must be active. Each of the aspects is going to be transformed into metrics ranging from 0 to 1 (or 0 to 100%), and the mean metric will result in the agent quotient as given by the following equation:

$$\mathbf{A}_Q = \frac{Re + Co + Ro + Au + So + In + Sa}{7} \tag{D.1}$$

Similarly for a MAS quotient we have

$$\mathbf{MAS}_Q = \frac{\sum_{i=1}^{N} w_i A_{Q_i}}{\sum_{i=1}^{N} w_i} \tag{D.2}$$

where i represents an individual agent and w_i is the designed weight of the agent in the system.

The following is a brief description of each metric along with the equivalent mathematical form.

1. Responsiveness. This is the most essential aspect for a control system or agent system. If it is not responsive then nothing will work. This metric measures the latency of the agent which may depend of the specific application but typically it will be how fast it executes the main thread against the expected latency. The second parameter is the load that the agent is putting on the computer system against the expected load. The last two metrics are the number of messages sent (#Tx) and received (#Rx) compared again the expected values for

each

$$Re = \frac{Latency}{E(Latency)}\bigg|_{\leq 1} /f + \frac{Load}{E(Load)}\bigg|_{\leq 1} /f + \frac{\#Tx}{E(\#Tx)}\bigg|_{\leq 1} /f + \frac{\#Rx}{E(\#Rx)}\bigg|_{\leq 1} /f, f = 4 \quad \text{(D.3)}$$

2. Controllability. This is one of the most essential aspects for a control system or agent system. This property infers the stability of the system against a desired control goal. It also is a dual for observability. These two metrics will have to be defined for each problem and should be normalized (the total result is less or equal to one). In terms of the agent quotient we expand the definition of controllability. This metric can measure the stability and observability of a system, the number of control goals/requirements achieved by the system and the number of successful observations:

$$\begin{aligned} Co \ &= Controllability_{control}\bigg|_{\leq 1} /f + Observability_{control}\bigg|_{\leq 1} /f + \\ &\frac{\#Goals_{achieved}}{E(\#Goals)}\bigg|_{\leq 1} /f + \frac{\#Observations_{achieved}}{E(\#Observations)}\bigg|_{\leq 1} /f, f = 4 \end{aligned} \quad \text{(D.4)}$$

3. Robustness. This metric measures the resilience of a system against unexpected disturbances. It can be computed by counting the number of times a system recovered from a disturbance or a failure over a long period of time. This can also be the number of times a system came off the modeled bounds and came back to it's stable course, this is quantifying the stability of the system:

$$Ro = \frac{\#recoveries}{timeperiod}\bigg|_{\leq 1} + stability\bigg|_{\leq 1} \quad \text{(D.5)}$$

4. Autonomy. This metric measures the autonomy of a system by computing the number of commands extra received per time period (T) to change the course of operations.

The system is assumed to be fully autonomous at the beginning of the operations and will progressively trend to the real autonomy level:

$$Au = \left.\frac{E(\#commands)/T}{\#commands/T}\right|_{\leq 1} \tag{D.6}$$

5. Sociability. This metric measures the social aspect of a system, how 'well" it behaves in the presence of others. To compute this metric we define the sociability as the ratio of messages received (#Rx) by the number of messages sent (#Tx) against the expected values of each parameter:

$$So = \left.\frac{\frac{\#Rx}{\#E(\#Rx)}}{\frac{\#Tx}{\#E(\#Tx)}}\right|_{\leq 1} \tag{D.7}$$

6. Intelligence. This metric measures the number of rules that a system has or is learning over time. The set of rules can be a simple database or a complex mechanism of inference but the essence of number of rules/actions learned is universal:

$$In = \left.\frac{\#Rules}{\#E(\#Rules)}\right|_{\leq 1} + \left.\frac{\#Ruleslearned}{\#E(\#Ruleslearned)}\right|_{\leq 1} \tag{D.8}$$

7. Sagacity. For an agent system to be fully complete it must have some ability from the previous aspects but also must be able to use the rules (intelligence) it has collected. This metric measures the number of rules that a system has used or is using over time:

$$In = \left.\frac{\#Rulesused}{\#E(\#Rulesused)}\right|_{\leq 1} \tag{D.9}$$

APPENDIX E
6DOF E-GUIDANCE DERIVATION

At some instant $t = t_0$ the navigation agent provides the guidance agent of the current state of the spacecraft (position $\boldsymbol{r}$, velocity $\boldsymbol{v}$, attitude $\boldsymbol{q}$, attitude rate $\boldsymbol{\omega}$)

$$\begin{aligned}
\boldsymbol{r}(t_0) &= [x(t_0) \quad y(t_0) \quad z(t_0)] \\
\boldsymbol{v}(t_0) &= [\dot{x}(t_0) \quad \dot{y}(t_0) \quad \dot{z}(t_0)] \\
\boldsymbol{q}(t_0) &= [q_x(t_0) \quad q_y(t_0) \quad q_z(t_0) \quad q_w(t_0)] \\
\boldsymbol{\omega}(t_0) &= [\omega_x(t_0) \quad \omega_y(t_0) \quad \omega_z(t_0)]
\end{aligned} \tag{E.1}$$

and the desired target state is

$$\begin{aligned}
\boldsymbol{r}(T) &= [x_T \quad y_T \quad z_T] \\
\boldsymbol{v}(T) &= [\dot{x}_T \quad \dot{y}_T \quad \dot{z}_T] \\
\boldsymbol{q}(T) &= [q_{x_T} \quad q_{y_T} \quad q_{z_T} \quad q_{w_T}] \\
\boldsymbol{\omega}(T) &= [\omega_{x_T} \quad \omega_{y_T} \quad \omega_{z_T}]
\end{aligned} \tag{E.2}$$

and typically the final linear and angular velocities are set to zero

$$\begin{aligned}
\boldsymbol{v}(T) &= [0 \quad 0 \quad 0] \\
\boldsymbol{\omega}(T) &= [0 \quad 0 \quad 0]
\end{aligned} \tag{E.3}$$

The guidance law will command the spacecraft to maneuver from the current boundary conditions (E.1) to the desired boundary conditions (E.2). The problem in essence is to compute the linear and angular acceleration (force and torque) given the rendezvous two-point boundary-value conditions for $t_o \leq t \leq T$ such that at $t = T$ the spacecraft meets it's target state. The energy spent on this maneuver can be evaluated by $\int_{t_0}^{T} \sqrt{\boldsymbol{a}_T.\boldsymbol{a}_T} dt$ and this

parameter is to be minimized. While in orbit the spacecraft motion is bound to the following differential equations

$$\frac{\mathrm{d}^2\boldsymbol{r}}{\mathrm{d}\boldsymbol{t}^2} = \boldsymbol{g} + \boldsymbol{a}_T \tag{E.4}$$

which can be expanded assuming a spherical gravity field

$$\begin{aligned}
\ddot{x} &= -\mu\frac{x}{(x^2+y^2+z^2)^{3/2}} + a_{x_T} \\
\ddot{y} &= -\mu\frac{y}{(x^2+y^2+z^2)^{3/2}} + a_{y_T} \\
\ddot{z} &= -\mu\frac{z}{(x^2+y^2+z^2)^{3/2}} + a_{z_T}
\end{aligned} \tag{E.5}$$

By integrating the total acceleration from current time $t = t_0$ to a generic time t the equations is

$$\dot{x}(t) - \dot{x}(t_0) = \int_{t_0}^{t} \ddot{x}(s)ds \tag{E.6}$$

or equivalently

$$\dot{x}_T - \dot{x}_0 = \int_{t_0}^{T} \ddot{x}(s)ds \tag{E.7}$$

and integrating both sides

$$x_T - x_0 - \dot{x}(t_0)T_{go} = \int_{t_0}^{T} \left[\int_{t_0}^{t} \ddot{x}(s)ds\right] dt \tag{E.8}$$

where $T_{go} = T - t_0$. The two integral equations on the acceleration term will be transformed into a pair of simultaneous linear algebraic equations with two unknowns. There is a infinite number of possible solutions but one can limit the factors and propose the following form for the solution as

$$\ddot{x}(t) = c_1 p_1(t) + c_2 p_2(t) \tag{E.9}$$

where p_1 and p_2 are linearly independent and c_1 and c_2 are coefficients chosen to satisfy Eq. E.7 and E.8. Substituting the equations now is

$$\dot{x}_T - \dot{x}_0 = f_{11} c_1 + f_{12} c_2 \tag{E.10}$$

$$x_T - x_0 - \dot{x}(t_0) T_{go} = f_{21} c_1 + f_{22} c_2 \tag{E.11}$$

where

$$f_{11} = \int_{t_0}^{T} p_1(t) dt \tag{E.12}$$

$$f_{12} = \int_{t_0}^{T} p_2(t) dt \tag{E.13}$$

$$f_{21} = \int_{t_0}^{T} \left[\int_{t_0}^{t} p_1(s) ds \right] dt \tag{E.14}$$

$$f_{22} = \int_{t_0}^{T} \left[\int_{t_0}^{t} p_2(s) ds \right] dt \tag{E.15}$$

if p_1 and p_2 are integrable then the f functions are algebraic and the solution for c_1 and c_2 is found by solving the equation

$$\begin{bmatrix} \dot{x}_T - \dot{x}_0 \\ x_T - (x_0 + \dot{x}_0 T_{go}) \end{bmatrix} = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \tag{E.16}$$

or equivalently

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} \begin{bmatrix} \dot{x}_T - \dot{x}_0 \\ x_T - (x_0 + \dot{x}_0 T_{go}) \end{bmatrix} \tag{E.17}$$

The E-matrix is the inverse of the F-matrix

$$e_{11} = f_{22}/\Delta \tag{E.18}$$

$$e_{12} = -f_{12}/\Delta \tag{E.19}$$

$$e_{21} = -f_{21}/\Delta \tag{E.20}$$

$$e_{22} = f_{11}/\Delta \tag{E.21}$$

and $\Delta = f_{11}f_{22} - f_{12}f_{21}$. Now assuming $p_1(t) = 1$ and $p_2(t) = T - t$ and the E-matrix for this particular solution is

$$E = \begin{bmatrix} 4.0/T_{go} & -6.0/T_{go}^2 \\ -6.0/T_{go}^2 & 12.0/T_{go}^3 \end{bmatrix}. \tag{E.22}$$

so then c_1 and c_2 are given by

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 4.0/T_{go} & -6.0/T_{go}^2 \\ -6.0/T_{go}^2 & 12.0/T_{go}^3 \end{bmatrix} \begin{bmatrix} \dot{x}_T - \dot{x}_0 \\ x_T - (x_0 + \dot{x}_0 T_{go}) \end{bmatrix} \tag{E.23}$$

and the solution for the dynamics is finally

$$\ddot{x}(t) = c_1 + c_2(T - t) \tag{E.24}$$

APPENDIX F
C++11 FOR FLIGHT SOFTWARE AND SIMULATION

In this section we list the reasons that lead us to choose C++ as the main language do develop the simulation software for this project. We have considered the use of other modern programming languages such as Python, but none other was as flexible, compatible and stable as C++. With C++ we can reuse other libraries developed by the HSFL for the COSMOS project and also use part of the simulation code and port it directly into the embedded processors for small satellites. It also gives us the flexibility to make the software work seamlessly in Linux, Mac OS and Windows.

C++ is a very flexible object oriented programming language with optimal low level memory management manipulation features. It is one of the fastest programming languages and more complete in functional terms. Its abstraction capabilities make possible the creation of very systematic, organized and reusable code. C++ is designed with a bias for systems programming such as embedded systems, that in the case for this project where the target embedded system are spacecraft processors.

C++11 is the revision of the C++ programming language released in 2011. It replaced C++03 that is still prominent in the software development industry. The latest versions of modern compilers such as GCC, Clang and Intel C++ have full support for C++11, while others like MSVC have almost full support. Even embedded architectures using ARM processors are supporting the new modern C++ features. One of the major downsides of the C++ programming language that is usually referred is the steep learning curve, nonetheless with the new C++11 standard and so many learning resources readily available the trend is

changing in favor of this powerful programming language.

One of the major problems when using C/C++ is that the mathematical frameworks for matrix operations and numerical integration is not easily available. While this is true at first glance, there are however some readily available C++ libraries that are peer reviewed, open source and free, fast and optimized. For this project we use Eigen for matrix operations (`http://eigen.tuxfamily.org/`. Eigen is an open source high-level C++ library mostly for linear algebra, matrix and vector operations. For numerical integration we use Odeint (`http://headmyshoulder.github.io/odeint-v2/`), a modern C++ library for numerically solving Ordinary Differential Equations.

With complex simulation scenarios it is important to be able to visualize the data generated and interact with it, with C++ we can directly access graphics libraries for visualization of the simulation data such as OpenGl and Qt.

OpenGL is the Open Graphics Library, a multi-platform application programming interface (API) for rendering high performance 2D and 3D vector graphics. The OpenGL API interacts with a graphics processing unit (GPU), to achieve hardware-accelerated rendering making the visualization process independent of the simulation, leaving the main processor mostly for the simulation only.

Qt is a cross-platform application framework used mostly for graphic window creation but also with extended functionality over many other Operating System functions. It has a user friendly API for OpenGL making readily available to use in any Qt application. Finally we use QCustomPlot (`http://www.qcustomplot.com/`), that is a Qt/C++ based library for plotting and data visualization.

APPENDIX G
SATELLITE CONSTELLATIONS SUMMARY TABLE

Satellite Constellation	# Sat.	revisit	orbit type	resolution	swath width	weight	mission objective
COMPASS (Beidou-2)	30	NA	MEO	NA	NA	NA	Navigation
GLONASS	24	NA	64.8 deg, (20,000 km)	NA	NA	1,260 kg	Navigation
Global Positioning System	30	NA	CIRC 55 deg	NA	20,200 km	NA	Navigation
Galileo	30	NA	MEO (23 222km) 56 deg	NA	NA	675 kg	Navigation
Disaster Monitoring Constellation	5	24 hours	SSO (636km)	30-40 m	600 km	NA	Earth observations
A-train	7	N/A	SSO (690km)	NA	NA	NA	Earth observations (HD)
RapidEye	5	3-4 days	SSO (630 km)	5 meter	77 km	150 kg	Earth observations
(Envisat)	1	35 days	SSO (790km)	30 1 km	50- 400 km	8.5 T	Earth observations
(Landsat 7)	1	16 days	SSO (705km)	15 - 60 m	185 km	1973 kg	Earth observations
Broadband Global Area Network	3	NA	GEO (35,786 km)	NA	NA	NA	Communications
Globalstar	48	NA	52 deg	NA	1400 km	550 kg	Communications
Iridium satellite constellation	66	NA	86.4 deg (781 km)	NA	NA	680 kg	Communications
Molniya	16	NA	63 deg, (450 x 600 km)	NA	NA	NA	Communications
ORBCOMM	29	NA	775 km	NA	NA	42 kg	Communications
Sirius Satellite Radio	NA	NA	GEO	NA	NA	NA	Communications
Tracking and Data Relay Satellite	9	NA	GEO	NA	NA	NA	Communications
Teledesic	840	NA	NA	NA	NA	NA	Communications
COMMStellation	78	NA	1,000 km	NA	NA	NA	Communications

Table G.1: Satellite Constellations Summary Table

APPENDIX H
SATELLITE ATTITUDE TOOL 3D

The Satellite Attitude Tool 3D (SAT3D) was developed as part of this work to run ADCS simulations and visualize the attitude of the satellite. The tool runs on Windows, Mac OS and Linux.

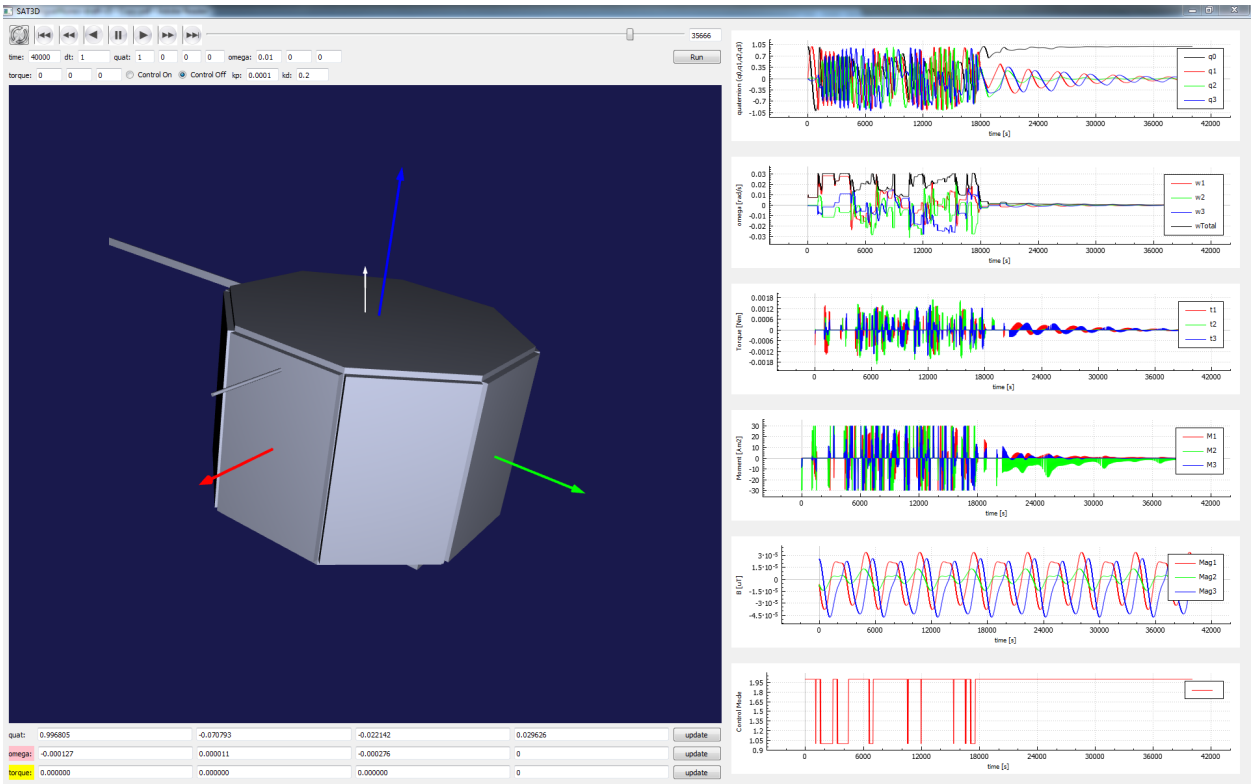

Figure H.1: Satellite Attitude Tool 3D

BIBLIOGRAPHY

[1] National Aeronautics and Space Administration, "NASA Technology Roadmaps TA 4 : Robotics and Autonomous Systems," Tech. Rep. May, National Aeronautics and Space Administration, 2015.

[2] O. Brown and P. Eremenko, "The Value Proposition for Fractionated Space Architectures," in *AIAA Space 2006.*, no. September, (San Jose, CA), pp. 1–22, 2006.

[3] O. Brown, P. Eremenko, and M. Bille, "Fractionated Space Architectures: Tracing the Path to Reality," *23rd Annual AIAA/USU Conference on Small Satellites*, pp. 1–10, 2009.

[4] C. Underwood, S. Pellegrino, V. J. Lappas, C. P. Bridges, and J. Baker, "Using CubeSat/micro-satellite technology to demonstrate the Autonomous Assembly of a Reconfigurable Space Telescope (AAReST)," *Acta Astronautica*, vol. 114, pp. 112–122, sep 2015.

[5] J. C. Mankins, "SPS - ALPHA: The First Practical Solar Power Satellite via Arbitrarily Large Phased Array (A 2011 - 2012 NASA NIAC Phase 1 Project)," tech. rep., 2012.

[6] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT press, 1999.

[7] D. V. Dimarogonas and K. J. Kyriakopoulos, "Formation control and collision avoidance for multi-agent systems and a connection between formation infeasibility and flocking behavior," *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference, CDC-ECC '05*, vol. 2005, no. 1, pp. 84–89, 2005.

[8] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Upper Saddle River, New Jersey: Pearson, third ed., 2009.

[9] S. Franklin, S. Franklin, A. Graesser, and A. Graesser, "Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents," *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pp. 21–35, 1997.

[10] D. Bernard, G. Dorais, C. Fry, J. Gamble, E.B., B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, and B. Williams, "Design of the Remote Agent experiment for spacecraft autonomy," *1998 IEEE Aerospace Conference Proceedings (Cat. No.98TH8339)*, vol. 2, 1998.

[11] C. García, P. F. Cárdenas, L. J. Puglisi, and R. Saltaren, "Design and modeling of the multi-agent robotic system: SMART," *Robotics and Autonomous Systems*, vol. 60, pp. 143–153, feb 2012.

[12] G. Kaminka, "Robots are agents, too!," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07*, no. December, p. 1, 2007.

[13] K. H. Movric and F. L. Lewis, "Cooperative optimal control for multi-agent systems on directed graph topologies," *IEEE Transactions on Automatic Control*, vol. 59, no. 3, pp. 769–774, 2014.

[14] J. R. Wertz, *Spacecraft Attitude Determination and Control.* D. Reidel, 1978 editi ed., 1978.

[15] T. Bak, R. Wisniewski, and M. Blanke, "Autonomous attitude determination and control system for the Orsted satellite," in *1996 IEEE Aerospace Applications Conference. Proceedings*, vol. 2, pp. 173–186, IEEE, 1996.

[16] M. Pavone and J. Starek, "Spacecraft Autonomy Challenges for Next Generation Space Missions," *Springer Lecture Notes in Control and Information Sciences*, pp. 1–34, 2014.

[17] R. Sanchez-Maestro, a. Agenjo-Diaz, and a. Cropp, "PROBA-3 Formation Flying High Performance Control," *5th International Conference on Spacecraft Formation Flying Missions and Technologies*, no. 1, pp. 1–15, 2013.

[18] W. Cai, X. Liao, and D. Y. Song, "Indirect Robust Adaptive Fault -Tolerant Control for Attitude Tracking of Spacecraft," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 5, pp. 1456–1463, 2008.

[19] R. Ginosar, "Survey of Processors For Space," *Data Systems in Aerospace (DASIA). Eurospace*, pp. 1–5, 2012.

[20] T. C. Sorensen, D. C. Oswald, R. M. Shook, and J. Van Gaasbeck, "Spacecraft Autonomous Operations Experiment Performed During the Clementine Lunar Mission," *Journal of Spacecraft and Rockets*, vol. 32, pp. 1049–1053, nov 1995.

[21] J. Riedel, S. Bhaskaran, S. Desai, and D. Han, "Using Autonomous Navigation for Interplanetary Missions: Mission Operations with Deep Space 1 Autonav," *Space*, pp. 1–13, 2000.

[22] N. Muscettola, P. Nayak, B. Pell, and B. C. Williams, "Remote Agent: to boldly go where no AI system has gone before," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 5–47, 1998.

[23] T. C. Sorensen, C. V. Hude, M. H. Kobayashi, E. J. Pilger, A. K. Sanyal, L. K. Yoneshige, and M. A. Nunes, "LEO-1: Development of a University Microsatellite for Flight Testing New Technologies," in *AIAA SPACE 2009 Conference & Exposition*, (Reston, Virigina), p. 1, American Institute of Aeronautics and Astronautics, sep 2009.

[24] T. Sorensen, L. French, W. Doi, J. Chan, E. Gregory, M. Kobyashi, Z. Lee-Ho, M. Nunes, E. Pilger, A. Yamura, and L. Yoneshige, "Hawai'iSat-1: Development Of A University Microsatellite For Testing a Thermal Hyperspectral Imager," in *AIAA SPACE 2010 Conference & Exposition*, (Reston, Virigina), p. 1, American Institute of Aeronautics and Astronautics, aug 2010.

[25] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey:: Pearson, second ed., 2003.

[26] H. S. Nwana, "Software agents: an overview," *The Knowledge Engineering Review*, vol. 11, no. 03, p. 205, 1996.

[27] P. Iigo-Blasco, F. Diaz-Del-Rio, M. C. Romero-Ternero, D. Cagigas-Muiz, and S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development," *Robotics and Autonomous Systems*, vol. 60, no. 6, pp. 803–821, 2012.

[28] G. Wen, *Distributed cooperative control for multi-agent systems*. Phd, Ecole Centrale de Lille, 2013.

[29] S. D. J. McArthur, E. M. Davidson, V. M. Catterson, a. L. Dimeas, N. D. Hatziargyriou, F. Ponci, and T. Funabashi, "Multi-Agent Systems for Power Engineering Applications - Part I: Concepts, Approaches, and Technical Challenges," *Power Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 1743–1752, 2007.

[30] G. Weiss, *Multiagent Systems*. The MIT Press, 2nd editio ed., 2013.

[31] M. Software, "A.I. driven character animation for Film."

[32] F. Castanedo, J. Garcia, M. A. Patricio, and J. M. Molina, "A multi-agent architecture to support active fusion in a visual sensor network," in *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*, pp. 1–8, IEEE, sep 2008.

[33] G. Di Fatta and G. Fortino, "A customizable multi-agent system for distributed data mining," in *Proceedings of the 2007 ACM symposium on Applied computing - SAC '07*, (New York, New York, USA), p. 42, ACM Press, 2007.

[34] K. Stohlmann and D. Lüdtke, "OBC-NG : Towards a Reconfigurable On-board Computing Architecture for Spacecraft," 2014.

[35] A. S. Keys, M. D. Watson, D. O. Frazier, J. H. Adams, M. A. Johnson, and E. A. Kolawa, "High Performance, Radiation-Hardened Electronics for Space Environments," in *International Planetary Probes Workshop 5*, no. June, (Bordeaux, France), NASA, 2007.

[36] A. Elkady and T. Sobh, "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography," *Journal of Robotics*, vol. 2012, pp. 1–15, 2012.

[37] C. Schlegel, T. Hassler, a. Lotz, and a. Steck, "Robotic software systems: From code-driven to model-driven designs," *2009 International Conference on Advanced Robotics*, 2009.

[38] T. C. Sorensen, E. J. Pilger, M. S. Wood, M. A. Nunes, and B. D. Yost, "Development of a Comprehensive Mission Operations System Designed to Operate Multiple Small Satellites," in *AIAA/USU Small Satellite Conference, Logan, UT. #SC11-IX-3*, p. 1, 2011.

[39] T. C. Sorensen, E. J. Pilger, M. S. Wood, and M. A. Nunes, "A University-developed Comprehensive Open-architecture Space Mission Operations System (COSMOS) to Operate Multiple Space Vehicles.," in *SpaceOps 2012 Conference. Cleared for public release by NASA Ames*, p. 1, 2012.

[40] M. A. Nunes, T. C. Sorensen, and E. J. Pilger, "On the development of a 6DoF GNC framework for docking multiple small satellites," *AIAA Guidance, Navigation, and Control Conference, AIAA Science and Technology Forum*, pp. 1–14, 2015.

[41] S. T. Crites, P. G. Lucey, R. Wright, J. Chan, H. Garbeil, K. A. Horton, A. Imai, M. Wood, and L. Yoneshige, "SUCHI: The Space Ultra-Compact Hyperspectral Imager for small satellites," p. 873902, may 2013.

[42] W. Graham, "Super Strypi conducts inaugural launch Fails during first stage."

[43] M. A. Nunes, T. C. Sorensen, E. J. Pilger, H. M. Garbeil, J. R. Lewis, D. M. Azimov, and M. Wood, "Expanding the Comprehensive Open-architecture Space Mission Operations System (COSMOS) for Integrated Guidance, Navigation and Control of Multiple Small Satellites," in *13th International Conference on Space Operations 2014*, (Reston, Virginia), p. 1, American Institute of Aeronautics and Astronautics, may 2014.

[44] C. P. Bridges and T. Vladimirova, "Real-time agent middleware experiments on java-based processors towards distributed satellite systems," *IEEE Aerospace Conference Proceedings*, 2011.

[45] A. P. M. Chiaradia, H. K. Kuga, and d. A. P. Antonio Fernando Bertachini, "Onboard and Real-Time Artificial Satellite Orbit Determination Using GPS," *Mathematical Problems in Engineering*, vol. 2013, no. 3-4, pp. 1–8, 2013.

[46] D. A. Vallado, *Fundamentals of Astrodynamics and Applications*. Microcosm Press, 4th ed., 2013.

[47] M. M. Berry and L. M. Healy, "Implementation of Gauss-Jackson Integration for Orbit Propagation," *The Journal of the Astronautical Sciences*, vol. 52, no. 3, pp. 331–357, 2004.

[48] C. Lafleur, "Spacecraft Encyclopedia, Spacecraft (Known) Failures."

[49] NovAtel, "OEMV® Family Firmware Reference Manual, OM-20000094 Rev 8," tech. rep., 2010.

[50] E. G. Lightsey, "A GPS Receiver Adapts to Space," *GPS World*, no. June 2000, 2001.

[51] R. K. Lopes da Silva, "Coarse Orbit Determination as a Dividend from Attitude Sensors for Spin Stabilized Satellites ," *J. Brazilian Society of Mechanical Sciences, XXI, special issue, 1999, pp. 346-355]*, vol. 1999, 1999.

[52] J. R. Vetter, "Fifty years of orbit determination: Development of modern astrodynamics methods," *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, vol. 27, no. 3, pp. 239–252, 2007.

[53] S. Spangelo, A. Klesh, and J. Cutler, "Position and Time System for the RAX Small Satellite Mission," no. August, pp. 1–17, 2010.

[54] D. F. Lawden, *Optimal Trajectories for Space Navigation.* Butterworths, 1963.

[55] A. Farhad, "Optimal control of a space manipulator for detumbling of a target satellite," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3019–3024, 2009.

[56] B. C. Raschke, S. Roemer, and K. Grossekatthoefer, "Test bed for verification of attitude control system," 1993.

[57] W. Fehse, *Automated Rendezvous and Docking of Spacecraft.* Cambridge: Cambridge University Press, 2003.

[58] Y. Roux and P. Da Cunha, "The GNC Measurement System for the Automated Transfer Vehicle," *International Symposium on Space Flight Dynamics (ISSFD)*, vol. 1, no. 1, 2004.

[59] J. Bowen, A. Tsuda, J. Abel, and M. Villa, "CubeSat Proximity Operations Demonstration (CPOD) mission update," in *2015 IEEE Aerospace Conference*, pp. 1–8, IEEE, mar 2015.

[60] NASA-AMES Mission Design Division Staff, "Small Spacecraft Technology State of the Art," Tech. Rep. February, Ames Research Center, Moffett Field, California, 2014.

[61] S. B. McCamish, M. Romano, S. Nolet, C. M. Edwards, and D. W. Miller, "Flight Testing of Multiple-Spacecraft Control on SPHERES During Close-Proximity Operations," *Journal of Spacecraft and Rockets*, vol. 46, pp. 1202–1213, nov 2009.

[62] NASA Press Release, "Cubesat Proximity Operations Demonstration (CPOD)."

[63] P. Melroy, L. Hill, E. E. Fowler, R. Hunter, J. Eagen, B. R. Sullivan, P. Will, and J. Palmer, "DARPA Phoenix Satlets: Progress towards Satellite Cellularization," in *AIAA SPACE 2014 Conference and Exposition*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, aug 2015.

[64] M. Kobilarov and S. Pellegrino, "Trajectory Planning for CubeSat Short-Time-Scale Proximity Operations," *Journal of Guidance, Control, and Dynamics*, vol. 37, pp. 566–579, mar 2014.

[65] J. M. Phillips, L. E. Kavraki, and N. Bedrosian, "Spacecraft Rendezvous and Docking With Real-Time Randomized Optimization," in *AIAA Guidance, Navigation and Control Conference*, 2003.

[66] G. Boyarko, O. Yakimenko, and M. Romano, "Optimal Rendezvous Trajectories of a Controlled Spacecraft and a Tumbling Object," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 4, pp. 1239–1252, 2011.

[67] S. B. McCamish, M. Romano, and X. Yun, "Autonomous distributed control of simultaneous multiple spacecraft proximity maneuvers," *IEEE Transactions on Automation Science and Engineering*, vol. 7, pp. 630–644, 2010.

[68] G. W. Cherry, "E-Guidance - a General Explitit Optimizing Guidance Law for Rocket - Propelled Spacecraft, Apollo Guidance and Navigation," tech. rep., MIT Instrumentation Lab, 1964.

[69] K. Xie, Z. Yang, Z. Huang, and Y. Zhou, "LEO Space-Based Radar Constellation Design Using a Genetic Algorithm," *Journal of Electronics*, no. I, pp. 1–4, 2006.

[70] C. Zehner, "Monitoring volcanic ash from space," Tech. Rep. May, 2012.

[71] L. Vargo, "Orbital Patterns for Satellite Systems," *Advances in the Astronautical Sciences*, vol. 6, pp. 709–725, 1960.

[72] R. Luders, "Satellite Networks for Continuous Zonal Coverage," *American Rocket Society Journal*, vol. 31, pp. 179–184, 1961.

[73] J. Walker, "Some Circular Orbit Patterns Providing Continuous Whole Earth Coverage," *Journal of the British Interplanetary Society*, vol. 24, pp. 369–384, 1971.

[74] A. D. Biria and B. G. Marchand, "Constellation Design for Space-Based Situational Awareness Applications: An Analytical Approach," in *AAS/AIAA Astrodynamics Specialists Conference*, (Girdwood, AK), pp. 1–20, 2011.

[75] D. J. Pegher and J. A. Parish, *Optimizing Coverage And Revisit Time In Sparse Military Satellite Constellations: A Comparison Of Traditional Approaches And Genetic Algorithms*. Master of science thesis, Naval Postgraduate School, may 2004.

[76] T. A. Ely, W. A. Crossley, and E. A. Williams, "Satellite constellation design for zonal coverage using genetic algorithms," *Journal Of The Astronautical Sciences*, vol. 47, no. 3-4, pp. 207–228, 1999.

[77] T. J. Lang, "Walker Constellations to Minimize Revisit Time in Low Earth Orbit," *AAS/AIAA Space Flight Mechanics Meeting*, vol. AAS 03-178, 2003.

[78] M. R. Bussieck and A. Pruessner, "Mixed-Integer Nonlinear Programming," *http://www.gamsworld.eu/minlp/siagopt.pdf*, 2003.

[79] T. J. Lang, "Symmetric circular orbit satellite constellations for continuous global coverage," *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference, Kalispell, MT, Aug. 10-13,*, pp. 1111–1132, 1987.

[80] A. T. Takano and B. G. Marchand, "Optimal constellation design for space based situational awareness applications," in *AAS/AIAA Astrodynamics Specialists Conference*, (Girdwood, AK), 2011.

[81] E. A. Williams, W. A. Crossley, and T. J. Lang, "Average and maximum revisit time trade studies for satellite constellations using a multiobjective Genetic Algorithm," *Journal of the Astronautical Sciences*, vol. 49, no. 3, pp. 385–400, 2001.

[82] T. J. Lang and W. S. Adams, "A Comparison of Satellite Constellations for Continuous Global Coverage," *International Workshop on Mission Design and Implementation of Satellite Constellations, France, November*, 1997.