DISTRIBUTED CONTROL OF SERVICING SATELLITE FLEET USING HORIZON SIMULATION FRAMEWORK

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Aerospace Engineering

by

Scott Plantenga

June 2023

COMMITTEE MEMBERSHIP

TITLE: Distributed Control of Servicing Satellite Fleet Using Horizon Simulation Framework

AUTHOR: Scott Plantenga

DATE SUBMITTED: June 2023

COMMITTEE CHAIR: Eric Mehiel, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Kira Jorgensen Abercromby, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Leonardo A. B. Torres, Ph.D.
Professor of Aerospace Engineering

COMMITTEE MEMBER: Stathis Charalampidis, Ph.D.
Professor of Mathematics

ABSTRACT

Distributed Control of Servicing Satellite Fleet Using Horizon Simulation Framework

Scott Plantenga

On-orbit satellite servicing is critical to maximizing space utilization and sustainability and is of growing interest for commercial, civil, and defense applications. Reliance on astronauts or anchored robotic arms for the servicing of next-generation large, complex space structures operating beyond Low Earth Orbit is impractical. Substantial literature has investigated the mission design and analysis of robotic servicing missions that utilize a single servicing satellite to approach and service a single target satellite. This motivates the present research to investigate a fleet of servicing satellites performing several operations for a large, central space structure.

This research leverages a distributed control approach, implemented using the Horizon Simulation Framework (HSF), to develop a tool capable of integrated mission modeling and task scheduling for a servicing satellite fleet. HSF is a modeling and simulation framework for verification of system level requirements with an emphasis on state representations, modularity, and event scheduling. HSF consists of two major modules: the main scheduling algorithm and the system model. The distributed control architecture allocates processing and decision making for this multi-agent cooperative control problem across multiple subsystem models and the main HSF scheduling algorithm itself. Models were implemented with a special emphasis on the dynamics, control, trajectory constraints, and trajectory optimization for the servicing satellite fleet.

The integrated mission modeling and scheduling tool was applied to a sample scenario in which a fleet of 3 servicing assets is tasked with performing 12 servicing activities for a large satellite in Geostationary Orbit. The tool was able to successfully determine a

schedule in which all 12 servicing activities were completed in under 32 hours, subject to the fuel and trajectory constraints of the servicing assets.

ACKNOWLEDGMENTS

Thanks to:

- My parents, for their support and encouragement through all my life
- My wife, Danica, for her unwavering love and support
- My advisor, Dr. Eric Mehiel, for his mentorship and guidance through this journey and my time as an undergraduate

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Satellite servicing has previously been employed to successfully repair and refuel target satellites using both human and robotic servicing techniques. However, such missions have relied on astronauts, anchored robotic arms, or limited usage of a single servicing satellite operating on a single target satellite. Substantial literature exists researching the dynamics, control, mission architecture, and servicing satellite design for a single servicing satellite to approach, rendezvous with, and operate on a single target satellite. Research exists exploring utilization of a fleet of cooperative satellites in formation flight, but not the application of such a fleet to satellite servicing missions. The present research investigates the effectiveness of a fleet of cooperative servicing satellites performing multiple operations for a single target satellite.

1.1 Motivation for Robotic Satellite Servicing

Satellite servicing performed via human Extravehicular Activity (EVA) has been demonstrated with the Hubble Space Telescope (HST) and the International Space Station (ISS) [4]. However, utilizing human EVAs for all servicing operations is not cost effective nor universally applicable [9]. As a means for servicing, using human EVAs requires transport of astronauts to and from the target satellite, which poses a massive expense and risk for target satellites outside of Low Earth Orbit (LEO). Robotic servicing satellites are necessary when the target is inaccessible to humans, such as in Geostationary Earth Orbit (GEO) or at a Sun-Earth Lagrange point such

as L1 or L2 [4],[9]. Satellites placed in these distant orbits tend to be large, expensive platforms with advanced payloads and thus represent a strong candidate for on-orbit servicing [9]. Robotic servicing satellites also remove the risk to human life and allow for specialized operations exceeding the dexterity, strength, positioning accuracy, and/or speed of a human operating from within a spacesuit [4]. Development of robotic servicing technology has been of great interest in the 21st century for both government and commercial missions [4],[17],[37], [47],[48].

1.2 Applications of Satellite Fleets

Coordinated satellite fleets operating in formation flight is an enabling technology offering improved flexibility, responsivity, and payload capability [3],[30],[31]. A "fractionated" satellite fleet is a proposed satellite fleet configuration in which the form and function of a traditional monolithic satellite is replaced with several free-flying modules that operate cooperatively to achieve the desired function [3],[31]. Independent research from the United States Defense Advanced Research Projects Agency (DARPA) and Massachusetts Institute of Technology (MIT) both concluded that a fractionated satellite configuration offers better maintainability, scalability, flexibility, reconfigurability, extensibility, and uncertainty handling than the traditional monolithic approach to space missions [3],[31]. This fractionated satellite concept was explored by DARPA with the System F6 program, which sought to demonstrate the necessary formation flight, resource sharing, and reconfiguration technologies critical to a successful fractionated satellite fleet [54]. Similarly, the United States Air Force Research Laboratory (AFRL) explored the application of a satellite fleet in close formation flight to perform flexible, advanced mission operations with distributed X-band transmit and receive payloads with the TechSat 21 program [30]. The mis-

sion was intended to demonstrate capability to perform radio frequency (RF) sparse aperture sensing to perform imaging, precision geolocation, ground moving target indication (GMTI), single-pass digital terrain elevation data (DTED), electronic protection, single-pass interferometric synthetic aperture radar (IF-SAR), and perform high data-rate, secure communications all with the same constellation of 3 small satellites in close formation flight [30]. No equivalent research or demonstration programs exist for formation flight and coordination of satellites with the purpose of satellite servicing. Recent research examined development of collision-free trajectories for approaching servicing satellites, though this work was focused solely on obstacle avoidance [16].

1.3 Thesis Objective

The importance of robotic servicing and limited focused literature regarding servicing satellite fleets motivates the present research. This thesis aims to investigate the mission design and feasibility of a servicing satellite fleet performing servicing operations for a large, central target. The culmination of this work is an integrated mission modeling and scheduling tool capable of rapidly assessing servicing fleet architectures and generating potential mission schedules to optimally achieve mission objectives. This integrated mission modeling and scheduling tool is realized with an implementation of a distributed control architecture using the Horizon Simulation Framework, a flexible task-based systems-of-systems modeling and simulation toolset.

Chapter 2 describes the past, present, and future of robotic satellite servicing as well as pertinent technical background regarding the Horizon Simulation Framework,

relative orbital motion, optimal relative transfer trajectories, and cooperative control. Chapter 3 describes the technical work performed to implement the system model and modify the Horizon Simulation Framework to achieve the integrated mission modeling and scheduling tool. This includes details regarding all utilized equations and algorithms. Chapter 4 presents the sample scenarios examined and the pertinent results and discussions from these scenarios. Chapter 5 provides summaries of the previous chapters as well as concluding remarks regarding the resulting tool, the journey to achieve this tool, and the possible future work enabled by the modifications to the Horizon Simulation Framework.

Chapter 2

BACKGROUND

This chapter provides further background on existing satellite servicing efforts, mission modeling for satellite servicing, and control architectures for distributed systems. This includes in-depth descriptions of the history and current status of satellite servicing, existing model-based systems engineering (MBSE) tools, the Horizon Simulation Framework (HSF), previous applications of HSF, relative orbital motion and control, and cooperative control architectures for multi-agent autonomous systems.

2.1 Robotic Satellite Servicing

Satellite servicing has historically been performed by human Extravehicular Activity (EVA) with the Hubble Space Telescope (HST) and the International Space Station, but there is a rising need and interest for robotic satellite servicing missions for servicing of targets in orbits that are inaccessible to humans or that require specialized operations [4],[9].

2.1.1 Previous Robotic Servicing Missions

The technology for robotic on-orbit servicing has been demonstrated with both government and commercial missions [4],[17]. AFRL developed the Experimental Spacecraft System Number 11 (XSS-11) to develop the necessary technology and demon-

strate the capability for safe, autonomous rendezvous and proximity operations [4]. The XSS-11 was successfully operated in LEO from 2005 to 2006 and demonstrated the capability to autonomously perform safe, reliable rendezvous and proximity operations (RPO) with a non-cooperative resident space object (RSO) [4]. In 2007 the Orbital Express mission, sponsored by DARPA, provided the first demonstration of successful end-to-end robotic satellite servicing activities [4]. The mission involved two specially designed satellites to demonstrate this technology [4]. During the mission, the chaser satellite successfully performed autonomous docking with the target satellite and demonstrated servicing activities including fuel transfer, insertion of battery, and change-out of a flight computer [4]. More recently, Northrop Grumman's Mission Extension Vehicles (MEV-1 & MEV-2) demonstrated the capability for autonomous rendezvous, docking, and servicing in GEO [17]. In February 2020, MEV-1 performed autonomous rendezvous and docking with a functioning but not transmitting communication satellite residing in the GEO graveyard [17]. The communication satellite was still functional, but due to low fuel had been moved to the GEO graveyard for decommission [17]. MEV-1 maneuvered the fuel-deficient communication satellite back to the operational GEO belt and now provides attitude and orbital control, allowing for profitable operation of the communication satellite to resume [17]. In April 2021, MEV-2 performed a similar autonomous rendezvous and docking, this time with a communication satellite that was actively transmitting from the GEO belt [17]. MEV-2 demonstrated another successful autonomous rendezvous and docking in GEO, as well as demonstration that rendezvous and docking can be performed with an active satellite without interrupting the satellite's function [17].

2.1.2 Current Development of Robotic Servicing Missions

In addition to the ongoing MEV missions, there is ongoing development of government-sponsored satellite servicing missions such as the OSAM-1 and RSGS programs [37],[48]. Part of NASA's Exploration and In-space Services (NExIS) efforts is the development of the On-orbit Servicing, Assembly, and Manufacturing 1 satellite (OSAM-1), a robotic system designed to perform a refueling, assembly, and in-space manufacturing flight demonstration [37]. Launch of OSAM-1 is scheduled for 2024 [37]. The mission for OSAM-1 is to rendezvous with, grasp, refuel, and relocate a target satellite to extend its life [37]. OSAM-1 will also utilize its Space Infrastructure Dexterous Robot (SPIDER) to assemble 7 elements to form a functional communications antenna and manufacture a 10-meter lightweight composite beam [37]. DARPA is executing the Robotic Servicing of Geosynchronous Satellites (RSGS) program to develop technologies that will enable cooperative inspection and servicing in GEO, and demonstrate those technologies on orbit within the next 5 years [48]. The goals of the RSGS program are to demonstrate that robotic servicing vehicles can perform safe, reliable, useful, and efficient operations on operational satellites in or near GEO, and to support the development of a servicing satellite with sufficient propellant and payload robustness to enable dozens of servicing missions across several years [48]. The United States Space Force is also actively pursuing satellite servicing technology. Northrop Grumman was recently selected for the Rapid On-orbit Space Technology Evaluation Ring (ROOSTER) effort to produce a ride-share satellite supporting operational and prototype missions in GEO for the technology needed to conduct on-orbit refueling for future missions [47]. Launch for the ROOSTER mission is scheduled for 2026 [47].

2.1.3 Servicing and Assembly of Large Space Structures

The Hubble Space Telescope (HST) was successfully serviced 5 times from 1993 to 2009 with crewed missions to bring astronauts to HST and perform servicing operations [4],[15]. HST was designed to allow for astronauts to perform repairs, replace parts, and update its technology with new instruments [15]. While this did demonstrate the capability to perform a variety of on-orbit servicing activities for a large, complex space structure, it was performed entirely with human EVAs, cost roughly $1-2 billion for each servicing mission, and required serviceability to be a cornerstone of the HST design [4]. The James Webb Space Telescope (JWST), the nearly $10 billion successor to HST that launched in December 2021, is not designed to be serviceable [24]. NASA cites the design redundancy, extensive 7 year integration and test program, and distant orbit about the Sun-Earth second Lagrange point (L2) for this decision [24].

The assembly and servicing of the International Space Station (ISS) is performed with a combination of human EVAs and robotic activity [53]. The Mobile Servicing System (MSS) is the robotics suite onboard the ISS that performs the robotic assembly, maintenance, and resupply activities for the ISS [53]. MSS consists of 3 components: the Space Station Remote Manipulator System (SSRMS, also known as Canadarm2), the Special Purpose Dexterous Manipulator (SPDM, also known as Dextre), and the Mobile Base System (MBS) [53]. The MBS can slide along truss rails and provides a storage area for EVAs and anchor points for the SSRMS (Canadarm2) and the SPDM (Dextre) [53],[12]. SSRMS (Canadarm2) is a large, 7-degrees-of-freedom robotic arm that was used extensively for the assembly of the ISS and currently performs servicing activites for the ISS and grappling of visiting satellites for rendezvous and docking [53],[13]. SPDM (Dextre) is a smaller robotic system with dual-arms, precision handling capability, lights, video equipment, a tool platform, and four tool holders to

perform complex servicing activites and reduce the need for EVAs [53],[14]. Dextre can be attached to the MBS directly, or to the end of the Canadarm2 for even greater mobility [14]. The purpose of this MSS robotic suite is to limit the amount of human EVAs required, however it is still common for human EVAs to be performed for certain operations [53].

The China Manned Space Agency (CMSA) is currently assembling their own space station, called Tiangong, using a similar combination of human EVA and robotic activity [22]. Tiangong assembly began in May 2021 with the launch of the first module, Tianhe, and subsequent visiting cargo and crew satellites have performed autonomous rendezvous and docking with Tianhe [22]. The crewed missions aid in the set-up of Tianhe alongside the external robotic arm used to help position the 2 research modules that were launched and achieved rendezvous, docking, and integration in 2022 [22]. The external robotic arm has been used to aid in EVAs and transpositioning a research module between the forward port and the lateral port [23]. Once completed, Tiangong will be joined by a large space telescope that will share Tiangong's orbit and be able to perform rendezvous and docking for the possibility of repairs, maintenance, and upgrades of the telescope [22].

As part of NASA's Artemis mission to return humans to the surface of the Moon, the Gateway will be an outpost in permanent orbit around the Moon that provides infrastructure for long-term human activity on the lunar surface, as well as a staging point for deep space human and robotic exploration [40]. The Canadian Space Agency (CSA), the developers of Canadarm2, has an agreement with NASA to provide a similar advanced external robotic arm called Canadarm3 for Gateway [40]. Canadarm3 will be designed to maintain, repair, berth, and inspect visiting vehicles, as well as relocate Gateway modules, visually inspect Gateway, install science payloads, and assist astronauts during EVAs and experiments [40]. Canadarm3 is designed to work

autonomously, but could also be operated by flight controllers on the ground or the Gateway crew during EVAs to aid in servicing and assembly operations [40].

2.2 Model-Based Systems Engineering Methodologies

Model-Based Systems Engineering (MBSE) is an approach to the development of complex systems or systems-of-systems in which models of the system, its subsystems, and its requirements are the center-point of the development process [52]. The International Council on Systems Engineering (INCOSE) defines MBSE as the "formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [6]. System relationship models are developed to show relationships between system functions, requirements, developers, and users [6]. These models support 3 key aspects of systems engineering: System Architecture or Functional Flows, System Requirements Traceability, and System/Organizational Process Flows [6]. MBSE is the union of 3 critical engineering concepts: modeling, systems thinking, and systems engineering [52]. In this context, modeling is any abstraction or simplification of a system such that its structure and behavior are representative of the true system while keeping complexity manageable [52]. Systems thinking is an approach in which all subsystems are considered as part of the larger system and great emphasis is placed on the interconnectedness of the components of the system as a whole to assess the behavior of the larger system as it emerges from the activities of the components and subsystems [52]. Systems engineering is the application of systems thinking to determine the necessary architecture, requirements, implementation, design, analysis, integration, verification, and validation necessary for successful development of complex systems

and systems-of-systems [52]. Several methodologies for MBSE have been developed and surveyed [10].

2.2.1 SysML

In practice, most MBSE methodologies make use of System Markup Language (SysML) [6],[10]. SysML is a visual modeling language developed by the Open Modeling Group as an extension of the Unified Modeling Language (UML) to further support the specification, analysis, design, verification, and validation of complex systems [18]. Individual models are created as diagrams to represent elements of the system and relationships between elements [18]. Various diagram types exist in SysML to model the system structure, system behavior, requirements, and parametric relationships between element parameters and their behavior [18].

2.2.2 JPL State Analysis

The State Analysis (SA) methodology, developed by the NASA Jet Propulsion Laboratory (JPL), is a publicly available MBSE methodology [10]. SA is an MBSE methodology with special focus on state and goal-based control [10]. States are defined in SA as a broader extension beyond the classical control theory state-vector definition (position, velocity, attitude) to also include system component modes, available resources, and all other physical or logical parameters that can be used to represent a momentary condition of the evolving system [10]. The states and models together are meant to fully define everything needed to operate the system, predict future states, control towards a desired state, and assess system performance [10]. SA is primarily used

for state-based behavioral modeling, state-based software design, and goal-directed operations engineering [10]. This state-based and goal-directed approach to MBSE is also utilized in the Horizon Simulation Framework, which is the MBSE framework utilized for the present research.

2.2.3 Horizon Simulation Framework

The Horizon Simulation Framework (HSF) is a modeling and simulation framework for verification of system level requirements with an emphasis on state representations, modularity, flexibility, and event scheduling [35],[39],[57]. HSF consists of two major modules with a carefully controlled interface: the main scheduling algorithm and the system model [35],[39]. This fundamental modularity allows for independent development of both the scheduling algorithm and the system model, allowing for much greater user flexibility [35],[39]. The system model consists of a collection of assets, with each asset represented by subsystem models [35]. The system model only interacts with the scheduling algorithm when the scheduling algorithm passes a current schedule and a proposed new event to the system model to evaluate if the system can perform this new event [35],[39],[57]. An event defines the proposed pairing of assets with tasks for the given time step [35],[39],[57]. If the system model is capable of performing the proposed event, this is indicated back to the scheduling algorithm via the "canPerform" method, and the event is added to the tree of possible schedules [35],[39],[57]. The scheduling algorithm exhaustively explores possible schedules through a breadth-first-search scheduling algorithm [35],[39],[57]. As this tree of possible schedules expands, each schedule is scored by evaluating the combined value of the tasks completed, and the lowest scoring schedules are pruned [35],[39],[57].

Users create a model of their scenario by defining the targets and assets for the simulation [35],[39],[57]. Targets are defined by a state vector, the type of task to be performed at each target, and the score for completing the task [35],[39],[57]. Assets are defined by specifications of their subsystem models, constraints, and interactions/dependencies between the subsystems [35],[39],[57]. Definitions of these dependencies dictates how information is shared between subsystem models and the order of subsystem model evaluation [35],[39],[57]. Subsystem requirements are built into the simulation in the form of constraints, which compare the values of subsystem state variables to user-defined limiting ranges or conditions [35],[57]. All subsystem models can store their state data to the system state, a "bulletin board" from which all subsystems can access state data [35]. The access of state data between different subsystems is intended to be carefully controlled with the usage of the dependencies to ensure the integrity and recency of the state data being accessed [35].

Users specify simulation parameters, such as initial states and timescale parameters, in an input scenario XML file to fully define the simulation scenario [35],[57]. The system model structure of assets, subsystems, constraints, dependencies, and custom subsystem parameters are included in an input system model XML file as well [35]. System-level requirements are evaluated by developing numerous schedules and comparing the events executed in each schedule to determine which schedule(s) best meets the system objectives/requirements [35],[57]. A diagram of the HSF architecture is shown below in Figure 2.1 and a sample system model diagram is shown below in Figure 2.2 [35].

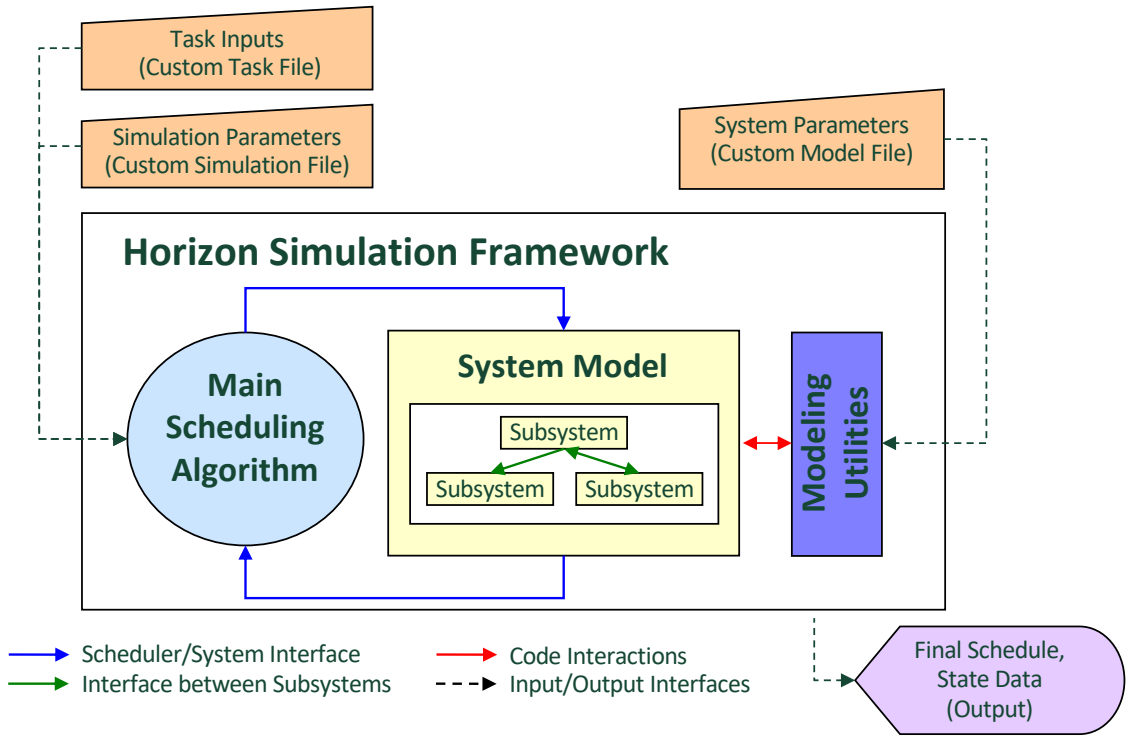

Figure 2.1: HSF Architecture

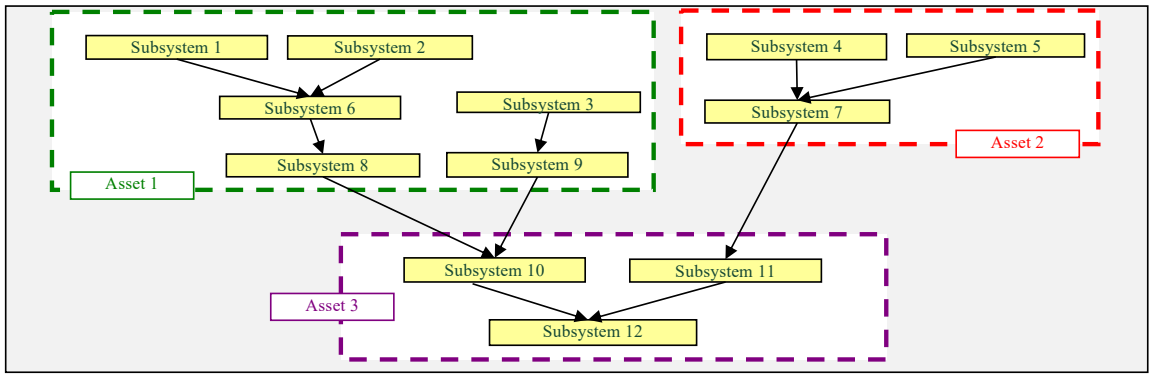

Figure 2.2: Sample System Model Architecture

HSF was originally built with C++ and SysML, but has since been adapted to utilize XML input/output files, a C# scheduler and core code base, and Python scripting capability for subsystem model definition [57]. A major goal of the on-going HSF

project is to foster an open-source community to further develop the models, libraries, and applications of HSF. Previous work has been done to develop modular unit tests and code standardization for the main scheduling algorithm in order to better support such a community [1]. The exemplar mission for HSF is the hypothetical Aeolus Constellation, which consists of two observational satellite assets tasked with imaging numerous ground-point locations and down-linking the image data to the ground through a small collection of ground stations [39],[57].

In addition to the Aeolus Constellation, HSF has previously been utilized for modeling and analysis of complex guidance, navigation, and control (GNC) systems by Maclean [29], Frye [11], and Johnson [21]. Maclean's research involved modeling an active control system for a sounding rocket based on linear-quadratic regulator (LQR) control theory [29]. This work demonstrated effective usage of HSF for modeling a GNC subsystem with an active control system. Frye's research involved modeling the performance of a swarm of UAV's through the use of digital pheromones, pheromone mapping, and an LQR set-point controller to maneuver the pheromone map [11]. This work further demonstrated usage of HSF for GNC modeling, and demonstrated effective usage of HSF for modeling a cooperative control scheme to control a large collection of assets. Johnson's research involved modeling an astronomy CubeSat, with special focus on the attitude dynamics and control [21]. This work demonstrated the applicability of scripted Python models in HSF for spacecraft dynamics and control. These research efforts laid the foundation for the present work to perform modeling of the dynamics and distributed control of a servicing fleet with HSF.

2.3 Relative Orbital Motion and Control

The dynamics of interest for the modeling and simulation of a fleet of servicing satellites is the relative orbital motion of the servicing satellites and their central target. Relative orbital motion can be modeled with a varying degree of fidelity, depending on what assumptions are made and what perturbation forces are considered. Relative orbital motion is controlled through mass-expulsion thrusters, which can be modeled as impulsive changes in velocity or accelerations over time [28]. The pertinent literature on the modeling and optimal control of relative orbital motion is discussed in the following sections.

2.3.1 Relative Orbital Motion

The orbital motion of any Earth-orbiting satellite is governed by a dominant 2-body gravitational attraction towards the Earth, several disturbance or perturbation accelerations arising from various phenomena, and control forces [28]. Relative orbital motion is often expressed as the motion of the "chaser" (or "deputy") object as it moves relative to the "target" (or "chief") object. If the target is in a circular orbit and the target/chaser are closely spaced, the relative motion can be most simply described with the Clohessy-Wiltshire (CW) equations, a famous set of linear constant-coefficient differential equations commonly presented in astrodynamics textbooks [8],[55]. The solution to these differential equations can be expressed with a simple set of matrix equations, referred to as the CW matrices [8],[55]. The equations for the in-plane motion can be parameterized by four constants that represent the following: the size of the relative motion ellipse, the instantaneous location of the center of this ellipse, the drift parameter describing how this ellipse drifts over time, and

the instantaneous position on this ellipse [19]. The out-of-plane motion is completely decoupled from the in-plane motion and follows a sinusoidal trajectory [8],[55],[19]. Simple inversion of the CW matrix equations yields a solution for impulsive transfer trajectories. For a two-impulse maneuver with known start/end states and transfer time, the transfer trajectory and required delta-V vectors for the maneuver can be solved for with another set of matrix equations [8].

The relative orbital motion can also be analyzed for a target in an unperturbed elliptical target and close-by chaser through the linearized equations of relative motion (LERM) and time-varying transformations to apply CW-like equations to elliptical target orbits [51]. The LERM are the linear but time-varying differential equations that emerge from unperturbed 2-body elliptical orbital motion and the assumption that there is a small separation between the chaser and target [51]. The analytical solutions to these differential equations are not as simple as the CW equations and are referred to as the Tschauner-Hempel (TH) equations [51]. The time-varying transformations arise from the Virtual-Chief (VC) and Virtual-Time (VT) methods [51]. In the Virtual-Chief method, a fictitious/virtual satellite with zero eccentricity is used as the chief (target) satellite for the CW equations, with both the actual chaser and target satellites acting as chasers in this Virtual-Chief frame [51]. The relative motion is then defined by matrix equations representing the difference between the actual chaser and target states [51]. In the Virtual-Time method, the time-varying behavior due to the eccentricity of the target orbit is captured by evaluating the CW equations at a virtual time to represent how the solutions follow the same trajectory paths but different motion along these paths [51]. The CW, VC, VT, and TH methods were all compared to direct integration of the full nonlinear unperturbed relative motion differential equations for six cases of closely spaced, elliptical orbits [51]. It was found that the VC and VT methods had considerably less error than the CW

method and the TH method offered at least four orders of magnitude of improved accuracy [51].

The relative orbital motion can also be analyzed with perturbations through exploitation of Gauss' and Cowell's variational equations [41]. Work has been done to model the relative motion dynamics with the effects of primary gravitational (J2) and atmospheric drag perturbations, two effects with a strong influence on relative motion for LEO satellites [41]. A resulting linear time-varying solution can be obtained from assuming small separation between chaser and target, similar to the TH equations, except for the inclusion of the J2 and drag perturbations [41]. Simulation results indicate that this linear time-varying perturbed solution yields more accurate results than the TH equations for chasers and targets in inclined LEO orbits [41].

2.3.2 Optimal Control of Relative Orbital Motion

To preserve fuel and execute operations quickly and safely, the relative orbital motion of a servicing satellite swarm must be controlled with some degree of optimality and consideration of trajectory constraints. There exists many textbook numerical and analytical optimization techniques/approaches for unconstrained and constrained parameter optimization and optimal control, applied to both general problems and specific trajectory optimization problems [2],[5],[7],[27],[38].

When considering the inverted CW matrix equations for a two-impulse maneuver with known start and end states, the only variable that dictates the entirety of the transfer trajectory and the required delta-V is the transfer time [8]. Analytical approaches have been employed to this single-variable parameter optimization problem and reduced the optimization problem to a root-solving problem that has a unique

solution [20],[25]. However, this does not consider any path constraints along the transfer trajectory [20],[25]. Furthermore, this approach does not consider the possibility of improved optimality (i.e., lower delta-V cost or shorter transfer time) with an increased number of impulses along the transfer trajectory.

For fuel-optimal impulsive control of a linear system, it has been shown that the optimal control solution requires at most as many impulses as there are specified final state variables [45]. Thus, up to 6 impulses are required for optimal control of relative orbital motion governed by the linear CW equations, corresponding to the 6-element state vector of position and velocity. Analytical solutions for a fixed time fuel-optimal transfer with two, three, and four impulses have been obtained [43],[44]. However, these solutions are only for co-planar transfers, and do not explore the added optimality from free time, nor do they implement path constraints [43],[44]. The open-time fuel-optimal transfer has been studied, and analytical solutions derived for impulsive transfers [19]. The in-plane and out-of-plane motion, as governed by the CW equations, is decoupled and thus the in-plane (co-planar) and out-of-plane motion can be analyzed separately [19]. Impulsive transfers between two co-planar elliptical relative motion orbits with similar drift parameters were found to be fuel-optimal for 3 impulses, with each impulse occurring when the velocity in the radial direction is zero [19]. For 2 relative orbits with a substantial difference in drift parameter, only sub-optimal strategies with 2 or 3 impulses are assured [19]. Impulsive transfers for out-of-plane motion control were found to be fuel-optimal for a single impulse occurring at the instant of plane intersection [19]. This work also proposed a feedback controller for flying these trajectories with finite thrust to produce a trajectory and delta-V cost similar to the optimal impulsive solution [19]. While this work yields an extremely valuable analytical solution to the unconstrained problem, it does not consider path constraints [19]. There has been further study of optimal impulsive relative orbit control with consideration for J2 perturbation and elliptical

orbits, but this too does not consider path constraints [46]. Research has been done to develop methodologies to achieve impulsive formation control without violating path constraints, however this work does not include fuel minimization [50].

2.4 Cooperative Control Architectures for Multi-Agent Systems

Multi-agent autonomous systems require implementation of a cooperative control architecture to allocate how the multi-agent system will accomplish its mission-level goal and how each agent will be controlled on its own to achieve agent-level tasks. Multi-agent systems can be controlled with centralized control, decentralized control, or distributed control.

2.4.1 Centralized Control

In a Centralized Control Architecture, a single central agent obtains and processes all state information and system goals to dictate the behavior for all other agents [42]. Thus, full system control, cooperation, and optimization is directly computed by the central agent and detailed commands are sent to each agent [42]. The drawback to centralized control is that complexity can grow exponentially with the amount of agents and mission goals, leading to failure of large-scale multi-agent systems [42]. An example of an aerospace application of centralized control is a single ground station controlling the trajectories and activities of a satellite constellation. Centralized control is implemented with HSF for the exemplar Aeolus Constellation mission; the centralized scheduling algorithm processes all state information for the assets and

determines the optimal schedule through its breadth-first-search algorithm [39],[57].

2.4.2 Decentralized Control

In a Decentralized Control Architecture, each agent operates independently based on its own local controller and the available shared or estimated state information of other agents [42]. This eliminates the need for any centralized controller with omnipotent state knowledge of all agents, and reduces complexity growth as the scale of the problem increases [42]. An example of decentralized control is swarm control, which has been studied in aerospace contexts for cooperative control of multi-agent systems [36],[11]. For example, a leader-follower control algorithm with two control regimes was applied to a simulated flock of birds in flight and successfully steered the flock from a random initial state into an exponentially stable "V" flight formation similar to that which naturally occurs for large flocks of birds [36]. Decentralized control has been implemented with HSF for the control of a swarm of UAV's tasked with either search and rescue or reconnaissance using digital pheromones, pheromone mapping, and a local controller for each agent to maneuver the pheromone map [11].

2.4.3 Distributed Control

In a Distributed Control Architecture, the state processing, decision making, autonomy, and control is distributed between some central entity/agent and each of the other agents. This enables a trade-off between the optimality enabled by centralized control and the scalability enabled by decentralized control. While a decentralized control architecture is more focused on locally available data, a distributed control

architecture utilizes more information sharing between assets to improve the overall performance of the multi-agent system. Constraints on the exchanged information can be imposed to establish a trade-off between system performance and complexity. A well-designed distributed control architecture must properly balance how much state processing and decision making will be centralized and how much will be allocated to the individual agents, with proper consideration for state uncertainties, communication capabilities, and control algorithm complexity.

2.5 Thesis Statement

The promising value of robotic servicing for improvements in space utilization and the limited volume of research studying servicing satellite fleets motivates the present research. This thesis aims to investigate the mission design, high-level system requirements, and concept feasibility for a servicing satellite fleet performing operations for a large, central target in Geostationary Orbit.

Mathematical models of the servicing satellite fleet are implemented using the Horizon Simulation Framework (HSF), an extensible task-based system-of-systems modeling and simulation toolset for assessing aerospace systems and determining day-in-the-life system schedules for all assets in the system. HSF is architected with an emphasis on state representations and event scheduling, and consists of two major components: the scheduling algorithm and the system model.

The HSF system models for the servicing satellite fleet are constructed with a special emphasis on the dynamics, optimal control, and trajectory constraints. Relative orbital motion and control have been studied extensively, and several approaches

exist with varied model fidelity, transfer trajectory optimality, transfer trajectory constraints, and algorithm complexity.

The current HSF iteration and exemplar Aeolus mission utilizes centralized decision making and planning, all within the main scheduling algorithm. This work expands the capabilities and mission applications of HSF to include an example of a distributed control architecture in which only high-level objectives are controlled by the main scheduling algorithm, while individual trajectory optimization and inter-asset constraint evaluations are performed by separate subsystem models. This includes the development and integration of an un-tasked/passive asset in the HSF system model.

The culmination of this work is an integrated mission modeling and scheduling tool that is useful for rapid evaluation of servicing fleet architectures and generation of potential mission schedules to optimally achieve the servicing goals. This integrated mission modeling and scheduling tool is applied to a sample design reference mission (DRM) to demonstrate the utility of the tool and investigate the concept feasibility.

Chapter 3

METHODOLOGY

This chapter describes the methodology implemented to develop the integrated mission modeling and scheduling tool capable of assessing servicing fleet architectures and generating optimal mission schedules. This includes description of the mission concept, HSF subsystem modeling architecture, distributed control architecture, necessary alterations and extensions to HSF, relative orbital motion and maneuvering models, optimization algorithms for fuel-optimal trajectories with path constraints, multi-objective optimization for configurable time-optimal and fuel-optimal trajectories, modeling of the servicing activity, and implementation of collision avoidance using the distributed control architecture and the HSF scheduler. Information for obtaining and utilizing the source code is provided in Appendix B.

3.1 Mission Concept

The mission concept explored is a fleet of small servicing satellites tasked with rapidly performing numerous operations for a single large space structure in Geostationary Orbit. A cartoon of this concept is shown below in Figure 3.1. The servicing satellites each possess similar mass properties and agile propulsion capabilities, but varied tooling capabilities for various specialized servicing operations. The central space structure has several external target locations at which a servicing operation is required. The individual operations are abstracted in this work, but could represent refueling, repair, parts replacement, or assembly/integration of new components. Some

operations require little servicing time and only simple tooling that is available on all servicers within the fleet, while other operations require longer servicing times and specialized tooling that is only present on some of the servicing assets. The servicing operations are assigned relative scores, and thus the HSF scheduling algorithm objective function is easily formulated as the sum of the scores of the servicing tasks completed in the allotted time.

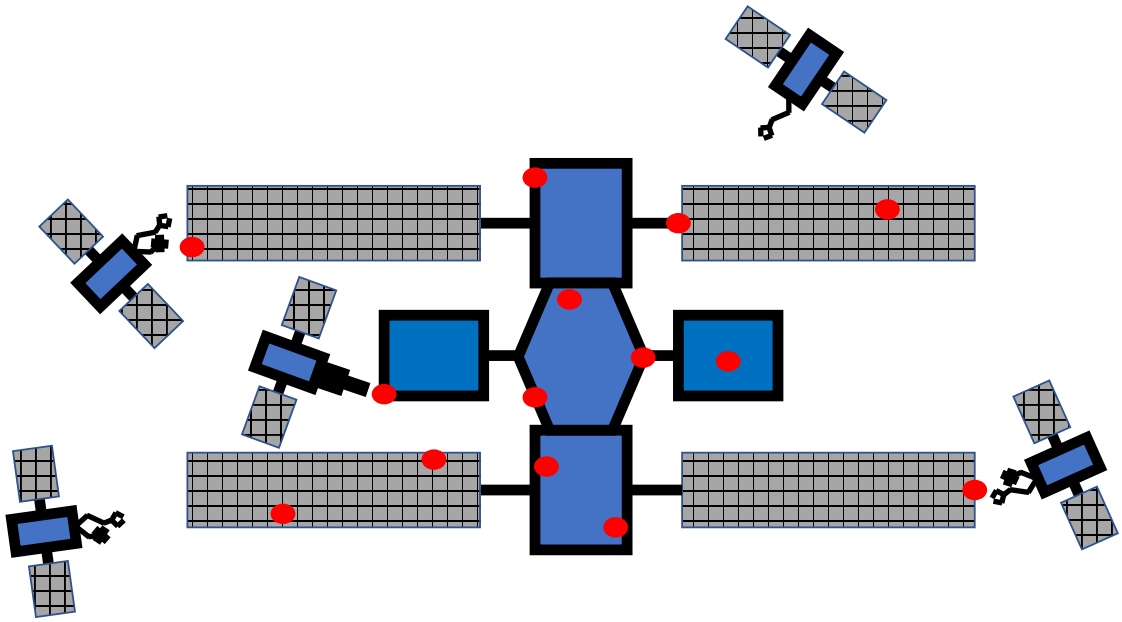

Figure 3.1: ConOps Cartoon of Servicing Fleet

A safety keep-out zone (KOZ) is modeled around the central space structure to ensure the servicers do not collide with the target while transferring. For any valid transfer trajectories to be determined, the target locations that mark the end-points of the transfers must be located outside of this KOZ. The target locations should be near the surface of the KOZ to represent "entry points" at which separate sensors and controllers can be utilized to perform the terminal rendezvous, docking, and servicing operations. The mission scenario begins with all servicing satellites in a common natural motion circumnavigation (NMC) relative orbit about the central space structure. This enables the servicing satellites to perform inspection of the central space

structure from all angles prior to engaging in any servicing operations. This also ensures that their drift trajectories do not cause collisions with each other or the central space structure itself. Upon mission start, the servicing satellites maneuver from their initial NMC to the various target locations around the central space structure and continue to rapidly perform subsequent servicing operations until all servicing operations are completed, or the allotted mission time span ends.

3.2 HSF System Model Architecture

The system model for the servicing satellite fleet is composed of several servicing assets and a fictitious "watchdog" asset to monitor the trajectories of the servicers. The fictitious "watchdog" asset has no physical meaning in the mission, but exists in the HSF system model with the *collisionAvoidance* subsystem model to evaluate the computed trajectories to ensure no vehicles are expected to have a collision with each other. The servicing targets are defined by their relative value or score, the tooling type required for the operation, and their fixed location in the central spacecraft's RIC (radial, in-track, cross-track) reference frame. This corresponds to a central space structure with a fixed attitude in the RIC frame. The model for each servicing asset is defined by its *tool* and *guidance* subsystems. The *tool* subsystem models how the asset's tools interact with the task types for the target operation. The *guidance* subsystem computes and models an optimal transfer trajectory to the proposed target location. Note that this is abstracted to only model the relative position, not the attitude. This architecture is diagrammed below in Figure 3.2, showing the subsystems and their evaluation order for a sample configuration with 5 servicer assets.

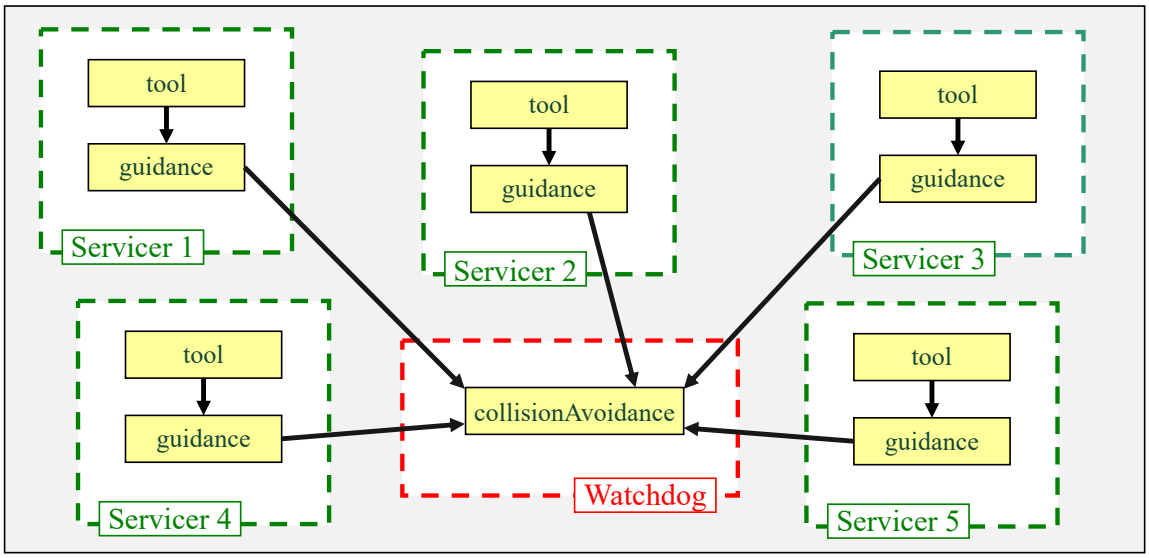

Figure 3.2: Servicing Fleet System Model Architecture

3.2.1 Distributed Control Architecture

To achieve optimal trajectory planning and mission schedules with consideration of individual path constraints and collective asset spacing constraints, a distributed control architecture was implemented to allocate decision making, route planning, and schedule feasibility evaluation across the main HSF scheduling algorithm, individual servicing assets, and a fictitious "watchdog" asset. The main HSF scheduling algorithm maintains its typical role as task proposer and schedule generator by proposing new events that pair each servicing asset with either a target servicing operation or a null task (to allow for time delays). Each proposed event is evaluated by the system model to determine if the proposed new event for each current schedule can be performed by the system. Each individual transfer trajectory for each active servicer/-target pair is computed by the *guidance* subsystem model for the individual servicer asset. The fictitious "watchdog" asset collectively evaluates these trajectories with the *collisionAvoidance* subsystem model to determine if the proposed schedule would cause any potential collisions between the servicer assets. If there is a risk of collision,

the *collisionAvoidance* model indicates that the system cannot perform the proposed schedule. Proposed schedules include all combinations of active and passive assets that allow the scheduler to discover feasible schedules that achieve the mission goals.

3.2.2 Evaluation of the canPerform Function

The system model determines and returns the canPerform value to the HSF scheduling algorithm based on the proposed event and the current state of the assets. This is architected as follows for the system model. First, the *tool* subsystems are evaluated for each active servicer asset to determine if the asset is capable of performing the proposed task at all based on the available tooling. If any of the assets cannot perform their proposed task, the canPerform is returned as False to the HSF scheduler without evaluating the *guidance* or *collisionAvoidance* models at all. Next, the individual optimal transfer trajectories are computed with the *guidance* subsystems for each active asset. The *guidance* subsystem model returns False for the canPerform if the solver cannot determine a feasible trajectory solution subject to the path and fuel constraints for that servicer asset. Lastly, a single evaluation of the *collisionAvoidance* model on the fictitious "watchdog" asset is done to evaluate if the individual transfer trajectories bring the servicers too close together at any time. If the *collisionAvoidance* models succeeds, the system model returns True for the canPerform back to the HSF scheduler. This order of evaluation is achieved with careful definitions of the constraints on the subsystems; the HSF scheduler evaluates the subsystem models according to how the constraints are defined.

3.3 HSF Alterations and Extensions

In order to achieve this system model, critical alterations and extensions were implemented in the main HSF codebase. The primary alteration involves allowing assets to be passive for a given time step or for the entire simulation. Additionally, some "tricks" were implemented and exploited to side-step the default structure and behavior of HSF in order to enable the necessary modeling extensions.

To allow for the servicing assets to have idle periods that enable the HSF scheduler to discover collision-free schedules, individual assets must be able to be made passive, or un-tasked, for any given time step. This is achieved by introducing an empty/null task that is given a fictitious "EmptyTarget" and a completion score of zero. Thus, each subsystem model must begin their canPerform evaluation by checking if the proposed task for the asset is for the "EmptyTarget" and handling this case appropriately. The *tool* and *guidance* models return True for the canPerform if assigned the empty task, thus pacifying the asset for that time step. By including this empty task in the stack of possible tasks, the HSF scheduler now generates new proposed events, including every possible combination of passive and active assets. For example, if there are 2 assets (call them asset A and B) and 2 targets (call them task 1 and 2), previous iterations of the HSF scheduler would propose $2^2 = 4$ new possible events: A1+B1, A1+B2, A2+B1, A2+B2. With this new update to include the empty task (call it task 0), the HSF scheduler now proposes $3^2 = 9$ new possible events: A0+B0, A0+B1, A0+B2, A1+B0, A1+B1, A1+B2, A2+B0, A2+B1, A2+B2. Notice that this includes an event in which both assets are passive (i.e., A0+B0) as well as several events in which both assets are active (e.g., A1+B2). It is also possible for subsystem models to return a canPerform of False if presented with an "EmptyTarget" to not allow passive time steps, or they may perform additional calculations and update their

states accordingly when tasked with an "EmptyTarget". This extension enables the subsystem models to process how the states evolve when the assets are not assigned an active task, allowing for time delays to be introduced in the trajectories in order to achieve collision avoidance. The duration of each time delay is the length of the fundamental time step specified with the input XML files.

So that it is possible for the "watchdog" asset to participate in the system model even though it cannot complete any real tasks itself, individual assets must be able to be completely passive for the entire simulation. This is achieved by implementing an IsTaskable member variable for the Asset.cs class to indicate if a given asset is a taskable asset or not. By default, the IsTaskable variable is set to True so that all assets are taskable by default. This default behavior better supports legacy applications of HSF such as Aeolus and aligns with the goals of the HSF scheduler: schedule events with high-scoring tasks being completed by the assets. The IsTaskable variable will be set to False if it is specified as such in the XML input file that defines each asset. When checking a newly proposed event, the HSF scheduler will first check that any assets that are un-taskable must be assigned to the "EmptyTarget" for the canPerform to be True for the proposed event. This enforces that any un-taskable asset will never be assigned any "real" tasks, thus allowing for the "watchdog" asset to participate in the system model and utilize its *collisionAvoidance* subsystem model without performing any "real" tasks. These "passive asset" extensions to HSF enable broader applications of the tool to mission scenarios in which not all assets require constant tasking for the system to operate.

Another critical extension of HSF was to enable the dynamicState (position & velocity state vector) for an asset to be null. This enables the system model to side-step the traditional built-in HSF model structure in which every asset's dynamicState is propagated using integration of differential equations of motion selected from a set

of built-in options. This traditional built-in HSF model was developed for inertial orbital motion and relies on the equations of motion being defined in differential form [35]. Side-stepping this modeling structure enables a more abstract definition of an asset and allows for analysis of systems in which the subsystem models dictate the behavior of the position & velocity state vectors. This was a useful extension to enable direct usage of analytical solutions for the relative orbital motion of servicing satellites without the need for numerically integrating the differential equations. Additionally, this enabled easy updating of the relative state vectors with the application of maneuvers computed by the *guidance* subsystem model. With this side-step extension, the *guidance* subsystem model defines and tracks the relative state vector information using carefully controlled dictionary keys within the system state, which contains all subsystem state information in a common "bulletin board" [35]. Gathering all asset's relative state vectors onto the common "bulletin board" also enables extraction of these relative states for the conjunction analysis performed by the *collisionAvoidance* subsystem model. Furthermore, this extension enables broader applications of HSF for systems in which the asset's state vectors are more conveniently defined by subsystem models, and systems in which assets are more abstract entities that do not have position & velocity state vectors. The restrictiveness of this current framework adds motivation to on-going work to methodically and robustly improve the framework and functionality of dynamics modeling in HSF.

Lastly, in order for the "watchdog" asset to access the computed position & velocity state vectors from all of the individual *guidance* subsystems for conjunction analysis, a "trick" was implemented and exploited to side-step the dependency feature of HSF. Dependencies are the intended means for sharing any state data between subsystems and ensuring subsystems are evaluated in the proper order [57],[35]. Dependencies are intended to provide directional linkages between the subsystems to describe the subsystem tree and enforce the subsystem evaluation order such that when a constraint

for a subsystem is evaluated, any and all dependent information from other subsystems is calculated first, thus ensuring the integrity and recency of the accessed state data [57],[35]. The dependency functions are intended to act as the go-between to access state data, but with the current HSF framework the state data for all subsystems is available on the public "bulletin board" [57],[35]. Due to limitations with the current dependency framework and functionality, it is nearly impossible to effectively implement dependencies that span across multiple assets, which is a critical feature of the "watchdog" asset in the system model. The availability of the public "bulletin board" of state data was exploited to avoid usage of HSF dependencies altogether. To handle the issue of state integrity and recency, the subsystem constraints were carefully constructed. As diagrammed in Figure 3.2, the *collisionAvoidance* model depends on the information computed from each of the *guidance* models, which in turn each depend on the corresponding *tool* model for that asset. This model evaluation order is achieved by defining constraints for each of the *tool* and *guidance* subsystem models and not defining any for the *collisionAvoidance* model, exploiting the fact that the HSF scheduler will first evaluate each subsystem that has a constraint before evaluating subsystems that do not have constraints. To achieve evaluation of the *tool* models before each of the *guidance* models, the constraints are defined for each asset in the system model XML input file such that the *tool* constraint is listed above the *guidance* constraint for each asset. This exploits the fact that the HSF scheduler evaluates the subsystem constraints in the order that they are listed and parsed in the system model XML input file that defines the assets and their subsystems. The *tool* subsystem constraints do not represent anything physical and will never fail, they are only included as "dummy" placeholder constraints in order to exploit this behavior. This "trick" is only effective for this system model because the flow of the dependent subsystems is fairly simple. Further development and implementation of dependencies within the HSF codebase can enable complex dependency flows to be

handled internally with proper usage of the dependencies and dependency functions. The limitations of the current dependency framework in HSF adds motivation to ongoing work to methodically and robustly improve the framework and functionality of subsystem dependencies in HSF.

3.4 Relative Orbital Motion Modeling

The relative orbital motion was modeled with the Clohessy-Wiltshire, or CW equations. The CW equations model the orbital motion of a Chaser object relative to a Target object and assumes that the Chaser and Target are closely spaced relative to the size of the Target orbit, the Target orbit is nearly circular, and there are no orbital perturbations. These assumptions are valid for first-order mission analysis of the servicing of a GEO target, which has an orbital radius of 42,164km, and servicing satellites operating within 1km of the target. While there are errors between the CW model and high fidelity full-force propagation, the CW equations are a useful first-principal modeling tool for proof-of-concept assessment of servicing satellites operating in close proximity to the central target.

Derivations for the analytical solution to the linear constant-coefficient differential equations arising from these assumptions can be commonly found in astrodynamics textbooks [8],[55]. The analytical solution defines the evolution of the relative position and velocity of a Chaser object about a Target in the Target RIC frame as a function of only the initial relative state, the elapsed time, and the mean motion of the Target's inertial orbit n. This analytical solution can be written in a condensed matrix format to define the evolution of the position and velocity vectors ($\vec{R}$ & $\vec{V}$), as seen below.

$$\vec{R}(t) = \Phi_{RR}(n,t)\,\vec{R}_i \;+\; \Phi_{RV}(n,t)\,\vec{V}_i \tag{3.1}$$

$$\vec{V}(t) = \Phi_{VR}(n,t)\,\vec{R}_i \;+\; \Phi_{VV}(n,t)\,\vec{V}_i \tag{3.2}$$

For these equations, $\vec{R}_i$ and $\vec{V}_i$ are the initial relative position and velocity vectors, Φ_{RR}, Φ_{RV}, Φ_{VR}, and Φ_{VV} are the 3x3 state transition matrices, n is the mean motion of the Target orbit in radians per second, and t is the time elapsed since the initial states in seconds. State vectors can be defined with any consistent distance unit, though are typically defined in meters and meters per second or kilometers and kilometers per second, depending on the proximity of the objects. These state transition matrices contain both periodic and secular terms, as seen below in their definitions.

$$\Phi_{RR} = \begin{bmatrix} 4 - 3\,\cos(nt) & 0 & 0 \\ 6\,(\sin(nt) - nt) & 1 & 0 \\ 0 & 0 & \cos(nt) \end{bmatrix} \tag{3.3}$$

$$\Phi_{RV} = \frac{1}{n}\begin{bmatrix} \sin(nt) & 2\,(1 - \cos(nt)) & 0 \\ 2\,(\cos(nt) - 1) & 4\,\sin(nt) - 3\,nt & 0 \\ 0 & 0 & \sin(nt) \end{bmatrix} \tag{3.4}$$

$$\Phi_{VR} = n\begin{bmatrix} 3\,\sin(nt) & 0 & 0 \\ 6\,(\cos(nt) - 1) & 0 & 0 \\ 0 & 0 & -\sin(nt) \end{bmatrix} \tag{3.5}$$

$$\Phi_{VV} = \begin{bmatrix} \cos(nt) & 2\sin(nt) & 0 \\ -2\sin(nt) & 4\cos(nt) - 3 & 0 \\ 0 & 0 & \cos(nt) \end{bmatrix} \tag{3.6}$$

3.4.1 Relative Motion Trajectories

For relative motion trajectories governed by the CW equations, the out-of-plane motion (Z component in RIC frame) is completely decoupled from the in-plane motion (X & Y components in RIC frame) and follows a sinusoidal trajectory [8],[55],[19]. The in-plane motion can take the form of a variety of free-flight trajectory types including hops, V-bar holds, R-bar drifts, and natural motion circumnavigation (NMC) trajectories [56]. Hops are the result of inertial phasing maneuvers and thus hop trajectories that are "above" the Target in the radial direction result in drifting "backwards" in the negative in-track direction while hops that are "below" the Target result in drifting "forwards" [56]. V-bar holds are the result of the Chaser sharing the same inertial orbit as the Target but being spaced apart in true anomaly, there is no in-plane motion for a V-bar hold [56]. R-bar drifts are the result of the Chaser occupying a slightly larger or smaller circular orbit than the Target, resulting in straight-line drifts at a constant radial distance. R-bar drifts move "forwards" if they lie "below" the Target and "backwards" if they lie "above" the target, and they drift at speeds proportional to the radial distance away from the Target [56]. NMCs are the result of the Chaser occupying an elliptical orbit with the same period as the Target, thus leading to periodic natural motion in which the Chaser maps out a signature "football" shape that is twice as wide (in-track direction) as it is tall (radial direction) [56]. Shown below are cartoons visualizing these relative motion trajectories in the

inertial and RIC frames, in Figures 3.3, 3.4, 3.5, 3.6. Note that these figures come from Dr. Woffinden's dissertation [56], and slightly different terminology is used for the reference frame. In these figures, Woffinden utilizes the LVLH frame instead of the RIC frame, which uses slightly different definitions for x, y, and z. These plots label the radial direction as "Altitude", and the in-track direction as "Downrange". Despite these nuanced differences, these cartoons are an excellent visualization of the relative motion trajectories arising from the CW equations and are included in this work for completeness.

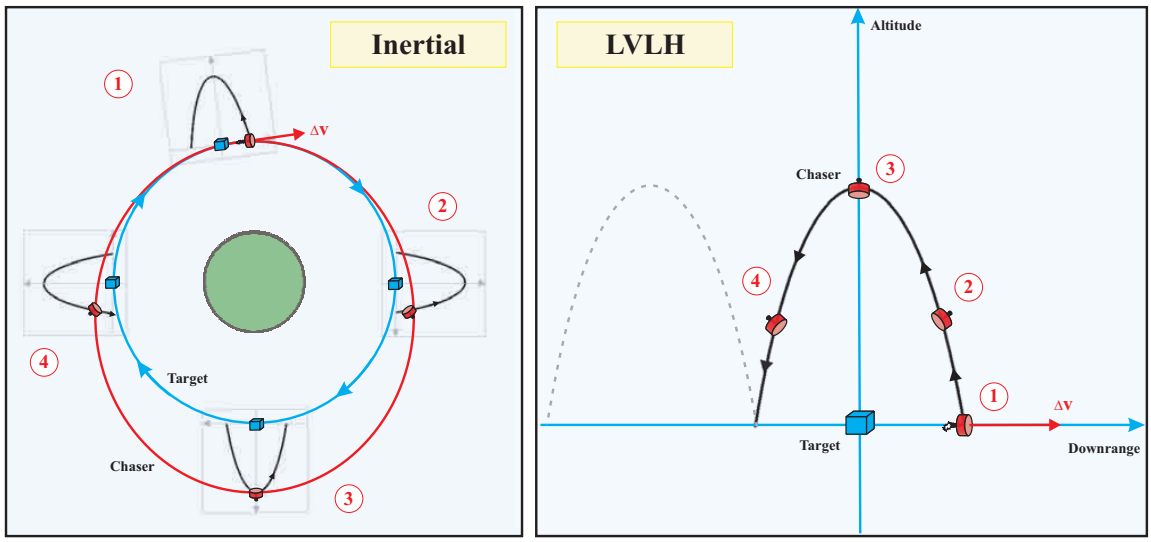

Figure 3.3: Hop Trajectory [56]

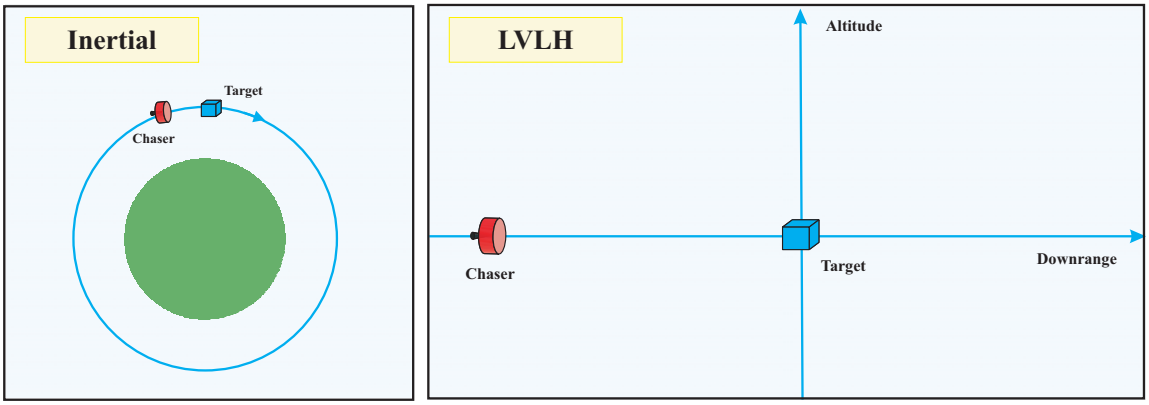

Figure 3.4: V-Bar Hold Trajectory [56]

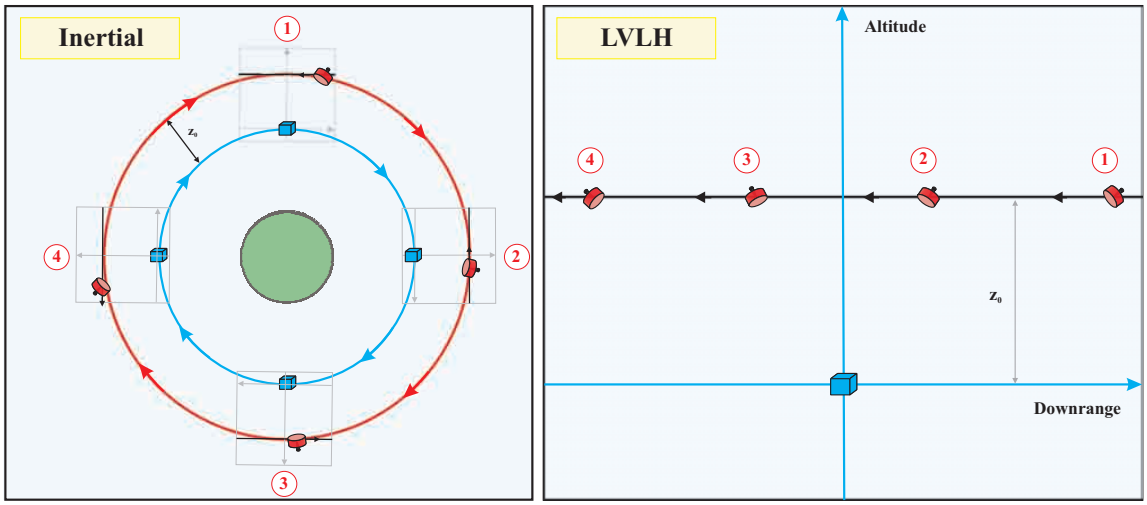

Figure 3.5: R-Bar Drift Trajectory [56]

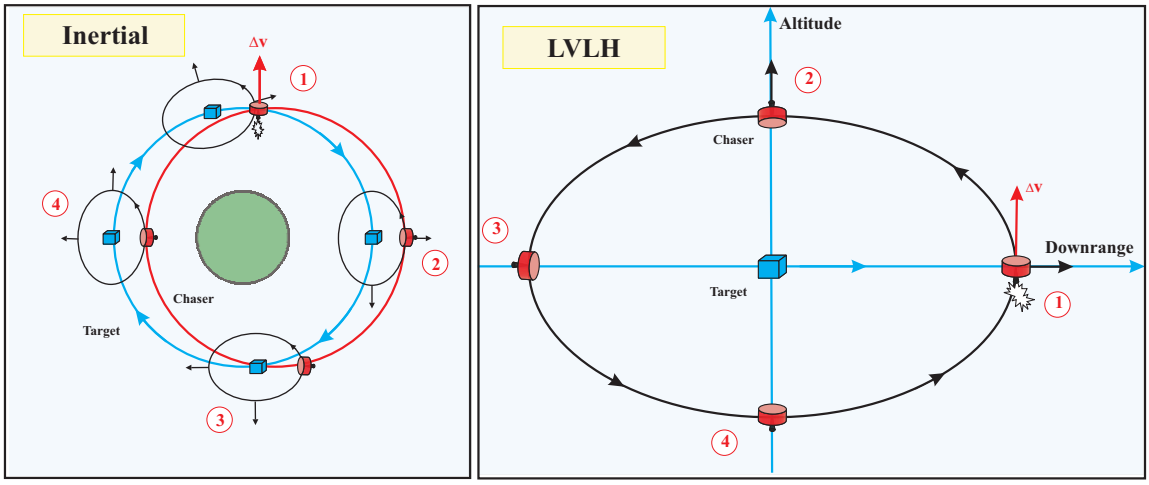

Figure 3.6: NMC Trajectory [56]

3.4.2 Two Impulse Transfer Trajectories

Consider a two-impulse transfer from some arbitrary initial and final states in which the impulses are applied at the boundary states: the first impulse starts the transfer trajectory from the current state towards the final state, and upon arriving at the final position, the second impulse changes the current velocity to match the desired final velocity. Note that this is commonly presented in astrodynamics textbooks as a

rendezvous problem in which the Chaser's final state corresponds to zero velocity and zero position at the origin of the Target RIC frame for which these CW equations apply, but in this work the final state remains arbitrary.

The current or initial state vectors will hereafter be referred to as $\vec{R}_i$ & $\vec{V}_i$, the desired or final state vectors will be referred to as $\vec{R}_f$ & $\vec{V}_f$, and the transfer time will be referred to as t. For a given set of boundary states (initial & final) and a transfer time, a unique, analytical solution exists that fully defines the transfer trajectory and the required delta-V for both impulses.

First, consider the velocity vector immediately after the first impulse to be $\vec{V}_i^{\,+}$. The state vector after the first impulse is now $\vec{R}_i$ & $\vec{V}_i^{\,+}$, and the state evolution must result in $\vec{R}(t) = \vec{R}_f$. Substituting into equation (3.1) yields:

$$\vec{R}_f = \Phi_{RR}(n,t)\,\vec{R}_i \;+\; \Phi_{RV}(n,t)\,\vec{V}_i^{\,+} \tag{3.7}$$

Solving for $\vec{V}_i^{\,+}$ and removing the (n,t) notation yields:

$$\vec{V}_i^{\,+} = (\Phi_{RV})^{-1}\,(\vec{R}_f \;-\; \Phi_{RR}\,\vec{R}_i) \tag{3.8}$$

With the starting state for the transfer trajectory $\vec{R}_i$ & $\vec{V}_i^{\,+}$ fully determined, the ending state for the transfer trajectory can be determined with forward propagation. Consider the velocity immediately before the second impulse to be $\vec{V}_f^{\,-}$. Substituting into equation (3.2) and removing the (n,t) notation yields:

$$\vec{V}_f^{\,-} = \Phi_{VR}\,\vec{R}_i \;+\; \Phi_{VV}\,\vec{V}_i^{\,+} \tag{3.9}$$

The delta-V required for the transfer can now be calculated as:

$$\Delta\vec{V}_i = \vec{V}_i^{\,+} \; - \; \vec{V}_i \tag{3.10}$$

$$\Delta\vec{V}_f = \vec{V}_f \; - \; \vec{V}_f^{\,-} \tag{3.11}$$

Note that equations (3.10) and (3.11) utilize the difference in relative velocities measured in the Target RIC frame to compute the delta-V for the impulses, even though the delta-V cost is the difference in absolute (i.e., inertial) velocities. It can be shown that these are in fact equivalent in magnitude, and thus it is appropriate to treat the magnitude of the difference in relative velocities as the magnitude of the inertial delta-V cost for the maneuver [8].

3.5 Delta-V Optimization of Unconstrained Two-Impulse Transfers

Consider the optimization cost function to be the total delta-V cost of the transfer, as defined by the sum of the magnitudes of $\Delta\vec{V}_i$ and $\Delta\vec{V}_f$. As previously noted, these delta-V magnitudes can be computed from changes in relative velocity in the Target RIC frame [8]. Expanding vector components, this becomes:

$$J = \sqrt{(\Delta V_{X_i})^2 + (\Delta V_{Y_i})^2 + (\Delta V_{Z_i})^2} + \sqrt{(\Delta V_{X_f})^2 + (\Delta V_{Y_f})^2 + (\Delta V_{Z_f})^2} \tag{3.12}$$

This cost function can be used to determine the fuel-optimal unconstrained transfer trajectory. To obtain an optimal solution, the standard single-variable first order

necessary condition for optimality $\frac{\partial J}{\partial t} = 0$ is required. Solutions to this non-linear equation are obtained via numeric root solving, but an initial investigation into an analytical approach yields valuable insight into the behavior of this non-linear cost function and the singularities that arise for certain transfer times.

3.5.1 An Analytical Search for Singularities

First, we obtain a scalar equation for each component of the cost function (3.12). We can determine the components for the first impulse by substituting equation (3.8) into equation (3.10) to obtain:

$$\Delta\vec{V}_i = (\Phi_{RV})^{-1}\,(\vec{R}_f - \Phi_{RR}\,\vec{R}_i) - \vec{V}_i \tag{3.13}$$

To proceed analytically, an analytical solution for the matrix inverse $(\Phi_{RV})^{-1}$ is required. It can be shown that this analytical solution is:

$$(\Phi_{RV})^{-1} = n \begin{bmatrix} \dfrac{3nt\ -\ 4\sin(nt)}{\text{denom}(n,t)} & \dfrac{2\,(1-\cos(nt))}{\text{denom}(n,t)} & 0 \\ \dfrac{-2\,(1-\cos(nt))}{\text{denom}(n,t)} & \dfrac{-\sin(nt)}{\text{denom}(n,t)} & 0 \\ 0 & 0 & \dfrac{1}{\sin(nt)} \end{bmatrix} \tag{3.14}$$

With

$$\text{denom}(n,t) = 3\,nt\,\sin(nt)\ +\ 8\,\cos(nt)\ -\ 8 \tag{3.15}$$

This analytical formulation for $(\Phi_{RV})^{-1}$ is also useful for efficient implementation of the two-impulse delta-V cost calculation described with equations (3.8), (3.9), (3.10),

and (3.11), as it removes the need for an iterative numerical method to determine the matrix inverse. By substituting equations (3.14) and (3.3) into equation (3.13), carefully expanding, and separating terms by component, we obtain scalar functions directly describing the first impulse change in velocity components as functions of the boundary states and the transfer time, as shown below in equations (3.16), (3.17), (3.18).

$$\begin{aligned}\Delta V_{X_i} = &\frac{3n^2t \ - \ 4n\sin(nt)}{\text{denom}(n,t)}(X_f - X_i(4 - 3\cos(nt))) \ + \ ... \\ &+ \ \frac{2n(1-\cos(nt))}{\text{denom}(n,t)}(Y_f - Y_i - 6X_i(\sin(nt) - nt)) \ - \ V_{x_i}\end{aligned} \tag{3.16}$$

$$\begin{aligned}\Delta V_{y_i} = &\frac{-2n(1-\cos(nt))}{\text{denom}(n,t)}(X_f - X_i(4 - 3\cos(nt))) \ + \ ... \\ &+ \ \frac{-n\ \sin(nt)}{\text{denom}(n,t)}(Y_f - Y_i - 6X_i(\sin(nt) - nt)) \ - \ V_{y_i}\end{aligned} \tag{3.17}$$

$$\Delta V_{z_i} = \frac{n}{\sin(nt)}(Z_f - Z_i\ \cos(nt)) \ - \ V_{z_i} \tag{3.18}$$

It is evident from these scalar $\Delta\vec{V_i}$ component equations that the cost function contains singularities when the denominator in some terms evaluates to zero. Singularities occur for the ΔV_{X_i} and ΔV_{Y_i} components whenever denom$(n,t) = 0$. This occurs whenever $t = \frac{2k\pi}{n}$, where k is a non negative integer, and for additional irrational relationships that must be determined numerically. Singularities occur for the ΔV_{Z_i} component when $\sin(nt) = 0$, which occurs whenever $t = \frac{k\pi}{n}$, where k is a non negative integer.

Next, we can determine the components for the second impulse by substituting equation (3.8) into equation (3.9) and the resulting intermediary equation into equation (3.11) to obtain:

$$\Delta\vec{V_f} = \vec{V_f} \; - \; \Phi_{VR}\vec{R_i} \; - \; \Phi_{VV}(\Phi_{RV})^{-1}\,(\vec{R_f} - \Phi_{RR}\,\vec{R_i}) \tag{3.19}$$

By substituting equations (3.3), (3.5), (3.6), and (3.14) into equation (3.19), carefully expanding, and separating terms by component, we obtain scalar functions directly describing the second impulse change in velocity components as functions of the boundary states and the transfer time, as shown below in equations (3.20), (3.21), (3.22).

$$\Delta V_{x_f} = V_{x_f} - 3nX_i\sin(nt) - \frac{n}{\text{denom}(n,t)}f_1 \tag{3.20}$$

$$\Delta V_{y_f} = V_{y_f} - 6nX_i(\cos(nt) - 1) - \frac{n}{\text{denom}(n,t)}f_2 \tag{3.21}$$

$$\Delta V_{z_f} = V_{z_f} + nZ_i\sin(nt) - n\frac{\cos(nt)}{\sin(nt)}(Z_f - Z_i\cos(nt)) \tag{3.22}$$

Where

$$\begin{aligned} f_1 = {} & (3nt\cos(nt) - 4\sin(nt))(X_f - X_i(4 - 3\cos(nt))) + ... \\ & + (2\cos(nt) - 2\cos^2(nt) - 2\sin^2(nt))(Y_f - Y_i - 6X_i(\sin(nt) - nt)) \end{aligned} \tag{3.23}$$

$$f_2 = (8(\cos^2(nt) + \sin^2(nt)) - 6nt\sin(nt) - 14\cos(nt) + 6)(X_f - X_i(4 - 3\cos(nt))) + ...$$
$$- \sin(nt)(Y_f - Y_i - 6X_i(\sin(nt) - nt)) \tag{3.24}$$

It is evident from these scalar $\Delta\vec{V}_f$ component equations that the cost function contains singularities when the denominator in some terms become zero. Singularities occur for the same conditions as for the scalar $\Delta\vec{V}_i$ component equations, which is whenever any of the following conditions are met:

$$t = \frac{k\pi}{n} \text{ for } k = 0, 1, 2, 3, ... \tag{3.25}$$

$$\text{denom}(n, t) = 3\,nt\,\sin(nt)\,+\,8\,\cos(nt)\,-\,8 = 0 \tag{3.26}$$

Due to the squared nature of the cost function, singularities result in the value of the cost function to approach a non-physical infinite value. These singularities occur due to the periodic nature of the CW equations and are artifacts of the assumptions and approximations made in deriving the CW equations; no inertial transfer trajectory of non-zero duration actually requires an infinite impulsive delta-V. If the transfers were instead computed in an inertial frame using an approach such as an iterative solution to Lambert's problem, the delta-V costs may be large, but would remain bounded for any positive-value transfer time.

Although the CW equations and the two-impulse transfer solution are functions of both the mean motion n and the transfer time t, the singularities can be expressed as a function of their product (nt), which represents the central angle subtended by the Target's circular orbital motion during the transfer time. The solutions to equation (3.25) re-written in terms of nt are simply all non negative multiples of π. The solutions to equation (3.26) re-written in terms of nt is all non negative multiples of 2π and a sequence of irrational numbers that must be determined numerically. Shown

below in Figure 3.7 is the denom(nt) function for the first 5 Target orbital periods. Notice the function evaluates to zero at each multiple of 2π as well as intermediate values between each multiple of 2π, though not quite centered at odd multiples of π.

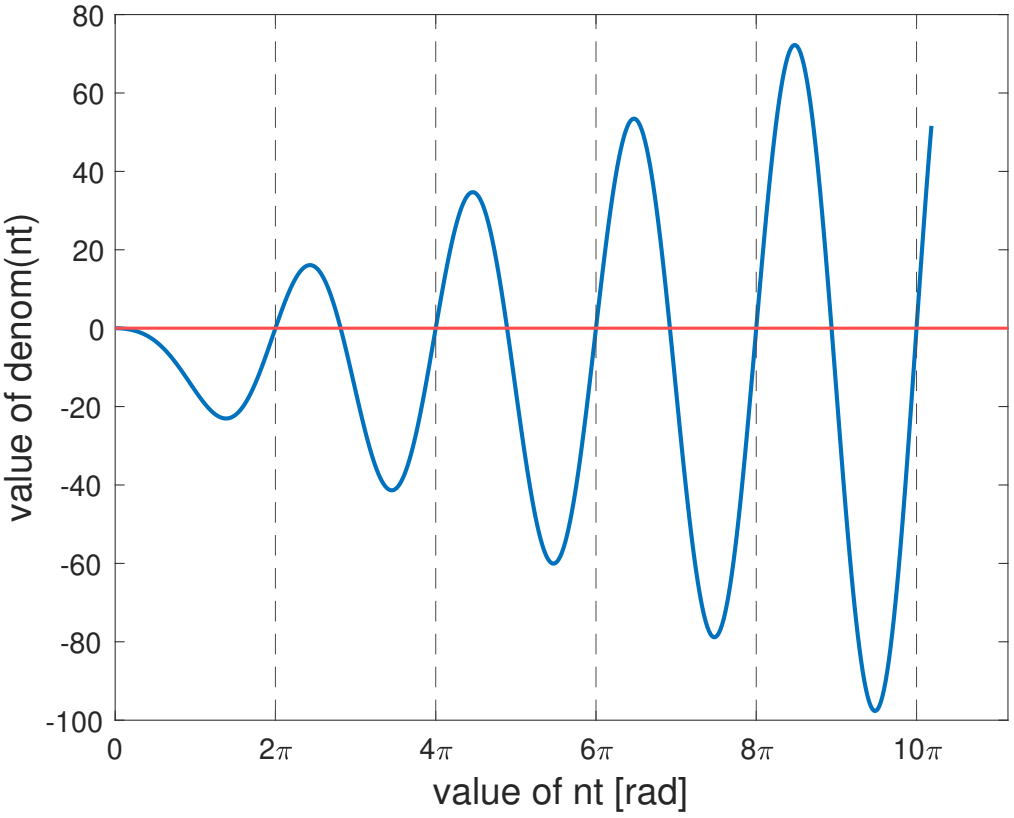

Figure 3.7: Values for denom(nt) During First Five Target Orbital Periods

The intermediate non-π-multiple irrational roots for this range of nt values were found numerically using the MATLAB fzero function, which uses a state-of-the-art Brent-Dekker algorithm [34]. In summary, the first 15 singularity values of nt occurring within the first 5 Target orbital periods are shown below in Table 3.1.

Table 3.1: First 15 Singular Values of *nt*

Index	Value of nt [rad]	Multiple of π
1	0	0π
2	3.141592653589793	1π
3	6.283185307179586	2π
4	8.838742844152041	
5	9.424777960769379	3π
6	12.566370614359172	4π
7	15.364261290786979	
8	15.707963267948966	5π
9	18.849555921538759	6π
10	21.747123605878745	
11	21.991148575128552	7π
12	25.132741228718345	8π
13	28.085001796594980	
14	28.274333882308138	9π
15	31.415926535897931	10π

As can be seen above in Table 3.1, the non-π-multiples approach the odd π multiples as the value of nt increases and the difference between them decreases. This is expected as the denom(nt) function becomes dominated by the secular $3\,nt\,\sin(nt)$ term, which has roots at odd multiples of π. The implications of this on the optimization problem are discussed in the following section.

3.5.2 Unconstrained Optimization of Two-Impulse Transfers

These singularities naturally form the boundaries for solution "brackets" such that within each bracket the cost function is continuous and differentiable. Furthermore, because the cost value rises to an infinite value at these bracket boundaries, the slope of the cost function just positive of a singularity is steeply negative and the slope of

the cost function just negative of a singularity is steeply positive. By inspection of a sample cost function with given start/end states and as claimed in the pertinent literature, the cost function is convex within each bracket and thus there exists a single minima in each bracket [25],[20]. A sample cost function is shown below in Figure 3.8.

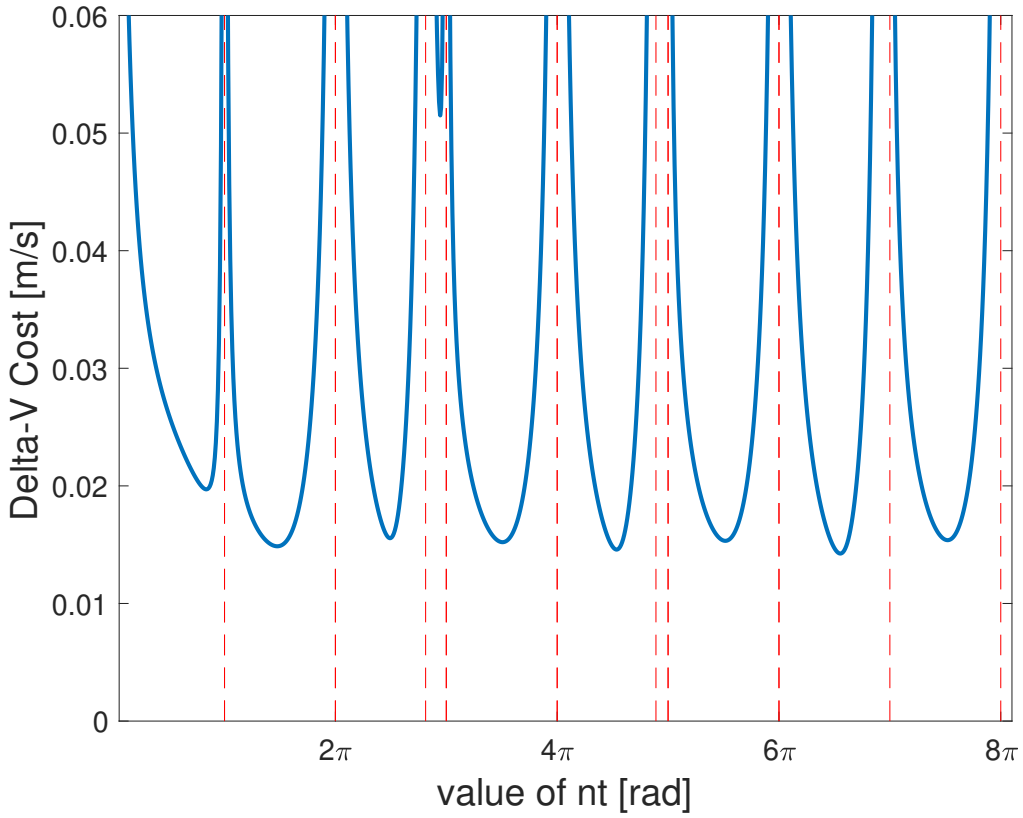

Figure 3.8: Sample Delta-V Cost Function Over First Four Target Orbital Periods for GEO Target with Singularities Shown in Red

The singularities and convexity within each solution bracket is apparent in Figure 3.8. This sample cost function is for a transfer around a GEO Target between an initial relative state of $\Delta\vec{R}_i = (-120, 50, 21)\ [m]$ & $\Delta\vec{V}_i = (-2, 20, 5)\ [mm/s]$ to a final position of $\Delta\vec{R}_f = (12, -3, 14)\ [m]$ with a terminal relative velocity of zero. This figure highlights how narrow the non-π-multiple solution brackets become and how

the solution cost is extremely high within these thin brackets. For this sample transfer, the minimum cost within the first thin bracket just before 3π is over twice the delta-V cost as minimum costs within the regular π-width brackets, and the minimum cost for the increasingly thinning non-π brackets do not appear in the figure at all. For this reason, only the first thin bracket just before 3π is considered by the solver, all increasingly thin later brackets are ignored to avoid wasting solver time exploring narrow brackets that yield expensive solutions. This figure also reveals that the local minima are not equivalent; for this sample transfer the local minima within the first solution bracket is distinctly more expensive than the local minima for the later solution brackets. The global minimum for the first four orbits occurs when nt is between 6π and 7π, though this is difficult to see in the figure.

The minima for each bracket is a solution to the first order necessary condition for optimality $\frac{\partial J}{\partial t} = 0$. To numerically determine solutions to this root-solving problem, a finite difference scheme is implemented to numerically approximate the partial derivative of the cost function with respect to transfer time $\left(\frac{\partial J}{\partial t}\right)$. The equation for the implemented two-point forward finite difference approximation for the first derivative is shown below in equation (3.27) [26],[49]. This finite difference approximation requires two function evaluations to approximate the derivative and is a first order approximation [26]. In other words, the truncation error of this approximation is proportional to the step size h that is used, commonly denoted as $\mathcal{O}(h)$ [26],[49]. The forward difference scheme is used instead of a higher-order method, such as the centered difference, to minimize function evaluations and thus save runtime in the solver; centered difference requires twice as many function evaluations to approximate the derivative [26],[49],[32].

$$\frac{\partial J}{\partial t} \approx \frac{J(t+h) - J(t)}{h} \tag{3.27}$$

To most effectively utilize the finite difference approximation, the step size h is selected such that the truncation error $\mathcal{O}(h)$ that arises from the finite difference approximation is of the same order as the numerical round-off error that arises from the limitations of computer precision when subtracting two nearly equal numbers. A computer's precision is quantified by its machine epsilon, commonly denoted as ϵ and defined as the smallest number such that $1 + \epsilon > 1$ is evaluated correctly. The value of ϵ can be computed with any system as $\epsilon = 7.0/3.0 - 4.0/3.0 - 1.0$, and for double precision it is $2^{-52} \approx 2.2204\text{e}{-16}$. To achieve this balance of truncation error and round-off error, a step size of $h = t\sqrt{\epsilon}$ is used [32],[49].

With bracketed solution regions and a numerical method for approximating the partial derivative in place, the problem is well suited for a bracketing method to solve for the roots of the numerically-defined partial derivative. The bisection method was utilized for this. The bisection method is a simple but robust root-solving algorithm for any function $f(x)$ that is continuous within the initial interval or "bracket" $[a, b]$, given that the initial bracket surrounds the desired root [5],[49]. A root is bracketed if the initial bracket is constructed such that one side evaluates to a positive value and the other evaluates to a negative value, such that $\text{sign}(f(a)) \neq \text{sign}\,(f(b))$ [5],[49]. This criteria is already satisfied within each starting bracket for this problem. As noted previously, due to the nature of the singularities at the bracket boundaries, the slope of the cost function just above a singularity (lower edge of bracket) is negative and the slope of the cost function just below a singularity (upper edge of bracket) is positive.

Each iteration of the algorithm works by evaluating the function at a new midpoint $x = (a+b)/2$ and comparing the sign of the new value $f(x)$ to the sign of $f(a)$ and $f(b)$ to determine which half of the current bracket the root resides within, and thus which edge to update [5],[49]. For example, if $f(a)$ is negative, $f(b)$ is positive, and $f(x)$ is positive, this means the root must lie within the interval $[a, x]$, so the upper edge of

the bracket is updated to be the midpoint (i.e., $b = x$) [5],[49]. Iterations continue until some stopping criteria is met or the midpoint evaluation $f(x)$ is exactly zero [5],[49].

For this implementation, it is known a priori that the lower side of the starting bracket will have a steep negative slope and the upper side of the starting bracket will have a steep positive slope, allowing for simplification of the standard bisection algorithm, which typically handles uncertainty regarding the signs of the function at the edges of the starting bracket. For this implementation, the stopping criteria is on a tolerance for when the slope of the cost function is near zero, or if the number of iterations exceeds the maximum limit. Once the stopping criteria has been reached, the last midpoint value is considered the optimal value. To avoid evaluating at the actual singularity points, the starting brackets are initialized to be 0.1% of the bracket's initial width away from each singularity.

The solution found within each bracket is only a *local* minima, the minimum delta-V cost for each evaluated bracket must be compared to determine the *global* minimum. The XML definition for the *guidance* model for each subsystem allows for customization of how many brackets to evaluate, enabling users to control the upper limit for how long assets are allowed to spend transferring. Only the brackets bound by the singularities occurring within 5 Target orbital periods can be considered in this global search, as any transfers longer than that are deemed "too slow" for the mission goal of rapidly completing numerous servicing tasks.

It is worth noting that there are a variety of alternative approaches to solving this single-variable nonlinear unconstrained optimization problem [5],[38],[49],[34]. The secant method was initially explored, but it was found to frequently diverge or cause the solution to "jump" to other brackets. Newton's method can offer more rapid convergence, but requires an additional finite difference scheme and several more function

evaluations to approximate the second derivative of the cost function, and can be less robust than the bisection method [5],[38],[49]. The state-of-the-art Brent's method is a hybrid approach that has the robustness of the bisection method and can attain the same rapid convergence as other less-robust methods [49],[34].

3.6 Delta-V Optimization of Path Constrained Two-Impulse Transfers

The previous section described the fundamental equations, assumptions, and algorithms necessary to set up and solve for the minimum delta-V unconstrained two-impulse transfer between any two relative orbital states. However, this formulation does not consider the path constraint to ensure that the servicing assets do not collide with the central space structure that requires the servicing operations. The actual structure and surrounding keep-out zone (KOZ) for a real space vehicle requires a complex geometric definition and consideration of the vehicle's attitude and articulations. For simplicity, this is modeled here as an ellipsoid centered at the origin of the Target's RIC frame. The ellipsoid is parameterized by its principal semi-axes (a, b, c) and is aligned with the RIC frame unit vectors. Thus, the size and shape of the ellipsoid can be specified, but the orientation must be aligned with the RIC frame.

This inequality path constraint is evaluated by assessing a configurable number of grid points along the transfer trajectory. Each point is evaluated using the standard cartesian equation for an ellipsoid centered at the origin and aligned with the axes, shown below in equation (3.28). If the left hand side of the ellipsoid equation (3.28) evaluates to less than 1, then the servicing asset is within the keep-out zone and thus the proposed transfer trajectory violates the path constraint. If (3.28) is satisfied for

all points along the transfer trajectory there will be no collision with the keep-out zone.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} \geq 1 \tag{3.28}$$

To determine a fuel-optimal transfer that satisfies this inequality path constraint, a "modified bisection" algorithm is used. First, the unconstrained solution is obtained using the method described in section 3.5. The unconstrained solution is then evaluated against the path constraint; if the solution trajectory is entirely outside of the keep-out zone then the unconstrained solution also satisfies the constraint and is returned. If the unconstrained solution fails the constraint check, the bisection method is utilized again to solve for the boundary of the path constraint. This is done by searching for the closest transfer time that is greater than the optimal time such that the transfer trajectory passes the path constraint check. This construction is based on the convex nature of the cost function and an empirical and intuitive observation that a longer transfer time corresponds to a wider relative path that keeps the servicing vehicle farther away from the central space structure, while a shorter transfer time results in a more direct path that is more likely to cut through the keep-out zone. A sampling of trajectory options for a given transfer with increasing time of flight (TOF) is shown below in Figure 3.9 to illustrate this empirical and intuitive claim. As is seen in the figure, increasing the time of flight leads to an increasingly wide relative motion trajectory, confirming this claim.

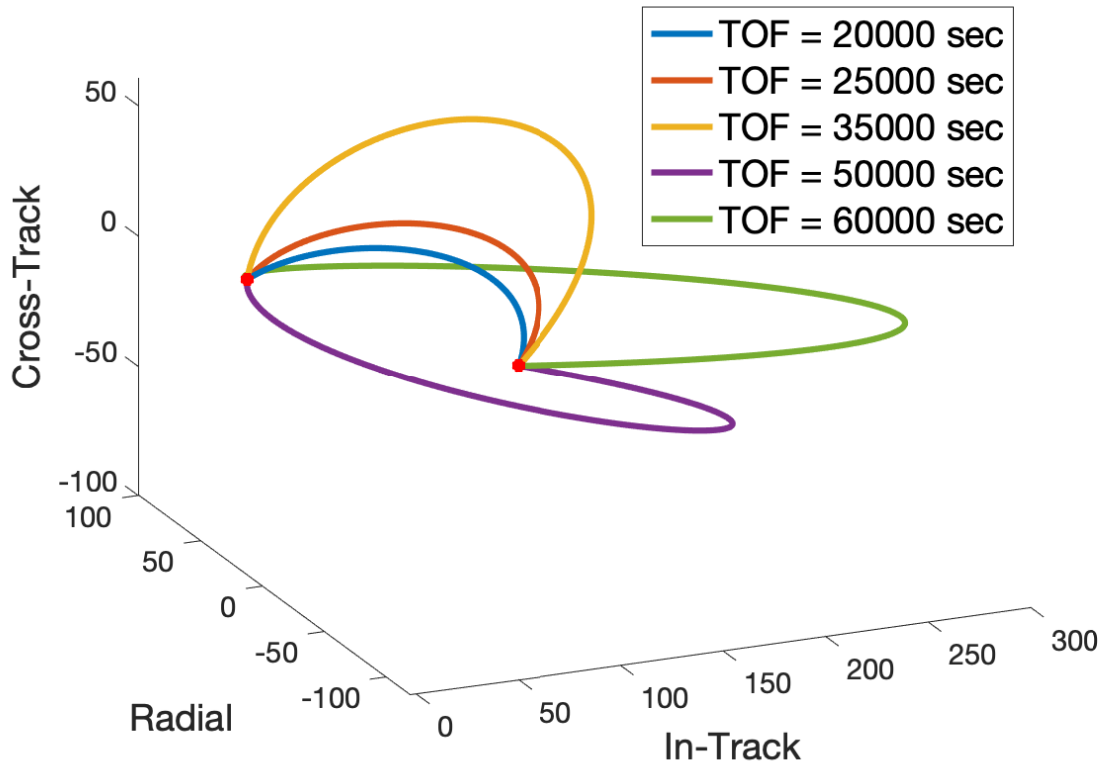

Figure 3.9: Sample Transfer Trajectories with Increasing Time of Flight

With this construction, the remaining solution region is any time between the optimal unconstrained transfer time and the upper edge of the current bracket (i.e., the next singularity in the cost function) under evaluation. Prior to starting the bisection algorithm, a coarse brute-force search is performed by discretizing the remaining solution region into eight sections to determine which eighth of the remaining solution region contains the critical point at which the path constraint first becomes feasible. If none of the discretized points result in a feasible solution that satisfies the path constraint, the bracket is considered invalid and no solution is returned. Once a feasible point is found, the selected section is optimized using the bisection method to determine the shortest feasible transfer time.

This second round of bisection method is no longer root solving the derivative of the cost function, it is solving for the status of the constraint check. If the new midpoint satisfies the constraint, then the upper limit of the bracket is reduced to this midpoint; if the new midpoint fails the constraint, then the lower limit of the bracket is raised to this midpoint. The stopping criteria is on a tolerance for when the width of the solution bracket is sufficiently small or the number of iterations exceeds the maximum limit. Once the stopping criteria has been reached, the last upper limit of the bracket, where the solution is known to be feasible, is considered the optimal value. Again, the solution found within each bracket is only a *local* minima and all of the brackets defined by the configured number of brackets to examine within the first 5 Target orbital periods are evaluated to determine the *global* minimum. If all of the evaluated brackets were found to be invalid, then the proposed servicing activity cannot be performed by the asset in its current state and the *guidance* model returns False for the canPerform.

3.7 Multi-Objective Optimization

With the mission goal to rapidly perform numerous servicing operations using a cost-effective quantity of servicing assets, transfer trajectories that minimize time as well as fuel are of value. This is achieved by augmenting the minimum delta-V cost function (3.12) to include consideration for minimizing transfer time. This can be implemented directly into the cost function as shown below in Equation (3.29), where w defines the relative weighting assigned to transfer time in the cost function.

$$J = \sqrt{(\Delta V_{X_i})^2 + (\Delta V_{Y_i})^2 + (\Delta V_{Z_i})^2} + \sqrt{(\Delta V_{X_f})^2 + (\Delta V_{Y_f})^2 + (\Delta V_{Z_f})^2} + wt \quad (3.29)$$

Setting w to zero reduces the cost function to the original minimum delta-V formulation (3.12). The addition of the wt term does not introduce any new singularities. This augmented cost function is still convex, as the second partial derivative $\frac{\partial^2 J}{\partial t^2}$ of the wt term is zero. Thus, all of the algorithmic and numerical formulations described in the previous sections can be immediately utilized with this new multi-objective cost function, and control of the weighting w allows users to control the behavior of each servicing asset to best fit specific mission objectives and constraints related to fuel and time.

3.7.1 Modeling of Fuel Constraint

The computed optimal delta-V transfer trajectory solution is converted from delta-V cost to fuel expenditure using the famous Tsiolkovsky rocket equation and the 2-norm of the delta-V vector, as shown in (3.30) [8]. The use of the 2-norm corresponds to vehicle configurations in which the transfer delta-V impulse is obtained with a single large thruster or a set of thrusters aligned along the delta-V vector.

$$\Delta m = m_i - m_f = m_i \left(1 - \exp\left(\frac{-\Delta V}{Isp\ g0}\right)\right) \quad (3.30)$$

This fuel expenditure is used to update the value of the available fuel mass tracked by the system state. The available fuel mass is a constraint on the subsystem model; the HSF scheduler will not allow any events which cause an asset's fuel mass to become

negative, regardless of the value returned for the canPerform. As mentioned previously, the construction of this constraint is also critical to ensure that the *guidance* models are all evaluated prior to the *collisionAvoidance* model.

3.8 Docking, Servicing, and Departure

The docking, servicing, and undocking operations for a servicing satellite require sophisticated mechanisms, robotics, 6-degrees-of-freedom vehicle estimation and control, sensors, and sensor processing software. For simplicity, these complex operations are largely ignored in the present research and the entire process is only parameterized by the type of servicing operation to be performed and the total time required to complete docking, servicing, and undocking.

The functionality of each *tool* subsystem is parameterized by a single dictionary defining which task types the servicing asset can perform and how long it takes the servicer to perform each task type. This abstracts any actual hardware and software the servicers may possess and reduces the model to high-level requirements such as "Servicer 1 shall be capable of performing task type A within 3 hours." If an asset cannot perform the necessary operations for a certain task type, then this task type is not included in the dictionary. The canPerform of the *tool* subsystem model will return True if the proposed task type is a key in the tooling dictionary and will return False if it is not. These task types are assigned to each of the targets in the target XML input file, thus consistent naming syntax is required. The required servicing times in the tooling dictionary are accessed in the *tool* model and assigned to the system state so that the *guidance* model can access this data for its calculations.

The *guidance* model accesses this servicing time data directly from the "bulletin board" to compute the required fuel expenditure for the constant burns required to maintain the relative state difference between the servicing asset and the central space structure. Due to the orbital motion that both vehicles are experiencing, only a servicing asset positioned directly ahead or behind the central structure in a V-bar hold will naturally remain fixed in the relative RIC frame; any other relative positions will naturally drift according to the CW equations (3.1) & (3.2). In order to compute the necessary control acceleration to maintain a constant relative position during servicing operations, consider the CW differential equations with the addition of a control acceleration $\vec{C} = (C_x, C_y, C_z)$, as shown below in (3.31), (3.32), (3.33).

$$\ddot{x} - 3n^2x - 2n\dot{y} = C_x \tag{3.31}$$

$$\ddot{y} + 2n\dot{x} = C_y \tag{3.32}$$

$$\ddot{z} + n^2z = C_z \tag{3.33}$$

We wish to choose $\vec{C}$ such that for any given relative state (x, y, z) with zero relative velocity, the relative acceleration will be zero, thus ensuring a constant hold. These control accelerations are determined by inspection of (3.31), (3.32), (3.33). A control acceleration of $C_x = -3n^2x$ results in zero acceleration in x, given that y velocity is zero as well. No control acceleration is required in y; for a zero x velocity the y acceleration is naturally zero. A control acceleration of $C_z = n^2z$ results in zero acceleration in z. In summary, the control acceleration required for a constant hold is shown below in (3.34), (3.35), (3.36).

$$C_x = -3n^2x \tag{3.34}$$

$$C_y = 0 \tag{3.35}$$

$$C_z = n^2z \tag{3.36}$$

Assuming that this precision continuous position control is achieved using separate thrusters, the fuel cost for maintaining this hold is determined by the sum of the absolute values of the acceleration components, or the 1-norm of the acceleration vector. The 1-norm of the acceleration can be used with the required servicing time to determine the delta-V that would be achieved in the absence of gravity, and this theoretical delta-V value can be used in the Tsiolkovsky rocket equation (3.30) to determine the fuel mass required for the constant hold. These delta-V calculations are shown below in (3.37), (3.38). As is clear in these equations, the fuel cost is directly proportional to the x and z components of the relative state vector, and thus constant holds around a KOZ that is elongated close to the y axis such that the x and z coordinates remain small requires less fuel expenditure.

$$\|\vec{C}\|_1 = |-3n^2x| + |n^2z| = n^2(3|x| + |z|) \tag{3.37}$$

$$\Delta V = \|\vec{C}\|_1 \Delta t = n^2 \Delta t(3|x| + |z|) \tag{3.38}$$

After completing the servicing maneuver, the servicing asset maintains the constant burn to hold the current position so that it does not drift into the central space structure or drift away from the central space structure. Drifting into the central space structure would be an undesirable collision and drifting away from the central

space structure could lead to expensive maneuvers being required to transfer back to another servicing target location. This approach is based on an assumption that the servicing assets will not be left un-tasked for long periods of time, such that the fuel costs for maintaining this constant hold is not a major burden. The servicer must maintain this constant hold for the remaining portion of the fundamental simulation time step between completion of the servicing activity and the end of the time step that the servicing activity ends within. This required hold time is summed with the required servicing time to determine the total constant burn time used in the fuel cost calculation described by (3.37), (3.38), (3.30). This servicing and idling constant hold fuel expenditure is used in summation with the fuel expenditure for the two-impulse transfer trajectory to fully define the required fuel mass usage for any given asset/target pairing.

In addition to this constant burn to hold the position while waiting for the next fundamental simulation time step, the same constant burn hold is implemented for any time step in which the servicing asset is left un-tasked/passive that does not occur while the asset is on the starting NMC. The fuel cost for this again follows the same fuel cost calculation described by (3.37), (3.38), (3.30), this time with the Δt hold time being the length of the fundamental simulation time step. This fuel cost is again used to update the value for the available fuel mass tracked by the system state, though this is now for a new event in which the asset is passive for this new time step.

Note that various alternatives exist for handling this post-servicing idling. For example, the assets could perform a series of maneuvers to achieve a safe-drift or safe-hold trajectory, such as an R-bar drift or hop trajectory that safely drifts the servicer away from the central space structure, an NMC about the central space structure, or a static V-bar hold. The constant burn to hold the current position is implemented

for model simplicity and to ensure the servicer remains close by for its next transfer. Additionally, if the asset is immediately tasked again on the following simulation time step, any substantial maneuver would likely cost more fuel than the small constant burn to briefly maintain the constant hold. Furthermore, maintaining a constant position while idling is beneficial for the use of time delays to achieve collision avoidance, as described in the following section.

3.9 Collision Avoidance

Collision avoidance is critical for any fleet or swarm of vehicles, and is especially critical on orbit where orbital debris is a growing concern and any impact could be catastrophic for the swarm or the central space structure receiving the servicing. Collision avoidance is achieved in the system model through evaluation of the *collisionAvoidance* model as part of the "watchdog" asset. The *collisionAvoidance* model, which is evaluated last after all other subsystem models have been evaluated, analyzes the proposed NMC drifts, transfers, idling holds, and servicing holds computed by the *guidance* and *tool* models for the given asset-target pairings to ensure that all of these trajectories are consistent and safe together. The necessary state information is made available to the *collisionAvoidance* model through use of the data published to the system state "bulletin board". The model first checks that the scheduler did not double-assign more than one asset to the same target to ensure unique pairings. This is necessary because the HSF scheduler does not enforce unique tasking as a constraint on its own.

Next, all assets (both active and passive) are assessed for collisions along their trajectories across the current fundamental simulation time step, the length of which is set

in the scenario XML file. These trajectories are assessed using a uniform grid with a configurable number of points, and at each burn epoch when an asset begins or ends its transfer trajectory. These grid and burn points are assessed by computing the distance between each asset at the given instant and checking that the distances are all greater than the minimum safety distance defined in the system model XML file. Asset states at the given instant are determined through careful extraction of the last defined state and the current mode from the "bulletin board" and propagation using the CW equations (if the asset is moving). For model simplicity, the safe-distance comparison does not directly consider state uncertainties, nor how state uncertainties may grow over time. Thus, this is most applicable for mission concepts in which state uncertainty remains relatively constant. For example, if there are frequent observations and measurements to routinely maintain the relative state estimates and keep state uncertainties small.

After assessing all assets' trajectories across the fundamental simulation time step, the *collisionAvoidance* model assesses the safety of all remaining active assets whose transfer and servicing activities extend beyond a single fundamental time step. These remaining active assets are again evaluated using an extension of the same uniform grid and at any burn epochs that occur beyond the first fundamental time step. Trajectories are determined and evaluated for any asset that is active in the subsequent fundamental time step until a fundamental time step is found in which less than 2 assets are still active. An asset's trajectory is set during a given fundamental time step if the asset is active at any point in the time step, but if it is not active at all during a given time step then it can be tasked to perform some other transfer during subsequent schedule generations, and these resulting future trajectories will be evaluated again by the *collisionAvoidance* model on the subsequent time steps.

If at any evaluation epoch a collision or unsafe spacing between any assets is detected, the model returns False for the canPerform method. If the full evaluation of all trajectories passes all conjunction evaluations, the canPerform method returns True and the proposed event is added to the tree of possible schedules.

While the *collisionAvoidance* model evaluates if a given set of trajectories pose a collision risk, the actual avoidance is achieved by leveraging the HSF scheduler and tuning of the fundamental simulation time step duration to achieve the appropriate delays in the trajectories to eliminate the collisions. The premise for this is as follows: consider a set of trajectories in which there is a single collision. The scheduler will not accept this event with this set of trajectories due to failure of the *collisionAvoidance* canPerform, but it will accept the very similar event in which one of the colliding objects remains passive and thus does not follow the same transfer trajectory. On the very next time step, the scheduler can now task this asset to service the same asset it previously could not due to the collision, and if the time delay is long enough the trajectories will no longer cross paths at the same time and thus there will no longer be any collisions. This notion of introducing time delays via the HSF scheduler is how the collisions detected with the *collisionAvoidance* model are avoided and feasible mission schedules are obtained. The fundamental time step that the scheduler uses must be reasonably short such that this approach does not waste too much time or fuel while idling, however it must not be too short or else the runtime of the simulation will become impractical as the tree of possible schedules rapidly branches. It is recommended to first analyze the system with a single fundamental time step to determine appropriate time step length and pruning value for the system.

3.10 Methodology Summary

In summary, this chapter described the technical work performed, including formal definition of the mission concept, all equations and algorithms utilized to implement the system model, and all modifications made to the Horizon Simulation Framework. The mission concept consists of several small servicing assets on an initial NMC maneuvering about a large central space structure to rapidly perform numerous servicing operations. Control of the system is distributed across the individual assets to uniquely determine their own optimal trajectories and a theoretical "watchdog" to monitor the collective movements and enforce collision avoidance.

The servicing fleet system model is implemented with a *tool* and *guidance* subsystem model for each servicing asset and a "watchdog" asset with a *collisionAvoidance* subsystem model. The *tool* subsystem models how an asset's tools interact with the task types and defines the servicing time needed for the target operation. The *guidance* subsystem computes and models the optimal two-impulse transfer trajectory to the proposed target, and performs fuel consumption calculations for the impulsive transfers and constant holds necessary for servicing and idling. Optimal trajectories are determined using the CW equations, a multi-objective cost function, an ellipsoid KOZ path constraint, and a modified bisection algorithm for parameter optimization of the optimal transfer time. The terminal rendezvous, docking, servicing, and departure operations are abstracted in this work and modeled simply as a constant burn fuel expenditure to maintain the constant relative state. Servicing assets idle in the place of their last completed operation while waiting to be re-tasked to service a new target location. The *collisionAvoidance* subsystem collectively evaluates all asset trajectories to determine whether the trajectories pose a collision risk, as parameterized by a constant minimum safety distance. Thus, the target locations

should be further apart than this minimum distance so that nearby operations can be performed simultaneously. The HSF scheduler is utilized with small fundamental time steps to achieve collision avoidance by scheduling assets to be passive for some time steps in order to introduce time delays in the trajectories.

Critical alterations, extensions, and "tricks" were implemented in the main HSF codebase and subsystem models to achieve this novel system model. The ability to pacify the servicing assets was achieved by introducing an empty/null task that is assigned a fictitious "EmptyTarget" and a completion score of zero. The inclusion of this "EmptyTarget" requires each of the subsystem models to handle this proposed target. The ability to completely pacify the "watchdog" asset was achieved by implementing an IsTaskable member variables for the Asset.cs class to control if a given asset is taskable or not. To utilize the analytical solution to the CW equations and perform trajectory optimization, HSF was extended to allow the dynamicState for an asset to be null, enabling the *guidance* subsystem model to fully control the state definitions without relying on the ODE infrastructure typically required for dynamics modeling with HSF. Lastly, some "tricks" were implemented to side-step the dependency feature of HSF and directly access state data using the "bulletin board", an approach that was necessary due to current limitations of the dependency feature.

This work culminates in an integrated mission modeling and scheduling tool capable of assessing servicing fleet architectures and generating optimal mission schedules. The following chapters detail the application of the tool to simplified mission scenarios to validate the system model and the HSF alterations are operating correctly, the results obtained from analysis of a design reference mission (DRM), discussions regarding further applications of the tool, and concluding remarks regarding the journey to achieve this tool and possible future work.

Chapter 4

RESULTS

The previous chapter described the work performed to achieve the integrated servicing fleet mission modeling and scheduling tool. The current chapter contains the results and pertinent discussion regarding the application of this tool to simplified verification scenarios and to a full DRM scenario to explore "proof of concept" mission feasibility for a servicing fleet.

The 3 simplified scenarios studied for validation of the system model and the HSF extensions are enumerated below.

1. Highly Simplified Model to Verify Task Assignment & Tool Model
2. Two-Asset Two-Target Unconstrained Scenario to Verify Unconstrained Trajectory Optimization & Collision Avoidance
3. Single-Asset Constrained Scenario to Verify Constrained Multi-Objective Trajectory Optimization

4.1 Validation I: Task Assignment

The highly simplified system model consists of servicing assets with only the *tool* subsystem model and a simplified *collisionAvoidance* model to verify that task assignment and servicing time is done correctly with the implemented models and HSF extensions. This verification scenario is composed of 2 assets (Asset A & Asset B), 3 targets (Target 1, Target 2, Target 3), and 2 fundamental simulation time steps.

The targets are assigned prime numbers for their task completion values so that the combination of their scores in the final schedule values are unique, allowing for verification of the schedule scores. The *collisionAvoidance* model is configured with the input model XML file to only evaluate for task collisions, thus preventing multiple assets from servicing the same target at once. The tooling dictionary for the *tool* models and the task types for the targets are varied to validate the behavior of the *tool* model and HSF scheduler.

For the first variation examined, the tooling dictionary is configured to be the same for both assets and to match the task type for all 3 targets. This isolates the HSF scheduler and task collision checking in *collisionAvoidance* to enable a combinatorics verification that the appropriate tasks are being proposed by the scheduler and approved by *collisionAvoidance*. The details of this combinatorics analysis is presented in Appendix A, which shows detailed explanation of how each schedule variation is properly attributed to the expected valid task combinations proposed by the HSF scheduler.

Though tedious, this combinatorics activity validates that the HSF extensions to include an "EmptyTarget", an inactive asset, and the task collision evaluation in the *collisionAvoidance* model are functioning properly.

Next, the tooling dictionaries were made asymmetric such that Asset A can service Targets 1 and 2 while Asset B can only service Target 3. Additionally, the servicing times in the tooling dictionaries were made distinct, though both were kept within the length of the fundamental time step.

As expected, by introducing this tooling availability constraint that limits the possible pairings of assets and targets, there are far fewer valid schedules generated. The state logs outputted from the *tool* model confirm that the servicing time is assigned

correctly from the tooling dictionary, with the servicing time for Asset A being posted to its *tool* model bulletin board twice (corresponding to Target 1 & Target 2) and the distinct servicing time for Asset B being posted to its *tool* model bulletin board once (corresponding to Target 3). Similar combinatorics analysis was again performed to account for all of the schedule variations and re-verify asset assignments, but the details are omitted here for brevity.

4.2 Validation II: Unconstrained Optimization & Collision Avoidance

The second validation scenario includes the addition of the *guidance* subsystem model and full utilization of the *collisionAvoidance* subsystem model for analysis of a simplified DRM in which there are only 2 assets (Asset 1 & Asset 2) tasked with servicing 2 targets. The central space structure has a KOZ of zero size to eliminate the path constraint and the cost function is formulated with zero weighting for time of flight. This isolates the unconstrained solver and enables validation of the unconstrained fuel-optimal trajectories with direct comparison to minimums obtained from delta-V cost curves, such as presented in Figure 3.8.

The scenario is initialized with the 2 servicing assets spaced apart on the initial shared NMC. The *tool* models and target task types are configured such that each asset can only service a single target, thus prescribing the expected task pairings for valid schedules. The *guidance* models are configured such that only the first solution bracket is explored, simplifying the scenario further and ensuring the transfers occur relatively quickly. The initial states along the NMC and the target locations are set such that the computed optimal trajectories pass close by each other but the target locations are further apart than this closest approach distance. The minimum safety

distance is varied to exercise and validate the functionality of the *collisionAvoidance* model and HSF scheduler to achieve collision-free schedules.

For the first variation examined, the minimum safety distance for the *collisionAvoidance* model is set to zero, allowing the near-miss trajectories to occur. This fully isolates the *guidance* optimization solver. The impulsive transfer delta-V vectors computed by the solver and published to the "bulletin board" are accessed for post-processing comparison with an external MATLAB tool to verify the optimality and accuracy of the *guidance* model. The optimal trajectories, as computed by the external MATLAB tool, are shown below in Figure 4.1.

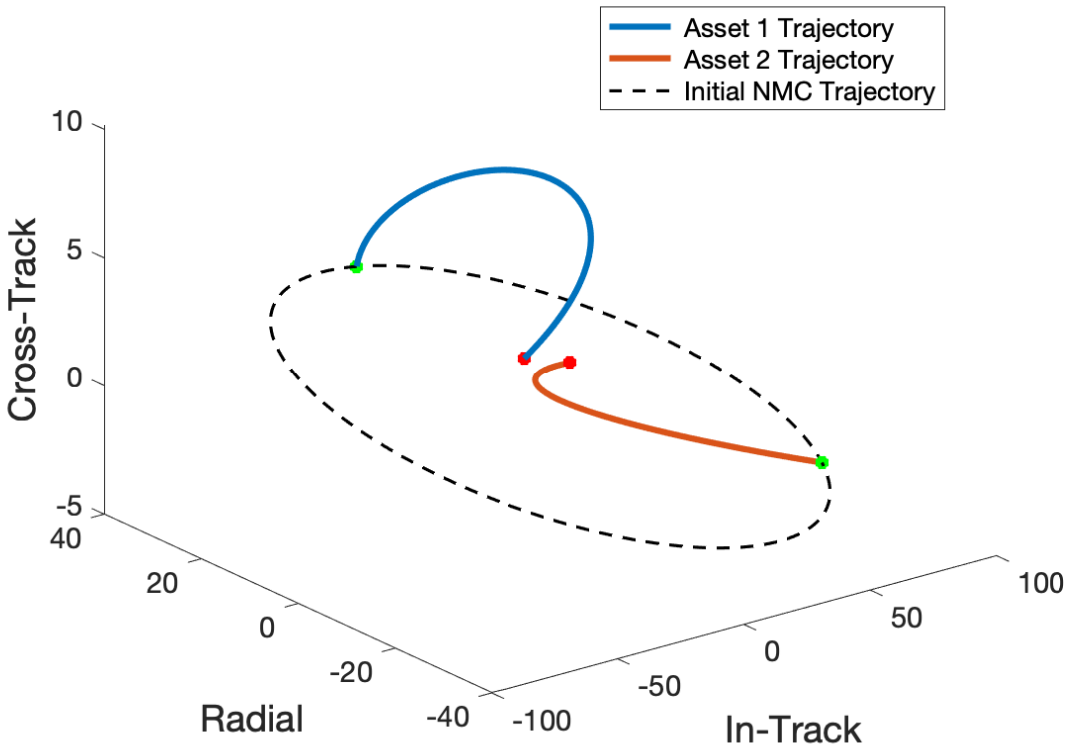

Figure 4.1: Transfer Trajectories for Simplified DRM

The starting points for the assets are shown in green and the target end points are shown in red. Notice that both starting green points share the common initial NMC trajectory. These optimal trajectories are determined by utilizing MATLAB's fminunc() method to determine the optimal transfer time of flight using an initial guess found from examination of the numerically-defined delta-V cost function. The MATLAB fminunc() method utilizes the BFGS Quasi-Newton method with a cubic line search procedure [33]. The cost functions for each transfer trajectory are shown below in Figures 4.2, 4.3, along with the optimal solution found using MATLAB and the bisection algorithm implemented in the *guidance* model.

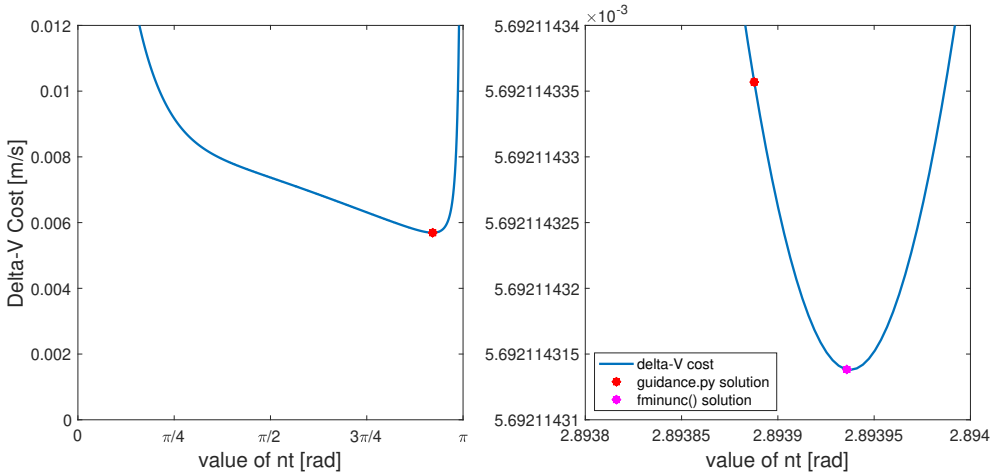

Figure 4.2: Delta-V Cost Function & Solutions for Asset 1

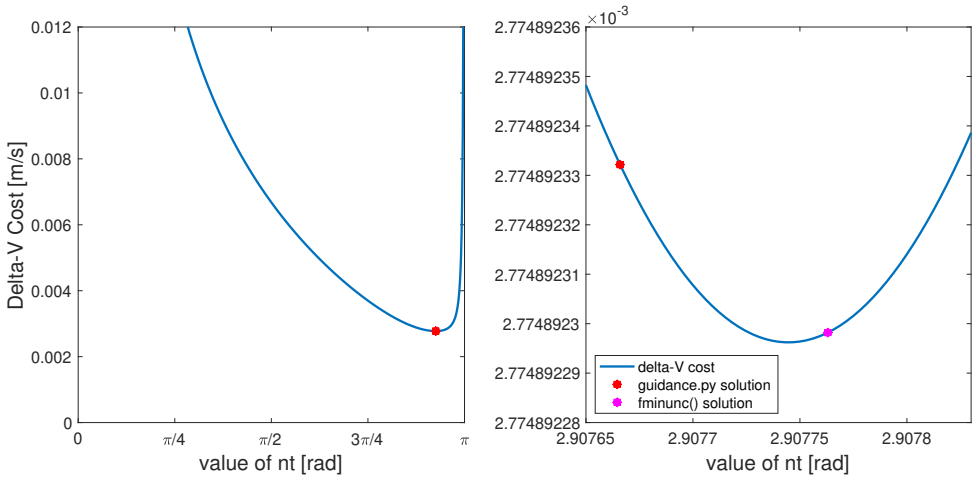

Figure 4.3: Delta-V Cost Function & Solutions for Asset 2

As shown in the figures, the bisection algorithm implemented in the *guidance* model adequately finds the minimum delta-V cost solution. As shown in the dramatically zoomed-in right-side subplots, there is some discrepancy between the MATLAB fminunc() and *guidance* model solutions. These differences amount to less than 1 second difference in time of flight, corresponding to delta-V cost differences well under a nanometer per second. Notice that for both transfers, fminunc() is slightly closer to the true minima. These small discrepancies could be reduced further by decreasing the exit criteria tolerances for the *guidance* bisection algorithm and MATLAB fminunc(), but this would cause an increase in iterations and thus runtime. The impulsive transfer delta-V vectors computed by the solver and published to the "bulletin board" match the solution from the MATLAB fminunc() within a micrometer per second.

These near-perfect matches between the fuel-optimal unconstrained transfer trajectories computed by the *guidance* model and the MATLAB equations and fminunc() solver validate the accuracy and optimality of the *guidance* model.

Next, the minimum safety distance for the *collisionAvoidance* model is set such that the closest approach distance is within the minimum safety distance, but the tar-

get locations are outside of the minimum safety distance. This is achieved with a minimum safety distance of 10 meters. As expected, this causes failure from the *collisionAvoidance* canPerform and there are no valid schedules in which both targets can be serviced with this single time step. Now the optimal schedule is the one in which the higher-valued target is serviced. As expected, the optimal transfer time and impulses logged for this single transfer are exactly the same as computed when the minimum safety distance was set to zero.

Allowing for a second time step in the simulation enables the HSF scheduler to introduce time delays by assigning an asset the "EmptyTarget" for the first time step so that the objects no longer occupy the same space at the same time, and thus no longer trigger failure from the *collisionAvoidance* model. As expected, this is ineffective if the time step is too small, such as only 10 minutes. A time step length of 3 hours is effective (for this specific scenario) for introducing a sufficient time delay that enables the HSF scheduler to determine a feasible schedule. As expected, HSF now produces 2 variations of the optimal schedule, one in which Asset 1 is active on the first time step and another in which Asset 2 is active on the first time step. As expected, the transfer scheduled to begin on the first time step is identical to the transfer computed when the minimum safety distance was zero. The collision-free trajectories for which Asset 2 is active on the first time step are shown below in Figure 4.4. Notice that Asset 1 at first follows the initial NMC trajectory before transferring to the target location.

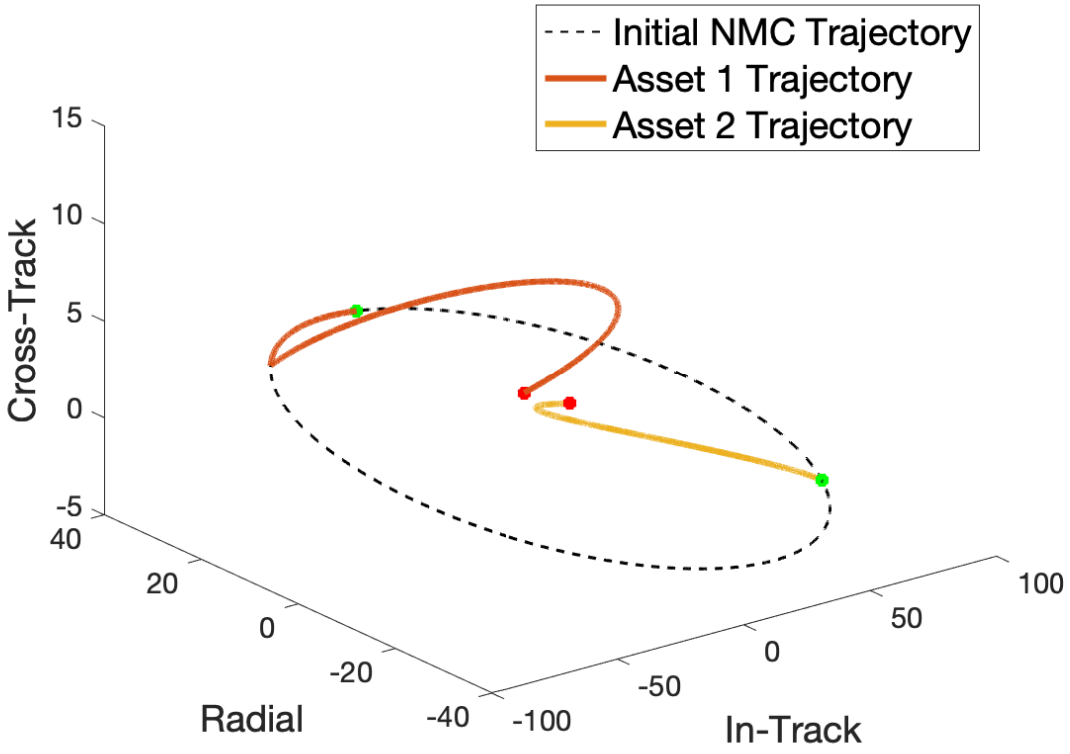

Figure 4.4: Collision-Free Transfer Trajectories

4.3 Validation III: Constrained Multi-Objective Optimization

The final verification scenario consists of a single asset with a non-zero keep out zone (KOZ) ellipsoid for the central space structure, designed to isolate the constrained solver. This scenario includes experiments with the simulation time span and time of flight weighting in the multi-objective cost function to verify the functionality of the constrained multi-objective trajectory optimization solver.

First, consider again the unconstrained single time step scenario with negligible KOZ and minimum safety distance, as shown in Figure 4.1. Considering just Asset 1 and setting the KOZ to an ellipsoid with semi-axes of [10m, 10m, 20m], the minimum-fuel

unconstrained trajectory now fails the KOZ check, thus activating the constrained solver. As expected, the constrained solver succeeds in finding a feasible transfer trajectory that maneuver around the KOZ, but requires a slightly greater transfer time and delta-V cost. The delta-V cost increased from 5.692 mm/s up to 5.867 mm/s and the transfer time of flight increased from 40,193 seconds to 41,573 seconds. Both trajectories and the non-zero KOZ are shown below in Figure 4.5. As can be seen in the figure, the unconstrained trajectory cuts through the KOZ volume, while the constrained trajectory takes a slightly wider arc that passes outside the KOZ volume. This validates the functionality of the modified bisection algorithm implemented to solve for an optimal trajectory subject to the KOZ path constraint.

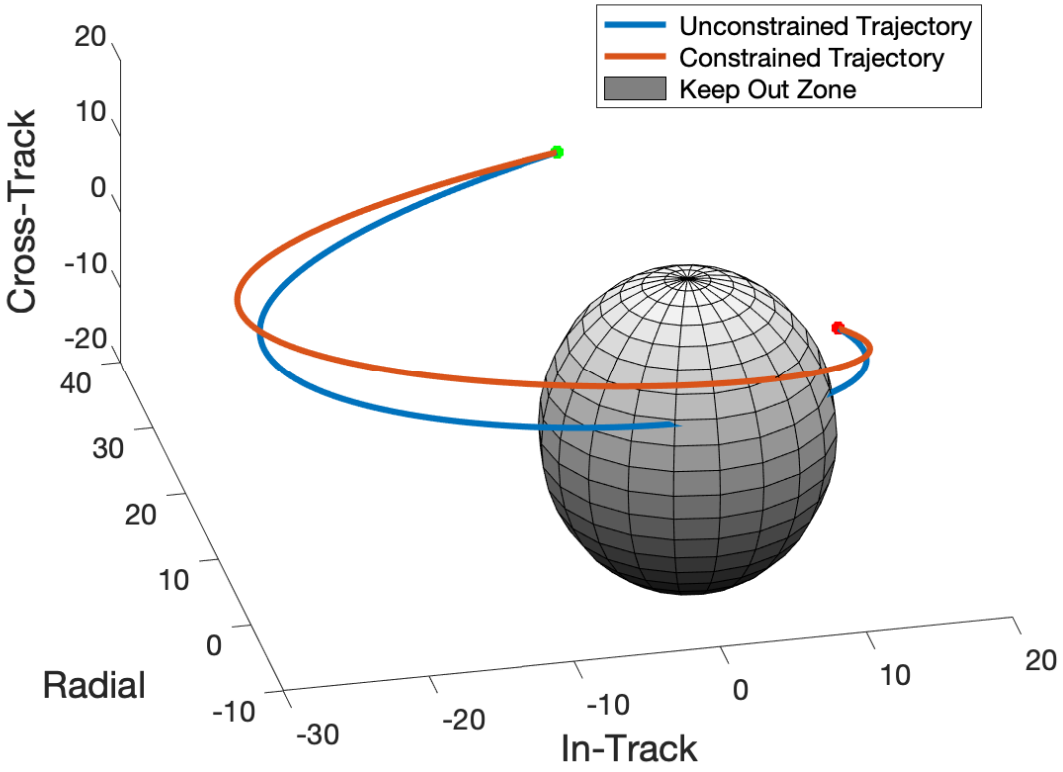

Figure 4.5: KOZ Trajectories

Next, consider reducing the KOZ volume back to zero and adding a second target location to be serviced. With the time of flight cost function weighting set to zero, to achieve this unconstrained 2-target servicing schedule, 2 fundamental time steps are required and the duration of each must be at least 11 hours. The end of the final servicing occurs 20 hours and 19 minutes after the start of the simulation. The ability of the *guidance* model to compute and perform multiple transfers and servicing activities for a single asset verifies that the subsystem model is functioning properly.

The time of flight cost function weighting can be utilized to achieve this schedule in less time. By using a time of flight weighting value of 1.5×10^{-6}, both servicing tasks can be completed in only 3 hours and 13 minutes. As expected, this weighting forces an increased delta-V cost, which increases from a total delta-V cost of 6.982 mm/s up to 19.442 mm/s.

Lastly, a non-zero KOZ volume is utilized to investigate the interaction between the KOZ and the cost function time of flight weighting. For trajectories that are KOZ-constrained, the time of flight weighting cannot drive the solution to a shorter transfer trajectory without violating the KOZ path constraint. This interaction is exemplified with this current scenario; the *guidance* model is able to compute an unconstrained transfer trajectory to Target 1 that utilizes the time of flight weighting, but is KOZ-limited for the second transfer between Target 1 and Target 2. The first transfer and servicing is achieved in under 2 hours, but the second transfer and servicing requires nearly 10.5 hours. Both trajectories and the non-zero KOZ are shown below in Figure 4.6. The asset starting point and 2 target locations are annotated in this figure to better represent the 3-dimensional trajectories in a 2-dimensional figure.

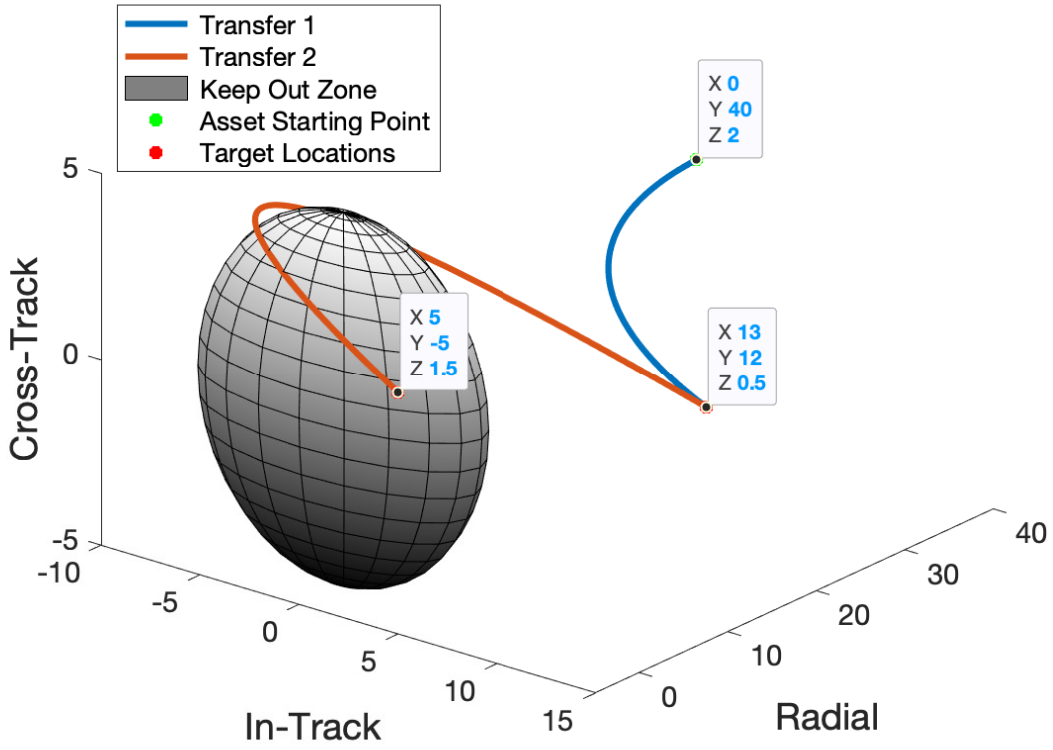

Figure 4.6: KOZ-Constrained Multi-Objective Trajectories

4.4 DRM Analysis

The design reference mission (DRM) analyzed is a full mission scenario with 3 servicing assets and 12 targets dispersed just outside of a large ellipsoid to represent the KOZ volume of the central space structure. The KOZ ellipsoid is defined with semi-axes of (70m, 120m, 50m). The 12 target locations were dispersed around the KOZ roughly 5m outside of the surface of the KOZ volume. These locations were determined first as 36 uniformly-spaced locations 5m outside the KOZ surface, then manually down-selected to 12 candidate locations, and finally rearranged slightly using a random number generator. The 12 targets are distributed evenly across 3 different servicing task types (i.e., 4 targets of each type), that require 20 minutes

to 1 hour of servicing time to complete. The task completion values were arbitrarily set to values between 8 and 12, with a maximum possible schedule value of 124. The assets each have 2 of the 3 tooling capabilities necessary for the 3 different servicing task types. Each asset has a 180kg dry mass, 20kg of fuel, and 200sec Isp thrusters, equating to a delta-V budget of roughly 200 m/s, as computed with a re-arrangement of (3.30). The minimum safety distance is 20m, which is less than the closest spacing between any 2 target locations. The initial setup for this DRM is shown below in Figure 4.7.

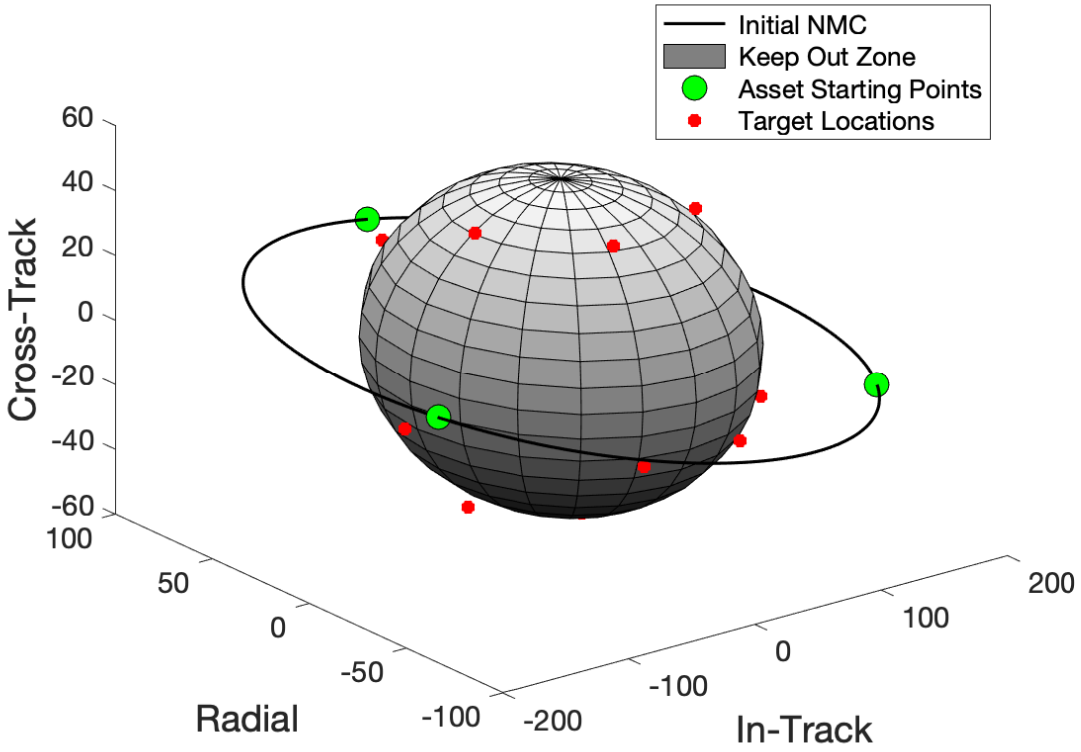

Figure 4.7: DRM Setup

Through initial testing of a single time step, it was determined that 200m/s is ample delta-V available; to service 12 targets with 3 assets in a short period of time it is appropriate to expend excess fuel to accomplish transfers quickly. Thus, the *guidance*

model is configured to only examine the first solution bracket and utilize a relatively large time-of-flight (TOF) weighting of 0.01. As expected, the optimal schedule with only a single time step was to service 3 targets with the maximum defined task completion value of 12, leading to an optimal schedule value of 36. Examining this optimal schedule and the 3 next sub-obtimal schedules further reveals that tasks take from 2000 seconds to 10 hours for completion (transfer & servicing), with the longer tasks being KOZ-limited (i.e, increasing TOF weighting will not decrease transfer time). A total of 622 valid schedules were generated with this single time step, and the runtime was less than 1.5 minutes. This timescale, number of schedules, and runtime information informs how long step sizes and how many steps should be in the full-scale simulation such that all tasks can be completed on time without requiring excessive runtime. This is the recommended approach for developing simulations with this tool: first examine the results of single time step simulation to determine appropriate time of flight weighting, schedule pruning parameters, time step parameters, and expected runtime.

Next, the simulation was expanded to 4 time steps of 10,000 seconds each. Ideally, the solver would be able to determine optimal schedules such that each asset can perform a full transfer and servicing activity within each 10,000 second time step and thus quickly complete all 12 tasks within the 4 time steps. Since there are only 3 assets, collisions should be relatively rare and collision avoidance should be relatively easy. Thus, the schedule pruning was such that only the 100 best performing schedules are kept after each time step.

This 4-step simulation requires 12 minutes of runtime and obtains schedules which accomplish nearly all of the servicing tasks. The simulation yielded 5 variations of the best performing schedule, which achieves a schedule score of 88, with is 71% of the maximum potential schedule value. For each of these 5 variations, Asset 1 and

Asset 2 are active for all but one time step, but Asset 3 is active for only 2 of the 4 time steps. Thus, only 8 of the 12 servicing activities are completed. The scheduler succeeds in finding the "best possible" schedule, although only 8 out of 12 of the tasks were accomplished, the best schedule score was >70% of the possible score because the scheduler selected schedules that completed the highest-scoring tasks. The assets are unable to be active during all time steps because of long-lasting transfers that are required to transit around the KOZ without crossing through it. In several cases, the solver is unable to find any valid solution for the proposed transfer within the single solution bracket explored. As expected, this 4-step simulation again generates 622 schedules on the first time step. There are 2915 schedules generated on the second step, 881 on the third step, and 900 on the fourth and final step, though only the top 100 schedules are kept by the HSF scheduling algorithm after evaluation of each time step.

By increasing the time step duration to 13,000 seconds and the schedule pruning to accept the best 200 schedules after each time step, the value of the best schedule increases. However, this configuration still fails to find a schedule in which all targets are serviced. This updated configuration finds 4 variations of a schedule with a score of 106 that services 10 of the 12 targets. This expansion of the schedule pruning value increases the runtime of the tool dramatically; it now takes about 36 minutes to run the simulation, as compared to only 12 minutes when the pruning value was set to 100 schedules. As expected, with this expanded pruning value there are more schedules generated: 622 schedules generated on the first step, 7556 on the second step, 1138 on the 3rd step, and 1192 on the fourth and final step.

Adding a fifth time step, decreasing the time step duration to 12,000 seconds, and expanding the schedule pruning to accept 250 schedules at each time step still does not achieve the full 12-task schedule, but does achieve 6 variations of a schedule that

services 11 of the 12 targets and has a schedule value of 116, just short of the 124 maximum. This simulation takes nearly 1 hour of runtime. The last servicing activity ends within 61,800 seconds (17 hours and 10 minutes) of the simulation start epoch.

By substantially expanding the time step duration up to 24,000 seconds, a schedule can be obtained that successfully completes all 12 servicing activities. This is achieved with 4 time steps and a schedule pruning value of 80. The pruning value can be kept low because the time step duration is so large. With this small pruning value, the simulation only requires 16 minutes of runtime. The tool discovered only 1 schedule with all tasks completed, but also discovered 52 distinct sub-optimal schedules in which 11 of the 12 servicing activities were completed. For the optimal schedule in which all tasks are completed, Asset 1 expends 12.5g of fuel, Asset 2 expends 179g of fuel, and Asset 3 expends 329g of fuel. All of these are well within the 20kg limit of available fuel. The last servicing activity ends within 112,913 seconds (31 hours and 22 minutes) of the simulation start epoch. The resulting trajectories for this successful mission schedule are shown below in Figure 4.8. Studying the figure and the full 3-D rendering reveals that the trajectories traverse between the target locations just outside of the KOZ volume. This is the expected finding; with the time of flight weighting set to a large value it is expected that the trajectories are all at the edge of the KOZ volume and performing the shortest possible transfer.

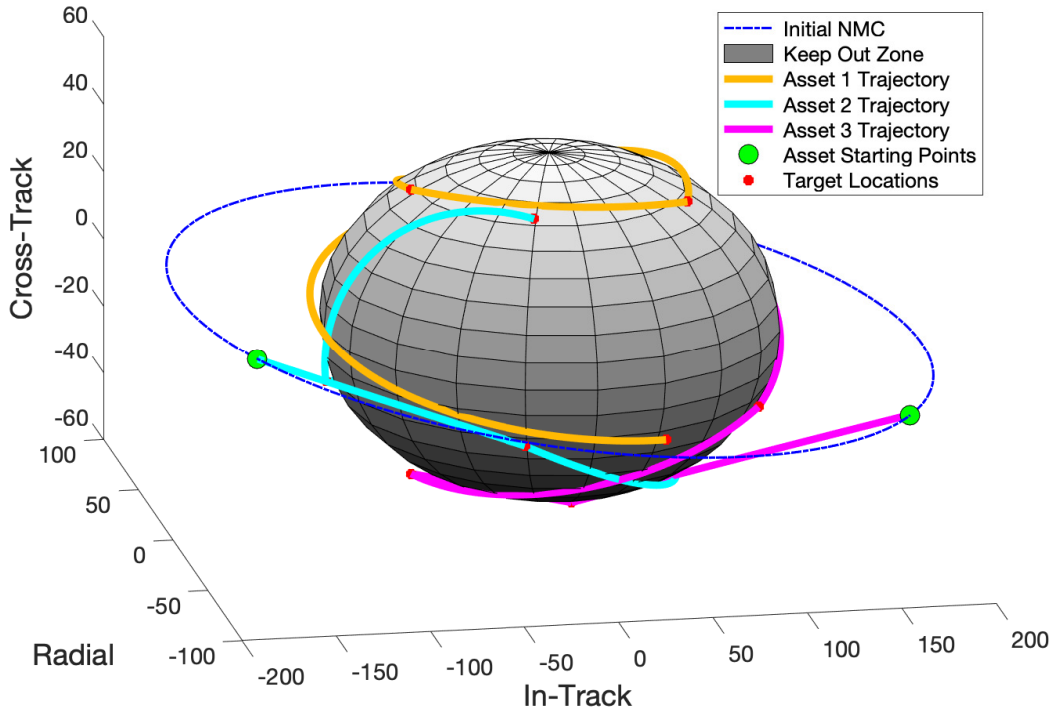

Figure 4.8: DRM Results

A summary of the results from the various simulations with varied time scale and pruning parameters is presented below in Table 4.1. As can be seen in the table, simulating with longer time scales and allowing more schedules to be accepted after each time step results in improved schedules and increased runtime. This result is expected, allowing for longer transfers and analysis of more task combinations/orders should result in improved schedules.

Table 4.1: Summary of DRM Analysis Results

Time Step	No. Time Steps	Pruning Value	No. Assets Serviced	Runtime
10,000 sec	4	100	8/12	12 min
13,000 sec	4	200	10/12	36 min
13,000 sec	5	250	11/12	1 hr
24,000 sec	4	80	12/12	16 min

A more optimal solution that requires less time likely exists, but due to the rapidly growing runtime, it was not discovered in this work. A robust approach for finding the minimum total mission time requires simulating with small time steps (e.g., 5,000 seconds or less) with a large schedule pruning value and iteratively increasing the pruning value and number of time steps until all tasks are complete. However, this approach requires a large amount of runtime as the number of evaluated schedules grows exponentially.

Larger-scale analysis of a DRM with additional servicers and targets was desired and intended, but memory usage issues with HSF were uncovered and resolving these issues is out of the scope of the present research. A scenario with 4 assets and 20 targets was initially studied, but the heavy memory usage caused HSF to crash. Furthermore, increasing the scale of the problem leads to rapid growth in runtime, unless the schedule pruning is kept extremely tight, which limits the ability to determine collision-free schedules that require numerous time delays.

4.5 Results Summary

In summary, this chapter presented the setup and results for application of the integrated servicing fleet mission modeling and scheduling tool to 3 verification scenarios and a full DRM scenario. Each of the 3 verification scenarios produced the expected results, verifying the correctness and optimality of the changes to the main HSF codebase and the implemented *tool*, *guidance*, and *collisionAvoidance* subsystem models. Investigation of a sample DRM with 3 assets and 12 targets provides "proof of concept" support that this servicing satellite fleet mission concept is feasible and can be successfully used to rapidly perform numerous servicing activities. Utilizing 3 assets,

it is possible to complete all 12 servicing activities in about 31.3 hours, or 11 of the 12 servicing activities in about 17.2 hours.

Chapter 5

CONCLUSIONS

The previous chapters detailed the relevant context, implemented algorithms and approaches, and pertinent results of the integrated modeling and scheduling tool for analysis of distributed control of a servicing satellite fleet using the Horizon Simulation Framework. Chapter 1 introduced the notion of a servicing satellite fleet and outlined the present research. Chapter 2 provided further background regarding robotic satellite servicing, the Horizon Simulation Framework, relative orbital motion, optimal transfers, and cooperative control to establish the context of the present research. Chapter 3 detailed the technical work performed for the present research to implement the system model and all necessary modifications to the Horizon Simulation Framework. Chapter 4 discussed pertinent results from examination of 3 validation scenarios and a full-scale DRM. The current chapter provides a summary of the work performed and concluding remarks regarding the resulting tool, the journey to achieve this tool, and the possible future work enabled by the modifications implemented to the Horizon Simulation Framework for this unique application.

5.1 Summary of Mission Concept

The mission concept explored is a fleet of small servicing satellites attempting to rapidly perform several operations for a large central space structure in a circular Geostationary Orbit with target servicing locations dispersed around the surface of an elliptical KOZ. A cartoon of this concept was shown in Chapter 3 with Figure 3.1.

Each servicing satellite has similar mass properties and agile propulsion capabilities. The individual servicing operations and tooling capability of each servicing asset is abstracted to high-level requirements such as "Servicer 1 shall be capable of performing servicing task type A within 3 hours". The mission scenario begins with all servicing satellites in a common NMC about the central space structure. Upon simulation start, the servicing satellites maneuver from this initial NMC to the various target locations and continue to rapidly perform subsequent transfers and servicing operations until all servicing operations are completed, or the simulation time span ends.

Each servicing asset is implemented in HSF with a *tool* and a *guidance* subsystem model. A full mission scenario includes a fictitious "watchdog" asset with a *collisionAvoidance* subsystem model. Each mission scenario is defined by the number of targets and assets, the starting relative states of these targets and assets, the target task types, and parameters to fully define the behavior of the subsystem models. These parameters are the asset tooling dictionaries, parameterization of the KOZ ellipsoid volume, minimum safe collision-avoidance distance, transfer time weighting for the multi-objective solver, vehicle mass and thruster properties, and various algorithm tuning values for the *guidance* and *collisionAvoidance* subsystem models. This work uses identical parameters for defining each of the *guidance* subsystems in a given mission scenario, though this is not required. For simulation of a mission scenario using HSF, the time step duration, total number of time steps, and schedule pruning value must be specified as well.

5.2 Summary of Methodology

Control of the system is distributed across the individual assets to uniquely determine their own optimal transfer trajectories, the fictitous "watchdog" asset to evaluate for collision risks, and the HSF scheduler to assign tasks and pauses for collision avoidance.

The *tool* subsystem dictates what task types can be performed and how long each task takes for a given asset. The *guidance* subsystem computes the optimal two-impulse transfer trajectory to the proposed target and the fuel consumption necessary for these impulses and constant holds for servicing and idling. Optimal trajectories are determined using the CW equations, a multi-objective cost function, an ellipsoid KOZ path constraint, and a modified bisection algorithm for parameter optimization of the optimal transfer time.

The *collisionAvoidance* subsystem evaluates all assets' trajectories to determine if any objects collide or come within the minimum safety distance. The HSF scheduler is utilized with small fundamental time steps to introduce time delays in colliding trajectories to achieve collision avoidance.

Critical alterations, extensions, and "tricks" were implemented in the main HSF codebase and subsystem models. An empty task with a fictitious "EmptyTarget" was introduced so that the HSF scheduler can schedule time delays. Updates to the Asset.cs class were made to enable the "watchdog" asset to participate in the simulation without receiving any tasks. HSF was extended to allow the dynamicState for an asset to be null so that the *guidance* subsystem model can control the state definitions without relying on the rigid ODE infrastructure that is currently in place. Some "tricks" were implemented to side-step the dependency feature of HSF and di-

rectly access state data using the "bulletin board", an approach that was necessary due to current limitations of the dependency feature.

5.3 Summary of Results

Chapter 4 presented the setup and results for application of the integrated servicing fleet mission modeling and scheduling tool to 3 verification scenarios and a full DRM scenario. Each of the 3 verification scenarios produced the expected results, verifying the correctness and optimality of the changes to the main HSF codebase and the subsystem models. Investigation of a sample DRM with 3 assets and 12 targets provides "proof of concept" support that this servicing satellite fleet mission concept is feasible. Utilizing 3 assets, it is possible to complete all 12 servicing activities in about 31.3 hours, or 11 of the 12 activities in about 17.2 hours.

5.4 Further Applications of the Resulting Tool

This work examined proof-of-concept results with a single DRM; the integrated mission modeling and scheduling tool can be used to rapidly assess concept feasibility for a variety of additional servicing satellite architectures. The tool could be applied to:

- Further analysis of time scales and schedule pruning with the reference DRM, as well as investigation into benefits from additional servicers and scalability to servicing of additional targets.
- Trade study between servicer agility/fuel versus number of servicers.

- Investigation into what targets are well-suited for efficient servicing by varying the KOZ dimensions to examine a variety of target size and shapes.
- Investigation of "chained analysis" in which numerous servicers are deployed to multiple targets distributed across GEO, and each scenario is assessed with individual executions of the tool.

5.5 Future Work for System Model Extensions & Enhancements

As discussed in Chapter 3, a variety of assumptions and simplifications were made in the definition and implementation of the system model. Future work could extend and enhance the existing system model to improve model fidelity, solution optimality, runtime, and breadth of applicable mission concepts. Future work could seek to:

- Improve the fidelity of the conjunction analysis performed by the *collision-Avoidance* model by directly considering state uncertainties through covariance propagation, uncertainty reduction through observations, and a probabilistic assessment of these uncertainties to determine the likelihood of a collision given a set of trajectories, uncertainties, and observations.
- Improve the fidelity of the KOZ model with a more generalized polyhedron that better fits the true KOZ for a complex space structure, though this can drastically increase the difficulty of determining if a trajectory crosses into this KOZ.
- Include more trajectory constraints to better represent realistic vehicle limitations, such as a constraint on the solar phase angle during terminal approach to ensure the docking sensors on the servicing asset can function properly, or

a limit on the magnitude of the second impulse to avoid plume impingement issues.

- Investigate the usage of a different approach to the single-variable constrained parameter optimization performed by the *guidance* model to improve runtime, robustness, and/or optimality.

- Enhance the trajectory optimization solver in the *guidance* model by considering additional impulses to improve optimality and enable live re-routing for collision avoidance.

- Utilize a higher-fidelity dynamics model than the CW equations to consider finite burns rather than impulses and/or better model non-circular targets (LERM applies) or distant targets such as in a cislunar orbit (full-force modeling required), though this would require a full re-work of the trajectory optimization approach. Utilizing a closed-loop controller with high-fidelity dynamics modeling would be a major improvement to the model fidelity and enhance the value of the obtained results for practical usage.

- Extend the tool to consider alternative mission approaches, such as alternatives to idling when un-tasked, or dedicated consideration of exit trajectories, for example requiring the assets to return to the initial NMC when all servicing activities are complete.

- Extend the tool to assess further mission concepts, such as including the ability for the servicing assets to periodically refuel when attached to the central space structure, ability to rendezvous with and service moving targets that are not fixed in the RIC frame (i.e., multiple satellites in close vicinity), or ability to move the targets for large-scale assembly operations.

- Consider a dramatically different approach to this cooperative control problem, such as the use of digital pheromones or rule-based controllers, as briefly discussed in Chapter 2.

5.6 Assessment of HSF & Future Work

This work extended the capability of HSF to model and assess new mission concepts for aerospace systems and other applications, and uncovered some underlying issues with the HSF architecture and functionality that motivate on-going and future work. Future work could investigate new mission concepts enabled by the extensions implemented to HSF, such as:

- Utilizing the distributed control approach presented here for other multi-agent systems, such as satellite constellations, drone swarms, or ground-based robotics.
- Utilizing the new time delay feature (i.e., asset passive on a time step) for aerospace missions in which collision avoidance between multiple maneuverable assets is of importance, such as with a swarm of delivery drones maneuvering throughout a dense urban area.
- Utilizing subsystem-defined dynamics to model control and tasking of ground-based robotics systems to complete complex objectives, such as autonomous warehouse operations, automated indoor vertical farming, or search-and-rescue operations in a hazardous environment.
- Utilizing subsystem-defined dynamics to model multi-domain systems, such as tasking a combination of satellite and aircraft assets to perform a search-and-rescue activity for a lost boat.

- Utilizing null dynamics with the modeling and scheduling architecture of HSF for a completely non-physical or static system, such as the nurse scheduling problem or other operations research problems.

Future work is needed to investigate solutions to the architectural and functional deficiencies and issues encountered in this research effort. This includes:

- Revisiting the definition and utilization of the asset's dynamicState. This work side-stepped the dynamicState and utilized the *guidance* model to define, manage, propagate, and manipulate the state vectors for the assets. Future work could revisit the rigid ODE structure of the current dynamicState feature and consider more flexible alternatives for best user value.

- Revisiting the dependency feature of HSF, which was side-stepped in this work with a "hack" to enable cross-asset data access directly from the system state "bulletin board". This "hack" was only effective due to the relatively simple flow of the dependencies. This approach contradicts best-practice modular software development principles and lead to a tangled, monolithic codebase that is not easy to re-use.

- Revisiting the target MaxTimes functionality, which has some issues and required work-around in this work to check in the *collisionAvoidance* model that assets were not assigned the same target.

- Extending the Python subsystem scripting functionality to enable utilization of third-party Python modules, which is currently not supported. This inability slowed the development of the *guidance* and *collisionAvoidance* models, as all algorithms and matrix operations had to be hand-coded or reliant on built-in HSF utilities, often requiring messy conversions between zero-indexed Python

lists and one-indexed HSF matrices. Ability to use the powerful numpy and scipy modules is desirable for developers. Integrating with third-party Python modules would enable rapid development of complex Python subsystem models and broader use-case applications of HSF.

- Refactoring the memory usage and for-loop architecture for schedule and system state creation, evaluation, and maintenance. As discussed in Chapter 4, while the integrated modeling and scheduling tool is effective for relatively-small problems, it does not scale well due to memory and runtime issues. There is on-going work to revisit the class definitions and memory usage as part of a dedicated parallelization effort that should result in substantial runtime improvements.

- Updating the version of Python that is compatible with HSF to enable utilization of modern Python runtime enhancements and capabilities.

- Developing a debugging and test environment, or "sandbox", to enable debugging, testing, and model verification of developers' in-work Python subsystem models. This thesis work required arduous use of print-statements for debugging, testing, and verification during the development and testing of the *tool*, *guidance*, and *collisionAvoidance* Python subsystem models. An environment in which modelers can carefully control the inputs to the system model (e.g., system state & proposed event) and obtain access to changes to the system state and canPerform would greatly improve the development speed and quality of subsystem models and associated unit tests.

5.7 Final Remarks

This thesis work sought to develop a tool using HSF for analysis of servicing satellite fleets. This was implemented with a distributed control architecture to spread pro-

cessing and decision making across subsystem models and the HSF scheduler itself. The resulting integrated modeling and scheduling tool was applied to a sample DRM and found to be successful at determining optimal mission schedules. However, it was found that this tool did not scale well, with larger simulations becoming too memory intensive and slow for practical use with large-scale fleets or vast numbers of servicing locations.

While this work provides stand-alone value, it also enables broader application of HSF with the changes implemented to enable passive assets and subsystem-defined dynamics. Additionally, some of the struggles encountered in this thesis effort motivate future work to improve the interfaces, architectures, and functionality of the Horizon Simulation Framework.

BIBLIOGRAPHY

[1] J. Balfour. Testing and verification for the open source release of the horizon simulation framework. Master's thesis, California Polytechnic State University, San Luis Obispo, 11 2022.

[2] J. T. Betts. Survey of numerical methods for trajectory optimization. *AIAA Journal of Guidance, Control, and Dynamics*, 21(2), 1998.

[3] O. Brown and P. Eremenko. Fractionated space architectures: A vision for responsive space. *4th Responsive Space Conference*, 2006.

[4] G. S. F. Center. On-orbit satellite servicing survey. Technical report, National Aeronautics and Space Administration, 10 2010.

[5] E. Chong and S. Zak. *An Introduction to Optimization, Fourth Edition*. John Wiley & Sons, Ltd., 2013.

[6] N. S. R. Consortium. Model based systems engineering (MBSE). https://www.nasa.gov/consortium/ModelBasedSystems, 02 2020.

[7] B. Conway. *Spacecraft Trajectory Optimization*. Cambridge University Press, 2010.

[8] H. Curtis. *Orbital Mechanics for Engineering Students, 4th Edition*. Butterworth-Heinemann, 2019.

[9] A. Ellery, J. Kreisel, and B. Sommer. The case for robotic on-orbit servicing of spacecraft: Spacecraft reliability is a myth. *Acta Astronautica*, 63:632–648, 2008.

[10] J. Estefan. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Initiative*, 2008.

[11] A. Frye. Modeling and simulation of vehicle performance in a UAV swarm using horizon simulation framework. Master's thesis, California Polytechnic State University, San Luis Obispo, 10 2018.

[12] M. Garcia. Mobile base system. https://www.nasa.gov/mission_pages/station/structure/elements/mobile-base-system/, 2018.

[13] M. Garcia. Remote manipulator system (canadarm2). https://www.nasa.gov/mission_pages/station/structure/elements/remote-manipulator-system-canadarm2/, 2018.

[14] M. Garcia. Special purpose dextrous manipulator. https://www.nasa.gov/mission_pages/station/structure/elements/special-purpose-dextrous-manipulator/, 2018.

[15] R. Garner. About - hubble servicing missions. https://www.nasa.gov/mission_pages/hubble/servicing/index.html, 2021.

[16] D. Ge and X. Chu. Robust learning for collision-free trajectory in space environment with limited a priori information. *Acta Astronautica*, 187, 2021.

[17] C. Gebhardt. Mission extension vehicles succeed as northrop grumman works on future servicing/debris clean-up craft. https://www.nasaspaceflight.com/2021/05/mev-success-ng-future-servicing/, 05 2021.

[18] M. Hause. The SysML modelling language. In *Fifteenth European Systems Engineering Conference*, 2006.

[19] Y. Ichimura and A. Ichikawa. Optimal impulsive relative orbit transfer along a circular orbit. *Journal of Guidance, Control, and Dynamics*, 31, 2008.

[20] D. J. Jezewski and J. D. Donaldson. An analytic approach to optimal rendezvous using Clohessy-Wiltshire equations. *Journal of the Astronautical Sciences*, 27, 1979.

[21] A. Johnson. Cubesat astronomy mission modeling using the horizon simulation framework. Master's thesis, California Polytechnic State University, San Luis Obispo, 09 2019.

[22] A. Jones. China's tiangong space station. https://www.space.com/tiangong-space-station, 2021.

[23] A. Jones. China moves tiangong space station module to side docking port. https://www.space.com/china-moves-module-tiangong-space-station-video, 2022.

[24] N. G. JWST. James webb space telescope faq for scientists. https://jwst.nasa.gov/content/forScientists/faqScientists.html, 2021.

[25] J. A. Kéchichian. *Orbital Relative Motion and Terminal Rendezvous, Analytical and Numerical Methods for Spaceflight Guidance Applications*. Springer, 2021.

[26] R. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM, 2007.

[27] J. M. Longuski, J. M. Guzmán, and J. E. Prussing. *Optimal Control With Aerospace Applications*. SPRINGER-VERLAG NEW YORK, 2014.

[28] M. Lovera. Control-oriented modelling and simulation of spacecraft attitude and orbit dynamics. *Mathematical and Computer Modelling of Dynamical Systems*, 12, 2006.

[29] S. Maclean. Modeling and simulation of a sounding rocket active stabilization system. Master's thesis, California Polytechnic State University, San Luis Obispo, 06 2017.

[30] M. Martin, P. Klupar, S. Kilberg, and J. Winter. Techsat 21 and revolutionizing space missions using microsatellites. In *National Needs and Objectives*, Small Satellite Conference, 2001.

[31] C. Mathieu and A. L. Weigel. Assessing the flexibility provided by fractionated spacecraft. *AIAA Space 2005 Conference*, 2005.

[32] I. MathWorks. fmincon. https://www.mathworks.com/help/optim/ug/fmincon.html.

[33] I. MathWorks. fminunc. https://www.mathworks.com/help/optim/ug/fminunc.html.

[34] I. MathWorks. fzero. https://www.mathworks.com/help/optim/ug/fmincon.html.

[35] E. Mehiel. Horizon. https://github.com/emehiel/Horizon.

[36] E. Mehiel and M. Balas. A rule based algorithm that produces exponentially stable formations of autonomous agents. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2002.

[37] N. G. NExIS. Osam-1: Robotic servicing mission. https://nexis.gsfc.nasa.gov/OSAM-1.html.

[38] J. Nocedal and S. Wright. *Numerical Optimization, Second Edition*. Springer, 2006.

[39] C. O'Connor, E. Mehiel, and B. Butler. Horizon 2.1: A space system simulation framework. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, 2008.

[40] O. of Audits. Nasa's management of the gateway program for artemis missions. Technical report, NASA Office of Inspector General, 2020.

[41] M. Okasha and B. Newman. Modeling, dynamics and control of spacecraft relative motion in a perturbed keplerian orbit. *International Journal of Aeronautical and Space Sciences*, 16, 2015.

[42] C. Peng. *Optimization Based Control for Multi-agent System with Interaction*. PhD thesis, University of California, Berkeley, 2019.

[43] J. Prussing. Optimal four-impulse fixed-time rendezvous in the vicinity of a circular orbit. *AIAA Journal*, 7, 1969.

[44] J. Prussing. Optimal two- and three-impulse fixed-time rendezvous in the vicinity of a circular orbit. *AIAA Journal*, 8, 1970.

[45] J. Prussing. Optimal impulsive linear systems: Sufficient conditions and maximum number of impulses. *The Journal of Astronautical Sciences*, 43, 1995.

[46] L. Riggi and S. D'Amico. Optimal impuslive closed-form control for spacecraft formation flying and rendezvous. In *American Control Conference*, 2016.

[47] Contracts for july 8, 2022. https://www.defense.gov/News/Contracts/Contract/Article/3087981/.

[48] M. A. Saplan. Robotic servicing of geosynchronous satellites (rsgs). https://www.darpa.mil/program/robotic-servicing-of-geosynchronous-satellites.

[49] T. Sauer. *Numerical Analysis, Second Edition*. Pearson, 2012.

[50] A. Shakouri. On the impulsive formation control of spacecraft under path constraints. *IEEE Transactions on Aerospace and Electronic Systems*, 55, 2019.

[51] R. E. Sherrill. *Dynamics and Control of Satellite Relative Motion in Elliptical Orbits using Lyapunov-Floquet Theory*. PhD thesis, Auburn University, 05 2013.

[52] N. Shevchenko. An introduction to model-based systems engineering (MBSE). https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/, 12 2020.

[53] M. T. Suffredini. Reference guide to the international space station, utilization edition. Technical report, National Aeronautics and Space Administration, 09 2015.

[54] System f6 (archived). https://www.darpa.mil/program/system-f6.

[55] D. Vallado. *Fundamental of Astrodynamics and Applications, 4th Edition*. Microcosm Press, 2013.

[56] D. C. Woffinden. *Angles-Only Navigation for Autonomous Orbital Rendezvous*. PhD thesis, Utah State University, 2008.

[57] M. Yost. An iteration on the horizon simulation framework to include .net and python scripting. Master's thesis, California Polytechnic State University, San Luis Obispo, 06 2016.

APPENDICES

Appendix A

COMBINATORICS ANALYSIS OF TASK ASSIGNMENT

This section details the combinatorics analysis performed to confirm the tasks are assigned correctly for the first validation scenario, as described in Chapter 4.

As expected, the status printout from HSF indicates that 14 valid schedules were generated on the first time step. These 14 schedules are attributed as follows:

- 12 schedules arising from the non-repeating permutations of drawing 2 non-repeating elements (1 Target per Asset) from a set of 4 elements (3 Targets plus "EmptyTarget"), as computed by $P_{(n,r)} = \frac{n!}{(n-r)!} = \frac{4!}{(4-2)!} = 12$
- 2 empty schedules: one with both assets assigned the "EmptyTarget" and the default null schedule always included by the HSF scheduler

As expected, the output log file from Horizon indicates 24 variations of the highest scoring schedule that service all 3 targets, 16 variations of each of the 3 sub-optimal schedules that service 2 of the 3 targets, 8 variations of each of the 3 sub-optimal schedules that service 1 of the 3 targets, and 4 schedule variations with zero targets serviced. This sums to a total of 100 generated schedules, which is confirmed in the status printout.

The 24 variations of the highest scoring schedule can be attributed by considering the schedules as a sequence of digits corresponding to the targets. Considering the "EmptyTarget" as target number zero, there are 4 unique targets (digits) to draw

from. A schedule can be represented by 4 digits if you consider the first digit to be the target assigned to asset A on the first time step, the second digit to be the target assigned to asset B on the first time step, the third digit to be the target assigned to asset A on the second time step, and the fourth digit to be the target assigned to asset B on the second time step. In order to achieve the highest scoring schedule, 3 of the 4 asset-target pairings must be with the 3 real targets, so that there is only 1 "EmptyTarget" assigned, and thus these 4 digits are non-repeating for this permutation. Thus, the 24 variations are attributed to the permutations of 4 digits without repetition, $4! = 24$.

The 16 variations of each sub-optimal schedule with servicing for 2 of the 3 targets can be attributed as follows:

- 8 schedules in which both tasks are completed in the same time step: 4 schedules for the first time step and 4 schedules for the second time step. Each time step has 4 schedule variations attributed to the product of the 2 ways the tasks can be assigned (A1 & B2 vs A2 & B1) and the 2 types of empty steps (A0 & B0 vs default HSF null time step).
- 2 schedules in which both tasks are completed by asset A, depending on which time step each task is completed.
- 2 schedules in which both tasks are completed by asset B, depending on which time step each task is completed.
- 4 schedules in which asset A completes a task on one of the time steps and asset B completes the other task on the other time step. These 4 schedule variations are attributed to the product of the 2 orders for the assets completing the task (A first vs B first) and the 2 ways the tasks can be assigned (A1 & B2 vs A2 & B1).

The 8 variations of each single-servicing schedule can be attributed to the product of 2 assets that can service it (A or B), the 2 time steps it can serviced on (first or second), and the 2 types of empty steps (A0 & B0 vs default HSF null time step).

The 4 variations with zero targets serviced can be attributed to the product of the 2 empty step types (A0 & B0 vs default HSF null time step) with 2 time steps.

Appendix B

ACCESS TO SOURCE CODE

Source Python code and XML input files can be obtained from https://github.com/scottplantenga/thesis_source. For this work, the source Python code (guidance.py, collisionAvoidance.py, tool.py) and input XML files with a specific instance of the Horizon Simulation Framework, as defined by the commit SHA 65a1d43. The main HSF repository is at https://github.com/emehiel/Horizon and includes source code, examples, and documentation.

For further information regarding how to use the code or access to the MATLAB code used for validation and analysis, feel free to contact me via email at splantenga@hotmail.com.