Relative-Motion Trajectory Generation and Maintenance for Multi-Spacecraft Swarms

by

Rahul Rughani

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ASTRONAUTICAL ENGINEERING)

May 2021

Acknowledgements

I would like to acknowledge and sincerely thank a number of people for making this work possible. First, I thank my research advisor, David Barnhart, who has shown me that a rigorous approach and a steady supply of snacks can solve any problem. Thanks to Dr. Mike Gruntman for your advice in navigating the winding path of academic journal publications. Thank you to Tyler Presser for your assistance in analyzing the potential applications of swarm trajectories for Geosynchronous orbit slot sharing. Thanks to Kyle Clarke for your assistance in characterizing the state, or rather chaos, or low-earth spacecraft density over the next few decades. Thank you to Dell, Linda, Marlyn, and the rest of the Astronautics department for all their logistical and financial support during my time as a Teaching Assistant. Thank you to the USC Space Engineering Research Center (SERC) and the Information Sciences Institute (ISI) for all the resources and research opportunities offered throughout this dissertation process. Thank you as well to the Center for Advanced Research Computing (CARC) at the University of Southern California for providing computing resources that have contributed to the research results reported within this dissertation (https://carc.usc.edu).

I would like to thank my committee members for their support and advice throughout the course of my research, and Missy Rogers for her help with proofreading and editing. I would like to thank all of my friends for helping me de-stress during these hectic times, especially Matt, Jon, Eric, and Rohan for the weekly Civilization VI games where I could vent my frustrations by way of global domination.

Thank you to my parents, Avnish and Dipti Rughani, and my sister Shalini Rughani for, among many things, teaching me to be inquisitive and to reach for the stars. I would like to thank all of my extended family for their support and encouragement, and to forgive them for all their continual inquisitions of "when are you going to be done with school?" (Sonya, do we really ever stop learning?)

And finally, above all, I would like to thank my partner, Becca Rogers, for her constant encouragement and support that has helped me to finally conclude this adventure.

Table of Contents

List of Tables

List of Figures

Abstract

This work investigates methods for spacecraft Rendezvous and Proximity Operations (RPO) for swarms of spacecraft operating cooperatively. Swarm RPO is the task of operating a group of spacecraft cooperatively to rendezvous with another spacecraft, or with each other. The goal of swarm RPO is to enable cooperative on-orbit construction and assembly projects, allowing for the creation of large structures in space while potentially using in-situ resources. A spacecraft swarm's operational redundancy improves its reliability, making it a useful tool for asteroid and deep-space exploration and enabling success for missions that would typically be seen as high risk. Previous RPO missions have involved two spacecraft engaging in proximity or docking operations (e.g., Shuttle and ISS, Apollo and Soyuz, Probe and Asteroid), and there is also extensive research on and demonstrations of spacecraft formation flying, with multiple spacecraft maintaining a static configuration over time. However, swarm RPO is a relatively new field of research, and up until now was restricted by high barriers to entry such as cost and system complexity. Only recently has the advent of CubeSats and micro-satellites, along with decreases in mission launch costs, enabled the design of small maneuverable spacecraft that operate in close proximity on orbit. Whereas prior work has focused on one-on-one spacecraft RPO and statically configured spacecraft formations, this work considers spacecraft swarms of an arbitrary size that are able to dynamically reorganize their configurations based on the mission requirements and on their individual tasks over the course of a construction project. The primary focus of this dissertation is on efficient trajectory generation and maintenance. These trajectory generation and verification methods are designed to minimize the overall risk of collisions between any spacecraft in the

swarm, or with any object that is in close proximity to the swarm. The methodological solutions presented in this thesis use Genetic Algorithms to evolve a set of initial conditions into a viable and efficient solution, while also applying a Sensor Fusion Unscented Kalman Filter to predict the relative positions of the spacecraft for real-time collision detection and avoidance. These algorithms result in a set of trajectories for each spacecraft that enable it to achieve its mission goals within its ΔV budget, while also leveraging the combined sensor data of the entire swarm to accurately determine the relative positions between each spacecraft to a higher precision than GPS data alone. The scope of this research is limited to the trajectory generation, maintenance, and reconfiguration for an arbitrarily numbered spacecraft swarm, up until the point of final rendezvous and docking within a few meters of a client spacecraft. The exclusion of final docking procedures from the thesis scope is due to the fact that such operations are highly specific to the design of the spacecraft and have been extensively researched and demonstrated. This research, however, strives to solve the as-yet untackled problem of multi-spacecraft coordination for in-space manufacturing. It tests Genetic Algorithm and Filtering approaches in simulated trials for in-space manufacturing of an interplanetary spacecraft, with simulated sensor data and noise inserted into the system. Finally, this work includes an extensive literature survey of spacecraft swarm operations, traditional RPO methods, spacecraft formation flying, and swarm robotic solutions used in terrestrial robotic applications.

Chapter 1

Introduction

1.1 Motivations

This dissertation develops algorithms and techniques to generate and maintain trajectories for a swarm of spacecraft operating in close proximity. The goal of swarm Rendezvous and Proximity Operations (RPO) is to enable cooperative on-orbit construction and assembly projects, allowing for the creation of large structures in space while potentially using in-situ resources. These trajectory generation and verification methods are designed to minimize the overall risk of collisions between any spacecraft in the swarm, or with any object that is in close proximity to the swarm. The methodological solutions presented in this thesis use Genetic Algorithms to evolve a set of initial conditions into a viable and efficient solution, while also applying a Sensor Fusion Unscented Kalman Filter to predict the relative positions of the spacecraft for real-time collision detection and avoidance. These algorithms result in a set of trajectories for each spacecraft that enable it to achieve its mission goals within its ΔV budget, while also leveraging the combined sensor data of the entire swarm to accurately determine the relative positions between each spacecraft to a higher precision than GPS data alone.

With the emergence of the space servicing sector and the return of manned missions beyond low earth orbit (LEO), there is a push to realize the next big step forward in space exploration: in-space manufacturing. This advancement will require large swarms of spacecraft cooperating

in close proximity to each other, all subject to the same laws of orbital mechanics [1]. All these spacecraft operating in close proximity to each other require safe and efficient trajectories, along with methods to maintain these trajectories, accounting for deviations from gravitational perturbations and sensor inaccuracies. Additionally, there is currently an unprecedented surge of new constellations with not just hundreds but thousands of new satellites to be launched over the next few years. This cluttering of Low Earth Orbit (LEO) is yet another driver behind the need for more advanced Space Situational Awareness (SSA) and RPO capabilities. Figure 1.1 shows the current spatial density plot vs. mean orbital altitude for all satellites (and trackable debris) as of March 2020, and Figure 1.2 then shows a projection over time of some of the proposed constellations currently identified [2–4].

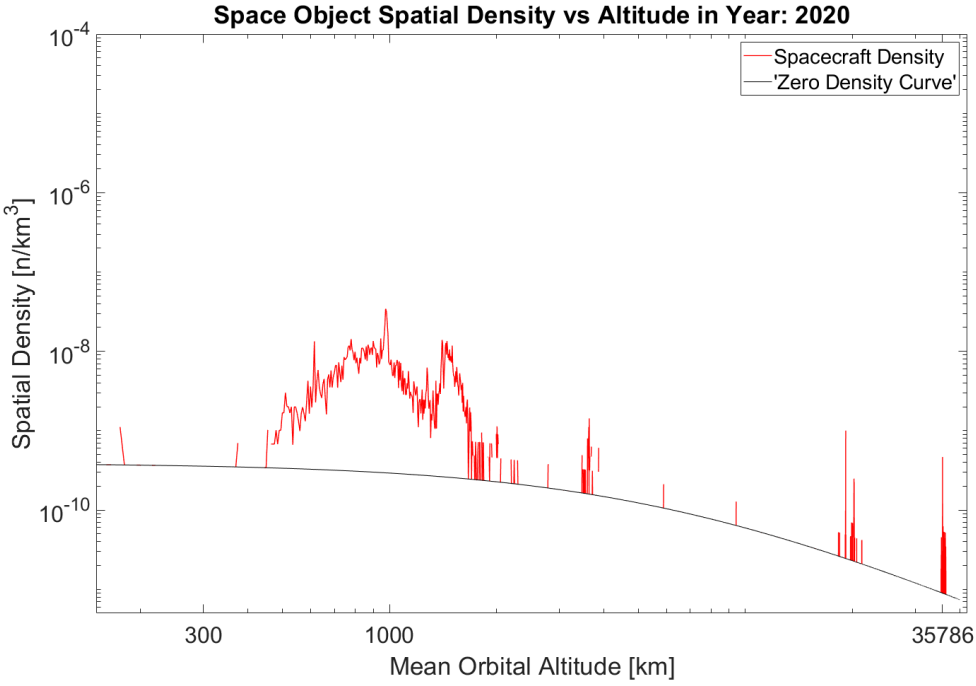


Figure 1.1: Current Spatial Density in Orbit (Figure courtesy of Kyle Clarke)

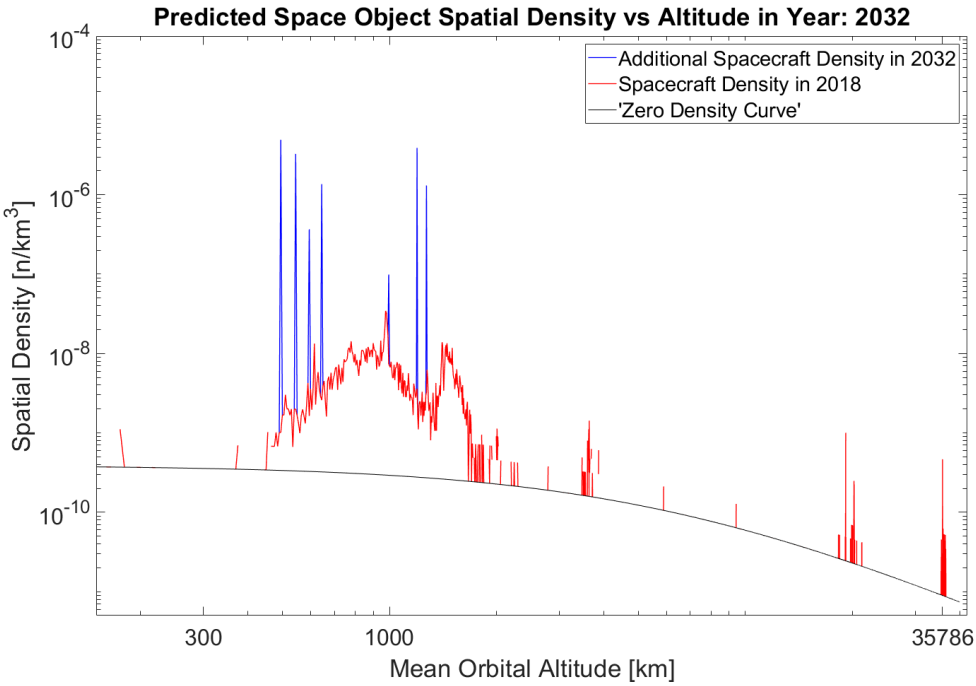


Figure 1.2: Predicted Future Spatial Density in Orbit (Figure courtesy of Kyle Clarke)

The stark contrast between the peaks in figures 1.1 & 1.2, in some cases orders of magnitude higher density in high-trafficked LEO orbits, highlights the increased risk of collision in the coming years, and the need for a methodology to efficiently predict and deconflict trajectories for multiple spacecraft in close proximity.

While techniques for one-on-one spacecraft rendezvous have matured over several decades, along with spacecraft formation flying for multiple spacecraft in a static configuration, the generation and maintenance of trajectories for an arbitrarily large number of spacecraft in a dynamic swarm configuration is a largely untackled problem. Swarm RPO is a relatively new field of research, and up until now was restricted by both cost and system complexity. Only recently has the advent of CubeSats and micro-satellites, along with decreases in mission launch costs, enabled the design of small maneuverable spacecraft that can operate in close proximity on orbit. Such spacecraft system architectures, though currently conceivable purely through the lens of a hardware problem, require significant improvement to the guidance and control framework that

will enable them to operate autonomously and cooperatively without creating dangerous space debris through inadvertent collisions. This dissertation aims to provide such a framework, and proposes a semi-autonomous solution which keeps ground operators in the loop, while allowing some autonomy for each spacecraft in the swarm according to a set of guidelines.

One of the first attempts to formalize the concept of the spacecraft swarm outside of science fiction was performed by U.S. Defense Advanced Projects Agency (DARPA) in the late 2000s, under the name of Future, Fast, Flexible, Fractionated Free-Flying Concept (F6) [5]. F6 sought to replace large monolithic satellites with wirelessly networked clusters of like modules incorporating the various payload and infrastructure functions. This fractioned architecture, a spacecraft swarm, was designed to provide resilience against attacks for critical national security infrastructure. Such a system would require a trajectory determination framework to prevent intra-swarm collisions. Although the program was ultimately cancelled before any flight tests were performed, it identifies a viable use case for spacecraft swarms, as well as showing that a swarm relying on inter-satellite communications can still function if the communications are shut off temporarily.

In the early 2000s, there was also a foray into in-space assembly at the University of Southern California's Information Sciences Institute (USC ISI) [6]. The focus of this research and demonstration was to provide a framework for configurations of like-shaped spacecraft to form a larger object by docking multiple individual components. The experiments demonstrated one of the primary use cases of spacecraft swarm: forming an aggregate structure. Such cellular spacecraft, derived from the DARPA F6 concept, can be mass produced using individual nodes, which together can be reconfigured to serve a variety of functions and missions. Such a spacecraft would be more resilient to damage, and could serve multiple missions throughout its lifetime [7–13].

Through the Phoenix project, DARPA again pursued another technology related to spacecraft swarms: in-space manufacturing [14,15]. Although Phoenix used a monolithic spacecraft, its goal was to use robotics to re-purpose existing hardware in Geosynchronous orbit (GEO) into a new spacecraft, fueling further research into in-space manufacturing and aggregation [16–18]. Phoenix

also incorporated the use of *Satlets*, small free-flying spacecraft that are able to aggregate into larger structures, demonstrating conceptually one of the first examples of in-space manufacturing and aggregation [7, 19–23].

The concept of *Satlets* aboard the Phoenix spacecraft has spawned renewed interest in spacecraft swarms led by NovaWurks and their HISAT platforms [24]. NovaWurks has built and demonstrated the hardware for cellular aggregation of spacecraft in orbit, resulting in a spacecraft that was assembled aboard the ISS from HISAT building blocks and deployed as a free-flyer [25]. The HISAT free-flying spacecraft is the perfect candidate for the framework identified in this dissertation, providing trajectory generation and collision avoidance algorithms to existing cellularized spacecraft.

More recently, the Defense Innovation Unit (DIU) arm of the Department of Defense (DOD) has been pushing for the creation of an *Orbital Outpost* [26], a commercially owned robotic space station capable of manufacturing and deploying spacecraft in orbit. Such an open contract for a commercial outpost shows that the industry is moving closer toward the ability to cheaply manufacture and deploy swarms of spacecraft by the dozens or hundreds. In such a scenario, having the ability to efficiently and rapidly generate and maintain collision-free trajectories for large swarms of spacecraft would potentially make in-space manufacturing more desirable and allow for new and exotic structures to be constructed in orbit.

The scope of this research is limited to the trajectory generation, maintenance, and reconfiguration for an arbitrarily numbered spacecraft swarm, up until the point of final rendezvous and docking within a few meters of a Client. The exclusion of final docking procedures from the thesis scope is due to the fact that such operations are highly specific to the design of the spacecraft and have been extensively researched and demonstrated. This research, however, strives to solve the as-yet untackled problem of multi-spacecraft coordination for in-space manufacturing. It tests Genetic Algorithm and Filtering approaches in simulated trials for in-space manufacturing of an interplanetary spacecraft, with simulated sensor data and noise inserted into the system.

1.2 Current State of the Art

Before diving into detail on new and novel methods of trajectory generation and maintenance for a large swarm of spacecraft operating in close proximity, which will be described throughout this dissertation, it is useful to note the current state of the art for multi-spacecraft rendezvous and proximity operations, and what avenues are currently being pursued by researchers in the field.

There has been some prior research into the use of safety metrics and collision avoidance for RPO in Earth orbit [27]; however, no previous research has sought to combine the use of all of these for swarm operations. For example, Gaylor, Brent, and Barbee [28] outline a framework for safe rendezvous algorithms using safety ellipses and linear optimization techniques, but it is applied only to one-on-one RPO, and is not suited for swarm operations. Izzo and Pettazzi [29] analyze the use of equilibrium potential functions to determine optimal orbits for satellites in the swarm to avoid collisions. Lopez and McInnes [30] employ virtual vector fields to control the final approach trajectory during RPO, but only for one-on-one satellite rendezvous. Slater, Byram, and Williams [31] create probability functions to determine the collision risk of members of satellite formations with foreign objects or drifting members of the formation. However, they do not set up the formation (swarm) in a way that reduces the probability of these collisions in the first place. Ross, King, and Fahroo [32] investigate optimization techniques for formations and swarms, though they concentrate on propellant optimization rather than safety and collision avoidance optimization.

Mauro et al. have developed a control law for low-thrust continuous maneuvers for formation flying reconfiguration, using linearized C-W equations [33]. Though not yet demonstrated on flight experiments, this novel technique forms the basis for more advanced trajectory reconfiguration maneuvers that will be discussed in Section 4.3.

Morgan et al. performed extensive research on spacecraft swarms of hundreds, potentially even thousands, of spacecraft operating in close proximity [34]. This paper forms one of the first

instances of true research into spacecraft swarms, as opposed to extensions of existing formation flying models. The methodology used by Morgan et al. is an energy based strategy to control the allowable distances between the spacecraft. Although this is quite an effective method to generate a set of swarm trajectories, its use case remains limited to large scale semi-static swarms, such as distributed aperture space telescopes or interferometry investigations, where the spacecraft do not need to be reconfigured often. In order to be used for rapidly reconfiguring swarms of spacecraft, such as those required to perform in-space manufacturing and aggregation of spacecraft, an energy based approach is not sufficient. One method of achieving this goal, which will be discussed in future sections, is the use of Genetic Algorithms coupled with decision-based goal oriented programming in order to achieve a prescribed set of goals or targets to achieve a complex mission architecture.

Saaj, Lappas, and Gazi developed a method to use artificial potential fields to design a sliding mode control algorithm for multi-spacecraft swarms [35]. Nallapu and Thangavelautham divided swarms into five distinct classes and developed an attitude control system for multiple spacecraft in orbit around a small and irregular body, enabling collaborative optical measurements of a celestial body [36]. Bandyopadhyay et al. developed a trajectory planning method using convex programming to generate a set of trajectories for a small swarm of spacecraft around an asteroid, with the ability to actively avoid debris [37, 38]. Lippe and D'Amico developed a novel methodology to control spacecraft swarms about a single asteroid with arbitrary gravitational coefficients [39]. D'Amico, in his doctoral dissertation, developed a method for efficient and autonomous formation flying control in LEO using eccentricity/inclination vector alignment and filtering of GPS data [40]. Bezouska and Barnhart created a decentralized system for relative state estimation within a swarm of spacecraft, enabling accurate and reliable pose knowledge and cooperative localization of the entire swarm using distributed processing [41]. Together with the

research by Morgan et al. [34, 42], this literature forms the basis for swarm RPO, and the foundation upon which this dissertation will build to push the envelope and advance the start of the art.

1.3 Problem Statement

A spacecraft swarm is composed of N spacecraft, where N is any positive integer value. These spacecraft are assumed to be maneuverable, either using chemical or electric propulsion methods, equipped also with attitude control systems and relative positioning sensors for relative ranging. Each satellite is able to independently determine its position, velocity, and orientation relative to all other spacecraft in the swarm, using the shared sensor data of the swarm when available. The following research problems are addressed in this dissertation:

1. Given a co-located swarm of N free-flying spacecraft capable of relative position, velocity, and orientation determination, generate a set of trajectories that enable these spacecraft to complete their individual tasks within their ΔV budgets, while mitigating and collision risks over a minimum 24hr period.

2. Given an existing set of co-located swarm trajectories as generated by the solution to the first problem, maintain these trajectories in real-time, accounting for deviations due to injection errors, unaccounted for higher-order or non-gravitational perturbations, sensor errors, or system noise.

3. Given an existing set of co-located swarm trajectories as generated by the solution to the first problem, generate a new set of trajectories for a modified swarm, with some spacecraft either added or removed, while minimizing the ΔV required to re-position the existing swarm spacecraft to accommodate the new spacecraft.

1.4 Solution Approach

The solutions presented in this dissertation are the cumulative results of a series of publications by the author [1, 27, 43–47]. The first and third problems are solved using Genetic Algorithms, which employ an evolutionary solution process to form a family of solutions to the given problem. The lowest ΔV solution from this family is then selected, using the initial state to propagate the solution through time to determine when a non-zero collision risk will arise. It should be noted that although the solution satisfies the given constraints, it is not a globally optimal solution, but instead a locally optimized solution of the obtained solution family subset. As the problem is only loosely bounded, there exists an infinite set of these solution family subsets that can be solved for, each one satisfying the problem's constraints. The second problem is solved using a Sensor Fusion Unscented Kalman Filter, which combines data from sensors aboard each spacecraft to collectively determine the position and velocity of each spacecraft in the swarm, computed independently aboard each spacecraft, to a higher degree of precision than GPS data alone could provide. This filter enables evasive maneuvers to be taken if a spacecraft begins to drift significantly from its assigned trajectory.

1.5 Assumptions and Spacecraft Design

Nallapu and Thangavelautham devised a classification system for different types of spacecraft swarms [36]. This system is as follows:

Class 0 Swarm: A collection of multiple spacecraft that exhibits no coordination in either movement, sensing, or communication.

Class 1 Swarm: Each spacecraft coordinates its movement resulting in formation flying, but there is no explicit communication coordination or sensing coordination.

Class 2 Swarm: Each spacecraft coordinates movement and communication including using Multiple-Input-Multiple-Output (MIMO) or parallel channels. The swarm has collective sensing capabilities but is not optimized with respect to the swarm layout or is post-processed.

Class 3 Swarm: Each spacecraft coordinates sensing/perception with communication and positioning/movement but is still not collectively optimized. Individual losses can have uneven outcomes including total loss of the system.

Class 4 Swarm: Each spacecraft exploits concurrent coordination of positioning/-movement, communication, and sensing to perform system level optimization. The system acts as if it is a single entity. Communication, computation, and sensing are evenly distributed within the swarm. Individual losses result in gradual loss of system performance.

Previous methods of swarm configuration control investigated by Nallapu and Thangavelautham considered swarms of Class 0, 1, and 2 [36,48,49]. The swarm method that will be described in this dissertation uses Genetic Algorithms to generate an overall set of trajectories that avoid collisions and set each spacecraft on trajectories compatible with their mission goals, minimizing the ΔV usage of the swarm. This results in a swarm that falls somewhere between a Class 3 and a Class 4 swarm, which will be dubbed *Class 3.5 Swarm*. This is a swarm in which each spacecraft coordinates position, movement, and communication, while there is still a centralized computation authority (which can be transferred if needed).

Although future research will strive to upgrade this method to a Class 4 Swarm, this will require in-depth investigations into distributed computational schemes that are outside the scope of the current research. Given this, the following assumptions are used to simplify the problem:

1. **All spacecraft in the swarm have a known mass, moment of inertia, and center of gravity. Any changes to these values are tracked by the system as fuel is consumed or replenished, and as spacecraft are aggregated or disaggregated:**

   It is assumed that the swarm is made of similar spacecraft with all properties known to a high-degree of accuracy.

2. **The number of spacecraft, N, in the swarm is known, and finite:**

   This is a reasonable assumption, as the constellation designer would have this information before commencing an operation. The number of satellites in the swarm is not assumed to be constant, as the swarm may change in size as mission parameters change or additional components are aggregated together. This requires that all safety ellipses and swarm orbits be re-computed every time a spacecraft is added or removed from the swarm.

3. **The relative-motion trajectory's reference point is moving in a circular orbit:**

   All rendezvous operations take place in a relative motion coordinate system, which is a non-inertial reference frame, translating and rotating around the Earth. The reference point is the origin of this relative motion coordinate system, which is itself an orbital trajectory in an inertial reference frame. This is in most cases the center of gravity of a target spacecraft, but in some cases, there will not be a target spacecraft, and this could be the location all objects in the swarm are grouping around. This can be modified to allow elliptical reference orbits [50], however typical trajectories that would be useful for a spacecraft swarm are in circular or near-circular orbits, along with the majority of spacecraft in LEO and GEO.

4. **The central body has a gravitational field model that can be expressed using spherical or zonal harmonics:**

   The central body must have a well-known and mapped gravitational field in order to enable the precise trajectory predictions for a swarm of spacecraft to the degree necessary for collisions avoidance. This allows for simulations to account for the effects of the Earth's

oblateness (J2), and perturbations related to longitudinal variations in the Earth's gravitational field. [51]

5. **Communications delay is negligible:**

   All satellites in the swarm are assumed to be in close proximity, as this is the definition of Rendezvous and Proximity Operations. The speed-of-light transit time between the spacecraft is thus small, so any communications delays are negligible. Additionally, it is assumed that any signal processing delays on board the spacecraft are also negligible.

6. **All clocks are perfectly synchronized across the swarm:**

   The on-board clocks on the spacecraft are assumed to always be in perfect synchronization with each other. Although this will never be the case in reality, with continuous low-latency communication between the spacecraft, the clocks will be in sync to a high degree of accuracy, enabling effective synchronization for the timescales of RPO.

7. **Spacecraft that lose communication with the rest of the swarm will enter a passive mode until communications are regained:**

   The swarm is able to function in close-proximity operations primarily due to the communications network between the spacecraft, enabling accurate position knowledge of each spacecraft, and coordinated operations when altering trajectories of the swarm. However, if a spacecraft loses the ability to communicate with the rest of the swarm, it will be designed to fall into a passive *safe mode*, such that it will not perform any maneuvers unless the on-board sensors predict that a collision is imminent. This eases the burden on the remainder of the swarm to avoid such a *zombie spacecraft*.

8. **Each spacecraft will have a radio beacon signal with a unique identifier:**

   Each spacecraft in the swarm will have a beacon transponder, similar to an aircraft IFF system, that will be constantly identifying itself with a unique identification code. This will allow every spacecraft in the swarm to know which other nearby spacecraft exist and are

part of the swarm, as well as to know when a spacecraft enters or leaves the vicinity of the swarm.

9. **The scope of this research ends at the final docking phase:**

   The scope of this research is limited to the trajectory generation, maintenance, and reconfiguration for an arbitrarily numbered spacecraft swarm, up until the point of final rendezvous and docking within a few meters of a client spacecraft (Client). The reason for the exclusion of final docking procedures from the thesis scope is because such operations are highly specific to the design of the spacecraft and has been extensively researched and demonstrated [52–71], while this research strives to solve the as-yet untackled problem of multi-spacecraft coordination for in-space manufacturing.

The methodology and analysis that will be covered in later chapters assumes certain characteristics about the member spacecraft in the swarm. Specifically, the spacecraft must all have the following capabilities to be a viable member of the swarm:

1. On-board propulsion system and fuel source (preferably refuelable) with a minimum 200 m/s Δv capacity.
2. Three-axis attitude control, and attitude knowledge with 0.5° accuracy.
3. Relative motion position sensors to determine range to nearby spacecraft with 5% accuracy
4. Relative motion speed sensors to determine the speed of nearby spacecraft with 1% accuracy
5. Redundant communication systems to transmit and receive data between all spacecraft in the swarm.

A variety of propulsion systems can be used to satisfy the above criteria, either chemical or electric in nature. For electric propulsion scenarios, the simulations were run using an idealized 50 µN thruster with 2000 s of I_{SP}. There exist thrusters currently available [72–77], and in development [78–80], that can satisfy this criteria, including one being developed at USC [81].

An example of an existing spacecraft with these capabilities is the NovaWurks HISAT [24], pictured in Figure 1.3 below. The HISAT is a spacecraft designed as a building block, which can be aggregated into larger structures, and reconfigured when needed. Figure 1.4 shows an example of such a configuration [25].

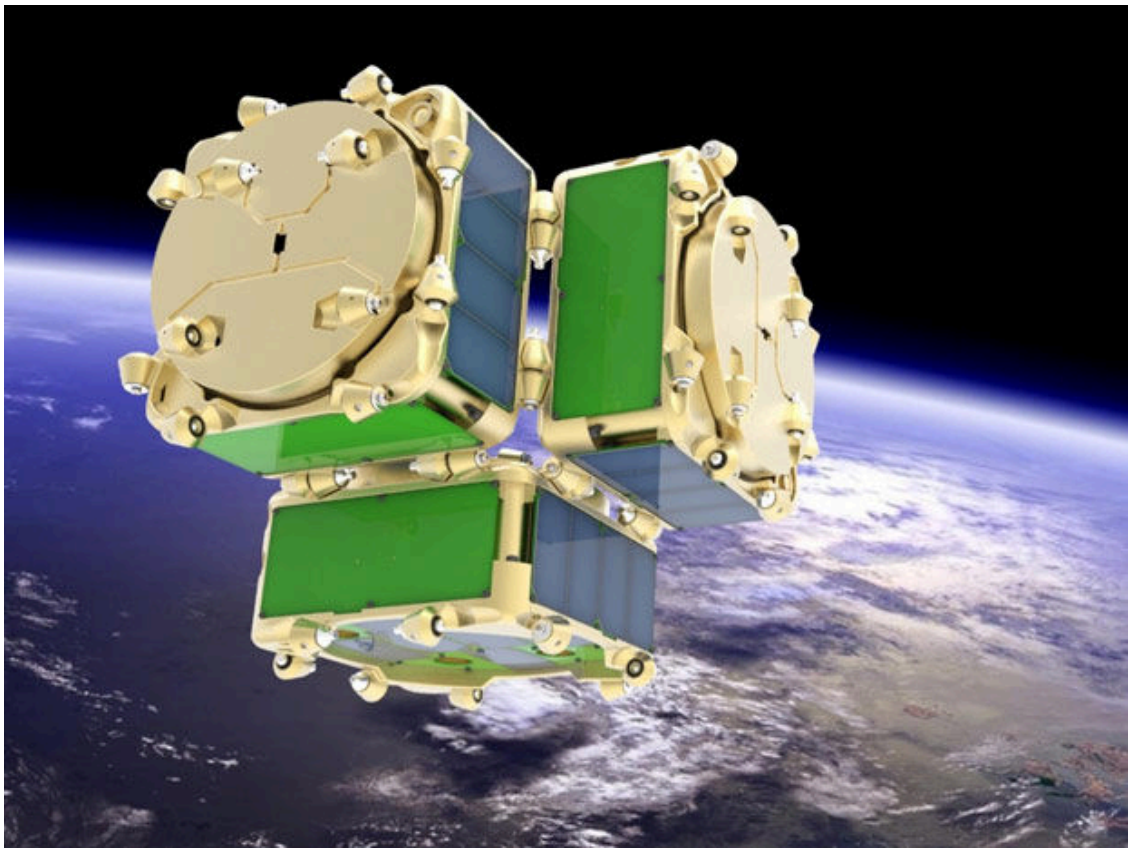


Figure 1.3: NovaWurks HISAT Spacecraft

Figure 1.4: HISAT Platform Configuration

1.6 Challenges

Automating the generation and maintenance of spacecraft swarm trajectories is challenging due to the nonlinear nature of relative motion trajectories in a non-uniform gravitational field, coupled with the large number of spacecraft operating in close proximity. The increased risk of collisions between spacecraft requires more stringent trajectory and navigation constraints on the member spacecraft of the swarm, and all of this will require additional Δv expenditure which will need to be budgeted into the operational costs of the mission.

While trajectory propagation is quite straightforward to perform, if the equations of motion of the system are known, trajectory determination – finding an initial state that will propagate to a desired state after a given time – is not so trivial. Due to the nonlinear nature of the perturbed gravitational field equations, there is no quick analytical solution available to this problem. Instead, nonlinear iterative methods must be used, propagating the equations of motion numerically towards a solution. To improve the solver's efficiency, it can be combined with

estimated solutions using linearized equations of motion for the initial guess to the iterative solver.

Another major challenge of swarm operations to overcome is how to deal with loss of communication between one or more spacecraft in the swarm, and how that will affect the operations of the other spacecraft. One of the edge cases considered in this thesis is that of an unexpected vehicle loss. If a member of the swarm were to go offline mid-mission, either entirely or from a communications standpoint, it would be considered a *zombie satellite*, for all intents and purposes a piece of debris that all spacecraft in the swarm must avoid. This avoidance is handled in the collision avoidance scheme, where a safety corridor of 10 m around this trajectory is marked as a restricted zone, forcing the solver to generate trajectories that do not cross into this zone. However, this is only a short term solution to this problem. Over the long term, this keep-out zone will grow as position errors propagate over time, resulting in the need to either restore communications to the spacecraft by operators on the ground, or to nudge the failed spacecraft out of the swarm so that it no longer poses a risk to the rest of the spacecraft in the swarm.

1.7 Overview of Thesis

1.7.1 Chapter 2 – Background

Chapter 2 describes the necessary background information to build the foundation of swarm rendezvous operations, as well as the progression of prior research in the field, and related research in other fields. The chapter begins by providing a detailed mathematical foundation for Rendezvous and Proximity Operations (RPO), especially in non-uniform gravitational fields, such as the Earth's. Next, an overview of formation flying is given, along with specific research used as a foundation for swarm trajectory generation. This is followed by a list of ground-based analogs to spacecraft swarm operations, including the relatively new field of cooperative drone operations.

Finally, the chapter concludes with an overview of Kalman filtering techniques, and Genetic Algorithm (GA) machine learning techniques.

1.7.2 Chapter 3 – Generating Initial Trajectories for the Spacecraft Swarm

Chapter 3 describes in detail the process used to generate a set of RPO trajectories for a swarm of spacecraft, using Genetic Algorithms (GAs). First, the properties of the swarm are defined, such as the number of spacecraft, the orbital elements of the reference point (either a Client spacecraft, or location of a construction site). Then, the constraints on each spacecraft are defined, such as the allowable approach ranges to other spacecraft, any line-of-sight requirements for mission success, and available Δv budgets. Next, a set of nested GAs are used to solve for a set of trajectories that satisfy both the individual spacecraft constraints and the necessity to prevent inter-spacecraft collisions, while also minimizing ΔV consumption. This yields a set of initial state vectors for each spacecraft such that, when realized, the swarm will be formed without any risk of collision for a prescribed amount of time (user defined – at least 24hrs).

After the trajectories are defined and achieved, a new set of GA solvers can be used to modify the existing set of trajectories to enable new spacecraft to be inserted into the swarm, or for new tasks to be assigned to the various spacecraft in the swarm. This is done while once again minimizing the required Δv for any set of maneuvers. Provisions are defined for how to deal with an in-space construction site, where the size of the Client object is growing in size as the mission proceeds, thus requiring reactive changes in the swarm to increase its size over time. Finally, a comparison of the GA method to arbitrary trajectories is shown, using a Monte Carlo based simulation approach.

1.7.3 Chapter 4 – Trajectory Maintenance for Spacecraft Swarms

Chapter 4 describes the methodology used to maintain existing swarm trajectories, and derives the equations used to compute when and how to perform small trajectory correction burns to prevent any drift due to sensor errors and thruster inaccuracies. Next, the Kalman filtering method is described, using sensor fusion techniques to combine data from multiple sensors distributed across the swarm, in order to more accurately determine the relative positions of all spacecraft in the swarm, and filter out any readings that are not physically possible or probable.

The chapter also describes a method for patched RPO trajectories using Kalman filtering to provide real-time updates during the transfer, enabling accurate transfer operations without requiring extremely precise instrumentation aboard the spacecraft. Finally, it covers the various stationkeeping maneuvers and schemes used to maintain the generated swarm trajectories, showing example simulation results with estimated Δv figures. It also describes the considerations taken into account for continuous thrust engines during these stationkeeping maneuvers.

1.7.4 Chapter 5 – Behavioral Stresses of the System

Chapter 5 describes in detail a select few edge cases of the spacecraft swarm framework that are used to probe the behavioral stresses of the system. This is useful to determine the regime of operations for the swarm framework, to better understand in what scenarios it is practical to apply the framework.

The first of these edge cases probes what happens in the event of the unexpected loss of a vehicle. In this case, the position and velocity of this spacecraft is constantly updated from Kalman filtering of sensor readings from the other spacecraft. The trajectory of this now-defunct spacecraft is labelled as a restricted keep-out zone, and any trajectories that are predicted to intersect with it are immediately modified using methods defined in Chapter 3.

The second edge case looks at how to respond to a dynamic construction environment, such as that described at the end of chapter 3, in which the swarm is centered around a constantly growing in-space construction site. In such an environment, a large structure (e.g., space station) is being aggregated from components in space. As its size grows, its center of mass and moment of inertia change, and with them, its rotation rate. All of these factors must be taken into account for the trajectory generation and maintenance of the swarm. This is much more than an operator on the ground could handle, and thus is handled by an automated system, which the ground controller can review and provide inputs when needed.

The third edge case considers the collision avoidance scheme in use by the swarm, and describes the algorithms that the swarm employs to detect and react to any collision risk, using different avoidance methods depending on the estimated time to the collision. In a worst-case scenario, this may result in a spacecraft being ejected from the swarm, or moved into a position of safety where it can no longer complete its mission, in order to save the rest of the swarm.

1.7.5 Chapter 6 – Swarm Configuration Example Scenario

Chapter 6 details an example swarm configuration from start to finish for an on-orbit construction project. This includes the definition of the swarm, the results from the trajectory generation process, a modification to the swarm to introduce new spacecraft to increase its capabilities, and the loss of a spacecraft to an unsolvable error, resulting in a piece of debris in the vicinity of the remainder of the swarm. It also includes the computation of the stationkeeping maneuvers used to recycle this set of trajectories once the orbital drift becomes too large, and a summary of the Δv usage for each operation. Kalman filtering is used during all propagation simulations between each step to verify safe operation of the swarm, even under random error conditions.

1.7.6 Chapter 7 – Other Considerations

Chapter 7 lists additional aspects of spacecraft swarms that can be considered for future study, and touches briefly on each of them without going into detail. These include orbital reconnaissance, self-aggregating swarms, computational distribution throughout the swarm, light-time delay with mission control for deep space missions, foreign object threats, and irregular keep-out zone restrictions.

1.7.7 Chapter 8 – Application to GEO Spacecraft

Chapter 8 covers a unique application of this swarm methodology to small swarms of spacecraft co-located in a single Geostationary orbit slot. It describes the method of using cooperative satellite swarm trajectory generation and maintenance in order to reduce the propellant utilization of spacecraft in GEO, maintaining a dynamic formation flying configuration which enables each spacecraft to perform their individually required operations, while also choosing trajectories that prevent collision risks under free-flight trajectories for an extended duration. This dynamic trajectory formulation is performed using Genetic Algorithms in order to search the entire solution space for a family of solutions that satisfies all the constraints set to the problem, and is able to find minimal Δv solutions even with the nonlinear equations of motion.

1.7.8 Chapter 9 – Conclusion

Chapter 9 summarizes the results of this thesis and discusses avenues for potential future research paths. The advantages of an automated system to generate and maintain trajectories for an arbitrarily sized swarm of spacecraft are described, along with the advantages of using sensor fusion Kalman filtering to maintain accurate tabs on the position and velocity of all spacecraft in the swarm. Additionally, several drawbacks of the proposed methodology are discussed, along with possible solutions that remain to be investigated in more detail.

Chapter 2

Background

Previously, Chapter 1 provided an overview of the thesis and discussed the motivations, current state of the art, problem statement, and assumptions for this research, along with the various challenges associated with implementing and achieving this goal. This chapter provides the relevant background information on the research topic, and provides an overview of Rendezvous and Proximity Operations (RPO) and the methods used to compute trajectories in relative motion within the confines of a planetary gravitational field. Prior research into spacecraft swarms and formation flying are reviewed, as well as an overview of ground based analogs.

2.1 Rendezvous and Proximity Operations

In terms of orbital mechanics, Rendezvous and Proximity Operations (RPO) is the process of a spacecraft (Servicer) approaching and matching the orbit of another spacecraft (Client) [82]. In-space RPO has been performed successfully since the 1960s, first demonstrated during the Gemini missions [83]. Current RPO methods still focus only on one-to-one operations [84–94]; that is, a single Servicer and a single Client.

When starting this research, the author was sponsored by the Consortium for the Execution of Rendezvous and Servicing Operations (CONFERS) to perform an in-depth study into the

state of the art of Rendezvous and Proximity Operations for satellite servicing, identifying key improvements to safety that would be useful for the future of satellite servicing. Among the results of this study [27, 43, 95], the primary takeaway was that passively safe trajectories should be the primary collision avoidance method, with active methods being used as secondary avoidance. The results from this study are incorporated into this thesis to develop an overall swarm trajectory generation method.

To foster an environment open to advanced multi-platform operations (i.e. in-space manufacturing or assembly), there first needs to be a framework in place to allow multiple Servicers to operate on a single client, or even multiple Servicer's to multiple clients in the same vicinity. This is referred to as Swarm RPO.

Swarm RPO can enable multiple new capabilities on orbit, where two next-generation operations may be adaptive formation flying and satellite aggregation. Adaptive formation flying is the process of multiple spacecraft operating in relative motion orbits, within a few kilometers of each other, working towards a common goal. For example, this could be scanning a Client spacecraft using optical, radar, and LIDAR sensors, or manufacturing large space platforms. Satellite aggregation is the process of constructing platforms or spacecraft in orbit, using smaller spacecraft as the building blocks, both in a structural and a software sense [96]. With swarms of spacecraft operating in close proximity to each other, from a safety perspective it will be necessary to have a method to optimize the trajectories of each spacecraft, minimizing the risk for collisions, while allowing them to fulfill their mission operations.

For the purposes of this analysis, the definition of a swarm is a group of two or more spacecraft cooperating towards a common task or goal, within close proximity to each other (approximately 25 km - 50 km in LEO). The analysis is performed in the relative motion non-inertial coordinate system, which can be defined by the linearized Clohessy-Wiltshire equations [50], or by numerically integrating the relative motion equations of motion to account for perturbations from gravitational and other sources [51].

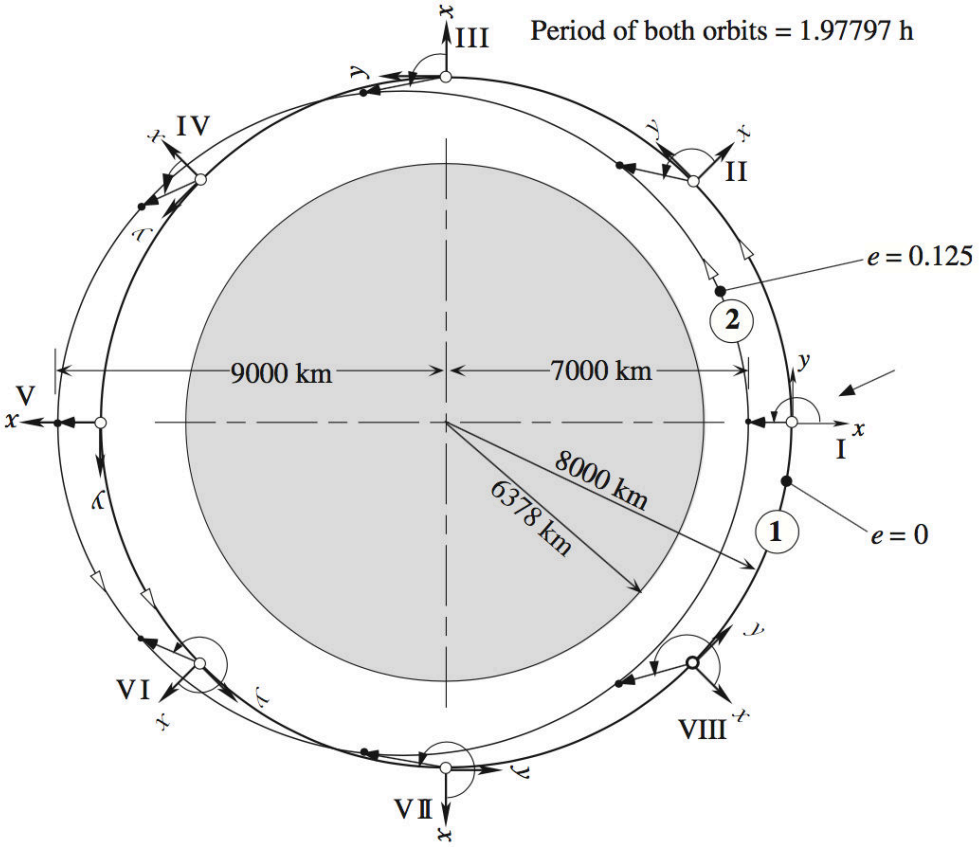
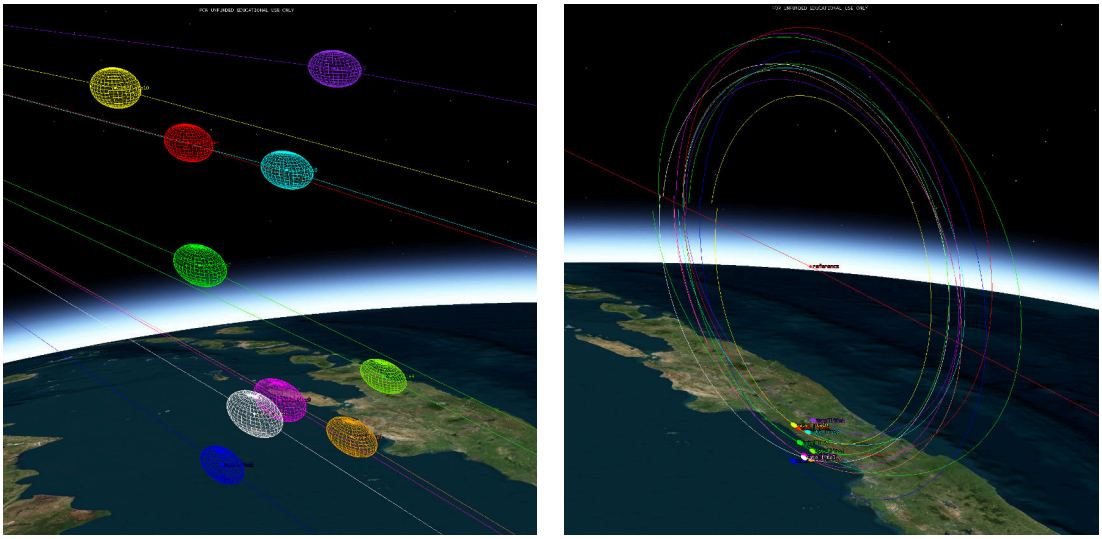


Figure 2.1: Slightly eccentric orbit allows relative motion [50]

As seen in Fig. 2.1, a spacecraft in a relative motion trajectory is in a slightly eccentric orbit from a reference observer on a circular orbit. As viewed from a co-located reference frame, moving and rotating with the observer, the spacecraft is "orbiting" around the observer. The mechanics of the free-trajectory motion following these relative motion orbital tracks are well known and

understood, having been used for more than fifty years, prior to the Apollo missions [83]. However, methods to autonomously maintain and guide such relative motion trajectories are not as well understood, given that robust automated rendezvous techniques have been available for just over a decade [52]. Fig. 2.2 shows a depiction of what a set of swarm orbits may look like in the relative motion frame. Fig. 2.2a shows the swarm as viewed in inertial space, each with slightly different eccentricities and inclinations, such that if viewed from a co-moving reference frame, the trajectories appear as in Fig. 2.2b.



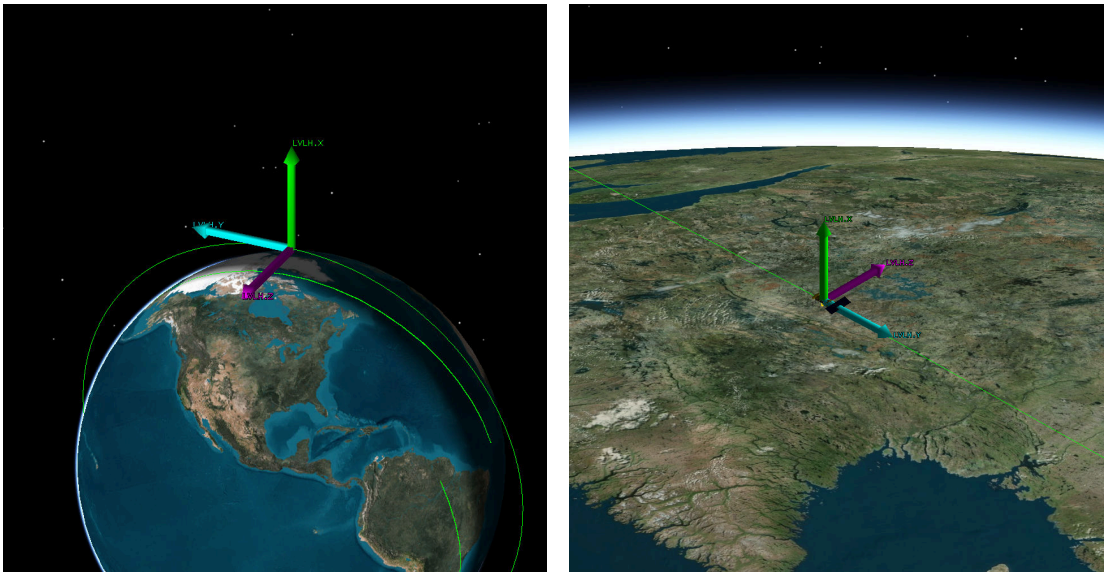
(a) Inertial View (b) Co-located View

Figure 2.2: Swarm of Spacecraft in Relative Motion

2.1.1 Mathematical Formulation

Relative orbital motion takes place in the Local-Vertical Local-Horizontal (LVLH) rotating reference frame. This non-inertial reference frame is centered on a point in space, in orbit around the Earth, which could be a Client spacecraft, a waypoint, or some other point of interest. The x-axis (radial) is directed along the outward radial vector from the center of the Earth to the target, the z-axis (cross-track) is normal to the orbital plane of the target, and the y-axis (in-track) lies within the orbital plane, constrained by the x- and z-axes to form a triad. This is depicted in

Figure 2.3, where the radial vector is displayed in green, the in-track in blue, and the cross-track in purple.



(a) Inertial View (b) Co-located View

Figure 2.3: LVLH Coordinate Frame

This motion can be described by the following equations of motion, where R is the vector from the center of the Earth to the Client, and δr is the vector from the Client to the Servicer vehicle:

$$\delta\ddot{\vec{r}} = -\ddot{\vec{R}} - \mu\frac{\vec{R}+\delta\vec{r}}{\left\|\vec{R}+\delta\vec{r}\right\|^3} \tag{2.1}$$

These equations of motion are a nonlinear system of equations; however, a linearized approach is desired to use in a real-time guidance application. If the target spacecraft is restricted to be in a circular orbit, the system can be defined in a closed-form linearized approximation by the Clohessy-Wiltshire (C-W) equations [50], laid out below in Equations 2.2 - 2.4.

$$\delta\ddot{x} - 3n^2\delta x - 2n\delta\dot{y} = 0 \tag{2.2}$$

$$\delta\ddot{y} + 2n\delta\dot{x} = 0 \tag{2.3}$$

$$\delta\ddot{z} + n^2\delta z = 0 \tag{2.4}$$

These differential equations are valid while the following criterion from the linearization process holds:

$$\frac{\|\delta\vec{r}\|}{\|\vec{R}\|} << 1 \tag{2.5}$$

A closed form solution of these coupled partial differential equations can be obtained, expressed in matrix form in Equations 2.6 - 2.7 below, enabling the computation of position and velocity at any point in time:

$$\delta\vec{r}(t) = [\boldsymbol{\Phi_{rr}}(t)]\delta\vec{r}_0 + [\boldsymbol{\Phi_{rv}}(t)]\delta\vec{v}_0 \tag{2.6}$$

$$\delta\vec{v}(t) = [\boldsymbol{\Phi_{vr}}(t)]\delta\vec{r}_0 + [\boldsymbol{\Phi_{vv}}(t)]\delta\vec{v}_0 \tag{2.7}$$

where the initial position and velocity are

$$\delta\vec{r}_0 = \begin{bmatrix} \delta x_0 \\ \delta y_0 \\ \delta z_0 \end{bmatrix}, \ \delta\vec{v}_0 = \begin{bmatrix} \delta u_0 \\ \delta v_0 \\ \delta w_0 \end{bmatrix}$$

n : angular rotation rate of orbit (rad/s)

t : time since initial conditions

$$\boldsymbol{\Phi_{rr}}(t) = \begin{bmatrix} 4 - 3\cos nt & 0 & 0 \\ 6(\sin nt - nt) & 1 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{2.8}$$

$$\boldsymbol{\Phi_{rv}}(t) = \begin{bmatrix} \frac{1}{n}\sin nt & \frac{2}{n}(1 - \cos nt) & 0 \\ \frac{2}{n}(\cos nt - 1) & \frac{1}{n}(4\sin nt - 3nt) & 0 \\ 0 & 0 & \frac{1}{n}\sin nt \end{bmatrix} \tag{2.9}$$

$$\boldsymbol{\Phi_{vr}}(t) = \begin{bmatrix} 3n\sin nt & 0 & 0 \\ 6n(\cos nt - 1) & 0 & 0 \\ 0 & 0 & -n\sin nt \end{bmatrix} \tag{2.10}$$

$$\boldsymbol{\Phi_{vv}}(t) = \begin{bmatrix} \cos nt & 2\sin nt & 0 \\ -2\sin nt & 4\cos nt - 3 & 0 \\ 0 & 0 & \cos nt \end{bmatrix} \tag{2.11}$$

Although the C-W equations are linearized approximations of a nonlinear system, the approximations are sufficient for the purposes of orbital rendezvous and proximity operations. The solutions diverge when the distance from the target is a significant percentage of the mean orbital radius of the target, as this is when the Earth's curvature will have an effect on the direction of the gravitational perturbations. Thus, for LEO, based on the linearization criterion (Equation 2.5), these solutions can be used with relatively high accuracy within a few dozen kilometers of the target, and in GEO within a few hundred kilometers of the target [50].

2.1.2 Orbit Maintenance

Now that the nature of relative orbits and the trajectory that an object in relative motion will follow has been described, the next step is to define how to maintain a relative motion trajectory. Even if a spacecraft were injected perfectly into its orbit, there are gravitational perturbations to be considered, such as the Earth's oblateness, the Moon, and the Sun, all of which will impart tiny forces to perturb the spacecraft's orbit over time. Additionally, deviations to the planned trajectory are caused by imperfect injections into the desired orbit, leading to a drift in the trajectory compared to the nominal path.

It is possible to compute the exact acceleration deviations caused by the perturbing gravitational bodies and come up with a control system to compensate for this using periodic application of thrust forces. However, for swarm RPO, this is not necessary for the most part. A rigid trajectory is generally not required since, during swarm RP,O much of the focus is on entering a relative motion closed-form trajectory around a target spacecraft or body. If this trajectory deviates by a few meters, it will not affect the mission so long as all the spacecraft in the swarm are sufficiently far enough apart that a deviation of a few meters will not cause a collision (see Figs. 2.4 and 2.5). Rather than use the limited fuel resources to maintain a given trajectory, when significant deviations occur a new trajectory can be computed, which can be transitioned to while conserving propellant.

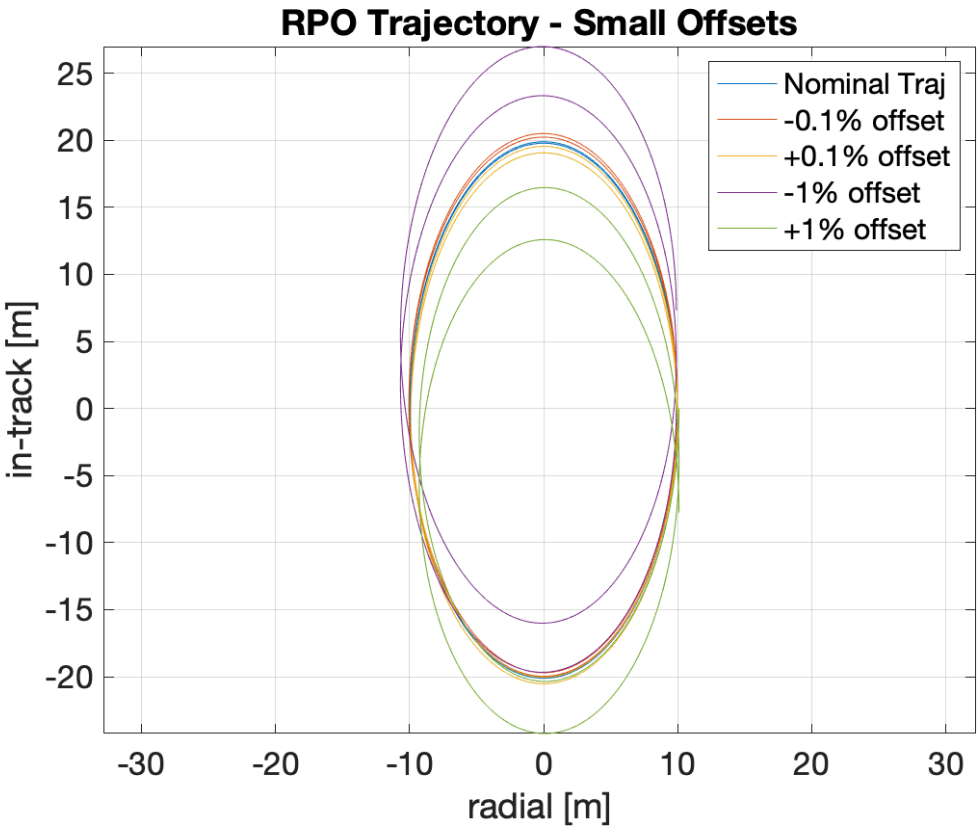


Figure 2.4: Trajectory Offsets for Various Levels of Position Injection Error

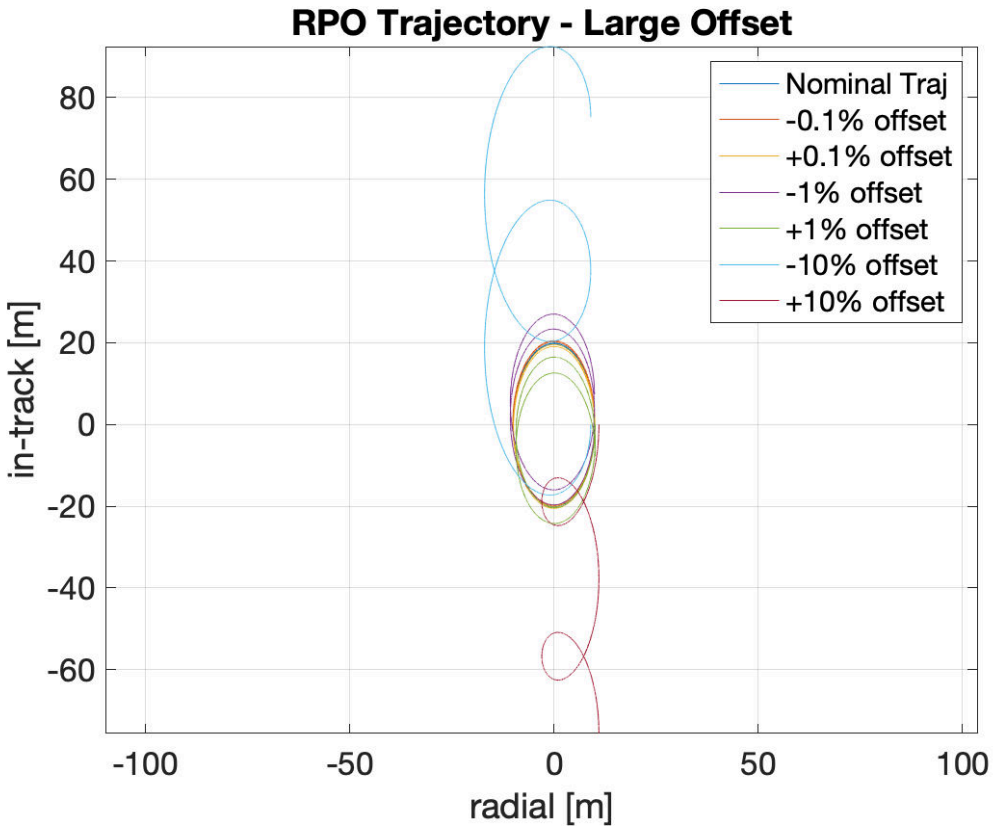


Figure 2.5: Trajectory Offsets for Large Injection Errors

2.1.3 Perturbation Effects

In order to take into account the perturbation of the J2 effect of Earth's oblateness (the primary gravitational perturbation below Geostationary orbit), a modified set of C-W equations must be derived. This mathematical problem has been solved already [97], with the equations of motion as follows:

$$
\begin{aligned}
x(t) = &\left(\frac{5s+3}{s-1}x_0 + \frac{2\sqrt{1+s}}{n(s-1)}\dot{y}_0\right. \\
&\left. + \frac{1}{4}\frac{A_{J2}(3k - 2n\sqrt{1+s})\sin^2 i}{k(-n^2+n^2s+4k^2)}\right)\cos\left(nt\sqrt{1-s}\right) \\
&- \frac{1}{4}\frac{A_{J2}(3k - 2n\sqrt{1+s})\sin^2 i}{k(-n^2+n^2s+4k^2)}\cos\left(2kt\right) \\
&+ \frac{\dot{x}_0}{n\sqrt{1-s}}\sin\left(nt\sqrt{1-s}\right) - \frac{4(1+s)}{s-1}x_0 - \frac{2\sqrt{1+s}}{n(s-1)}\dot{y}_0
\end{aligned}
\tag{2.12}
$$

$$
\begin{aligned}
y(t) = &\left(\frac{2(5s+3)\sqrt{1+s}}{(1-s)^{3/2}}x_0 + \frac{4(1+s)}{n(1-s)^{3/2}}\dot{y}_0\right. \\
&\left. + \frac{1}{2}\frac{A_{J2}(2ns - 3k\sqrt{1+s} + 2n)\sin^2 i}{k\sqrt{1-s}(-n^2+n^2s+4k^2)}\right)\sin\left(nt\sqrt{1-s}\right) \\
&- \frac{1}{8}\frac{A_{J2}(5n^2s + 4k^2 + 3n^2 - 6nk\sqrt{1+s})\sin^2 i}{k^2(-n^2+n^2s+4k^2)}\sin\left(2kt\right) \\
&- \frac{2\sqrt{1+s}}{n(s-1)}\dot{x}_0\cos\left(nt\sqrt{1-s}\right) + \left(\frac{2n(5s+3)\sqrt{1+s}}{(s-1)}\dot{x}_0\right. \\
&\left. + \frac{5s+3}{s-1}\dot{y}_0 + \frac{A_{J2}\sin^2 i}{4k}\right)t + \frac{2\sqrt{1+s}}{n(s-1)}\dot{x}_0 + y_0
\end{aligned}
\tag{2.13}
$$

$$
z(t) = z_0\cos\left(nt\sqrt{1+3s}\right) + \frac{\dot{z}}{n\sqrt{1+3s}}\sin\left(nt\sqrt{1+3s}\right) \tag{2.14}
$$

with the terms s, c, k, and A_{J2} defined as follows:

$$
s = \frac{3J_2R_\oplus^2}{8r^2}(1 + 3\cos 2i) \tag{2.15}
$$

$$
c = \sqrt{1+s} \tag{2.16}
$$

$$
k = nc + \frac{3\sqrt{\mu}J_2R_\oplus^2}{2\|\delta r\|^{7/2}}\cos^2 i \tag{2.17}
$$

$$
A_{J2} = -3n^2J_2\frac{R_\oplus^2}{\|\delta r\|} \tag{2.18}
$$

$R_\oplus$: Mean equatorial radius of central body

J_2 : Measure of central body oblateness

Propagating a set of initial conditions using the standard linearized C-W equations (Equations 2.6 – 2.7), and the non-linear C-W equations with J2 perturbations (Equations 2.12 – 2.14), the trajectories over a period of 3 orbits can be seen in Fig. 2.6. All trajectories in orbit will drift naturally over time, however it should be noted that when taking into account the J2 perturbation of Earth's gravity (the dominant gravitational perturbation), the direction of drift changes. This is significant for any station-keeping schema, and must be taken into account, as will be done for this analysis.

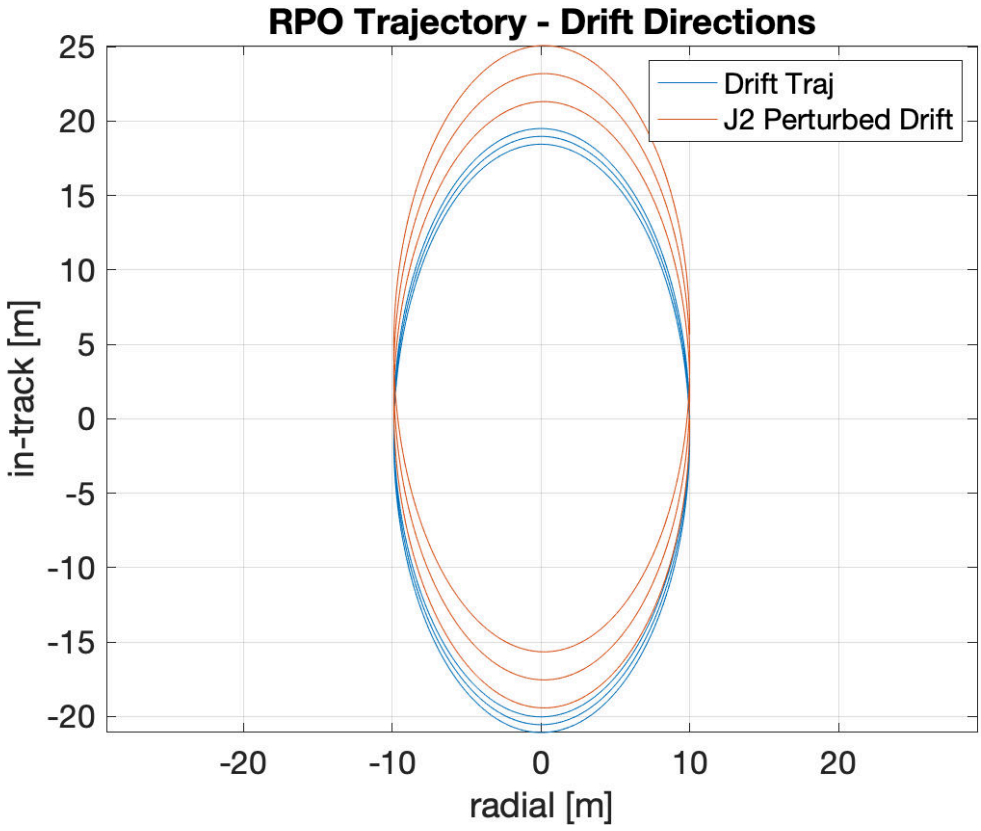


Figure 2.6: Trajectory Drift for different gravity models

2.1.4 Spherical Harmonic Representation of Earth's Gravitational Field

Planetary gravitational fields, like the Earth's, are not perfectly symmetric. The Earth is not a perfect sphere, nor is its mass evenly distributed, thus it has a non-uniform gravitational field. These gravitational perturbations can be described using a spherical harmonic gravitational model, which is a method of adding up progressively higher degrees of Legendre polynomial equations to represent an irregular but spherical object, similar to how a Fourier transform can represent a sinusoidal function using polynomials [98]. This gravity model is described as follows, in terms of the gravitational potential function, rather than as a force, as derived by Vallado [51].

$$\begin{aligned} U = \frac{\mu}{r}\Bigg[1 - \sum_{l=2}^{\infty} J_l \left(\frac{R_\oplus}{r}\right)^l P_l[\sin(\phi_{gc_{sat}})] \\ + \sum_{l=2}^{\infty}\sum_{m=1}^{l} \left(\frac{R_\oplus}{r}\right)^l P_{l,m}[\sin(\phi_{gc_{sat}})]\left\{C_{l,m}\cos(m\lambda_{sat}) + S_{l,m}\sin(m\lambda_{sat})\right\}\Bigg] \end{aligned} \tag{2.19}$$

Where J, the zonal coefficient, is defined as:

$$J_l = -C_{l,0} \tag{2.20}$$

Note also that $\phi_{gc_{sat}}$ and λ_{sat} are, respectively, the geocentric latitude and longitude of the spacecraft. If only latitudinal variations of the gravitational field are being considered, then only the *zonal* coefficients and the first summation is required. The second, double summation is referred to as the *tesseral* component, and considers longitudinal variations of the gravitational field.

The $P_{l,m}[\sin(\phi_{gc_{sat}})]$ coefficients are the Legendre Polynomials, mapped for spacecraft latitude in spherical coordinates, the first few of which are listed in Table 2.1 below:

Table 2.1: **Associated Legendre Polynomials.** This table gives a few sample expansions for the associated Legendre function, with geocentric latitude used [51].

$P_{0,0}$	1	$P_{3,2}$	$15\cos^2(\phi_{gc_{sat}})\sin(\phi_{gc_{sat}})$
$P_{1,0}$	$\sin(\phi_{gc_{sat}})$	$P_{3,3}$	$15\cos^3(\phi_{gc_{sat}})$
$P_{1,1}$	$\cos(\phi_{gc_{sat}})$	$P_{4,0}$	$\frac{1}{8}\{35\sin^4(\phi_{gc_{sat}}) - 30\sin^2(\phi_{gc_{sat}}) + 3\}$
$P_{2,0}$	$\frac{1}{2}\{3\sin^2(\phi_{gc_{sat}}) - 1\}$	$P_{4,1}$	$\frac{5}{2}\cos(\phi_{gc_{sat}})\{7\sin^3(\phi_{gc_{sat}}) - 3\sin(\phi_{gc_{sat}})\}$
$P_{2,1}$	$3\sin(\phi_{gc_{sat}})\cos(\phi_{gc_{sat}})$	$P_{4,2}$	$\frac{15}{2}\cos^2(\phi_{gc_{sat}})\{7\sin^2(\phi_{gc_{sat}}) - 1\}$
$P_{2,2}$	$3\cos^2(\phi_{gc_{sat}})$	$P_{4,3}$	$105\cos^3(\phi_{gc_{sat}})\sin(\phi_{gc_{sat}})$
$P_{3,0}$	$\frac{1}{2}\{5\sin^3(\phi_{gc_{sat}}) - 3\sin(\phi_{gc_{sat}})\}$	$P_{4,4}$	$105\cos^4(\phi_{gc_{sat}})$
$P_{3,1}$	$\frac{1}{2}\cos(\phi_{gc_{sat}})\{15\sin^2(\phi_{gc_{sat}}) - 3\}$		

Further coefficients can be determined using recursive relations from the first three entries, saving computational time and system memory for high-fidelity computations. Additionally, the C and S coefficients are required to solve for the potential, and these are determined empirically, and are specific to the gravitational field in question. For Earth, this can be found in data from the GRACE mission by NASA and UT Austin, up to the 2160^{th} degree [99,100]. For the purposes of this analysis, a fourth order analysis using both zonal and tesseral terms is used for orbits in MEO and GEO, whereas a second order analysis with only zonal terms is used for LEO orbits, due to the negligible variations of longitudinal perturbations in LEO compared to the latitudinal perturbations (J2 effect). See Appendix B for a list of coefficients used.

To recover the gravitational acceleration from the potential function, which is useful for numerical integration of the equations of motion, the gradient of the potential is taken:

$$\vec{a} = \frac{\partial U}{\partial r}\left(\frac{\partial r}{\partial \vec{r}}\right)^T + \frac{\partial U}{\partial \phi_{gc_{sat}}}\left(\frac{\partial \phi_{gc_{sat}}}{\partial \vec{r}}\right)^T + \frac{\partial U}{\partial \lambda_{sat}}\left(\frac{\partial \lambda_{sat}}{\partial \vec{r}}\right)^T \tag{2.21}$$

Where r is the magnitude of the position vector, and $\vec{r}$ is the position vector in Cartesian coordinates.

Additionally, though the perturbations due to the non-spherical nature of Earth's gravitational field are the most significant perturbations in LEO, in GEO they are joined by additional perturbations from the Sun and the Moon. These are the Sun-Moon gravitational perturbations, and the Solar Radiation Pressure perturbations, which are described in equations 2.22 and 2.23, respectively [101].

$$\vec{a}_{sun-moon} = -GM_{\odot}\frac{\vec{r}_{sun-earth} + \vec{r}}{\|\vec{r}_{sun-earth} + \vec{r}\|^3} - GM_{\leftmoon}\frac{\vec{r}_{moon-earth} + \vec{r}}{\|\vec{r}_{moon-earth} + \vec{r}\|^3} \tag{2.22}$$

where $\vec{r}_{sun-earth}$ is the vector pointing from the Sun to the Earth, $\vec{r}_{moon-earth}$ is the vector pointing from the Moon to the Earth, and $\vec{r}$ is the vector from the Earth to the spacecraft.

$$\vec{a}_{SRP} = -(1-\beta)\frac{F_s}{c}\frac{A_c}{m}\frac{\vec{r}_{sun-earth} + \vec{r}}{\|\vec{r}_{sun-earth} + \vec{r}\|} \tag{2.23}$$

Where F_s is the solar flux at the orbital altitude of the spacecraft, A_c is the cross sectional area of the spacecraft, as viewed from the Sun, c is the speed of light, m is the mass of the spacecraft, and β is the angle of the Sun's elevation with respect to the spacecraft's orbital plane.

Note that in both Equation 2.23, and the solar term in Equation 2.22, the vector $\vec{r}$ from the Earth to the spacecraft can be safely neglected for simplicity, as it is orders of magnitude smaller than the distance from the Sun to the Earth. This is not, however, the case for the lunar component of Equation 2.22.

2.2 Formation Flying

Formation flying, when applied to spacecraft, is the act of two or more spacecraft operating in close proximity to each other to hold a pre-defined configuration. Typically, formation flying is a static configuration, where spacecraft in formation maintain a static formation geometry, usually for the purposes of gathering and comparing data [29, 102]. A prime example of this is the Magnetospheric Multiscale (MMS) Mission by NASA, a robotic space mission to study the Earth's magnetosphere using four identical spacecraft flying in a tetrahedral formation [103]. This is in contrast to swarm operations, which are dynamic congregations of spacecraft, rather than static configurations.

Although swarm operations are more complex than traditional formation flying, techniques used in formation flying can be used as a foundation upon which to build a swarm control methodology. This includes the use of kalman filtering to improve the accuracy of position sensors, especially the use of GPS at high altitudes [104]. Formation flying missions over the past 40 years [105, 106] have proven that cooperative operations between spacecraft in LEO, GEO, and even HEO are possible, and can be done with minimal propellant utilization [107–109].

Formation flying has been used to maintain static configurations of small numbers of satellites [110,111], typically to perform precision measurement of planetary properties, such as the Earth's gravitational or geomagnetic field [99, 103]. In order to achieve such missions, where relative distances between the spacecraft needed to be precisely maintained for significant periods of time, new techniques were developed to be able to accurately determine the absolute and relative position of each spacecraft using limited data, such as high-altitude GPS data. At high altitudes, GPS data is increasingly limited, and above MEO, where the GPS satellites reside, GPS data can only be gleaned from signals transmitted on nearly the opposite side of the Earth. This results in huge errors in the position knowledge of each spacecraft that need to be corrected, using Kalman filtering with additional inertial sensors on-board each spacecraft. NASA, as the designer

and operator of many of these historic formation flying missions, developed a set of algorithms to reduce this inertial position error significantly, achieving 50 m accuracy at and above GEO orbit. This is referred to as the Goddard Enhanced Onboard Navigation System (GEONS), an extended Kalman filter coupled with a high-fidelity dynamics model to process GPS pseudorange measurements referenced to the onboard clock [112, 113].

The GEONS method, and similar applications for LEO, form the basis of the inertial measurement system to determine the position of each member of the swarm in inertial space. For the purposes of this analysis, Kalman filtering of GPS data is not actually performed in the simulations, since prior research and experimental results have shown that GPS data can be resolved to 3 m resolution in LEO, and 50 m in GEO [113,114]. Instead, the focus in this analysis is on improving the relative motion measurements between swarm members in the LVLH relative-motion reference frame, as inertial measurements are only needed periodically to account for long-term drift, while real-time swarm operations require a high-precision knowledge of the relative positions and velocities of all spacecraft in the swarm.

During the course of the TanDEM-X/TerraSAR-X mission, operated by DLR, D'Amico and Montenbruck developed a system to use eccentricity and inclination vector separation in LEO, a method previously only used in GEO, to enable multiple spacecraft to fly in formation with virtually no collision risk [115]. This method functions by aligning the relative eccentricity and inclination vectors of the orbits of two spacecraft in close-proximity to be parallel (or anti-parallel). So long as the reference trajectory is a circular orbit and the radial and cross-track components of relative position and velocity can be determined to a higher degree of accuracy than the in-track separation, then natural orbital perturbing forces will maintain adequate separation between the two spacecraft. This method of formation control, although not well suited for large numbers of spacecraft, can be used as a good starting point for a higher fidelity solver to take over and refine the solution for more efficient propellant utilization, and to extend to the case of non-circular mean trajectories.

2.3 Ground-Based Analogs

Although the use of multiple vehicles to perform operations cooperatively has not yet been demonstrated in space, these robotic operations have been the focus of unmanned aerial vehicle (UAV) research over the past decade, following the reduced cost of entry into the field of compact robotic avionics [116]. This reduction of technological and financial barriers to entry that hit the robotics field over the past 15 years is slowly making its way into the space industry, with the advent of CubeSats and the rise of space venture capital and space startups [117–120]. Although the space environment is much harsher than sea-level atmospheric environments, and requires specialized hardware to deal with these challenges, the overall principles of controlling large number of vehicles algorithmically is fundamentally quite similar between spacecraft and drones, making UAVs the perfect analog to describe and test swarm spacecraft operations.

An important aspect of ground-based robotics that lends itself to be ported to space-based applications is the concept of *formation control*. Over the past 20 years, with the increased availability of small and distributed computing systems, great strides have been made in research and demonstration of formation control; that is methods and algorithms to control the relative positions, velocities, and orientations of multiple robots working and moving together as a whole. Fierro et al. demonstrate theoretically and practically the ability to control a formation of three wheeled robots using vision-based navigation system to guide the trajectories around unforeseen obstacles [121, 122]. Similar methods have been created by others to control larger number of robots [123–134]. Although the kinematics and equations of motion for these formations are much simpler than that of the LVLH RPO environment in Earth orbit, the general methodology for formation control can be ported to space-based applications, providing a tested and proven jumping off point from which to begin developing more sophisticated models tailored to in-space manufacturing applications.

A common aspect shared by many of these formation control systems is the use of *graph theory* to distribute and assign roles to the various members in the swarm [135]. Graph theory is the use of mathematical structures to model pairwise relations between objects, resulting in a map of *vertices*, which are connected by *edges*. These edges are then given directions, resulting in the creation of *directed graphs*, visually denoted by arrows. A sample graph for a swarm can be seen in Figure 2.7. By employing graph theory, a hierarchy can be formed, where certain swarm members will follow the movements of other members that are leading. These leaders may in turn be followers themselves of other swarm members. This hierarchy enables a deterministic approach to solve the problem of how to automate obstacle avoidance without the avoidance maneuver posing a greater risk of self-collision within the swarm; using graph theory with a top-down hierarchy, each swarm member is able to perform its evasive maneuvering while knowing the maneuvering plan of those ahead of it in the queue. This enables reactive maneuvering without time-consuming negotiations between the swarm as a whole, allowing quick maneuvering to occur in real-time.

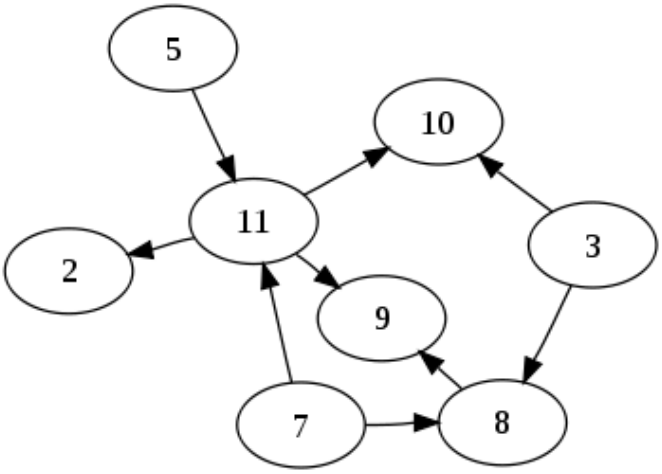


Figure 2.7: Directed Graph Diagram

Although this method works quite well on the ground, it is not quite suitable for space-based applications without first making a few modifications. When considering ground-based operations, such as UAVs, speed and reaction time is typically favored over resource utilization efficiency, since

the relative speeds between objects can be quite high, and obstacles in urban environments can appear and move quite rapidly. However, in space, it is rare to have an object of significant size enter the sphere of influence of a spacecraft without having some sort of advance warning. Additionally, consumables such as propellant are extremely valuable, given that there typically is no refueling capability available in orbit. This results in a scheme that does not favor graph theory methods, at least not globally, although it could be used in small local clusters, while promoting swarm control methods that involve a great deal of cooperation and negotiation between the spacecraft. This would enable a trajectory solution that is acceptable for all spacecraft, minimizing the ΔV requirements overall. Numerous methods of formation control have been devised and tested over the years that can satisfy these constraints for in-space operations, many of which have been catalogued and compared by Brambilla et al. [136]. The most intriguing amongst these is the application of evolutionary robotics to solve the problem of efficient and recurring trajectory generation for swarms of spacecraft in Earth orbit. This dissertation will investigate a trajectory generation and control method using *Genetic Algorithms* for in-space swarms.

2.3.1 Insect Swarm Comparisons

In addition to comparisons to drone-based swarms, comparisons can also be made to insect swarms, specifically swarms of bees. Colonies of bees exhibit collective behaviors, where multiple agents carry out their individual tasks which add up to an overall goal [137], similar to the concept behind swarm spacecraft for in-space manufacturing. Looking at swarms of bees for inspiration, certain concepts of the colony behavior were implemented into the spacecraft swarm control framework, such as the distinction between *Drones* and *Workers*, and the presence of intra-swarm communication for task coordination and collision avoidance [138]. Bees separate colony members into *Drones* and *Workers*, where the *Drones* perform the day-to-day work (aggregation, in the case of spacecraft swarms), and *Workers* take on the task of maintaining the swarm (logistics and communication nodes, in the case of spacecraft swarms). They communicate with each

other, a necessary tactic to complete a common task and prevent collisions between other bees, a concept which has also been ported into drone swarms, and will be demonstrated for spacecraft swarms in this thesis. Although the majority of the analogues used to build the spacecraft swarm methodology come from drone swarm technology, it is important to remember that nature has had millions of years of evolution to come up with the optimal solution for how to operate colonies of workers, and this can be leveraged to our advantage by studying them [139–144].

2.4 Kalman Filtering

In real-world operations, it is impossible to know the position and velocity of a spacecraft with 100% precision. Position and velocity are measured using on-board sensors, which have inherent error tolerances. These errors, from inputs such as GPS and relative Radar ranging, result in a covariance matrix attached to the state vector for each spacecraft in the swarm. These covariance matrices can be computed not only with respect to an inertial state, but also between each spacecraft in the swarm. This means that if the covariance between spacecraft A and B is desired, sensor fusion between all other spacecraft and spacecraft B can be used to refine the state vector and covariance matrix of spacecraft B, thus minimizing the measurement error. By combining the measurements from multiple spacecraft, the measurement accuracy can be improved over a simple computation of the covariances on each spacecraft separately.

A Kalman filter can be used to reduce the error in an estimated state by propagating a set of points through time, each corresponding to the boundaries of the covariance "bubble of uncertainty" that surrounds the spacecraft. As this is propagated, the covariance ellipsoid is refined by using measurements taken from a sensor at a known position, with a known precision. This is used successively over time to predict what states are more likely, and which are less likely, using a weighted scheme to determine where the spacecraft lies within a 3-sigma Gaussian distribution. The Kalman filtering method is useful for not only simulations, but for real-time

operations, since the computational cost of the algorithm is very low, and can be run in real-time onboard a satellite.

2.4.1 Mathematical Formulation

There are two common types of Kalman filters used in practice, an extended Kalman filter (EKF) and an unscented Kalman filter (UKF). The EKF, although more computationally efficient, requires that the Jacobian matrix of the equations of motion be known and well defined, which is quite complex to derive for the nonlinear perturbation equations of RPO. Instead, the UKF uses what is known as sigma-points, a set of virtual points surrounding the unknown object at a 3-sigma distance, which are then propagated using the equations of motion to determine the covariance drift over time. While this is more computationally intensive than modelling the covariance drift using a Kalman filter, it is less sensitive to nonlinear changes in the system, and can be computed in real-time without a-priori knowledge of the Jacobian matrix [145].

To run a step of a Kalman filter, first the previous state is needed. This can either be the initial state, or the end state of a previous step of the Kalman filter. We'll define these known quantities using $\hat{x}^+_{0,k-1}$ for the spacecraft position, and $\hat{x}^+_{i,k-1}$ for each of the sigma points. Note that there are two sigma points for each dimension of the problem, including position, velocity, and noise dimensions. The plus sign denotes that this is a corrected estimate, and thus has been run through a filter (or is the initial state). The $k-1$ indicates that it is from the previous time-step, and the hat denotes that it is an estimate generated by the filter, and not a raw measurement from a sensor. Over time, when using a Kalman filter, the estimate will converge to a minimal covariance offset from the truth value so long as the system remains observable [146]. This minimal covariance will depend on the accuracy of the measurement sensors, and the process noise variances – how much the measurements should be trusted above the model.

First, the points can be run through a propagation function, which in this case is the equations of motion for RPO with gravitational perturbations.

$$\hat{x}_{i,k}^{-} = f(\hat{x}_{i,k-1}^{+}, q_{i,k-1}) \tag{2.24}$$

Here, $q_{i,k-1}$ represents the process noise, which are unknowns that affect the propagation. Typically, in terrestrial applications, this is attributed to wind or other variable sources. There are few of these sources in LEO, though this could be used to model aberrations in atmospheric drag, or thruster variations during a maneuver. For our purposes, the process noise is left as zero for simplification.

Once the sigma points have all been calculated and propagated, the mean predicted state is computed as a weighted average of all the sigma points.

$$\hat{x}_{k}^{-} = W_0 \hat{x}_{0,k}^{-} + W \sum_{i=1}^{n} \hat{x}_{i,k}^{-} \tag{2.25}$$

Where W_0 and W are the weights associated to the middle point and the sigma points, respectively. Using this weighted information, the sample covariance can be computed around this new covariance, with the covariance weights W_c and W, similar to the mean predicted state

$$P_{k}^{-} = W_c(\hat{x}_{0,k}^{-} - \hat{x}_{k}^{-})(\hat{x}_{0,k}^{-} - \hat{x}_{k}^{-})^T + W \sum_{i=1}^{n} (\hat{x}_{i,k}^{-} - \hat{x}_{k}^{-})(\hat{x}_{i,k}^{-} - \hat{x}_{k}^{-})^T \tag{2.26}$$

The next step, now that the estimated position and covariance has been computed, is to use this knowledge to attempt to correct the estimated position, to get our predicted state. To do this, the overall predicted measurement $\hat{z}_k$ is computed

$$\hat{z}_{i,k} = h(\hat{x}_{i,k}^{-}, r_{i,k}) \tag{2.27}$$

Where $r_{i,k}$ is the *measurement noise* with covariance $\boldsymbol{R}$, a property of the sensors in use. Then the weighted mean of this predicted measurement is computed:

$$\hat{z}_k = W_0 \hat{z}_{0,k} + W \sum_{i=1}^{n} \hat{z}_{i,k} \tag{2.28}$$

This predicted measurement is then used to correct the state using y_k, which is the *innovation vector*, or residual. This is the vector pointing from the predicted measurement to the actual measurement, which is scaled using $\boldsymbol{K}$ the *Kalman gain*.

$$y_k = z_k - \hat{z_k} \tag{2.29}$$

$$\hat{x}_k^+ = \hat{x}_k^- + K y_k \tag{2.30}$$

The Kalman gain is a scaling matrix that enables mapping of the estimates to the true location, based on the covariance matrices. Thus, to compute the predicted measurement, and move forward to the next time-step in the filter, the Kalman gain must first be computed. This is done by first computing the weighted sample covariances of both the estimated measurement value with the estimated position, and the estimated measurement value with itself.

$$P_{xy} = W_c(\hat{x}_{0,k}^- - \hat{x}_k^-)(\hat{z}_{0,k}^- - \hat{z}_k^-)^T + W \sum_{i=1}^{n} (\hat{x}_{i,k}^- - \hat{x}_k^-)(\hat{z}_{i,k}^- - \hat{z}_k^-)^T \tag{2.31}$$

$$P_{yy} = W_c(\hat{z}_{0,k}^- - \hat{z}_k^-)(\hat{z}_{0,k}^- - \hat{z}_k^-)^T + W \sum_{i=1}^{n} (\hat{z}_{i,k}^- - \hat{z}_k^-)(\hat{z}_{i,k}^- - \hat{z}_k^-)^T \tag{2.32}$$

The Kalman gain is then the ratio of these two covariance matrices

$$K = P_{xy} P_{yy}^{-1} \tag{2.33}$$

Finally, the covariance values need to be corrected, similar to how the position estimates were corrected, in order to be used in the next time-step of the filter.

$$\hat{x}_{i,k}^{+} = \hat{x}_{i,k}^{-} + K(z_k - \hat{z}_{i,k}) \tag{2.34}$$

The new covariance matrix can be computed as a sample covariance of the corrected sigma points, P_k^+:

$$P_k^+ = W_c(\hat{x}_{0,k}^+ - \hat{x}_k^+)(\hat{x}_{0,k}^+ - \hat{x}_k^+)^T + W\sum_{i=1}^{n}(\hat{x}_{i,k}^+ - \hat{x}_k^+)(\hat{x}_{i,k}^+ - \hat{x}_k^+)^T \tag{2.35}$$

Which can be simplified to be:

$$P_k^+ = P_k^- - KR_kK^T \tag{2.36}$$

Where R_k is the covariance of the measurement error, taken from the datasheet of whatever sensor is in use (in this case a radar or LIDAR sensor). The covariance values can then be used to generate an error ellipse, or *egg of death*, around the spacecraft, which provides the volume of space within which we expect to find the spacecraft with 3σ precision.

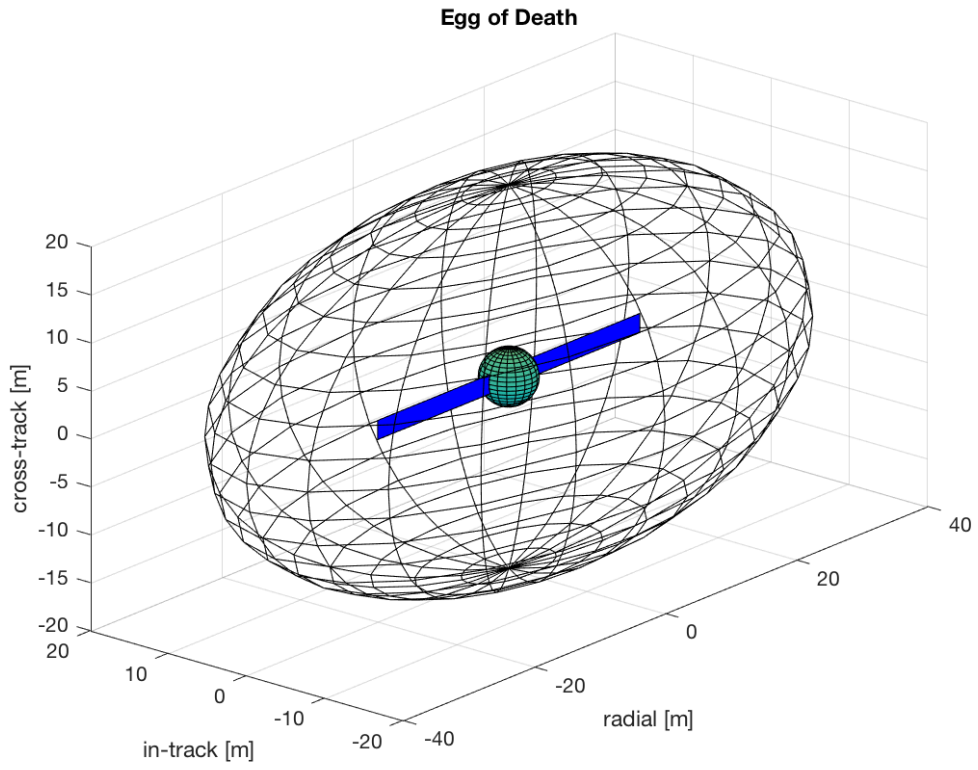


Figure 2.8: Example Error Ellipse for a Satellite

Figure 2.8 above shows an example egg of death around a satellite. The size of the egg is primarily based on two factors; the measurement error from the radar ranging system used to determine the position and velocity of the satellite, and the propagation of error between measurements. The measurement error (covariance) itself cannot be removed, but it can be compensated for and narrowed down using a Kalman filter to sort out readings that are not very likely given the physics of orbital mechanics.

2.4.2 Sensor Fusion Kalman Filter

In order to take into account the shared data of the swarm, which is the combination of the radar ranging sensors on each spacecraft, a sensor-fusion Kalman Filter is used. This is a modification of the standard Kalman filter described above, which uses multiple measurement update cycles

to incorporate the shared data of the swarm to further refine the covariance ellipsoid for each spacecraft. Figure 2.9 shows an example of the sensor fusion process, where the covariance of the position between *Sat #1* and the *Client* spacecraft can be improved by fusing the data from all the swarm spacecraft, even taking into account the GPS position errors defining the locations of each swarm spacecraft with respect to *Sat #1*.

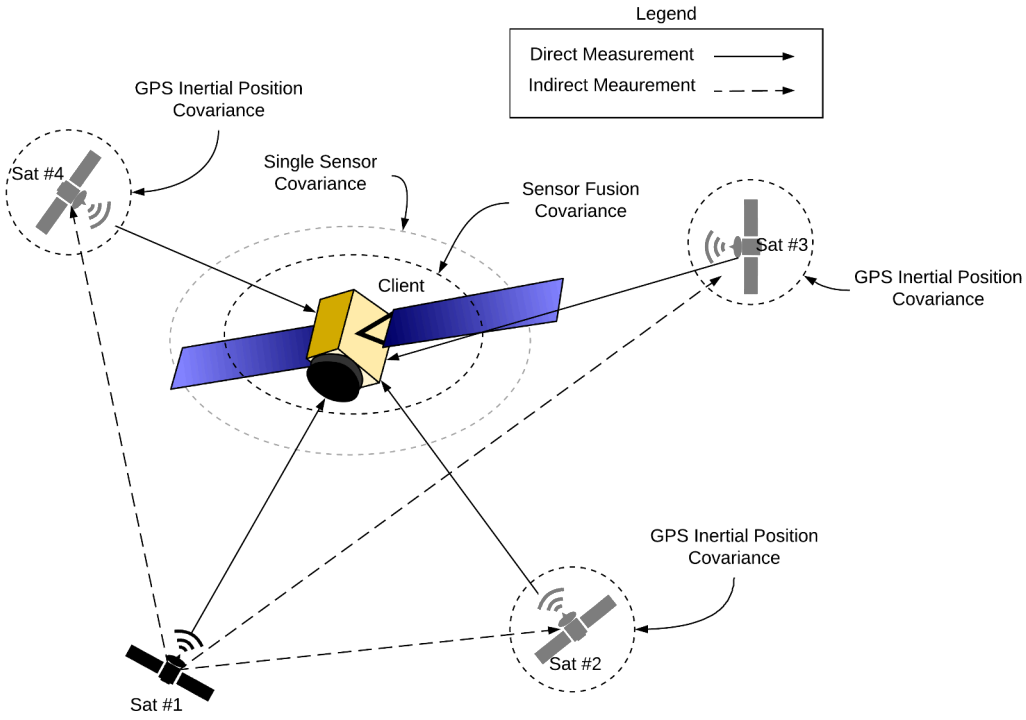


Figure 2.9: Sensor Fusion Diagram

In order to take into account multiple sensor inputs, the unscented Kalman filter algorithm itself must be modified to be able to process this additional data. As described in Section 2.4.1 the Kalman filter operates by propagating an initial state in time using known equations of motion, and then a position and covariance update step is performed using data from an onboard sensor to filter out noise and erroneous data from the sensor data. With a sensor fusion Kalman filter, the propagation step remains the same (see Equation 2.24), however the correction and update step

(see Equation 2.30) is performed multiple times – once for each sensor. This process is depicted in Figure 2.10 using pseudocode.

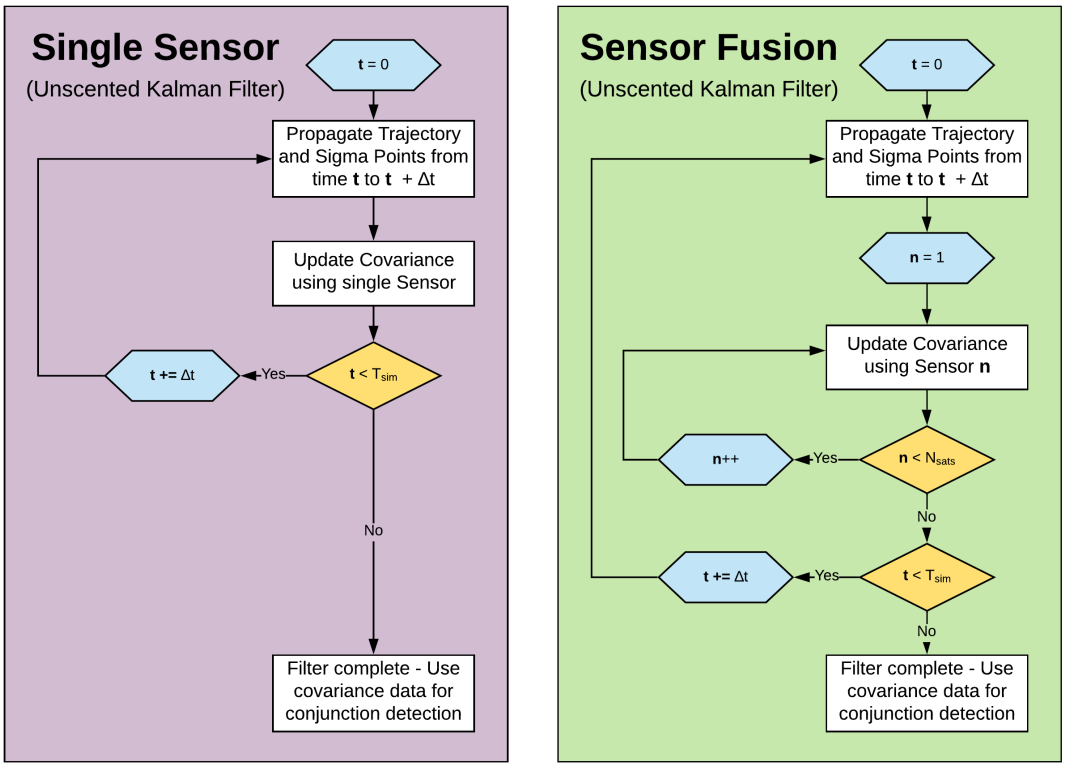


Figure 2.10: Sensor Fusion Kalman Filter Process

Since the majority of the computation time in the Kalman filter is spent on the propagation step, rather than the update step, repeating the update step adds very little computational overhead, while greatly improving the spacecraft covariance.

2.5 Genetic Algorithms

Genetic Algorithms (GAs) are a method of optimization, applicable to a wide variety of problems, that use a process similar to Darwinian evolution to *evolve* a set of random (or pseudo-random) initial conditions to find an acceptable solution, or even a globally optimal solution, to a problem

[147]. These initial conditions form the initial population, of size N_{pop}. This initial population is then propagated, in this case using the C-W equations, to the final state at time t_f. Fig. 2.11 shows a depiction of the GA process, explained in detail in the following section.

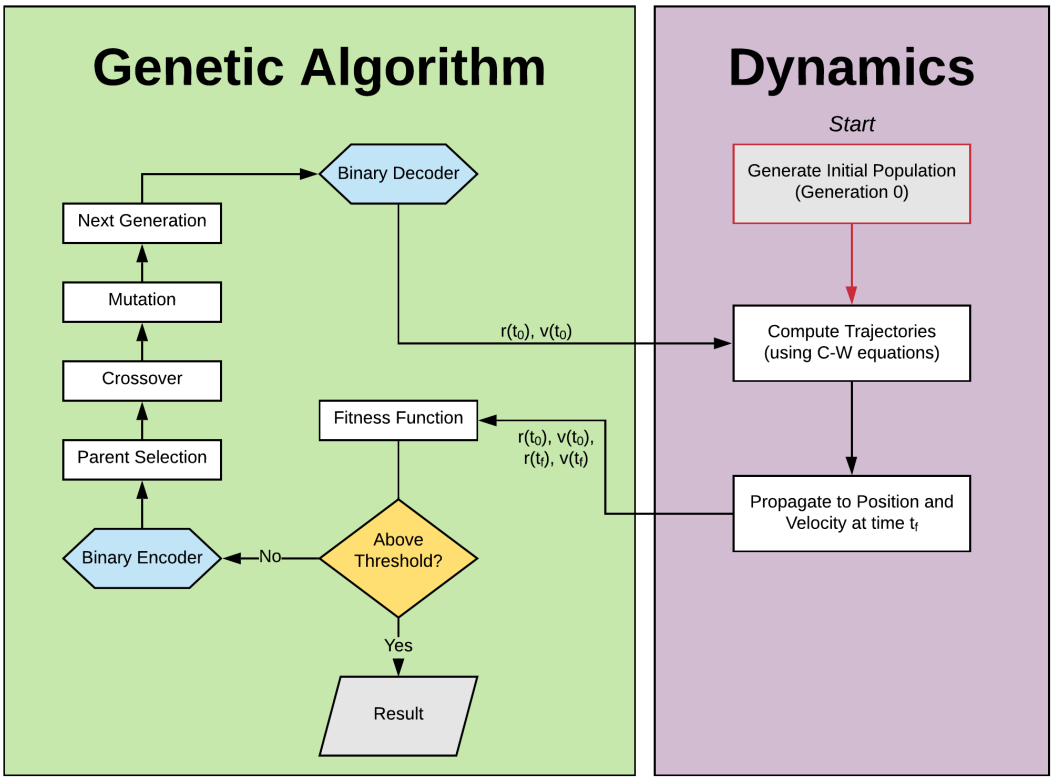


Figure 2.11: GA Example Flowchart

Once the initial population is created and propagated, the solutions are ranked based on how close they come to the desired solution, using a fitness function. For simplicity, we created our fitness function such that it ranges from 0 to 1, where a value of 0 has no attributes of a desired solution, and a value of 1 is the desired solution. For the initial problem of finding closed and repeating trajectories in the LVLH frame, this was defined as:

$$F = \left(1 + C_r \left\|\vec{r}(t_f) - \vec{r}(t_0)\right\| + C_v \left\|\vec{v}(t_f) - \vec{v}(t_0)\right\|\right)^{-1} \tag{2.37}$$

where

C_r : coefficient of position

C_v : coefficient of velocity

Given a start time t_0 and end time t_f, Equation 2.37 defines a fitness function that prefers solutions that are closed trajectories. The closer the final conditions, $\vec{r}(t_0)$ and $\vec{v}(t_0)$, are to the initial conditions, $\vec{r}(t_f)$ and $\vec{v}(t_f)$, the higher the fitness function's value will be, since a desired solution is one where the final conditions and initial conditions are the same. Once the population members are ranked based on their fitness, the bottom half is culled as they are not desirable solutions. However, we need to rebuild the population back to size N_{pop} for the next generation ($N_{pop} = 200$ in our case), so this is where genetic crossover is implemented. To perform crossover, each member of the population (known as a chromosome) should be represented in binary notation in order to represent the data with the most number of genes (string elements), since binary is lowest-order possible data-encoding scheme, with a radix of 2. In the case of swarm trajectories, where our population is composed of 3 position and 3 velocity variables, each of these are represented in binary as 16 bit floats and appended to form a 96 bit string, called a *chromosome*, seen in Fig. 2.12.

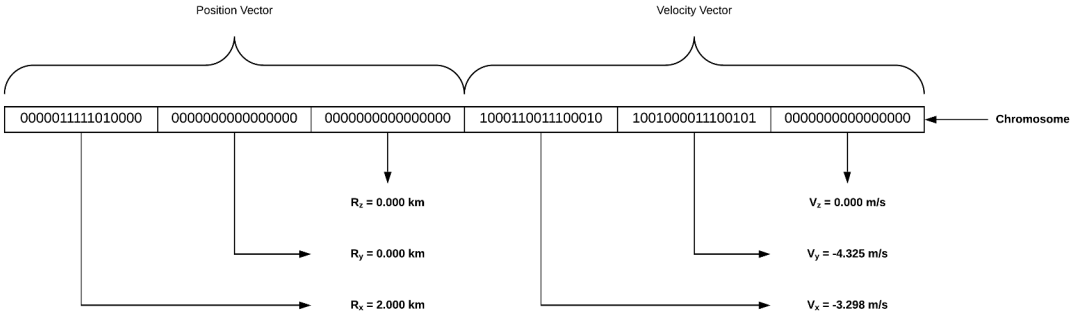


Figure 2.12: GA Binary Representation

Crossover is then performed by choosing two of the remaining solutions as *parents*, and taking portions of their *chromosomes* (in this case represented as bits) to form members of the next

generation. There are many methods of genetic crossover that can be used in GAs, the simplest of which is random pairing [148]. As random pairing is inefficient at reaching a solution, our method uses roulette selection, which assigns a weighting factor to each parent based on their fitness values. Then pairs are selected to be mated using the weighting factors, such that the chance of selecting a parent with a fitness value of 0.5 is five times higher than selecting one with a fitness value of 0.1. When mating pairs for the crossover, a random number between 1 and 95 is selected for each crossover event, to determine at which point in the *chromosome* to cut and swap, as depicted in Fig. 2.13. Then, the *chromosomes* of each of the two parents are cut at this crossover point and swapped to make two new *offspring*. This is done until the population size has been rebuilt to N_{pop} for the next generation's computations.

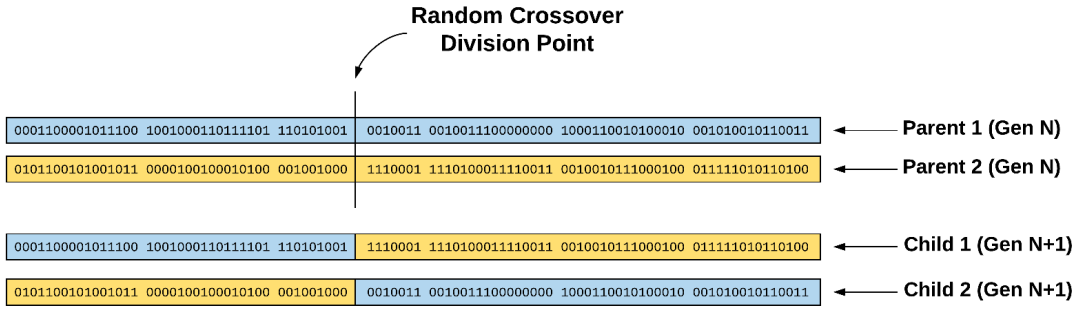


Figure 2.13: GA Crossover Example

After crossover is completed, the final step of the GA sequence is to perform a mutation on the chromosomes. The crossover process spreads genetic diversity throughout the population, but does not introduce any new possibilities to the population. This is where mutation comes in; mutation allows new structures or solutions to appear by randomly flipping bits throughout all the chromosome. A variable, p_{mut}, is used to control this probability, and thus a small subset of all bits in all chromosomes are flipped, introducing new and random solution possibilities (see Fig. 2.14). Good mutations will survive to the next generation and undesirable mutations will not, by means of the fitness function. Although mutation is an important part of the GA process,

it must be used sparingly to avoid conflicting with the crossover process. In this case, we use a probability of mutation of 0.2% ($p_{mut} = 0.002$).

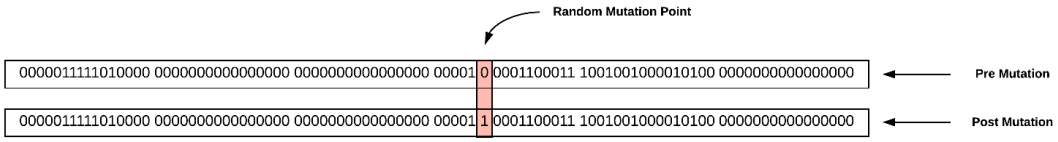


Figure 2.14: GA Mutation Example

Once the mutation is completed, the binary data is then decoded back into their separate variables, and the process begins again for the next generation. This process continues until a fitness value of one is achieved for a member of the generation, or the maximum number of generations has been reached ($N_{gen} = 100$). In practice, however, a threshold must be specified, since it is impossible to converge to an exact solution [148]. For an accurate solution, a threshold of 0.001 is used; however, in practice it is more computationally efficient to use a threshold of 0.01 to get near the solution and use another targeted optimization technique to further refine the solution. This is due to the fact that the Genetic Algorithm (GA) method is designed to search across the entire solution space and find a solution among many possible solution spaces, and thus is very good at identifying the location of an optimal solution, but lacks efficiency in arriving at the exact solution itself [147].

Chapter 3

Generating Initial Trajectories for the Spacecraft Swarm

3.1 Defining the Swarm Parameters

The first step in solving for a set of trajectories for a spacecraft swarm is to define the constraints and requirements of the swarm. This includes identifying the number of spacecraft in the swarm operation, what range restrictions (if any) are assigned to each spacecraft (in the LVLH coordinate system), and how much the entire maneuver ΔV limit is (if any).

These constraints and requirements will vary depending on the nature of the swarm and the intended goal. For example, a swarm that is composed of building block spacecraft that are aggregating together to form a larger structure will have spatial constraints that require them to be in very close proximity to each other and to the central LVLH reference point. However, for a swarm of spacecraft that are observing and mapping a structure from a distance using remote sensors, each spacecraft will be far away from each other, and far away from the target structure, requiring stringent constraints to specify that. These requirements are enforced within the GA solver using a fitness function. See Appendix A.3 for a detailed software implementation of this method in Python.

3.2 Solving for Spacecraft Swarms

3.2.1 Overview of Trajectory Generation

Once the constraints for a swarm have been defined, the genetic algorithm solver is used to create a set of trajectories satisfying all the constraints. This is an iterative process which yields an initial state vector for each spacecraft, probabilities for collision, and estimated insertion ΔV. Insertion ΔV is determined by assuming all spacecraft are launched into LEO from the same launch vehicle, deposited at most 5 km in-track from the target point.

After the swarm is defined using initial state vectors, which are propagated using numerical integration of the equations defined in Section 2.1, additional changes to the swarm can be made using a separate set of Genetic Algorithms to determine how to modify the swarm when a spacecraft is added or removed, incurring the least amount of ΔV during the reconfiguration.

3.2.2 Initial Trajectory Generation

In order to solve for a set of trajectories for a swarm of spacecraft, multiple Genetic Algorithms are used, one for each spacecraft, all nested within a larger GA to de-conflict for collisions [47].

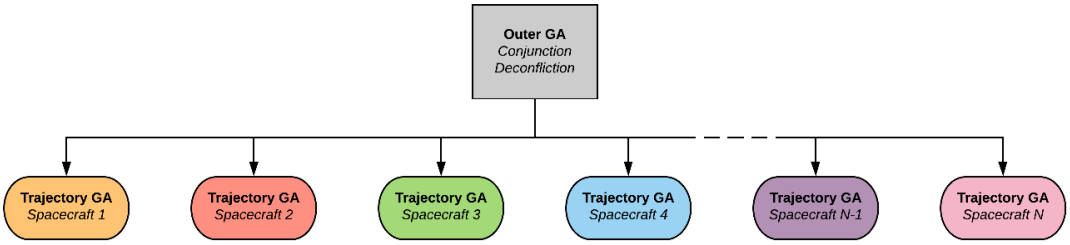


Figure 3.1: Hierarchy of Genetic Solvers

Each spacecraft is assigned its own fitness function defined by the mission requirements for a spacecraft. For example, a spacecraft that has a requirement to be within d_{max} but no closer than d_{min} from a Client spacecraft will have a fitness function as defined in Equation (3.1).

$$F = \left(1 + C_r \left\|\vec{r}(t_f) - \vec{r}(t_0)\right\| + C_v \left\|\vec{v}(t_f) - \vec{v}(t_0)\right\| + C_d \delta_{dist}\right)^{-1} \tag{3.1}$$

where

$$\delta_{dist} = \begin{cases} d_{min} - r_{min} & \text{if } r_{min} < d_{min} \\ r_{max} - d_{max} & \text{if } r_{max} > d_{max} \\ 0 & \text{otherwise} \end{cases}$$

C_r : coefficient of position

C_v : coefficient of velocity

C_d : coefficient of distance

r_{min} : closest range to Client spacecraft [km]

r_{max} : farthest range to Client spacecraft [km]

d_{min} : closest permissible distance to Client spacecraft [km]

d_{max} : farthest permissible distance to Client spacecraft [km]

Note that the three coefficients can be used to tweak which parameters are desired to be solved to a higher accuracy. By default they are all set to 1, but if velocity knowledge is valued at higher precision over position knowledge, then C_v can be set lower (e.g., $C_r = 1$ and $C_v = 0.5$ will result in a twofold increase in precision for velocity).

These separate fitness functions allow the optimizer to solve for each spacecraft using its own GA, which can be run in parallel to save on computation time. Once a trajectory is generated for each spacecraft (identified by its initial position and velocity vectors), the outer GA checks

for collisions. This is done by propagating each of the trajectories forward in time, sampling at a fixed timestep (60 s in our case). Trajectories are propagated to a set time interval, specified by the needs of the scenario, but no less than 24hrs. This ensures passively collision-free trajectories for at least 24hrs, giving ground operators buffer room to troubleshoot any anomalies before there is a collision risk within the swarm. Initially this propagation was done using the linearized C-W equations (see Equation 2.6), however when expanding the scope of the research to include gravitational perturbations due to a non-uniform central body, the errors in the approximations were found to be on the order of magnitude of these perturbations. This required implementing a propagation method that numerically integrates the perturbed equations of motion (see Equations 2.19 & 2.21). These position vs. time values are compared for all the spacecraft to determine if there is a chance of collision. Collision probability is determined by using a Sensor Fusion Kalman Filter propagation of the trajectory, using sensor inputs to compute the estimated positions of each spacecraft with only the information available to each spacecraft, and comparing the resulting covariance matrices for an overlap [149].

If there is a collision predicted, then the outer GA will isolate the two spacecraft that are involved in the collision and determine how to most efficiently mitigate it, as well as which spacecraft has the least restrictions on it to modify its trajectory. The simplest solution is not to change the trajectory at all, but instead adjust the insertion time of one into its trajectory so as to adjust its phase, thereby avoiding a collision. If this is not possible, or if this results in further collisions, then the solver will try slight variations of the trajectories, implementing a modified shooting method solver [150–152] to obtain one which does not result in any conjunction. During this modified trajectory search, existing methods for formation flying trajectory optimization are also applied concurrently, such as eccentricity/inclination vector alignment [115], albeit modified to prioritize lower ΔV over the course of the mission.

Running this for a set of 10 spacecraft, with zoning restrictions set out in Table 3.2.2 with respect to the Client spacecraft, and to avoid conjunctions within a 50 m buffer corridor from

each spacecraft, Fig. 3.2 shows a set of closed and repeating relative motion trajectories that satisfy this criteria. The trajectories shown are for a 10 day propagation of the initial states determined by the GA solver, during which the Sensor Fusion Kalman Filter determined that there was no probability of collision within a 3-sigma covariance. The full pseudocode for the trajectory generation algorithms can be found in Appendix A.1.

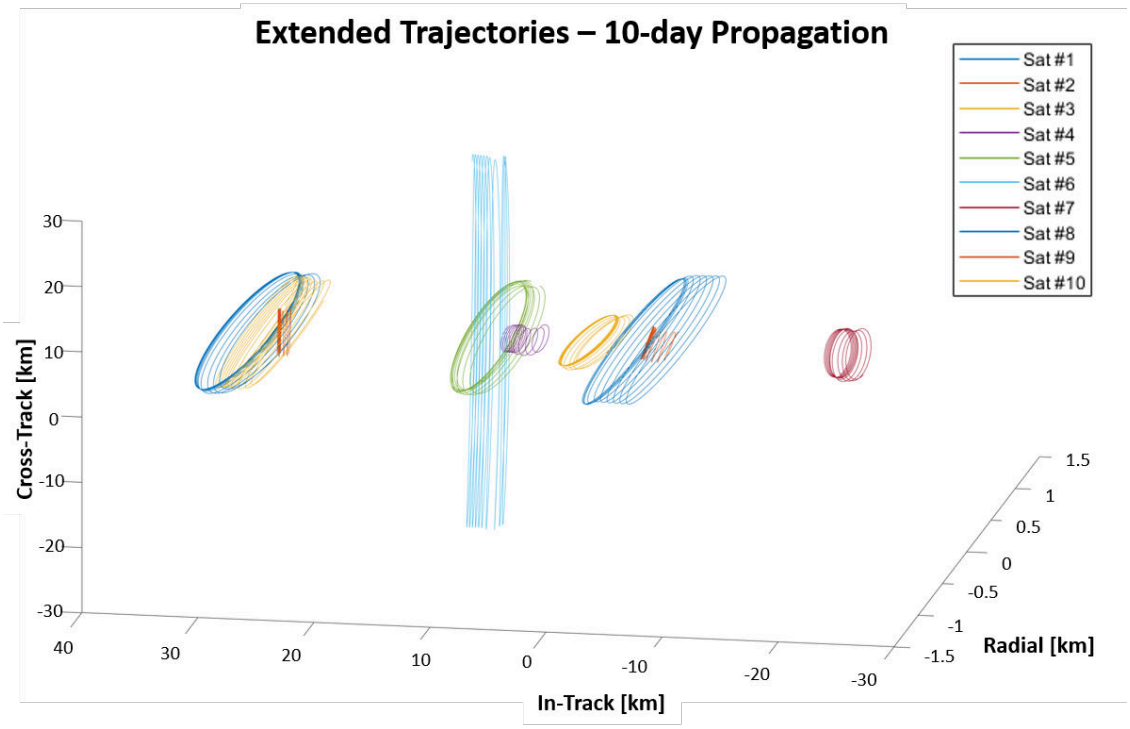


Figure 3.2: Swarm Solution for 10 Spacecraft

Table 3.1: Constraints for 10 Spacecraft Swarm Example

Swarm Member	**Constraint Ellipsoid Dimensions [km]**	**Constraint Ellipsoid Center [km]**
Spacecraft 1	[2,2,2]	[0,0,0]
Spacecraft 2	[2,2,2]	[0,-5,0]

Spacecraft 3	[3,3,3]	[0,2,0]
Spacecraft 4	[5,5,5]	[0,7,3]
Spacecraft 5	[2,2,4]	[0,10,0]
Spacecraft 6	[2,2,4]	[0,10,0]
Spacecraft 7	[2,2,4]	[0,-20,0]
Spacecraft 8	[5,5,5]	[0,30,0]
Spacecraft 9	[5,5,5]	[0,30,0]
Spacecraft 10	[5,5,5]	[0,30,0]

It should be noted, however, that this is not a unique solution. There is a family of an infinite number of solutions that satisfy this criteria, while only one of these that satisfies the constraints is required.

3.2.3 Trajectory Generation for Large Swarms

When dealing with large swarms of spacecraft (>20), it is advantageous to use parallel computing schemes to speed up the computations, as opposed to the previously discussed scenarios which were run on a laptop. Using parallel processing, it is possible to simulate real-world operations, where the computational load is distributed over the spacecraft in the swarm, while also enabling simulations to run faster than on a single machine. Larger swarm simulations were processed using a high-performance computing cluster at the USC Center for Advanced Research Computing (CARC). Figures 3.3 & 3.4 show results from a selection of these large swarm computations.

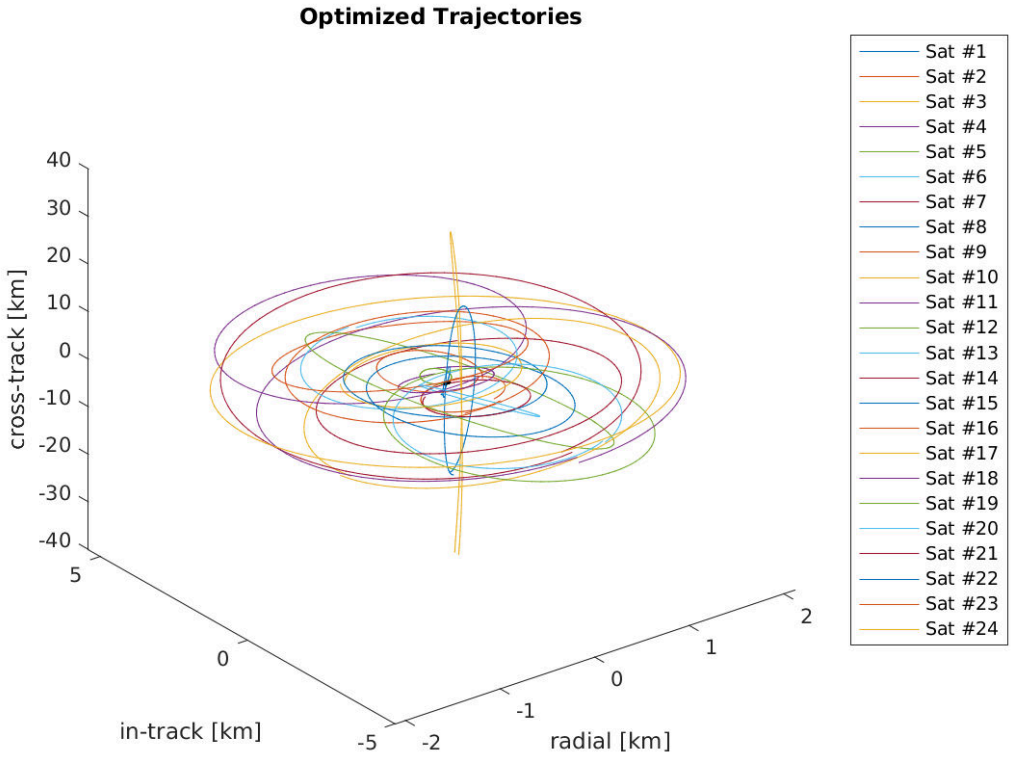


Figure 3.3: Swarm Solution for 24 Spacecraft – Computed Using Parallel Processing

In Figure 3.4, it can be seen that the visualization of large swarms on a single plot starts to become quite meaningless, as the trajectories seem to saturate the image into a solid set of colors. This illustrates the main issue with spacecraft swarm operations, and why a semi-autonomous method such as the one described in this thesis is needed to control these spacecraft and prevent collisions. The amount of data is too much for a team of mission operators on the ground to control without the aid of software platforms to maintain the myriad of day-to-day functions autonomously.

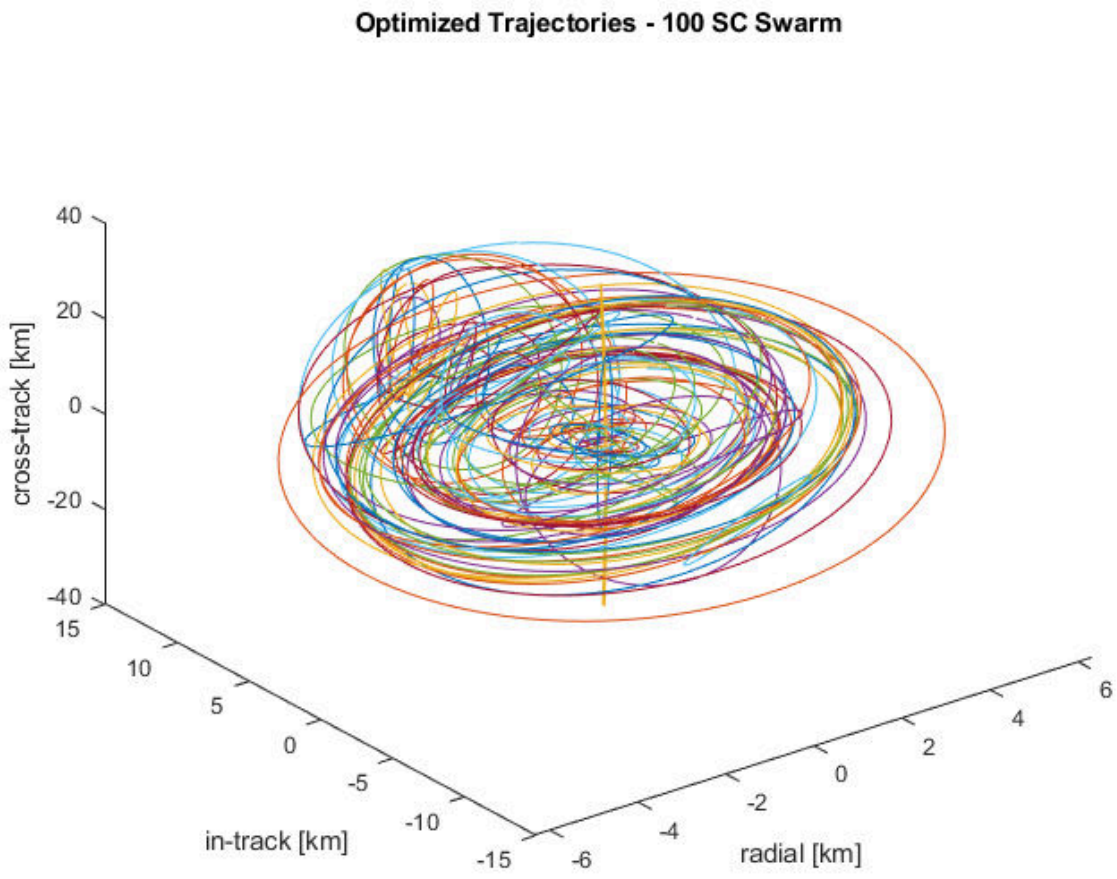


Figure 3.4: Swarm Solution for 100 Spacecraft – Computed Using Parallel Processing

3.2.4 Trajectory Modification for New Spacecraft Insertion

Now that a set of trajectories have been generated for the swarm, the next problem to tackle is the dynamic nature of the swarm: what to do when the number of spacecraft or their requirements changes?

The problem of adding or removing a spacecraft from a set of swarm trajectories that have already been generated is fundamentally different from the previous problem (see Section 3.2.2), since the trajectories cannot simply be regenerated for all spacecraft, as there already exists a set of spacecraft in their respective trajectories. When adding a spacecraft to the swarm, it

is understood that some or all of the other spacecraft in the swarm may have to modify their trajectories, thereby using some of their fuel reserves to enable a set of safe, mission-specific trajectories for the new swarm. However, it is desirable to do this in such a way that the ΔV used by the swarm as a whole is minimized, as well as the ΔV used by individual members of the swarm so as not to excessively deplete the reserves of a single spacecraft.

This is performed once again by using Genetic Algorithms, using the same nested GA scheme (see Fig. 3.1), but with a modification to the outer de-confliction GA to take into account the ΔV cost to attain a given trajectory from an existing one, and a modification to the spacecraft-level GA fitness function that uses the existing trajectory at the starting point for a solution rather than a random seed (see Eq. (3.2)).

$$F_{insert} = \left(1 + C_r \left\|\vec{r}(t_f) - \vec{r}(t_0)\right\| + C_v \left\|\vec{v}(t_f) - \vec{v}(t_0)\right\| + C_d \delta_{dist} + \boldsymbol{\Delta v}\right)^{-1} \tag{3.2}$$

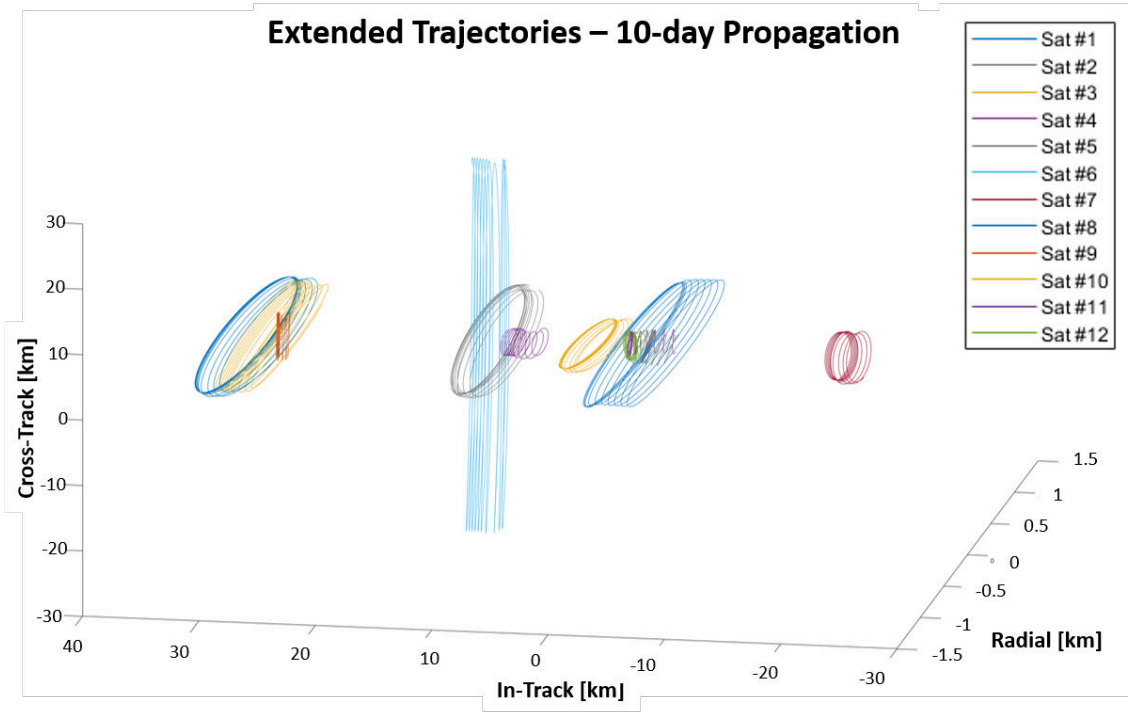


Figure 3.5: Modified swarm solution including the addition of an 11th and 12th spacecraft

An example of this can be seen in Fig. 3.5, which depicts a modification of the solution shown in Fig. 3.2 with an 11^{th} and 12^{th} spacecraft added into the swarm. The trajectories of the original 10 spacecraft have been modified slightly to allow for the two additional spacecraft, conserving Δv.

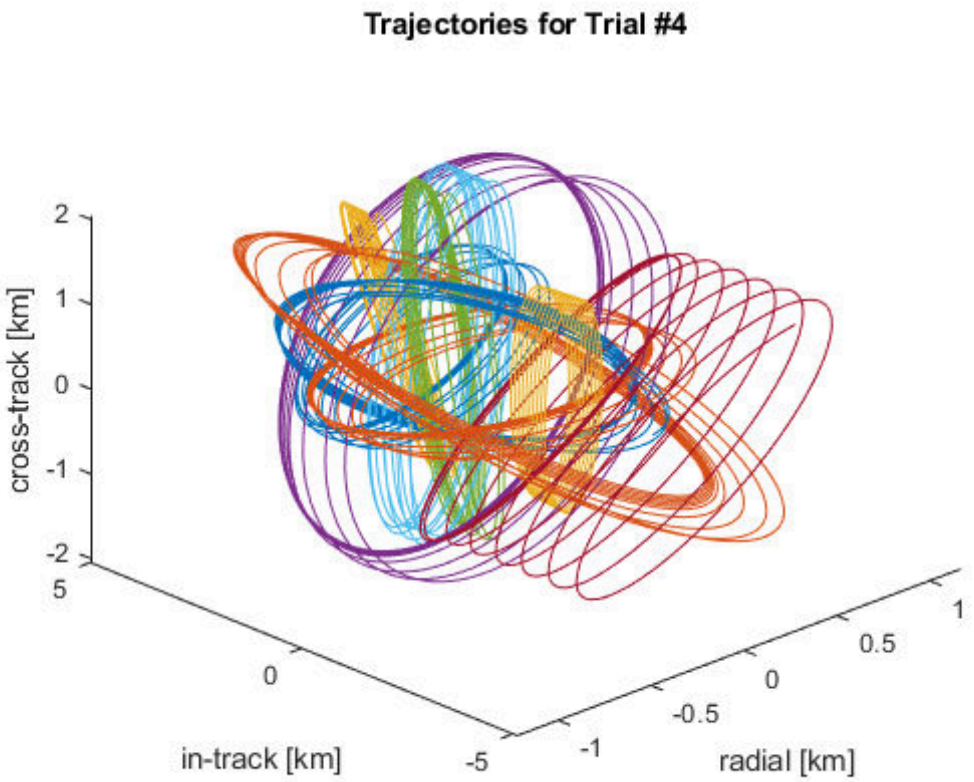
3.2.5 Considerations for Construction and Aggregation

When applying this methodology to in-space construction or aggregation of swarm members, consideration needs to be taken not only for the addition and removal of members from the swarm, but also for the dynamically changing dimensions, mass, and moment of inertia of the Client being constructed. As the structure grows, so will the keep-out zone specified for all spacecraft, especially if it is spinning.

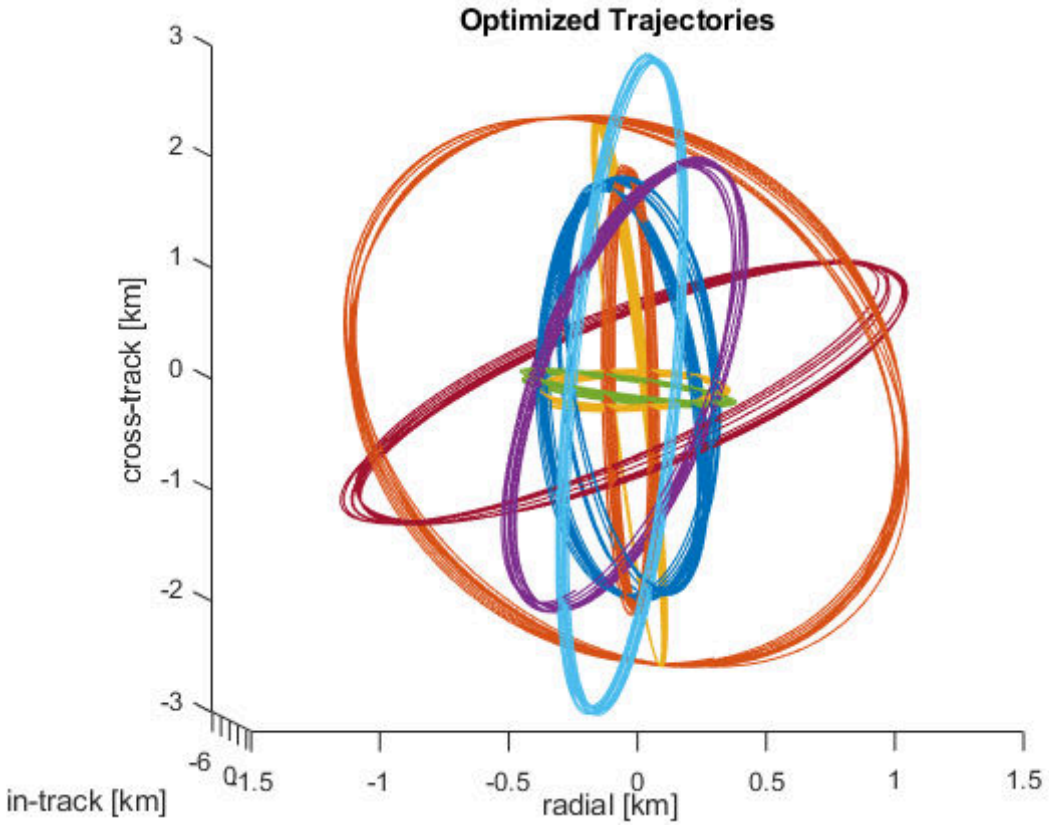
Strategies for how to deal with such dynamic situations will be detailed in Chapter 5 on the Behavioral Stresses of the System.

3.3 Comparing the GA Method to Arbitrary Trajectories

In order to determine whether this method of using GAs is safer and more efficient than any arbitrary trajectory choice, a set of randomly generated initial RPO states for closed trajectories were generated and propagated for a 50-spacecraft swarm until a collision was predicted. These spacecraft were confined to trajectories that kept them within 3 km of the target point. The same problem was then computed again, this time using the collision-avoidance optimization scheme outlined in this chapter.



(a) Trial #4 Trajectories (Random Start)



(b) Optimized Set Trajectories

Figure 3.6: Trial Trajectories for 10 of 50 spacecraft in swarm

Figure 3.6 shows an example of one of these propagated swarm sets, showing only the first 10 out of 50 spacecraft for both a single trial run, and the optimized case. This arbitrary trajectory generation and propagation was then performed for 100 different randomly determined initial states, and the times until collision were averaged to obtain the mean time until collision. This resulted in a mean time until collision of 58.5 minutes in the case of no collision avoidance optimization, with the distribution of collision times over the various states shown in Figure 3.7. For the spacecraft trajectories optimized using Genetic Algorithms, no collisions were predicted over a 10 day period. Table 3.2 lists the orbital elements of the target point used to setup this scenario.

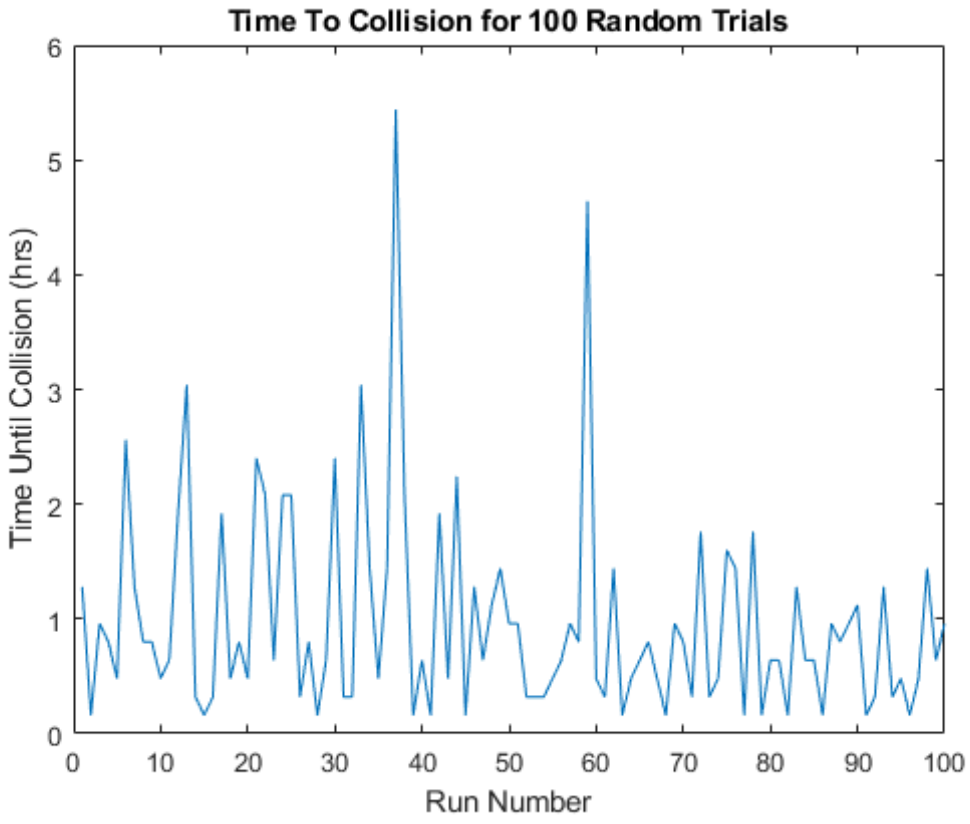


Figure 3.7: Time until collision for 50 spacecraft with random initial conditions, confined to a range of 3 km from target

Table 3.2: Orbital Elements of Reference Trajectory

Orbital Element	**Value**
Semi-major Axis	6978.14 km
Eccentricity	0
Inclination	0°
Right Ascension	0°
Arg. of Perigee	0°
True Anomaly	0°

This example shows the need for a method to perform collision avoidance verification on a large swarm of spacecraft, with the proposed method being the use of Genetic Algorithms to arrive quickly and efficiently to a solution. Note that these GA solutions, although optimized to prevent collisions with certain Δv and range constraints, are not a globally optimal solution. Rather, they are part of an infinite subset of solutions that are acceptable for the proposed mission constraints. There does exist an optimal solution within this subset, however this method does not claim to, nor is it designed to, solve for the globally optimal solution. The goal in this case is to solve for a solution which satisfies the constraints set out by the mission designer. Any further optimizations will be left to the reader, or to future research endeavors.

Chapter 4

Trajectory Maintenance for Spacecraft Swarms

4.1 Overview

Once a set of trajectories has been generated for a swarm of spacecraft, and the individual members inserted into their respective trajectories, this swarm configuration must then be maintained. This can be done using the proposed method for sensor fusion Kalman filtering, as described in Section 2.4. Given that there will be some level of error introduced to the system when the spacecraft are inserted into their trajectories, as well as error accumulated during relative motion and velocity measurements using on-board sensors, a Kalman filter is very useful to be able to reduce the overall error over time. This allows the solver to converge from the known position to the actual position, given enough time-varied sensor readings. This enables real-time collision avoidance, performing small stationkeeping maneuvers when necessary.

4.2 Kalman Filtering for Real-Time Operations

In order to maintain safe trajectories that avoid collisions between spacecraft in the swarm, a Sensor Fusion Unscented Kalman Filter is used to accurately determine, in real-time, the position and velocity of each spacecraft. This is done using only the information available to the swarm

members themselves, through external sensors, and without additional information from operators on the ground. Using the Kalman filter, a set of covariance matrices are obtained for each predicted position and velocity, setting the upper limits of the error bars on the measured and processed data.

A Sensor Fusion Kalman Filter (SFKF) is the extension of an unscented Kalman filter to incorporate the data from multiple sensors Section 2.4.2. This is implemented using multiple measurement update cycles to incorporate the shared data of the swarm to further refine the covariance ellipsoid for each spacecraft. Figure 4.1 shows an example of the sensor fusion process, where the covariance of the position between *Sat #1* and the *Client* spacecraft can be improved by fusing the data from all the swarm spacecraft, even taking into account the GPS position errors defining the locations of each swarm spacecraft with respect to *Sat #1*.

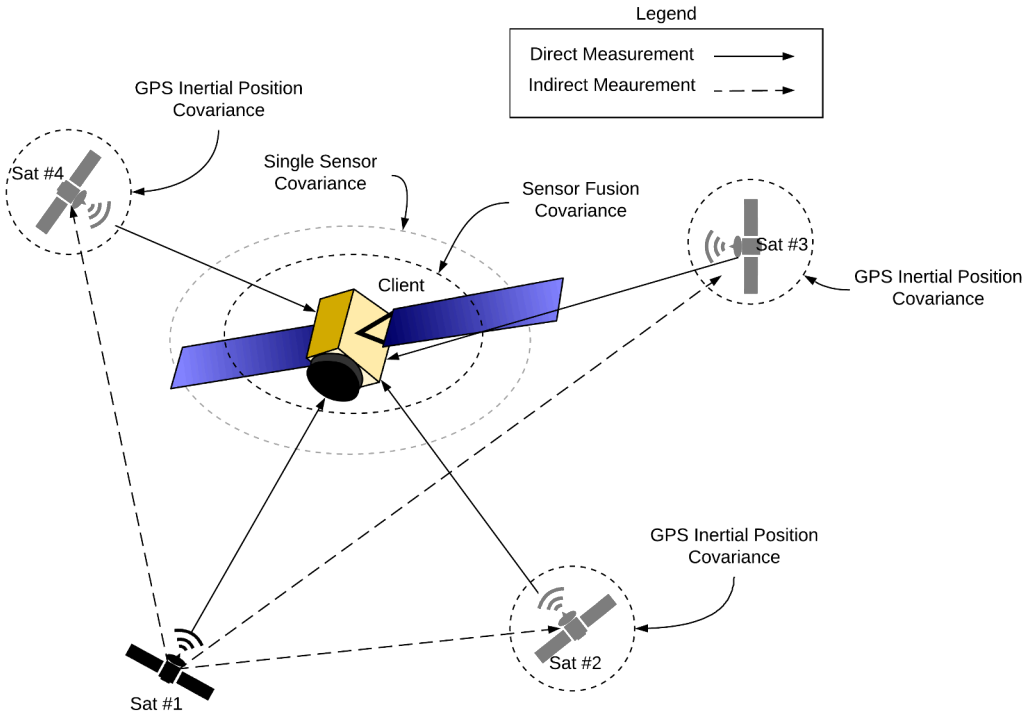


Figure 4.1: Sensor Fusion Diagram

Using Kalman filtering, trajectories can also be assigned a specific corridor, where if the spacecraft drifts too far from the designated trajectory (as determined by fusing sensor data and filtering it through the SFKF), a small correction maneuver is performed to re-align the spacecraft to its target.

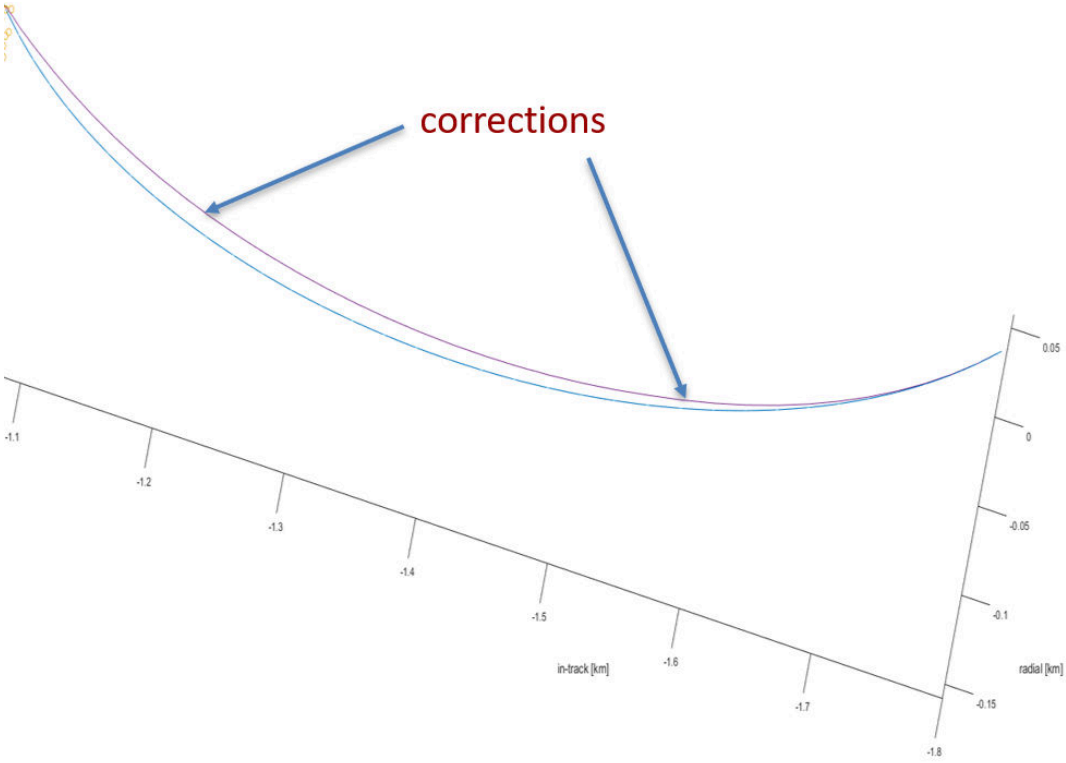


Figure 4.2: Filtered Rendezvous Maneuver

Figure 4.2 shows a two-stage rendezvous process, where the first step of the trajectory is a non-intersecting free-flight trajectory. This means that if the burn to transition from the first to the second segment were to fail, there would be no collision. The total Δv for this maneuver is 4.1 m/s, with 0.29 m/s of that due to small course corrections applied by the automated Kalman filter system to remain on-track. The blue trajectory is the projected course, and the purple trajectory is the path perceived by the spacecraft to be where it has actually travelled. This differs from the nominal trajectory due to injection errors, attitude knowledge errors on the spacecraft,

and uncertainty of true position. Two corrective maneuvers are performed during this transfer, dictated by the Kalman filter, to maintain the desired destination. In this case there is no *truth* trajectory of where the spacecraft truly was, as there is no independent observer to determine this. Instead, the swarm Kalman filtering method uses sensors aboard all spacecraft to compute the relative position of each swarm member, and to determine if any sensors are aberrant.

4.3 Patched RPO using Kalman Filtering

In order to transition between various relative motion trajectories, a method of patched RPO has been devised. This method takes two defined states, separated by a defined period of time, to determine the most efficient set of impulsive maneuvers to be able to safely transition between these states. This is done using two, three, or four impulsive maneuvers, depending on what method yields the lowest Δv consumption, while maintaining safe operations with nearby spacecraft.

Each impulsive maneuver begins a new trajectory segment. To compute the shape of the transfer trajectory, and the initial velocity vector required to place the spacecraft on that trajectory, a two stage solver is used, using the solution of the linearized C-W equation to seed the nonlinear solver for the perturbed gravitational field solution.

$$\vec{v}_0 = \boldsymbol{\Phi}_{\boldsymbol{rv}}^{-1} \left(\vec{r}_f - \boldsymbol{\Phi}_{\boldsymbol{rr}} \vec{r}_0 \right) \tag{4.1}$$

$$\vec{v}_f = \boldsymbol{\Phi}_{\boldsymbol{vr}} \vec{r}_0 + \boldsymbol{\Phi}_{\boldsymbol{vv}} \vec{v}_0 \tag{4.2}$$

Equations 4.1 & 4.2 describe the C-W equations used to solve for the initial and final velocities on a transfer arc, when the initial and final positions are known, as well as the transfer time. Φ_{rr}, Φ_{rv}, Φ_{vr}, and Φ_{vv} are defined in Equations 2.8, 2.9, 2.10, and 2.11, respectively. When solving with the C-W equations, the value for $\vec{v_0}$ is close to the desired solution, however it is a linearized

approximation of the unperturbed solution. The perturbed solution can be solved iteratively, numerically solving the second order ODE in Equation 4.3.

$$\frac{d^2\vec{r}}{dt^2} = -\mu\frac{\vec{r}}{\|\vec{r}\|^3} + \vec{a}_{grav-pert} + \vec{a}_{sun-moon} + \vec{a}_{SRP} \tag{4.3}$$

Where $\vec{a}_{grav-pert}$ is computed using Equation 2.21, $\vec{a}_{sun-moon}$ is computed using Equation 2.22, and $\vec{r}_{SRP}$ is computed using Equation 2.23. This system of second order ODEs is then solved iteratively to find the initial velocity $\vec{v}_{0_{pert}}$ such that the final position, $\vec{r}_f$, is the desired trajectory endpoint. This was done using the *fsolve* method in MATLAB, using the velocity $\vec{v}_0$ from the C-W linear solution as the initial solver guess to jumpstart the iterative process.

This is initially done prior to the start of the maneuver, in order to plan out the trajectory being travelled and prepare the spacecraft for the burn. Once the initial Δv is applied, the spacecraft has been injected onto the transfer trajectory. A Kalman filter is then used to compute the estimated true position of the spacecraft in real-time, using its onboard sensors. This estimated position is then used to recompute the target point at the end of the transfer arc, at a rate of once per second. If the trajectory is determined to deviate by more than 5 m, then Equation 4.3 is solved again iteratively, and a small impulsive maneuver is performed to align to the updated trajectory, maintaining the original target point as the destination. Figure 4.3 depicts this process graphically. This is done continuously in real-time during all swarm operations to maintain trajectories within their corridors.

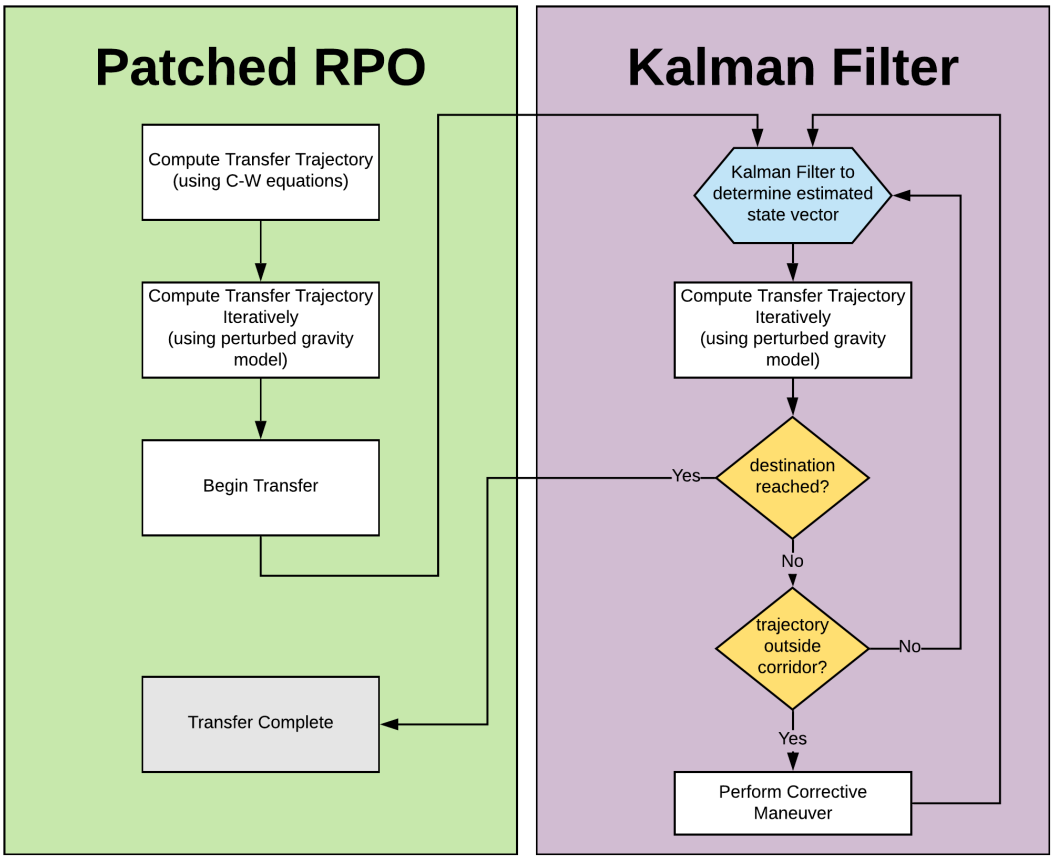


Figure 4.3: Patched RPO Process

4.4 Stationkeeping Maneuvers

Although the swarm trajectories are propagated forwards in time for 24hrs (if not longer) when generated to check for collisions, they will eventually begin to drift away from each other due to orbital perturbations. In order to repeat this trajectory, a two or three impulse trajectory change maneuver must be performed to either reset the swarm onto the same trajectories as the initial conditions, or to generate a new set of trajectories with minimal deviation from the current end state, while maintaining the requirements of the swarm. To *reset* the swarm trajectories, in essence plotting a relative motion trajectory from the end of the propagated swarm trajectory

back to its initial state vector, a two or three impulse trajectory change maneuver is used, as described in Section 4.3.

Figure 4.4 depicts a set of these return trajectories visually for a swarm of three spacecraft. The trajectories in grey are the previously travelled swarm trajectory generated using GAs, and the colored trajectories are the three return transfer arcs, using numerically computed two-impulse trajectories as depicted in Figure 4.3.

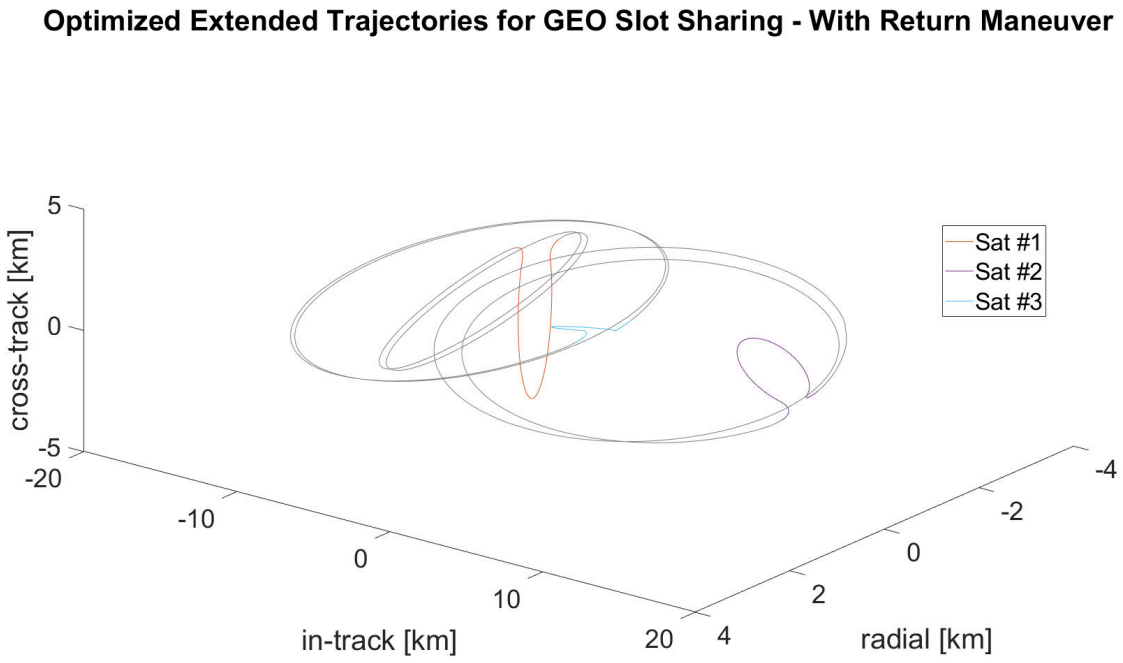


Figure 4.4: Return Trajectories

4.5 Considerations for Electric Propulsion

The previously mentioned trajectory change maneuvers are first modelled as instantaneous burns, using the method outlined in Section 4.3. However, many spacecraft in Geostationary orbit, as well as some smaller spacecraft in LEO, operate using Electric Propulsion (EP) thrusters rather than chemical (impulsive) thrusters. Once computed, the impulsive maneuvers are then modified into EP-compatible trajectories in order to be useful for GEO applications. This is done by iteratively

solving for a set of spline trajectories that bridges the two sides of the impulsive maneuver into a smooth trajectory that can be navigated within the limits of the EP thruster, while minimizing ΔV.

Although there are methods that can generate more optimized trajectories for EP thrusters [153–158], that is not the focus of this research, and thus it is left as an additional task for the reader, to be implemented in the future. The spline trajectory method used does include ΔV accommodations for gravitational and SRP perturbations on the spacecraft. The prime goal of this research is to prove that such a method of swarm RPO trajectory generation and maintenance is possible, and is operationally feasible, given that the upper bound on the EP trajectory estimation method is conservative compared to industry standards.

Chapter 5

Behavioral Stresses of the System

5.1 Overview

In order to fully understand the limits of the set of algorithms that form the spacecraft swarm framework, a closer look at the various edge cases of the system is required. This chapter will dive into a selection of these edge cases, and identify the results obtained from probing these cases in simulated trials. These include how the system deals with the unexpected loss of a vehicle that is part of the swarm, how the system responds to a central object which is increasing in size and mass over time, and how the system deals with internal collision warnings when the spacecraft begin to drift too far off their nominal trajectories.

Figures 5.1 & 5.2 depict graphically the covariance ellipses of the position of each spacecraft in their relative motion trajectories. Over time, these are constantly updated using relative ranging measurements between the spacecraft, fed into the Sensor Fusion Kalman Filter running aboard each vehicle. This enables each member of the swarm to keep tabs on its neighbors such that evasive maneuvers can be considered if the covariance ellipses will intersect – resulting in a non-zero probability of collision.

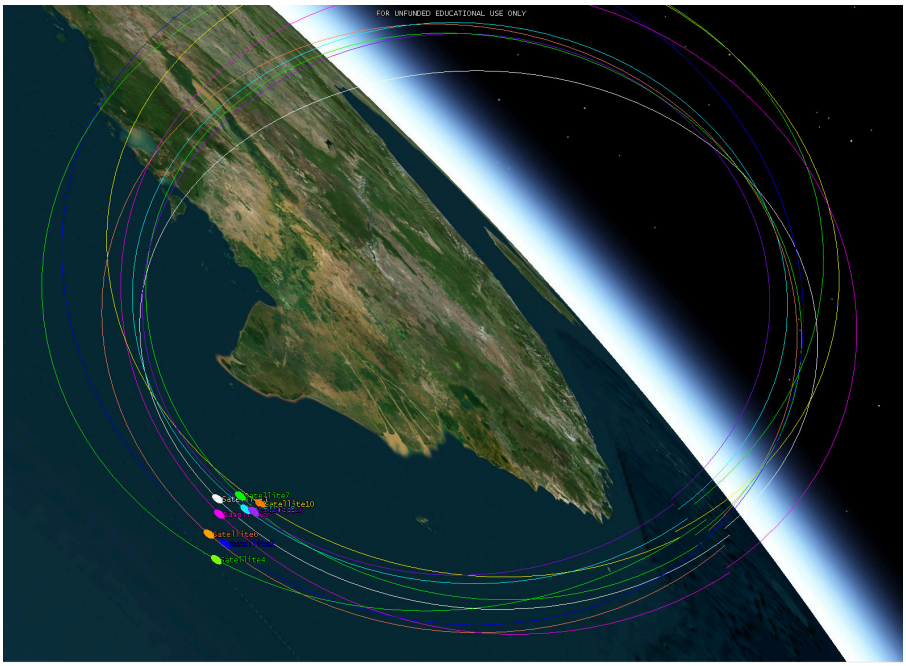


Figure 5.1: Swarm Trajectories with Covariance Ellipses

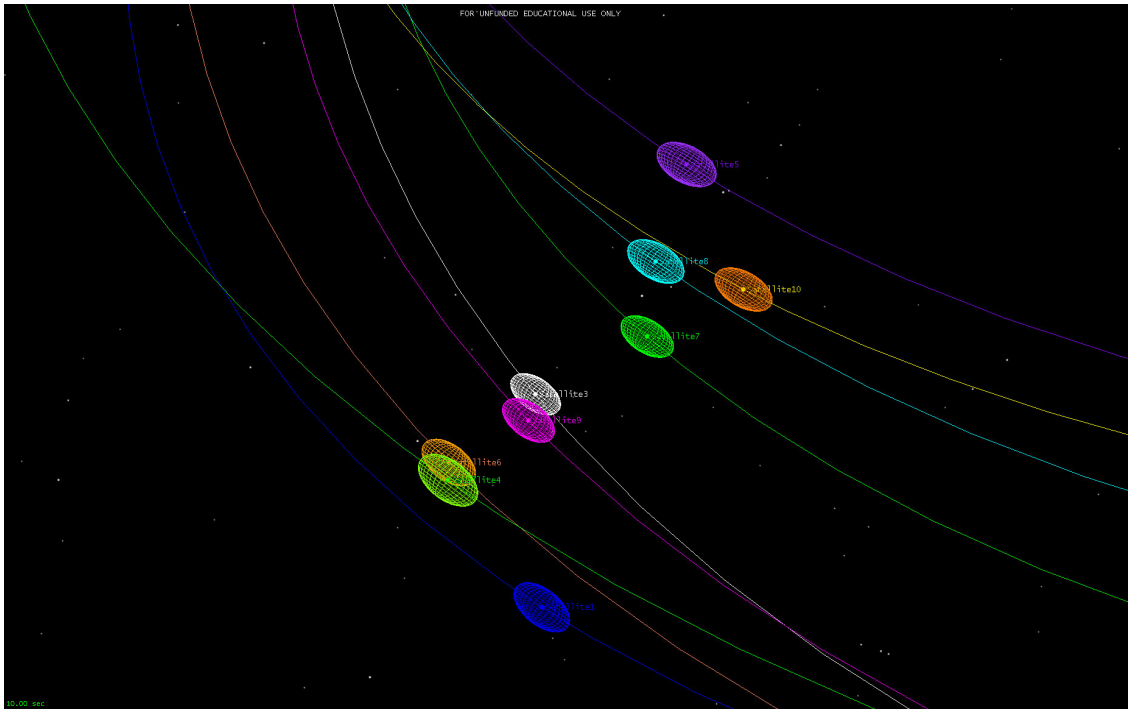


Figure 5.2: Swarm Trajectories with Covariance Ellipses (Zoomed)

Figure 5.3 depicts graphically the safe free-flight corridors assigned to each spacecraft in the swarm, where each spacecraft is free to drift within its corridor before any corrective maneuvers are taken, thus saving propellant by reducing the amount of corrective maneuvers required. This corridor is by default set to 50 m, with larger values requiring more computational time to de-conflict potential collisions between spacecraft.

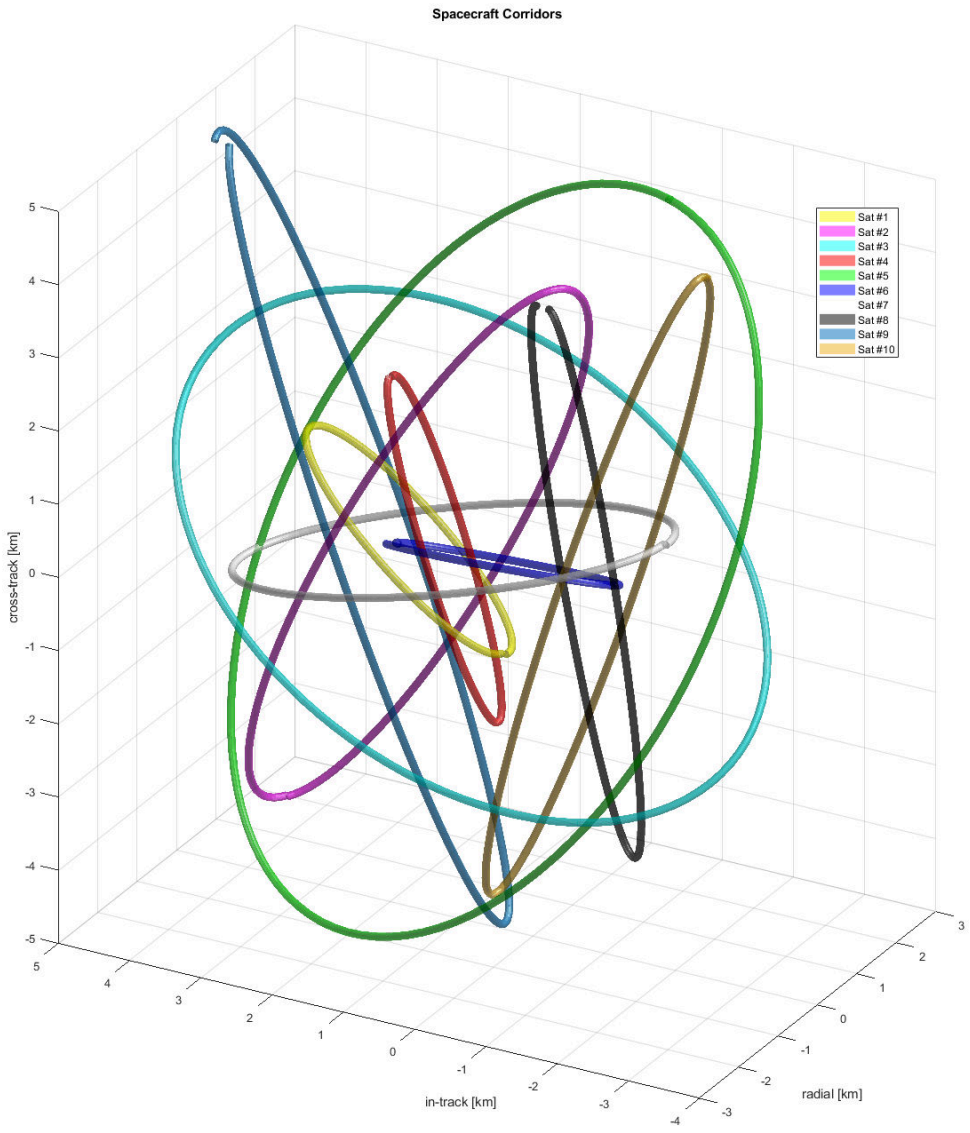


Figure 5.3: Swarm Trajectories with Free-Flight Corridors

Using these covariance ellipses and safe trajectory corridors to quantify a given probability of collision over the course of the mission, the genetic algorithm framework utilizes a variety of subroutines to deal with the scenarios in which a spacecraft deviates from its trajectory, becomes unresponsive, or is predicted to collide with another spacecraft in the swarm, or a foreign object (debris). The following sections will highlight these scenarios and the respective subroutines and methodology used to deal with these situations to preserve the functionality of the swarm.

5.2 Unexpected Loss of Vehicle

One of the edge cases considered for the swarm framework is that of an unexpected vehicle loss. If a member of the swarm were to go offline mid-mission, either entirely or from a communications standpoint, it would be considered a *zombie satellite*, for all intents and purposes a piece of debris that all spacecraft in the swarm must avoid. This avoidance is handled in the collision avoidance scheme in the genetic algorithm, where a safety corridor of 10 m around this trajectory is marked as a restricted zone, forcing the solver to generate trajectories that do not cross into this zone. By the nature of the swarm framework, the trajectories at the time of loss-of-control will not be able to intersect the trajectory of the zombie spacecraft, since the swarm trajectories are passively safe for at least 24hrs (and typically longer, depending on the scenario in play – see Section 3.2.2).

Note that even if the spacecraft is still functional in all but communications, the system is designed such that this spacecraft will fall back into a passive mode, not performing any maneuvers unless the onboard sensors predict a collision is imminent (see Section 1.5). This eases the burden on the remainder of the swarm to avoid the zombie spacecraft, as it will be on a known state from which a trajectory can be plotted deterministically.

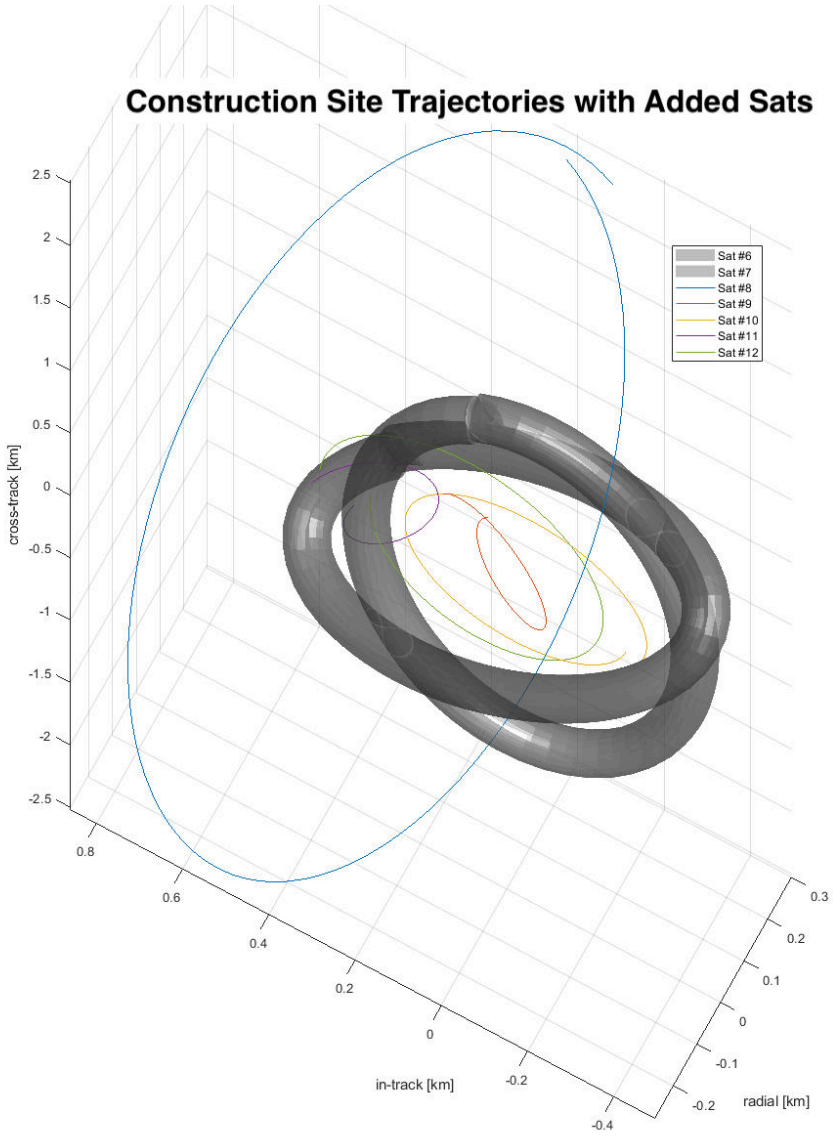


Figure 5.4: Swarm Trajectories with Zombie Spacecraft Keep-Out Zones

Figure 5.4 shows this graphically for a set of seven spacecraft, where two spacecraft are unresponsive and considered zombie satellites. These are marked in gray and shown as tubes rather than lines, to identify their restricted keep-out zones.

5.3 Response to Dynamic Construction Environment

Another edge case of the swarm framework to consider is that of a dynamic construction environment, in which a structure or body is being aggregated over the course of the mission. As this aggregation occurs, the object will grow in size and mass, with its rotational inertia properties changing significantly as well. This will result in a variable set of boundaries to which the swarm will have to adapt in order to avoid a collision with the aggregate body. To account for this, a method of phased boundaries around the aggregate body has been developed, enabling the swarm spacecraft to react to the changing size and shape of the object without requiring constant maneuvering.

The phased boundary method consists of setting a bounding volume, an ellipsoid, around the aggregated object and restricting this as a hazardous zone, where only the spacecraft actively engaging in close-quarters proximity operations with the object will enter. The phased portion of this strategy comes from the method used to increase the boundary as the object grows. At the point when the aggregated object exceeds the bounding volume, this volume is increased by 75%, and any spacecraft currently predicted to enter the bounding volume will be redirected onto new trajectories. This method ensures there will be no constant effort to react to the changing size and rotation of the aggregate object, and instead these reactionary efforts can be implemented in phases to conserve ΔV while also preventing conjunction risks. The value of 75% was not arbitrarily chosen but instead determined experimentally by running trials over various scenarios with various inflation factors to determine which one worked best. Figure 5.5 shows the ΔV per spacecraft vs the volume growth factor used, and it can be seen that the optimal values are approximately 1.1 or 1.75. To choose the appropriate factor between the two, it is useful to also consider the overhead logistical cost associated with the swarm growth maneuvers. 1.75 is used for the remainder of the simulations, as it requires less logistical overhead than to move the

swarm out by a factor of 1.2 or less at each growth point, which would result in near-constant reconfigurations of the swarm.

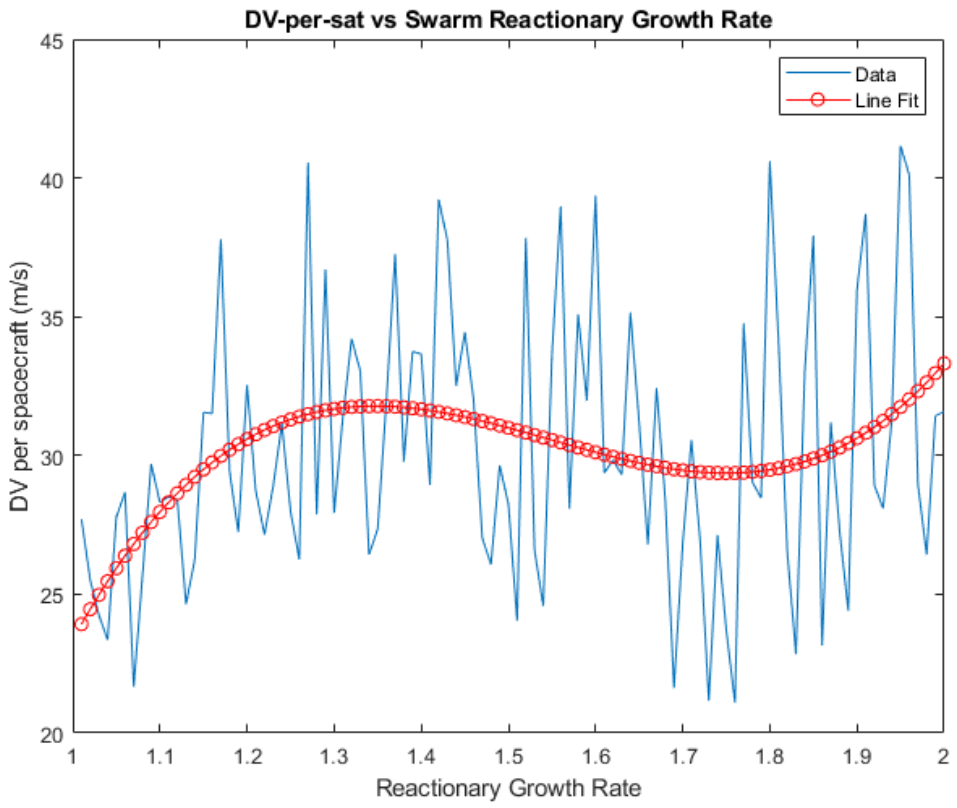
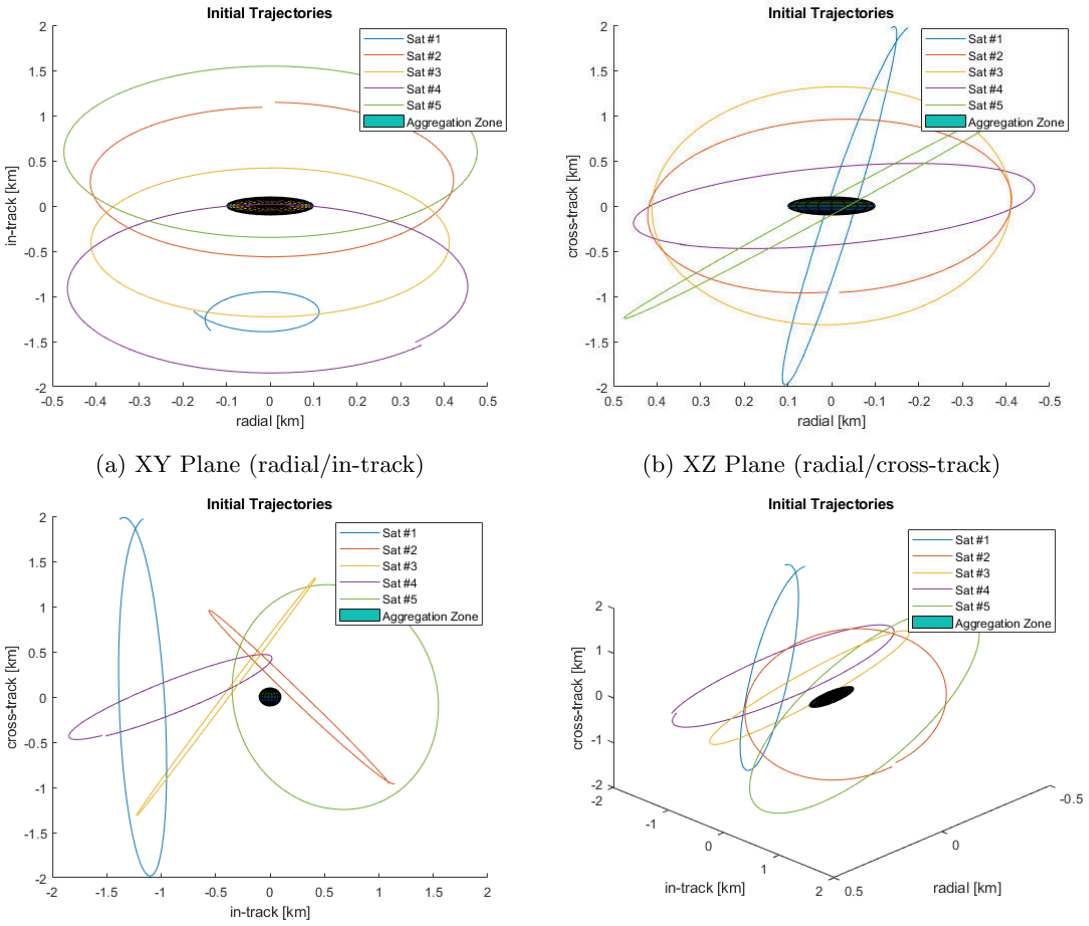


Figure 5.5: ΔV vs Swarm Growth Factor

Using a swarm volume growth factor of 1.75 (75% growth when the aggregate object exceeds its boundaries), a set of simulations were run to determine what would happen to a swarm of spacecraft in close proximity to this aggregate object. As pieces are aggregated to the object, it will grow in size. When this size exceeds the defined safe aggregation zone, then it is no longer safe for the swarm to maintain its current trajectories. At this point, the swarm must *grow* to move to a shell further away from the aggregate object. This is done by applying a growth factor ($f_{growth} = 1.75$) to the bounding limits that are fed to the GA solver's fitness function. Using this,

new trajectories are then computed, as well as transfer trajectories to patch the initial trajectories to the new trajectories.



(a) XY Plane (radial/in-track)

(b) XZ Plane (radial/cross-track)

(c) YZ Plane (in-track/cross-track)

(d) Isometric View

Figure 5.6: Initial Trajectories for Structural Aggregation

Figure 5.6 above shows the initial trajectories generated for a 5-spacecraft swarm designed to aggregate a structure in orbit. When the swarm arrives, this structure has a radius of 10 m and a keep-out zone of 100 m. This keep-out zone, or aggregation zone, is depicted by the solid surface in Figure 5.6 and the following sets of figures in this section. Figure 5.7 shows a perspective view of the same trajectories, where each axis is on the same scale. The keep-out zone is the sphere in the center of the plot, and all the spacecraft can be seen to be avoiding this sphere.

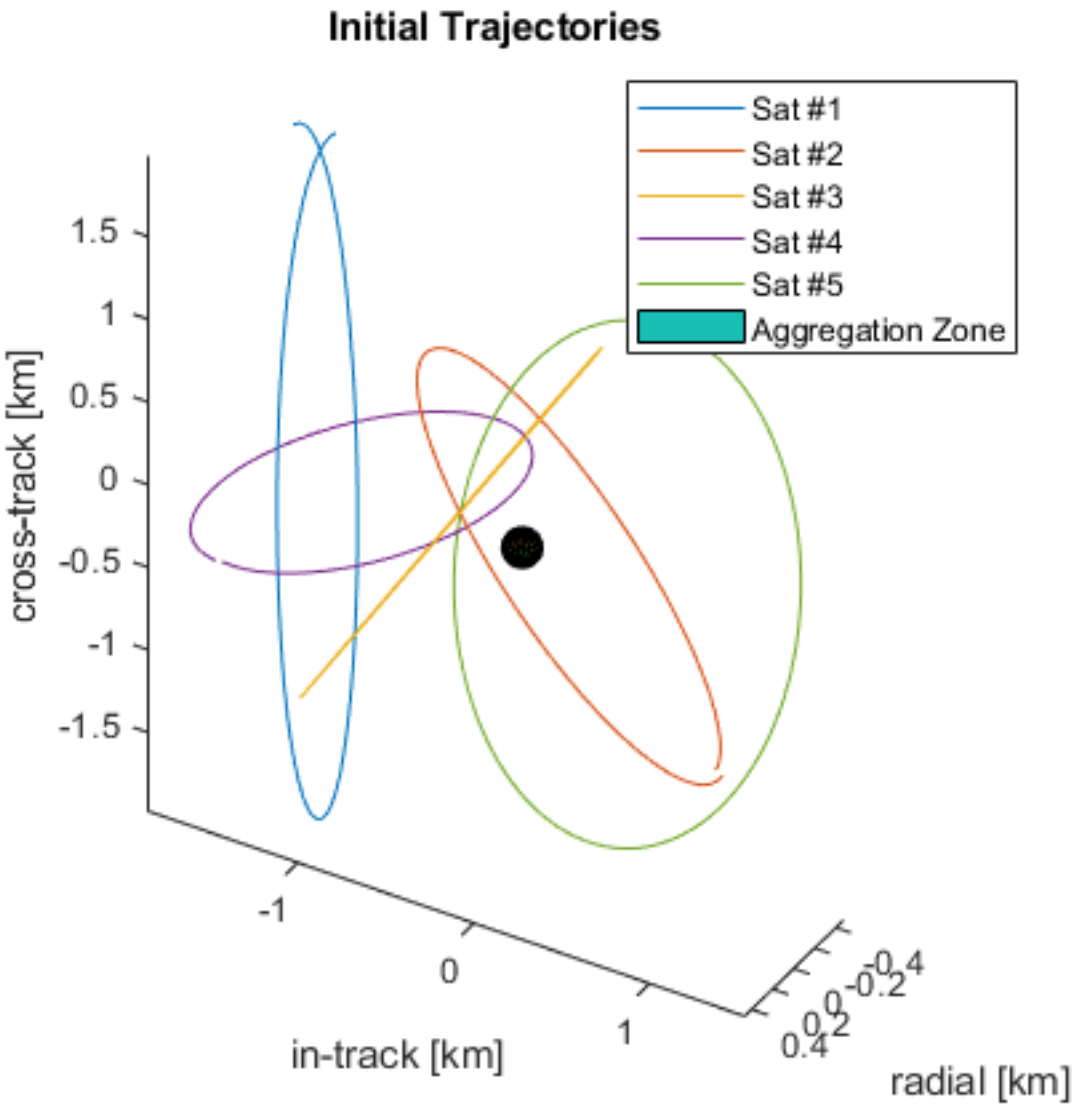
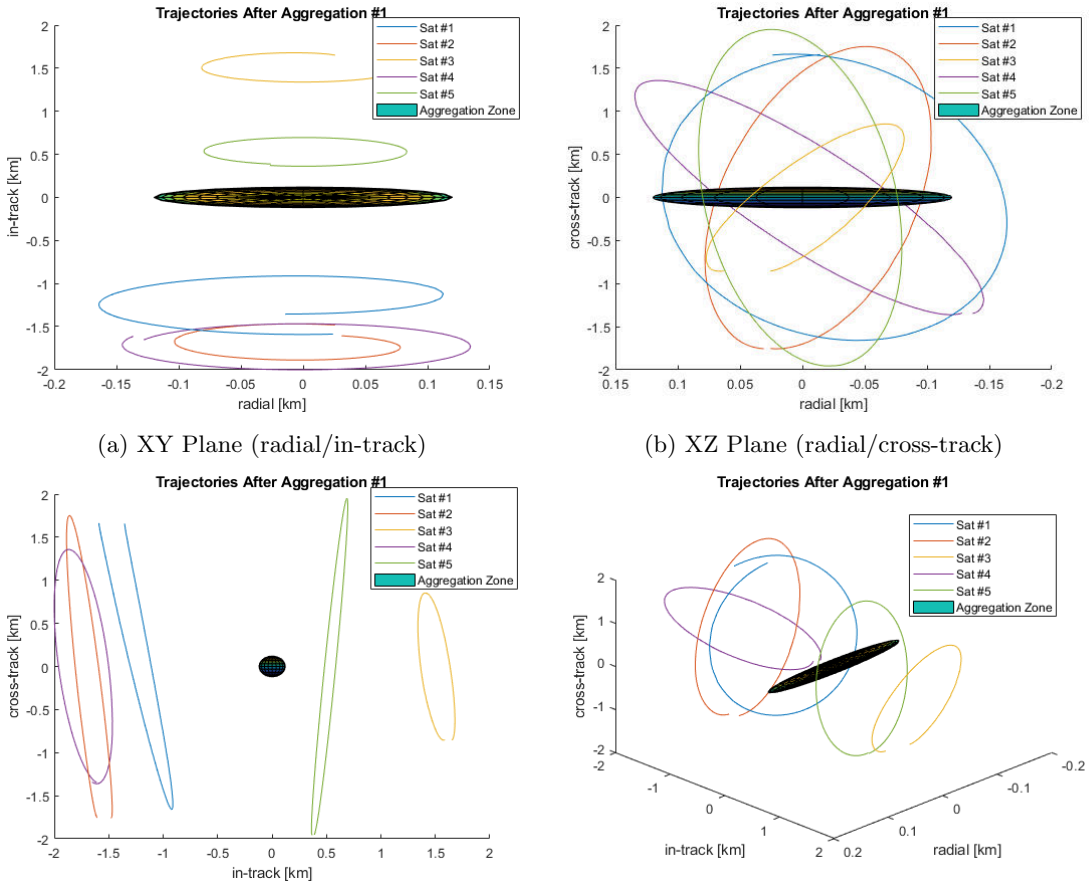


Figure 5.7: Initial Trajectories for Structural Aggregation (Perspective View)

In the time between trajectory reconfigurations, the spacecraft in the swarm are transiting between the swarm trajectories and the aggregation zone, bringing raw materials back and forth for the construction and aggregation. Doing so uses fuel, which is why it is not desirable to keep the swarm far away from the aggregate body. However, it is also undesirable to keep the swarm too close to the aggregate body, for fear of increased collision risk.

Once the aggregate body grows past the keep-out zone, a set of trajectory reconfiguration algorithms are run to determine a new set of trajectories that fit in a new shell. In the case of these simulations, a 2 km shell is used at each step. After the first growth phase, the aggregate body is 100 m in radius, and thus the keep-out zone is extended to be 175 m. This is seen in Figures 5.8 & 5.9.



(a) XY Plane (radial/in-track)

(b) XZ Plane (radial/cross-track)

(c) YZ Plane (in-track/cross-track)

(d) Isometric View

Figure 5.8: Trajectories for Structural Aggregation After First Growth Phase

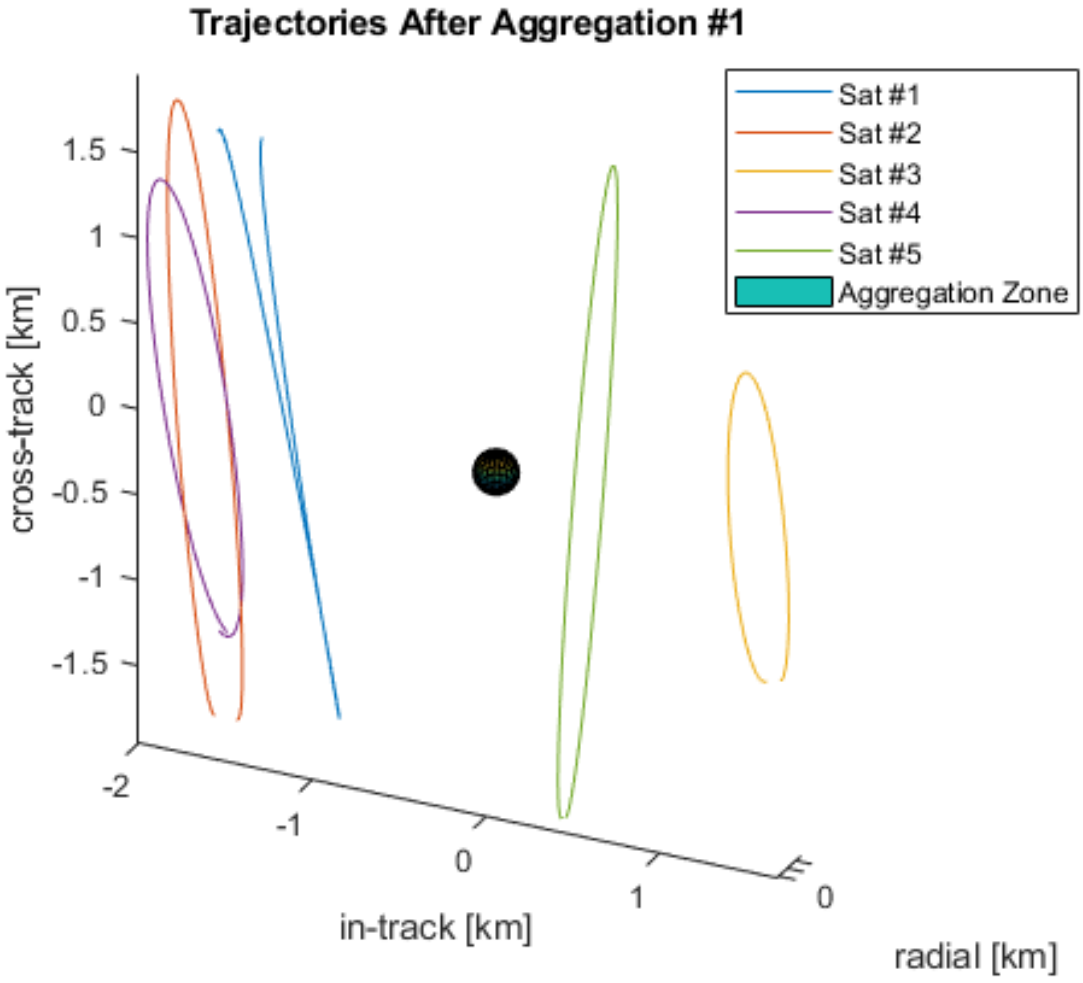
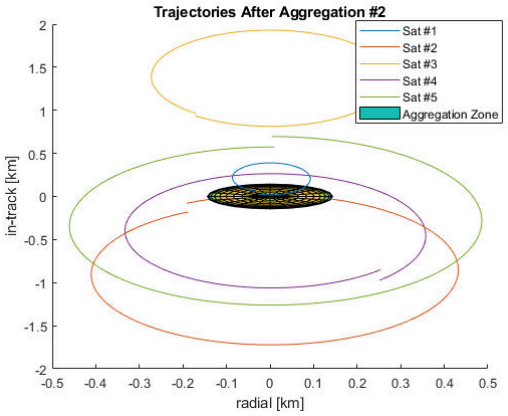
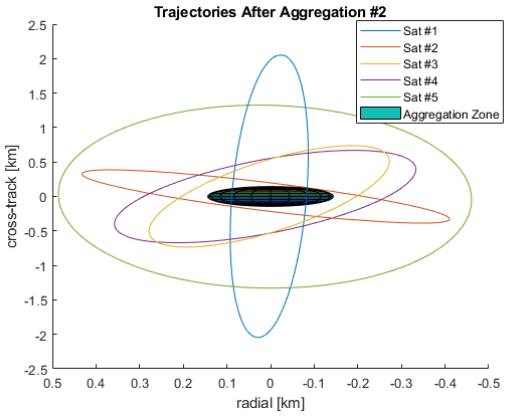


Figure 5.9: Trajectories for Structural Aggregation After First Growth Phase (Perspective View)

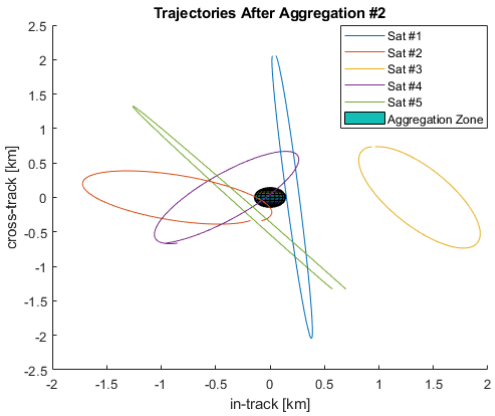
And finally, Figures 5.10 & 5.11 show the swarm after the second growth phase. At this point the aggregate object is 175 m in radius, and the keep-out zone is extended to 306 m.



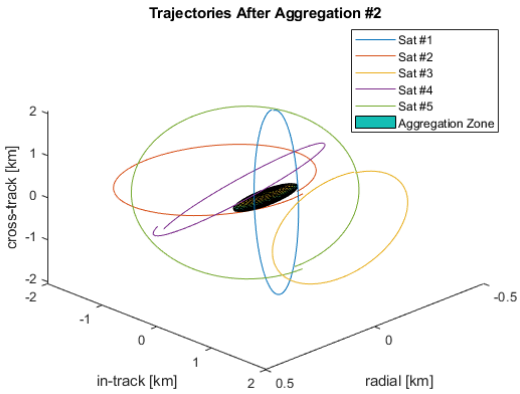
(a) XY Plane (radial/in-track)



(b) XZ Plane (radial/cross-track)



(c) YZ Plane (in-track/cross-track)



(d) Isometric View

Figure 5.10: Trajectories for Structural Aggregation After Second Growth Phase

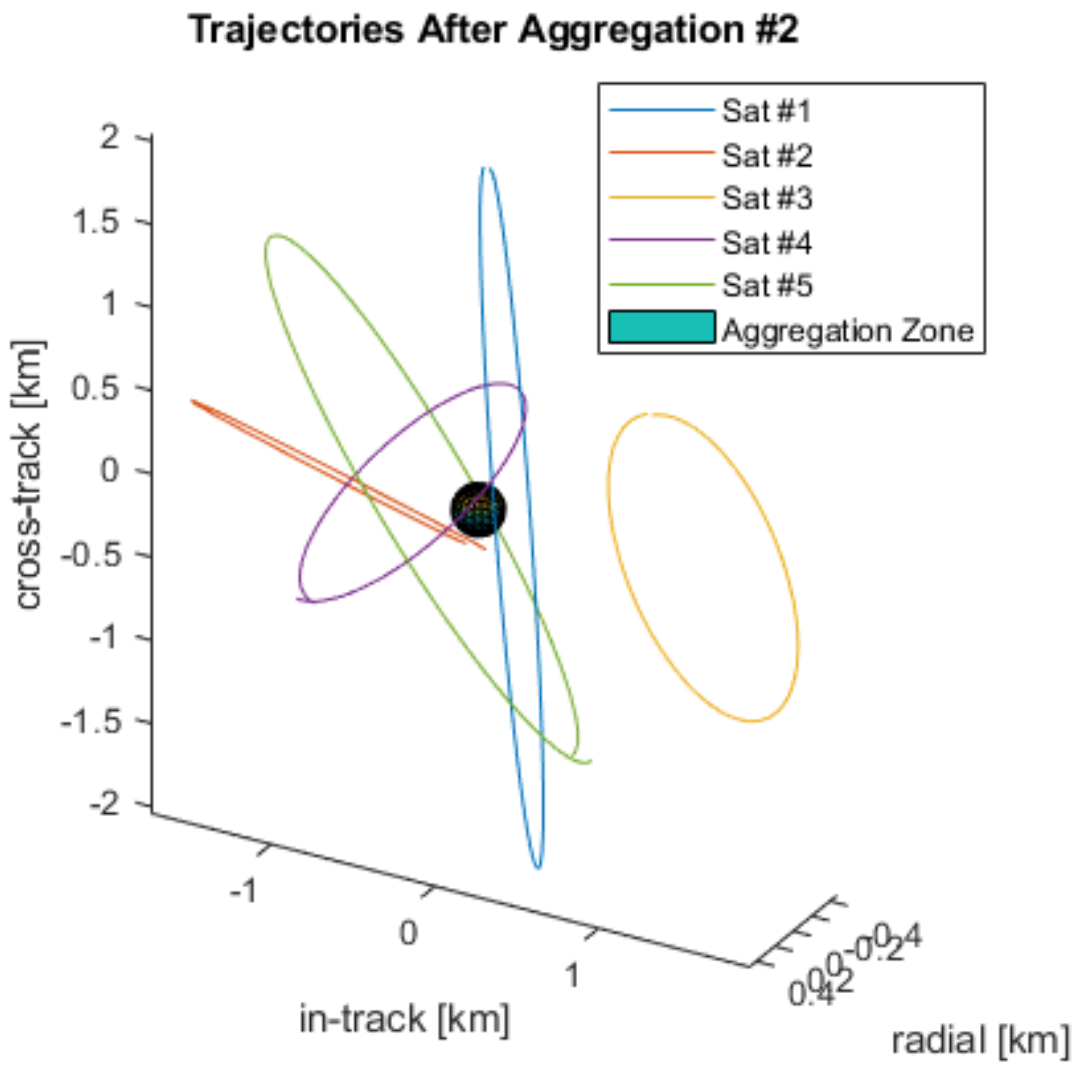


Figure 5.11: Trajectories for Structural Aggregation After Second Growth Phase (Perspective View)

It should be noted that there is an intuitive reason for why the growth factor of 1.2 appears to be more desirable than larger values, and that is because if larger values are used, the swarm will be quite far away from the aggregation zone for a considerable amount of time. This means that when any piece needs to be aggregated to the structure, a spacecraft from the swarm needs to perform a trajectory change maneuver to rendezvous closer in, and then come back out to the swarm to pick up a new piece. If the swarm is far away from the aggregation zone, this will require

a significant amount of fuel. Conversely, if the swarm is too close to the aggregation zone, then the swarm will need to move each time the aggregate object grows by a few meters, requiring large amounts of fuel for each transfer, thus shortening the mission lifetime. The 20% growth factor is somewhat of a *sweet spot* which falls between the two extreme cases.

5.4 Collision Avoidance Schemes

The Sensor Fusion Kalman Filter is a great tool for updating the state vectors of each spacecraft in the swarm, and thus a good to tool to predict collisions between spacecraft. However, predicting the collision is only the first step to rectifying the problem in the swarm. In the case of a projected collision between two (or more) spacecraft in the swarm, the trajectory maintenance algorithms use a hierarchical set of collision avoidance schemes in order to prevent a catastrophic collision between the spacecraft. The use of a hierarchical system enables multiple different methods to be applied to address the impending collision, thus broadening the applicable scope of the scheme. This hierarchical system is built upon existing collision avoidance practices and expanding on them to extend them to swarm applications [31, 159–184].

Firstly, the simplest method of collision avoidance is applied: alternate trajectories are computed for each spacecraft involved, slightly increasing the offset from the current trajectories until a set of trajectories is found that no longer result in a predicted collision within the specified mission period. The required ΔV is also computed to transition to these newly generated trajectories, and if there are multiple solutions found then the one with the least ΔV is used. If the total ΔV is larger than a specified threshold then this method of collision avoidance is not feasible, and the trajectory maintenance algorithm moves on to the next method in the hierarchy. In simulated scenarios, this threshold was set to 20% of the mission ΔV budget, but this will vary between scenarios and missions.

The second method of collision avoidance is similar to the first, however it is applied not only to the spacecraft directly involved in the collision prediction, but also their directly adjacent neighbors, solving for a new set of trajectories for these spacecraft that avoid collisions over the specified mission period (10 days for most simulations discussed in this thesis), and also remain within the 20% ΔV capacity threshold.

The third method of collision avoidance is a more resource intensive method, and typically will work better with continuous thrust propulsion systems, rather than chemical propulsion systems, as it employs a non-keplerian trajectory for a short period of time in order to avoid a collision. This is similar to the Bouncing Ball method by Kim, Mesbahi, and Hadaegh [159], modified to be used in a gravity well rather than interplanetary space. This method uses a maneuver which plots a spline trajectory joining two safe points on either side of the predicted zone of collision, such that if this trajectory is traversed, no collision will occur. However, this is not a free-flight trajectory, and as such navigating it will use a significant amount of propellant. This is much more easily done with continuous thrust electric propulsion methods than a chemical propulsion method, which would require numerous burns with attitude adjustments in between.

The fourth method of collision avoidance is more of a stopgap measure, where if neither the first, the second, nor the third methods yielded a viable solution, a solution will then be obtained for which no collision is predicted for a shorter time period (24 hrs in the case of the simulations performed). This is done using a method similar to the first method. Since this will not cover the entire mission period, this is only a stopgap method, as there is still the possibility of a collision in the future. While the previous two methods are designed to operate autonomously, this method buys time for ground controllers to analyze the situation and come up with a unique solution tailored to the specific scenario at hand, something that an automated system is ill suited to do. If no solution can be found, then the hierarchical avoidance system will flow into the fifth and final method.

Finally, the last resort method of collision avoidance is to eject one (or more) of the affected spacecraft from the swarm itself, moving them to stable trajectories 10 km - 20 km outside the swarm so that they no longer pose a threat to the rest of the swarm, while the situation is reassessed. This is a last resort maneuver, used only when no previous method yields a viable solution, as this will likely result in a temporary loss of mission resources, unless spare spacecraft are present in the swarm.

Chapter 6

Swarm Configuration Example Scenario

This chapter will go through an example swarm configuration from start to finish for an on-orbit construction project. This includes the definition of the swarm, the results from the trajectory generation process, a modification to the swarm to introduce new spacecraft to increase its capabilities, and the loss of a spacecraft to an unsolvable error, resulting in a piece of debris in the vicinity of the remainder of the swarm. It also includes the computation of the stationkeeping maneuvers used to recycle this set of trajectories once the orbital drift becomes too large, and a summary of the Δv usage for each operation. Kalman filtering is used during all propagation simulations between each step to verify safe operation of the swarm, even under random error conditions.

The example scenario in use for this test is the robotic assembly of an interplanetary transport ship, launched from Earth in pieces sized to the limitations of a rocket fairing, to be assembled in LEO. The data in the example is derived from the fictional ship *Hermes* from the film *The Martian*, and its source material of the same name, a Nuclear Electric Propulsion (NEP) powered spacecraft [185, 186].

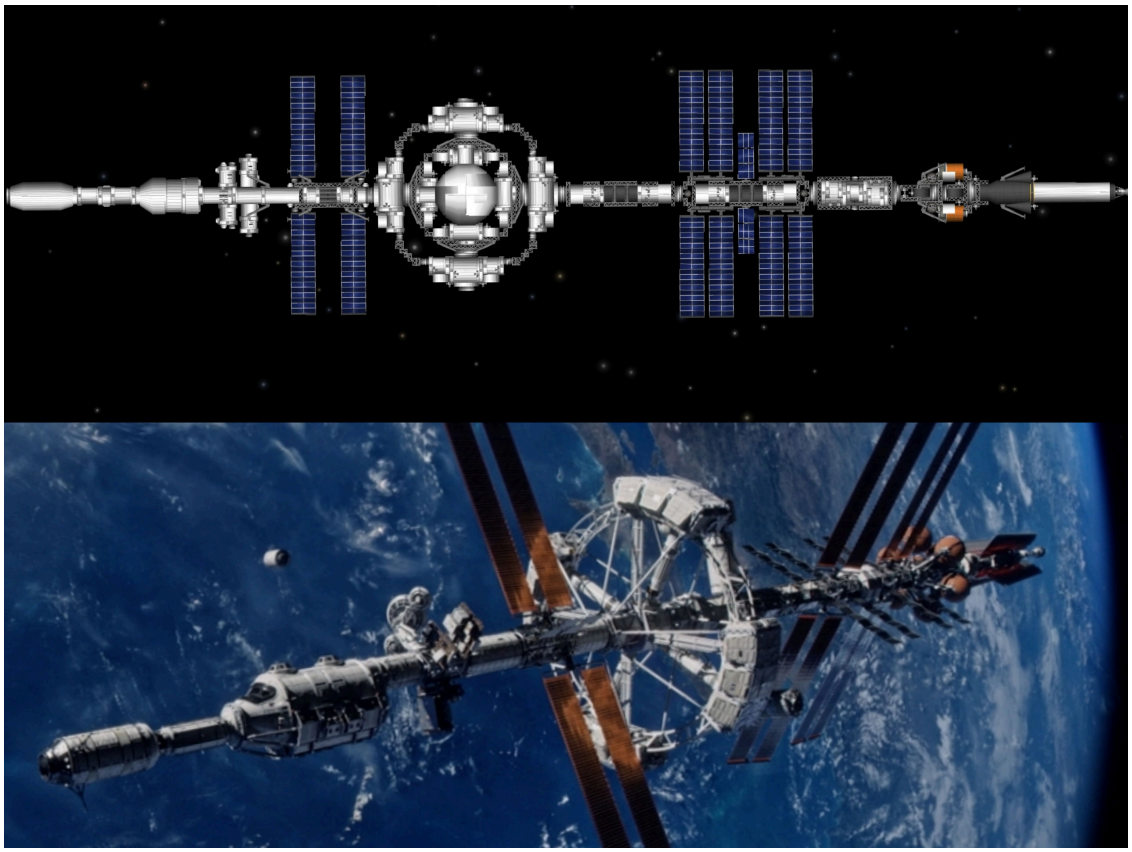


Figure 6.1: Hermes Spacecraft from *The Martian* [185, 187]

6.1 Initial Conditions

The NEP ship is separated into 5 segments, with each one launched separately into nearby orbits, waiting to be assembled. Each segment is in a circular orbit at the same altitude (600 km), with a 1 km in-track separation between each, as depicted in Figure 6.2. The swarm of 10 spacecraft will then rendezvous with each of these segments and transport them, in order, to the assembly site. This site is the origin of the relative motion LVLH coordinate system in use for this scenario.

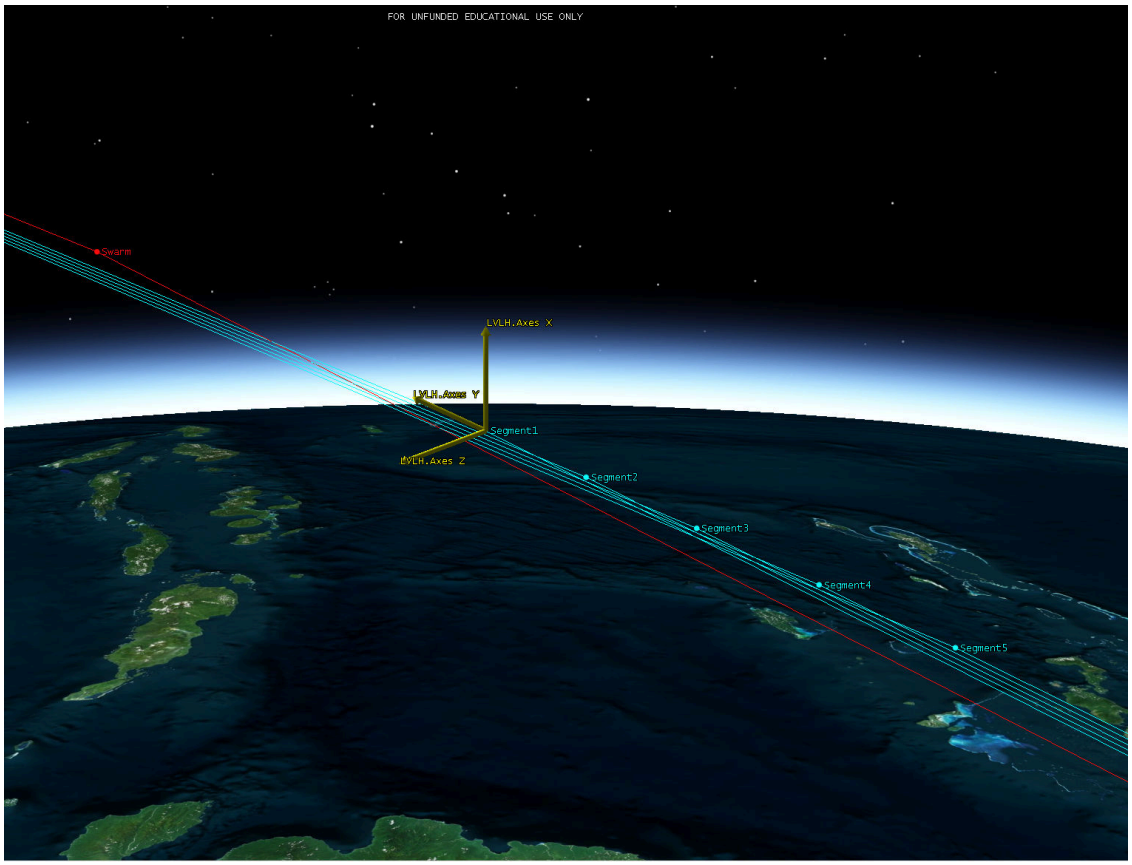
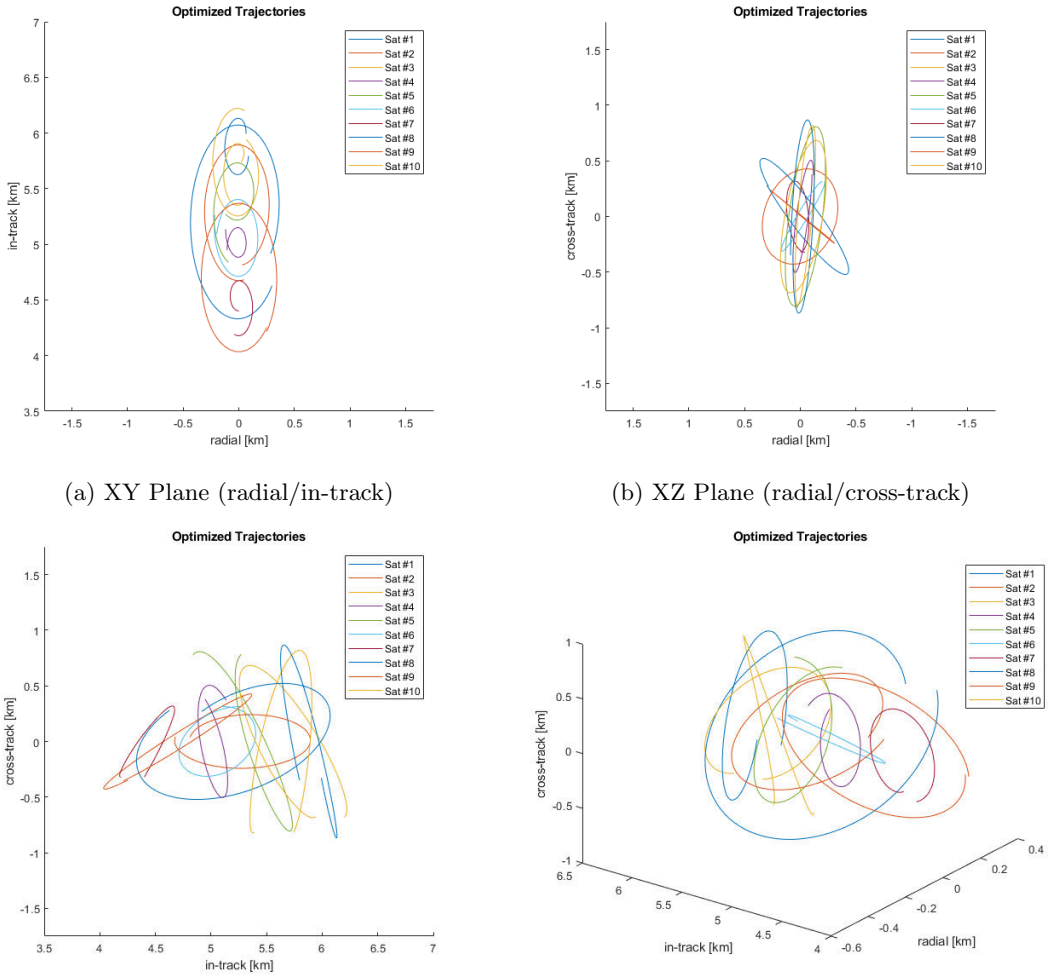


Figure 6.2: Initial Orbits of Hermes Segments

6.2 Results

6.2.1 Trajectory Generation

Figure 6.3 below shows the initial set of trajectories generated for the spacecraft swarm, located at a distance of 5 km ahead of the construction zone, in the in-track direction. The relative position to the construction site can be seen also in Figure 6.2 above. These trajectories were computed in the manner described in Section 3.2.2, with the swarm restricted to a box 1 km x 4 km x 2 km in size. These form the primary trajectories of the swarm, where the spacecraft will wait in a holding pattern until it is time to acquire the *Hermes* segments and commence the aggregation process.



(a) XY Plane (radial/in-track)

(b) XZ Plane (radial/cross-track)

(c) YZ Plane (in-track/cross-track)

(d) Perspective View

Figure 6.3: Initial Spacecraft Trajectories

6.2.2 Transfer Trajectories to Acquire Spacecraft Sections

Following the insertion of the swarm into its initial trajectories, and the observation of the *Hermes* segments, the next phase of the mission is to transfer five of the swarm spacecraft to acquire the segments and rendezvous them in the construction zone, while the remaining five spacecraft setup for support and reconnaissance operations in the construction zones. Transfer trajectories are computed using a minimum two-segment, three-impulse maneuver method that enables passive safety for the most dangerous part of the transit. This functions by firstly designating a keep-out

zone around the target object, as seen in Figure 6.4. In this example scenario, this keep-out distance is 50 m. The first segment of the transfer arc then seeks to connect, using the minimal amount of ΔV, the initial point and a holding point 100 m from the target, while also constraining the transfer trajectory to not enter the keep-out zone, even under a free-flight extension.

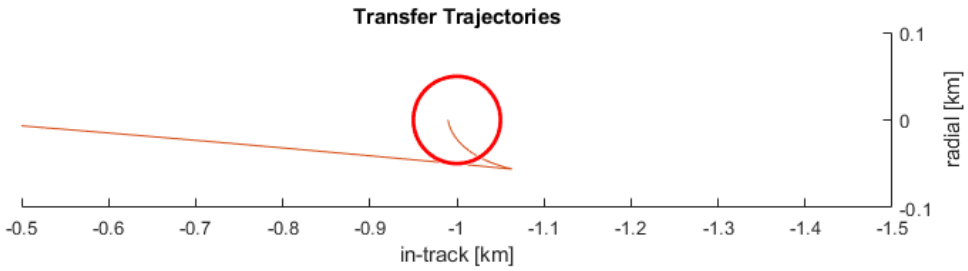


Figure 6.4: Three-Impulse Transfer with Keep-Out Zone

The second segment then connects the trajectory, from this hold point, to a secondary hold point 5 m from the target spacecraft. Figures 6.5 & 6.6 show these transfer trajectories for the five swarm spacecraft en-route to rendezvous with the five objects to be aggregated. The analysis leaves off at this point, as the final rendezvous sequence from 5 m to 0 m is highly specific to the spacecraft being used, as well as the type of docking system in use, whether it be cooperative, non-cooperative, robotic, electroadhesive, mechanical, etc [188–194]. Rather, the goal of this analysis is to find a way to safely and efficiently position and re-position a group of spacecraft

into such a close-range RPO location where they can perform their tasks, largely unencumbered by the complexities of orbital mechanics, as perturbations and non-inertial rotational effects are insignificant at such distances and timescales [195, 196]. This is performed for spacecraft #1 through #5 to gather the objects to be aggregated. This process uses 4.6 m/s of ΔV.

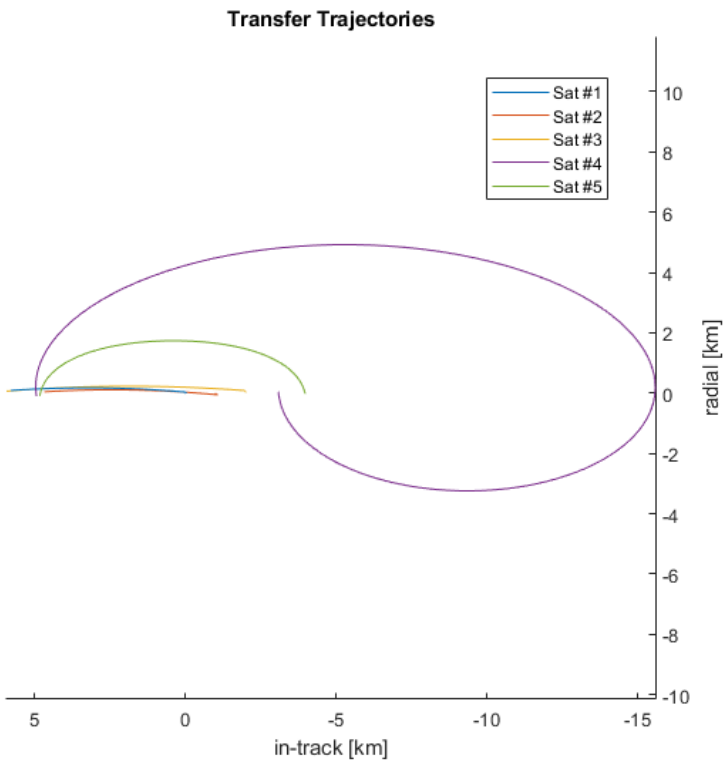


Figure 6.5: Transfer Trajectories from Parking Orbits to Segment Rendezvous

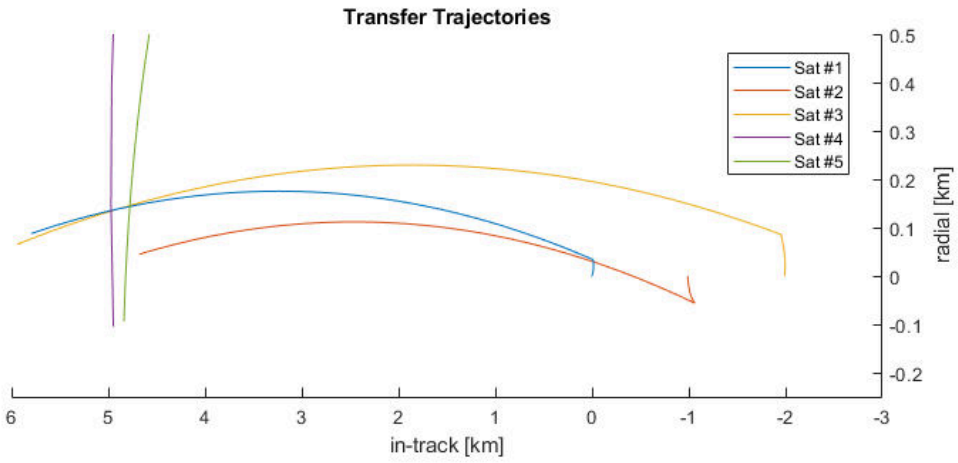


Figure 6.6: Transfer Trajectories – Close Up of Sats 1-3

Figure 6.6 shows a close view of three of these transfer trajectories, highlighting the two-stage maneuver visually. It can be seen that the free flight trajectories from the first of the two transfer trajectories for each spacecraft will not collide with the target object even if the stopping burn does not occur. Rather, it will pass by at a safe distance, even when accounting for insertion errors, ensuring safety of the cargo in a swarm failure event.

6.2.3 Transfer Trajectories to Rendezvous in Assembly Zone

Following the rendezvous with the spacecraft segments, the next step is to collect these and return them to the construction site for assembly, which is (0,0,0) in the coordinate frame shown in Figure 6.5. This process is computed in a similar manner to the transfer in section 6.2.2. The resultant trajectories can be seen in Figures 6.7 & 6.8, using 8.38 m/s of ΔV.

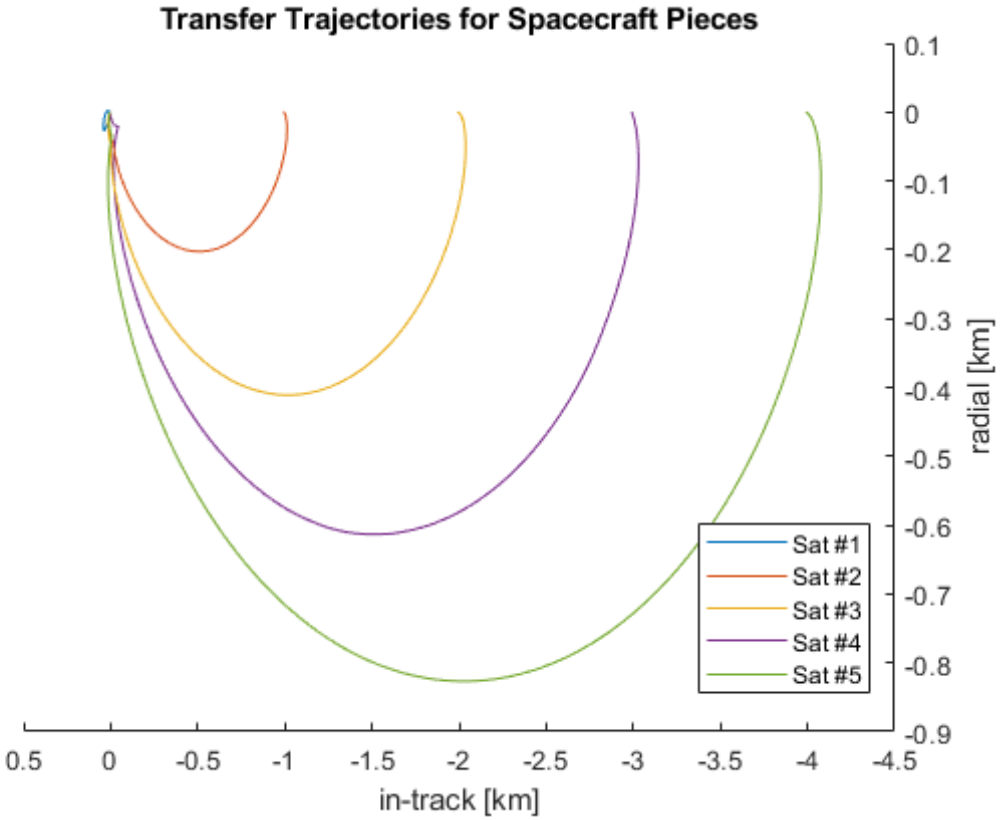


Figure 6.7: Transfer Trajectories to Construction Zone

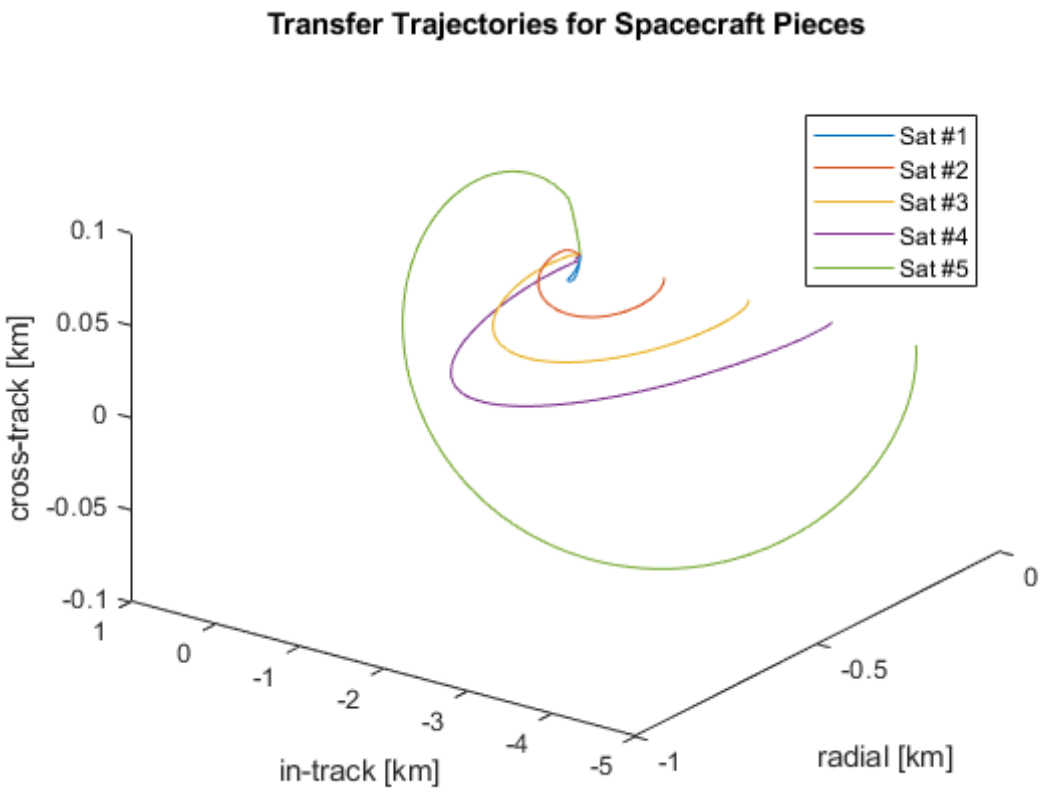
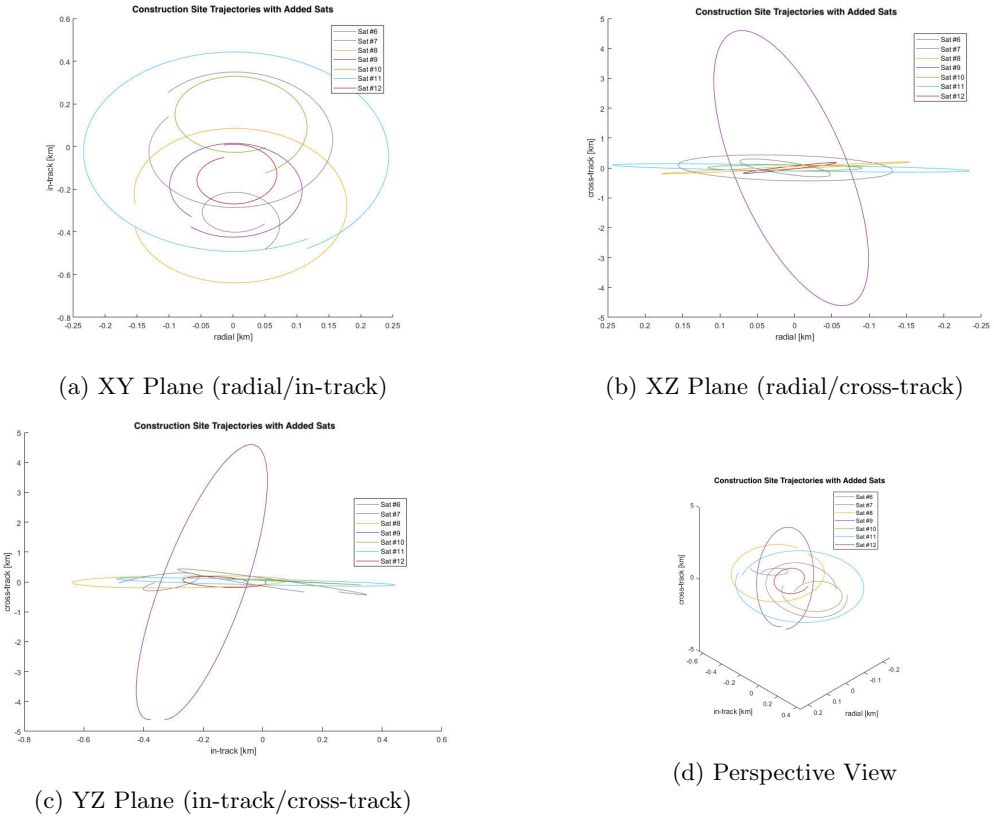


Figure 6.8: Transfer Trajectories to Construction Zone – Perspective View

6.2.4 Death of a Spacecraft – Debris Generation

This scenario considers an important case of swarm operations: what happens when a spacecraft in the swarm fails? In this trial case, observer spacecraft #6 and #7 are set to unexpectedly fail after the pieces to be aggregated have arrived in the construction zone. They fail, as designed, in a passive manner, such that they are not actively thrusting, simply drifting through space in their existing trajectories, propagating forward through time. These spacecraft are referred to as *zombie* spacecraft, and all other spacecraft must take care to avoid them whenever entering new trajectories (the existing trajectories were already computed to be collision free for 10 days, so this is not an issue).

In this scenario, once the spacecraft fail, two replacement spare spacecraft are brought into the swarm from a holding point a few kilometers away. The method described in section 3.2.4 is used for this process, while also flagging the zombie spacecraft as inoperable and thus immovable for the purposes of swarm reconfiguration. The remaining spacecraft are reconfigured slightly to enable safe integration into the swarm for the two replacement spacecraft. This results in the following trajectories in the vicinity of the construction site, as shown in Figure 6.9, with the trajectories in gray being those of the zombie spacecraft.



(a) XY Plane (radial/in-track)

(b) XZ Plane (radial/cross-track)

(c) YZ Plane (in-track/cross-track)

(d) Perspective View

Figure 6.9: Trajectories around Construction Site with Zombie and Replacement Spacecraft

Although the swarm can operate safely for a short period of time (10-20 days) with these spacecraft in the vicinity, the longer they are not dealt with, the more they will hinder the swarm operations as their error envelope grows and the safe volume accessible to all other spacecraft

around the construction site shrinks. Thus, maneuvering using other members of the swarm to nudge these zombie spacecraft out of the swarm will be required.

6.2.5 Stationkeeping Maneuvers to Recycle Trajectories

In order to maintain these trajectories and prevent accumulated errors from causing a significant drift, stationkeeping maneuvers are performed to maintain these trajectories. As described in section 4.3, this method uses a patched RPO scheme, using a two or three burn trajectory, optimizing for minimal ΔV usage over the entire swarm. Running these computations for this swarm scenario, for all 10 spacecraft in the swarm, while also maneuvering to avoid the zombie spacecraft, results in a usage of 27.38 m/s over a 10-day period.

6.2.6 ΔV Usage at Each Stage

Finally, Figure 6.10 shows the ΔV capacity of the swarm over time as the assembly mission progresses. The simulation used models the burns as completely impulsive maneuvers for simplicity, as seen by the sharp vertical lines on the plot. The initial ΔV capacity of each spacecraft is 200 m/s, and on average each spacecraft consumes 6 m/s of ΔV over the course of the scenario's maneuvers.

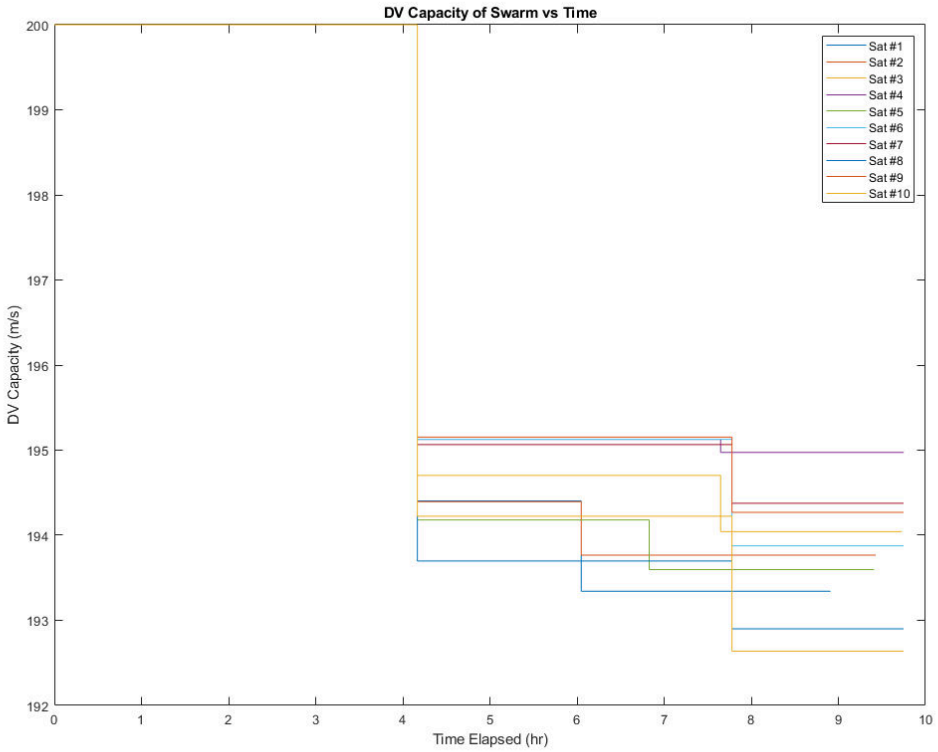


Figure 6.10: ΔV Capacity vs Time for Swarm Spacecraft

This ΔV usage is incurred when injecting into the initial relative motion trajectories, when transferring to each of the spacecraft segments, when transferring to the assembly site, and when performing periodic stationkeeping maneuvers. This plot covers only the first 10 hours of the mission, when the majority of the trajectory transfer maneuvers take place. After the first 10 hrs, the ΔV usage is 27.38 m/s per 10-days for stationkeeping, until the swarm is re-tasked for its next mission.

Chapter 7

Other Considerations

Although this thesis covers much of the fundamentals of swarm trajectory generation and maintenance, primarily in the context of in-space construction, it is in no way a comprehensive analysis of all possible swarm scenarios. During the course of this research, there were dozens of avenues uncovered for further potential research that could not be explored due to time constraints or topical relevancy to the thesis. This chapter will touch briefly on these topics and their importance for the future of multi-spacecraft swarm operations.

7.1 Orbital Reconnaissance

Applications of spacecraft swarms to orbital reconnaissance is a very interesting side problem that came up during the research for this dissertation. Rather than using a swarm for assembly operations, it is possible to use a swarm of spacecraft to cooperatively image or scan an object in orbit at close range, thus characterizing it for a future OOS operation. Not only is this possible in Earth orbit, but in orbit around other celestial bodies as well. The most interesting and complex of these cases is the application to low-gravity objects with irregular gravitational fields, such as asteroids and comets.

Inserting a swarm into orbit around an asteroid to scan and image it from multiple perspectives would be a highly desirable application of the swarm trajectory framework, as it would provide a high degree of reliability for such a mission, since the failure of a single node would not cause the failure of the entire mission. In order to continue this line of research, the trajectory generation method described in this thesis would need to be modified to account for the irregular gravitational field of an asteroid, and the various perturbations associated with nearby celestial bodies.

7.2 Self-Aggregating Swarm

A self aggregating swarm is an interesting sub-problem of swarm spacecraft interactions, since a self aggregating swarm would use the members of the swarm itself to form a larger structure, rather than building a structure from raw materials. This is different from the scenarios considered in this thesis, since each time a part of the structure is aggregated, there is one less free-flying member in the swarm, and thus one less independent sensing source for the Sensor Fusion Kalman Filter. However, when a spacecraft is aggregated to the structure, there will also be one less spacecraft to consider for collision avoidance purposes. An interesting problem to solve would be to see for what size swarm this would be feasible with respect to sensor accuracy and attitude control systems, determining the threshold before the size of the swarm becomes unmanageable with the techniques laid forth in this thesis.

7.3 Computational Distribution

During the course of this research, it is assumed that the computations for trajectory generation and collision avoidance are performed either on the ground, or aboard the spacecraft individually. However, it would be much more efficient to split up the computations and perform them in parallel aboard all the spacecraft in the swarm simultaneously. This has already been explored to a certain extent with the simulations done in this dissertation, and parallel computing is a very

well known method to speed up computational efforts by binning them into parallel processes that can be run independently of each other [197, 198].

To immediately see the increased performance potential of parallel computing for swarm operations, we need only to look at the computations for conjunction analysis. Equation 7.1 shows the number of computations that are required to check conjunction risk between all members of the swarm, using binomial representation. Each spacecraft must be compared to every other spacecraft in the swarm.

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)\cancel{(n-2)!}}{2\cancel{(n-2)!}} = \mathcal{O}(n^2) \tag{7.1}$$

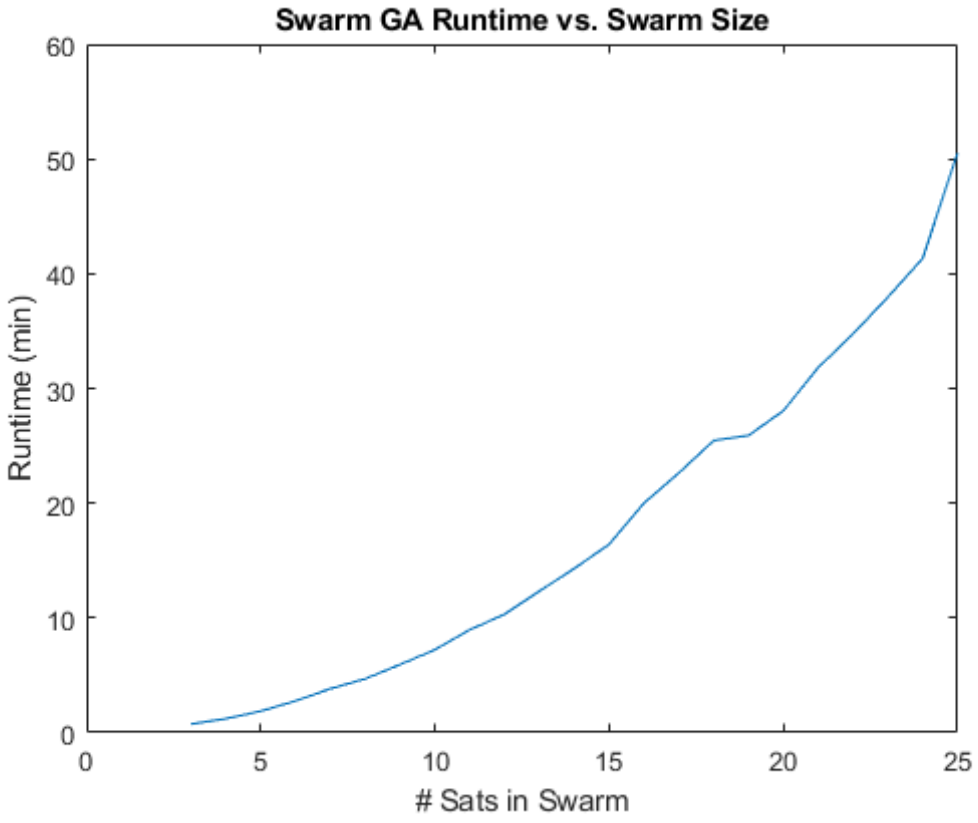


Figure 7.1: Runtime vs Swarm Size – Single Compute Node

Thus, for n spacecraft, n^2 computations must be run. This is also seen experimentally in Figure 7.1, where 100 Monte-Carlo simulations were run at various swarm sizes to determine the mean computational time per swarm size, without parallel processing.

However, as we increase the swarm size, if parallel computation is used, then the number of processing cores increases at the rate $\mathcal{O}(n)$. Thus the computational load will scale linearly as long as parallel computing is used, as seen in Equation 7.2.

$$\frac{1}{n}\binom{n}{2} = \frac{n!}{2!\,n(n-2)!} = \frac{\cancel{n}(n-1)\cancel{(n-2)!}}{2\cancel{n}\cancel{(n-2)!}} = \mathcal{O}(n) \tag{7.2}$$

Although in reality the $\mathcal{O}(n)$ is a theoretical limit, as some overhead computing is required to facilitate the parallel computing load, in practice it is possible to get within 10% of this value [199]. Figure 7.2 shows this experimentally on the CARC supercomputing cluster.

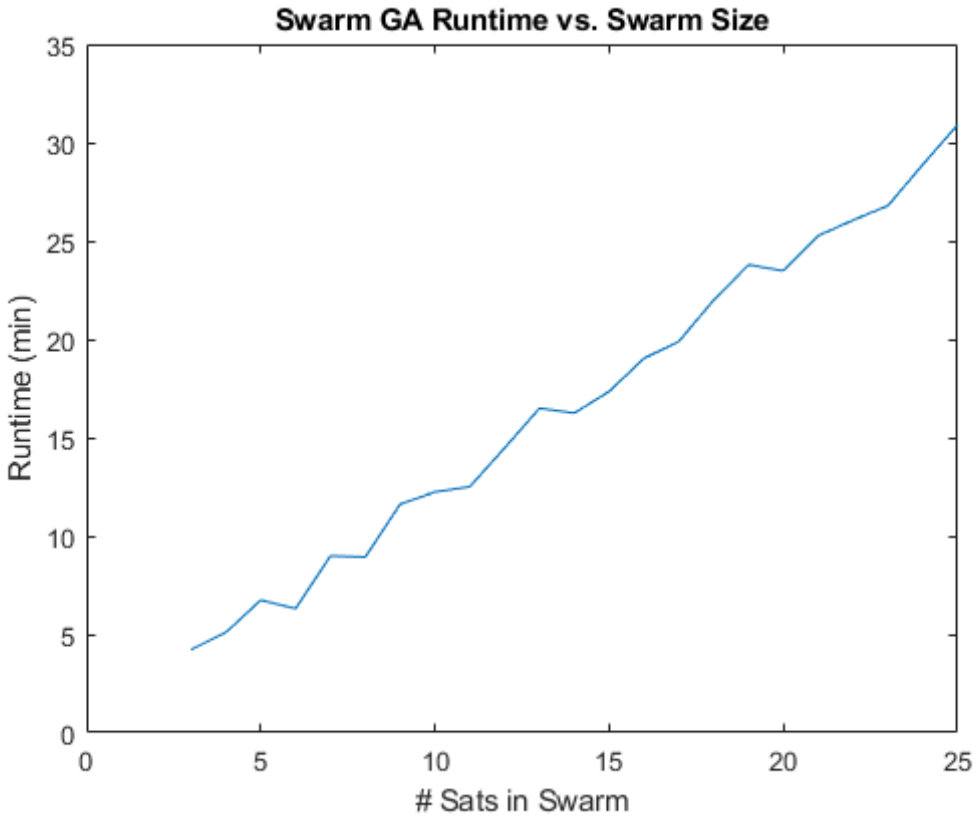


Figure 7.2: Runtime vs Swarm Size – Parallel Computing

The simulation results shown in this dissertation for large swarms have been computed using parallel processing on a high-performance computing cluster, simulating this increased performance as if the computations were running on the individual spacecraft in the swarm. However, what has not been researched is how to do this between spacecraft that can be separated by dozens of kilometers, which can temporarily lose communications with each other. To implement this practically, a set of guidelines and hierarchies may need to be developed to govern what happens in the case that a spacecraft becomes unresponsive [29]; in such a scenario, which spacecraft will take on its computational burden? The implementation of a robust distributed computational scheme is the final step required to upgrade the swarm methodology from a Class 3.5 swarm to a Class 4 swarm, with full autonomy, as described by Nallapu and Thangavelautham [36, 48, 49].

7.4 Light-time Delay for Autonomous Operations

When considering spacecraft swarms in this dissertation, the context has always been that of a swarm in orbit around Earth. Although the same algorithms can be reliably applied to swarms around other celestial bodies, swapping the gravitational models and perturbation sources, there are other factors to consider for such a deep-space swarm than gravitational forces. For example, when considering a swarm constructing a large aperture radio relay system in Mars orbit, there is a 4 to 20 minute light-delay between Mars and Earth [200]. This poses a significant problem with the swarm control framework as it has been defined in this dissertation. One of the assumptions is that there is a possibility of remote intervention from the ground in the case that a serious problem occurs that the automated control modes cannot handle. When in orbit around Mars, this delay will cause any response time from mission operations to be increased by an order of magnitude.

To solve this issue, further research will need to be done in order to find ways to efficiently produce trajectories that can guarantee collision free trajectories for a longer period of time, without

significant ΔV overhead, or more autonomy will need to be added to the system. Such a scenario would be a prime example where primitive artificial intelligence systems can be incorporated into a swarm in order to mitigate complex problems without relying on aid from mission operators 20 light-minutes away [201–206].

7.5 Foreign Object Threats

Although this dissertation considers the effects of zombie spacecraft on the swarm as objects to be avoided, it does not consider any foreign threats, such as debris or solar radiation. Such effects will likely need to be considered for any practical implementation of these algorithms, and this can be done similarly to the designation of keep-out zones, such as used for the dead (*zombie*) spacecraft. Such keep-out zones can be designated to cover the trajectories of large pieces of debris that could cripple spacecraft. However, this will require significant modifications to the collision avoidance algorithms and the ΔV thresholds if the swarm is in a location with a high density of debris, as this will mean that more maneuvers will need to be taken to avoid this debris, with limited advance knowledge of the presence of such debris. Although more and more systems are being developed to track debris with high accuracy [207–217], there is still significant error on existing systems in place, necessitating large corridors to account for the position uncertainty of the debris.

7.6 Irregular Keep-Out Zones

In the case of operations in a highly regulated section of space, such as around the International Space Station, there are large and irregular keep-out zones defined to protect sensitive space assets. Modifications will need to be made to the trajectory determination and maintenance algorithms to enable ΔV efficient trajectories in the vicinity of such regions.

7.7 Search for Globally Optimal Solutions

As noted in Chapter 3, the trajectory solutions determined by the GA solver is not a globally optimal solution, but rather a locally optimal solution that satisfies all the constraints set by the user. Its possible to find better solutions to the problem that exceed the constraints set by the user, however that was not a part of the research goal of this dissertation. Future work can focus on methods by which to converge on a globally optimal solution from the locally optimized results presented in this thesis.

7.8 Implement Advanced Continuous-Thrust Trajectory Generation Techniques

Section 4.5 discusses the methods used in this dissertation to account for trajectory generation when using continuous-thrust transfers (Electric Propulsion). The methods used are quite primitive compared with industry standards [153–158], although they are able to get a good order-of-magnitude approximation of what a true EP-optimize trajectory would be. This was done to simplify the computations, as EP trajectory optimization is not the goal of this dissertation, and in fact is complex enough to make it the topic of its own doctoral thesis. Future research endeavors may involve the application of high-fidelity continuous-thrust trajectories for use in Swarm Trajectory Maintenance.

Chapter 8

Application to Geostationary Spacecraft Sharing Slots

In the decades since the first geostationary satellite was launched in 1964, the limited slots reserved for geostationary spacecraft have been steadily filling, with 554 active satellites currently and numerous more retired spacecraft in the graveyard orbit just above geostationary orbit. Allocation of these spacecraft are managed by the International Telecommunication Union (ITU), which has divided the GEO belt into 0.1 degree slots [218]. Each slot is approximately 73.6 km wide and 100 km tall, as viewed from the equator. To keep up with increased demand, satellite operators have begun sharing slots with up to six satellites in a slot. Not every slot is considered equal as the longitude of the slot determines its value, with slots over populated areas being more valuable than non-populated. Operators plan to increase the per-slot number to 10 or more over the next decade, forming swarms of spacecraft in order to meet rising telecommunication demands. To maintain safe distances between co-located swarm members, operators must perform station-keeping maneuvers for each satellite to offset orbital perturbations. However, it is possible to use these perturbations to the swarm's advantage. While typical geostationary satellites experience a low level of drift, those sharing slots try to minimize risk of inadvertent collision between co-located spacecraft. Using machine learning, it is possible to come up with a solution for a set of non-intersecting trajectories that enable a higher degree of drift than is typically allowed, while maintaining the strict safety criteria required by operators. This reduces ΔV consumption during

station-keeping maneuvers, and can allow many more spacecraft to be co-located in a single 74 km wide slot.

This chapter will cover the application of cooperative satellite swarm trajectory generation and maintenance in order to reduce the propellant utilization of spacecraft in GEO, maintaining a dynamic formation flying configuration. This enables each spacecraft to perform their individually required operations, while also choosing trajectories that prevent collision risks under free-flight trajectories for an extended duration.

8.1 Trajectory Generation & Collision Avoidance

Trajectories for co-located geostationary spacecraft are referred to as a *swarm* for the purposes of this analysis. This has been adapted from its original use case of an arbitrary sized swarm for robotic construction in LEO [1], extending it to be applicable to a small set of spacecraft in geostationary orbit which are all co-located within the same ITU slot. For GEO spacecraft, the principle constraint is to avoid contact with another satellite in a specified ITU slot, while also being able to point at a set of receivers on the ground.

The primary method of collision avoidance is implemented directly in the genetic algorithm (GA) trajectory generation method itself. Trajectories are generated such that, over a specified time period (at least 24hrs), there is no risk of collision between any spacecraft in the swarm. During the trajectory generation process, if there is a collision predicted, then the GA will isolate the spacecraft that are involved in the collision and determine how to most efficiently mitigate it, as well as which spacecraft has the least restrictions on it to modify its trajectory. The simplest solution is not to change the trajectory at all, but instead adjust the insertion time of a satellite into its trajectory so as to adjust its phase, thereby avoiding a collision. If this is not possible, or if this results in further collisions, then the solver will try slight variations of the trajectories until one is found that does not result in any conjunction.

The secondary method of collision avoidance is an active system that uses Kalman filtering to determine the estimated relative position of all objects in the swarm, and perform corrective maneuvers if any begin to drift. In order to maintain safe trajectories that avoid collisions between spacecraft in the swarm, a Sensor Fusion Kalman Filter (SFKF) is used to accurately determine, in real-time, the position and velocity of each spacecraft. This is done using only the information available to the swarm members themselves, through external sensors, and without additional information from operators on the ground. Using the filtered data, a set of covariance matrices are obtained for each predicted position and velocity, setting the upper limits of the error bars on the measured and processed data.

In the simulated scenarios that will be covered in Section 8.4, sensor data with random input errors were generated to feed into the SFKF, using the following sigma values for a normally distributed random error generation:

- GPS position error : 3 m for LEO and 1000 m for GEO [219]
- Radar/LIDAR ranging error : 5% of measured value along range vector
- Speed measurement error : 1% of measured value along range vector
- Attitude knowledge error : 0.5 deg

A sensor fusion Kalman filter is a extension of an unscented Kalman filter to incorporate the data from multiple sensors. This is implemented using multiple measurement update cycles to incorporate the shared data of the swarm to further refine the covariance ellipsoid for each spacecraft. Figure 8.1 shows an example of the sensor fusion process, where the covariance of the position between *Sat #1* and the *Client* spacecraft can be improved by fusing the data from all the swarm spacecraft, even taking into account the GPS position errors defining the locations of each swarm spacecraft with respect to *Sat #1*.

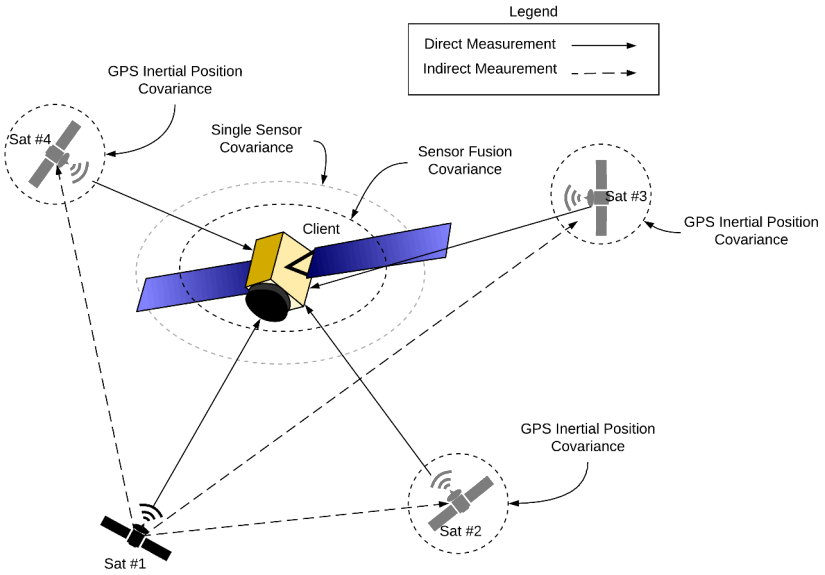


Figure 8.1: Sensor Fusion Diagram

Using Kalman filtering, trajectories can also be assigned a specific corridor, where if the spacecraft drifts too far from the designated trajectory (as determined by fusing sensor data and filtering it through the sensor fusion Kalman filter), a small correction maneuver is performed to re-align the spacecraft to its target.

8.2 Patched RPO

In order to transition between various relative motion trajectories, a method of patched RPO has been devised. This method takes two defined states, separated by a defined period of time, to determine the most efficient set of impulsive maneuvers to be able to safely transition between these states. This is done using two, three, or four impulsive maneuvers, depending on what method yields the lowest ΔV consumption, while maintaining safe operations with nearby spacecraft.

Each impulsive maneuver begins a new trajectory segment. To compute the shape of the transfer trajectory, and the initial velocity vector required to place the spacecraft on that trajectory, a

two stage solver is used, using the solution of the linearized C-W equation to seed the nonlinear solver for the perturbed gravitational field solution. The solution to the C-W equations [50] yields the initial velocity of the transfer trajectory, $\vec{v_0}$, which is close to the desired solution. However it is a linearized approximation of the unperturbed solution, and thus is not the final solution.

The perturbed solution can be solved iteratively, numerically solving the second order ODE in Equation 8.1.

$$\frac{d^2\vec{r}}{dt^2} = -\mu\frac{\vec{r}}{\|\vec{r}\|^3} + \vec{a}_{grav-pert} + \vec{a}_{sun-moon} + \vec{a}_{SRP} \tag{8.1}$$

Where $\vec{a}_{grav-pert}$ is computed using Equation 2.21, $\vec{a}_{sun-moon}$ is computed using Equation 2.22, and $\vec{a}_{SRP}$ is computed using Equation 2.23. This system of second order ODEs is then solved iteratively to find the initial velocity $\vec{v}_{0_{pert}}$ such that the final position, $\vec{r}_f$ is the desired trajectory endpoint. This was done using the *fsolve* method in MATLAB, using the velocity $\vec{v}_0$ from the C-W linear solution as the initial solver guess to jumpstart the iterative process.

This is done initially prior to the start of the maneuver, in order to plan out the trajectory being travelled and prepare the spacecraft for the burn maneuver. Once the initial ΔV is applied, the spacecraft has been injected onto the transfer trajectory. A Kalman filter is then used to compute the estimated true position of the spacecraft, using its onboard sensors. This estimated position is then used to recompute the target point at the end of the transfer arc, at a rate of once per second. If the trajectory is determined to deviate by more than 5 m, then Equation 8.1 is solved again iteratively, and a small maneuver is performed to maintain the original target point.

8.3 Stationkeeping Maneuvers

Although the swarm trajectories are propagated forwards in time for a period of 10 days to check for collisions, they will eventually begin to drift away from each other due to orbital perturbations. In order to repeat this trajectory, a two-part trajectory change maneuver must be performed to

either reset the swarm onto the same trajectories as the initial conditions, or to generate a new set of trajectories with minimal deviation from the current end state, while maintaining the requirements of the swarm. To *reset* the swarm trajectories, in essence plotting a relative motion trajectory from the end of the propagated swarm trajectory back to its initial state vector, a two impulse trajectory change maneuver is used. See Section 4.5 for an overview of the methodology used to deal with continuous thrust (Electric Propulsion) maneuvers.

Figure 8.2 depicts a set of these return trajectories visually for a swarm of three spacecraft. The trajectories in grey are the previously travelled swarm trajectory generated using GAs, and the colored trajectories are the three return transfer arcs, using numerically computed two-impulse trajectories.

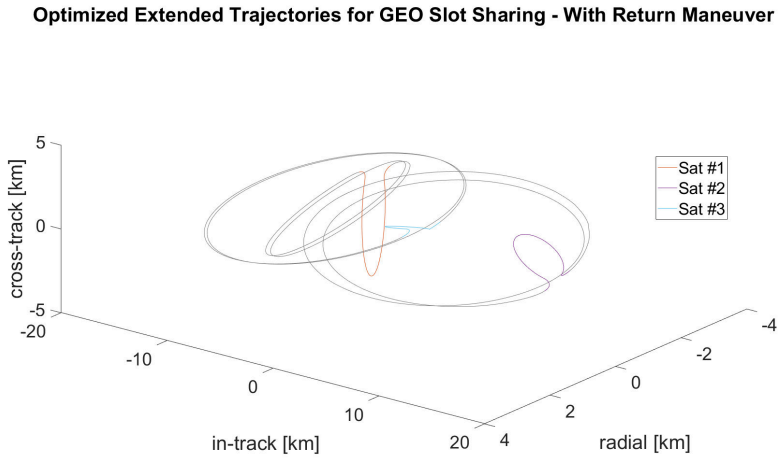


Figure 8.2: Return Trajectories

8.4 Numerical Results from Simulation Trials

8.4.1 Comparison to KOREASAT three satellite swarm

The algorithms developed in this paper were directly compared against existing geostationary collocation strategies. The first test case was that of the KOREASAT three satellite swarm. Lee et al. use the eccentricity and inclination (E/I) vector separation strategy to collocate the three

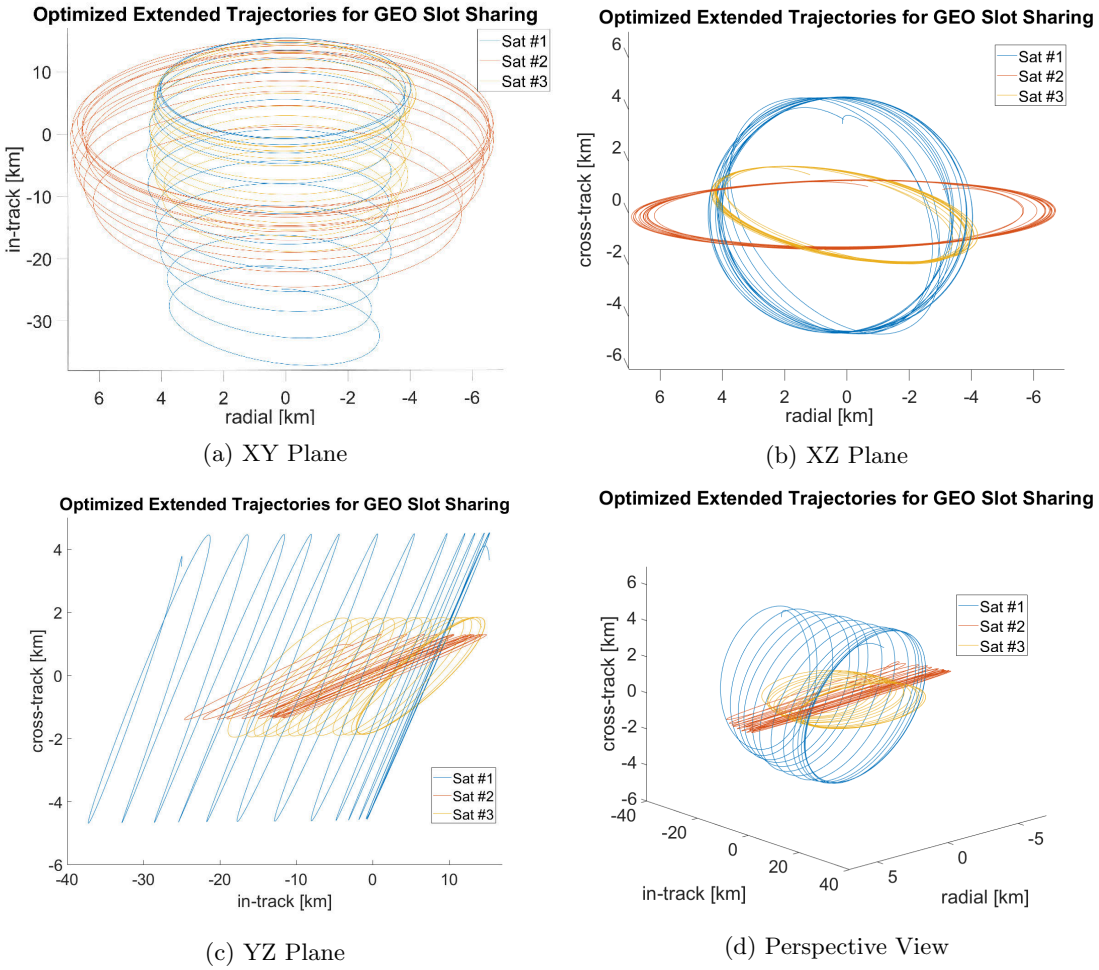
satellites [220]. For ease and simplicity, each satellite was set with a mass of 2300 kg and cross-sectional area of 57 m^2, which is the largest and heaviest of the trio. The satellites are located in a longitudinal control box at 116° E ±0.05°. Using the E/I method, Lee et al. calculated a ΔV total over a 14 week period to be 35.6131 m/s. To achieve this number all three satellites had to perform stationkeeping maneuvers every other day on average.

Now, using Genetic Algorithms when run with the same initial conditions the total ΔV for all three satellites is 16.3056 m/s. By allowing the satellites to drift naturally within their constrained boxes, the satellites are only required to fire their thrusters once per week to return them to their initial state to begin drifting again. Table 2 shows the direct comparison of these results.

Table 8.1: Bi-Weekly **ΔV** Results for KOREASAT Comparison

Satellite	**E/I ΔV (m/s)**	**GA Free Return ΔV (m/s)**
KS 1	11.7318	11.0342
KS 2	11.89025	1.8543
KS 3	11.99105	3.4170
Total ΔV	35.6131	16.3056

Figure 8.3 depicts the solution associated with the ΔV values from Table 8.1. The trajectories for each satellite in a 14 day period are shown. At the end of the period the satellites use their on board propulsion to return to their initial positions to begin drifting again with the two impulse method described above.



(a) XY Plane

(b) XZ Plane

(c) YZ Plane

(d) Perspective View

Figure 8.3: Co-Located Trajectories – KOREASAT Comparison

The three satellites shown in Figure 8.3 also maintain at least a 1 km separation at all times. What is also interesting to note is that by using the same mass for all of the satellites, this test incurs greater penalties from the orbital perturbations; however, the solution derived from the GA still requires less than 50% of the ΔV used by the traditional E/I method.

8.4.2 Comparison to Convex Optimization Strategy

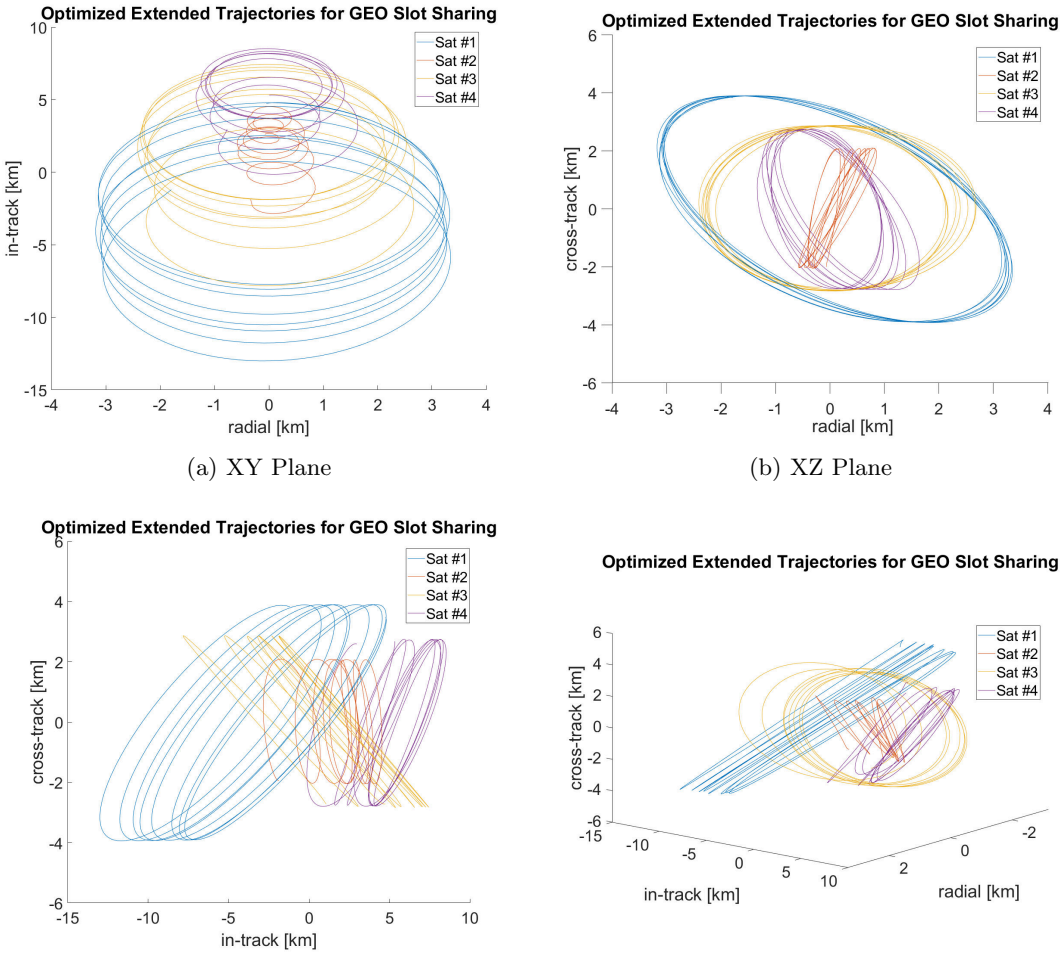
The second test case was a comparison to the German Aerospace Center's (DLR) convex optimization method for collocating Geostationary satellites [221]. This method is based on using a

leader-follower scheme for control. The optimization is done using a cost function to minimize propellant consumption and total number of maneuvers. The entire convex optimization method used in their paper is enabled by a linear time varying formulation of orbital dynamics in terms of non singular orbital elements. The simulation is constructed for a fleet of four satellites within a single GEO slot, with a seven day maneuver cycle.

The strategy used for orbital maneuvering for the lead satellite is a sun-pointing perigee strategy. To implement this strategy the leader follows a circle within the eccentricity plane with an eccentricity offset of 2×10^{-4}. The follower satellites normal states were chosen to be consistent with the E/I separation strategy as described in [220]. All four of the satellites were chosen to have a mass of 3000 kg and all had a surface area of 120 m^2 except for the lead satellite, which had a surface area of 90 m^2.

To setup the comparable test case with the use of Genetic Algorithms, all the satellites were configured with the 3000 kg mass. However for simplicity the surface area was chosen as a constant 120 m^2 for all four satellites. The maneuver cycle used for the GA test is eight days long.

Figure 8.4 depicts the solution associated with the ΔV values from Table 8.2. The trajectories for each satellite in a 14 day period are shown. At the end of the period the satellites use their on board propulsion to return to their initial positions to begin drifting again with the two impulse method described above.



(a) XY Plane

(b) XZ Plane

(c) YZ Plane

(d) Perspective View

Figure 8.4: Co-Located Trajectories – DLR Study Comparison

Table 8.2: Yearly ΔV Results for DLR Comparison

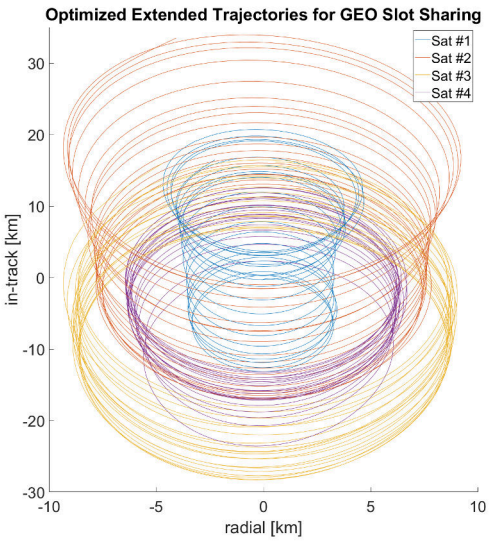
Satellite	**Actual ΔV (m/s)**	**GA Free Return ΔV (m/s)**
SAT 1	49.86	71.52
SAT 2	50.87	43.95
SAT 3	68.10	35.03
SAT 4	68.06	43.63
Total ΔV	236.89	194.14

The GA free return strategy offers clear ΔV benefits compared to the convex optimization method. The satellites on the eight day maneuver cycle used 18% less ΔV over the entire year as compared to the leader-follower seven day scheme. The GA method is able to save this ΔV by allowing the satellites to drift closer together, and react more quickly when compared to the convex method. Minimum separation in the GA test was 1 km where the minimum separation set by the DLR team was 6.03 km.

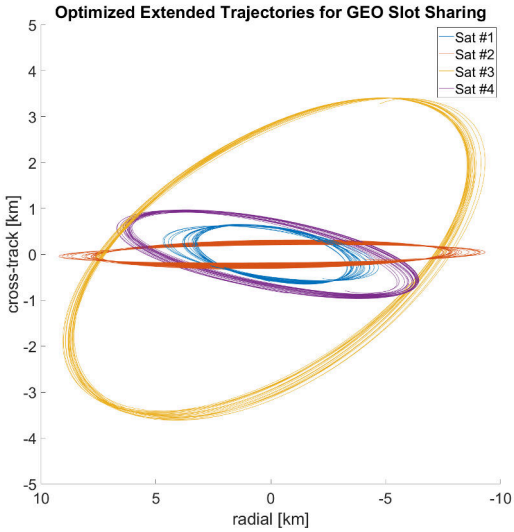
Another important aspect that allowed the GA to perform far better than the convex optimization problem is by combining many different solutions into one optimal trajectory plan for all four satellites. The GA was run eight consecutive times, and the four satellite trajectories which required the least ΔV over one year were then extracted. These extracted trajectories were then run through the conjunction deconflicter to determine if there were any collisions between the satellites. To avoid collisions the program would then delay the injection time of the satellites to avoid collisions.

8.4.3 Comparison to 4 co-located GEO spacecraft

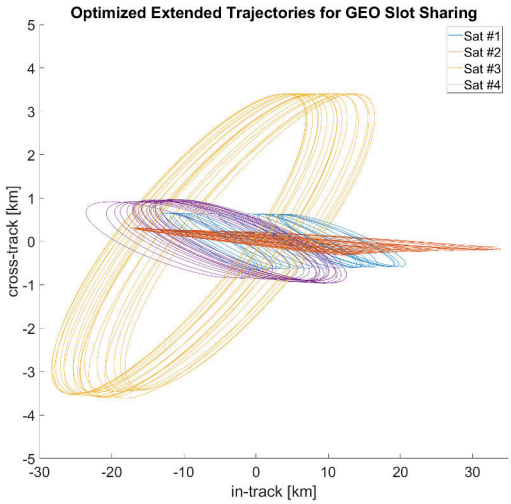
The third and final test case compares this genetic algorithm swarm trajectory generation method with a real-world case of four co-located GEO spacecraft. This test case is carried out with each spacecraft restricted to an angular separation of $\pm 0.05°$. Figure 8.5 shows the resultant trajectories for the GA solution for all four spacecraft. Each trajectory depicts a 10-day period of the spacecraft, which is designed to be repeating, maintained using periodic stationkeeping maneuvers.



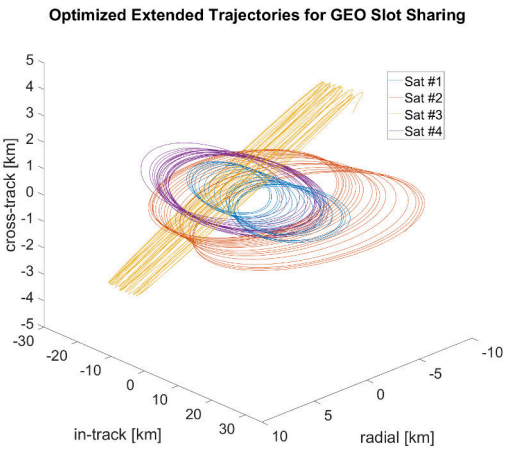
(a) XY Plane



(b) XZ Plane



(c) YZ Plane



(d) Perspective View

Figure 8.5: Co-Located Trajectories – Real-World Data Comparison

Table 8.3: Yearly ΔV Results for Real-World Comparison

Satellite	**Actual ΔV (m/s)**	**GA Free Return ΔV (m/s)**
GEO 1	46.845	38.502
GEO 2	46.336	56.958
GEO 3	47.542	41.946
GEO 4	47.395	29.287
Total ΔV	188.118	166.693

The plots and data above show that the genetic algorithm method in use for generating and maintaining co-located GEO spacecraft trajectories for this four-satellite solution is more efficient than traditional stationkeeping methods, primarily due to the ability of the sensor fusion Kalman filtering collision avoidance methods to allow a greater degree of freedom for the spacecraft to drift through the GEO slot without conjunctions.

8.5 Summary of GEO Applicaton Results

Although GEO spacecraft co-location has typically been limited to three or four spacecraft, the increasing demand for high-speed communication systems, along with the recent development of satellite servicing [222] will mean that there most likely will be more spacecraft stationed in Geosynchronous orbital slots, with increased average operational lifetimes. Given the anticipated crowding of the GEO belt, and the ever increasing size of deployments aboard such Geosynchronous communications satellites, it is more important now than ever to develop efficient and adaptable methods of multi-spacecraft swarm trajectory generation and maintenance, enabling spacecraft to operate in close proximity with reduced collision risk. Additionally, systems that include active collision avoidance by way of high fidelity relative motion sensing, which can be

achieved using a Sensor Fusion Kalman Filtering method, could be very beneficial if implemented on co-located spacecraft in the GEO belt.

Based on the three test cases covered in this paper, the swarm trajectory generation method outlined here is at least as efficient, if not more efficient, than existing methods when applied to swarms of three or four spacecraft, and its true gains can be seen when applied to larger swarms [223]. Although traditional methods for GEO slot sharing collision avoidance have worked well in the past for smaller swarms, such as eccentricity/inclination vector separation [224], such methods lose efficiency when scaling up to large swarm sizes. This occurs since the available relative positions in which to place a trajectory, such that the eccentricity and inclination vectors decrease as the swarm size increases, becomes limited for large swarms [225]. Thus, novel methods for swarm trajectory design and maintenance are required for future GEO spacecraft co-location, with one such method being proposed in this thesis.

Chapter 9

Conclusions and Ongoing Work

This dissertation has explored the problem of satellite swarm trajectory generation and maintenance, proposing a solution that employs machine learning and sensor filtering to achieve dynamic, reconfigurable trajectories that can handle external disturbances while minimizing propellant consumption. The method determined to effectively perform this task was the use of nested Genetic Algorithms, alongside a Sensor Fusion Kalman Filter (SFKF). Though this thesis does not dive deep into implementation techniques for specific spacecraft, instead seeking to provide a broader understanding of swarm configuration and control and its limitations, it provides numerous avenues for further research, which will be discussed below.

9.1 Real-Time Kalman Filtered Simulations

Chapters 3 and 4 describe the novel method for swarm trajectory generation and maintenance, which forms the foundation of this thesis. The most interesting results from applications of this method are shown in Chapters 5 and 6, which describe the capabilities and limitations of the system, as well as the generated trajectories for specific test cases.

Chapter 5 considers in detail the various behavioral stresses of the system, in order to determine the limits of the genetic algorithm and SFKF method. Probing the limits of the algorithm is

especially important, as it is one of the simplest, albeit somewhat computationally intensive, methods to determine the regime of operation for future users of the swarm GA method. One of the most interesting cases considered in this chapter is that of the unexpected loss of a vehicle. In this case, a swarm member is assumed to be lost, either from a communications standpoint, or physically nonfunctional, and thus unable to respond to commands, either from the ground or from other spacecraft in the swarm. In this case, it is considered a *zombie satellite*, for all intents and purposes a piece of debris. Recall that in Section 1.5, one of the top-level assumptions about the swarm in question is that it is designed to fail in a safe mode, such that a failed spacecraft will not maneuver until communications have been restored. In this case, the remaining functional spacecraft in the swarm will need to perform trajectory alteration maneuvers to avoid this *zombie satellite*, for which the uncertainty on its position will keep growing over time, using up precious ΔV, but preserving the integrity of the swarm.

Another edge case considered in Chapter 5 is the required response to a dynamic construction environment, where a structure or body is being aggregated over the course of the mission. As this aggregation occurs, the object will grow in size and mass, with its rotational inertia properties changing significantly as well. This will result in a variable set of boundaries that the swarm will have to adapt to avoid a collision with the aggregate body. The solution devised, and the simulations of this method, show that such a problem is in fact compatible with the GA framework, although it will increase the resource overhead as compared to a static swarm environment, as would be expected.

Chapter 6 considers an example swarm configuration from start to finish for an on-orbit construction project. The example used was the robotic assembly of an interplanetary transport ship, launched from Earth in pieces, to be assembled in LEO. The resulting simulation demonstrates how the GA framework works from the user's inputs to generate trajectories bridging two states in space and time, minimizing ΔV and collision risk along the way. The scenario also employs the use of a SFKF in order to correct in-transit deviations and measurement anomalies in real-time,

using conservative error estimates, demonstrating that the SFKF is a viable method for real-time collision avoidance.

9.2 GEO Swarms Analysis

Using swarm trajectory generation and maintenance techniques, the simulation results and comparison to real world data shown in Chapter 8 demonstrate that swarm co-location of spacecraft in GEO can be more efficient than traditional methods, if applied correctly. This can enable a larger number of spacecraft to co-locate in the already dwindling space in the GEO belt, while providing real-time collision avoidance methods for improved safety in such a densely populated orbit.

Although traditional methods for GEO slot sharing trajectory generation and collision avoidance have worked well in the past for smaller swarms, such as eccentricity/inclination vector separation [224], such methods lose efficiency when scaling up to large swarm sizes. This occurs since the available relative positions in which to place a trajectory, such that the eccentricity and inclination vectors decrease as the swarm size increases, becomes limited for large swarms [225]. Thus, novel methods for swarm trajectory design and maintenance are required for future GEO spacecraft co-location, with one such method being proposed in this thesis.

9.3 Hardware Testing

A common method of validation for real-time space operations with hardware-in-the-loop is the use of an Air Bearing Platform (ABP) for near-frictionless simulations in three to six degrees of freedom [44, 226–246]. The University of Southern California's Space Engineering Research Center (SERC) has developed an in-house manufactured 3-DOF Air Bearing Platform (ABP), which has the ability to simulate the frictionless environment of space in a single plane. This testbed is comprised of small floating platforms with pressurized air tanks that are able to use

circular air-bearing diaphragms to float on an air cushion over a calibrated optical glass surface. Using cold-gas thrusters, these *floatbots* are able to move across the glass surface, simulating a frictionless environment in space. This makes the platform ideal for testing RPO and docking activities without the expense of a microgravity simulator such as the *vomit comet* [247], or testing aboard the International Space Station (ISS) itself [248].

Another widely used method of hardware simulation that is highly applicable to swarm operations is the use of remote-controlled aerial drones, as these platforms have already been used to demonstrate air-based swarm operations on extremely large scales [133, 181, 249–257]. Using drones, with real-time processing capabilities that are on par or exceeding those of existing small satellites, it is possible to demonstrate real-time rendezvous and proximity operations for spacecraft swarms. This can allow ground-based testing of Sensor Fusion Kalman Filtering and changing swarm configurations in 6-DOF, rather than the 3-DOF enabled by ABPs.

Although hardware testing using both drones and ABPs were planned to demonstrate and validate this swarm trajectory generation and maintenance system using real-world sensors [46], this testing was pushed aside to the unforeseen circumstances surrounding the COVID-19 global pandemic [258, 259]. This testing and validation will thus be left for future research endeavors, with computer simulations sufficing for the purposes of this dissertation.

9.4 Future Work

Following hardware testing to validate and demonstrate the system, future avenues of research to continue this work include applications towards missions such as modular spacecraft assembly, on-orbit servicing, and on-orbit reconnaissance, among others. Modular spacecraft assembly requires multiple spacecraft to operate in close proximity to join pieces or components together to form a larger object, as seen in the example scenario in Chapter 6. On-orbit servicing is another example mission type that can benefit from swarm operations, as multiple spacecraft operating together

can provide a higher degree of redundancy for the mission, as well as to allow rapid scanning of an object in 3D from multiple vantage points. Orbital reconnaissance is another field where swarm trajectory generation can be applied, and deserves further research, especially around irregularly shaped bodies such as asteroids.

Regarding dynamic construction sites in LEO, an area for future work is to explore the relationship of an aggregating swarm, where each member of the swarm is itself a part of the object to be constructed, rather than an assembly robot moving pieces into position. In this case, the swarm would itself reconfigure and attach itself into a structure, where the swarm spacecraft would become nodes of a larger structure. In this case, the existing algorithms do not quite apply, and will need significant modification to be applicable to the scenario.

Finally, an area for future work is to analyze how to distribute the computational load evenly across the swarm to generate trajectories on-board the swarm rather than from the ground. This is especially important for missions where there is significant light delay between the mission controllers and the swarm, such as a construction mission in Martian orbit. This will require a computational framework to be developed that can distribute the computational tasks across the swarm, while also considering how to handle node or communications failures.

It is my hope that the algorithms, solutions, and results presented in this dissertation can contribute to a better understanding of how to control large swarms of spacecraft in close proximity, improving the safety and reliability of high-density in-space construction sites in the near future.

Reference List

[1] Rahul Rughani and David A Barnhart. Safe Construction in Space: Using Swarms of Small Satellites for in-Space Manufacturing. *2020 SmallSat Conference, Logan, Utah (Virtual)*, 2020.

[2] LaunchSpace. New Satellite Constellations Will Soon Fill the Sky. *SpaceDaily*, 2018.

[3] The 2018 Summer of Satellite IoT – 18 Startups, Over 1,600 Satellites. `https://www.spaceitbridge.com/the-2018-summer-of-satellite-iot-18-startups-over-1600-satellites.htm`.

[4] S. Alfano, D.L. Oltrogge, and R. Shepperd. "Leo Constellation Encounter And Collision Rate Estimation: An Update". In *2nd IAA Conference on Space Situational Awareness (ICSSA), Washington D.C., IAA-ICSSA-20-0021*, Jan 15, 2020.

[5] Owen Brown, Paul Eremenko, and Paul Collopy. Value-Centric Design Methodologies for Fractionated Spacecraft: Progress Summary From Phase I of the DARPA System F6 Program. In *AIAA Space 2009 Conference & Exposition*, page 6540, 2009.

[6] Jacob Everist, Kasra Mogharei, Harshit Suri, Nadeesha Ranasinghe, Berok Khoshnevis, Peter Will, and Wei-Min Shen. A System for in-Space Assembly. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2356–2361. IEEE, 2004.

[7] David Barnhart, Lisa Hill, Margaret Turnbull, and Peter Will. Changing Satellite Morphology Through Cellularization. In *AIAA SPACE 2012 Conference & Exposition*, page 5262, 2012.

[8] Haitao Chang, Panfeng Huang, Yizhai Zhang, Zhongjie Meng, and Zhengxiong Liu. Distributed Control Allocation for Spacecraft Attitude Takeover Control via Cellular Space Robot. *Journal of Guidance, Control, and Dynamics*, 41(11):2499–2506, 2018.

[9] Haitao Chang, Panfeng Huang, Zhenyu Lu, Zhongjie Meng, Zhengxiong Liu, and Yizhai Zhang. Cellular Space Robot and Its Interactive Model Identification for Spacecraft Takeover Control. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3069–3074. IEEE, 2016.

[10] Wilfried Hofstetter, Olivier de Weck, and Edward Crawley. 9.1. 3 Modular Building Blocks for Manned Spacecraft: A Case Study for Moon and Mars Landing Systems. In *Incose International Symposium*, volume 15, pages 1296–1312. Wiley Online Library, 2005.

[11] Olivier de Weck, William Nadir, Justin Wong, Gergana Bounova, and Thomas Coffee. Modular Structures for Manned Space Exploration: The Truncated Octahedron as a Building Block. In *1st Space Exploration Conference: Continuing the Voyage of Discovery*, page 2764, 2005.

[12] Charlotte Mathieu and Annalisa Weigel. Assessing the Flexibility Provided by Fractionated Spacecraft. In *Space 2005*, page 6700. American Institute of Aeronautics and Astronautics (AIAA), 2005.

[13] M Kortman, S Ruhl, Jana Weise, Joerg Kreisel, T Schervan, Hauke Schmidt, and Athanasios Dafnis. Building Block Based iBoss Approach: Fully Modular Systems With Standard Interface to Enhance Future Satellites. In *66th International Astronautical Congress (Jersualem, Israel*, 2015.

[14] David Barnhart, Brook Sullivan, Roger Hunter, Janine Bruhn, Erin Fowler, Lucille M Hoag, Scott Chappie, Glen Henshaw, Bernie E Kelm, Tim Kennedy, et al. Phoenix Program Status-2013. In *AIAA SPACE 2013 conference and exposition*, page 5341, 2013.

[15] Brook Sullivan, David Barnhart, Lisa Hill, Paul Oppenheimer, Bryan L Benedict, Gerrit Van Ommering, Laurie Chappell, John Ratti, and Peter Will. DARPA Phoenix Payload Orbital Delivery System (PODs):"FedEx to GEO". In *AIAA SPACE 2013 conference and exposition*, page 5484, 2013.

[16] Talbot Jaeger and Walter Mirczak. Phoenix and the New Satellite Paradigm Created by HISat. *2014 SmallSat Conference, Logan, Utah*, 2014.

[17] Leena Singh, Matthew Fritz, Sagar Bhatt, Nazareth Bedrossian, Timothy Henderson, and Bradley Moran. On the Phoenix ADCS-M3D Architecture. In *AIAA SPACE 2013 Conference and Exposition*, page 5535, 2013.

[18] Bruce Davis, William Francis, Jonathan Goff, Michael Cross, and Daniel Copel. Big Deployables in Small Satellites. *2014 SmallSat Conference, Logan, Utah*, 2014.

[19] John Lymer, William R Doggett, John Dorsey, Lynn Bowman, Alfred Tadros, Bruno Hollenstein, Bruce King, Ken Emerick, Mark Hanson, and Joel Boccio. Commercial Application of in-Space Assembly. In *AIAA SPACE 2016*, page 5236. American Institute of Aeronautics and Astronautics (AIAA), 2016.

[20] Lisa Hill, David Barnhart, Erin Fowler, Roger Hunter, Lucille M Hoag, Brook Sullivan, and Peter Will. The Market for Satellite Cellularization: A Historical View of the Impact of the Satlet Morphology on the Space Industry. In *AIAA SPACE 2013 Conference and Exposition*, page 5486, 2013.

[21] Máximo A Roa, Korbinian Nottensteiner, Armin Wedler, and Gerhard Grunwald. Robotic Technologies for in-Space Assembly Operations. *Advanced Space Technologies in Robotics and Automation*, 2017.

[22] Dave Waller, Jeanette Domber, and Roberta M Ewart. Serviceable Space Structures for the Future: Enabling on-Orbit Logistics. In *AIAA SPACE and Astronautics Forum and Exposition*, page 5174, 2017.

[23] Wendel K Belvin, William R Doggett, Judith J Watson, John T Dorsey, Jay E Warren, Thomas C Jones, Erik E Komendera, Troy Mann, and Lynn M Bowman. In-Space Structural Assembly: Applications and Technology. In *3rd AIAA Spacecraft Structures Conference*, page 2163, 2016.

[24] Talbot Jaeger, Walter Mirczak, and Bill Crandall. Cellularized Satellites-a Small Satellite Instantiation That Provides Mission and Space Access Adaptability. *2016 SmallSat Conference, Logan, Utah*, 2016.

[25] Talbot Jaeger, Walter Mirczak, and Bill Crandall. Cellularized Satellites–Initial Experiments and the Path Forward. In *31st Space Symposium, Technical Track, Colorado Springs*, 2016.

[26] Jeff Foust. Three Companies Studying "Orbital Outpost" Space Station Concepts for Defense Department. Retrieved from `https://spacenews.com/three-companies-studying-orbital-outpost-space-station-concepts-for-defense-department/`.

[27] David A Barnhart and Rahul Rughani. On-Orbit Servicing Ontology Applied to Recommended Standards for Satellites in Earth Orbit. *Journal of Space Safety Engineering*, 2020.

[28] David E Gaylor and Brent William Barbee. Algorithms for Safe Spacecraft Proximity Operations. In *AAS/AIAA Spaceflight Mechanics Meeting*, 2007.

[29] Dario Izzo and Lorenzo Pettazzi. Autonomous and Distributed Motion Planning for Satellite Swarm. *Journal of Guidance, Control, and Dynamics*, 30(2):449–459, 2007.

[30] Ismael Lopez and Colin R McInnes. Autonomous Rendezvous Using Artificial Potential Function Guidance. *Journal of Guidance, Control, and Dynamics*, 18(2):237–241, 1995.

[31] GL Slater, SM Byram, and TW Williams. Collision Avoidance for Satellites in Formation Flight. *Journal of Guidance, Control, and Dynamics*, 29(5):1140–1146, 2006.

[32] I Ross, J King, and Fariba Fahroo. Designing Optimal Spacecraft Formations. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, page 4635, 2002.

[33] G Di Mauro, R Bevilacqua, D Spiller, J Sullivan, and S D'Amico. Continuous Maneuvers for Spacecraft Formation Flying Reconfiguration Using Relative Orbit Elements. *Acta Astronautica*, 153:311–326, 2018.

[34] Daniel Morgan, Soon-Jo Chung, Lars Blackmore, Behcet Acikmese, David Bayard, and Fred Y Hadaegh. Swarm-Keeping Strategies for Spacecraft Under J2 and Atmospheric Drag Perturbations. *Journal of Guidance, Control, and Dynamics*, 35(5):1492–1506, 2012.

[35] Chakravarthini M Saaj, Vaios Lappas, and Veysel Gazi. Spacecraft Swarm Navigation and Control Using Artificial Potential Field and Sliding Mode Control. In *2006 IEEE International Conference on Industrial Technology*, pages 2646–2651. IEEE, 2006.

[36] Ravi Nallapu and Jekan Thangavelautham. Spacecraft Swarm Attitude Control for Small Body Surface Observation. *arXiv Preprint arXiv:1902.02084*, 2019.

[37] Saptarshi Bandyopadhyay, Francesca Baldini, Rebecca Foust, Soon-Jo Chung, Amir Rahmani, Jean-Pierre de la Croix, and Fred Y Hadaegh. Distributed Fast Motion Planning for Spacecraft Swarms in Cluttered Environments Using Spherical Expansions and Sequence of Convex Optimization Problems. *2017 International Workshop on Satellite Constellations and Formation Flying*, 2017.

[38] Saptarshi Bandyopadhyay, Francesca Baldini, Rebecca Foust, Amir Rahmani, Jean-Pierre de la Croix, Soon-Jo Chung, and Fred Hadaegh. Distributed Spatiotemporal Motion Planning for Spacecraft Swarms in Cluttered Environments. In *AIAA SPACE and Astronautics Forum and Exposition*, page 5323, 2017.

[39] Corinne Lippe and Simone D'Amico. Spacecraft Swarm Dynamics and Control About Asteroids. *arXiv Preprint arXiv:2001.11026*, 2020.

[40] Simone D'Amico. Autonomous Formation Flying in Low Earth Orbit. *TU Delft - Doctoral Dissertation*, 2010.

[41] William Bezouska and David Barnhart. Decentralized Cooperative Localization With Relative Pose Estimation for a Spacecraft Swarm. In *2019 IEEE Aerospace Conference*, pages 1–13. IEEE, 2019.

[42] Daniel Morgan, Soon-Jo Chung, and Fred Y Hadaegh. Model Predictive Control of Swarms of Spacecraft Using Sequential Convex Programming. *Journal of Guidance, Control, and Dynamics*, 37(6):1725–1740, 2014.

[43] David A Barnhart, Rahul Rughani, Jeremy J Allam, Brian Weeden, Frederick A Slane, and Ian Christensen. Using Historical Practices to Develop Safety Standards for Cooperative on-Orbit Rendezvous and Proximity Operations. *69th International Astronautical Congress (IAC), Bremen, Germany, 1-5 October 2018*, 2018.

[44] David A Barnhart, Ryan H Duong, Lizvette Villafana, Jaimin Patel, and Shreyash Annapureddy. The Development of Dynamic Guidance and Navigation Algorithms for Autonomous on-Orbit Multi-Satellite Aggregation. In *70th International Astronautical Congress (IAC), Washington, DC*, 2019.

[45] Rahul Rughani. Improved Reliability of Orbital Rendezvous Operations by Using Small Satellites for Reconnaissance. Master's thesis, University of Southern California, 2017.

[46] Rahul Rughani, Lizvette Villafana, and David A Barnhart. Swarm RPO and Docking Simulation on a 3DOF Air Bearing Platform. In *70th International Astronautical Congress (IAC), Washington DC, United States*, pages 21–25, 2019.

[47] Rahul Rughani and David Barnhart. Using Genetic Algorithms for Safe Swarm Trajectory Optimization. In *AIAA Scitech 2020 Forum*, page 1916, 2020.

[48] Ravi Teja Nallapu and Jekanthan Thangavelautham. Attitude Control of Spacecraft Swarms for Visual Mapping of Planetary Bodies. In *2019 IEEE Aerospace Conference*, pages 1–16. IEEE, 2019.

[49] Ravi teja Nallapu, Himangshu Kalita, and Jekanthan Thangavelautham. On-Orbit Meteor Impact Monitoring Using CubeSat Swarms. *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2018.

[50] Howard D Curtis. *Orbital Mechanics for Engineering Students*. Butterworth-Heinemann, 2013.

[51] David A Vallado. *Fundamentals of Astrodynamics and Applications*, volume 12. Springer Science & Business Media, 2001.

[52] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*, volume 16. Cambridge university press, 2003.

[53] Chi-Chang J Ho and N Harris McClamroch. Automatic Spacecraft Docking Using Computer Vision-Based Guidance and Control Techniques. *Journal of Guidance, Control, and Dynamics*, 16(2):281–288, 1993.

[54] NK Philip and MR Ananthasayanam. Relative Position and Attitude Estimation and Control Schemes for the Final Phase of an Autonomous Docking Mission of Spacecraft. *Acta Astronautica*, 52(7):511–522, 2003.

[55] JR Burton and WE Hayes. Gemini Rendezvous. *Journal of Spacecraft and Rockets*, 3(1):145–147, 1966.

[56] Barton C Hacker and James M Grimwood. *On the Shoulders of Titans: A History of Project Gemini*, volume 4203. National Aeronautics and Space Administration, 1977.

[57] WL Swan Jr. Apollo-Soyuz Test Project Docking System. *JPL 10th Aerospace Mechanical Symposium*, 1976.

[58] Robert D Langley. Apollo Experience Report: The Docking System. Technical report, National Aeronautics and Space Administration, 1972.

[59] Earle P Blanchard, Richard C Hutchinson, and Leonard B Johnson. Space Shuttle Vehicle Automatic Docking Study. Technical report, National Aeronautics and Space Administration, 1971.

[60] Elaine M Hinman and David M Bushman. Soviet Automated Rendezvous and Docking System Overview. *Automated Rendezvous and Capture Review. Executive Summary*, pages 34–35, 1991.

[61] VS Syromyatnikov. Docking Devices for Soyuz-Type Spacecraft. *6th Aerospace Mechanical Symposium*, 1972.

[62] Justin McFatter, Karl Keiser, and Timothy Rupp. NASA Docking System Block 1: NASA's New Direct Electric Docking System Supporting ISS and Future Human Space Exploration. In *44th Aerospace Mechanisms Symposium*, page 471, 2018.

[63] John L Goodman. History of Space Shuttle Rendezvous and Proximity Operations. *Journal of Spacecraft and Rockets*, 43(5):944–959, 2006.

[64] Stephane Ruel, Tim Luu, and Andrew Berube. Space Shuttle Testing of the TriDAR 3D Rendezvous and Docking Sensor. *Journal of Field Robotics*, 29(4):535–553, 2012.

[65] Markus Wilde, Zarrin K Chua, and Andreas Fleischner. Effects of Multivantage Point Systems on the Teleoperation of Spacecraft Docking. *IEEE Transactions on Human-Machine Systems*, 44(2):200–210, 2014.

[66] Braven C Leung, Nicholas R Goeser, Larry A Miller, and Sonia Gonzalez. Validation of Electroadhesion as a Docking Method for Spacecraft and Satellite Servicing. In *2015 IEEE Aerospace Conference*, pages 1–8. IEEE, 2015.

[67] Sungwook Cho, Sungsik Huh, and David Hyunchul Shim. Visual Detection and Servoing for Automated Docking of Unmanned Spacecraft. *Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan*, 12(APISAT-2013):a107–a116, 2014.

[68] Didier Pinard, Stéphane Reynaud, Patrick Delpy, and Stein E Strandmoe. Accurate and Autonomous Navigation for the ATV. *Aerospace Science and Technology*, 11(6):490–498, 2007.

[69] Andrey A Boguslavsky, Victor V Sazonov, Sergey M Sokolov, AI Smirnov, and KU Saigiraev. Automatic Vision-Based Monitoring of the Spacecraft Atv Rendezvous/Separations With the International Space Station. In *ICINCO-RA (1)*, pages 284–291, 2007.

[70] Daero Lee and George Vukovich. Robust Adaptive Terminal Sliding Mode Control on SE (3) for Autonomous Spacecraft Rendezvous and Docking. *Nonlinear Dynamics*, 83(4):2263–2279, 2016.

[71] Isao Kawano, Masaaki Mokuno, Toru Kasai, and Takashi Suzuki. Result of Autonomous Rendezvous Docking Experiment of Engineering Test Satellite-Vii. *Journal of Spacecraft and Rockets*, 38(1):105–111, 2001.

[72] Alexander Kramer, Philip Bangert, and Klaus Schilling. UWE-4: First Electric Propulsion on a 1U CubeSat—In-Orbit Experiments and Characterization. *Aerospace*, 7(7):98, 2020.

[73] Elaine Petro, Amelia Bruno, Paulo Lozano, Louis E Perna, and Dakota Freeman. Characterization of the TILE Electrospray Emitters. In *AIAA Propulsion and Energy 2020 Forum*, page 3612, 2020.

[74] David Krejci, Alexander Reissner, Tony Schönherr, Bernhard Seifert, Zainab Saleem, and Ricardo Alejos. Recent Flight Data from IFM Nano Thrusters in a Low Earth Orbit. In *Proceedings of the 36th International Electric Propulsion Conference, Vienna, Austria*, pages 15–20, 2019.

[75] David Krejci, Alexander Reissner, and Dennis Weihrauch. IFM Nano Thruster Electric Space Propulsion: From First Cubesat Demonstration to the First 100 Thrusters Produced. *33rd Annual AIAA/USU Conference on Small Satellites, Logan UT*, 2019.

[76] Duc M Bui. Plume Characterization of Busek 600W Hall Thruster. Technical report, Graduate School of Air Force Institute of Technology Wright-Patterson AFB, Ohio, 2012.

[77] Michael Tsay and Daniel Courtney. All-Electric CubeSat Propulsion Technologies for Versatile Mission Applications. In *2019 Proc. 32nd Int. Symp. Space Technology and Science*, 2019.

[78] MR Natisin and HL Zamora. Performance of a Fully Conventionally Machined Liquid-Ion Electrospray Thruster Operated in PIR. In *Proceedings of the 36th International Electric Propulsion Conference, Vienna, Austria*, pages 9–12, 2019.

[79] MR Natisin, HL Zamora, WA McGehee, NI Arnold, ZA Holley, MR Holmes, and D Eckhardt. Fabrication and Characterization of a Fully Conventionally Machined, High-Performance Porous-Media Electrospray Thruster. *Journal of Micromechanics and Microengineering*, 30(11):115021, 2020.

[80] CHEN Chong, CHEN Maolin, and ZHOU Haohao. Characterization of an Ionic Liquid Electrospray Thruster with a Porous Ceramic Emitter. *Plasma Science and Technology*, 22(9):094009, 2020.

[81] Robert J Antypas and Joseph J Wang. Pure Ionic Electrospray Extractor Design Optimization. In *International Electric Propulsion Conference*, page 372, 2019.

[82] David Charles Woffinden and David Keith Geller. Navigating the Road to Autonomous Orbital Rendezvous. *Journal of Spacecraft and Rockets*, 44(4):898–909, 2007.

[83] JP Mayer and RP Parten. Development of the Gemini Operational Rendezvous Plan. *Journal of Spacecraft and Rockets*, 5(9):1023–1028, 1968.

[84] ID Boyd, RS Buenconsejo, D Piskorz, B Lal, KW Crane, and E De La Rosa Blanco. On-Orbit Manufacturing and Assembly of Spacecraft. *IDA Paper P-8335, Institute for Defense Analysis, Alexandria, VA*, 2017.

[85] Louis S Breger and Jonathan P How. Safe Trajectories for Autonomous Rendezvous of Spacecraft. *Journal of Guidance, Control, and Dynamics*, 31(5):1478–1489, 2008.

[86] John A Christian. Relative Navigation Using Only Intersatellite Range Measurements. *Journal of Spacecraft and Rockets*, 54(1):13–28, 2016.

[87] Cornelius J Dennehy and James R Carpenter. A Summary of the Rendezvous, Proximity Operations, Docking, and Undocking (RPODU) Lessons Learned From the Defense Advanced Research Project Agency (DARPA) Orbital Express (Oe) Demonstration System Mission. Technical report, National Aeronautics and Space Administration, 2011.

[88] David E Gaylor and Brent William Barbee. Algorithms for Safe Spacecraft Proximity Operations. In *AAS/AIAA Spaceflight Mechanics Meeting*, 2007.

[89] Daero Lee and Henry Pernicka. Optimal Control for Proximity Operations and Docking. *International Journal Aeronautical and Space Sciences*, 11(3):206–220, 2010.

[90] Ping Lu and Xinfu Liu. Autonomous Trajectory Planning for Rendezvous and Proximity Operations by Conic Optimization. *Journal of Guidance, Control, and Dynamics*, 36(2):375–389, 2013.

[91] Yazhong Luo, Jin Zhang, and Guojin Tang. Survey of Orbital Dynamics and Control of Space Rendezvous. *Chinese Journal of Aeronautics*, 27(1):1–11, 2014.

[92] Shawn B McCamish. Distributed Autonomous Control of Multiple Spacecraft During Close Proximity Operations. *Naval Postgraduate School*, 2007.

[93] John Edwin Miller. *Space Navigation, Guidance and Space Control*. Technivision, 1966.

[94] Michael R Phillips. Spacecraft Collision Probability Estimation for Rendezvous and Proximity Operations. *Utah State University*, 2012.

[95] David A Barnhart, Rahul Rughani, Jeremy J Allam, and Kyle W Clarke. Initial Safety Posture Investigations for Earth Regime Rendezvous and Proximity Operations. *Journal of Space Safety Engineering*, 7(4):519–524, 2020.

[96] D. Barnhart, L. Hill, M. Turnbull, and P. Will. Changing Satellite Morphology Through Cellularization. In *AIAA SPACE 2012 Conference & Exposition-5262*, 2012.

[97] Jeffery Scott Ginn. Spacecraft Formation Flight: Analysis of the Perturbed J2-Modified Hill-Clohessy-Wiltshire Equations. *University of Texas, Arlington – Masters Thesis*, 2007.

[98] Richard Courant and David Hilbert. *Methods of Mathematical Physics: Partial Differential Equations*. John Wiley & Sons, 2008.

[99] BD Tapley and Ch Reigber. The GRACE Mission: Status and Future Plans. *Agufm*, 2001:G41C–02, 2001.

[100] FG Lemoine, SC Kenyon, JK Factor, RG Trimmer, NK Pavlis, DS Chinn, CM Cox, SM Klosko, SB Luthcke, MH Torrence, et al. The Development of the Joint NASA GSFC and the National Imagery and Mapping Agency (NIMA) Geopotential Model EGM96. Technical report, National Aeronautics and Space Administration, 1998.

[101] Chia-Chun George Chao. *Applied Orbit Perturbation and Maintenance*. American Institute of Aeronautics and Astronautics, Inc., 2005.

[102] Kyle Alfriend, Srinivas Rao Vadali, Pini Gurfil, Jonathan How, and Louis Breger. *Spacecraft Formation Flying: Dynamics, Control and Navigation*, volume 2. Elsevier, 2009.

[103] JL Burch, TE Moore, RB Torbert, and BL Giles. Magnetospheric Multiscale Overview and Science Objectives. *Space Science Reviews*, 199(1-4):5–21, 2016.

[104] Sunny Leung and Oliver Montenbruck. Real-Time Navigation of Formation-Flying Spacecraft Using Global-Positioning-System Measurements. *Journal of Guidance, Control, and Dynamics*, 28(2):226–235, 2005.

[105] Daniel P Scharf, Fred Y Hadaegh, and Scott R Ploen. A Survey of Spacecraft Formation Flying Guidance and Control (Part I): Guidance. *IEEE American Control Conference*, 2003.

[106] Daniel P Scharf, Fred Y Hadaegh, and Scott R Ploen. A Survey of Spacecraft Formation Flying Guidance and Control. Part Ii: Control. In *Proceedings of the 2004 American control conference*, volume 4, pages 2976–2985. IEEE, 2004.

[107] Trevor W Williams, Neil Ottenstein, Mitra Farahmand, and Eric Palmer. Initial Satellite Formation Flight Results From the Magnetospheric Multiscale Mission. In *AIAA/AAS Astrodynamics Specialist Conference*, page 5505, 2016.

[108] Trevor Williams, Neil Ottenstein, Eric J Palmer, and Jacob Hollister. Results of the Apogee-Raising Campaign of the Magnetospheric Multiscale Mission. *AAS/AIAA Astrodynamics Specialist Conference, Stevenson WA*, 2017.

[109] Simone D'Amico, J-S Ardaens, and Robin Larsson. Spaceborne Autonomous Formation-Flying Experiment on the PRISMA Mission. *Journal of Guidance, Control, and Dynamics*, 35(3):834–850, 2012.

[110] John Bristow, David Folta, and Kate Hartman. A Formation Flying Technology Vision. In *Space 2000 Conference and Exposition*, page 5194, 2000.

[111] Chris Sabol, Rich Burns, and Craig A McLaughlin. Satellite Formation Flying Design and Evolution. *Journal of Spacecraft and Rockets*, 38(2):270–278, 2001.

[112] Mitra Farahmand, Anne Long, and Russell Carpenter. Magnetospheric Multiscale Mission Navigation Performance Using the Goddard Enhanced Onboard Navigation System. In *Proceedings of the 25th International Symposium on Space Flight Dynamics*, pages 19–23, 2015.

[113] Luke B Winternitz, William A Bamford, and Samuel R Price. New High-Altitude GPS Navigation Results From the Magnetospheric Multiscale Spacecraft and Simulations at Lunar Distances. In *Proceedings of the 30th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2017)*, pages 1114–1126, 2017.

[114] Oliver Montenbruck. Kinematic GPS Positioning of LEO Satellites Using Ionosphere-Free Single Frequency Measurements. *Aerospace Science and Technology*, 7(5):396–405, 2003.

[115] Simone D'Amico and Oliver Montenbruck. Proximity Operations of Formation-Flying Spacecraft Using an Eccentricity/Inclination Vector Separation. *Journal of Guidance, Control, and Dynamics*, 29(3):554–563, 2006.

[116] Zhongli Liu, Zupei Li, Benyuan Liu, Xinwen Fu, Ioannis Raptis, and Kui Ren. Rise of Mini-Drones: Applications and Issues. In *Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing*, pages 7–12, 2015.

[117] Alexander Chin, Roland Coelho, Ryan Nugent, Riki Munakata, and Jordi Puig-Suari. CubeSat: The Pico-Satellite Standard for Research and Education. In *AIAA Space 2008 Conference & Exposition*, page 7734, 2008.

[118] Armen Toorian, Ken Diaz, and Simon Lee. The Cubesat Approach to Space Access. In *2008 IEEE Aerospace Conference*, pages 1–14. IEEE, 2008.

[119] Jason Crusan and Carol Galica. NASA's CubeSat Launch Initiative: Enabling Broad Access to Space. *Acta Astronautica*, 157:51–60, 2019.

[120] Joon Wayn Cheong, Benjamin J Southwell, William Andrew, Elias Aboutanios, Chung Lam, Tom Croston, Luyang Li, Shannon Green, Alexander Kroh, Eamonn P Glennon, et al. A Robust Framework for Low-Cost Cubesat Scientific Missions. *Space Science Reviews*, 216(1):8, 2020.

[121] Rafael Fierro, Aveek K Das, Vijay Kumar, and James P Ostrowski. Hybrid Control of Formations of Robots. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 1, pages 157–162. IEEE, 2001.

[122] Aveek K Das, Rafael Fierro, Vijay Kumar, James P Ostrowski, John Spletzer, and Camillo J Taylor. A Vision-Based Formation Control Framework. *IEEE Transactions on Robotics and Automation*, 18(5):813–825, 2002.

[123] Wei Ren and Randal W Beard. *Distributed Consensus in Multi-Vehicle Cooperative Control*, volume 27. Springer, 2008.

[124] Yang Quan Chen and Zhongmin Wang. Formation Control: A Review and a New Consideration. In *2005 IEEE/RSJ International conference on intelligent robots and systems*, pages 3181–3186. IEEE, 2005.

[125] Jiangping Hu and Gang Feng. Distributed Tracking Control of Leader–Follower Multi-Agent Systems Under Noisy Measurement. *Automatica*, 46(8):1382–1387, 2010.

[126] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. Multirobot Systems: A Classification Focused on Coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028, 2004.

[127] Michael Defoort, Thierry Floquet, Annemarie Kokosy, and Wilfrid Perruquetti. Sliding-Mode Formation Control for Cooperative Autonomous Mobile Robots. *IEEE Transactions on Industrial Electronics*, 55(11):3944–3953, 2008.

[128] Steven M LaValle. *Planning Algorithms*. Cambridge university press, 2006.

[129] Jaydev P Desai, James P Ostrowski, and Vijay Kumar. Modeling and Control of Formations of Nonholonomic Mobile Robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 2001.

[130] Peng Song and Vijay Kumar. A Potential Field Based Approach to Multi-Robot Manipulation. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 1217–1222. IEEE, 2002.

[131] Tim D Barfoot and Christopher M Clark. Motion Planning for Formations of Mobile Robots. *Robotics and Autonomous Systems*, 46(2):65–78, 2004.

[132] Wolfgang Hönig, James A Preiss, TK Satish Kumar, Gaurav S Sukhatme, and Nora Ayanian. Trajectory Planning for Quadrotor Swarms. *IEEE Transactions on Robotics*, 34(4):856–869, 2018.

[133] James A Preiss, Wolfgang Honig, Gaurav S Sukhatme, and Nora Ayanian. Crazyswarm: A Large Nano-Quadcopter Swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3299–3304. IEEE, 2017.

[134] Nora Ayanian. Dart: Diversity-Enhanced Autonomy in Robot Teams. *The International Journal of Robotics Research*, 38(12-13):1329–1337, 2019.

[135] Jan Carlo Barca, A Sekercioglu, and Adam Ford. Controlling Formations of Robots With Graph Theory. In *Intelligent Autonomous Systems 12*, pages 563–574. Springer, 2013.

[136] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm Robotics: A Review From the Swarm Engineering Perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[137] Dervis Karaboga and Bahriye Akay. A Survey: Algorithms Simulating Bee Swarm Intelligence. *Artificial intelligence review*, 31(1-4):61, 2009.

[138] Mark L Winston. *The Biology of the Honey Bee*. Harvard University Press, 1991.

[139] Thomas D Seeley and Susannah C Buhrman. Group Decision Making in Swarms of Honey Bees. *Behavioral Ecology and Sociobiology*, 45(1):19–31, 1999.

[140] Salim Bitam, Mohamed Batouche, and El-ghazali Talbi. A Survey on Bee Colony Algorithms. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW)*, pages 1–8. IEEE, 2010.

[141] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. A Comprehensive Survey: Artificial Bee Colony (ABC) Algorithm and Applications. *Artificial Intelligence Review*, 42(1):21–57, 2014.

[142] Anguluri Rajasekhar, Nandar Lynn, Swagatam Das, and Ponnuthurai N Suganthan. Computing with the Collective Intelligence of Honey Bees–a Survey. *Swarm and Evolutionary Computation*, 32:25–48, 2017.

[143] Xin-She Yang. Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 317–323. Springer, 2005.

[144] Dušan Teodorovic and Mauro Dell'Orco. Bee Colony Optimization–a Cooperative Learning Approach to Complex Transportation problems. *Advanced OR and AI methods in transportation*, 51:60, 2005.

[145] Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation With Applications to Tracking and Navigation: Theory Algorithms and Software*. John Wiley & Sons, 2004.

[146] Mike West and Jeff Harrison. *Bayesian Forecasting and Dynamic Models*. Springer Science & Business Media, 2006.

[147] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[148] R.L. Haupt and S.E. Haupt. *Practical Genetic Algorithms*. Wiley InterScience electronic collection. Wiley, 2004.

[149] F Kenneth Chan et al. *Spacecraft Collision Probability*. Aerospace Press El Segundo, CA, 2008.

[150] Raymond W Holsapple. *A Modified Simple Shooting Method for Solving Two-Point Boundary Value Problems*. PhD thesis, Texas Tech University, 2003.

[151] Andrew J Abraham, David B Spencer, and Terry J Hart. Particle Swarm Optimization of Two-Maneuver, Impulsive Transfers From Leo to Lagrange Point Orbits via Shooting. *International Symposium on Space Flight Dynamics*, 2014.

[152] Diogene A Dei Tos and Francesco Topputo. Trajectory Refinement of Three-Body Orbits in the Real Solar System Model. *Advances in Space Research*, 59(8):2117–2132, 2017.

[153] Prashant R Patel. *Automating Interplanetary Trajectory Generation for Electric Propulsion Trade Studies*. PhD thesis, University of Michigan, 2008.

[154] Prashant R Patel, Alec D Gallimore, Thomas H Zurbuchen, and Daniel J Scheeres. An Algorithm for Generating Feasible Low Thrust Interplanetary Trajectories. In *Proceedings of the International Electric Propulsion Conference, Michigan, USA*, 2009.

[155] Jon Sims, Paul Finlayson, Edward Rinderle, Matthew Vavrina, and Theresa Kowalkowski. Implementation of a Low-Thrust Trajectory Optimization Algorithm for Preliminary Design. In *AIAA/AAS Astrodynamics specialist conference and exhibit*, page 6746, 2006.

[156] Fanghua Jiang, Hexi Baoyin, and Junfeng Li. Practical Techniques for Low-Thrust Trajectory Optimization With Homotopic Approach. *Journal of Guidance, Control, and Dynamics*, 35(1):245–258, 2012.

[157] Charles R Hargraves and Stephen W Paris. Direct Trajectory Optimization Using Nonlinear Programming and Collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

[158] Baolin Wu, Danwei Wang, Eng Kee Poh, and Guangyan Xu. Nonlinear Optimization of Low-Thrust Trajectory for Satellite Formation: Legendre Pseudospectral Approach. *Journal of Guidance, Control, and Dynamics*, 32(4):1371–1381, 2009.

[159] Yoonsoo Kim, Mehran Mesbahi, and Fred Y Hadaegh. Multiple-Spacecraft Reconfiguration Through Collision Avoidance, Bouncing, and Stalemate. *Journal of Optimization Theory and Applications*, 122(2):323–343, 2004.

[160] Thomas Uriot, Dario Izzo, Luis Simoes, Rasit Abay, Nils Einecke, Sven Rebhan, Jose Martinez-Heras, Francesca Letizia, Jan Siminski, and Klaus Merz. Spacecraft Collision Avoidance Challenge: Design and Results of a Machine Learning Competition. *arXiv Preprint arXiv:2008.03069*, 2020.

[161] Qinglei Hu, Hongyang Dong, Youmin Zhang, and Guangfu Ma. Tracking Control of Spacecraft Formation Flying With Collision Avoidance. *Aerospace Science and Technology*, 42:353–364, 2015.

[162] Ti Chen, Hao Wen, Haiyan Hu, and Dongping Jin. Output Consensus and Collision Avoidance of a Team of Flexible Spacecraft for on-Orbit Autonomous Assembly. *Acta Astronautica*, 121:271–281, 2016.

[163] Noelia Sánchez-Ortiz, Miguel Belló-Mora, and Heiner Klinkrad. Collision Avoidance Manoeuvres During Spacecraft Mission Lifetime: Risk Reduction and Required DV. *Advances in Space Research*, 38(9):2107–2116, 2006.

[164] Rune Schlanbusch, Raymond Kristiansen, and Per J Nicklasson. Spacecraft Formation Reconfiguration With Collision Avoidance. *Automatica*, 47(7):1443–1449, 2011.

[165] Arthur Richards, Tom Schouwenaars, Jonathan P How, and Eric Feron. Spacecraft Trajectory Planning With Avoidance Constraints Using Mixed-Integer Linear Programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[166] Daero Lee, Amit K Sanyal, and Eric A Butcher. Asymptotic Tracking Control for Spacecraft Formation Flying With Decentralized Collision Avoidance. *Journal of Guidance, Control, and Dynamics*, 38(4):587–600, 2015.

[167] Célia Martinie, Eric Barboni, David Navarre, Philippe Palanque, Racim Fahssi, Erwann Poupart, and Eliane Cubero-Castan. Multi-Models-Based Engineering of Collaborative Systems: Application to Collision Avoidance Operations for Spacecraft. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*, pages 85–94, 2014.

[168] Kwangwon Lee, Chandeok Park, and Sang-Young Park. Near-Optimal Guidance and Control for Spacecraft Collision Avoidance Maneuvers. In *AIAA/AAS Astrodynamics Specialist Conference*, page 4114, 2014.

[169] Sanny R Omar and Riccardo Bevilacqua. Spacecraft Collision Avoidance Using Aerodynamic Drag. *Journal of Guidance, Control, and Dynamics*, 43(3):567–573, 2020.

[170] Zheyao Xu, Yukun Chen, and Zhexuan Xu. Optimal Guidance and Collision Avoidance for Docking With the Rotating Target Spacecraft. *Advances in Space Research*, 63(10):3223–3234, 2019.

[171] James W Sales. Trajectory Optimization for Spacecraft Collision Avoidance. *Air Force Institute of Technology, Wright-Patterson Air Force Base – Thesis*, 2013.

[172] Zhang Rongzhi and Yang Kaizhong. *Spacecraft Collision Avoidance Technology.* Academic Press, 2020.

[173] Yi Wang, Yuzhu Bai, Dechao Ran, Qian Chen, Qing Ni, and Xiaoqian Chen. Dual-Equal-Collision-Probability-Curve Method for Spacecraft Safe Proximity Maneuvers in Presence of Complex Shape. *Acta Astronautica*, 159:65–76, 2019.

[174] David Berry. Automated Spacecraft Conjunction Assessment at Mars and the Moon. In *SpaceOps 2012*, page 1262983. International Astronautical Federation (IAF), 2012.

[175] Pini Gurfil and Konstantin V Kholshevnikov. Manifolds and Metrics in the Relative Spacecraft Motion Problem. *Journal of Guidance, Control, and Dynamics*, 29(4):1004–1010, 2006.

[176] Louis Breger and Jonathan P How. Safe Trajectories for Autonomous Rendezvous of Spacecraft. *Journal of Guidance, Control, and Dynamics*, 31(5):1478–1489, 2008.

[177] Avishai Weiss, Morgan Baldwin, Richard Scott Erwin, and Ilya Kolmanovsky. Model Predictive Control for Spacecraft Rendezvous and Docking: Strategies for Handling Constraints and Case Studies. *IEEE Transactions on Control Systems Technology*, 23(4):1638–1647, 2015.

[178] Christopher Jewison, R Scott Erwin, and Alvar Saenz-Otero. Model Predictive Control With Ellipsoid Obstacle Constraints for Spacecraft Rendezvous. *IFAC-PapersOnLine*, 48(9):257–262, 2015.

[179] Jacob Broida and Richard Linares. Spacecraft Rendezvous Guidance in Cluttered Environments via Reinforcement Learning. In *29th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–15, 2019.

[180] Xu Huang, Ye Yan, and Yang Zhou. Underactuated Spacecraft Formation Reconfiguration With Collision Avoidance. *Acta Astronautica*, 131:166–181, 2017.

[181] Wei Ren and Randal W Beard. Decentralized Scheme for Spacecraft Formation Flying via the Virtual Structure Approach. *Journal of Guidance, Control, and Dynamics*, 27(1):73–82, 2004.

[182] Yaohua Guo, Jun Zhou, and Yingying Liu. Distributed RISE Control for Spacecraft Formation Reconfiguration With Collision Avoidance. *Journal of the Franklin Institute*, 356(10):5332–5352, 2019.

[183] Xianghong Xue, Xiaokui Yue, and Jianping Yuan. Connectivity Preservation and Collision Avoidance Control for Spacecraft Formation Flying in the Presence of Multiple Obstacles. *Advances in Space Research*, 2020.

[184] Xiaohan Lin, Xiaoping Shi, and Shilun Li. Optimal Cooperative Control for Formation Flying Spacecraft With Collision Avoidance. *Science Progress*, 103(1):0036850419884432, 2020.

[185] Ridley Scott. *The Martian*. 20th Century Fox, 2015.

[186] A. Weir. *The Martian*. 2018 Great American Read. Broadway Books, 2015.

[187] [SillySlimes]. (2019, March 12). Yet Again Another Hermes Spacecraft From "The Martian" [Online Forum Post]. Retrieved from `https://www.reddit.com/r/SpaceflightSimulator/comments/b31mcb/yet_again_another_hermes_spacecraft_from_the/`.

[188] Donald Ruffatto, Dzenis Beganovic, Aaron Parness, and Matthew Spenko. Experimental Results of a Controllable Electrostatic/Gecko-Like Adhesive on Space Materials. In *2014 IEEE Aerospace Conference*, pages 1–7. IEEE, 2014.

[189] Stanford. Stanford Engineers Design a Robotic Gripper for Cleaning Up Space Debris. `https://news.stanford.edu/2017/06/28/engineers-design-robotic-gripper-cleaning-space-debris/`, 2017. [Online; accessed 30-September-2019].

[190] Jason L Forshaw, Guglielmo S Aglietti, Nimal Navarathinam, Haval Kadhem, Thierry Salmon, Aurélien Pisseloup, Eric Joffre, Thomas Chabot, Ingo Retat, Robert Axthelm, et al. RemoveDEBRIS: An in-Orbit Active Debris Removal Demonstration Mission. *Acta Astronautica*, 127:448–463, 2016.

[191] Sriram Narayanan, David Barnhart, Rebecca Rogers, Donald Ruffatto, Ethan Schaler, Nikko Van Crey, Gabriella Dean, Alisha Bhanji, Sofia Bernstein, Amrita Singh, et al. REACCH-Reactive Electro-Adhesive Capture Cloth Mechanism to Enable Safe Grapple of Cooperative/Non-Cooperative Space Debris. In *AIAA Scitech 2020 Forum*, page 2134, 2020.

[192] Lennon Rodgers, Simon Nolet, and David W Miller. Development of the Miniature Video Docking Sensor. In *Modeling, Simulation, and Verification of Space-based Systems III*, volume 6221, page 62210E. International Society for Optics and Photonics, 2006.

[193] A. Boesso and A. Francesconi. ARCADE Small-Scale Docking Mechanism for Micro-Satellites. *Acta Astronautica*, Vol. 86, 2013.

[194] Robotics Automation NASA Goddard Space Flight Center and Control Technology Transfer Program. Robotic Gripper for Satellite Capture and Servicing. *NASA Technology Transfer Program*, 2019.

[195] Larry Jay Friesen, Albert A Jackson IV, Herbert A Zook, and Donald J Kessler. Analysis of Orbital Perturbations Acting on Objects in Orbits Near Geosynchronous Earth Orbit. *Journal of Geophysical Research: Planets*, 97(E3):3845–3863, 1992.

[196] Najafi Alamdari. Perturbations in Orbital Elements of a Low Earth Orbiting Satellite. *Journal of the Earth and Space Physics, Vol 33*, 2005.

[197] Blaise Barney et al. Introduction to Parallel Computing. *Lawrence Livermore National Laboratory*, 6(13):10, 2010.

[198] Gene H Golub and James M Ortega. *Scientific Computing: An Introduction With Parallel Computing*. Elsevier, 2014.

[199] Richard P Martin, Amin M Vahdat, David E Culler, and Thomas E Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. *ACM SIGARCH Computer Architecture News*, 25(2):85–97, 1997.

[200] Robert C Durst, Patrick D Feighery, and Keith L Scott. Why Not Use the Standard Internet Suite for the Interplanetary Internet, 2000.

[201] Walt Truszkowski, Mike Hinchey, James Rash, and Christopher Rouff. NASA's Swarm Missions: The Challenge of Building Autonomous Software. *IT Professional*, 6(5):47–52, 2004.

[202] Dario Izzo, Marcus Märtens, and Binfeng Pan. A Survey on Artificial Intelligence Trends in Spacecraft Guidance Dynamics and Control. *Astrodynamics*, 3(4):287–299, 2019.

[203] Michael G Hinchey, Roy Sterritt, and Chris Rouff. Swarms and Swarm Intelligence. *Computer*, 40(4):111–113, 2007.

[204] Brian Drabble. Spacecraft Command and Control Using Artificial Intelligence Techniques. *Journal of the British Interplanetary Society*, 44:251–254, 1991.

[205] Douglas Bernard, Gregory Dorais, Edward Gamble, Bob Kanefsky, James Kurien, G Man, William Millar, Nicola Muscettola, Pandurang Nayak, Kanna Rajan, et al. Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment. *AIAA Space Technology Conference and Exposition, Albequerque NM*, 1999.

[206] Daniela Girimonte and Dario Izzo. Artificial Intelligence for Space Applications. In *Intelligent Computing Everywhere*, pages 235–253. Springer, 2007.

[207] Dieter Mehrholz, L Leushacke, W Flury, R Jehn, H Klinkrad, and M Landgraf. Detecting, Tracking and Imaging Space Debris. *ESA Bulletin(0376-4265)*, pages 128–134, 2002.

[208] Craig H Smith and Ben Greene. The EOS Space Debris Tracking System. In *AMOS Conf. Techn. Papers*, 2006.

[209] Ben Greene, Yuo Gao, Chris Moore, Y Wang, A Boiko, J Ritchie, J Sang, et al. Laser Tracking of Space Debris. In *13th International Workshop on Laser Ranging Instrumentation, Washington DC*, 2002.

[210] Francis Bennet, Celine D'Orgeville, Yue Gao, William Gardhouse, Nicolas Paulin, Ian Price, Francois Rigaut, Ian T Ritchie, Craig H Smith, Kristina Uhlendorf, et al. Adaptive Optics for Space Debris Tracking. In *Adaptive Optics Systems IV*, volume 9148, page 91481F. International Society for Optics and Photonics, 2014.

[211] Steven John Tingay, David L Kaplan, Benjamin McKinley, Franklin Briggs, Randall B Wayth, Natasha Hurley-Walker, J Kennewell, Craig Smith, Kefei Zhang, Wayne Arcus, et al. On the Detection and Tracking of Space Debris Using the Murchison Widefield Array. I. Simulations and Test Observations Demonstrate Feasibility. *The Astronomical Journal*, 146(4):103, 2013.

[212] Zhipeng Liang, Xue Dong, Makram Ibrahim, Qingli Song, Xingwei Han, Chengzhi Liu, Haitao Zhang, and Guohai Zhao. Tracking the Space Debris From the Changchun Observatory. *Astrophysics and Space Science*, 364(11):1–9, 2019.

[213] Georg Kirchner, Franz Koidl, Fabian Friederich, Ivo Buske, Uwe Völker, and Wolfgang Riede. Laser Measurements to Space Debris From Graz SLR Station. *Advances in Space Research*, 51(1):21–24, 2013.

[214] RM Goldstein, SJ Goldstein, and DJ Kessler. Radar Observations of Space Debris. *Planetary and Space Science*, 46(8):1007–1013, 1998.

[215] Xu Yang, Yiming Pi, Tong Liu, and Haijiang Wang. Three-Dimensional Imaging of Space Debris With Space-Based Terahertz Radar. *IEEE Sensors Journal*, 18(3):1063–1072, 2017.

[216] Magdalena Mendijur, Massimo Sciotti, and Piermario Besso. Management of Radar Resources for Space Debris Tracking. In *Sensors and Systems for Space Applications V*, volume 8385, page 83850Z. International Society for Optics and Photonics, 2012.

[217] Giacomo Muntoni, Giorgio Montisci, Tonino Pisanu, Pietro Andronico, and Giuseppe Valente. Crowded Space: A Review on Radar Measurements for Space Debris Monitoring and Tracking. *Applied Sciences*, 11(4):1364, 2021.

[218] Christy Collis. The Geostationary Orbit: A Critical Legal Geography of Space's Most Valuable Real Estate. *The Sociological Review*, 57(1_suppl):47–65, 2009.

[219] Anthony S Williams. Expected Position Error for an Onboard Satellite GPS Receiver. Technical report, Graduate School of Air Force Institute of Technology Wright-Patterson AFB, Ohio, 2015.

[220] Byoung-Sun Lee, Jeong-Sook Lee, and Kyu-Hong Choi. Analysis of a Station-Keeping Maneuver Strategy for Collocation of Three Geostationary Satellites. *Control Engineering Practice*, 7(9):1153 – 1161, 1999.

[221] Frederik J. de Bruijn, Stephan Theil, Daniel Choukroun, and Eberhard Gill. Collocation of Geostationary Satellites Using Convex Optimization. *Journal of Guidance, Control, and Dynamics*, 39(6):1303–1313, 2016.

[222] Nola Taylor Redd. Bringing Satellites Back From the Dead: Mission Extension Vehicles Give Defunct Spacecraft a New Lease on Life-[News]. *IEEE Spectrum*, 57(8):6–7, 2020.

[223] Rahul Rughani and David A Barnhart. Using Genetic Algorithms for Safe Swarm Trajectory Optimization. In *30th AIAA/AAS Space Flight Mechanics Meeting. Orlando Fl, USA*, 6-10 January 2020.

[224] MC Eckstein, CK Rajasingh, and P Blumer. Colocation Strategy and Collision Avoidance for the Geostationary Satellites at 19 Degrees West. In *International Symposium on Space Flight Dynamics*, volume 6, 1989.

[225] Adam W Koenig and Simone D'Amico. Robust and Safe N-Spacecraft Swarming in Perturbed Near-Circular Orbits. *Journal of Guidance, Control, and Dynamics*, 41(8):1643–1662, 2018.

[226] David Barnhart, Tim Barrett, Jeff Sachs, and Peter Will. Development and Operation of a Micro-Satellite Dynamic Test Facility for Distributed Flight Operations. In *AIAA SPACE 2009 Conference & Exposition*, page 6443, 2009.

[227] Yashwanth Kumar Nakka, Rebecca C Foust, Elena Sorina Lupu, David B Elliott, Irene S Crowell, Soon-Jo Chung, and Fred Y Hadaegh. A Six Degree-of-Freedom Spacecraft Dynamics Simulator for Formation Control Research. *18th AAS/AIAA Spaceflight Mechanics Meeting, Galveston TX*, 2018.

[228] Michael Smat, David Barnhart, Lizvette Villafaña, and Kirby Overman. Cellular Based Aggregated Satellite System: The Design and Architecture of a Three Degree of Freedom Near-Frictionless Testbed for Ground Validation of CubeSat Operations. *2020 SmallSat Conference, Logan, Utah (Virtual)*, 2020.

[229] Youngho Eun, Chandeok Park, and Sang-Young Park. Design and Development of Ground-Based 5-Dof Spacecraft Formation Flying Testbed. In *AIAA modeling and simulation technologies conference*, page 1668, 2016.

[230] Jana Lyn Schwartz. *The Distributed Spacecraft Attitude Control System Simulator: From Design Concept to Decentralized Control*. PhD thesis, Virginia Tech, 2004.

[231] David Miller, A Saenz-Otero, J Wertz, A Chen, G Berkowski, C Brodel, S Carlson, D Carpenter, S Chen, S Cheng, et al. SPHERES: A Testbed for Long Duration Satellite Formation Flying in Micro-Gravity Conditions. In *Proceedings of the AAS/AIAA space flight mechanics meeting*, volume 105, pages 167–179. Clearwater, Florida, January, 2000.

[232] Soon-Jo Chung, David W Miller, and Olivier L de Weck. ARGOS Testbed: Study of Multidisciplinary Challenges of Future Spaceborne Interferometric Arrays. *Optical Engineering*, 43(9):2156–2168, 2004.

[233] Brij N Agrawal and Richard Eric Rasmussen. Air-Bearing-Based Satellite Attitude Dynamics Simulator for Control Software Research and Development. In *Technologies for Synthetic Environments: Hardware-in-the-Loop Testing VI*, volume 4366, pages 204–214. International Society for Optics and Photonics, 2001.

[234] D Gallardo and R Bevilacqua. Six Degrees of Freedom Experimental Platform for Testing Autonomous Satellites Operations. In *Proceedings of the 8th International ESA GNC Conference*, 2011.

[235] K Saulnier, D Pérez, RC Huang, D Gallardo, G Tilton, and R Bevilacqua. A Six-Degree-of-Freedom Hardware-in-the-Loop Simulator for Small Spacecraft. *Acta Astronautica*, 105(2):444–462, 2014.

[236] Daniel P Scharf, Jason A Keim, and Fred Y Hadaegh. Flight-Like Ground Demonstrations of Precision Maneuvers for Spacecraft Formations—Part II. *IEEE Systems Journal*, 4(1):96–106, 2010.

[237] Martin W Regehr, Ahmet B Acikmese, Asif Ahmed, M Aung, KC Clark, P MacNeal, J Shields, G Singh, R Bailey, C Bushnell, et al. The Formation Control Testbed. In *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No. 04TH8720)*, volume 1, pages 557–564. IEEE, 2004.

[238] Richard Zappulla, Josep Virgili-Llop, Costantinos Zagaris, Hyeongjun Park, and Marcello Romano. Dynamic Air-Bearing Hardware-in-the-Loop Testbed to Experimentally Evaluate Autonomous Spacecraft Proximity Maneuvers. *Journal of Spacecraft and Rockets*, 54(4):825–839, 2017.

[239] Panagiotis Tsiotras. ASTROS: A 5DOF Experimental Facility for Research in Space Proximity Operations. In *AAS Guidance and Control Conference*, volume 2014-114, 2014.

[240] Daniele Gallardo, Riccardo Bevilacqua, and Richard Rasmussen. Advances on a 6 Degrees of Freedom Testbed for Autonomous Satellites Operations. In *AIAA Guidance, Navigation, and Control Conference*, page 6591, 2011.

[241] Markus Wilde, Brian Kaplinger, Tiauw Go, Hector Gutierrez, and Daniel Kirk. ORION: A Simulation Environment for Spacecraft Formation Flight, Capture, and Orbital Robotics. In *2016 IEEE Aerospace Conference*, pages 1–14. IEEE, 2016.

[242] David C Sternberg, Christopher Pong, Nuno Filipe, Swati Mohan, Shawn Johnson, and Laura Jones-Wilson. Jet Propulsion Laboratory Small Satellite Dynamics Testbed Simulation: On-Orbit Performance Model Validation. *Journal of Spacecraft and Rockets*, 55(2):322–334, 2018.

[243] Sasi Prabhakaran Viswanathan, Amit Sanyal, and Lee Holguin. Dynamics and Control of a Six Degrees of Freedom Ground Simulator for Autonomous Rendezvous and Proximity Operation of Spacecraft. In *AIAA guidance, navigation, and control conference*, page 4926, 2012.

[244] Markus Schlotterer, Eviatar Edlerman, Federico Fumenti, Pini Gurfil, Stephan Theil, and Hao Zhang. On-Ground Testing of Autonomous Guidance for Multiple Satellites in a Cluster. In *Proceedings of the 8th International Workshop on Satellite Constellations and Formation Flying*, volume 6, 2015.

[245] Soon-Jo Chung and David W Miller. Propellant-Free Control of Tethered Formation Flight, Part 1: Linear Control and Experimentation. *Journal of Guidance, Control, and Dynamics*, 31(3):571–584, 2008.

[246] Jana L Schwartz, Mason A Peck, and Christopher D Hall. Historical Review of Air-Bearing Spacecraft Simulators. *Journal of Guidance, Control, and Dynamics*, 26(4):513–522, 2003.

[247] Zero Gravity Corporation. ZERO-G Research Programs. `https://www.gozerog.com/index.cfm?fuseaction=Research_Programs.welcome`, 2017. [Online; accessed 7-October-2019].

[248] NASA. Space Station Research Experiments. `https://www.nasa.gov/mission_pages/station/research/experiments_category`, 2018. [Online; accessed 7-October-2019].

[249] Alexis Pospischil. Multihop Routing of Telemetry Data in Drone Swarms. In *Proceedings of the International Telemetering Conference (ITC)*. International Foundation for Telemetering, 2017.

[250] Gábor Vásárhelyi, Csaba Virágh, Gergő Somorjai, Tamás Nepusz, Agoston E Eiben, and Tamás Vicsek. Optimized Flocking of Autonomous Drones in Confined Environments. *Science Robotics*, 3(20), 2018.

[251] Junyan Hu and Alexander Lanzon. An Innovative Tri-Rotor Drone and Associated Distributed Aerial Drone Swarm Control. *Robotics and Autonomous Systems*, 103:162–174, 2018.

[252] Preston Kemp, Silvan Li, Moradeke Olumogba, Derek Watson, Rebecca Bevers, and Aaron Spak. Creating a Software Framework to Choreograph Performances With Quadcopters. *Georgia Tech – Student Capstone Project*, 2019.

[253] Axel Bürkle, Florian Segor, and Matthias Kollmann. Towards Autonomous Micro Uav Swarms. *Journal of Intelligent & Robotic Systems*, 61(1-4):339–353, 2011.

[254] Grzegorz Chmaj and Henry Selvaraj. Distributed Processing Applications for UAV/drones: A Survey. In *Progress in Systems Engineering*, pages 449–454. Springer, 2015.

[255] Jawad Naveed Yasin, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. Formation Maintenance and Collision Avoidance in a Swarm of Drones. In *Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control*, pages 1–6, 2019.

[256] Bo Ryu, Nadeesha Ranasinghe, Wei-Min Shen, Kurt Turck, and Michael Muccio. BioAIR: Bio-Inspired Airborne Infrastructure Reconfiguration. In *MILCOM 2015-2015 IEEE Military Communications Conference*, pages 774–779. IEEE, 2015.

[257] Wei-Min Shen, Peter Will, Aram Galstyan, and Cheng-Ming Chuong. Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms. *Autonomous Robots*, 17(1):93–105, 2004.

[258] Domenico Cucinotta and Maurizio Vanelli. WHO Declares COVID-19 a Pandemic. *Acta Bio-Medica: Atenei Parmensis*, 91(1):157–160, 2020.

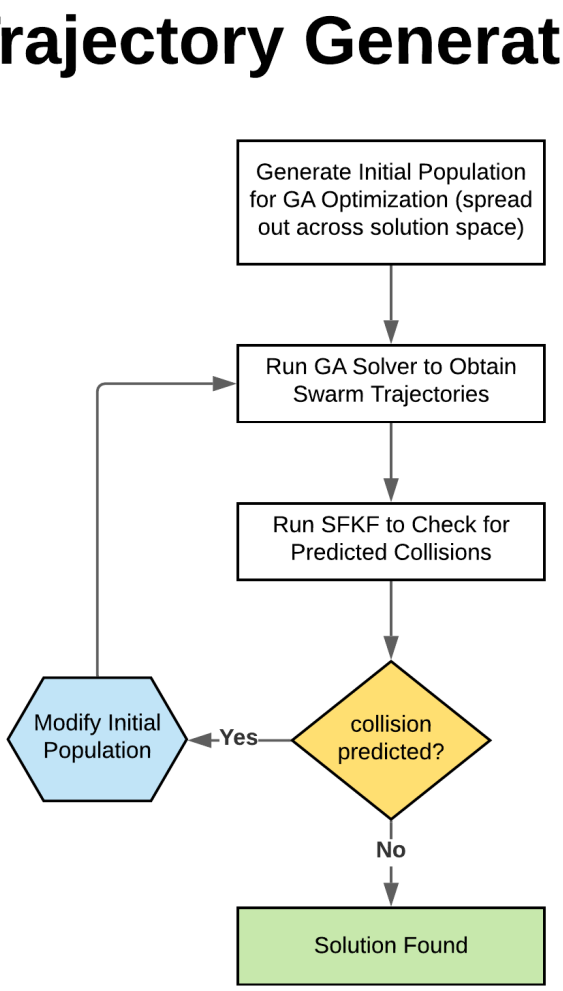
[259] M Bishr Omary, Jeetendra Eswaraka, S David Kimball, Prabhas V Moghe, Reynold A Panettieri, Kathleen W Scotto, et al. The COVID-19 Pandemic and Research Shutdown: Staying Safe and Productive. *The Journal of Clinical Investigation*, 130(6), 2020.

Appendix A

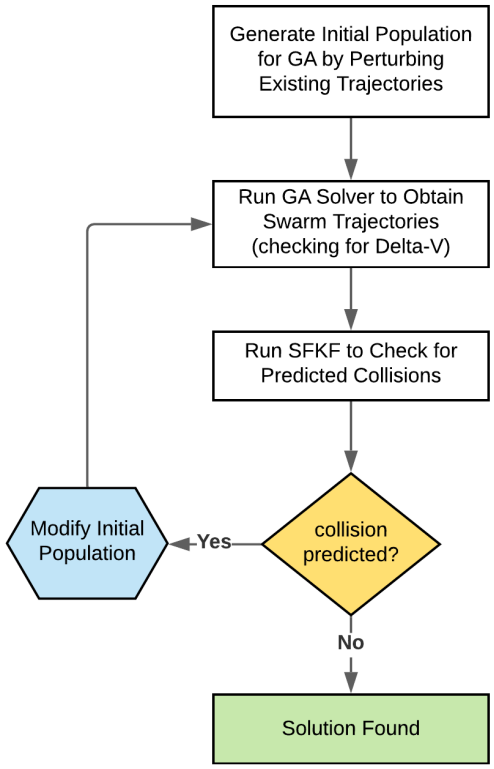
Computational Processes

This appendix contains block-diagram format descriptions of how the computations for the swarm trajectory generation and maintenance are done, from a top level perspective.

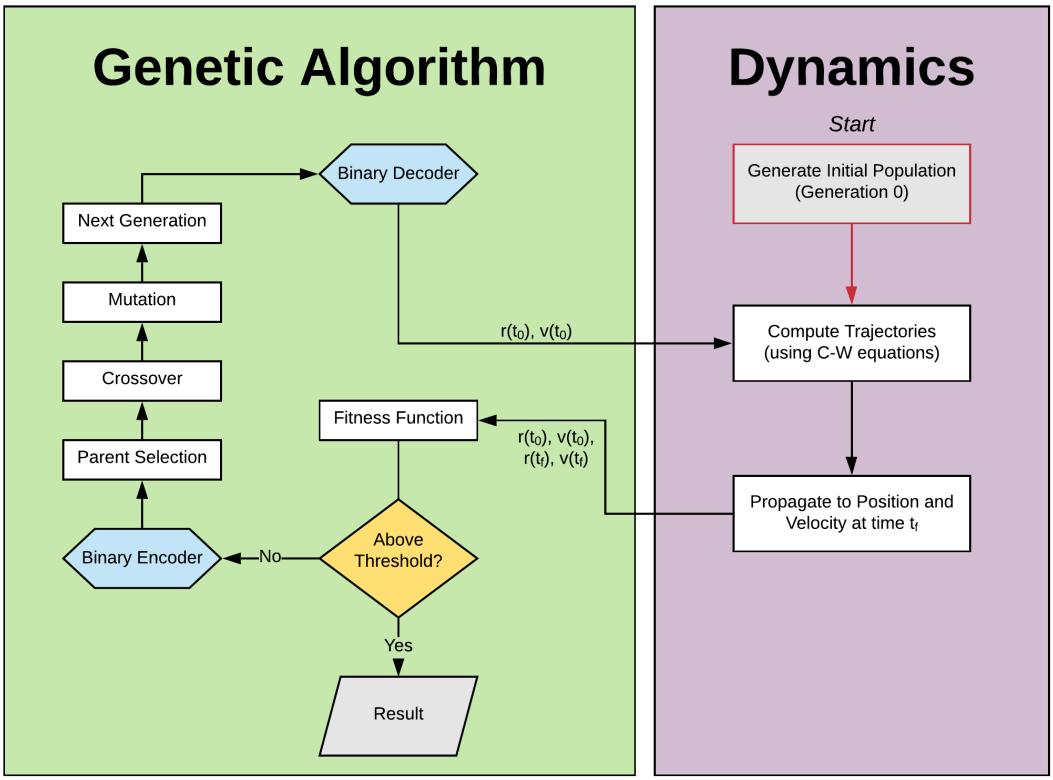
A.1 Swarm Generation



A.2 Swarm Modification



A.3 Genetic Algorithms



The following Python files implement the process shown in the above flowgraph for a set of five spacecraft.

Listing A.1: SwarmOptimization_Smart.py

```
""#Swarm Genetic Algorithms

#  696679..6 f7566696e647 468697349.      .4f5561.
# 626'     'f7 '474'      '6 'c65    '6f6.   677'   '686
# 9736b.        6579          |||    .d88' 888
#  '"Y8888o.    888ooooo8      888ooo88P'  888
#       '"Y88b  888     "      888'88b.    888
# oo      .d8P  888        o   888  '88b.  '88b    ooo
# 8""88888P'  o888ooooooood8 o888o  o888o  'Y8bood8P'



# (C) 2020, Rahul Rughani

# Created: May 29, 2020
# Converted from existing MATLAB code


# This code uses nested Genetic Algorithms to solve for a swarm set of N
# spacecraft in relative orbits around a common target point, while
# optimizing to prevent conjunctions between the spacecraft. Accounts for
# min and max ranges for all spacecraft from the target, and parameters
# for conjunction criteria.

```

```
25
26
27
28  ###############
29  # INITIALIZE #
30  ###############
31
32  # standard libraries
33  import numpy as np
34  import time
35  import scipy
36  import julian
37  import datetime
38  import pandas as pd
39  import os
40  import math
41  import warnings
42  import random
43
44  # custom functions
45  import OrbitFuncs as orb
46  import GAFuncs as GA
47  import RoatationalKinematics as rot
48
49
50  # define Figure and Data directories
51  currentDir = os.path.dirname(os.path.abspath(__file__)) # get current path
52  figDir = os.path.join(currentDir, 'Figures/')
53  datDir = os.path.join(currentDir, 'Data/')
54
55  # define constants
56  Re = 6378.14    # mean equatorial radius of Earth [km]
57  mu = 398601.2   # gravitational parameter - Earth [km^3/s^2]
58
59  # define target orbit parameters
60  h = 600     # altitude [km]
61
62  a = Re+h    # semi-major axis [km]
63  e = 0       # eccentricity [-]
64  inc = 0     # inclination [deg]
65  RAAN = 0    # right ascension [deg]
66  omega = 0   # arg. of perigee [deg]
67  theta = 0   # true anomaly [deg]
68
69  # get initial state vector
70  x = orb.COE2Cartesian(a, e, inc, RAAN, omega, theta, mu)
71
72  # decompose state vector
73  r0 = x[0:3]
74  v0 = x[3:6]
75
76  T = 2*math.pi*math.sqrt(a**3/mu)    # orbital period [sec]
77  n = 2*math.pi/T
78
79
80  # define population parameters
81  nSats = 5       # number of members in the swarm
82  norb = 1        # number of orbits to generate trajetories
83  dt = norb*T     # chaser orbit period in rel. frame [sec]
84  dLim = [[8,10], # set min and max distances allowable from
85          [8,10], # target for each spacecraft in swarm [km]
86          [8,10],
87          [0.5,2],
88          [3 5]]
89
90
91  # take into account case where a single set of min/max distances are applied to the
92  # entire swarm
93  if len(dLim) == 1:
94      dLim = [dLim]*nSats
95
96
97  # set bounds for random dv0 magnitudes
98  dv_min = 0          # [m/s]
99  dv_max = 10         # [m/s]
100
```

```
# set conjunction parameters
dist_coll = 0.02     # close approach distance qualifying as a collision between two
                     # spacecraft [km]. Note that this is after taking into account
                     # covariance standard deviations from sensor measurement errors

# time to propagate out to check for collisions
ndays = math.ceil(norb*T/86400)
dt_conj = 86400*ndays

res_conj = norb*1500     # discretization resolution to use for conjunction analysis


# store all data relevant to the swarm's orbital state in
swarm_data = GA.SwarmData(a,e,inc,RAAN,omega,theta)


########################
# DEFINE GA PARAMETERS #
########################

# parameters for conjunction optimization
npop = 20        # population size (must be divisible by 4)
ngen = 50        # max number of generations
nkeep = 2        # number of population members to not mutate (best)
pcross = 1       # probability of crossover
pmut = 1/npop/5 # probability of mutation
tol = 0.01       # convergence tolerance
cconj = 3        # weighting factor for conjunction

# parameters for individual spacecraft optimization
npop_sc = 100    # population size (must be divisible by 4)
ngen_sc = 50     # max number of generations
nkeep_sc = 2     # number of population members to not mutate (best)
pcross_sc = 1    # probability of crossover
pmut_sc = 0.002 # probability of mutation
tol_sc = 0.01    # convergence tolerance
cr = 1           # weighting factor for position
cv = 1           # weighting factor for velocity
cd = 5           # weighting factor for distance

# combine sc parameters into a single structure
params_sc = GA.Params(npop,ngen,nkeep,pcross,pmut,tol,cr,cv,cd)


nbins_sc = 6
nvars_sc = 12

nvars = 6*nSats # total number of variables used in the GA simulation

# set estimated values for position and velocity offsets from orbital insertion
r0_insert = [0 -20 0]*1e3    # [m]
v0_insert = [0 0 0]          # [m/s]

# set max limit for transition time to insert into optimized trajectories
dt_max = 30000       # [sec]

# define plot properties
res = norb*100       # plot points per segment
ext = 10             # factor to use for extended time plots
convView = false     # show convergence plots [bool]

# define properties of target satellite for visualization
scale = 10           # scale to use on plot
rad = 3/1e3          # spherical bus radius [km]
panel_diam = 40/1e3 # solar panel extension diameter [km]
panel_width = 2/1e3 # solar panel width [km]

# input QA [replace with exceptions instead of warnings in the future]
if npop % 4 > 0:
    warnings.warn('npop must be divisible by 4')

if npop_sc % 4 > 0:
    warnings.warn('npop_sc must be divisible by 4')


################################
```

```
# DEFINE SPACECRAFT PROPERTIES #
################################

# combining the range and angular measurement accuracies, an ellipsoid can  be formed,
# defined by three entries in a diagonal covariance matrix
rng_acc = 0.05   # range measurement accuracy (in LOS direction) [-]
ang_acc = 0.5    # az/el measurement accuracy (offset from LOS)  [deg]
spd_acc = 0.01   # speed measurement accuracy (in LOS direction) [-]

# package sensor accuracies into one array for convenient message passing
sens_acc = GA.SensAcc(rng_acc, ang_acc, spd_acc)



############################
# BUILD INITIAL POPULATION #
############################

print('Building Initial Population for %u SC' % nSats)

popRnd = []
rhat = np.array([1,0,0])     # initial position unit vector to randomize (LVLH frame)

for j in range(nSats):
    print('* Building SC#%u Pop' % j)

    vhat = rhat

    dr0 = []
    dv0 = []

    for i in range(npop_sc):
        dr0_mag = random.uniform(*dLim[j])
        rotAlp = random.uniform(0,2*math.pi)
        rotBet = random.uniform(0,2*math.pi/nSats) + 2*math.pi*j/nSats

        rhatRot = rot.rotSTD(rotBet,'x')*rot.rotSTD(rotAlp,'z')*np.reshape(rhat, (3,1))
        rhatRot = np.array(rhatRot)

        dr0.append( dr0_mag*rhatRot.flatten() )

        dv0_mag = random.uniform(dv_min,dv_max)
        rotTh = random.uniform(0,2*math.pi)
        rotPhi = random.uniform(0,2*math.pi)

        vhatRot = rot.rotSTD(rotPhi,'x')*rot.rotSTD(rotTh,'z')*np.reshape(vhat, (3,1))
        vhatRot = np.array(vhatRot)

        dv0.append( dv0_mag*vhatRot.flatten() )


    # propagate the initial population
    drf = []
    dvf = []
    for i in range(npop_sc):
        dr,dv = orb.cwEqn(dr0[i]*1e3, dv0[i], n, dt)

        drf.append(dr/1e3)  # convert distances back to km
        dvf.append(dv)


    popRnd.append(GA.Pop(dr0,dv0,drf,dvf))


##############################
# OPTIMIZE INITIAL POPULATION #
##############################

print('Optimizing Swarm Orbits')

tic = time.time()

# allocate variables to store initial pos/vel results
r = []
v = []
fittest_sc = []
```

```
gen_sc = []

popInit = []
for i in range(nSats):

    error = True

    # use while loop and try/catch to account for rare case where no GA solution is
    # found. In this case, try again
    while True
        try:
            result = GA.relOrbitOptimize(GA.fitnessGA, popRnd, dt, n, dLim, params_sc)
            r.append(result[0])
            v.append(result[1])
            popInit.append(result[2])
            fittest_sc.append(result[3])
            gen_sc.append(result[4])

            break
        except:
            pass


toc = time.time()
```

Listing A.2: GAFuncs.py

```
"""#Genetic Algorithm Functions

# .oooooo..o oooooooooooo ooooooooo.     .oooooo.
# d8P'    `Y8 `888'     `8 `888   `Y88.  d8P'  `Y8b
# Y88bo.       888          888   .d88' 888
#  `"Y8888o.   888oooo8     888ooo88P'  888
#      `"Y88b  888    "     888`88b.    888
# oo     .d8P  888       o  888  `88b.  `88b    ooo
# 8""88888P'  o888ooooood8 o888o  o888o  `Y8bood8P'

# (C) 2020, Rahul Rughani

# Created: May 29, 2020


import numpy as np
import time
import scipy
import warnings
import random

import OrbitFuncs as orb

# define classes to store data properties for Genetic Algorithm operations
class Axis: # data structure to store axes for a coordinate frame
    def __init__(self):
        self.X = np.array(np.zeros(3))  # x-axis unit vector
        self.Y = np.array(np.zeros(3))  # y-axis unit vector
        self.Z = np.array(np.zeros(3))  # z-axis unit vector

class SwarmData: # data structure to store swarm initial properties
    def __init__(self,nSats,a,e,i,RAAN,omega,M):
        self.nSats = nSats
        self.a = a
        self.e = e
        self.i = i
        self.RAAN = RAAN
        self.w = omega
        self.M = M

class Pop:
    def __init__(self,r0,v0,rf,vf):
        self.r0 = r0
        self.v0 = v0
        self.rf = rf
        self.vf = vf

    def sort(self, index):
        self.r0 = [self.r0[i] for i in index]
        self.v0 = [self.v0[i] for i in index]
        self.rf = [self.rf[i] for i in index]
        self.vf = [self.vf[i] for i in index]


class Coeff:
    def __init__(self,cr,cv,cd):
        self.cr = cr
        self.cv = cv
        self.cd = cd

class Params: # data structure to store genetic algorithm parameters
    def __init__(self,npop,ngen,nkeep,pcross,pmut,tol,cr,cv,cd):
        self.npop = npop
        self.ngen = ngen
        self.nkeep = nkeep
        self.pcross = pcross
        self.pmut = pmut
        self.tol = tol
        self.C = Coeff(cr,cv,cd)

class SensAcc:
    def __init__(self, rng, ang, spd):
        self.rng = rng
        self.ang = ang
```

```
75          self.spd = spd
76
77
78  def dExtrema(r0, v0, n, T, res):
79      # compute the minimum and maximum distances away from the target satellite
80      # attained by the chaser members in the population (npop>=1)
81
82      dist = []
83
84      for i in np.arange(0,T,T/res):
85          temp = orb.cwEqn(r0*1e3, v0, n, i)
86          dist.append( np.linalg.norm(temp/1e3, axis=1) )
87
88      dMin = min(dist)
89      dMax = max(dist)
90
91      return dMin, dMax
92
93  # encodes a decimal number to a gene for use in a genetic algoirthm optimization
94  # process. The first bit will determine sign - 1st bit  0: positive
95  #                                                       1: negative
96  def binEnc(num, nDec, bits):
97      #num:   decimal number to be encoded (can be float)
98      #nDec:  number of decimal places to keep in the encoding (shifted)
99      #bits:  number of bits to encode to
100
101     ## INSERT ERROR HANDLING HERE ##
102
103     shifted = num * 10**nDec
104     rounded = np.around(shifted)
105
106
107     if any( rounded > 2**(bits-1)-1):
108         ##### THROW ERROR SINCE NUMBER IS TOO LARGE TO ENCODE TO BIT SIZE ######
109
110     sign = np.tile("0", (len(num),1))
111
112     for i in range(len(num)):
113         if num[i] < 0:
114             sign[i] = '1'
115
116     abs = np.absolute(rounded)
117
118     bin = np.array( [list( format(abs[i], '0{}b'.format(bits-1))) for i in range(len(num
        ))] )
119
120     return np.hstack((sign,bin))
121
122 def binDec(bin, nDec):
123     #bin:   binary string array to be decoded (can be non-integer)
124     #nDec:  number of decimal places the encoded number has (shifted)
125
126     # check sign
127     sign = 1
128     if bin[i] == '1':
129         sign = -1
130
131
132     converted = int(bin[1:-1], 2)
133     shifted = converted / 10**nDec
134     signed = shifted * sign
135
136     return signed
137
138
139
140 def mate(chrom1, chrom2, nCross=1):
141     #chrom1:    chromosome of first parent (char array)
142     #chrom2:    chromosome of second parent (char array)
143     #nCross:    number of crossover points to use (default 1)
144
145     # determine length of chromosomes
146     lchrom = len(chrom1)
147     lchrom_div = math.floor(lchrom/nCross)
148
149     crossPt = np.array(np.zeros(nCross,1))
```

```
    for i in range(nCross):
        crossPt[i] = math.ceil(math.random()*(lchrom_div-1))+lchrom_div*i

    kid1 = []
    kid2 = []
    for i in range(nCross):

        ma = []
        pa = []
        if i % 2:
            ma = chrom1
            pa = chrom2
        else:
            ma = chrom2
            pa = chrom1

        frm = []
        if i==0:
            frm = 0
        else:
            frm = crossPt(i)

        kid1.append(ma[frm:crossPt[i]])
        kid2.append(pa[frm:crossPt[i]])


    temp = ma
    ma = pa
    pa = temp

    kid1.append(ma[crossPt[-1]+1:-1])
    kid2.append(pa[crossPt[-1]+1:-1])

    return kid1,kid2

def mutate(chrom, p):
    #chrom: chromosome to mutate (char array)
    #p:     probability of mutation (fraction)

    # determine how many nucleotides need to be mutated
    nmut = math.ceil(len(chrom)*p)

    # find number of rows and columns in input matrix
    nrow = np.size(chrom,0)
    ncol = np.size(chrom,1)

    # determine randomly which rows and columns to mutate
    mrow = np.ceil(np.random.rand(nmut,1)*nrow)
    mcol = np.ceil(np.random.rand(nmut,1)*ncol)

    # perform mutation
    mutated = chrom
    for i in range(nmut):
        if mutated[mrow[i],mcol[i]] == '1':
            mutated[mrow[i],mcol[i]] = '0'
        else:
            mutated[mrow[i],mcol[i]] = '1'

    return mutated


def fitnessGA(dr, dv, C, dRng, dLim):


    # decompose distance range avlues
    dMin = dRng[0]
    dMax = dRng[1]

    dSign = np.array(np.zeros(len(dMax),1))

    for i in range(len(dMax)):
        if dMin[i] >= dLim[0] && dMax[i] <= dLim[1]:
            dSign[i] = 0
        else:
            if dMin[i] < dLim[0]:
                dSign[i] = dLim[0] - dMin[i]
```

```
226
227             if dMax[i] > dLim[1]:
228                 dSign[i] = dSign[i] + dMax[i] - dLim[1]
229
230
231     return 1./(1 + C.cr*np.linalg.norm(dr,1) + C.cv*np.linalg.norm(dv,1) + C.cd*dSign)
232
233
234 def relOrbitOptimize(fitFun, pop, dt, n, dLim, params):
235
236     nbins = 6   # number of individual variables for the binary encoded optimization
237                 # process. Thereare 3 position and 3 velocity input floats. The 6
238                 # output floats are not binary encoded and thus not included here.
239
240     # define plot properties
241     res = 100
242
243     ##############################
244     # CREATE INITIAL POPULATION #
245     ##############################
246
247     dr0 = pop.r0
248     dv0 = pop.v0
249     drf = pop.rf
250     dvf = pop.vf
251
252     # compute max and min distances from target for each population member
253     # across its orbit
254     dMin,dMax = dExtrema(dr0, dv0, n, dt, res)
255
256     ####################################
257     # COMPUTE BASELINE FITNESS FUNCTION #
258     ####################################
259
260     fitArray = fitFun(dr0-drf, dv0-dvf, params.C, [dMin,dMax], dLim)
261     fittest = max(fitArray)
262
263     # sort fit results
264     fitIndex = np.argsort(fitArray)[::-1]
265     fitArray = fitArray[fitIndex]
266
267     pop.sort(fitIndex)
268
269     #0100100110101010011001001011100010010100010000 1001#
270     # CONVERT INITIAL POPULATION TO BINARY CHROMOSOMES #
271     #100110100101111101010000100010001010010110010######
272
273     # vars :: 6 position 6 velocity
274     # --> [dr0_x dr0_y dr0_z drf_x drf_y drf_z dv0_x dv0_y dv0_z ...
275     #       dvf_x dvf_y dvf_z]
276
277     bits = [16, 16, 16, 16, 16, 16] # num bits for each variable
278     dec =  [ 3,  3,  3,  3,  3,  3] # num decimal places for each variable
279
280     # allocate blank char array for genome storage
281     popChrom = np.tile("0", (npop, sum(bits) ))
282
283     popArray = np.hstack( (np.array(dr0), np.array(dv0)) )
284
285     for i in range(nbins):
286         frm = sum(bits[0:i+1]) - bits[i]
287         to = sum(bits[0:i+1])
288
289         popChrom[:,frm:to] = binEnc(popArray[:,i], dec[i], bits[i])
290
291
292
293     #########################
294     # RUN EVOLUTION PROCESS #
295     #########################
296
297     count = 0
298
299     while fittest < (1-params.tol) && count <= params.ngen:
300
301         # increment counter
```

```
302             count += 1
303
304             # determine population members to mate (use roulette method)
305             weight = fitArray[1:npop/2] / sum(fitArray[1:npop/2])
306             pairs = [np.random.choice(range(1,npop/2),npop/4,weight),np.random.choice(range
        (1,npop/2),npop/4,weight)]
307
308             # mate population
309             for i in range(pairs):
310                 # only mate at given probability
311                 if random.random() <= params.pcross:
312                     kids = mate(popChrom[pairs[i,1],:],popChrom[pairs[i,2],:])
313                     popChrom[len(popChrom)/2+2*i-1,:] = kids[0]
314                     popChrom[len(popChrom)/2+2*i,:] = kids[1]
315                 else:
316                     popChrom[len(popChrom)/2+2*i-1,:] = popChrom[pairs[i,1],:]
317                     popChrom[len(popChrom)/2+2*i,:] = popChrom[pairs[i,2],:]
318
319
320             # mutate population (except for top two results)
321             popChrom[params.nkeep:-1] = mutate(popChrom[params.nkeep:-1],params.pmut)
322
323
324             # decode chromosomes
325             for i in range(nbins):
326                 frm = sum(bits[0:i+1]) - bits[i]
327                 to = sum(bits[0:i+1])
328
329                 popArray[:,i] = binDec(popChrom[:,frm:to],dec[i])
330
331
332             # propagate to update final pos/vel
333             dr0 = []
334             dv0 = []
335             drf = []
336             dvf = []
337             for i in range(params.npop):
338                 dr0.append(popArray[i,0:2])
339                 dv0.append(popArray[i,3:5])
340                 dr,dv = orb.cwEqn(dr0[i]*1e3, dv0[i], n, dt)
341
342                 drf.append(dr/1e3)  # convert distances back to km
343                 dvf.append(dv)
344
345             # compute updated fitness values
346             dMin,dMax = dExtrema(dr0, dv0, n, dt, res)
347             fitArray = fitFun(dr0-drf, dv0-dvf, params.C, [dMin,dMax], dLim)
348             fittest = max(fitArray)
349
350             # sort fit results
351             fitIndex = np.argsort(fitArray)[::-1]
352             fitArray = fitArray[fitIndex]
353
354             pop.r0 = dr0
355             pop.v0 = dv0
356             pop.rf = drf
357             pop.vf = dvf
358
359             pop.sort(fitIndex)
360
361         ########################################
362         # CHECK PROPERTIES OF FINAL GENERATION #
363         ########################################
364
365         pop.sort(fitIndex)
366
367         return pop.r0, pop.v0, pop, fittest, count
```

Listing A.3: OrbitFuncs.py

```
"""#Orbital Mechanics Functions

# (C) 2020, Rahul Rughani

# Created: May 13, 2020


import numpy as np
import scipy
import warnings
import math



# solve Kepler's equation for eccentric anomaly (elliptical orbit)
def Kepler(M, e):
    #M: mean anomaly [rad]
    #e: eccentricity [-]. Must be less than 1 (elliptical orbit)

    err = 1     # initial error
    tol = 10e-6 # convergence tolerance [rad]

    # initial guess [rad]
    E = (M*(1-np.sin(M+e))+(M+e)*np.sin(M))/(1+np.sin(M)-np.sin(M+e))

    # iterate to solve using Newton-Raphson method
    while err > tol:
        f = E - e*np.sin(E) - M
        f_prime = 1 - e*np.cos(E)
        E_prev = E - f/f_prime
        err = abs(E-E_prev)

    return E

# solve Kepler's equation for eccentric anomaly (parabolic and hyperbolic orbits)
def KeplerHyp(M, e):
    #N: mean anomaly [rad]
    #e: eccentricity [-]. Must be greater than (or equal to) 1 (parabolic/
    #   hyperbolic orbits)

    err = 1     # initial error
    tol = 10e-6 # convergence tolerance [rad]

    H = np.arcsinh(N/e) # initial guess [rad]

    # iterate to solve using Newton-Raphson method
    while err > tol:
        f = E - e*np.sinh(H) - H - N
        f_prime = 1 - e*np.cosh(H) - 1
        H_prev = H - f/f_prime
        err = abs(H-H_prev)

    return H


# transform classical orbital elements into a cartesian state vector
def COE2Cartesian(a, e, i, RAAN, w, M, mu=398601.2, ang='deg'):
    #a:     semi-major axis [km]
    #e:     eccentricity [-]
    #i:     inclination [deg or rad]
    #RAAN:  right ascension [deg or rad]
    #w:     argument of perigee [deg or rad]
    #M:     mean anomaly [deg or rad]

    #mu:    standard gravitational parameter of central body (defaults to Earth
    #       if no input) [km^3/s^2]
    #ang:   unit of angle in use ('deg' or 'rad'). Defaults to 'deg' if none specified

    """
    Note that mu and ang are optional input parameters

    This function takes as inputs the (classical) orbital elements of the
    described orbit, as well as the gravitational parameter of the central
    body. It outputs the position and velocity vectors in cartesian coords;
```

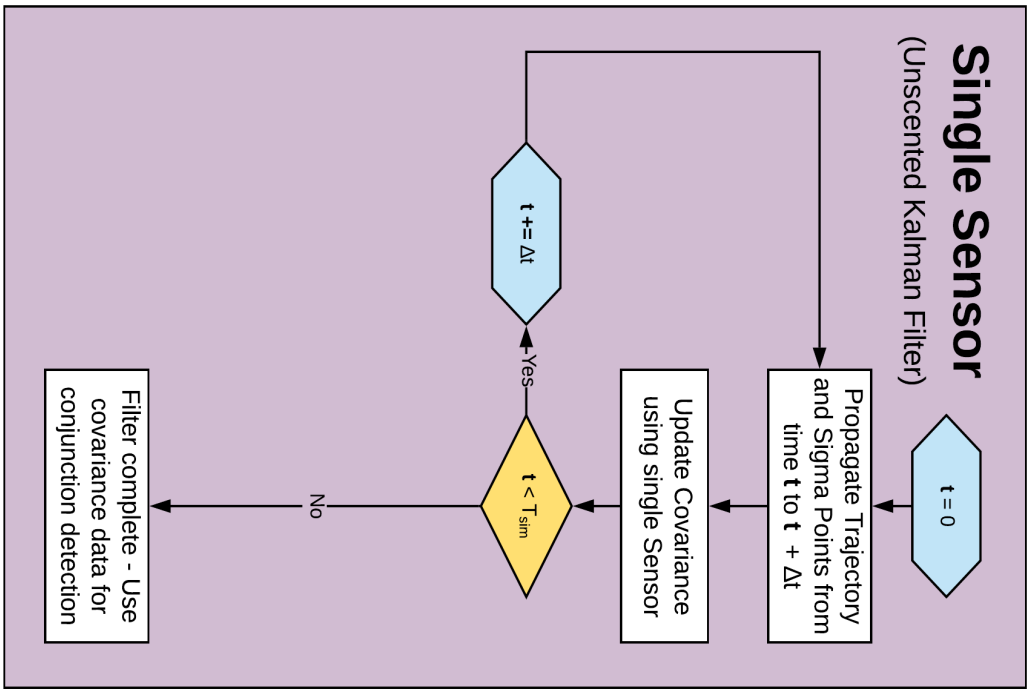
```
75
76      This function has compatibility for parabolic and hyperbolic cases
77      """
78      # convert deg to rad if needed
79      if ang=='deg':
80          i = np.radians(i)
81          RAAN = np.radians(RAAN)
82          w = np.radians(w)
83          M = np.radians(M)
84
85
86      if e < 1:
87          E = Kepler(M, e)
88          f = 2*np.arctan( math.sqrt((1+e)/(1-e)) * np.tan(E/2) )
89          R = a*(1 - e*np.cos(E))
90      else:
91          H = KeplerHyp(M, e)
92          f = 2*np.arctan( math.sqrt((e+1)/(e-1)) * np.tanh(H/2) )
93          R = a*(1 - e*np.cosh(H))
94
95
96      p = a*(1 - e**2)
97
98      # build position and velocity vectors in r,theta,z coord. system
99      r = np.reshape(np.array([R, 0, 0]), (3,1))
100     v = math.sqrt(mu/p) * np.reshape(np.array([e*np.sin(f), 1 + e*np.cos(f), 0]), (3,1))
101
102     # build rotation matrix for coordinate transform from r,theta,z to ECI
103     Q = np.matrix(np.zeros((3, 3)))
104
105     Q[0,0] = np.cos(w+f)*np.cos(RAAN) - np.sin(RAAN)*np.cos(i)*np.sin(w+f)
106     Q[0,1] = -np.cos(RAAN)*np.sin(w+f) - np.sin(RAAN)*np.cos(i)*np.cos(w+f)
107     Q[0,2] = np.sin(RAAN)*np.sin(i)
108
109     Q[1,0] = np.sin(RAAN)*np.cos(w+f) + np.cos(RAAN)*np.cos(i)*np.sin(w+f)
110     Q[1,1] = np.cos(RAAN)*np.cos(i)*np.cos(w+f) - np.sin(RAAN)*np.sin(w+f)
111     Q[1,2] = -np.cos(RAAN)*np.sin(i)
112
113     Q[2,0] = np.sin(w+f)*np.sin(i)
114     Q[2,1] = np.cos(w+f)*np.sin(i)
115     Q[2,2] = np.cos(i)
116
117     # transform position and velocity vectors
118     r = np.array( Q*r ).flatten()
119     v = np.array( Q*v ).flatten()
120
121     # build state vector
122     x = np.hstack( (r,v) )
123
124     return x
125
126
127 # check if spacecraft is in eclipse. Assumes that the distance from the center of the
128 # Earth to the spacecraft is NOT a significant fraction of the distance from the Sun
129 # to the Earth (vector from Earth to Sun is the same as the vector from the spacecraft
130 # to the Sun). This is true for all orbits out to CisLunar space. This algorithm
131 # considers only the Umbra as eclipse criteria, not the Penumbra.
132 def checkEclipse(x, rhatSun, R=6378.14):
133     #x:         state vector of Port w.r.t. ECI [km,km/s]
134     #rhasSun:   unit vector from center of Earth to Earth-Sun Barycenter (ECI)
135     #R:         equatorial radius of eclipsing central body [km].
136     #           Defaults to Earth's radius if none given
137
138     r = x[:3]   # position vector from center of Earth to spacecraft [km]
139
140         eclipse = False # initialize boolean to false
141
142     if np.dot(rhatSun.flatten(),r) < 0: # can only be in eclipse if Earth is
143                                         # between spacecraft and Sun
144         d = np.linalg.norm( np.cross(r,rhatSun) )
145         if d < R:
146             eclipse = True
147
148     return eclipse
149
150
```

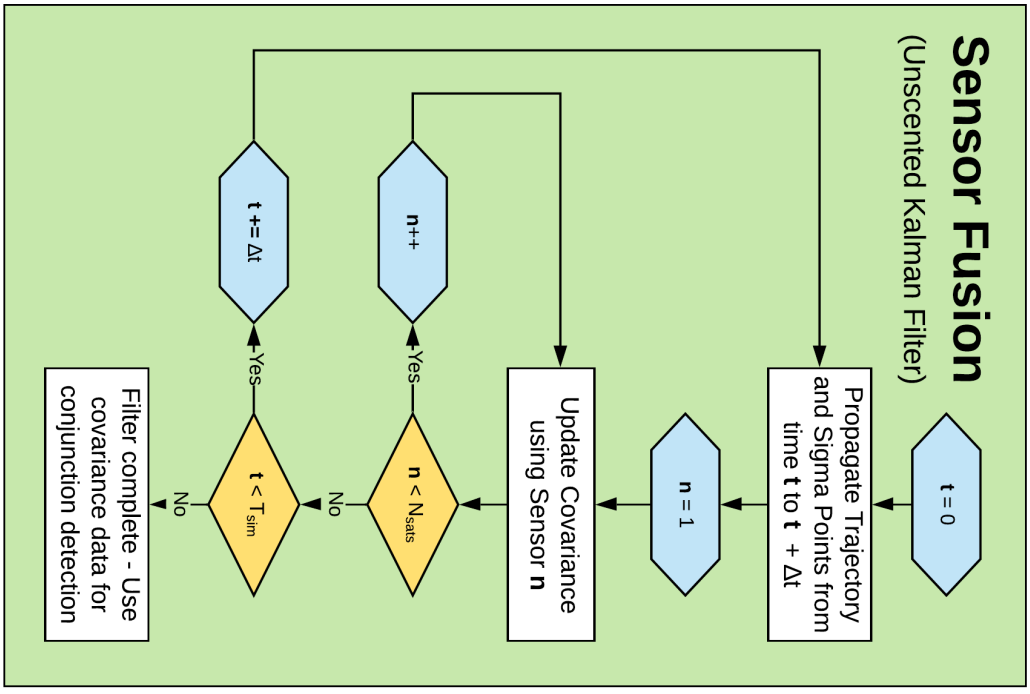
```
def cwEqn(r0,v0,n,t):
    # outputs the rel distance and vel of chaser to target s/c, given initial
    # pos and vel, as well as the mean orbital rate and the propagation time

    # ensure input is in column vector format
    r0 = np.reshape(r0, (3,1))
    v0 = np.reshape(v0, (3,1))

    w = n*t

    phi_rr = np.matrix([[4-3*np.cos(w),      0,          0], \
                        [6*(np.sin(w)-w),    1,          0], \
                        [0,                  0,  np.cos(w)]])

    phi_rv = np.matrix([[np.sin(w)/n,        2*(1-np.cos(w))/n,                  0], \
                        [2*(np.cos(w)-1)/n,  (4*np.sin(w)-3*w)/n,                0], \
                        [0,                  0,                    np.sin(w)/n]])

    phi_vr = np.matrix([[3*n*np.sin(w),      0,              0], \
                        [6*n*(np.cos(w)-1),  0,              0], \
                        [0,                  0,   -n*np.sin(w)]])

    phi_vv = np.matrix([[np.cos(w),          2*np.sin(w),            0], \
                        [-2*np.sin(w),       4*np.cos(w)-3,          0], \
                        [0,                  0,              np.cos(w)]])


    r = phi_rr*r0 + phi_rv*v0
    v = phi_vr*r0 + phi_vv*v0

    return r,v
```

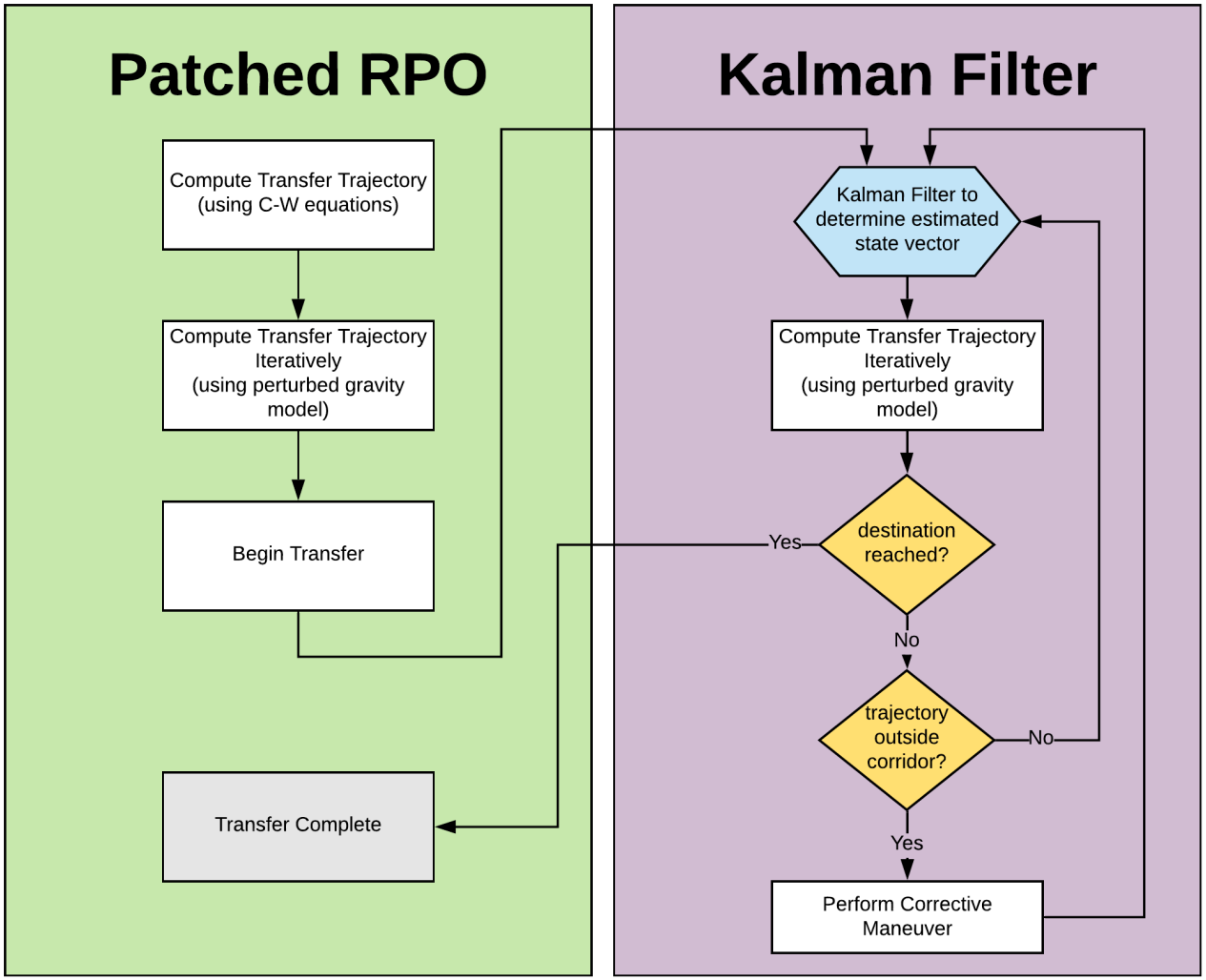
Listing A.4: RotationalKinematics.py

```
"""#Rotational Kinematics Functions

# (C) 2020, Rahul Rughani

# Created: May 29, 2020


def rotSTD(a,ax):
    #a:  rotation angle [rad]
    #ax: standard rotation axis. Valid options are 'x', 'y', or 'z'

    Q = np.matrix(np.zeros((3, 3)))

    if (ax=='x'):
        Q[0,0] = 1
        Q[1,1] = np.cos(a)
        Q[1,2] = -np.sin(a)
        Q[2,1] = np.sin(a)
        Q[2,2] = Q[1,1]

    elif (ax=='y'):
        Q[0,0] = np.cos(a)
        Q[0,2] = np.sin(a)
        Q[1,1] = 1
        Q[2,0] = -np.sin(a)
        Q[2,2] = np.cos(a)

    elif (ax=='z'):
        Q[0,0] = np.cos(a)
        Q[0,1] = -np.sin(a)
        Q[1,0] = np.sin(a)
        Q[1,1] = np.cos(a)
        Q[2,2] = 1

    else:
        warnings.warn('invalid axis selection. Returning Identity Matrix')
        Q[1,1] = 1
        Q[2,2] = 1
        Q[3,3] = 1

    return Q
```

A.4 Sensor Fusion Kalman Filter






Patched RPO
Compute Transfer Trajectory (using C-W equations)
Compute Transfer Trajectory Iteratively (using perturbed gravity model)
Begin Transfer
Transfer Complete
Kalman Filter
Kalman Filter to determine estimated state vector
Compute Transfer Trajectory Iteratively (using perturbed gravity model)
destination reached?
Yes
No
trajectory outside corridor?
No
Yes
Perform Corrective Maneuver

Appendix B

Spherical Harmonic Gravity Model Coefficients

Planetary gravitational fields, like the Earth's, are not perfectly symmetric. The Earth is not a perfect sphere, nor is its mass evenly distributed, thus it has a non-uniform gravitational field. These gravitational perturbations can be described using a spherical harmonic gravitational model, described as follows, with the full explanation being found in 2.1.4.

$$U = \frac{\mu}{r}\left[1 - \sum_{l=2}^{\infty} J_l \left(\frac{R_\oplus}{r}\right)^l P_l[\sin(\phi_{gc_{sat}})] + \sum_{l=2}^{\infty}\sum_{m=1}^{l}\left(\frac{R_\oplus}{r}\right)^l P_{l,m}[\sin(\phi_{gc_{sat}})]\left\{C_{l,m}\cos(m\lambda_{sat}) + S_{l,m}\sin(m\lambda_{sat})\right\}\right] \tag{B.1}$$

The C and S coefficients are required to solve for the potential, which are determined empirically, and are specific to the gravitational field in question. For Earth, this can be found in data from the GRACE mission by NASA and UT Austin, up to the 2160^{th} degree. For the purposes of this analysis, a fourth order analysis using both zonal and tesseral terms is used for orbits in MEO and GEO, whereas a second order analysis with only zonal terms is used for LEO orbits, due to the negligible variations of longitudinal perturbations in LEO compared to the latitudinal perturbations (J2 effect). The data below shows these coefficients up to the 5^{th} degree, retrieved from http://download.csr.utexas.edu/pub/slr/degree_5/CSR_Monthly_5x5_Gravity_Harmonics.txt on February 19, 2021. These values are updated monthly from the GRACE-FO satellites.

```
GGM05C coefficients:
earth_gravity_constant        3.986004415E+14
radius                        6.378136300E+06
errors                        calibrated  (sigmas have been adjusted to be more accurate)
norm                          fully_normalized
tide_system                   zero_tide
format                        2I5,2D20.12,2D13.5
```

L	M	C	S	sigma C	sigma S
2	0	-4.841694573200D-04	0.000000000000D+00	1.17430D-10	0.00000D+00
2	1	-3.103431067239D-10	1.410757509442D-09	4.29920D-11	4.29620D-11
2	2	2.439373415940D-06	-1.400294011836D-06	3.68360D-11	3.63870D-11
3	0	9.571647583412D-07	0.000000000000D+00	1.30040D-11	0.00000D+00
3	1	2.030446637169D-06	2.482406346848D-07	7.71580D-12	7.69980D-12
3	2	9.047646744100D-07	-6.190066246333D-07	1.17100D-11	1.17290D-11

```
3    3  7.212852551704D-07  1.414400065165D-06  2.34700D-11  2.34810D-11
4    0  5.399815392137D-07  0.000000000000D+00  6.73720D-12  0.00000D+00
4    1 -5.361808133703D-07 -4.735769769691D-07  5.29480D-12  5.28940D-12
4    2  3.504921442703D-07  6.625051657439D-07  7.61690D-12  7.61490D-12
4    3  9.908610311151D-07 -2.009508998058D-07  1.28570D-11  1.28620D-11
4    4 -1.884924225276D-07  3.088185785570D-07  1.34180D-11  1.33940D-11
5    0  6.865032345839D-08  0.000000000000D+00  3.20970D-12  0.00000D+00
5    1 -6.291457940968D-08 -9.434259860005D-08  2.56020D-12  2.55850D-12
5    2  6.520586031691D-07 -3.233430798143D-07  3.36790D-12  3.36650D-12
5    3 -4.518313784464D-07 -2.149423673602D-07  6.27900D-12  6.27680D-12
5    4 -2.953234091704D-07  4.981057884405D-08  7.88550D-12  7.89070D-12
5    5  1.748143504694D-07 -6.693546770160D-07  1.22840D-11  1.22710D-11
6    1 -7.594326587940D-08  2.652568324970D-08  2.52870D-12  2.52720D-12
```