© 2025 Richard Eason

MISSION DESIGN ANALYSIS METHODOLOGY FOR SPACE-BASED COMPUTATIONAL DATA CENTERS

BY

RICHARD EASON

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2025

Urbana, Illinois

Advisor:

Clinical Associate Professor Michael Lembeck

Abstract

The increasing volume and rate of data generated by Earth observation and other satellite constellations present significant challenges to traditional ground-based processing paradigms, primarily due to downlink bandwidth limitations and latency constraints. This thesis investigates the emerging concept of shifting computational tasks to orbit, utilizing dedicated space-based data center platforms networked with sensor spacecraft. Such distributed architectures offer potential benefits, including reduced data downlink requirements, lower latency for time-critical applications, and leveraging of in-space resources like solar power. However, designing these complex mission architectures involves navigating numerous interdependent technical and economic trade-offs across the "system of systems".

To address this, a mission design analysis methodology centered around a high-fidelity simulation tool developed within the a.i. Solutions' FreeFlyer environment is presented. This simulator models the dynamic interactions between key spacecraft subsystems—including electrical power, thermal management, communications (inter-satellite and space-to-ground), propulsion, orbital mechanics, and data generation/processing payloads across both data-generating and data-processing spacecraft roles. The tool enables the evaluation of various architectural configurations and operational scenarios, facilitating the analysis of design parameter sensitivities, assessment of trade-offs (e.g., centralization vs. distribution, power allocation, thermal rejection), and optimization of system performance against metrics like data throughput, latency, and mass-to-orbit. This work provides a framework and a quantitative tool intended to aid mission design engineers in the systematic design and analysis of future space-based computational data center missions.

To my parents, who have supported and encouraged me through every step on this path.

Acknowledgments

This work has been one of the most challenging and rewarding experiences of my life. It has left me profoundly grateful for the support, both technical and emotional, of the many wonderful and talented people in my life and academic path.

I am privileged and deeply grateful to have had the opportunity to learn from my advisor, Dr. Michael Lembeck. His unwavering support of myself, my friends, and my colleagues has taught me what it means to be a leader, and what it means to be an engineer. I do not know where I would be without him.

I would like to thank Dr. Rakesh Kumar and Dr. Nathaniel Bleier, whose insight and guidance was indispensable in the development of this idea.

I would like to thank my fellow lab members who worked in the LASSI research group and have put up with me these last seven years. Specifically, I would like to acknowledge Michael Harrigan, Eric Alpine, James Helmich, Qi Lim, Chris Young, Hongrui Zhao, Cameron Jones, and Ben Ochs. It has been an honor to know and work with each of you. I would in particular like to acknowledge our lab manager, Murphy Stratton. Her friendship and support have been instrumental in shaping who I am today. Each of these LASSI team members and many more have assisted me in part in creating this thesis.

I would like to give a special thanks to Young Sang Ryu, whose support and coaching have enabled me to complete this work.

I am especially thankful to Kelsey Eggimann, whose presence, support, and love have carried me to this point. Her hard work and late nights throughout her own educational journey have brought us both to this point. I am deeply proud of her, as I know she is of me.

Finally, this work would not have been possible without the unflinching support of my parents. Their tireless encouragement of my interests and education have given me more than I could ever express. Without them I wouldn't be where I am today. Thank you. I love you.

Table of contents

List of Abbreviations

ADC	Analog to Digital Converter
ADCS	Attitude determination and control
AI	Artificial Intelligence
BOL	Beginning of Life
CONOPS	Concept of Operations
COP	Coefficient of Performance (figure of merit in heat pumps)
CPU	Central Processing Unit (computer component)
CSV	Comma separated value (file type)
DCS	Data Center Spacecraft
DGS	Data Generator Spacecraft
DoD	Depth of Discharge (of battery)
EGM96	Earth Gravitational Model of 1996
EIRP	Effective isotropic radiated power
EO	Earth Observation
EOL	End of Life
EPS	Electrical Power System
FCC	Federal Communications Commission
FSPL	Free-space path loss
GND	Space-to-ground communications system
GS	Ground Station
IR	Infra-red
ISR	Intelligence, Surveillance, and Reconnaissance
ISS	International Space Station
IV	Current-Voltage relationship
MET	Mission Elapsed Time (Years)

ML	Machine Learning
MPPT	Maximum Power Point Tracker
MLI	Multi-layer insulation
NOAA	National Oceanic and Atmospheric Administration
QFD	Quality Functional Deployment
RAAN	Right-Ascension of the Ascending Node
RF	Radio frequency
RX	Receive
SMA	Semi-major axis (of an ellipse)
SNR	Signal to noise ratio
STD	Standard Deviation
$\angle SVE$	Sun-Spacecraft-Earth angle
TX	Transmit
UHF	Ultra-high frequency
UV	Ultra-Violet light

List of Symbols

$Q_{Collected}$	Energy collected
η	Efficiency
E_{solar}	Solar irradiant flux density
A	Area
Δt	Time step
J	Joules
sec	Seconds
m^2	Square meters
W	Watts
Wh	Watt-hours
mAh	milli-ampere-hours
ρ	Density
c	Specific heat capacity
kg	Kilograms
ϕ	Radiant flux
$\odot$	The Sun
$\oplus$	The Earth
$\angle$	Indicates a names angle
ϵ	Emissivity of a material
σ	Stefan-Boltzmann constant (5.670374e-8 $\frac{W}{m^2K^4}$)
k	Boltzmann's constant (1.380649e-23 $\frac{J}{K}$)
Hz	Hertz
MHz	Megahertz
bps	Bits per second
GB	Gigabyte
Gbps	Gigabits per second

Chapter 1

Introduction

The allocation of functionality in space-based systems has undergone profound transformations since the launch of the first satellite. Early Earth Observation (EO) missions typically involved single, large satellites tasked primarily with mapping the planet[1]. Data, often in the form of images, was collected and downlinked to a limited number of ground stations. Subsequent processing and analysis occurred terrestrially, largely unconstrained by significant temporal pressures, fulfilling the objective of building comprehensive planetary maps[2]. The volume of data from these individual platforms, while substantial for its time, was generally manageable within the existing ground infrastructure's processing and bandwidth capabilities[2].

Over time, a paradigm shift towards constellations of smaller, more numerous spacecraft has occurred. This evolution enabled new mission objectives focused less on static mapping and more on dynamic monitoring and detecting changes in near real-time[3]. Such applications demanded higher revisit rates and generated an unprecedented surge in data volume. Consequently, the traditional model of downlinking all raw data for ground-based processing encountered significant strain. The sheer quantity of imagery saturated available downlink bandwidth, creating bottlenecks and increasing delay between image capture and analysis. Furthermore, the computational load required to compare vast datasets for change detection escalated dramatically, driving up the cost and complexity of the necessary ground infrastructure[4].

In response to these mounting challenges, the aerospace and computing communities have begun exploring novel architectural solutions, as reflected in recent technical literature and program announcements[5][6][7][8][4]. A prominent approach involves shifting data processing capabilities from the ground directly onto the spacecraft themselves or dedicated orbital platforms[4][9]. By performing tasks like change detection or feature extraction in orbit, systems can transmit only the relevant information or condensed data products, drastically reducing the volume of data requiring downlink. This "space edge computing" model not only alleviates bandwidth constraints but also significantly reduces the latency between data acquisition and actionable insight, which is critical for time-sensitive applications[10]. Additionally, major benefits can be found in the potential to leverage unique on-orbit resources, such as abundant solar power, potentially offering environmental benefits compared to energy-intensive terrestrial data centers[9].

The architectural evolution extends further, envisioning systems where computational resources are concentrated into separate, dedicated platforms distinct from the sensing satellites themselves. This leads to the concept of networked "system of systems" architectures, comprising multiple data-generating sensor platforms serviced by specialized on-orbit data centers[4][9]. These data center spacecraft (DCS) can leverage high-bandwidth inter-satellite communication links to efficiently ingest data from constellations of sensors

and perform complex processing tasks, including computationally intensive AI/ML workloads[9][5][4].

This distribution of sensing and processing capabilities across multiple, interconnected space assets introduces a complex optimization problem. Designing such architectures requires careful consideration of numerous interdependent factors. Many questions arise, all with direct impact to mission and spacecraft design. What is the optimal ratio of "compute" platforms to "sensor" platforms (Figure 1.1)? What are the most critical design parameters? How can mission designers achieve the right balance between centralization and distribution to maximize overall system effectiveness and economic viability?

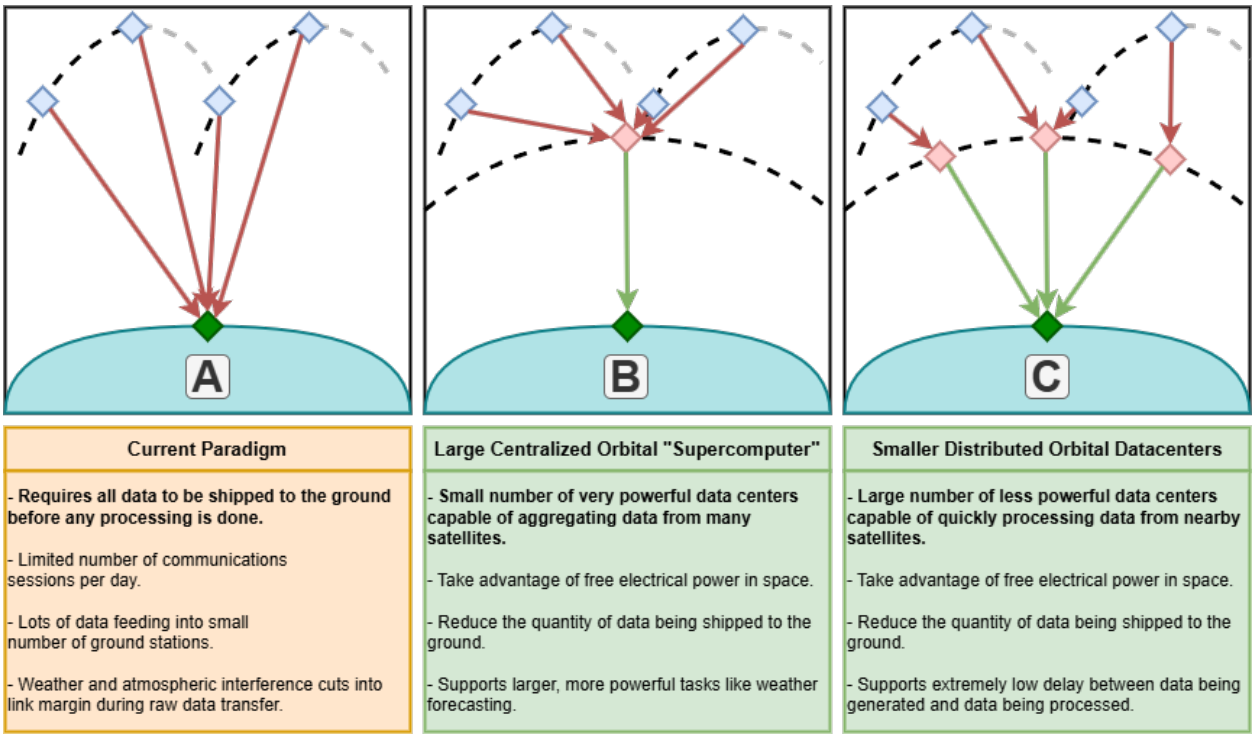

Figure 1.1: A simplified comparison of three different concepts of operations (CONOPS) for data processing architectures. The first system does not perform processing in space, with each data-generating spacecraft (blue diamond) sending its raw data (red arrow) to the ground for processing. The second panel depicts a system where many data-generating spacecraft communicate to a single orbiting supercomputer (red diamond) that sends its processed data (green arrow) to the ground. The third system distributes raw data among multiple data processing spacecraft, each sending processed data to the ground.

Addressing these questions is crucial for the successful realization of next-generation space infrastructure. It is the objective of this thesis to provide a methodology and a tool to investigate the complex optimization issues inherent in large-scale, distributed sensor and on-orbit compute architectures.

Chapter 2

Motivation and Background

Interest has risen significantly in the concept of performing significant amounts of computation in space[5][7][4][9]. Currently, nearly all artificial satellites that generate data do so without processing it in order to draw insight from it. Fundamentally, *insight* is what is desired from these satellites and their data; a file containing radar return data is useless to a farmer who needs to know if it will rain tomorrow. Satellites generate this data in a raw (albeit usually compressed) format, then transmit it to the ground where it is processed through a combination of decoding and calibrating the signals, geotagging them with location and time information, and then converting them into human-usable formats like images or datasets that can be spatially and temporally analyzed for the desired insights[11]. This strategy is fine for situations where expediency is not a requirement, and the amount of data being downlinked is low. However, availability of a ground station and the data rate of the downlink channel can often be limiting[3][11]. This leads to a major bottleneck in some systems, where large amounts of data are being delayed from being downlinked to the ground due to uncontrolled factors such as extreme weather or ground hardware failure. There are also many situations where rapid insight from the collected data is highly desirable, such as during weather emergencies or detecting missile launches. In such situations, any delay for a queue of data to be downlinked before distilling and getting access to critical information could have significant consequences. Figure 1.1 illustrates some of the limitations of the current approach.

The ultimate purpose of collecting *data* on orbit is to obtain *information*. Today, all of this processing of data into information happens on the ground, requiring satellite assets to transmit the bulk raw data down to Earth. Data must be processed, filtered, transformed, and formatted to gain insight from it. This processing, if accomplished on board or near the sourcing satellite could significantly reduce the size, time, and cost of any downlink. Figure 1.1 illustrates how data being processed in orbit can alleviate the constraints seen by current systems.

Inter-satellite links (ISL) are communications hardware systems designed to communicate from one spacecraft to another, as opposed to space-to-ground communications. ISL systems benefit from increased RF bandwidth at microwave frequencies without the attenuation introduced by the atmosphere, allowing for dramatically increase data rates. This makes it much easier to ship large amounts of unprocessed data from a spacecraft that generates it to another spacecraft that can process it.

Additional benefits arise in the reduction of operating costs as ground compute is moved to orbit. In 2023, the U.S. Department of Energy found that data centers consumed 176 TWh of energy, accounting for 4.4% of the entire U.S. electrical energy consumption that year[12]. This energy is purchased from electricity supply

utilities and is used to power both the computers themselves and the industrial cooling systems required for those computers. Operating costs expand further as land, water, and facilities must be acquired, maintenance and engineering staff employed, and taxes paid. When these compute systems are in orbit however, many of these overhead costs go away. Electrical power is harvested for free from solar arrays to power computers and cooling is through radiative means into deep space. There is no land to build on, maintain, and be taxed for, and what engineers are on-staff to remotely operate the system can operate multiple spacecraft simultaneously, increasing their cost effectiveness as employees.

Chapter 3

Component Systems of Spacecraft Missions

This work aims to identify a robust approach for the simulation and analysis of data collection and processing architectural configurations and complementary operational scenarios. These methodologies will help identify key design parameter sensitivities, allow the evaluation of design trade-offs, and provide insights and guidance for future mission designers navigating this challenging and rapidly evolving domain.

The systems engineering process decomposes mission objectives into a set of integrated functional and performance elements comprising a "system of systems." Figure 3.1 represents the system of systems decomposition for data generation spacecraft elements, data processing spacecraft elements, and ground control components. The data generation spacecraft and the data processing spacecraft share many functional subsystems, as both require similar functionality from their subsystems to execute their missions.

The development of a simulator responsible for accurately modeling the function of these subsystems is required to accurately model the function and interfaces of the system of systems. This tool can then be used to evaluate and optimize architectural options for maximizing system efficiency in terms of lofted mass, total data throughput, and data latency.

To develop a system design of acceptable fidelity, the system-of-systems must be unraveled to identify the subsystems that require sizing analysis to achieve a design solution that closes sufficiently to meet the mission objectives. In the case of the data generation and data processing spacecraft, these subsystems include:

- Electrical power systems
- Thermal management systems
- Mechanical structure and interfaces
- External communications
- Propulsion
- Attitude determination and control (ADCS)
- Flight management
- (Data Generation or Data Processing) Payloads

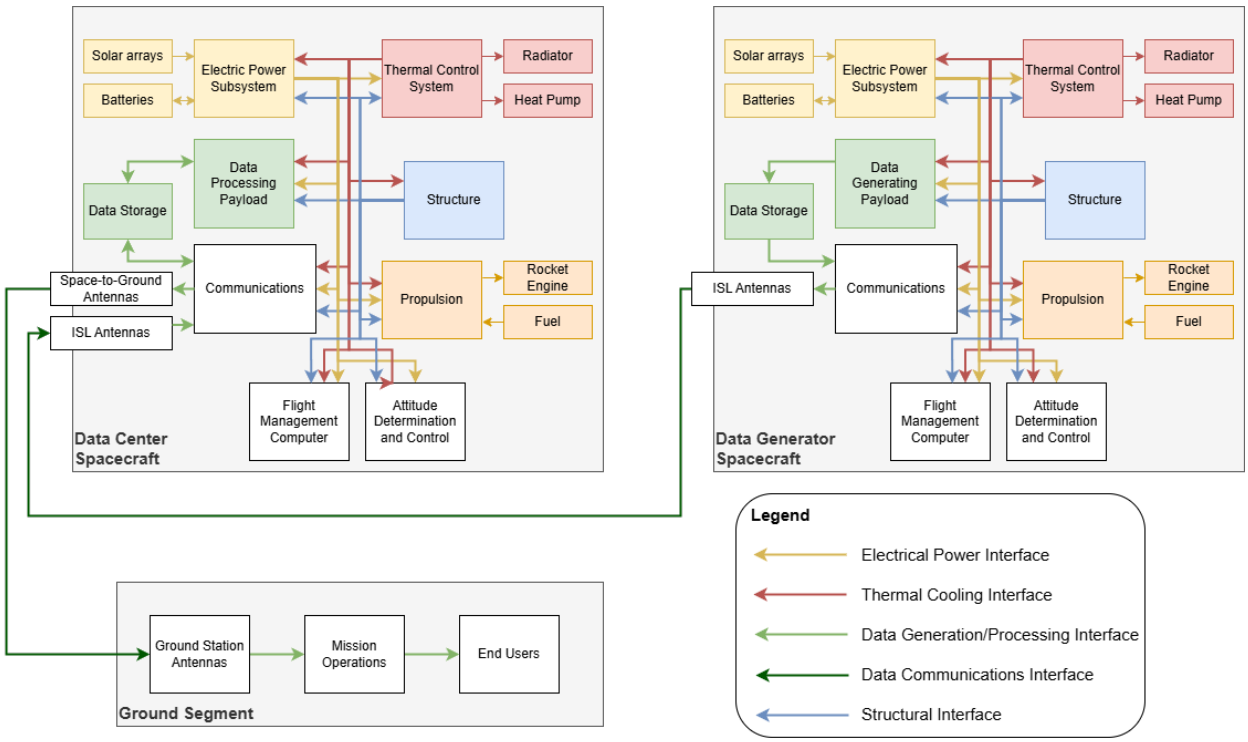

Figure 3.1: A functional breakdown of the system of systems involved in the simulation.

Design of the spacecraft evolves from a decomposition of the system of systems into a set of functional and performance requirements that define their precise behavior. Analysis is then performed to affirm that a given design will, when implemented, achieve the mission requirements.

3.1 Electrical Power System

A spacecraft EPS generates and distributes the necessary electrical energy for all onboard subsystems to operate effectively throughout the mission lifecycle. This includes powering critical functions such as communication, telemetry, attitude control, propulsion, scientific instrumentation, and thermal management. The EPS ensures the continuous and reliable supply of electrical power that is fundamental to mission success. The EPS encompasses power generation, energy storage, power conditioning, and distribution, all managed to optimize efficiency and longevity within the constraints of the space environment.

The power generation source is typically photovoltaic arrays (i.e., solar panels) that convert sunlight into electricity. The energy storage system, usually consisting of rechargeable batteries, stores energy generated during periods of sunlight and provides power when the primary source is eclipsed and during periods of peak demand. A distribution component regulates the flow of power, conditions it to the required voltage and current levels for different subsystems and distributes it efficiently throughout the spacecraft .

3.2 Thermal Management System

The purpose of a spacecraft's thermal management system is to maintain the operating temperatures of all onboard components within their specified limits, ensuring optimal performance and longevity. In the vacuum of space, heat transfers throughout the spacecraft primarily through radiation between nearby components, conduction between components in direct contact. Thermal management is crucial for both dissipating internally generated heat from electronics and other equipment, as well as mitigating extreme temperature fluctuations caused by solar radiation, Earth albedo (i.e., light reflected from the Earth), and infrared emissions. Effective thermal management is paramount for the reliable operation of sensitive instruments, propulsion systems, power generation units, and the overall structural integrity of the spacecraft.

The basic components of a thermal management system may include passive and active control elements. Passive thermal control relies on material properties and surface treatments to regulate heat flow. This includes multi-layer insulation (MLI) to minimize radiative heat transfer, thermal coatings with specific emissivity and absorptivity characteristics, and thermal straps or fillers to enhance conductive heat transfer between components and heat sinks. Active thermal control involves mechanical or electrical systems to move heat. Examples include radiators to radiate waste heat into space, heat pipes to efficiently transport heat from one location to another, and fluid loops that circulate a coolant to collect heat from various sources and transport it to radiators for dissipation .

3.3 Communications

A spacecraft communication system facilitates the exchange of information between the spacecraft and ground-based control centers or other designated entities. This bidirectional link enables the transmission of telemetry data, which includes crucial information about the spacecraft's health, status, and performance. Simultaneously, it allows for the uplink of commands and instructions from ground control to manage the spacecraft's operations, adjust its trajectory, and control its scientific instruments. The most important task of a communications system is downlinking payload data collected or processed by the spacecraft.

The basic components of a spacecraft communication system include antennas, transceivers, and data handling systems. Antennas serve as the interface for transmitting and receiving radio frequency (RF) signals. Their size and design depend on the operating frequency, data rate requirements, and the distance over which communication needs to be established. Transceivers encompass both transmitters, which modulate and amplify signals for up/downlink, and receivers, which detect and demodulate incoming signals. Data handling systems are responsible for encoding and decoding data, implementing error correction techniques to ensure data integrity, and managing the flow of information between the spacecraft's subsystems and communications hardware. Doppler shifts caused by the relative motion of the spacecraft and ground station must also be accounted for to maintain a reliable communication link.

Currently, most wireless communications traffic to and from spacecraft is done over space-to-ground channels. For high data rate systems this is usually done over microwave bands out of necessity, but many smaller spacecraft operate in UHF[13].

For a number of reasons, spacecraft-to-spacecraft links in orbit are easier to handle. Primarily this is due to the lack of an appreciable atmosphere or other material media in between most spacecraft. While there are conditions where occlusion between spacecraft occurs (primarily when Earth is in between them) this is a somewhat binary state change to the link, where either the planet is not in the way and the link is not

subject to its occlusion, or the planet is in the way and the link fails.

Using RF bands to handle ISL communication generally presents fewer complexities as compared to space-to-ground communications. ISL links only have to content with free-space path loss. However, because distances between spacecraft are often much larger than distances between spacecraft and ground stations, this path loss can quickly become severe.

3.4 Propulsion

A spacecraft's propulsion system provides the means to alter the spacecraft's velocity and orientation in space, enabling a wide range of mission critical maneuvers. This includes orbital insertion to place the spacecraft into its desired orbit, trajectory correction maneuvers to maintain the planned flight path, and station-keeping to counteract orbital perturbations.

The basic components of a spacecraft propulsion system include propellant tanks and plumbing, control and isolation valves, propellant, and thrusters. Chemical thrusters use the heat created through the oxidization or catalyzed decomposition of propellant to generate gas pressure inside a confined space, forcing propellant to accelerate out of the rocket engine. Electric propulsion systems use electrical power in various forms to accelerate propellant at higher specific impulses than chemical thrusters.

3.5 Structure

The purpose of a spacecraft's structural system is to provide the mechanical framework that supports and protects all other spacecraft subsystems and payloads throughout the mission. This includes withstanding the intense vibrations of launch, maintaining integrity under thermal stress, absorbing external micrometeoroid impacts, and ensuring the precise alignment and stability of critical components such as sensors, antennas, and solar arrays.

3.6 Orbital Mechanics

In addition to the physical components in an spacecraft, its performance is driven significantly by orbital mechanics. The spacecraft's orbital parameters, such as altitude, inclination, and eccentricity, dictate its period of revolution around the Earth, directly impacting the frequency and duration of communication windows with ground stations. For Earth observation missions, the choice of orbit determines the geographical coverage and revisit time, while for telecommunications satellites, geostationary orbit provides continuous coverage over a specific region. The distance from the Earth also affects the strength of the terminal communication signals.

Gravitational perturbations from the Sun and Moon, as well as atmospheric drag, continuously act upon a spacecraft, causing its orbit to deviate from the intended path. To maintain the desired orbit and ensure proper functioning, spacecraft operations often require periodic orbital maneuvers, which consume valuable propellant and necessitate precise timing and execution. Station-keeping maneuvers are essential for satellites in geostationary orbit to prevent them from drifting out of their designated slot. Understanding and accurately predicting orbital mechanics is crucial for mission success, as it directly impacts fuel budgeting, operational scheduling, and the lifespan of the spacecraft.

Chapter 4

Simulator Implementation

4.1 Simulator Purpose and Structure

a.i. Solutions' FreeFlyer is a space mission design and analysis software tool used for both mission development and flight operations[14]. FreeFlyer natively supports high-fidelity modeling and analysis of astrodynamics propagation, maneuvering and optimized targeting, and RF coverage and contact analysis. The software is based around an internal scripting engine which can be used to configure the tool for the specific needs of each mission.

The simulator approaches user-modifiable configuration parameters in one of two ways: 1) Global parameters such as material properties and temperature limits, and 2) Spacecraft-specific parameters such as a spacecraft's dry mass and orbital eccentricity. Global parameters apply to all spacecraft and are stored as variables directly in FreeFlyer. Spacecraft-specific parameters can vary between spacecraft and are specified in a separate spacecraft configuration CSV file.

To use the simulator, the file Initial_Analysis.MissionPlan is opened in FreeFlyer. All critical settings and required global configuration parameters are located in the FreeForm block titled "Initialize MissionPlan". Figure 4.1 shows the top of a FreeForm script section, including the file path variables used to point the simulator to the spacecraft configuration file, as well as the desired output data and console log files.

The spacecraft definition file is a user-generated CSV file that contains a list of all spacecraft and ground stations, as well as their design parameters. The file definition table and description is detailed in appendix A.

Once the appropriate parameters are set and the configuration file is written, the user runs the FreeFlyer script, and the display changes to a three-panel status annunciator as shown in Figure 4.2. The top panel is the log console, that displays information updates as they occur. The real-time log is culled periodically, however if logging to a file is enabled in global configuration settings, then console alerts are not lost. The middle panel is a quick-reference runtime telemetry display. Its values are updated every 1000 simulator steps, and it provides a clear display of the progress of the simulation including estimated completion time. The lower panel is a user configurable 3D rendering of the spacecraft and ground stations in the simulation. If enabled in the global configuration settings, this display will update in real-time with the simulator, however it is recommended to leave real-time graphics updates disabled as it consumes a lot of compute time and slows the execution of the simulation.

Once a simulation run is complete, some critical output parameters (such as total data quantities) are

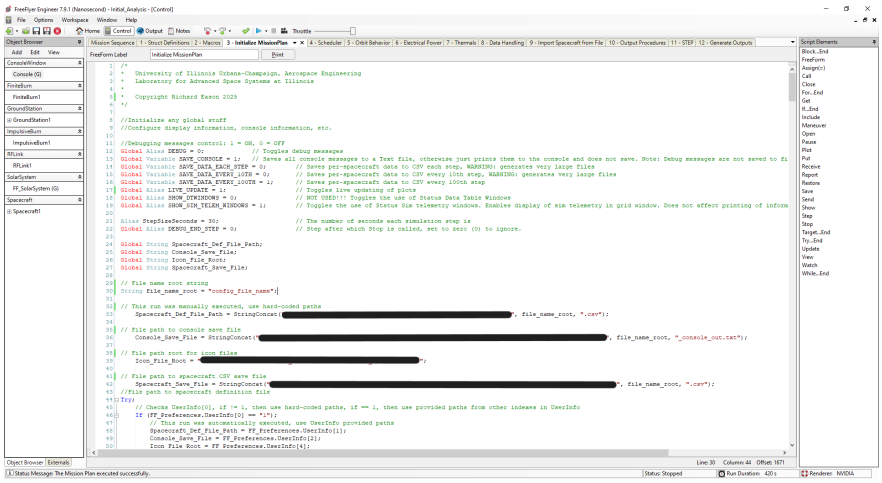

Figure 4.1: FreeForm script module showing file locations. Actual file locations on a private server are redacted.

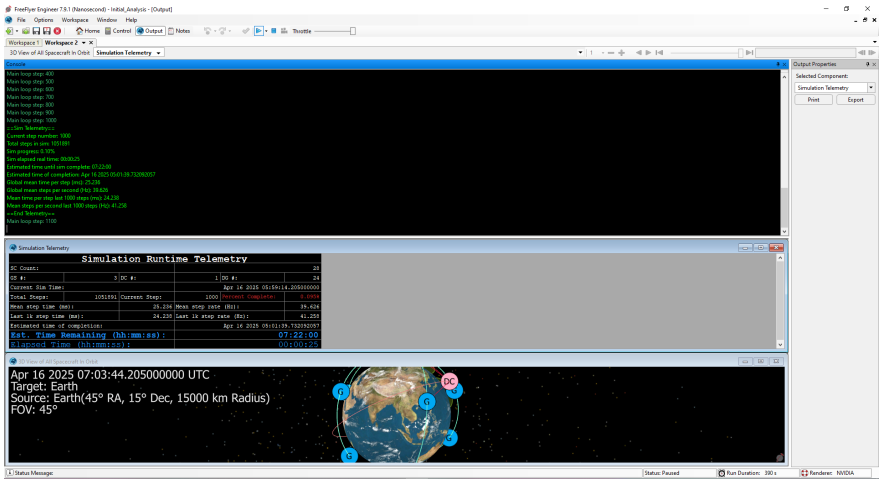

Figure 4.2: Running sim status annunciator example.

written to the console log file, and all exported simulation data is presented in a output CSV file. This file contains a complete set of all per-spacecraft values and parameters. A detailed specification of what data is output in this file can be found in Appendix B.

The simulator tool breaks a space-based compute mission architecture into three different roles:

1. **Data Generator Spacecraft (DGS)**, which generate data to be sent to data centers for processing.
2. **Data Center Spacecraft (DCS)**, which receive data from data generators and process it for downlink to the ground.
3. **Ground Stations (GS)**, which are stationary on Earth surface and receive data from DCS.

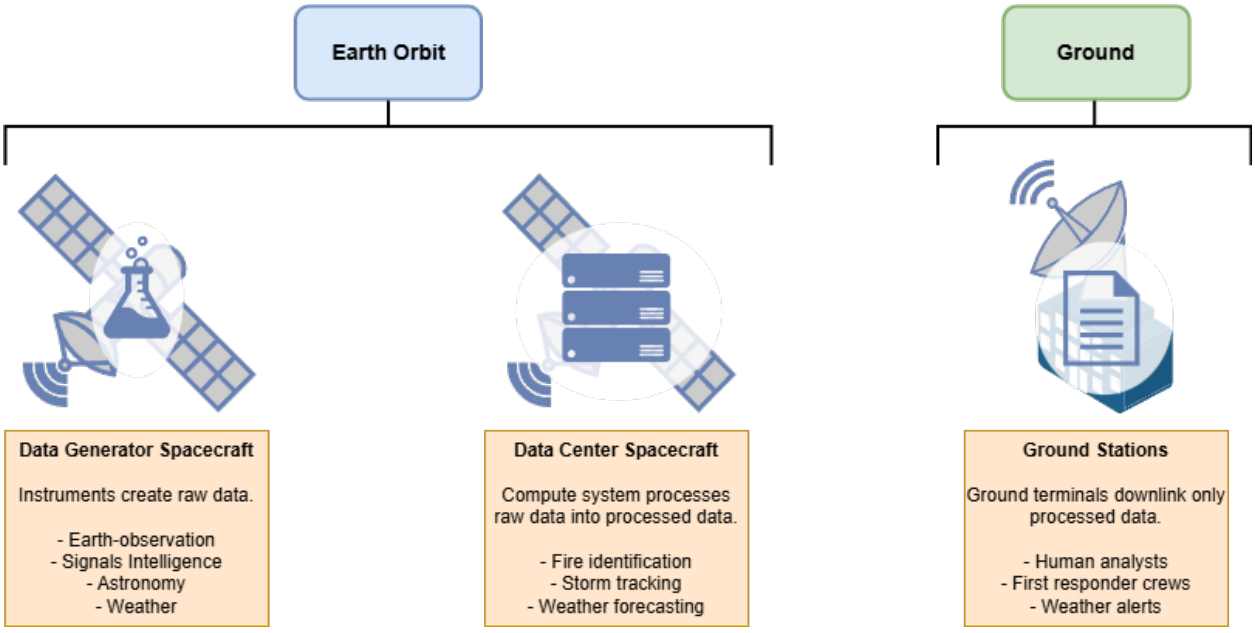

Figure 4.3: Example architecture depicting distinct data generators, data centers, and ground station roles.

Each of these systems is defined in a FreeFlyer system configuration file, which allows the various design parameters of each system to be defined. Table 4.1 shows an abridged set of design parameters to demonstrate what part of a spacecraft configuration file might look like, refer to Appendix A for full file specification. In this example, if the type field is configured for a ground station, then the fields used for GS latitude and longitude are read in and the orbital parameters are ignored. Similarly, if the type is a spacecraft, then spacecraft-related parameters are read in such as orbital, propulsion, and power parameters.

Qty	Type	Orbit Inclination	Orbit SMA	Orbit Eccentricity	Data Generation Rate	Data Processing Rate	Solar Array Area	Radiator Area	Mass	GS Latitude	GS Longitude
2	DGS	98	6,853	0.0	500k	-	0.4	0.1	5.2	-	-
1	DCS	40	7,200	0.1	-	10,000k	0.2	0.1	4.6	-	-
1	DCS	40	7,500	0.1	-	10,000k	0.2	0.1	4.6	-	-
1	GS	-	-	-	-	-	-	-	-	40.111705	-88.228171

Table 4.1: Example abridged spacecraft configuration file with two identical DGS, two DCS in different orbits, and one GS.

Figure 4.4 describes the core execution flow. At runtime, FreeFlyer performs a series of initialization steps where it configures itself for a given architecture. It then reads in the spacecraft configuration file and loops through each line of the file, instantiating spacecraft and ground stations as appropriate until the configuration file is fully loaded. The script then proceeds into the main execution loop.

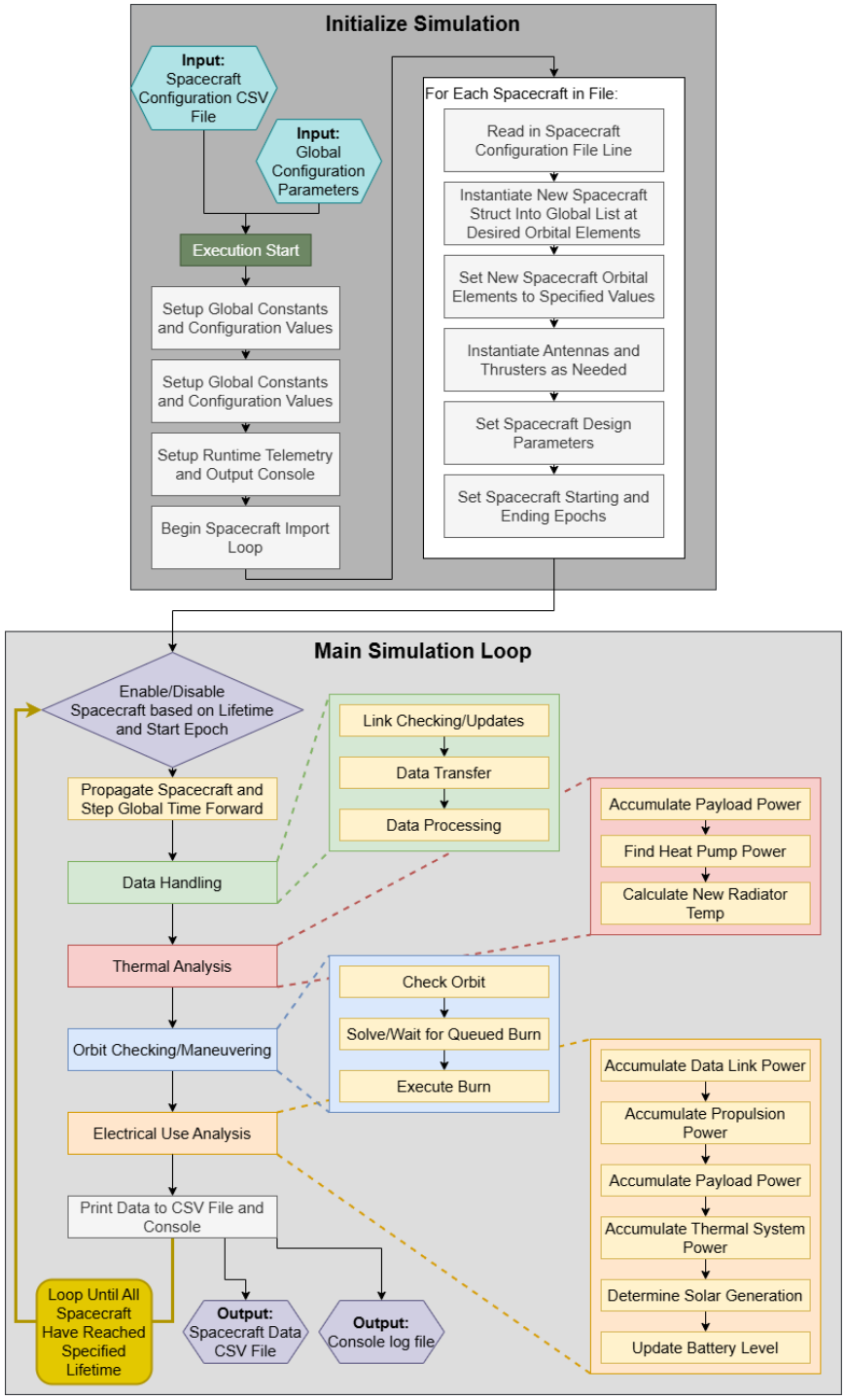

Figure 4.4: Main simulation execution diagram. Primary inputs are the spacecraft configuration file and all global configuration parameters. Primary outputs are the spacecraft data CSV file and the console log file.

The main execution loop checks to see that each spacecraft which is supposed to be operational (has a starting epoch in the past) is set to operational status, as well as checking that each spacecraft that has reached specified end of life (has a starting epoch and lifetime in the past), or has demised in the atmosphere, has been disabled. From there it steps each spacecraft forward in time and executes the simulation components of each spacecraft sequentially. It starts with data handling and processing, then thermal analysis (which relies on up-to-date data handling quantities), then orbital maneuvering, then electrical power consumption and generation (which relies on up-to-date data handling, thermal, and propulsion energy use quantities). Once all spacecraft have been simulated up to the current timestep, all spacecraft data is written to the output CSV file, and the loop starts again a single time step into the future.

4.2 Development of Simulator in FreeFlyer

The simulator script written for this work comprises 7,572 lines of FreeFlyer FreeForm code and took more than a year to develop. Through this process, several limitations of the FreeFlyer system were encountered and significant effort was made to work around them. The lessons learned from this development effort are considerable, and color the depth of work involved in this process.

4.2.1 Complexities of Using FreeFlyer for Arbitrary Spacecraft Configurations

The FreeFlyer software is extremely powerful, but it is also built around a set of assumptions and expectations about the type of work which is being done. Central to this concept is that FreeFlyer is designed to provide high-accuracy results for a single scenario that is configured through the program user interface prior to runtime. The intended workflow involves the manual creation of each spacecraft, ground station, RFLink, graphical element, and procedural segment. FreeFlyer is *not* designed to rapidly vary mission design parameters with little to no manual input from the user. Unfortunately, this capability is fundamental to the type of solution space optimization that this simulator tool is designed to support.

FreeFlyer is a structurally-closed scripting environment, but it does support the creation of data structure objects (analogous to the `struct` type in C/C++) and the creation of "Procedures" (functions). Custom object class prototypes and procedure prototypes are not supported. This means that the only way to attach additional custom variables to a spacecraft is to place both the custom variable object and the spacecraft object into a `struct` and treat that `struct` as a custom class. This forces all interaction with the spacecraft object to first go occur through the `struct` instance it is housed inside. In the simulator , there are four major "custom classes" defined in this manner:

1. `DefinedAntenna`, which stores information about that communications node's maximum limits, as well as index pointers to the `DefinedSpacecraft` it is attached to, the FreeFlyer Sensor object on that spacecraft which represents this antenna (Sensors are used to check occlusion and view factor), and index pointers used to keep track of what spacecraft and antenna it is talking to when it is used in an active communications link.

2. `CommsLink`, which stores index pointers to the transmitting and receiving `DefinedSpacecrafts` and which `DefinedAntennas` it is linked between, as well as the RFLink and VisibilitySegment FreeFlyer objects, which are used to do math on the communications link.

3. `ScheduledEvent`, which stores information about the desired timing of a scheduled burn, what

`DefinedSpacecraft` it is associated with, and the index of the intended maneuver in the list of maneuvers inside the `DefinedSpacecraft`. The scheduler traverses the global master list of `ScheduledEvent` to check for events to execute each step.

4. `DefinedSpacecraft`, this is the largest `struct`, which contains the spacecraft object itself, as well as `struct` lists of that specific spacecraft's `DefinedAntenna` (set up during spacecraft import), and lists of all burns which are associated with that specific spacecraft (appended to by burn solvers as burns are found and scheduled). The `DefinedSpacecraft` `struct` also stores all of the additional parameters which the simulator tracks per-spacecraft such as solar array area, battery capacity, thermal heat values, etc. as well as all stepwise and lifetime accumulators for all output data products.

This method of accessing all objects through lists of `structs` is on its own not a severe problem, however it is compounded by the fact that FreeFlyer does not support arbitrary object handles or pointers. While each object does have its own unique "id" parameter set when it is instanced, this is not useful as a handle to access that object's parameters. The solution to this is to store all of the instances of a specific `struct` in a globally-accessible list of that `struct`. This "master list" can be traversed by a loop to act on every object for things like propagating all spacecraft or printing data products from each spacecraft. The list can also, most importantly, be indexed directly to access a single object as-needed.

This requires two considerations to work as a method of intended arbitrary access. First, each `struct` type that interacts with another `struct` type (such as a `DefinedSpacecraft` having several associated `ScheduledEvents`) must keep track of the *current* index of the object it is associated with. In practice, this means that each `struct` has one or more variables in it that store those indexes. The second concern is related to the first, specifically in that the master lists of these `structs` are just lists, they have no additional infrastructure to access them other than indexing. This means that if an object needs to be deleted from a list and that object is not already the last element, then the index of all of the objects after the deleted one will shift down by one. Any variables inside other objects which reference specific indexes of this list are now off by one, critically *not all of them are wrong*, only those which point to objects which occurred later in the list than the deleted element. To account for this, each time an object is deleted from a list, a procedure executes which checks for any objects occurring after the deleted object, traverses into any `structs` which point *into* that object, and corrects the index pointer variables for the list being updated.

Another area of concern for handling larger sets of spacecraft in FreeFlyer is that, as of this writing (FreeFlyer version 7.9.1), there is no support for multi-threading components of the script. While this can reduce the complexity of script development, it also severely bottlenecks the speed at which a simulation can be executed. This scales extremely poorly as the number of spacecraft increases, as each one must perform many calculations sequentially that could easily be performed in parallel, such as orbit propagation, thermal and power propagation, and propulsion calculations.

4.2.2 Handling Propulsion Maneuvering While Maintaining Lock-Step Timing Between Spacecraft

During certain propulsion maneuvers, when FreeFlyer is configured to use a fixed step-size integrator, it can sometimes propagate a spacecraft forward an amount of time other than the fixed step size. This would normally not be a problem however, this simulator requires that all spacecraft always exist at the same moment in time in order to properly calculate RFLink parameters. As such, additional code is required to check that two spacecraft trying to make a communications link are at the same epoch. If the RFLink object

detects a mismatch between the two spacecraft epochs, it throws an unrecoverable error and halts execution of the simulation.

To handle this problem gracefully, when two spacecraft are found to have mismatching epochs, the spacecraft whose epoch is further away from the global simulation time is stepped a non-fixed amount of time forward or backward to match with the spacecraft closer to the global time.

4.2.3 Need for a Scheduler

FreeFlyer handles propulsion events (i.e., thruster burns) by directly calculating their effects on the spacecraft's mass and velocity. FreeFlyer supports two different methods for modeling propulsion: impulsive burns, and finite burns. Impulsive burns are executed instantly and directly modify the velocity vector of the spacecraft. They are used to model maneuvers where the engine is highly impulsive and the burn duration is a relatively short fraction of the orbital period, such as with larger chemical thrusters. Finite burns are executed for a set duration, given engine parameters they compute the force applied to the spacecraft during that period and add that force to the force model of the spacecraft to be included in the propagation calculations. It is important to note that finite burns require the spacecraft to be stepped forward in order to take effect, while impulsive burns do not.

A "traditional" impulsive maneuver process in FreeFlyer that performs an orbit raising Hohmann transfer takes roughly the following form:

1. Step the spacecraft to its periapsis.
2. Call a targeting optimizer to determine the optimal impulsive burn parameters to raise the orbit periapsis to the target altitude.
3. Execute the maneuver.
4. Step the spacecraft to its apoapsis.
5. Call a targeting optimizer to determine the optimal impulsive burn parameters to raise the orbit periapsis to the target altitude.

While this is an acceptable process for situations where there is only one spacecraft present, or where automation is not in use, a significant problem lies in that *stepping the spacecraft forward in time* is required to find the optimal burns, which the simulator needs to do automatically. Because of this, the performance of a useful impulsive maneuver automatically pulls the spacecraft in question out of synchronous time from the rest of the simulation. This action breaks the communications tooling inside the RFLink object and is not allowed.

Finite burns are used to model electric propulsion systems, as they generally produce extremely small amounts of thrust, and often need to burn for weeks or months at a time. The primary way this is implemented in FreeFlyer is to execute a finite burn of a duration equal to the time step at each step of the simulation, until the desired orbital conditions are met. Both types of thrust maneuvers require the stepping of the spacecraft in time, but each requires a different implementation. Implementing the traditional methods for both types of burns would cause execution errors in FreeFlyer and would not work. The solution to this problem comes in the form of a task scheduler for the simulator.

The scheduler takes the form of a master list of all `structs` that contain information about each burn, such as the spacecraft it is associated with, the type of burn (impulsive or finite), the epoch the burn is

scheduled to occur at, an index reference for the burn parameters themselves (burn parameters are stored inside lists of the appropriate FreeFlyer burn objects inside each spacecraft `struct`), and a state variable to track if a burn has been executed.

When a spacecraft is found to have a difference between its desired orbital parameters (specified in the spacecraft definition file) and its actual orbital parameters of greater than the allowed difference (globally configured for the simulator), it calls the appropriate maneuver solver procedure. The maneuver solver procedures (described later) internally call procedures to schedule burn events as burn parameters are found. For impulsive burns this effectively means that those burns are pre-computed and then executed when the spacecraft reaches the desired epoch. For finite burns, only the next burn is solved for (i.e., the burn solver procedure is called once each step until that spacecraft achieves its target orbit). This behavior reduces confusion, as burns can only be executed by the scheduler at the time step closest to when they are scheduled to occur. The only way a spacecraft could have performed a burn is if the burn was scheduled already, and burns are only scheduled by the orbit solver procedures. This reduces complexity in debugging the relatively complex solver procedures. It is important to note that, due to the constant-size time step behavior set for the propagator, it is possible that scheduled burns will not execute at exactly the epoch they are scheduled for. The difference between the actual execution time and the scheduled execution time will fall between zero and one-half the step size. While this is not ideal for impulsive burns (whose optimality relies on execution at apoapsis and periapsis), it results in only slightly suboptimal behavior. This slightly conservative estimation is considered acceptable for this tool.

4.3 Implementation of Simulated Components

While FreeFlyer is a very powerful tool, it does not have out-of-the-box support for every subsystem of spacecraft design. As such, many of the core design areas in the system of systems required manual implementations. This necessitates the manual validation of each part component not provided through FreeFlyer directly.

4.3.1 Implementation of EPS Modeling

All simulated spacecraft generate electrical power through photovoltaic solar cells, storing electrical energy in chemical batteries. It is also assumed that all solar cells and arrays are managed by a maximum power point tracker, a device that adjusts the voltage and current being drawn from the solar array to maximize the instantaneous power output. Figure 4.5 illustrates the relationship between power, current, and voltage of a solar cell, and indicated the point of maximum power.

Figure 4.5 shows an example MPPT IV curve, where I_sc indicates the current if the terminals of the solar cell are shorted together, V_oc indicates the voltage if the terminals of the solar cell are left open. A circuit attached to the solar cell can be configured to operate at a voltage and current anywhere along the red line. However, because power is the product of voltage and current, there exists a point on the IV (red) curve where the power produced is the largest. This is the maximum power point. The purpose of an MPPT is to operate the cell at this point.

Solar cells experience performance degradation over time caused by radiation damage, surface contamination, UV damage to cover material, and impact damage from debris and micrometeoroids.[15] This damage is cumulative and is generally modeled as a fixed percentage degradation-per-year factor. The percentage of performance lost over a given time can vary significantly depending on factors such as material choice, flight

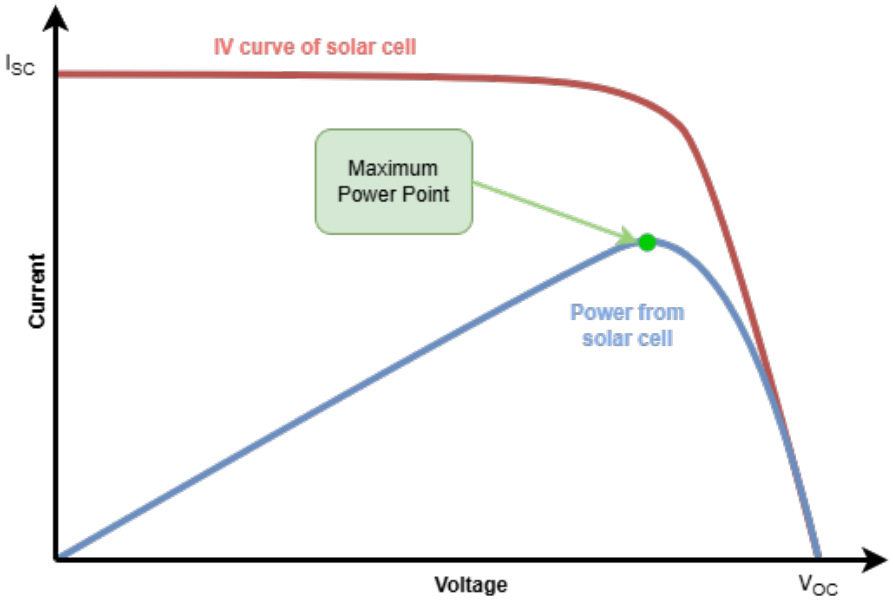

Figure 4.5: An example of a solar cell IV plot.

profile, and orbit characteristics. Real-world mission data analysis for the Mir and ISS programs has shown that annual degradation amounts to between 0.45% and 3.7%.[15][16] For array lifetime analysis, 3% per year is typically used as the solar array degradation parameter, as it provides an acceptably conservative estimate of damage without being unreasonably pessimistic. Mathematically, the array degradation is implemented as a scaling factor multiplied onto the ideal (non-degraded) solar power generated in the current timestep, shown in equation 4.1.

$$(1 - 0.03)^{MET} \tag{4.1}$$

The precise amount of luminous energy illuminating the solar array varies depending on the spacecraft attitude and location of the spacecraft in its orbit. This is both a factor of the array's angle of illumination and the spacecraft's distance from sun. The presence of any eclipsing planetary body can also intermittently block the sunlight. FreeFlyer has built-in instantaneous methods that can be called to determine a solar array's view of, relative position to, and distance from the sun.

For real spacecraft, significant work is undertaken during mission design to understand the pointing requirements of the solar arrays. For the purposes of this analysis, a FreeFlyer built-in function maintains the arrays pointed at the sun for maximum efficiency throughout their orbit.

Another area where ideal behavior affects the EPS system is in the power management and distribution systems. In real-world implementations, wires and power supplies have conversion inefficiencies (i.e., heat loss) when transporting or converting electrical energy. FreeFlyer deals with this by adjusting `busEPSBulkEfficiency`. This value can be set to one to assume ideal behavior, or it can be set to a lower value based on the assumed overall efficiency of the EPS system. The energy lost to inefficiency, is found by multiplying the subsystem energy use by `(1 - busEPSBulkEfficiency)`, and the resulting heat loss is added to the bus heat load in the next timestep.

After a check that the spacecraft is not in the shadow of the Moon or Earth, the solar flux is obtained

and multiplied by the solar array efficiency, the area of the solar array, the degradation factor (see equation 4.1), and the step size of the integrator in seconds. This results in the energy (in Watt-hours) that has been generated by the solar panels in the last timestep (Equation 4.2).

$$Q_{Collected} = \eta \cdot \phi_{solar} \cdot A \cdot \Delta t \cdot (1 - 0.03)^{MET} \cdot \frac{1}{3600} \tag{4.2}$$

$$Wh = Dimensionless \cdot \frac{\frac{J}{sec}}{m^2} \cdot m^2 \cdot sec \cdot Dimensionless \cdot \frac{Wh}{J} \tag{4.3}$$

where:

$Q_{Collected}$ is the solar energy collected in Watt-hours.
η is the solar cell efficiency.
ϕ_{solar} is the solar luminous flux on the spacecraft in $\frac{W}{m^2}$.
A is the solar array collection area in m^2.
Δt is the simulator time step size in seconds.
MET is the mission elapsed time in years.

Equation 4.3 is provided as a dimensional analysis to and verification of equation 4.2.

For this analysis, Lithium-Ion (Li-Ion) batteries provide power storage for use during periods of eclipse. Modern Li-Ion cells are very energy dense (e.g., commercial cells advertise upwards of 260 Watt-hours per kilogram[17]) and are capable of providing very good cycle lifetimes with low depth-of-discharge.[18]

All chemical battery cells exhibit performance degradation over their operating lifetimes. Generally, the most significant and impactful change in cell behavior is a reduction in energy capacity. A reduction in the storage capacity of a battery system over time is a significant concern for spacecraft mission design engineers, as most spacecraft will not undergo maintenance or servicing in their lifetimes. Spacecraft must be designed and launched with enough additional battery capacity to meet mission lifetime requirements. Unfortunately, batteries are often some of the heaviest components on a spacecraft, which places a significant cost burden on launching any more battery mass than is absolutely necessary.

Having a sufficiently accurate model of battery performance loss as a function of time is very important for determining an appropriate spacecraft battery size. Accurately modeling this behavior requires accurate models of the battery operating conditions, such as cell temperature, charge/discharge current, and calendar lifetime since manufacture [19]. However, for the purposes of the relative performance analysis being performed here, an estimate of the degradation of battery cell capacity can be inferred from the charge-discharge cycle count of the cell[18]. This is relatively easy to determine in real-time in a simulation, as the amount of energy used by the spacecraft over time is recorded, and the battery capacity is known. By dividing the total energy used by the actual battery capacity, the full-cycle-equivalent cycle count for the battery can be estimated.

One of the important factors in the cycle count of the battery is a quantity known as Depth of Discharge (DoD). The DoD is the fraction of the battery's total capacity that is actually discharged during a real charge-discharge cycle in orbit. This differentiates a *real* charge cycle from an *equivalent full* charge cycle. In a real charge-discharge cycle, energy is drawn from the battery when the spacecraft is in eclipse, but the battery is not fully discharged during this period, leaving some energy remaining as eclipse ends and charging can occur again. The DoD represents how deeply the battery was actually discharged during any cycle. For example, if during an orbit the battery experiences a maximum state of charge of 100%, and a minimum state of charge of 80%, then the depth of discharge is 20%. Because of the repetitive nature of orbital mechanics, if a spacecraft experiences a certain DoD over the course of an orbit, then it generally will experience a similar

DoD over each orbit in the near-term. Over the course of many orbits the battery capacity degrades and the DoD increases due to the real energy needs of the spacecraft not changing.

Battery DoD is an important factor in the preservation of battery capacity over long mission lifetimes. This is due to two factors; (1) a lower DoD means it takes more real cycles of the battery to reach one equivalent full cycle, and (2) charging and discharging the battery near the extremes of its capacity can increase the rate of degradation.[18][20][21]

For this analysis, battery degradation is calculated using a first-order model that applies a linear derating factor to the battery capacity of each spacecraft based on total full charge-discharge cycles. These charge discharge cycles are determined by continuously accumulating the total amount of energy consumed by the spacecraft, then dividing it by the non-degraded (ideal) capacity of the battery. This gives an approximation of the total full cycles on the battery at the current timestep.

The precise amount of capacity loss in the battery per charge/discharge should be configured based on the battery technology and degradation model being investigated. LG Chem MJ1 INR18650 battery cells represent a feasible choice of a battery with spaceflight heritage.[22] According to section 4.2.3 of the INR18650 specification[17], the most aggressive rate of degradation should be seen when the cell starts at the high end of its initial capacity (3500 mAh). After 400 standard full charge-discharge cycles the cell should retain an energy capacity of 80% of the minimum starting capacity. C_{min} is 3400 mAh, and 80% of that is 2720 mAh. This results in an overall worst-case capacity after 400 cycles of 77.71% (Eqn. 4.4), and a proportional capacity loss per-cycle of 0.0557% (Eqn. 4.5).

$$\frac{2720}{3500} = 0.7771429 \tag{4.4}$$

$$\frac{(1 - 0.7771429)}{400} = 0.00055714286 \tag{4.5}$$

Next, the number of total full-equivalent cycles is found and multiplied by the per-cycle proportional loss, which is then multiplied by the non-degraded capacity of the battery to yield the number of Watt-hours that have been lost so far. This value is subtracted from the non-degraded capacity of the battery (the capacity at the start of the mission) and the result, the current degraded capacity, is loaded into the battery capacity variable. The process for finding the current degraded capacity of the battery occurs each timestep.

In real systems, the flow of energy through a spacecraft is a continuous process. However, for the purpose of a digital simulation, this process is discretized using a fixed timestep. In the FreeFlyer simulation, each spacecraft sequentially steps through each subsystems' analysis scripts. During this process, the code that determines what each subsystem should be doing also determines the amount of power each system consumed in the last step.

For example, for the propulsion system, if the spacecraft is performing an engine burn, then the current propulsion system power variable, `STEP_propPower`, is set to the thruster power consumption specified in the spacecraft configuration file. If the spacecraft is not performing a maneuver, then `STEP_propPower` is set to zero.

For the spacecraft bus, the power consumption per-step is computed as a function of the dry mass of the spacecraft, as generally larger spacecraft require more power to operate components like attitude control systems. A second-order model approximation is used to map the spacecraft mass to a constant bus power consumption level. At each step the current bus power variable, `STEP_busPower`, is set to this constant value, except in cases where the spacecraft is out of power and has shut down, when it is set to zero.

The data generation subsystem, representing a payload on a data generation spacecraft, consumes power only when data is being generated, and the amount of power used is proportional to the amount of data produced. At every timestep, each DGS attempts to generate data. If it has sufficient power and is not too hot, it will perform a simple random check to determine if data is to be generated or not. This ensures that the duty cycle specified in the spacecraft configuration file is respected, but that oscillations in the simulator data flow are not accidentally induced. The amount of power consumed to generate this data is set by the spacecraft configuration file and is multiplied by the amount of data generated during the time step to determine the power consumption for the data system. This power is loaded into the same variable used to store the processor power consumption on DCS, `STEP_cpuPower`, because the generation/processing of data is considered part of the same system from an energy perspective.

Similarly, for each DCS with a processor payload, a step check determines that there is enough power to operate the payload, and that the temperature is not too high. When those checks are passed, the CPU processes the amount of data it can handle during this step (set by the spacecraft configuration file) and multiplies that data amount by the configured power consumption per bits per second. This resulting value is the amount of power consumed by the CPU during processing this step and is added to the `STEP_cpuPower` variable. Figure 4.6 illustrates the logic flow used to simulate the processor payload.

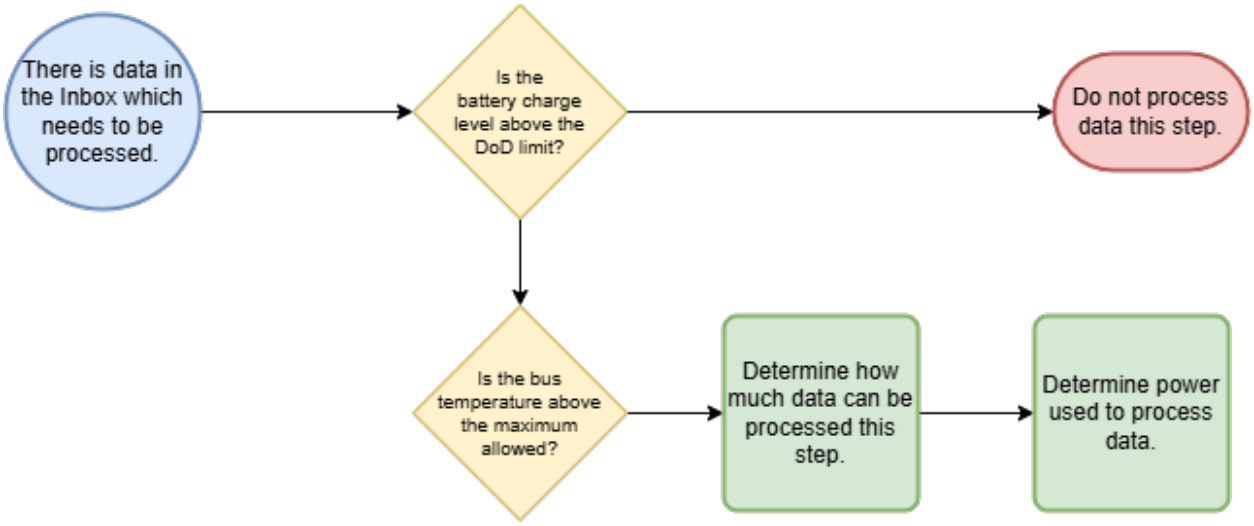

Figure 4.6: Logical flowchart showing the checks performed to determine if processing should be performed at this time.

The communications system consumes power based on the amount of data transmitted by it, as well as the distance between the transmitter and the receiver. Because the communications system calculates the actual channel capacity of the link each time a transfer is performed, and the simulation is used to determine design sizing, the energy used per-bit to transmit data is adjusted upwards in conditions where a link cannot be closed. This means that the transmit power specified in the initial configuration file can be increased, if needed to ensure the link is closed at all times. The transmit power is never adjusted downwards, so in conditions better than the worst-case, the higher power level simply results in better data rates. To find the amount of power the communications systems consumed on any given timestep, the amount of data transmitted is multiplied by the current `commISLPower` (or `commGNDPower` for space-to-ground links) variable, which stores the transmit power in watts per bit per second. The result is the amount of power consumed to

transmit data in the current step.

The thermal management system consumes electrical power through a heat pump. The heat pump actively maintains the temperature of the bus (and therefore the CPU within it) by switching on when the bus temperature rises too high, and pumping heat from the bus into the radiator. The amount of power that the heat pump consumes is specified by the spacecraft configuration file, and the amount of energy the heat pump transports is determined by the coefficient of performance (COP), which is based on the temperatures of the bus and radiator. When the heat pump is running, the variable `STEP_thermHPPower` is set to the heat pump max power level, `busHPMaxPower`, which is specified by the spacecraft configuration file.

The final consumer of electrical power is the inefficiency of the EPS system itself. The variable `busEPSBulkEfficiency` is used to set the overall efficiency of the EPS systems on each spacecraft. The value can be set to 1 for an ideal system, or between 0 and 1 for a non-ideal system. For every spacecraft, at each step, all the power consumed by the spacecraft is summed together and is divided by the `busEPSBulkEfficiency` value. This results in an increase in the power consumed from the spacecraft's power supply, as the subsystems will draw whatever power they need and do not care about the inefficiency of the EPS, so the inefficiency losses are seen as a higher load on the solar arrays/battery than would otherwise be expected.

Once all the power consumption from each subsystem has been accounted for, the total amount of electrical power consumed in the current step is stored in the `STEP_electricPowerUsed` variable. This, however, is power (Watts), not energy. Energy is found simply by multiplying `STEP_electricPowerUsed` by the simulation step size in seconds, yielding the number of Joules of energy consumed in the current step. Energy capacity of a battery is measure in Watt-hours, so Joules is divided by 3600 to yield Watt-hours. From here, the energy consumed in Watt-hours is subtracted from the battery variable, `TANK_BattLevel`, and added to the total discharge accumulator variable, `busBattDischargeAccum`, which is used to determine battery cycle count.

After all energy consumption is accounted for, the `solar_charging()` procedure is called. This procedure calculates the amount of energy collected from the solar arrays during the current timestep using the process described earlier. It also checks to see if the amount of energy collected exceeds the remaining capacity of the battery and, if so, the amount of energy collected in the current step (`STEP_busSolarCollected`), is set only to the amount of energy needed to fully charge the battery. This prevents the battery from being overcharged. From here, the total battery discharge accumulation is corrected based on the amount of energy which was taken from the battery specifically in the current step. Because energy consumption is determined prior to energy generation, the battery total discharged energy accumulator is set assuming that all power came from the battery. While the spacecraft is in sunlight this is not actually the case, so the discharge accumulator is corrected based on how much energy was generated by the solar arrays.

4.3.2 Implementation of Thermal System Modeling

Two behaviors dominate the transfer of heat between different parts of a spacecraft. Conduction through the structure of the spacecraft and its components, and radiation transferring heat into and out of the spacecraft.[23]

Due to the variability of designs for spacecraft structures and subsystems, it is beyond the scope of this work to build or implement a model of the conduction behavior inside the spacecraft in this simulation. Instead, the spacecraft thermal behavior is modeled using two bulk-heat control regions: the bus, and the radiator. The bus control region consists of the structure of the spacecraft, the fuel tanks, battery, ADCS, flight computer, processor/data generator payload, communication radios, and its own exterior thermal

insulation. The radiator control region consists of the deployable radiator itself.

This configuration leads to several assumptions related to the thermal behavior inside the different control regions: (1) It is assumed that all the systems inside a given control region are at effectively the same temperature. (2) It is assumed that very little passive heat conduction occurs between the bus and the radiator. (3) The waste heat produced during operation of the heat pump is not perfectly transferred to the radiator, as the heat pump has significant thermal contact with both control regions. The fraction of heat transferred back into the bus is configured by the global variable `heatPumpIsolation`. Figure 4.7 illustrates this concept. (4) None of the materials in the spacecraft undergo physical phase transitions in flight.

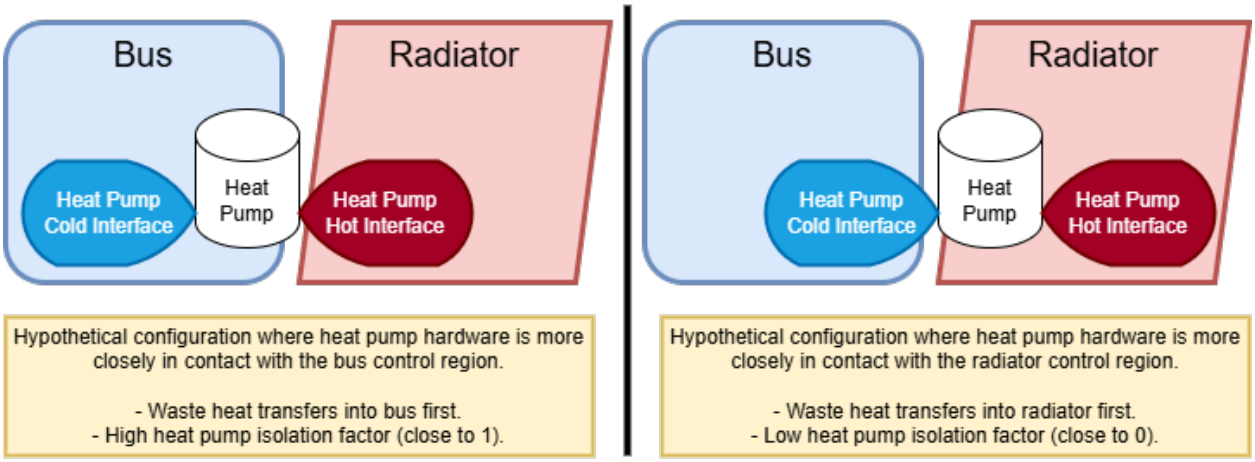

Figure 4.7: Comparison between hypothetical configurations he heat pump systems. One system has a heat pump in closer contact with the bus, leading to a high isolation factor. The other system has a heat pump in closer contact with the radiator, leading to a low isolation factor.

The relationship between the temperature and the heat present in a control region is driven by the specific heat of the materials used, and their masses. The radiator is the simpler of the two systems, it is assumed to be made of aluminum, and all radiators share a globally configurable thickness, density, and specific heat. The effective area of the radiator is defined on a per-spacecraft basis in the configuration file. Through these parameters, the relationship between the bulk heat present in the radiator control region and the temperature of the radiator becomes their product, as shown in equations 4.6 and 4.7.

$$Q_{rad} = c_{rad} \cdot m_{rad} \cdot t_{rad} = c_{rad} \cdot t_{rad} \cdot (d_{thick} \cdot A_{rad} \cdot \rho_{rad}) \tag{4.6}$$

$$t_{rad} = \frac{Q_{rad}}{c_{rad} \cdot m_{rad}} = \frac{Q_{rad}}{c_{rad} \cdot (d_{thick} \cdot A_{rad} \cdot \rho_{rad})} \tag{4.7}$$

where:

Q_{rad} is the thermal energy in the radiator in Joules.
c_{rad} is the specific heat of the radiator material in $\frac{J}{kg \cdot K}$.
m_{rad} is the mass of the radiator in kg.
t_{rad} is the temperature of the radiator in Kelvin.
d_{thick} is the radiator thickness in meters.
A_{rad} is the radiator surface area in m^2.

ρ_{rad} is the density of the radiator material in $\frac{kg}{m^3}$

For the bus, the system becomes slightly more complicated, as there are a variety of materials that contribute to the bulk heat capacity and mass. The primary contributors to the thermal mass of the bus control region are the spacecraft structure, the processor/data generator, fuel tanks, and the battery. All other subsystems are assumed to not significantly contribute to the overall mass of the bus and are therefore excluded from contributing to the bus thermal control region. The overall math is very similar to the radiator, but with multiple sets of mass and specific heat pairs, as seen in equation 4.8.

$$Q_{bus} = (c_{struct} \cdot m_{struct}) \cdot (c_{CPU} \cdot m_{CPU}) \cdot (c_{batt} \cdot m_{batt}) \cdot (c_{fuel} \cdot m_{fuel}) \cdot t_{bus} \tag{4.8}$$

where:

Q_{bus} is the thermal energy in the bus in Joules.
c_{part} is the specific heat capacity of the respective component's material in $\frac{J}{kg \cdot K}$.
m_{part} is the mass of the respective component in kg.
t_{bus} is the temperature of the bus in Kelvin.

The mass of the CPU is based on the performance throughput of the CPU (specified in the spacecraft configuration file), and the globally defined CPU specific heat, and CPU specific performance. The CPU mass is found in equation 4.9 and the justifying dimensional analysis in equation 4.10.

$$m_{CPU} = \texttt{cpuThruput} \cdot \texttt{cpuSpecificPerformance} \tag{4.9}$$

$$kg = \frac{bits}{second} \cdot \frac{kg}{\frac{bits}{second}} \tag{4.10}$$

where:

m_{CPU} is the mass of the processing payload in kg.
`cpuThruput` is the CPU processing rate, specified in the configuration file, in bits per second (bps).
`cpuSpecificPerformance` is the specific performance of the CPU, specified as a global configuration variable, in $\frac{kg}{bps}$.

It is not uncommon for spacecraft to use some form of active temperature control system to keep instruments and payloads from exceeding their operating temperatures. For the case of space-based computation systems, there is a significant need to keep the compute payload (or more broadly the spacecraft bus section, as it is a stated assumption that the temperatures of both the compute system and bus as a while are very similar) below the maximum operating temperature of the computers.

In the simulation, the transfer of heat from the bus control region to the radiator control region is done through an active heat pump. The heat pump behavior is captured in a set of global and spacecraft-specific parameters, that are discussed below.

Heat pumps are especially valued for their high performance, measured by the dimensionless value: the Coefficient of Performance (COP). The COP of a heat pump system measures the amount of heat that is moved for each unit of work put into the cycle. As both work and heat are forms of energy, measured in Joules, the COP shows the relationship between how many joules can be moved for each joule expended.[24] Critically, the COP can, and often does, significantly exceed one. In such a case the heat pump is able to move more energy than it costs to run.

The COP derives from a combination of the First Law of Thermodynamics[25] and the nature of a heat pump cycle. Because the heat pump cycle is closed, the relationship shown in equation 4.11 holds.

$$Q_H + Q_C + W = \Delta U = 0 \tag{4.11}$$

where:

Q_H is the heat of the hot reservoir in Joules.
Q_C is the heat of the cold reservoir in Joules.
W is the work done during the cycle in Joules.
ΔU is the change in internal energy of the heat pump system in Joules.

Because of the closed-loop nature of the cycle, the internal energy does not change.

$$W = -Q_H - Q_C \tag{4.12}$$

$$|Q_H| = -Q_H \tag{4.13}$$

Equations 4.12 and 4.13 show the rearrangement of equation 4.11. The work done is equal to the difference in the heat in the reservoirs. This leads to equation 4.14, the COP for an ideal heat pump.

$$COP = \frac{Q_H}{Q_H - Q_C} \tag{4.14}$$

However, this is not in a form that is immediately useful. Borrowing some of the work that Planck published,[26] COP can be written in the more useful form shown in equation 4.15

$$COP = \frac{T_H}{T_H - T_C} \tag{4.15}$$

where:

T_H is the thermodynamic temperature of the hot reservoir in Kelvin.
T_C is the thermodynamic temperature of the hot reservoir in Kelvin.
COP is the Coefficient of Performance, and is dimensionless.

It is important to note here that this relationship is for an *ideal* heat pump operating at perfect Carnot efficiency. In the real-world systems the observed COP can be dramatically lower than ideal. In the simulation, the COP can be calculated ideally, and a derating factor applied such that it is configured globally using the variable `heatPumpDerating`.

Another area where the non-ideal behavior of the heat pump is of note is in the thermal isolation between the heat pump and radiator (hot side) and the spacecraft bus (cold side). As these two sets of components are mechanically coupled, and both directly interface with the heat pump itself as the most significant mode of interaction between the two control regions, there is some amount of conduction of heat "backwards" from the heat pump back into the bus. This is modeled by diverting a fraction of the heat pump's power use back into the bus as a heat load. The fraction of heat sent back into the bus is controlled globally by the variable `heatPumpIsolation` which unless otherwise specified, is 0.1, meaning 10% of the heat pump's power consumption goes back into the bus instead of directly into the radiator.

For the purposes of this simulation, a simple bang-bang thermostatic control is implemented to control the bus temperature. When the CPU temperature exceeds the globally configured `cpuTargetTemperature`, and there is sufficient energy stored in the battery such that the depth-of-discharge limits are not violated, the heat pump turns on and consumes `busHPMaxPower` electrical power. The amount of heat pumped for the

power consumed is based on the COP, which is updated at each step based on the current temperature of the bus and radiator.

In the Earth orbit environment, there are several ways in which heat can enter the spacecraft. All objects with a temperature above absolute zero emit some energy in the form of blackbody radiation. This is a form of infrared radiation that carries some of the heat energy away from the object. In the Earth-orbiting environment there are four primary external sources of heat through radiation: (1) direct sunlight; (2) sunlight reflected off of the Earth (albedo); (3) blackbody radiation from the Earth surface and atmosphere; (4) blackbody radiation from deep space.

Direct sunlight is modeled using FreeFlyer's internal `SolarFlux` function to identify the real-time solar luminous flux, the same heating value from sunlight is used. If the spacecraft were made of some ideal blackbody material, one that absorbs all light that hits it, then all of the solar flux would indeed be absorbed as heat.

Heating from the Earth's reflection of sunlight (albedo) is only relevant during parts of the spacecraft's orbit where it is in view of the sunlit side of the Earth. Additionally, there is some consideration required for the angle of reflection of the sunlight off the planet's surface. Near the terminator there is far less luminous flux incident per-area on the planet's surface than if the spacecraft were directly between the Sun and Earth. As such there is a cosine factor involved in the computation of incident albedo on the spacecraft. When the spacecraft is either behind Earth or otherwise on the dark side of the terminator, the reflected sunlight (albedo) is zero. When the spacecraft is on the illuminated side of the terminator, the calculations in equation 4.16 is performed to roughly determine the incident flux from Earth albedo.

$$\phi_{\oplus Albedo} = -cos(\angle SVE) \cdot \phi_{\odot} \cdot Albedo \tag{4.16}$$

where:

$\phi_{\oplus Albedo}$ is the radiant flux incident on the spacecraft due to Earth's albedo, in $\frac{W}{m^2}$.
$\angle SVE$ is the instantaneous Sun-Spacecraft-Earth angle.
$\phi_{\odot}$ is the radiant flux from the Sun, in $\frac{W}{m^2}$.
Albedo is the Earth's mean albedo (reflectivity) value.

Albedo is a globally configured variable containing the measured mean albedo value of Earth, 0.297.[27] Using the negative cosine in this way leads to the coefficient being zero when the spacecraft is over the terminator (when $\angle SVE$ is 90 degrees), and one when the spacecraft is overhead at solar noon (when $\angle SVE$ is 180 degrees). Both the cosine and albedo terms are dimensionless, leaving only $\frac{W}{m^2}$ and $\frac{W}{m^2}$. The described angle is illustrated in figure 4.8.

Blackbody radiation from the Earth, while also a source of heat traveling from the surface to the spacecraft, is roughly uniform throughout the spacecraft's orbit. This is because the surface of the Earth does not instantly become as cold as deep space the moment sunlight is not present. For the purposes of this analysis, the Earth is modeled as having a uniform surface emissivity and a uniform temperature, specifically the 2023 mean surface temperature of 288 Kelvin[28], and an emissivity of 0.95.[29] This emissivity value was chosen to be at the higher end of the observed range, which skews the simulation slightly towards a less ideal thermal condition for the spacecraft, where it receives more heat than it would in real life. This effect is marginal, but helps to make the simulator overall somewhat more conservative.

The last source of incident heat on the spacecraft is blackbody radiation from deep space. While in terms of net heat transfer deep space is generally a sink, here the heat loads into and out of the spacecraft are being considered separately in their direct quantities, not as a net combination. The actual heat load from deep

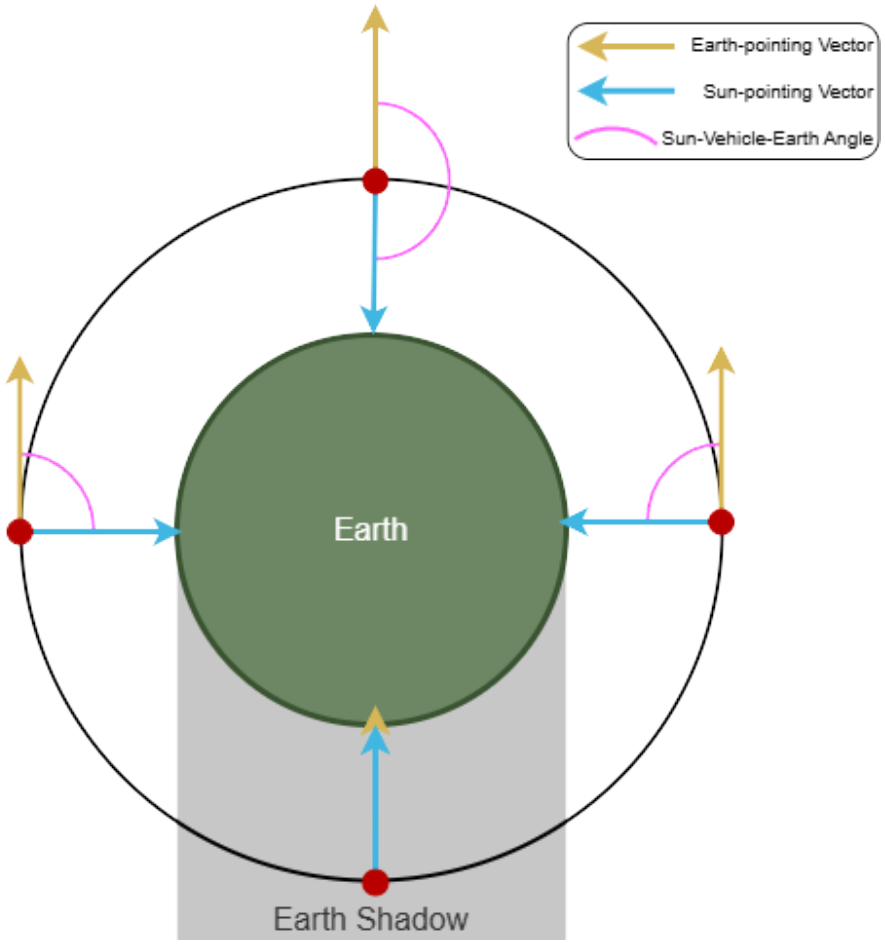

Figure 4.8: Illustration of the Sun-Vehicle-Earth $\angle SVE$ angle (pink), where the spacecraft is represented by the red dot at four positions in its orbit. The yellow arrow is a vector pointing towards the sun, and the blue arrow is a vector pointing towards the Earth.

space is generally very low, as the temperature of deep space is very low at 2.7 Kelvin[30], however this heat source is always present and approaches the spacecraft from all directions which are not occluded by celestial bodies, so it cannot be fully ignored. As this is simply blackbody radiation, the Stefan-Boltzmann law (Eqn. 4.18) is used to determine the flux and incident power. This is shown in equation 4.17.

$$P_{absorbed} = \sigma \cdot t_{deep\_space}^4 \cdot A_{bus} \cdot \epsilon \tag{4.17}$$

where:

$P_{absorbed}$ is the thermal power absorbed by the spacecraft from deep space, in Watts.
σ is the Stefan-Boltzmann constant (5.670374e-8 $\frac{W}{m^2 K^4}$)
t_{deep_space} is the temperature of deep space in kelvin.
A_{bus} is the surface area of the spacecraft in m^2.
ϵ is the emissivity of the radiating surface.

While these four external sources of heat are all incident on the spacecraft, each spacecraft (regardless of type, data center or data generator) is modeled thermally as two distinct bulk components, the bus and the radiator. While the specific shape of these components is not modeled, some assumptions have been made

regarding their general geometry. First, it is assumed that the bus is roughly a box or prismatic volume in shape, where its surface area to internal volume ratio is relatively lower than that of the radiator. Second, the radiator is assumed to be roughly a flat planar structure, where its surface area to volume ratio is relatively higher than that of the bus. With these two general assumptions the attitude of the bus structure itself does not significantly change the bus' thermal conditions, as the visible cross-sectional area of the bus from the perspective of different sources of heat is invariant.

However, because the radiator is assumed to be flat, its attitude in space does have a significant effect on the cross-sectional area exposed to heat sources. This means that it is desired to point the two large faces of the radiator out to deep space and point the edges of the radiator towards larger sources of heat like the sun and Earth. This is illustrated in figure 4.9.

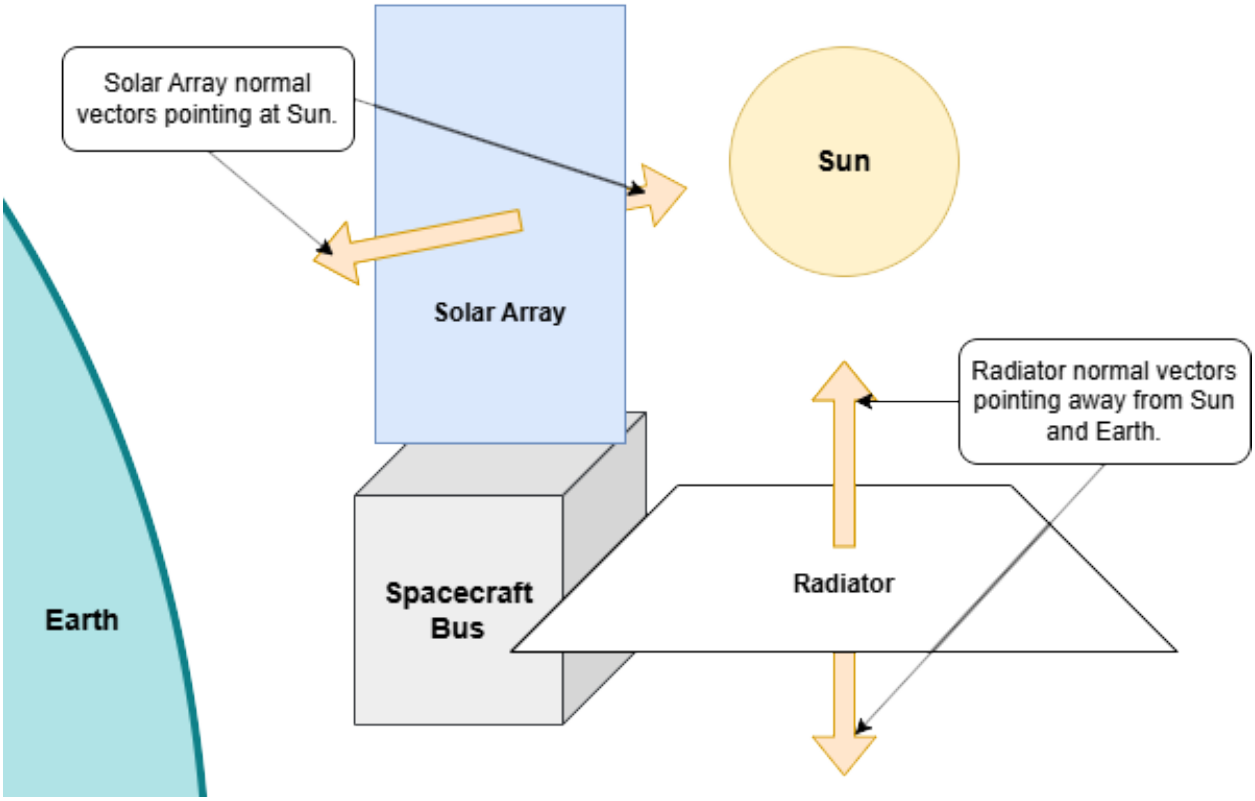

Figure 4.9: Diagram showing general ideal orientation considerations for radiator (white plane), solar arrays (blue plane), and bus (gray cube) relative to sun (yellow circle), and Earth (teal body). The normal faces of the solar array and radiator are indicated (orange arrows). Note that the radiator is set edge-on to both the Earth and the sun, with both its faces pointing to deep space. Note also that the solar array is oriented with the generating face pointed towards the sun, and the inverse face pointed towards deep space.

Blackbody radiation is the mechanism by which heat is removed from the spacecraft. The amount of power radiated through this mechanism is related significantly to the temperature of the object in question. Specifically, this is governed by the Stefan-Boltzmann law, and takes the form shown in equation 4.18.

$$M = \epsilon \sigma T^4 \tag{4.18}$$

where:

M is the radiant exitance in $\frac{W}{m^2}$.
ϵ is the emissivity of the radiating surface.
σ is the Stefan-Boltzmann constant (5.670374e-8 $\frac{W}{m^2K^4}$)
T is the temperature of deep space in kelvin.

Of note here is that the amount of power radiated from an object increases with temperature to the fourth power, so significant increase in the power emitted from the spacecraft can be achieved if the heat pump system is used to increase the temperature of the radiator above the equilibrium temperature of the spacecraft.

It is important to remember that the amount of power going into the spacecraft and the amount of power emitted from the spacecraft do not influence each other. An object moved from shadow into direct sunlight does not radiate less heat when it is in the sun, it simply absorbs more heat than it is emitting, so the temperature increases until the heat emitted matches the heat absorbed and equilibrium is reached.

For the purposes of the simulation, for each time step, the current temperature of the bus and radiator is calculated, based on the bulk heat stored and the relative mass of the different constituent materials present. The total power radiated outwards is then computed using the Stefan-Boltzmann law, the cross-sectional area of the bus or radiator (per-spacecraft, from the configuration file), and the ratio between the cross-sectional area and the surface area of the bus (globally configured using the `busSurfaceAreaToCrossSection` variable). For the radiator, because of the flat geometry, the cross-sectional area is simply multiplied by two because the radiator has a face on each side.

Once this is done, the size of the time step is known, and the power emitted for the bus and for the radiator is known, allowing the computation of the energy emitted in the last time step for the bus and for the radiator. These values are then subtracted from their respective bulk heat variables and their temperatures re-computed.

The net accumulation of heat inside the spacecraft must also be addressed. Adding to this are heat sources operating inside the spacecraft itself, such as computers, batteries, and payloads. On a spacecraft, nearly all of the electrical energy used onboard is turned into heat inside the bus. Some limited exceptions exist, including energy radiated out of the communications systems, and kinetic energy electric propulsion systems, however these generally make up a relatively small (and easily accounted for) fraction of the total electrical energy used.

For the purposes of this simulation, the calculation itself is relatively straightforward. Because the thermal power being absorbed by the bus and radiator has already been computed, and because the electrical power consumption for the bus is already known, and because the simulation steps through time in discrete increments, the power by the time per-step can be multiplied to get the energy moved in that step. From there the energy is added, subtracted and moved around per figure 4.10.

The absorbed external heat, heat pumped from the bus, and the heat of the heat pump itself (scaled by the isolation factor discussed earlier) is added to the radiator control region heat variable. The bus systems waste heat, payload waste heat, communications waste heat, electric propulsion waste heat (simply the propulsion electrical power scaled by the thruster efficiency), the non-isolated heat from the heat pump (1 – the isolation factor for the heat pump), the absorbed external heat for the bus, and the waste heat due to distribution inefficiency in the EPS system are all added to the bus control region heat variable, as described in equation 4.19.

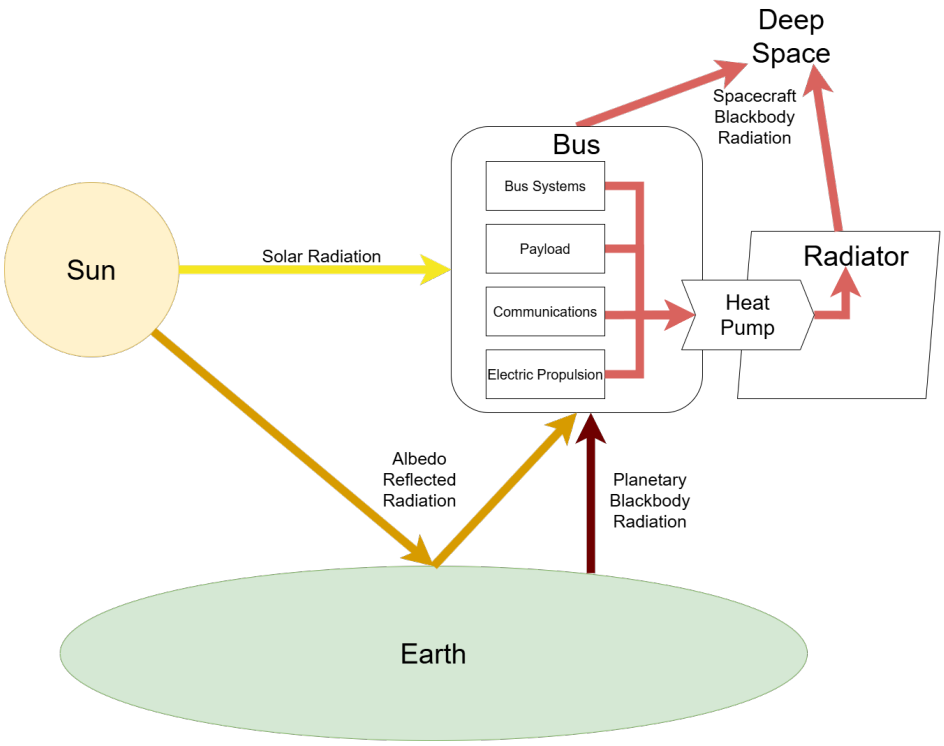

Figure 4.10: Diagram illustrating the net flow of heat through the simulation.

$$
\begin{aligned}
Q_{bus} \mathrel{+}= \Delta T \cdot (&P_{CPU} \\
&+P_{ISL\_TX} \\
&+P_{GND\_TX} \\
&+(P_{prop} \cdot (1 - \eta_{EP\_Thruster})) \\
&+P_{bus} \\
&+(P_{heat\_pump} \cdot \texttt{heatPumpIsolation}) \\
&+P_{absorbed} \\
&+P_{inefficiency\_losses})
\end{aligned}
\tag{4.19}
$$

where:

Q_{bus} is the heat stored in the bus in Joules.
ΔT is the step size of the simulation in seconds.
$\eta_{EP\_Thruster}$ is the efficiency of the electric propulsion system (if present).
P_{part} is the electrical power used by the given system this step,in Watts.

The amount of heat pumped by the heat pump, which is determined based on the COP of the pump at the current radiator and bus temperatures, is known already, and is subtracted from the bus heat variable. The last step is to update the total heat variable, which is the sum of the radiator and bus heat storage variables.

4.3.3 Implementation of Communications System Modeling

The design of digital radio-based wireless communications systems is a well-understood problem. At its most basic, the design of a wireless communications system is governed by the Shannon-Hartley theorem. Shannon-Hartley addresses the relationship between capacity of an information channel (measured in bits per second), bandwidth (the range of frequencies in use to transmit the signal, measured in Hertz), and the signal-to-noise ratio of the channel (a dimensionless linear power ratio).[31] Formally, the Shannon-Hartley theorem is shown in equation 4.20.

$$C = Blog_2\left(1 + \frac{S}{N}\right) \tag{4.20}$$

where:

C is the channel capacity in symbols per second.
B is the bandwidth in Hz.
S is the average signal power in the passband.
N is the average noise power in the passband.

Fundamentally, this is a straightforward problem, to maximize the channel capacity bandwidth and/or signal power can be increased, and noise decreased. However, the design problem is complicated by practical and regulatory realities. Because of this, in nearly all radio communications systems, bandwidth use and transmit power are tightly regulated by government bodies to ensure that the limited bandwidth that exists is allocated most effectively between government, commercial, and public use.

Signal power is observed by the receiver and there are several methods available to increase this value. The most direct method is to simply increase the field strength of the transmitted signal by using a more powerful amplifier. The directivity of the transmitting and receiving antennas can also be improved.

An antenna is a device that smoothly transitions RF signals between free-space propagation and propagation through a feedline of some form (coaxial cable and hollow waveguide being the most common). Mathematically ideal antennas (while technically impossible for coherent waves) are known as "isotropic antennas" and as the name implies, radiate and absorb equal power in/from all directions. Isotropic antennas are directional (the ratio of how much an antenna radiates power in a specific direction over the radiated power averaged across all directions). Isotropic antennas do not increase received signal power. However, by adjusting the physical geometry of an antenna, its directivity can be improved. For an antenna that is being driven by a transmitting device, a higher directivity means the radiated energy is being concentrated more into a specific direction. For an antenna which is receiving a signal, a higher directivity means it is better at absorbing energy from a specific direction than others. Note that all antennas are reciprocal and can simultaneously receive and transmit RF power.

The final component that can be optimized is the noise power observed at the receiver. Noise power is essentially all of the RF energy the receiving radio sees that is *not* the signal desired. This noise can come from many sources, but the most common offenders are other radio transmissions (both intended and spurious), and thermal noise introduced by the fields propagating through, or interacting with, any material media which is not at a temperature of absolute zero. Figure 4.11 illustrates the major components of an RF communications link.

The space environment imposes additional constraints on the design of wireless communications systems. Generally, wireless communications is used to send data both into and out of a spacecraft. For a traditional stand-alone spacecraft these segments would be called 'uplink' and 'downlink', where uplink describes sending

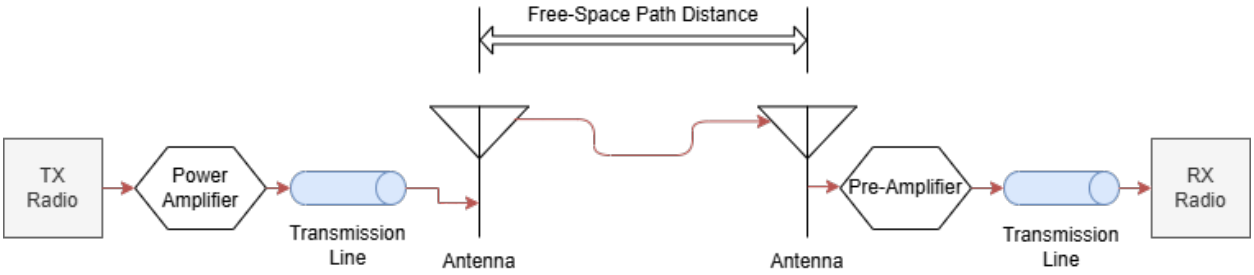

Figure 4.11: A diagram showing the basic components of a wireless communications system. The purpose of the power amplifier is to serve as the main source of drive power for the signal. The purpose of the pre-amplifier is to compensate for losses and noise introduced by the transmission line between the receiver and the receiving antenna.

data up from the ground to the spacecraft, and downlink refers to sending data down from the spacecraft to the ground. In most cases, downlink is the more constrained communications segment. This is due to the relative ease and lower cost of improving ground station hardware, especially power amplifiers. It is relatively easy to increase the gain of a power amplifier on the ground, as mass, power consumption, and launch costs are not major constraints on its acquisition and use.

For most spacecraft missions designed in the current launch economic environment, mass to orbit is the largest constraint on all components of spacecraft design.[5] For communications this primarily affects the power amplifier and antenna system. In order to increase the performance of the transmitting power amplifier on the spacecraft, the device by its nature must increase its power consumption. In order to accommodate this increase in power consumption, the solar arrays, EPS distribution systems, and batteries must be increased in their design capacities. This results in a significant increase in spacecraft launch mass for each marginal increase in transmit power.

The spacecraft communications antennas are less severely constrained by mass. This is because they do not consume significant amounts of other spacecraft resources, so other systems generally do not need to be upsized to accommodate increased antenna sizes. Antennas are primarily constrained in their mass, and the volume they take up when stowed for launch. Conveniently, a relatively good RF reflector can be made with relatively little material, so reflector-based antenna designs can be made relatively low mass for their size. As shown in equation 4.21, the gain of an antenna increases with the reflector diameter squared.

$$G = \left(\frac{\pi d}{\lambda}\right)^2 e_A \tag{4.21}$$

where:

G is the gain of the antenna in dBi.
d is the diameter of the parabolic reflector
λ is the wavelength of the operating frequency of the antenna.
e_A is the antenna aperture efficiency.

This means that it is a good idea to design a large reflector antenna which can fold up into something small. Such antennas have already been developed and have a rich flight heritage in space.[32] Often they come in the form of canister parabolic antennas, which operate somewhat like an umbrella unfolding, illustrated in figure 4.12.

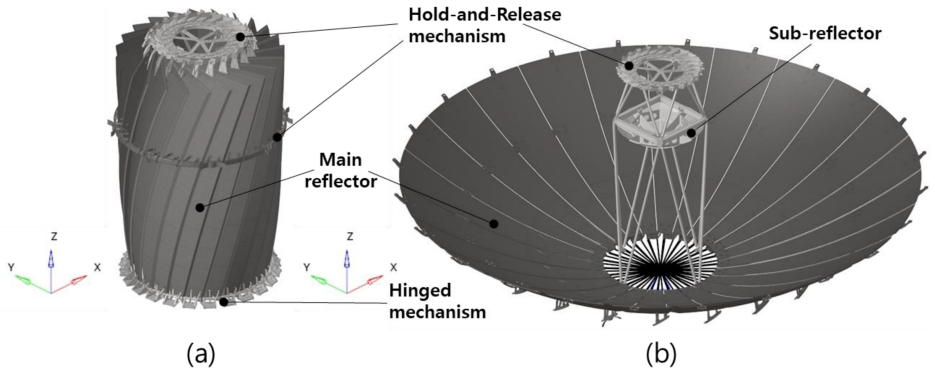

Figure 4.12: Illustration of a deployable parabolic antenna for spacecraft. Image credit to: [32].

The last significant constraint for spacecraft communications design is that of bandwidth use. Although this has been changing with the recent deployment of mega-constellations[33], generally there are significant constraints on non-governmental use of space-allocated bandwidth. This is due, generally, to the fact that spacecraft move around all over the world, so unlike nearly every other RF transmission source they cannot be reliably contained to a specific geographic area and effectively consume bandwidth globally. As such, regulators must treat spacecraft with increased scrutiny with regards to allocation of bandwidth. Further exacerbating the problem is that, while the distances are higher between transmitting spacecraft, there is also nothing in between many spacecraft to absorb or block unwanted interference power, so each spacecraft which is transmitting is also increasing the effective noise floor for other spacecraft in its band. Due to these factors, bandwidth is at a premium in space and is carefully allocated in restrictive quantities. Compensation for low bandwidths must be accommodated through other parts of the link.

Establishing communications links between arbitrary spacecraft in orbit for the purposes of this simulation is not something which can be calculated in advance. As such, a significant amount of the simulation code for communications exists in service of creating, destroying, updating, and handling the different communications links between each spacecraft. The simulator holds all information about each link in a `struct` called `CommsLink` containing: the link type (ISL or space-to-ground), the transmitting spacecraft and antenna, the receiving spacecraft and antenna, a VisibilitySegment object (a FreeFlyer object used to determine distance between two objects, as well as any occlusions), an RFLink object (a FreeFlyer object used to handle RF and communications math), and a Vector object (a FreeFlyer object used to render the links graphically when LIVE_UPDATE is enabled). When a new link is created between two antennas the objects are instanced and assigned, and the `struct` is appended to a global list of all `CommsLink structs`, which acts similarly to a stack. During link creation certain variables inside the transmitting and receiving spacecraft `structs` are also updated to point to the appropriate index of the `CommsLink` global list. Care is taken to ensure that whenever an object is removed from any such "stack-like" list, pointer indexes of all list elements after the removed element are updated to reflect the change in pointer value. This is necessary because FreeFlyer does not directly support object reference handles or pointers, so the only way for objects created at runtime to interact with each other is to store them all in global lists and reference their list index.

Spacecraft are defined in the configuration file with a fixed maximum amount of data storage space. This is dynamically split between two buffers, an "inbox" and an "outbox." Figure 4.13 illustrates the relationship between the inbox and outbox buffers on a DCS.

For DGS, all generated data is immediately loaded into the outbox, so the outbox is effectively allowed to

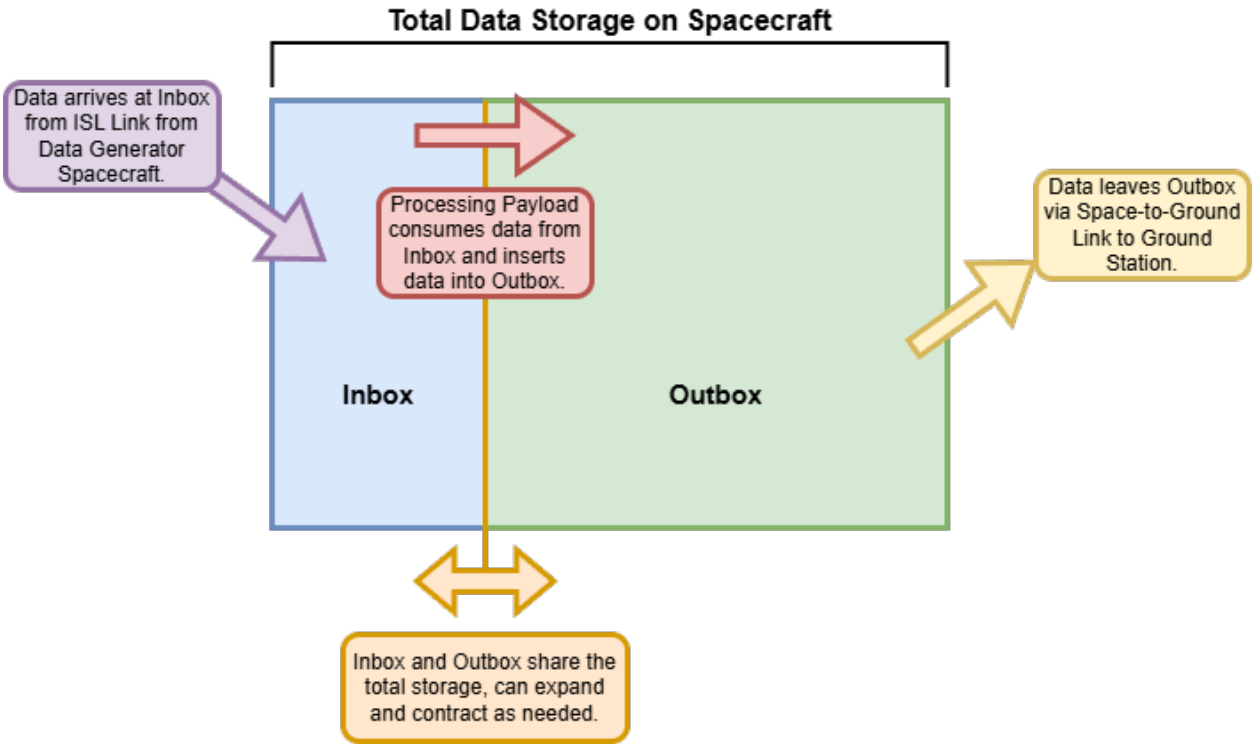

Figure 4.13: Illustration of the relationship between the inbox and outbox buffers on a DCS.

use the entire drive capacity. When a spacecraft has data in its outbox, it attempts to create a link with the nearest spacecraft of the correct type with an available antenna. DGS must link to DCS, they cannot link directly to the ground. Only DCS can link to the ground to transmit their outbox data. Once a link has been created, the receiving spacecraft is tasked to reach through the link and pull data out of the sender's outbox and into the receiver's inbox. The illustration in figure 4.14 explains this process.

FreeFlyer has a built-in tool for calculating RF channel parameters and communications link values. This tool is used by an object called the 'RFLink.' When given all of the appropriate system parameters, it performs the task of solving for the actual link behavior. When given antenna parameters, distance between the targets, bandwidth, known losses, temperatures, and minimum margins, it calculates the real antenna gains, noise powers, the effective isotropic radiated power (EIRP), the free-space path loss (FSPL), the received power, and the signal-to-noise ratio (SNR).

As implemented in the simulator, the first step performed once a link is established is to use the RFLink tool to determine the SNR of the link. This is accomplished by calling the method `GetSignalToNoiseRatio()` on the RFLink between the two spacecraft in question. Internally, this method finds the EIRP of the transmitting spacecraft, and the FSPL, to compute the received power. It then compares that to the noise power calculated at the temperature of the receiver (globally configured for the entire sim).

EIRP is a conceptual tool used to abstract the transmitting side of a link into a single value. Essentially it combines the gain of the transmitting antenna with the power send into that antenna and reports the power that would be needed to drive an isotropic antenna in order to look the same from the receiver's perspective. In this way, the receiver does not need to consider what the transmitter power or antenna characteristics are. It is computed with equation 4.22.

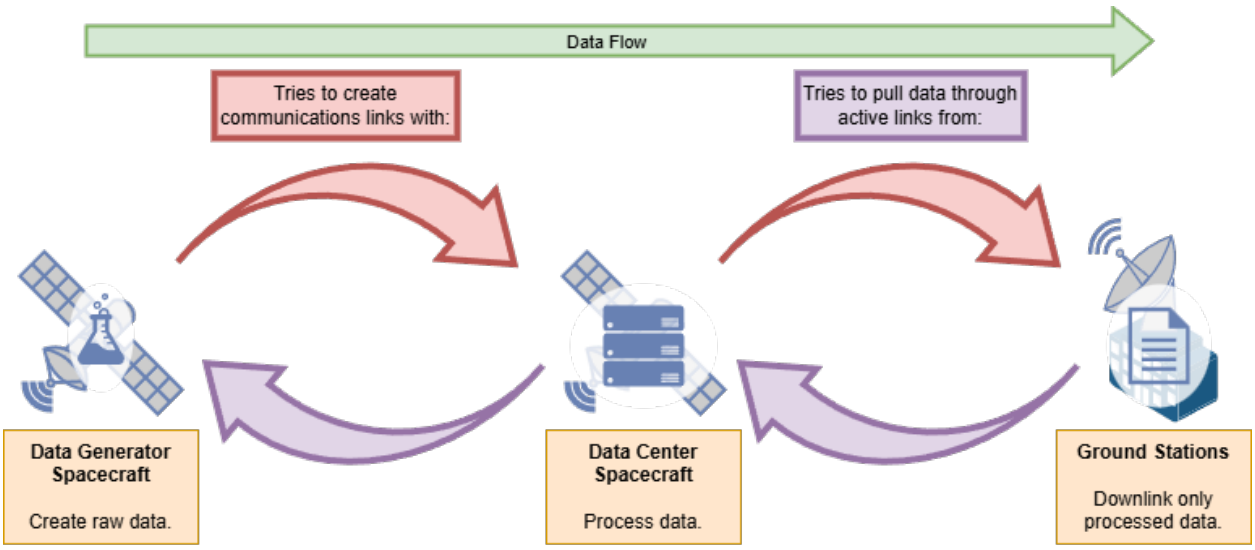

Figure 4.14: Diagram showing which types of link segments are created by which spacecraft, and which spacecraft are responsible for pulling data through active links.

$$EIRP = 10log(P_t) - L_{line\_TX} - L_{point\_TX} + G_{ant\_TX} \tag{4.22}$$

where:

$EIRP$ is the effective isotropic radiated power in dB.
P_t is the transmitter power in watts.
L_{line_TX} is the transmitter feedline loss in dB.
L_{point_TX} is the transmitter pointing loss in dB.
G_{ant_TX} is the real antenna gain in dBi.

The free-space path loss of the system is a function of distance between the spacecraft antennas. It is described in equation 4.23.

$$L_{fp} = 20log\left(\frac{4\pi R}{\lambda}\right) \tag{4.23}$$

where:

L_{fp} is the free-space path loss in dB.
R is the path distance in meters.
λ is the wavelength of the center frequency in Hz.

Received power is computed by summing the EIRP, FSPL, receiver antenna gain, receiver feedline loss, receiver polarization loss, and receiver atmospheric loss (zero in space). It is described in equation 4.24.

$$P_r = EIRP + G_{ant\_RX} - L_{fp} - L_{line\_RX} - L_{polar} - L_{atm} \tag{4.24}$$

where:

P_r is the received power in dB.
$EIRP$ is the effective isotropic radiated power in dB.

G_{ant_RX} is the gain of the receiving antenna in dBi.
L_{fp} is the free-space path loss in dB.
L_{line_RX} is the receiver feedline loss in dB.
L_{polar} is the polarization mismatch loss in dB.
L_{atm} is the atmospheric loss in dB (zero for ISL communications).

The last sub-component needed of SNR is the noise power of the receiver. This is calculated using the bandwidth of the receiver and a noise temperature, which is defined globally for the simulator. The calculation for noise power is shown in equation 4.25, and equation 4.26 combines noise and receiver power to find the SNR.

$$P_N = 10log(kT_NB) \tag{4.25}$$

where:

P_N is the noise power in dB.
k is Boltzmann's constant (1.380649e-23 $\frac{J}{K}$).
T_N is the noise temperature in Kelvin.
B is the bandwidth of the signal in Hz.

$$SNR = P_r - P_N \tag{4.26}$$

where:

SNR is the linear signal to noise power ratio in dB.
P_r is the received power in dB.
P_N is the noise power in dB.

Once the SNR is known from the RFLink object, the current channel capacity in bits-per-second is calculated using the Shannon-Hartley theorem, discussed above in equation 4.20. Once the channel capacity is found, it is multiplied by the number of seconds in the time step to find the actual number of bits which have been transferred between the two spacecraft in this time step. Finally, that number of bits is subtracted from the transmitting spacecraft's outbox and added to the receiving spacecraft's inbox.

The simulator allows the user to configure a behavior called "Dynamic Transmit Power." Able to be set separately between all ISL links and all space-to-ground links, this option allows the simulator to increase the transmit power of the communications radios to compensate for lower-than-desired datarates. If the calculated real-time link data rate is lower than the maximum specified in that spacecraft's configuration file, and the appropriate dynamic transmit power setting is enabled, then the spacecraft finds the difference between the current link margin and the minimum link margin (globally configured), and scales the spacecraft's transmit power variable by the amount required to meet the minimum margin. This functionality calculates what transmit powers would be needed to ensure that the target datarate is always achieved throughout the mission lifetime.

4.3.4 Implementation of Propulsion System Modeling

The mathematics that are used to derive the n-body equations of motion implemented in FreeFlyer reduce to a form with ten constant values. Three values for the linear momentum vector of the system, three for the velocity vector of the center of mass of the system, three for the angular momentum vector of the system, and one for the total energy of the system. These ten constants are the only constants of motion for the n-body

problem, but an issue arises almost immediately when attempting to apply this solution. In a two-body system there are 12 degrees of freedom. Three components each are needed to characterize the position and velocity vectors for both bodies. Therefore, it is not possible to derive a closed-form analytical solution to the n-body problem for n $\geqslant$ 2. This type of mathematical system is considered chaotic, meaning that even a small change in the initial conditions can lead to dramatically different resulting conditions. Conveniently, this chaos accumulates over time, so for situations where the conditions haven't changed considerably from their initial state, results are reliable.

To determine the position of a spacecraft with a useful level accuracy at some point far into the future, a numerical approach must be used. This entails repeatedly solving the equations of motion for the spacecraft in over short steps time, such that the error introduced in each individual step is minimized. Careful selection of the length of this time step is necessary to balance the need for accurate results with the real time required to compute each step.

FreeFlyer provides several different mathematical propagators to select from when configuring a spacecraft. Due to the relatively large constant time step selected in this simulator (30 seconds, but configurable globally), the RungeKutta8(9) (RK8(9)) propagator was selected as an appropriate choice for all spacecraft in the simulator. The RK8(9) propagator is the most accurate propagator FreeFlyer provides, at the cost of requiring more time to execute. In practice the compute time required to propagate the spacecraft was not the most significant contributor to simulator speed, so the use of a faster propagator was not required.

To accurately compute a spacecraft's position far into the future, an accurate account of all forces acting on the spacecraft must be tallied. The dominant force on any spacecraft is, of course, gravity from the primary body it orbits however, small perturbing forces are still present in most situations that need to be accounted for. For this purpose, FreeFlyer directly supports the modeling of solar radiation pressure, drag, lift, and non-uniform gravitational fields (i.e., lumpy gravity).

Gravity is the primary force acting on spacecraft in orbit. For the earth, the gravitational force pulls the spacecraft towards the center of the planet. The magnitude of this force is described in equation 4.27.

$$\overrightarrow{F_b} = \frac{G \cdot M \cdot m}{r^2} \hat{r} \tag{4.27}$$

where:

$\overrightarrow{F_b}$ is the gravity force vector on the spacecraft towards the planetary body in Newtons.
G is the universal gravitational constant (6.673e-11 $\frac{Nm^2}{kg^2}$).
M is the mass of the planetary body (Earth) in kg.
m is the mass of the spacecraft in kg.
r is the distance between the spacecraft and the center of the planetary body in meters.
$\hat{r}$ is the unit vector pointing from the spacecraft to the center of the planetary body.

For each large body involved in the orbital propagation of the spacecraft, this calculation is performed to obtain the force vector of gravity towards the bodies' equivalent point mass. In the simulator, the forces from Earth, the Moon, and the Sun are all calculated in this manner in the propagator.

While the force of gravity from the Sun and Moon are significant enough to merit inclusion in the propagation each step, they are sufficiently far away that they can be modeled as point masses, and they are assumed to have uniform gravity fields. This assumption does not hold for the Earth, as it is the central body all of the spacecraft are orbiting, so modeling the perturbing effects of non-uniformities Earth's gravity field is necessary to accurately assess the lifetime of and fuel requirements of the spacecraft over long mission timeframes. The modeling process for lumpy gravity fields follows a similar process to modeling all

perturbations. The instantaneous force vector acting on the spacecraft from this component is integrated to compute the spacecraft dynamics. In FreeFlyer, this is determined based on the latitude and longitude of the spacecraft over the planet as well as its altitude. Both are used to find the gravitational field anomaly at that location above Earth. By default, FreeFlyer uses the EGM96 gravity model[34], and this simulator is configured to use that model with 360 zonal and 360 tesseral terms. This provides an extremely high degree of accuracy when determining the perturbation due to non-uniform gravity around Earth.

Solar radiation pressure is the force imparted on a spacecraft by being illuminated with sunlight. While small, this force is continuous while the spacecraft is in sunlight and does have a perturbing effect on the spacecraft's orbit. The instantaneous force of solar radiation pressure is described in equation 4.28.

$$\overrightarrow{F_{srp}} = \frac{S}{c} \cdot C_R \cdot A \cdot \hat{r}_{ss} \tag{4.28}$$

where:

$\overrightarrow{F_{srp}}$ is the solar radiation force vector on the spacecraft away from the Sun.
S is the solar flux incident on the spacecraft in $\frac{W}{m^2}$.
c is the speed of light in $\frac{m}{sec}$.
C_R is the bulk coefficient of reflectivity of the spacecraft.
A is the cross-sectional area of the spacecraft which is illuminated by the Sun in m^2.
$\hat{r}_{ss}$ is the unit vector pointing from the sun to the spacecraft.

The reflectivity coefficient of the spacecraft is a value ranging from zero to two, which captures information about the bulk momentum exchange between the spacecraft and light. A C_R of zero indicates total transparency, a value of one indicates total absorption (such as with a blackbody surface), and a value of two indicates a perfect reflection. Typically this value is between 1.2 and 1.5.[35]

The last two important orbital perturbations are both interactions with the atmosphere: lift and drag. Lift is an aerodynamic effect observed in spacecraft more often at lower altitudes where they appear to "bounce" off the upper atmosphere instead of continuing to descend as a purely drag-based model would expect. Modeling lift is handled by the FreeFlyer propagator and is described in equations 4.29, 4.30, 4.31, and 4.32.

$$\overrightarrow{v} = \overrightarrow{v_{spacecraft}} - \overrightarrow{\omega_E} \times \overrightarrow{R} \tag{4.29}$$

where:

$\overrightarrow{v}$ is the atmosphere-relative velocity vector of the spacecraft.
$\overrightarrow{v_{spacecraft}}$ is the spacecraft's Earth-relative velocity vector.
$\overrightarrow{\omega_E}$ is the vector of Earth's spin axis.
$\overrightarrow{R}$ is the spacecraft's Earth-relative position vector.

$$\overrightarrow{U_L} = \overrightarrow{v} \times \left(\overrightarrow{R} \times \overrightarrow{v} \right) \tag{4.30}$$

where:

$\overrightarrow{U_L}$ is the non-unitized lift force vector of the spacecraft.
$\overrightarrow{v}$ is the atmosphere-relative velocity vector of the spacecraft, see Eqn. 4.29.
$\overrightarrow{R}$ is the spacecraft's Earth-relative position vector.

$$\hat{U}_L = \frac{\overrightarrow{U_L}}{|\overrightarrow{U_L}|} \tag{4.31}$$

where:

$\hat{U}_L$ is the lift force unit vector of the spacecraft.
$\overrightarrow{U_L}$ is the non-unitized lift force vector of the spacecraft, see Eqn. 4.30.

$$\overrightarrow{L} = C_L \cdot A_L \cdot \rho \cdot \frac{1}{2} \cdot \overrightarrow{v}^2 \cdot \hat{U}_L \tag{4.32}$$

where:

$\overrightarrow{L}$ is the lift force vector acting on the spacecraft.
C_L is the coefficient of lift of the spacecraft.
A_L is the cross-sectional area of the spacecraft used to calculate lift in m^2.
ρ is the density of the atmosphere at the spacecraft's current location in $\frac{kg}{m^3}$.
$\overrightarrow{v}$ is the spacecraft's current tangential velocity relative to the atmosphere in $\frac{m}{sec}$.
$\hat{U}_L$ is the lift force unit vector of the spacecraft.

Modeling drag is less complex than lift. The drag force is caused by collisions between the spacecraft and molecules of gas in the atmosphere. It is modeled as described in equation 4.33.[36]

$$\overrightarrow{D} = C_D \cdot A_D \cdot \rho \cdot \frac{1}{2} \cdot \overrightarrow{v}^2 \cdot \hat{U}_D \tag{4.33}$$

where:

$\overrightarrow{D}$ is the drag force vector acting on the spacecraft.
C_D is the coefficient of drag of the spacecraft.
A_D is the cross-sectional area of the spacecraft used to calculate drag in m^2.
ρ is the density of the atmosphere at the spacecraft's current location in $\frac{kg}{m^3}$.
$\overrightarrow{v}$ is the atmosphere-relative velocity vector of the spacecraft in $\frac{m}{sec}$, see Eqn. 4.29.
$\hat{U}_D$ is the drag force unit vector of the spacecraft, which is directly opposite the velocity vector of the spacecraft.

Both drag and lift utilize associated drag and lift coefficients. These values represent the complex geometric and physical properties of the spacecraft and how it interacts with the atmosphere. Due to the highly variable nature of spacecraft design and atmosphere-relative orientation during operations, the determination of these values is not given treatment here.

Modeling Impulsive Thrusters

Thrusters with a relatively high thrust output that burn for a relatively short period of time, are modeled with the `ImpulsiveBurn` FreeFlyer object. This built-in object manages the timing and performance parameters of the maneuver. When a maneuver is called for by a spacecraft, the `ImpulsiveBurn`'s `BurnDirection` vector is instantly added to the spacecraft's velocity vector. This process also automatically subtracts fuel burned from the associated thruster's fuel tank based on the configured thruster parameters. Fuel tanks and thruster objects are automatically created and configured by the simulator during import of the spacecraft configuration file.

The spacecraft definition file allows the user to specify both a set of initial orbital parameters (such as the orbit after release from the launch vehicle) and a set of desired orbital parameters (the operational orbit of the spacecraft). Generally speaking, for impulsive maneuvers between two in-plane, low-eccentricity orbits, the Hohmann Transfer is the most fuel-efficient approach. To this end, the simulator can automatically solve arbitrary Hohmann transfers to achieve a desired semi-major axis and eccentricity.

When a spacecraft is found to have its semi-major axis or eccentricity too far outside of the desired value, the simulator calls the `solve_burn_impulsive()` procedure. This procedure takes a snapshot of the given spacecraft's current state, then enters what is referred to as the "twilight zone." In the twilight zone, the spacecraft can be maneuvered and stepped through time asynchronously from the rest of the simulation. The simulator then executes a simple Hohmann transfer process:

1. The spacecraft is propagated to its periapsis, or if the orbit is currently circular, it is propagated to a point greater than one timestep into the future.
2. An optimizer is run to determine the optimal impulsive burn parameters to move the opposing side of the orbital ellipse to touch the desired orbital ellipse.
3. This maneuver is executed.
4. The spacecraft is stepped to the other side of its orbit.
5. An optimizer is run again to determine the optimal impulsive burn parameters to match the desired orbit semi-major axis and eccentricity.

Once this process is complete, each of the burns that was solved for is scheduled into events in the scheduler at their appropriate epochs. Once this is done, the spacecraft is reset to its initial snapshot point, thus exiting the twilight zone and rejoining synchronous operation with the rest of the simulation. In this manner, each spacecraft with an impulsive thruster can automatically solve for a near-optimal transfer to its target orbit without breaking the lock-step timing required for communications and data collection to work.

It is important to note that a truly optimal Hohmann-like maneuver set requires burn executing at exactly the apoapsis or periapsis of the orbit. However, the simulator and the scheduler's burn execution procedure operate on a fixed time step. This means that the desired burn execution time and the actual burn execution time may deviate slightly, by at most half a time-step. This results in a small amount of error however, this is considered acceptable because the orbital "good enough" precision levels are user-configurable globally. Further, slightly imperfect burns make the simulation slightly more conservative in its results, which is generally considered good.

Modeling Finite-Time Thrusters

Thrusters with a relatively low thrust output have a different set of considerations when planning maneuvers. It is not uncommon for small electric propulsion devices to need to burn continuously for weeks at a time in order to achieve the desired orbit. To get sufficient utility out of the fuel used for such maneuvers, the thrust vector must be redirected at each timestep[37]. FreeFlyer models these maneuvers using a `FiniteBurn` object. `FiniteBurns` are engine burn maneuvers where the exact thrust parameters are fixed and the burn is executed over a finite amount of time, instead of being applied instantly. To model long-duration, low-thrust maneuvers, such as those of a spacecraft with an electric propulsion system, a `FiniteBurn` event can be executed each time step, for the same duration as the time step size, and where the thrust vector is pointed in the direction which will generate the optimal (or nearly optimal) change in orbital parameters for the current position in the orbit.

In this mode it is not compute-efficient to pre-compute and pre-schedule all the burns required to drive a spacecraft from its initial orbit to its desired one, as this could easily result in scheduling thousands of events for each spacecraft. Instead, spacecraft with electric propulsion systems that are not currently in their

desired orbit have the `solve_burn_finite()` procedure called. This procedure executes the steps outlined in Ruggiero et. Al. 2011[37][38] for matching semi-major axis and eccentricity. It performs these calculations only for a single burn, scheduled for a single time step in the future. Once that burn is executed and the simulator has stepped forward one timestep, `solve_burn_finite()` is once again called, and the process is repeated until the spacecraft in question is within tolerance of its target orbit.

4.3.5 Implementation of Compute and Data Payload Modeling

While the overall purpose of this simulation tool is to help model different constellations of space-based data centers, the generation of the data moving between and through these spacecraft is important to the model. Ensuring that the quantity and rate of data generated is close to what is realistically expected is extremely important when sizing communications systems, and data processing components.

While the actual data generated by different types of instruments can vary, the overall behavior of each instrument is known and predictable. For example, a pushbroom-style optical image sensor records a strip of pixels repeatedly as it passes over the ground. The cross-track ground resolution per-pixel is determined by the altitude of the spacecraft, the field of view of the optics system, and the number of pixels in the imager as described in equation 4.34 and figure 4.15.

$$Resolution = 2\frac{Altitude \cdot tan\left(\frac{FoV}{2}\right)}{Pixels} \tag{4.34}$$

$$\frac{m}{pixel} = \frac{m \cdot tan(degrees)}{pixel} \tag{4.35}$$

where:

$Resolution$ is the smallest feature detail able to be resolved by the imager, in $\frac{m}{pixel}$.
$Altitude$ is the distance between the surface of Earth and the imager, in meters.
FoV is the field of view of the imager optics in degrees.
$Pixels$ is the quantity of pixels across on the sensor.

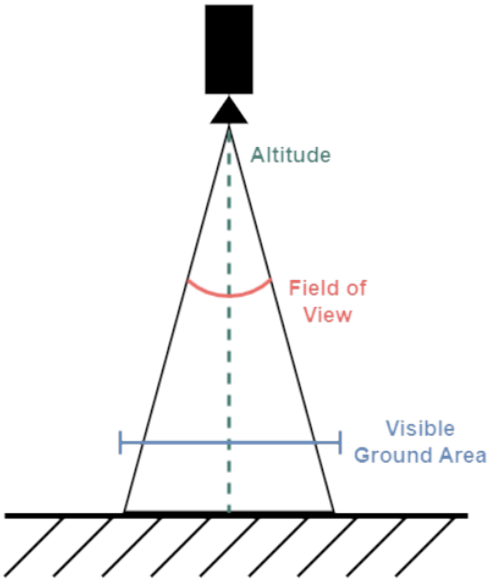

Figure 4.15: Diagram showing the two-dimensional geometry of a pushbroom sensor's view and resolution.

Each pixel generates multiple bits of data per ADC capture to keep quantization error low, often between 12 and 18 bits. The number of channels used to capture an image in multiple spectral frequencies, such as the three required for color images (or additional bands for infra-red, ultraviolet, and beyond), linearly scales the amount of data captured per image. All together this provides a good approximation for the rate of data generation, as shown in equation 4.36.

$$datarate = bands \cdot 2\frac{Altitude \cdot tan\left(\frac{FoV}{2}\right)}{Resolution} \cdot \frac{bits}{pixel \cdot capture} \cdot \frac{captures}{second} \tag{4.36}$$

where:

$datarate$ is the rate of data generation from the imager in bps.
$bands$ is the quantity of distinct spectral bands the imager can observe.
$Resolution$ is the smallest feature detail able to be resolved by the imager, in $\frac{m}{pixel}$.
$Altitude$ is the distance between the surface of Earth and the imager, in meters.
FoV is the field of view of the imager optics in degrees.
$\frac{bits}{pixel \cdot capture}$ is the ADC resolution of per-pixel, in bits.
$\frac{captures}{second}$ is the rate of image captures per second.

Depending on the nature of the data and the processing work required to obtain insight from raw data, the amount the data will "shrink" as it is processed will vary. It is important to note that this is not data compression, which is a method of repackaging data in a more efficient manner without (generally) losing any information in the process.

It is assumed that the overall effect of this processing on the data is a shrink of the quantity of data from the inbox of a DCS to the outbox. This shrink factor is set per-spacecraft in the `cpuDataShrink` field.

4.3.6 Implementation of Structure Model

In the FreeFlyer simulation tool, spacecraft bulk dry mass is specified on a per-spacecraft basis in the mission configuration file. This value is used directly as dry mass for orbital propagation and propulsion tasks. For thermal analysis purposes, the mass of the bus structure alone, `busStructMass`, is found by scaling the dry mass by the globally defined parameter `structureMassFraction`. This allows the user to estimate how much of their mass allocation will be structure alone for thermal analysis purposes.

4.3.7 Flight Management Computer

Similarly to the ADCS system, the flight computer exists purely to manage the other spacecraft subsystems. It has mass and consumes a relatively small amount of power continuously but does not otherwise change its behavior. As such, there is little to model in this simulation tool. The flight computer's mass is assumed to be included in the spacecrafts dry mass, and its power consumption is included in the bus power consumption calculation. Beyond these accommodations, it is assumed to work as designed for the duration of the spacecraft's lifetime.

4.3.8 Attitude Determination and Control Systems

The attitude determination and control system ensures that the spacecraft is pointed in the correct direction according to its mission and design requirements. This includes behaviors such as pointing the solar array at the sun, pointing the radiator panels to deep space, pointing the propulsion system towards the desired thrust vector, and pointing communications systems or instruments towards their intended targets.

While precise planning and design of such systems is important at a spacecraft design level, it is not necessary to simulate at the mission-design level. Such situation-specific design typically occurs later in the spacecraft development phase. As such, the simulator assumes that all spacecraft have implemented an ADCS systems that meets all operational requirements.

The dry mass of the spacecraft is assumed to include the ADCS system, and the bus electrical power consumption is calculated based on that mass. This accommodates the increased power consumption from ADCS associated with increased spacecraft mass.

4.4 Validating Simulated Components

With the simulator's underlying algorithms defined and implemented in FreeFlyer, modeling performance validation is performed. The validation of the simulator follows a first-principles approach for each subsystem. It must be shown that the data the simulator creates matches what the underlying physics dictates.

4.4.1 Validation of EPS Modeling

Validation of the EPS system is performed starting with a set of assumptions about the basic energy use behavior of each system. First, it is assumed that the spacecraft is using solar photovoltaic panels to generate electricity. Second, it is assumed that the spacecraft is storing electrical energy using Lithium-ion batteries. Third, it is assumed that the bus subsystems are operating in a high-power mode continuously, producing an EPS system solution that is generally oversized for the application at the beginning of life (BOL). This

conservative estimate is helpful for determining the upper bound of the size of the system, and therefore the upper bound on mass and cost fractions.

The photovoltaic energy collection system can be validated by comparing the energy generated by the program with manual calculations for an example situation. Starting with the total light flux at Earth's average distance from the sun, 1,361 $\frac{W}{m^2}$ [39], it is known that the chosen solar cells are 31.8% efficient at the beginning of their lifetime[40].

Assuming that the cells are operated at their maximum power point, an electrical power output of $0.318 \cdot 1361.0 = 432.798\ \frac{W}{m^2}$ of solar cell area should be observed. The test configuration specifies 10 m^2 solar arrays, so 4,327.98 watts should be drawn out of the cells when they are illuminated. At the 30 second timestep that the simulation uses, $30 \cdot 4,327.98 = 129,839.4$ Joules of energy should be collected each timestep when the spacecraft is recharging batteries and operating at the conditions where maximum power is drawn from the solar array. Indeed, in the test configuration, it is observed that 129,987.5 Joules were generated in a single simulator timestep during maximum draw (Figure 4.16).

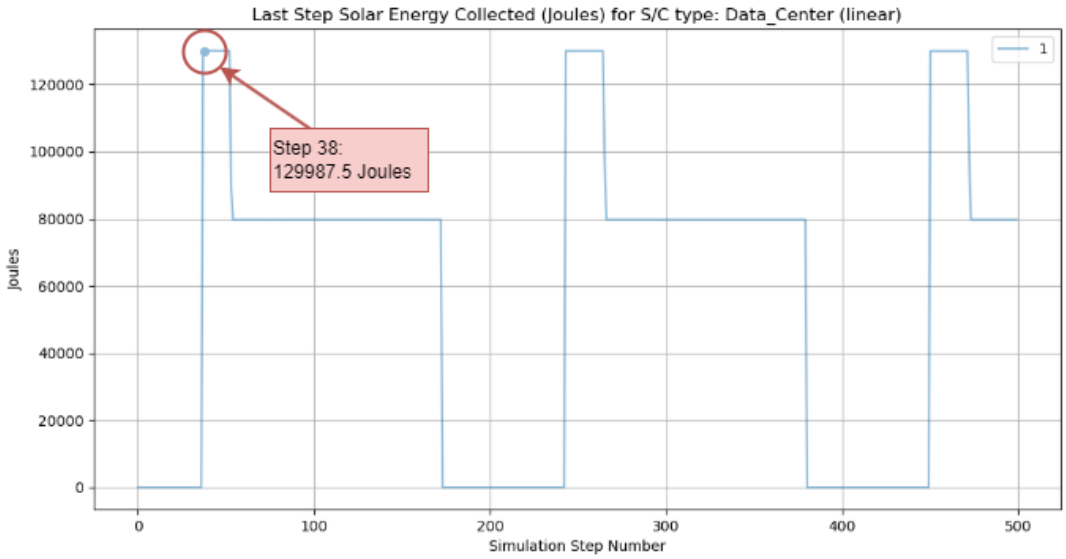

Figure 4.16: Energy output from the solar array in the prior timestep shown for a DCS with a 10 m^2 solar array at beginning of life. At step 38, the first step where the spacecraft was in sunlight and generating energy, the generated energy from the cells was 129,987.5 Joules. The small deviation can be accounted for by the fact that FreeFlyer uses the exact current distance from the Sun to calculate the solar luminous flux, instead of the mean distance of Earth.

To clearly show the degradation behavior expected as the solar cells age, the test configuration is set to demonstrate an unrealistically extreme 90% degradation rate per year. The purpose of this is to more easily observe that the rate of decay matches what is expected by amplifying its effect. This equates to a degradation factor of $d = (1 - 0.9)^{MET}$ where d is the degradation factor and MET is the mission elapsed time in years.

For one month of operation the result is $d = 0.1^{0.0833} = 0.82540$. This is a 17.46% degradation in performance and should result in a solar generation energy of $129,839.4 \cdot 0.8254 = 107,169.4$ Joules. In fact, this is what is observed at the end of the one-month test simulation (Figure 4.17).

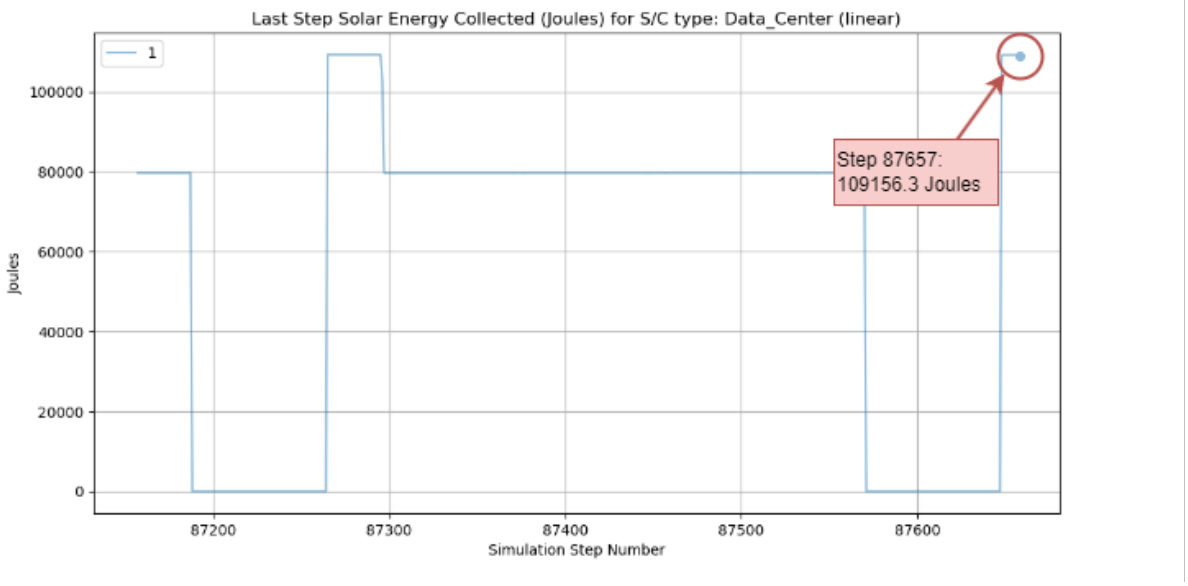

Figure 4.17: Energy output from the solar array in the prior timestep shown for a DCS with a 10 m^2 solar array at end of one month with a solar degradation rate of 90% per year. At the point where the spacecraft was last in sunlight and generating energy, the generated energy from the cells was 109,156.3 Joules. The deviation from the expected quantity can be accounted for by FreeFlyer using the exact current distance from the Sun to calculate the solar luminous flux, instead of the mean distance of Earth.

To determine that the simulation is correctly calculating the degradation of the battery as it is used, the battery cycle counts and effective capacity over a longer mission time can be observed and compared to the expected rated behavior of the cells. Due to the significant number of timesteps required before each spacecraft reaches 400 charge-discharge cycles (a benchmark value provided by the battery cell datasheet[17]), the data output from the simulation is culled to output data every 100 timesteps. The timestep number after which 400 cycles have occurred, and the battery capacity at that timestep, is then recorded for ten identical spacecraft that are spread out uniformly in a series of LEO orbits. The mean of both values is then taken across all ten spacecraft. Table 4.2 displays the data from the simulation.

According to the specifications of the battery cell used in the simulator, after 400 charge-discharge cycles the capacity of the cell should be no less than 77.7% of the original minimum rated capacity[17]. The ten spacecraft were all initialized with 1200 watt-hours of battery capacity. At 400 cycles the battery capacity has become $1200 * 0.777 = 932.4$ Whr. The observed data has an extremely small standard deviation across battery capacity at this point, which indicates that the simulation is self-consistent. Using the mean battery capacity of 932.56 Whr, the simulation is computing an effective degradation rate at these points of: $932.56/1200 = 0.777$, a negligible difference from the target value.

Validating the energy flow through the system can be done by observing the energy used by each subsystem in a single step, combining the total consumed energy with the energy added to the battery, and then comparing that to the total energy generated by the solar array. For any given step, the amount of energy being generated should be equal to the amount of energy being consumed and/or stored.

When observing output data for the test spacecraft in eclipse, the following is expected:

Spacecraft ID	Steps to reach 400 cycles	Battery capacity (Whr)
15	1889900	932.553
16	1897000	932.551
17	1900400	932.565
18	1903300	932.551
19	1907300	932.569
20	1894800	932.562
21	1912200	932.554
22	1896500	932.558
23	1880200	932.569
24	1898800	932.565
Mean:	**1898040**	**932.56**
STD:	**8476.7**	**0.0068**

Table 4.2: Ten identical spacecraft are uniformly distributed across LEO orbits, the simulation timestep at which they accumulated 400 total cycles, and the battery capacity of the spacecraft at that time step.

1. The power used by each subsystem, including inefficiencies, add up to match the stepwise power consumption total.

2. All of this power is being drawn from the battery, so the energy used in the last step should be equal to the difference between the battery charge in the last step and in this step.

Several timesteps have been selected from a period of eclipse in the test configuration results to validate the total energy consumption. These timesteps were chosen such that they varied in the power consumption of different subsystems, such as the propulsion system and heat pump, in order to provide a useful representation of different operating scenarios. (Table 4.3)

Step	Bus Power	CPU Power	Propulsion Power	Heat Pump Power	Radio TX Power	Inefficiency Losses	**Expected Total**	**Observed Total**
1470	76.56	0.0	0.0	0.0	34.60	3.44	114.60	114.60
1632	76.56	0.30	2000.0	500.0	0.0	79.70	2656.56	2656.56
7179	76.56	0.0	0.0	500.0	57.02	19.60	653.17	653.17

Table 4.3: All power units are in Watts. This table clearly demonstrates that the simulator correctly tabulates the total power consumed by the spacecraft in a given timestep, which meets the first expectation.

For each of the same time steps, the battery charge level should also decrease by the expected amount (Table 4.4).

Step	Battery Charge Last Step	Current Battery Charge	Energy Discharged from Battery	Energy Used
1470	611.2247	610.2697	0.9550	0.9550
1632	997.7910	975.6531	22.1380	22.1380
7179	639.9979	634.5548	5.4431	5.4431

Table 4.4: All energy units are in Watt-hours. This table demonstrates that the simulator is accurately removing energy consumed by the spacecraft from the battery in the test case. Together this validates that the simulator is behavior as expected in conditions where a spacecraft is not generating solar power.

When observing output data for a spacecraft in sunlight, the following is expected:

1. The difference between the power consumed and the power generated by the solar arrays should be equal to the amount of power being used to charge the batteries.

2. When the batteries are fully charged, the amount of power being generated by the solar arrays decreases to only supply what is being consumed by the active systems.

Once again, several time steps have been tabulated to demonstrate the expected behavior across different phases of the simulation.

Step	Power Consumed in Last Step (W)	Power Generated by Solar Arrays (W)	Battery Power (W)	Observed Change in Battery Charge (Wh)	Expected Change in Battery Charge (Wh)	Flight Conditions
4850	78.926	0.0	-78.926	0.6577	0.6577	In Eclipse, no solar power.
4862	78.926	4308.617	4229.688	35.247	35.247	Battery too low for heavy electrical load, mostly charging.
4871	2656.246	4308.647	1652.400	13.770	13.770	Charging battery as well as running high-load systems.
4883	2656.246	2656.246	0.0	0.0	0.0	Battery full, running all systems on solar power alone.

Table 4.5: As the test configuration demonstrates, both expected conditions are achieved by the simulator. The first expectation validated is that the battery charge power equals difference between the power consumed by the spacecraft systems and the power generated by the solar arrays. The second expectation validated is that the solar arrays only generate the power consumed by the spacecraft systems after the battery is fully charged.

4.4.2 Validation of Thermal System Modeling

Much of the thermal analysis is predicated on being able to reliably calculate the temperature of both control regions (the bus and the radiator), given their material makeup and the bulk heat energy stored in them. In the simulation code this is performed in both directions with a procedure called `updateTemperature()`. This FreeFlyer procedure is called on a spacecraft and updates the internal per-step radiator and bus temperature variables using the current bulk heat variables, physical material constants (globally configured) and spacecraft-specific parameters such as the radiator area and the payload data throughput rates. For the radiator temperature the computation looks like this (Eqn. 4.37):

$$STEP_{temp\_Rad} = Q_{Rad} \cdot \frac{1}{c_{Rad}} \cdot \frac{1}{\rho_{Rad} \cdot A_{Rad} \cdot d_{Rad}} \tag{4.37}$$

$$Kelvin = Joules \cdot \frac{grams \cdot Kelvin}{Joules} \cdot \frac{1}{\frac{grams}{meter^3} \cdot meters^2 \cdot meters} \tag{4.38}$$

where:

$STEP_{temp\_Rad}$ is the temperature of the radiator in the current timestep in Kelvin.
Q_{Rad} is the bulk heat in the radiator in Joules.
c_{Rad} is the specific heat capacity of the radiator in $\frac{J}{g \cdot K}$.
ρ_{Rad} is the density of the radiator material, in $\frac{g}{m^3}$.
A_{Rad} is the area of the radiator in m^2.
d_{Rad} is the thickness of the radiator in meters.

As equations 4.37 and 4.38 show, the temperature of the radiator can be computed directly from the material properties and geometry of the radiator, as specified through global configurable variables, and through spacecraft configurations. Note that an assumption is made here that the radiator is a uniform solid of the material properties specified in the density and specific heat variables.

The computation of the bus control region temperature is made more complex by the presence of more materials that constitute a significant fraction of the total mass. For this, three primary contributors to the thermal mass of the bus control region are identified; the payload (either instruments or processing), the bus structure, and the batteries. While some other systems may contribute non-negligible components to the spacecraft total mass, their contributions can vary significantly depending on implementation. The fuel system is a good example of this, where the placement and insulation of the tanks, as well as the quantity and type of fuel, are not guaranteed to be configured in such a way that they contribute to the bus thermal control region.

While the calculation itself is longer, the underlying math is basically the same as for the radiator:

$$\begin{aligned} STEP_{temp\_Bus} = Q_{Bus} \cdot \Bigg(& \left(\frac{1}{c_{payload}} \cdot \frac{1}{sp_{payload} \cdot perf_{payload}} \right) \\ & + \left(\frac{1}{c_{struct}} \cdot \frac{1}{m_{struct}} \right) \\ & + \left(\frac{1}{c_{batt}} \cdot \frac{1}{e_{batt} \cdot cap_{batt}} \right) \Bigg) \end{aligned} \tag{4.39}$$

$$\begin{aligned} K = J \cdot \Bigg(& \left(\frac{g \cdot K}{J} \cdot \frac{1}{\frac{g}{bps} \cdot bps} \right) \\ & + \left(\frac{g \cdot K}{J} \cdot g \right) \\ & + \left(\frac{g \cdot K}{J} \cdot \frac{1}{\frac{g}{Whr} \cdot Whr} \right) \Bigg) \end{aligned} \tag{4.40}$$

where:

$STEP_{temp\_Bus}$ is the temperature of the bus in the current timestep in Kelvin.
Q_{Bus} is the bulk heat in the bus in Joules.
$c_{payload}$ is the specific heat capacity of the payload in $\frac{J}{g \cdot K}$.
c_{struct} is the specific heat capacity of the bus structure in $\frac{J}{g \cdot K}$.
c_{batt} is the specific heat capacity of the battery in $\frac{J}{g \cdot K}$.
m_{struct} is the mass of the structure in grams.
$sp_{payload}$ is the specific performance of the payload in $\frac{g}{bps}$.
$perf_{payload}$ is the performance of the payload in bps.
e_{batt} is the specific energy of the battery in $\frac{g}{Whr}$.

cap_{batt} is the capacity of the battery in Whr.

As shown, the temperature of the bus can be computed directly from the material properties and geometry of the bus systems, as specified through global configurable variables, and through spacecraft-specific configurations. Note that an assumption is made here that the components in question are uniform solids of the material properties specified in their respective density and specific heat variables.

The bus absorbs infrared radiation from multiple sources that are added together. For validation, three different sets of conditions have been chosen from the test configuration: (1) The spacecraft is in sunlight near solar noon; (2) the spacecraft is in sunlight near the terminator; and (3) the spacecraft is in eclipse. By probing these different operating regimes, it can be shown that the simulated solution matches what is expected from a first-principles approach. What is expected is that, under each tested flight condition, the thermal power absorbed by the bus from each source of heat matches between the hand-assessed value and the simulation-generated value (Table 4.6). All calculations were performed on a test spacecraft with a bus cross section of 2 m^2 and a surface absorptivity of 0.019.

Notes	Step	Altitude	SVE angle	Deep Space Flux	Solar Flux	Earth Albedo Flux	Earth IR Flux	**Total Absorbed Power**
Expected Solar Noon	173	7299.46	171.97	3.01e-6	1405.12	413.23	370.598	**76.0845**
Sim Solar Noon	173	7299.46	171.97	3.01e-6	1405.12	413.23	370.599	**76.0846**
Expected Terminator	223	7304.08	93.39	3.01e-6	1404.98	24.675	370.598	**64.817**
Sim Terminator	223	7304.08	93.39	3.01e-6	1404.98	24.67	370.599	**64.817**
Expected Eclipse	273	7294.38	10.23	3.01e-6	0	0	370.598	**10.743**
Sim Eclipse	273	7294.38	10.23	3.01e-6	0	0	370.599	**10.743**

Table 4.6: The table illustrates several different thermal regimes encountered in orbit, the expected absorbed power from the bus matches very closely the simulated absorbed power from the bus in different flight conditions. Each pair of expected and sim-produced values is grouped in a blue circle. This meets expectation and indicates that the simulation code correctly implements radiant power absorption into the bus.

It is slightly less cumbersome to validate the outwards radiation of heat from the bus, as this is purely an application of the Stefan-Boltzmann law and is not affected by varying orbital conditions. For a spacecraft with a bus cross-sectional area of 2 m^2, a `busSurfaceAreaToCrossSection` of three, and an emissivity of 0.05, the expected radiated bus power at different temperatures is compared with the results of the simulation. If the simulation is correctly calculating the radiated power from the bus, a close match between the hand-calculated values and the simulated values is expected (Table 4.7).

Step	Bus Temperature (Kelvin)	Expected Emitted Power (W)	Simulation Emitted Power (W)
5	303.2721	143.901	143.945
251	312.5149	162.262	162.317
3282	305.0243	147.255	147.302

Table 4.7: A close match between the radiated power found from analysis, and the radiated power found by the simulation. Slight variation between the results is most likely due to small rounding errors occurring when the data is output from the simulator. As these differences are $\ll$1% they are be considered negligible.

The last component of the bus that requires validation is the process of accumulating all of the different sources and sinks of power and showing that they are being correctly combined and converted to heat on the

correct timescale. Much like before the expected values for different regimes of flight are compared with the simulation-produced value. All calculations are performed per-step with a constant step size of 30 seconds (Table 4.8).

	Eclipse, No Heat Pump	Sunlight, Propulsion	Sunlight, Propulsion, Heat Pump
Step	72	114	380
Payload Power (W)	0.3	0	0.3
Propulsion Inefficiency Losses (W)	0	1000	1000
Bus Power (W)	76.56	76.56	76.56
Radiated Heat (J)	4286.4	4288.8	4377.1
Heat Pump Backflow Power (W)	0	0	50
IR Absorbed Power (W)	10.76	64.13	76.04
EPS Inefficiency Power (W)	2.38	64.22	79.70
Pumped Heat (J)	0	0	30,000
Expected Change in Bus Heat (J)	**-1,586.4**	**+31,858.5**	**+4,100.9**
Sim Result Change in Bus Heat (J)	**-1,580.0**	**+31,860.0**	**+4,100.0**

Table 4.8: A close match is found between manually calculated thermal accumulation results and those found during simulation. This indicates strongly that the simulation is accurately handling the thermal conditions inside the spacecraft bus. There are small differences between what the simulation calculates and the values found manually, however the differences are below 1% and can be attributed to mid-process rounding error.

Like the bus, the radiator absorbs infrared radiation from multiple sources, which are added together. For this analysis, the same three sets of conditions have been chosen from the test configuration; the spacecraft is in sunlight near solar noon, the spacecraft is in sunlight near the terminator, and the spacecraft is in eclipse. Under each tested flight condition, the thermal power absorbed by the radiator from each source of heat should match both the hand-assessed value and the simulation-generated value. All calculations were performed on a spacecraft with a radiator surface area of 1 m^2 and a surface absorptivity of 0.019 (Table 4.9).

Notes	Step	Altitude	SVE angle	Deep Space Flux	Solar Flux	Earth Albedo Flux	Earth IR Flux	**Total Absorbed Power**
Expected Solar Noon	174	7300.05	172.14	3.01e-6	1404.52	413.23	370.61	**2.59**
Sim Solar Noon	174	7300.05	172.14	3.01e-6	1404.52	413.23	370.60	**2.46**
Expected Terminator	226	7303.90	90.30	3.01e-6	1404.37	2.17	370.61	**3.26**
Sim Terminator	226	7303.90	90.30	3.01e-6	1404.37	2.17	370.60	**2.80**
Expected Eclipse	278	7293.59	7.77	3.01e-6	0	0	370.61	**0.126**
Sim Eclipse	278	7293.59	7.77	3.01e-6	0	0	370.60	**0.063**

Table 4.9: Across several different thermal regimes encountered in orbit, the expected power absorbed by the radiator from each external source closely matches the power absorbed power found by the simulator. There is a larger discrepancy between the total absorbed power near the terminator. This is due to compounding rounding errors occurring in the manual calculation. Specifically, the low albedo surface flux (which is reliant on the sun-spacecraft-earth angle) is multiplied by sin(5 degrees) which roughly accounts for imperfect pointing (the sunlight is glancing off the radiator at a 5 degree incidence angle. Because the numbers are made so small mid-calculation, the percentage difference between the values is high (∼16%) while the actual difference on the overall simulation is small (< 1 Watt difference). A similar issue arises with the eclipsed comparison, as the difference between the two values is nearly 100%, that from a purely numerical perspective is a problem, however in context this constitutes a difference of 60 milliwatts, which is a negligible discrepancy. As such, it is reasonable to conclude that this meets our expectations and indicates that the simulation code correctly implements radiant power absorption into the radiator.

Much as with the bus section, it is less cumbersome to validate the emitted radiation from the radiator. For a spacecraft with a radiator single-surface area of 1 m^2 and an emissivity of 0.96, the expected radiated power at different temperatures can be compared with the results of the simulation. If the simulation is correctly calculating the radiated power, a close match is expected between the hand-calculated values and the simulated values (Table 4.10).

Step	Radiator Temperature (Kelvin)	Expected Emitted Power (W)	Simulation Emitted Power (W)
3	328.802	1272.45	1297.36
113	269.062	570.59	567.63
503	300.902	892.51	905.81

Table 4.10: A match is found between the radiated power found by analysis and the radiated power found by the simulation. Slight variations between the results are due to small rounding errors occurring when the data is output from the simulator. As these differences are less than 1% and can be considered negligible.

The last component of the radiator thermal simulation that requires validation is the accumulation of all the different sources and sinks of power. Much like before, the expected values for different regimes of flight are compared with the simulation-produced value. All calculations are performed on a per-timestep basis with a constant step size of 30 seconds (Table 4.11).

	Eclipse, No Heat Pump	Sunlight, Propulsion	Sunlight, Propulsion, Heat Pump
Step	78	109	172
Radiated Heat (J)	26,679.55	18,054.80	31,731.69
Heat Pump Power (W)	0	0	450
IR Absorbed Power (W)	0.108	2.75	2.48
Pumped Heat (J)	0	0	30,000
Expected Change in Radiator Heat (J)	**-26,676.3**	**-17,972.3**	**+11,842.71**
Sim Result Change in Radiator Heat (J)	**-26,677.0**	**-17,972.0**	**+11,842.0**

Table 4.11: As expected, a match is found between the manually accumulated heat values, and those accumulated automatically by the simulator. This indicates that the simulation is accurately handling the thermal conditions of the radiator. There are small differences between what the simulation computes and the values found manually, however the differences are below 1% and can be attributed to mid-process rounding error.

4.4.3 Validation of Communications System Modeling

Several components of the communications calculations must be validated to confirm that the simulator is correctly analyzing the data environment. FreeFlyer has built-in functionality to accurately determine free-space path loss, EIRP, and the SNR of the received signal. What FreeFlyer does not provide is the implementation of the Shannon-Hartley theorem. Finally, a check that the same quantity of data that is moved from the sending spacecraft's outbound buffer to receiver spacecraft's inbound buffer validates that data is not lost or duplicated during the communications process.

The most critical calculation required for a communications link to function is the determination of the actual data rate from the current SNR and bandwidth. This is discussed in equation 4.20. The table below compares the simulator-produced channel capacity values to the expected manually calculated values.

Step	Distance (km)	Power (W)	SNR (dB)	Calculated Bitrate (bps)	Simulated Bitrate (bps)
27	5,806.295	0.3539	1.20	113.74	113.7504
99	912.004	0.4693	18.504	428.55	428.5703
425	3,590.788	0.4693	6.60	292.59	292.6039

Table 4.12: Comparison of Calculated and Simulated communications bitrates at Various Steps

As shown in Table 4.12, the simulated channel capacity matches with the expected channel capacity for several different sets of spacecraft distances (affecting FSPL and SNR), and transmit powers (affecting SNR). This validates that the simulator is correctly calculating the channel capacity of the communications links.

The final validation step is to show that data is correctly removed from the transmitting spacecraft and correctly accumulated in the receiving spacecraft.

Step	TX Beginning Outbox (bits)	TX End Outbox (bits)	Sent Data (bits)	RX Beginning Inbox (bits)	RX End Inbox (bits)	Received Quantity (bits)
27	300,000	0	300,000	300,000	600,000	300,000
99	300,000	0	300,000	4,200,000	4,500,000	300,000
425	300,000	0	300,000	21,300,000	21,600,000	300,000

Table 4.13: Check of Inbox and Outbox buffer quantities and data transported at various step. All data is shown to be accounted for here, no data is lost in transport and no additional data is accumulated.

As shown in Table 4.13, the amount of data removed from the sender, and accumulated in the receiver matches. This validates that the communication system of the simulator is not leaking or creating extra data erroneously.

It has been shown that the determination of channel capacity in the communications link simulation is accurate and matches the expected first principles derived results. Further, the amount of data transmitted in each step is correctly accounted for and is received in the same step. This results in the simulator being able to approximate actual communications system capabilities, while allowing a user to configure the precise performance of the system to their needs, thus validating the communications system.

4.4.4 Validation of Propulsion System Modeling

Validating the simulation's propulsion mechanisms requires demonstrating that the following expected behaviors occur:

1. Spacecraft with chemical propulsion systems perform appropriate burns to achieve their desired orbit.
2. Spacecraft with chemical propulsion systems perform appropriate burns to maintain their desired orbit if they drift from it.
3. Spacecraft with electric propulsion systems perform appropriate burns to work to achieve their desired orbit.
4. Spacecraft with electric propulsion systems perform appropriate burns to work to maintain their desired orbit if they drift from it.

For the purpose of validating the behavior of spacecraft with chemical thrusters (ImpulsiveBurn thrusters), a test spacecraft with the following parameters is configured in the simulator (Table 4.14):

Dry Mass (kg)	1000	Initial SMA (km)	7000
Fuel Mass (kg)	3000	Initial Eccentricity	0
Thrust (N)	500	Inclination (deg)	33
Thruster I_{sp} (Sec)	150		
Drag Coefficient	0.6	**Desired SMA (km)**	**10,000**
Drag Area (m^2)	2	**Desired Eccentricity**	**0**

Table 4.14: Configuration parameters used to validate behavior of impulsive propulsion system.

A Hohmann transfer to raise a circular orbit into another circular orbit is performed with two burns. The first burn should raise the SMA and eccentricity, while the second burn should again raise the SMA but lower the eccentricity to zero. This behavior is observed in the test scenario and is shown in Figure 4.18.

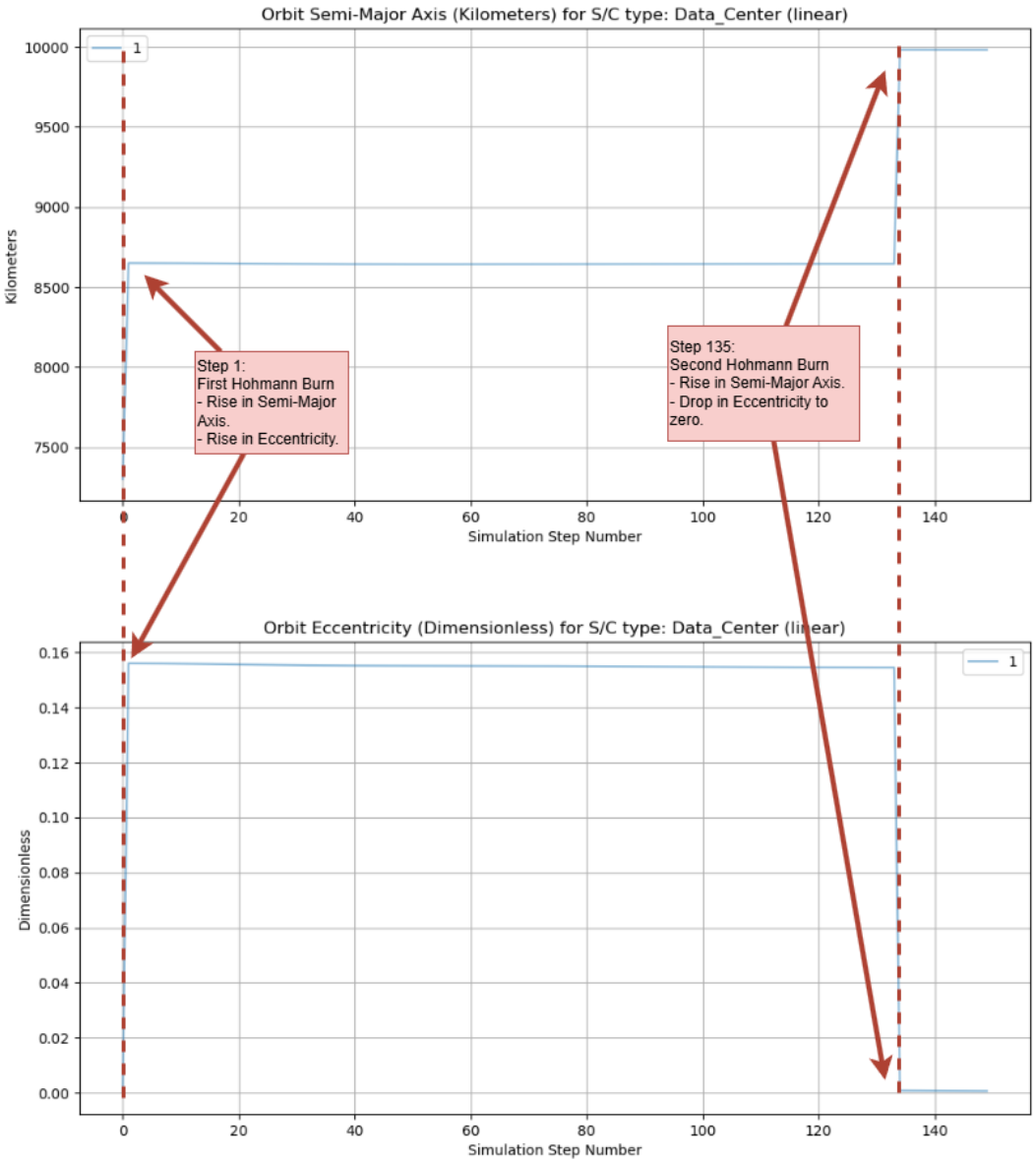

Figure 4.18: Plots of SMA and eccentricity for the test Hohmann transfer. Both the first and second maneuvers occurred as expected for a Hohmann transfer to a higher orbit. The first maneuver raised the SMA while also raising eccentricity, and the second maneuver raised SMA again to its target altitude of 10,000 km while lowering eccentricity to its target of zero.

As shown in figure 4.18, at the first timestep there is the expected behavior of a rise in SMA and eccentricity, and at the 135^{th} time step there is the expected behavior of a rise in SMA and a drop to zero in eccentricity. Further, an SMA of 10,000 km and an eccentricity of zero are the correct target orbit parameters for the test scenario. This validates the expected behavior for automated orbit raising.

The process for validating station-keeping behavior is similar, however the expected behavior changes

slightly. Instead of large changes to both eccentricity and SMA, only relatively small changes are expected, and maneuvers which pull the spacecraft away from its target orbit should be temporary. As Figure 4.19 shows, the automated station keeping behavior meets these expectations.

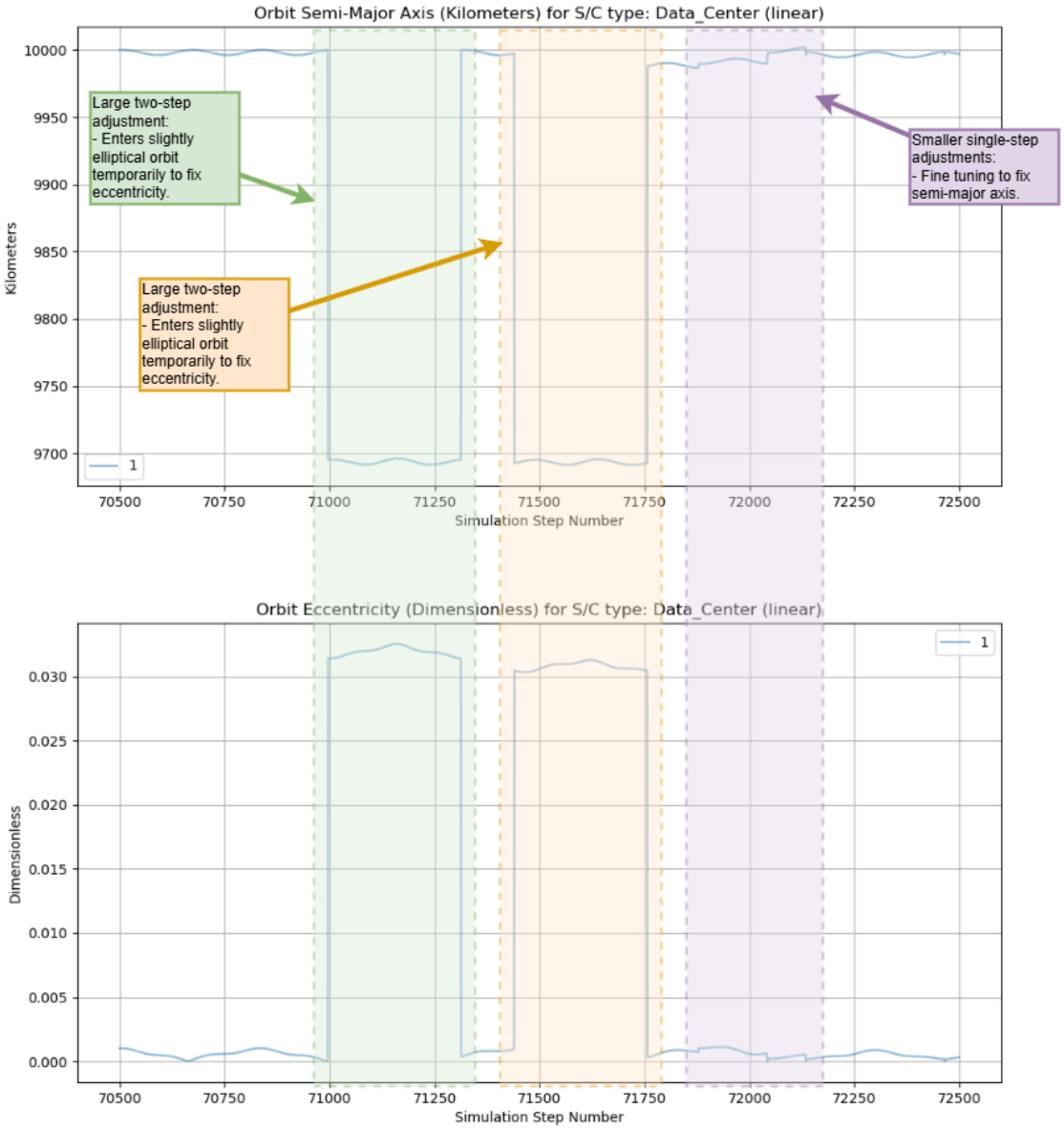

Figure 4.19: Plots of SMA and Eccentricity of spacecraft during station-keeping maneuvers. Highlighted regions have been added both to more easily associate areas between the two plots, and to correlate the changes in SMA and eccentricity with specific maneuvers and their purpose. The maneuvers automatically scheduled and executed here demonstrate the capability to automatically keep station at a target orbit.

As shown in Figure 4.19, the spacecraft needs to perform two deviations from its operating orbit in order to correct error in its eccentricity. Following this, the spacecraft needs to perform several very small

maneuvers to correct its semi-major axis towards the 10,000 km target value.

Between these two sets of maneuvers, it is shown that the simulator performs efficient and effective automated maneuvering of spacecraft with chemical propulsion systems. Validating the automated maneuvering of spacecraft with electric propulsion systems follows a similar process. While maneuvering with electric or low-thrust methods not discrete in execution, spacecraft with these thrusters are expected to exhibit behavior where their semi-major axis and eccentricity are trending towards their desired values. Once within tolerance of the target values, station-keeping should keep the spacecraft within range of its targets.

For the purpose of validating the behavior of spacecraft with electric thrusters (`FiniteBurn` thrusters), a simulated spacecraft with the following parameters is configured in the simulator (Table 4.15):

Dry Mass (kg)	1000	Initial SMA (km)	7,300
Fuel Mass (kg)	30	Initial Eccentricity	0
Thrust (N)	1	Inclination (deg)	33
Thruster I_{sp} (Sec)	600		
Drag Coefficient	0.6	**Desired SMA (km)**	**7,500**
Drag Area (m^2)	2	**Desired Eccentricity**	**0**

Table 4.15: Configuration parameters used to validate behavior of finite-time (electric) propulsion system.

A continuous maneuver is expected to raise the the semi-major axis of the spacecraft's orbit in a roughly-linear upward manner until the target is reached. The eccentricity may increase due to perturbing forces, but the propulsion solution should compensate for that and try to keep that as close to the target (zero) as possible. Such behavior is observed in the validation test scenario, as shown in Figure 4.20.

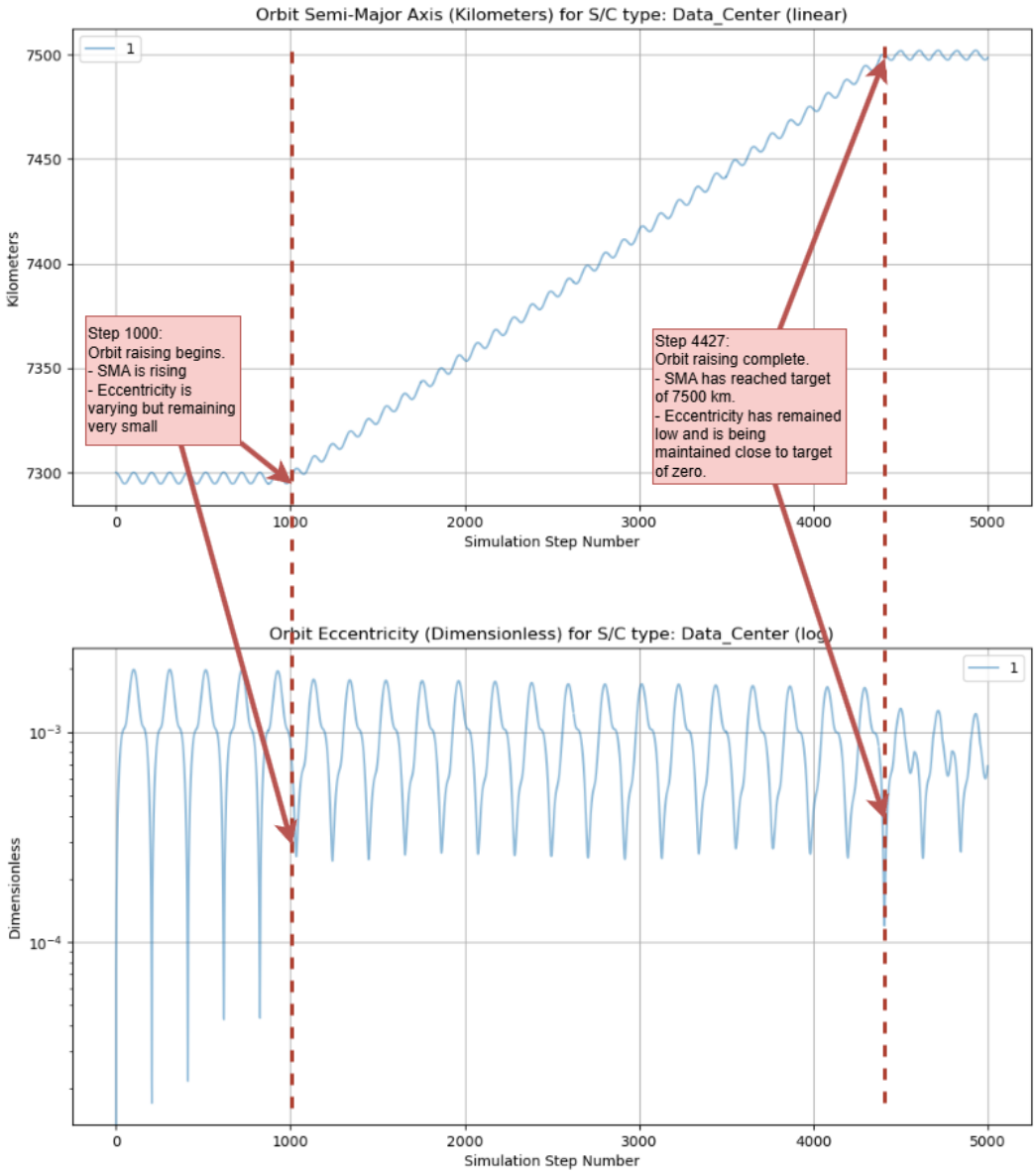

Figure 4.20: Plots of semi-major axis and eccentricity of orbit of test spacecraft with electric propulsion thruster raising orbit SMA from 7,300 to 7,500 km. The maneuver is started at timestep 1000 for clarity. The long-duration continuous maneuvering shown here demonstrates the capability for spacecraft with electric propulsion systems to automatically adjust their orbits toward their configured targets.

As shown in Figure 4.20, there is a clear upward trend in the semi-major axis for the duration of the simulation, the eccentricity is also clearly maintained at the target. Once the spacecraft reaches the target orbit, it terminates the maneuver and then maintains the orbit at the target altitude. This validates the expected behavior for both automated orbit raising and automated station keeping with electric thrusters.

4.4.5 Validation of Compute and Data Payload Modeling

Data generation is modeled with low-fidelity, and only bulk data quantity is being computed for the simulation. As such, validation of this process is not complex. Because the amount of data generated by a spacecraft and the duty cycle of that data generation, is specified by the user in the spacecraft configuration file, the process by which the user arrives at those values cannot itself be validated here. What is shown is that the amount of data generated in both a single timestep, as well as over longer periods of time, matches the expected values for both short and long time data rates.

In the short term, if a data generating spacecraft succeeds, its moves data to a DCS in time to make room for newly generated data. It is expected that the quantity of new data created in any timestep matches the number of bits that should be generated as specified in the `dataGenProfile` field of the configuration file. This behavior is validated in the test configuration, which uses a data generation rate of 10,000 bits per second, and a duty cycle of 0.2 , as shown in Table 4.16. The duty cycle of the data generation payload is the fraction of mission time it spends generating data. There are many reasons a sensor or instrument may not be able to generate useful data such as cloud cover obstructing a camera view or sensitive instruments being restricted to use during eclipse.

Step	Timestep Size (sec)	Expected Increase (bits)	Outbox Size Start (bits)	Outbox Size End (bits)	**Real Increase (bits)**
5	30	300,000	0	300,000	**300,000**
46	30	300,000	300,000	600,000	**300,000**
59	30	300,000	900,000	120,000	**300,000**

Table 4.16: Selection of timesteps in which data was generated by the test DGS, showing that the expected quantity of data generated matches the actual value for the size of time step.

The second component requiring validation is that over longer periods of time, the duty cycle is respected, and that the long-run quantity of data generated matches that of the raw data rate multiplied by the duty cycle. Indeed, this behavior is seen in the test configuration, as shown in Table 4.17.

Step	Elapsed Time (sec)	100% Duty Cycle Equivalent (bits)	Actual Data Generated (bits)	**Effective Duty Cycle**
76	2,280	22,800,000	3,900,000	**0.171**
1048	31,440	314,400,000	61,800,000	**0.197**
1243	37,290	372,900,000	73,500,000	**0.197**

Table 4.17: Selection of values of the total generated data accumulator on test DGS, as well as the amount of data that they would have generated if the duty cycle were 100%, and the effective duty cycle of the data generation.

As shown in Table 4.17, the effective duty cycle approaches the target of 0.2. There are some small differences between the effective duty cycle and the target value, but these are more prevalent early in the simulation where there are not enough data points for the overall rate to asymptote, as well as infrequent situations where the spacecraft completes a duty cycle check successfully, but cannot otherwise generate data, such as if the Outbox buffer is full or there is insufficient power.

With regards to the compute payload, the validation process is extremely simple. It must be shown that the data in the DCS's Inbox buffer is removed at the rate specified by `cpuThruput`, and that amount of data,

scaled by `cpuDataShrink`, is added to the DCS's Outbox buffer. This is shown clearly in Table 4.18, where the test configuration has a `cpuThruput` value of 10,000 bits per second, and a `cpuDataShrink` value of 0.1:

Step	Actual Inbox Quantity Removed (bits)	Expected Inbox Quantity Removed (bits)	Actual Outbox Quantity Added (bits)	**Expected Outbox Quantity Added (bits)**
7	300,000	300,000	30,000	30,000
34	300,000	300,000	30,000	30,000

Table 4.18: Selection of quantities of data removed from the test DCS Inbox buffer, scaled by the expected `cpuDataShrink` value of 0.1, and added to the Outbox buffer.

As shown in table 4.18, the DCS does process the data quantities it is expected to, given a set of specified rates and scale factors. It is also shown that the DGS generates data as expected by their configurations. All of this behavior meets the expectations for the data generation and processing sub-components of the simulation.

Chapter 5

Example Architecture Analysis Process

5.1 'Solution Space' Conceptual Approach

The viability of any space-based data processing system is determined by how well it can meet its performance requirements and how effectively it can mitigate its liabilities and risks. For a commercial system, the liabilities are primarily monetary, the capital investments for construction and maintenance are likely to be significant, and a system that does not meet its functional requirements is therefore represents a significant loss of money. For a government system, the liabilities are both monetary as well as security-oriented, a system which does not deliver the desired intelligence, surveillance, and reconnaissance (ISR) performance, for example, could jeopardize operational readiness and state security.

Due to the highly variable nature of each parameter in such systems, the design process must take into account the relative level of concern and impact for each parameter. For example, the government might care less about the total cost of operation than a business, or there may be different regulatory challenges depending on constellation size. Constructing a coherent and consistent method for comparing mission designs is necessary to be able to derive useful results from my simulation tool. In the field of systems engineering, this process is often wrapped into a methodology known as Quality Functional Deployment (QFD). For this analysis, the full implementation of QFD is excessive, but the fundamental concepts of applying variable weighting to different design parameters, requirements, and stakeholder desires is very useful.

When considering how changing a given design parameter can transform the effectiveness of a particular mission architecture, it can be useful to conceptualize the set of results for a range of inputs in terms of a solution space. Essentially, for a set of input parameters the simulation can produce a set of performance outputs that can be fed into a fitness function to produce a single quantitative "result" for the given set of inputs. As different input parameters are modified, the result value changes. Some results are unacceptable, such as solutions that exceed a maximum cost or do not meet performance requirement minimums. These unacceptable solutions are pruned from the solution space set, creating void regions in the set of valid architectures. Many other sets of input parameters produce results that meet requirements and are considered theoretically possible. Some valid solutions are better than others, some may be less expensive, or more capable of future expansion, or offer better price-to-performance relationships.

The identification of these regions where better options reside is where the solution space presents its

utility. In particular, the ability to observe the relative size of these "preferable" regions can provide useful insight into the practical viability of different topologies. For example, if the best options all came from a relatively small preferable region, that would indicate that those mission designs are quite sensitive to programmatic changes such as budget overruns, schedule slips, or technical failures, all of which would likely push the actual program outside of the simulated preferable region. In such cases, it may be beneficial to develop a mission topology that, while not the best, is less sensitive to such possibilities.

5.2 Fitness Function

A fundamental part of a solution space approach is the development and use of a "fitness function." Generally, a fitness function is a mathematical expression that, given input parameters, produces a result value that can be directly compared against other sets of inputs. Often optimization methods are used on systems to achieve a desired maximal result from the fitness function. At its core, the fitness function produces the actual 'result' from the solution space.

One of the most basic approaches to creating a fitness function is to integrate a weighted sum of input parameters. For the purposes of this work, it is not practical to prescribe weights for each possible real-life situation. As such, what is described here is a generally approach and set of considerations which mission design engineers can adjust as they deem appropriate. The weights associated with each input parameter are best conceptualized as the "importance" of the parameter, and often this will measure its relative effect on mission cost. Spacecraft mass, spacecraft quantity, and the number of launches required are all good examples of parameters which have significant impacts on mission costs. However, as real-world pressures can vary significantly between mission designs, it is possible that other parameters will need to take weighting precedence from time to time. For example, in the case of a race to market situation between multiple firms, the development and testing costs may be less important than meeting an aggressive schedule, and in such a situation the fitness function weighting for those parameters can easily be adjusted to account for that.

While the weights of input parameter values are generally selected by the design team as-needed, the parameters themselves are generally pulled directly from the mission design being analyzed. As mentioned previously these can be things like total mass to orbit, total quantity of spacecraft, mass of each spacecraft, etc. Some other parameters which are likely to be important for most (if not all) situations are:

- **Orbital characteristics:**
  - Geostationary orbit is saturated and expensive to get licensing for.
  - Generally, each distinct orbital inclination requires a separate launch, so multiple spacecraft distributed across a single orbit can be more desirable than each spacecraft being at a different inclination [41].
- **Costs of certain spacecraft components:**
  - Batteries are generally heavy.
  - Solar arrays are often very expensive due to high performance requirements.
  - Propulsion:
    - Many electric propulsion systems are not mass produced and can be extremely costly.
    - Certain fuels, such as Xenon, are very expensive.

- **Spacecraft lifetime requirements:**
  - Impacts fuel and launch requirements.
- **Lifetime data generation and processing quantities:**
  - As the total amount of work done by a system increases generally the profitability of a system rises as well, more data processing capability may be desirable.
- **Spacecraft resource management:**
  - Are periods of non-operation tolerable? How long/often are they acceptable?
  - Are situations such as filling data storage buffers have the same level of design impact as running out of power?

One of the major benefits of a fitness function-based approach to comparing the results of different simulations, is that more complex programmatic concerns can be included in the quantitative analysis. For example, a programmatic desire to "minimize regulatory headache" can be quantified by adjusting in the weighting of parameters to promote designs which reduce communications downlink bandwidth requirements, have fewer discrete spacecraft, and avoid geostationary orbit. Geopolitical risks, such as the implementation or threat of tariffs on raw materials, can increase the cost and procurement lead times for components such as batteries, solar cells, and advanced computer processors, so to account for this the weighting of those components can be adjusted to punish mission designs which require larger quantities of each. These examples are non-exhaustive, but are presented to provide direction to mission design engineers.

5.3 Mass-to-Orbit Priority Over Cost Of Materials/Workmanship

The cost basis of a traditional data center deployment changes when such a system is placed in orbit[5]. One of the most significant factors influencing cost is power consumption, which is dependent on the large impact of battery, solar array, heat pump, and radiator sizing on mass to orbit.

If the cost per unit mass of orbital launch were to fall significantly, many of the underlying assumptions common to spacecraft design, such as emphasis on reliability and rigorous testing programs, may shift dramatically. For most Earth-orbiting spacecraft, the driving force behind high-reliability design is the extreme cost of launch, and therefore the extreme cost of replacement in the event of failure. This can be seen in the recent launch campaigns by SpaceX with their Starlink program, where their launch costs are effectively only the cost of the upper stage and fuel, a significantly lower actual cost than nearly all other spacecraft launches. Low mass to orbit costs allows SpaceX to significantly reduce their design-phase costs for reliability and testing, as they can accept a higher failure rate and adopt an iterative design approach. Indeed, this is seen in their actual Starlink spacecraft failure rate[42].

Mass to orbit is the single most important factor in spacecraft mission design, and major changes to it have impacts in every other part of any program. As seen in recent years, the cost of launch has dropped significantly and is expected to continue to fall as superheavy rockets come into revenue service[43][44]. As such, tools such as this one which allow design teams to rapidly test different mission design scopes will be extremely important for developing useful and profitable large-scale compute missions.

5.4 Realistic Scenarios and Range of Mission Designs

As an example of the type of mission design variability that can be analyzed by the simulator, the area of weather prediction and modeling has been is illustrative. Weather modeling, in particular, is a useful example for space-based compute missions because weather models are both extremely compute-intensive to run and require incoming data to be as recent as possible. Further, real weather monitoring satellites already exist and can be used as a basis for the DGSs configured for the simulator.

In this example, there are six DGS, matching as closely as possible three geostationary and three LEO spacecraft already in use by the United States to monitor weather. The three geostationary spacecraft are modeled after GOES 16, 17 , and 18, and have similar mass, solar array sizes, orbits, and data generation rates as their real-life counterparts. Similarly, the three LEO spacecraft are modeled after NOAA 15, 18, and 19, and likewise have similar mass, solar array sizes, orbits, and data rates as the real ones.

What becomes much more interesting is the variability options found in the DCS designs. These spacecraft are not based on any existing systems, so even fundamental design parameters can vary significantly. One of the core parameters that can be adjusted is the power consumption and throughput of the compute payloads. A highly distributed system might consist of hundreds or thousands of small spacecraft with the compute performance of a laptop, while a highly centralized system might consist of a 200-kilowatt supercomputer in a single spacecraft comparable to the International Space Station. Any combination of compute power and spacecraft quantity could yield different design outcomes.

On the matter of quantity there is also the question of data center distribution in space. More specifically, what DCS occupy which orbits. For smaller numbers of spacecraft, it may be most cost effective to launch multiple spacecraft at a time. Due to the significant fuel penalty of trying to modify orbital parameters like inclination or RAAN, this generally results in all of the spacecraft being launched together and occupying roughly the same initial orbit. One important parameter to consider here is that due to the need for line-of-sight communications, there is a significant benefit to having more data centers in orbits where DGS can have them in view for longer periods of time. Similar increases in visibility time can be found in higher altitude orbits with lower quantities of spacecraft. GPS is a prudent example, despite the constellation having a relatively low number of spacecraft, every spacecraft in orbit around Earth can see at least one GPS satellite at any time.

Other parameters that can have a significant impact on mission design viability are propulsion capabilities and communication capabilities. Propulsion devices and their fuel systems are very mass and volume intensive, which increases the cost of launch. Electric propulsion systems are themselves often lighter than chemical systems, however those gains can easily be lost in requisite increased size of the solar arrays and batteries. The less thrust a thruster produces, the longer it must burn to perform a desired maneuver, for an electric propulsion system this translates directly to either increased EPS mass, reduced energy available for operating the payload, or both. Simulating different spacecraft configurations with different propulsion system properties can provide insight into which systems are better suited for a particular design.

While some DGS may only need a single transmitting antenna, it is easy to imagine that a DCS might require the ability to receive data from multiple spacecraft simultaneously, thus necessitating multiple antennas. It is important to optimize the design for minimizing launch mass without sacrificing capability. Simulating different configurations of spacecraft antenna quantities can help highlight where bottlenecks in the data flow are occurring.

While these examples are not an exhaustive list of things which might be important when designing a spacecraft, they do highlight the benefits of being able to rapidly simulate different mission designs.

Chapter 6

Results Analysis

6.1 Example Scenario

To demonstrate the power of this analysis approach, a realistic example is needed. In this scenario a set of 24 DGS based as closely as possible on PlanetLab's SuperDove cubesats have been deployed into three orbital rings of eight spacecraft each. In this simulation, it is assumed that the data generating spacecraft cannot directly communicate with the ground, and each one instead uses its communications system to send data to a DCS. Table 6.1 describes the important parameters in more detail.

	DGS Set A	**DGS Set B**	**DGS Set C**
Quantity	8	8	8
Orbit Inclination (deg)	98.9	98.9	98.9
Orbit SMA (km)	6853	6853	6853
Orbit Eccentricity	0	0	0
Orbit RAAN (deg)	0	120	240
Data Generation Rate (bps)	125,086,786	125,086,786	125,086,786
Data Generation Duty Cycle (%)	70	70	70
ISL Maximum Rate (bps)	1,800,000,000	1,800,000,000	1,800,000,000
Dry Mass (kg)	5.8	5.8	5.8
Data Storage (GB)	2000	2000	2000
Battery Capacity (Wh)	80	80	80
Solar Array Area (m^2)	0.21	0.21	0.21

Table 6.1: A subset of the spacecraft configuration file parameters for the DGS in both example scenarios. Each spacecraft can transmit a maximum of 1.8 Gbps of data to a DCS[3][45][46][47].

Three scenarios using these data generating spacecraft were examined. The three scenarios differ only in the number of DCS. In the first scenario, there is only one DCS. In the second scenario there are twenty-four

DCS spread over two orbital planes. The third scenario has forty DCS spread over four orbital planes. Table 6.2 illustrates the major differences between each scenario's data center quantities and orbits. Table 6.3 enumerates the design parameters shared across the DCS for all three scenarios. Note that in these scenarios none of the spacecraft have a propulsion system.

Parameter	**Single DCS**	**24 DCS**	**40 DCS**
Quantity	1	24	40
Inclination (deg)	45	45	45
RAAN (deg)	0	0, 90	0, 45, 90, 135
SMA (km)	7500	7500	7500

Table 6.2: A comparison between the different orbital parameters and quantities of the data centers in each scenario.

Parameter	**All DCS Shared Value**
CPU Throughput	500,000,000
Storage (GB)	2000
Solar Array Area (m^2)	0.21
Radiator Area (m^2)	0.09
Bus cross-sectional area (m^2)	0.21
Bus drag coefficient	0.8
Battery Starting Capacity (Wh)	80
ISL Antenna Qty	2
ISL Max Data rate	1,800,000,000
ISL Bandwidth (MHz)	1500
ISL Center Frequency (MHz)	26,250
GND Antenna Qty	1
GND Max Data rate	30,000,000
GND Bandwidth (MHz)	375
GND Center Frequency (MHz)	8,212
Dry Mass (kg)	5.8
Processing Shrink Factor	0.2

Table 6.3: The DCS configuration parameters shared across all three scenarios.

The three ground stations are identical for each scenario. Each ground station has six antennas, operating at the same bandwidth and frequency as the space-to-ground communications systems on the DCS. The ground stations are located in Hawaii, Virginia, and Alaska. Figure 6.1 shows a graphical representation between a scenario with only one DCS, and a scenario with multiple DCS.

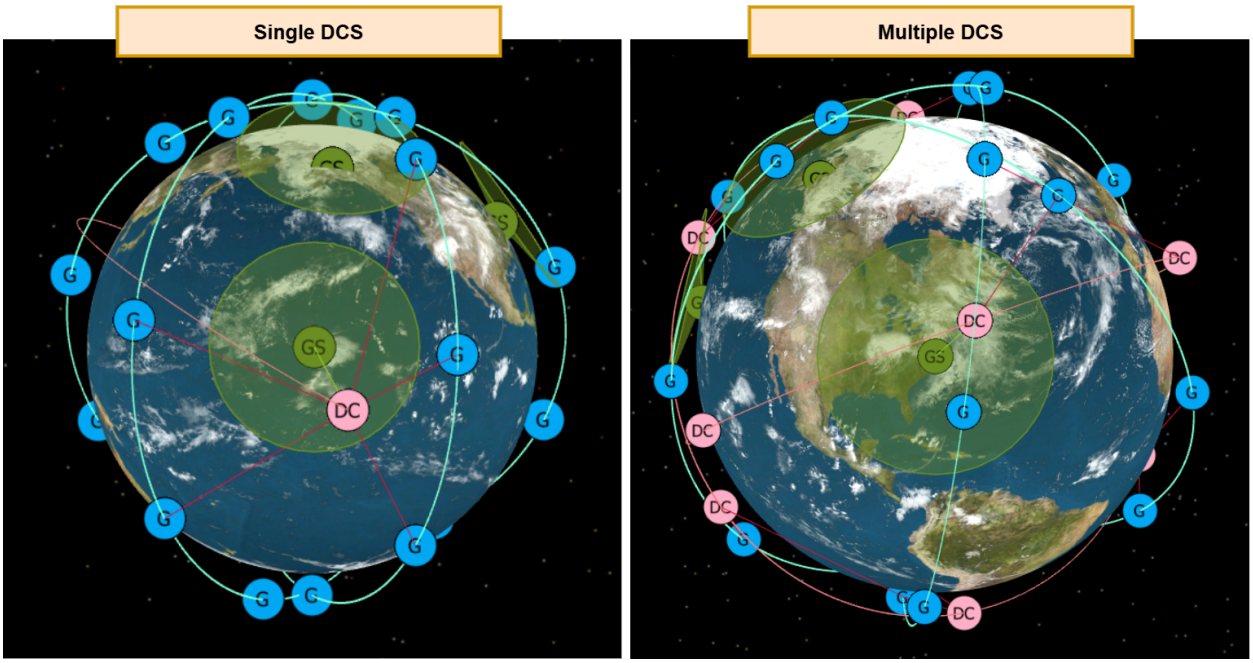

Figure 6.1: A comparison between two example scenarios as rendered by the FreeFlyer simulation. Each green circle labeled "GS" is a ground station, the translucent cone around each ground station indicates its field of view for communications. Each blue circle labeled "G" is a data generator spacecraft (DGS). DGS move along the blue lines. Each pink circle labeled "DC" is a data center spacecraft (DCS). DCS move along the pink lines. Red lines between DGS and DCS indicate active ISL communications links. Green lines between DCS and GS indicate active space-to-ground communications links.

In this example, the mission designer formulates several questions about each architecture:

1. How much data was processed and downlinked between the three scenarios?
2. What is the total mass to orbit of the DCS between the three scenarios?
3. Did any spacecraft run out of on-board storage, power, or other critical resource during operation?

Figure 6.2 shows the real-time onboard storage use of the DGS in each scenario respectively. Table 6.4 compares directly some of the most important results values from the simulation: the total raw data that was sent to DCS for processing, the total processed data sent to the ground, and the total mass to orbit in order to reach that level of performance.

	Single DCS	**24 DCS**	**40 DCS**
Total raw data sent from generators to DCS (GB)	36,567.519	870,638.041	1,438,517.507
Total processed data downlinked to ground from DCS (GB)	6,913.503	174,127.608	285,144.690
Total DCS mass to orbit (kg)	5.8	139.2	232
Separate Launches Required	1	2	4

Table 6.4: A comparison between critical result data of the two scenarios.

Table 6.4 clearly shows an improvement in the amount of data which can be processed as the number of DCS increases. However, the total mass to orbit and the number of required launches also increases in complementary fashion.

Figure 6.2 compares the real-time fraction of on-board storage in use throughout each scenario. In the scenario with only one DCS it can be clearly observed that each DGS spends nearly all of its time completely full of data. This indicates that there is not enough opportunity for any of the DGS to transmit their data for processing, due to both limited line-of-sight access to the sole DCS in that scenario, as well as the limited processing resources that the DCS possesses.

Figure 6.2 also provides a better illustration of the twenty four and forty DCS scenarios. In both cases, the DGS spend significantly less time full of data. In the twenty-four DCS scenario, the DGS appears to oscillate between periods of "good" and "poor" DCS service. This would suggest that there is enough raw processing power present with twenty-four DCS to meet the needs of the DGS (as there are phases where the DCS are able to catch up with the backlog), but that some other constraints appear to be active reducing bandwidth. Further analysis of this scenario may find that the quality of DCS service could improve with changes to their orbital distribution.

Finally, the forty DCS scenario shows that there is clearly enough processing capacity to meet the needs of the DGS. Interestingly, a similar periodicity can be seen when compared to the twenty-four DCS scenario, where the quality of service drops and then climbs again. The forty DCS scenario doesn't result in any DGS running out of storage space. It does however, present an excellent example of how there are multiple solutions that keep the DGS from running out of room to generate more data.

In this context, the ability to directly model these scenarios and comparing the resulting performance of each is extremely valuable. In the examples cited, the forty DCS scenario does technically work, but the less expensive twenty-four DCS scenario *could be made to work* with changes to the orbital distribution of the DCS, demonstrating the value of this tool for mission architects.

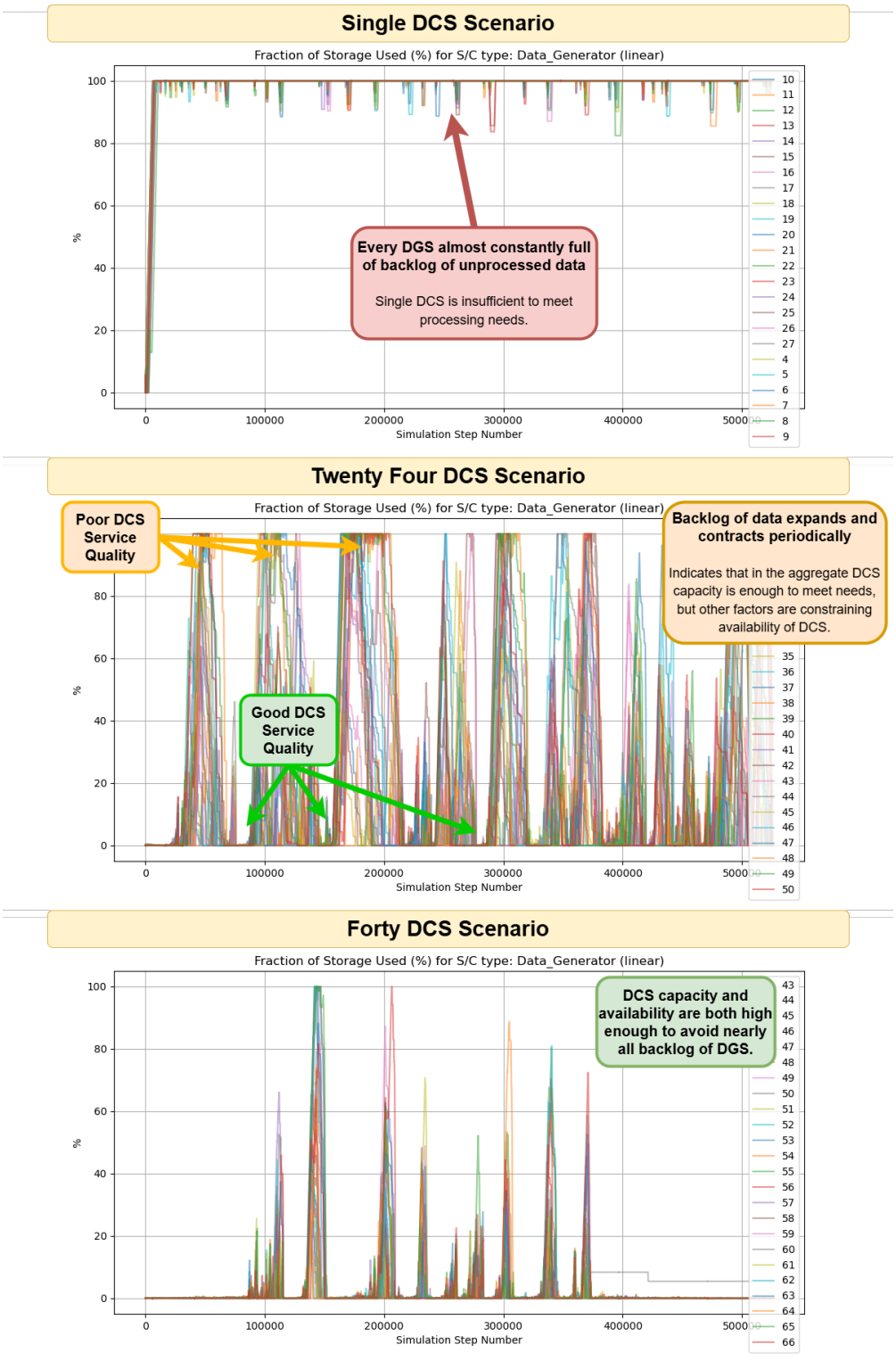

Figure 6.2: A direct comparison between the percentage of on-board storage used throughout each scenario on each DGS. The legends in each plot indicate which DGS ID in each scenario is associated with which plotted color, there is no difference between any of the DGS here. These plots demonstrate that a single DCS is insufficient to meet the processing needs of the DGS constellation, that forty DCS are sufficient to meet the needs of the DGS constellation, and that twenty four DCS are not capable of fully meeting the needs of the DGS constellation, but also that the twenty four DCS *may* be capable of meeting those needs were they in different orbits.

The results presented in these example scenarios demonstrate how this analysis process can be applied to various architectural scenarios. Mission designers and systems engineers start by enumerating stakeholder values as a set of measurable mission performance metrics. Then, the FreeFlyer tool can then be used to simulate permutations of spacecraft systems and orbital configurations and evaluate their performance and efficiency against the system objectives. Results can be used to inform design decisions and direct new candidate configurations for further simulation and analysis.

Chapter 7

Conclusion and Future Work

This thesis has developed a comprehensive methodology and simulation tool for the analysis of large-scale space-based compute mission designs. The increasing demand for timely insights from satellite-generated data, coupled with advancements in inter-satellite communication technologies, underscores the importance of exploring distributed computing architectures in orbit. This work addresses the limitations of traditional ground-based processing by providing a framework to model and evaluate the performance of various spacecraft subsystems and their interactions within a space-based data center network. The detailed implementation and validation of the electrical power, thermal management, orbital mechanics, communications, propulsion, and compute payload models within FreeFlyer demonstrate the tool's capability to simulate complex mission scenarios.

The significance of this research lies in its contribution to the nascent field of space-based computing. By providing a simulation platform, this work enables mission designers and engineers to analyze the feasibility, performance, and resource requirements of different architectural choices before actual deployment. The solution space analysis approach, coupled with customizable fitness functions familiar to many systems engineers, offers a valuable and novel approach for comparing diverse mission designs and identifying optimal solutions based on specific mission objectives and constraints. This capability is crucial in navigating the complexities of designing and deploying efficient and cost-effective space-based compute infrastructure.

Future work could focus on expanding the fidelity and scope of the simulation tool. Incorporating more detailed models for inter-satellite link performance under various environmental conditions, refining the thermal management models to account for spacecraft geometry and material properties more precisely, and integrating more sophisticated data processing algorithms would enhance the tool's accuracy. Furthermore, exploring a wider range of mission scenarios, including different orbital configurations, data processing workloads, and communication protocols, would provide a more comprehensive understanding of the design space.

Finally, future research could delve deeper into the analysis methodologies. Developing more sophisticated fitness functions that incorporate economic factors, risk assessments, and scalability metrics would offer a more holistic evaluation of mission viability. Investigating optimization algorithms to automatically identify optimal solutions within the defined solution space could further streamline the mission design process and accelerate the adoption of space-based computing for various applications.

References

[1] A. J. Tatem, S. J. Goetz, and S. I. Hay, "Fifty Years of Earth Observation Satellites," *American scientist*, vol. 96, no. 5, pp. 390–398, Sep. 2008, ISSN: 0003-0996. DOI: 10.1511/2008.74.390. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2690060/ (visited on 04/21/2025).

[2] P. Mack, *LANDSAT and the Rise of Earth Resources Monotoring*. [Online]. Available: https://www.nasa.gov/history/SP-4219/Chapter10.html (visited on 04/21/2025).

[3] E. Kulu, *Planet Labs - Satellite Constellation*, en. [Online]. Available: https://www.newspace.im/constellations/planet-labs.html (visited on 04/20/2025).

[4] Palantir, *Palantir Edge AI in Space*, en, Jun. 2023. [Online]. Available: https://blog.palantir.com/edge-ai-in-space-93d793433a1e (visited on 04/23/2025).

[5] N. Bleier, R. Eason, M. Lembeck, and R. Kumar, "Architecting space microdatacenters: A system-level approach," *IEEE HPCA*, Mar. 2025.

[6] E. Dunkel, J. Swope, Z. Towfic, *et al.*, "Benchmarking Deep Learning Inference of Remote Sensing Imagery on the Qualcomm Snapdragon And Intel Movidius Myriad X Processors Onboard the International Space Station," en, in *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, Kuala Lumpur, Malaysia: IEEE, Jul. 2022, pp. 5301–5304, ISBN: 978-1-66542-792-0. DOI: 10.1109/IGARSS46834.2022.9884906. [Online]. Available: https://ieeexplore.ieee.org/document/9884906/ (visited on 05/17/2024).

[7] *New AI Algorithms Streamline Data Processing for Space-based Instruments - NASA Science*, en-US. [Online]. Available: https://science.nasa.gov/science-research/science-enabling-technology/new-ai-algorithms-streamline-data-processing-for-space-based-instruments/ (visited on 05/17/2024).

[8] J. Swope, F. Mirza, E. Dunkel, *et al.*, "Benchmarking Remote Sensing Image Processing and Analysis on the Snapdragon Processor Onboard the International Space Station," en, in *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, Kuala Lumpur, Malaysia: IEEE, Jul. 2022, pp. 5305–5308, ISBN: 978-1-66542-792-0. DOI: 10.1109/IGARSS46834.2022.9883675. [Online]. Available: https://ieeexplore.ieee.org/document/9883675/ (visited on 05/17/2024).

[9] E. Feilden, A. Oltean, and P. Johnston, "Why we should train AI in space," en, 2024.

[10] Scott Manley, *Does It Make Sense To Put Data Centers In Space? Can They Really Cost Less To Operate?* Sep. 2024. [Online]. Available: https://www.youtube.com/watch?v=d-YcVLq98Ew (visited on 09/09/2024).

[11] S. Team, *What does real-time satellite data really look like? - SkyWatch*, en, Apr. 2022. [Online]. Available: https://skywatch.com/real-time-satellite-data/ (visited on 04/23/2025).

[12] A. Shehabi, S. Smith, D. Sartor, *et al.*, "United States Data Center Energy Usage Report," en, Tech. Rep. LBNL–1005775, 1372902, Jun. 2016, LBNL–1 005 775, 1 372 902. DOI: 10.2172/1372902. [Online]. Available: http://www.osti.gov/servlets/purl/1372902/ (visited on 04/20/2025).

[13] S. McClinton, *Why UHF Still Matters*, en, Mar. 2022. [Online]. Available: https://rbcsignals.com/blog/why-uhf-still-matters/ (visited on 04/21/2025).

[14] *FreeFlyer Space Mission Design, Analysis & Operations Software*, lang=. [Online]. Available: https://ai-solutions.com/freeflyer-astrodynamic-software/ (visited on 08/21/2024).

[15] T. Kerslake and E. Gustafson, "On-Orbit Performance Degradation of the International Space Station P6 Photovoltaic Arrays," en, in *1st International Energy Conversion Engineering Conference (IECEC)*, Portsmouth, Virginia: American Institute of Aeronautics and Astronautics, Aug. 2003, ISBN: 978-1-62410-088-8. DOI: 10.2514/6.2003-5999. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2003-5999 (visited on 02/12/2024).

[16] W. Kerslake and J. Hoffman, "Performance of the MIR Cooperative Array After 2.5 Years in Orbit," en, Vancouver, British Columbia, Canada: NASA, Aug. 1999.

[17] Y.-D. Youngdungpo-Gu, *Lithium Ion INR18650 MJ1 3500mAh PRODUCT SPECIFICATION*, en, Aug. 2014. (visited on 09/06/2024).

[18] J.-W. Lee, Y. K. Anguchamy, and B. N. Popov, "Simulation of charge–discharge cycling of lithium-ion batteries under low-earth-orbit conditions," en, *Journal of Power Sources*, vol. 162, no. 2, pp. 1395–1400, Nov. 2006, ISSN: 03787753. DOI: 10.1016/j.jpowsour.2006.07.045. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0378775306012948 (visited on 02/12/2024).

[19] I. Baghdadi, O. Briat, J.-Y. Delétage, P. Gyan, and J.-M. Vinassa, "Lithium battery aging model based on Dakin's degradation approach," en, *Journal of Power Sources*, vol. 325, pp. 273–285, Sep. 2016, ISSN: 03787753. DOI: 10.1016/j.jpowsour.2016.06.036. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0378775316307388 (visited on 02/12/2024).

[20] J. Olmos, I. Gandiaga, A. Saez-de-Ibarra, X. Larrea, T. Nieva, and I. Aizpuru, "Modelling the cycling degradation of Li-ion batteries: Chemistry influenced stress factors," en, *Journal of Energy Storage*, vol. 40, p. 102 765, Aug. 2021, ISSN: 2352152X. DOI: 10.1016/j.est.2021.102765. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2352152X21004904 (visited on 09/07/2024).

[21] B. Xu, A. Oudalov, A. Ulbig, G. Andersson, and D. S. Kirschen, "Modeling of Lithium-Ion Battery Degradation for Cell Life Assessment," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 1131–1140, Mar. 2018, Conference Name: IEEE Transactions on Smart Grid, ISSN: 1949-3061. DOI: 10.1109/TSG.2016.2578950. [Online]. Available: https://ieeexplore.ieee.org/document/7488267/?arnumber=7488267 (visited on 09/07/2024).

[22] D. Delafuente, *Lithium ion Batteries Cell-to-Cell Propagation Risk for Crewed Space Flight*, en, NASA Johnson Space Center, Oct. 2019. [Online]. Available: https://ntrs.nasa.gov/api/citations/20200003008/downloads/20200003008.pdf.

[23] *7.0 Thermal Control - NASA*, en-US, Section: S3VI, Feb. 2025. [Online]. Available: https://www.nasa.gov/smallsat-institute/sst-soa/thermal-control/ (visited on 04/22/2025).

[24] *Coefficient of Performance (COP) heat pumps*, en. [Online]. Available: https://www.grundfos.com/solutions/learn/research-and-insights/coefficient-of-system-performance (visited on 04/22/2025).

[25] I. Mills and International Union of Pure and Applied Chemistry, Eds., *Quantities, units, and symbols in physical chemistry*, en, 3rd ed. Cambridge, UK: RSC Pub, 2007, ISBN: 978-0-85404-433-7.

[26] M. Planck, *Treatise on Thermodynamics*. Dover Publications, 1945.

[27] D. Bhanderi and T. Bak, "Modeling Earth Albedo for Satellites in Earth Orbit," en, in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, California: American Institute of Aeronautics and Astronautics, Aug. 2005, ISBN: 978-1-62410-056-7. DOI: 10.2514/6.2005-6465. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.2005-6465 (visited on 05/14/2024).

[28] *2023 year-to-date temperatures versus previous years — Annual 2023 Global Climate Report — National Centers for Environmental Information (NCEI)*. [Online]. Available: https://www.ncei.noaa.gov/access/monitoring/monthly-report/global/202313/supplemental/page-1 (visited on 11/01/2024).

[29] *Joint Emissivity Database Initiative (JEDI)*, en-us. [Online]. Available: https://emissivity.jpl.nasa.gov (visited on 11/01/2024).

[30] D. J. Fixsen, "THE TEMPERATURE OF THE COSMIC MICROWAVE BACKGROUND," en, *The Astrophysical Journal*, vol. 707, no. 2, pp. 916–920, Dec. 2009, ISSN: 0004-637X, 1538-4357. DOI: 10.1088/0004-637X/707/2/916. [Online]. Available: https://iopscience.iop.org/article/10.1088/0004-637X/707/2/916 (visited on 04/04/2024).

[31] C. E. Shannon, "A Mathematical Theory of Communication," en, *The Bell System Technical Journal*, vol. 27, pp. 379–423, Oct. 1948.

[32] H.-G. Kim, D.-G. Kim, R.-H. Do, K.-R. Koo, and Y.-J. Yu, "Development of Deployable Reflector Antenna for the SAR-Satellite: Part 1. Design and Analysis of the Main Reflector Using Honeycomb Sandwich Composite Structure," en, *Applied Sciences*, vol. 14, no. 4, p. 1590, Feb. 2024, ISSN: 2076-3417. DOI: 10.3390/app14041590. [Online]. Available: https://www.mdpi.com/2076-3417/14/4/1590 (visited on 04/22/2025).

[33] J. Brodkin, *SpaceX tells FCC it has a plan to make Starlink about 10 times faster*, en, Oct. 2024. [Online]. Available: https://arstechnica.com/tech-policy/2024/10/spacex-claims-starlink-can-offer-gigabit-speeds-if-fcc-approves-new-plan/ (visited on 04/22/2025).

[34] *Egm96: The nasa gsfc and nima joint geopotential model*. [Online]. Available: https://cddis.nasa.gov/926/egm96/nasatm.html (visited on 02/20/2025).

[35] D. K. Skoulidou, F. Letizia, and S. Lemmens, "ESTIMATION OF THE UNCERTAINTIES OF THE ORBITAL LIFETIME OF SPACE DEBRIS," en, Darmstadt, Germany: ESA Space Debris Office, Apr. 2021. (visited on 02/20/2025).

[36] "Unclassified Publications of Lincoln Laboratory," en, vol. 14, Dec. 1988.

[37] A. Ruggiero, P. Pergola, S. Marcuccio, and M. Andrenucci, "Low-Thrust Maneuvers for the Efficient Correction of Orbital Elements," Sep. 2011.

[38] J. A. Reiter, A. K. Nicholas, and D. B. Spencer, "Optimization of many-revolution, electric-propulsion trajectories with engine shutoff constraints," Jan. 2015. [Online]. Available: https://www.researchgate.net/publication/270905125_Optimization_of_Many-Revolution_Electric-Propulsion_Trajectories_with_Engine_Shutoff_Constraints (visited on 04/23/2025).

[39] *Solar Irradiance — Sun Climate*, Jul. 2018. [Online]. Available: https://sunclimate.gsfc.nasa.gov/article/solar-irradiance (visited on 09/23/2024).

[40] *Azur Space Quadruple Junction GaAs Solar Cell Datasheet*, May 2019. (visited on 07/02/2024).

[41] R. Falck and L. Gefert, "A Method of Efficient Inclination Changes for Low-Thrust Spacecraft," Glenn Research Center: NASA, Oct. 2002.

[42] J. McDowell, *Jonathan's Space Report — Starlink Statistics*. [Online]. Available: https://planet4589.org/space/con/star/stats.html (visited on 03/24/2025).

[43] D. Chow, *To cheaply go: How falling launch costs fueled a thriving economy in orbit*, en, Apr. 2022. [Online]. Available: https://www.nbcnews.com/science/space/space-launch-costs-growing-business-industry-rcna23488 (visited on 04/24/2025).

[44] P. Lionnet, *SpaceX and the categorical imperative to achieve low launch cost*, en-US, Jun. 2024. [Online]. Available: https://spacenews.com/spacex-and-the-categorical-imperative-to-achieve-low-launch-cost/ (visited on 04/24/2025).

[45] Chief, Satellite Programs and Policy Division, Space Bureau, *Application for Modification of License*, May 2024. [Online]. Available: https://docs.fcc.gov/public/attachments/DA-24-446A1.pdf.

[46] K. Devaraj, *B14: The Cubesat with One of the World's Fastest Satellite Radios*, Company Blog, Aug. 2019. [Online]. Available: https://www.planet.com/pulse/b14-the-cubesat-with-one-of-the-worlds-fastest-satellite-radios/ (visited on 04/19/2025).

[47] *PlanetScope Product Specifications*, Dec. 2023. [Online]. Available: https://assets.planet.com/docs/Planet_PSScene_Imagery_Product_Spec_letter_screen.pdf.

Appendix A

Spacecraft Configuration File Specification

This is the file specification for the spacecraft configuration CSV file. A properly formatted file shall contain a comma-delimited list of the name field of each element as the first line to form column headers. Each subsequent line of the file shall be interpreted as definition parameters for a spacecraft or ground station (depending on the `type` field).

Element	Name	Unit	Description
0	qty	int	What is the quantity of spacecraft of this definition? If less than 1, then spacecraft will be evenly distributed around the defined orbit ring. Make zero to ignore line.
1	type	int	Type of DefinedSpacecraft: 1. 0 = Data Center (Spacecraft, has data processing capabilities, does not generate data) 2. 1 = Data Generator (Spacecraft, has no processing capabilities, generates data and passes it to somewhere else) 3. 2 = UNSUPPORTED 4. 3 = Ground Station (uses GroundStation object for comms referencing and ignores internal Spacecraft object)
2	epoch	MJD (TAI)	Starting Epoch of Spacecraft in this line.
3	orbitInc	Degrees	Inclination of initial orbit.
4	orbitRAAN	Degrees	Right ascension of the ascending node of initial orbit.
5	orbitSMA	km	Semi-major axis of initial orbit.
6	orbitEcc	Dimensionless	Eccentricity of initial orbit.
7	orbitomega	Degrees	Argument of periapse of initial orbit.
8	dataGenProfile	bps	Rate of data generation. Ignored if type is not 1.

9	dataGenDuty	Dimensionless	Duty cycle fraction of data generation subsystem. Ignored if type is not 1.
10	dataGenPower	w/bps	Power required to generate data, ignored if dataGenProfile is zero. Ignored if type is not 1.
11	cpuThruput	bps	Compute capability of the processor subsystem in INPUT bits per second. Ignored if type is not 0.
12	cpuPowerProfile	w/bps	Power consumption in watts of processor when running. Ignored if type is not 0.
13	cpuStorage	Bytes	Number of bytes of total storage on the spacecraft.
14	busSolarArea	m^2	Area of solar array.
15	busRadArea	m^2	Area of radiator.
16	busCrossArea	m^2	Cross-sectional area of spacecraft. Used for drag and lift calculations.
17	busDragCoef	Dimensionless	Drag coefficient of spacecraft.
18	busBattCap	wh	Capacity of batteries in spacecraft.
19	busISL	int	Quantity of inter-satellite communications antennas on the spacecraft.
20	busGNDAnt	int	Quantity of space-to-ground communications antennas on the spacecraft.
21	commISLMaxRate	bps	Maximum datarate through a single ISL antenna.
22	commISLPower	w/b	Power consumption of a single ISL transmitter.
23	commGNDMaxRate	bps	Maximum datarate through a single space-to-ground antenna.
24	commGNDPower	w/b	Power consumption of a single space-to-ground transmitter.
25	commISLBW	MHz	Bandwidth of each ISL channel.
26	commGNDBW	MHz	Bandwidth of each space-to-ground channel.
27	orbitTA	Degrees	True Anomaly of initial orbit, defaults to 0.
28	orbitSMADesire	km	Semi-major axis of target orbit. Ignored if thrustType is 0.
29	orbitEccDesire	Dimensionless	Eccentricity of target orbit. Ignored if thrustType is 0.
30	gsLatitude	Degrees	Planetodetic latitude of ground station. Ignored if type is not 3.
31	gsLongitude	Degrees	East longitude of ground station. Ignored if type is not 3.
32	scDryMass	kg	Dry mass of spacecraft.

33	thrustType	int	Type of thruster: 1. 0: No thruster. 2. 1: Electric Propulsion thruster. 3. 2: Monopropellant thruster. 4. 3: UNSUPPORTED - Solid rocket motor. 5. 4: UNSUPPORTED - Bipropellant thruster.
34	thrustISP	Seconds	Specific Impulse of thruster. Ignored if thrustType is 0.
35	thrust	Newtons	Thrust force of thruster during operation. Ignored if thrustType is 0.
36	thrustPower	w	Electric power consumption of thruster during operation. Ignored if thrustType is 0.
37	thrustFuelType	int	Type of fuel used in thruster. Enumeration must match thrustType. Ignored if thrustType is 0.
38	fuelMass	kg	Mass of fuel at start of mission. Ignored if thrustType is 0.
39	lifetime	Years	Lifetime of spacecraft. Ignored if type = 3.
40	thrustTankPressure	psi	Pressure of propellant feed tank. Ignored if thrustType is 0.
41	RESERVED	-	-
42	RESERVED	-	-
43	RESERVED	-	-
44	RESERVED	-	-
45	RESERVED	-	-
46	targetDoD	Dimensionless	Target limit for battery depth of discharge. Ignored unless Dynamic_Battery_Sizing is 1.
47	busHPMaxPower	w	Maximum electrical power the Heat Pump can consume during operation.
48	cpuDataShrink	Dimensionless	Shrink factor for data center processing. Ignored if type is not 0.

Appendix B

Spacecraft Output CSV File Specification

This is the file specification for the spacecraft output file. The output file is programatically generated and will vary in size and layout depending on the number of each type of spacecraft loading in the configuration file.

The simulator will output all of the variables and parameters of each spacecraft or ground station which are capable of changing throughout the mission. This means for example, that ground stations will not print out their location coordinates in this output file, as those parameters do not change. Only parameters which are relevant to the *type* of spacecraft are printed.

The top line of the CSV file is the column headers for every output parameter for every spacecraft and ground station in the simulation. Every subsequent line of the CSV file consists of the actual data aligned with each header. The first two columns will always be the current simulation step number, and the current time epoch of the simulator. All following columns are spacecraft-specific parameters are follow this naming structure:

List Index	_	Parameter Name	(Units)

For example: "7_Apogee Height (Kilometers)" is a valid header for the spacecraft at index 7 in `allSCList`. The header fields are structured always with only one underscore character, which separates the spacecraft index number from the human-readable name and units. Further, the units are always listed inside parentheses. This allows the user to easily configure their preferred plotting software to read in the parameter names and units from each header. Because the human-readable part of the header names are all the same for the same parameter between spacecraft (all spacecraft will have a "#_Apogee Height (Kilometers)" field), data for a specific header can be programatically grouped across spacecraft (i.e. plotting every spacecraft's Apogee Height together).

While the specific parameters which are printed for a given type of spacecraft vary (ground stations do not need to report battery charge level, etc.), all of the parameters related to a specific spacecraft will always start with the index number of that spacecraft (i.e. all parameters for the spacecraft at index 2 will have headers which start with "2". Further, all parameters related to a specific spacecraft will always be grouped together (i.e. all parameters for the spacecraft at index 2 will come together, followed by the parameters for spacecraft 3, etc.). The resulting table structure will be a larger version of this:

Step Number	Epoch	Headers for Index 0...	Headers for Index 1...	Headers for Index 2...
1	Date & Time...	Spacecraft 0 Data...	Spacecraft 1 Data...	Spacecraft 2 Data...
2	Date & Time...	Spacecraft 0 Data...	Spacecraft 1 Data...	Spacecraft 2 Data...
3	Date & Time...	Spacecraft 0 Data...	Spacecraft 1 Data...	Spacecraft 2 Data...
4	Date & Time...	Spacecraft 0 Data...	Spacecraft 1 Data...	Spacecraft 2 Data...