MODELING AND SIMULATION OF A CUBESAT WITH DUAL ELECTRIC PROPULSION SYSTEMS

BY

RAVI D. PATEL

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, 2021

Urbana, Illinois

Adviser:

Associate Professor Michael Lembeck

Abstract

The CubeSat mission analysis presented here is related to the Dual-Propulsion Experiment (DUPLEX) CubeSat in development by CUAerospace (CUA), with partners at the University of Illinois at Urbana-Champaign, NearSpace Launch, and NanoRacks. This CubeSat features two novel electric propulsion systems designed for use in CubeSats. This thesis presents the pre-flight technical analysis performed to determine the capabilities and limitations of these new propulsion systems.

In this study, the theory behind modeling low thrust trajectories for CubeSats is presented. This theory, implemented in high-fidelity thruster performance simulations, was used to develop and test mission profiles for the DUPLEX mission. The results are being shared with the NASA International Space Station operations personnel to assure safe and compatible operations of DUPLEX after deployment from ISS. Through this investigation, the capabilities and limitations of these two new electric propulsion systems for CubeSats have been determined.

To my parents, for their unyielding support. To the pursuit of space exploration. Per Aspera ad Astra.

Acknowledgments

During my time at the University of Illinois at Urbana-Champaign, I have been fortunate enough to have received guidance and support from many different people. This guidance allow me to participate in many unique opportunities and experiences without which I would not be where I am today.

Firstly, I would like to thank my advisor, Professor Michael Lembeck, for all the mentorship, encouragement, support, and guidance that you have provided me throughout my college career. In the midst of a global pandemic, you guided me through all of the challenges that came my way, both in my studies and my life. I am very grateful for all of the guidance and direction that you have provided me with. Through my experiences with you, I have learned many different things. Your patience and insight have always been valuable and words will never be enough to express my gratitude for all you have done for me.

I would also like to thank Professor Zachary Putnam for all your mentorship and guidance during my studies. As early as the first week of the first year of my undergraduate studies, you gave me the opportunity to work on a variety of different research projects and experiences that have led me to discover my passions and interests which has directly led to where I am today. I will be forever grateful for all of your help.

Additionally, I would like to thank Dr. David Carroll for the opportunity of working on the DUPLEX project. Your guidance and feedback were invaluable during the course of my graduate studies. I have learned many things from all of our interactions together.

I would like to thank a.i. solutions, Inc. for providing a FreeFlyer license for me to use for my studies. The use of the FreeFlyer software streamlined the modeling process and allowed me to spend more time on analysis.

I would also like to thank all of my peers who have made my experience at this university really amazing. Thanks to all of my friends in the LASSI lab, from whom I have gained exposure to so many different things, not only in my field but outside of my field as well. Thank you to all of the graduate students in the Putnam research group for the patience you had with me, and all of the guidance you gave me during my undergraduate years.

Lastly, I would like to thank all of my friends and family. I have been truly fortunate to have had you in my life. None of this would have been possible without the support you have provided me in all of my endeavors. I would especially like to thank my parents who have always been a crucial part of my life, and who have always provided me with support and guidance since the day I was born.

To all of those who have had an impact on my studies and my life, Thank you.

Table of contents

Chapter 1

Introduction

1.1 Motivation

The CubeSat small satellite form factor provides a low-cost platform for performing orbital and planetary science experiments, implementing commercial applications, and enhancing educational opportunities. The CubeSat standard was developed in 1999, with the base unit defined as a 10-cm x 10-cm x 10-cm cube [1][2]. This standard size can be seen in Figure 1.1.

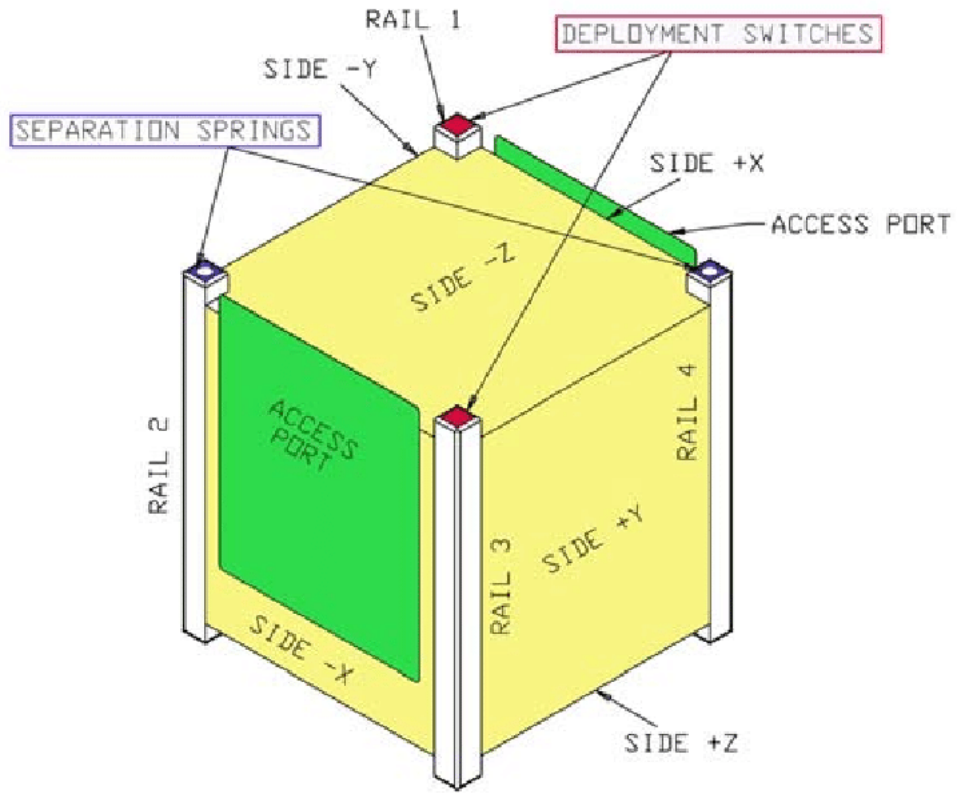

Figure 1.1: 1U CubeSat [3]

These 1-liter base units – known as U's – can be combined to form larger units, such as 2U, 3U, and 6U [4]. Each unit has a weight constraint of 1.33-kg per U [2]. The success of these CubeSat missions continues to drive interest in advancing the capabilities of the platform, specifically in expanding the platform's ability to perform orbital maneuvers such as inclination change, orbit maintenance, and orbit raising and lowering maneuvers.

Product	Thrust	Specific Impulse	Status
Hot Gas (Hydrazine)	0.5 – 4 N	150 – 250 s	TRL 6
Cold Gas	10 mN – 10 N	65 – 70 s GN2/Butane	TRL 9
Non-toxic Propulsion	0.1 – 27 N	220 – 250 s	HAN TRL 8, ADN TRL 6
Pulsed Plasma and Vacuum Arc Thrusters	1 – 1300 μN	500 – 3000 s	Teflon TRL 8, Titanium TRL 7
Electrospray Propulsion	10 – 120 μN	500 – 5000 s	TRL 6
Hall Effect Thrusters	10 – 50 mN	1000 – 2000 s	Xenon TRL 8, Iodine TRL 4
Ion Thrusters	1 – 10 mN	1000 – 3500 s	Xenon TRL 8, Iodine TRL 4

Table 1.1: Propulsion System Options - A variety of propulsion system products are available for small satellite applications today.[5]

Due to the mass and size constraints of the CubeSat platform, conventional propulsion methods are limited in their applicability. CubeSat propulsion systems need to produce thrust efficiently with limited mass and volume resources. Electric propulsion systems offer a low thrust, high-efficiency solution to this problem. Electric propulsion systems generally accelerate particles to very high velocities, allowing for the system to efficiently produce thrust at a very low propellant mass consumption. Because these systems have very high specific impulses, they allow for higher volume and mass fractions of the spacecraft to be dedicated to the payload [6]. For rapid maneuvering, requiring high thrust, CubeSats have generally used cold gas thrusters [7] [8]. These systems also tend to use toxic fuel [7]. The inclusion of an onboard pressure vessel complicates the system design and can lead to complications when considering the orbital debris and hazards for humans in ground handling and on-orbit risks related to system failures. Solid rocket motors have also

been designed and tested for small satellites, specifically for use in deorbit maneuvers [9]. These systems have the advantage of being highly reliable, however, the system is not throttleable and can only be used once. Electric propulsion systems that can reliably be throttled and operate at the high thrust levels required without using pressure vessels could provide a solution to this problem.

Two new electric propulsion systems have been developed by CUAerospace (CUA) for the CubeSat platform. These thrusters are called the Fiber-fed Pulsed Plasma Thruster(FPPT) [10], and the Monofilament Vaporization Thruster(MVP), a micro-resistojet. FPPT is a low thrust, high-efficiency thruster with a specific impulse of 3500s and allows for CubeSats to do slow and mass efficient maneuvers [11]. A picture of the thruster can be seen in These propulsion systems can be seen in Figure 1.2.

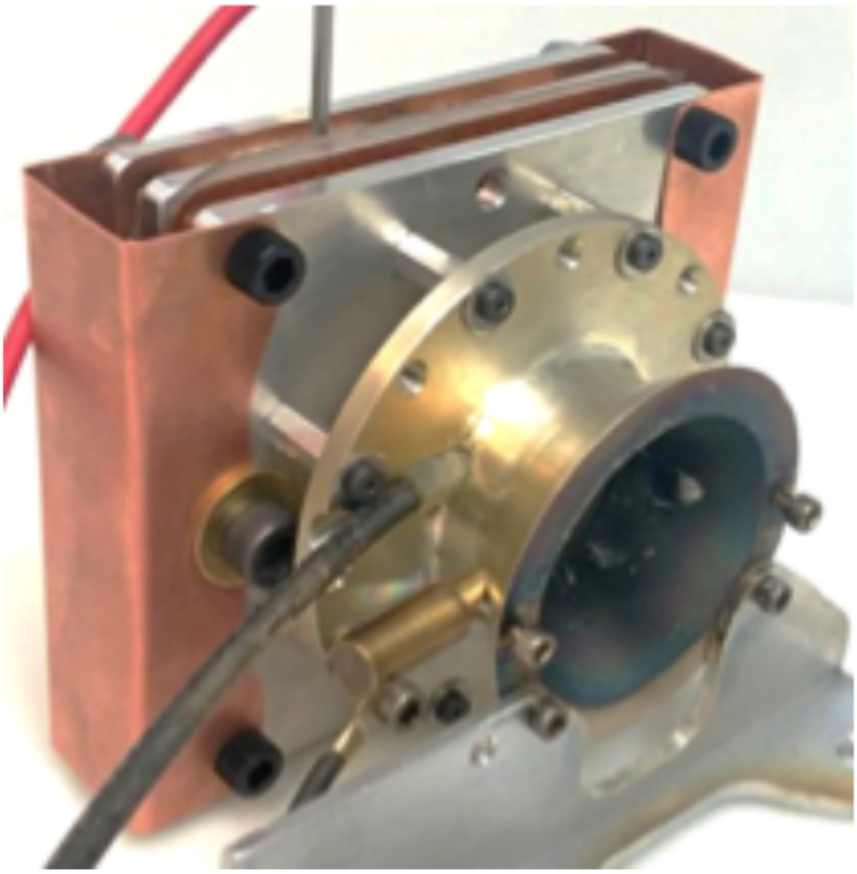

Figure 1.2: Fiber-fed Pulsed Plasma Thruster developed by CUAerospace [10]

MVP is a high thrust electric propulsion system with a specific impulse of 66s that allows for rapid maneuvers to be made [12]. A rendering of the MVP thruster can be seen in Figure 1.3.

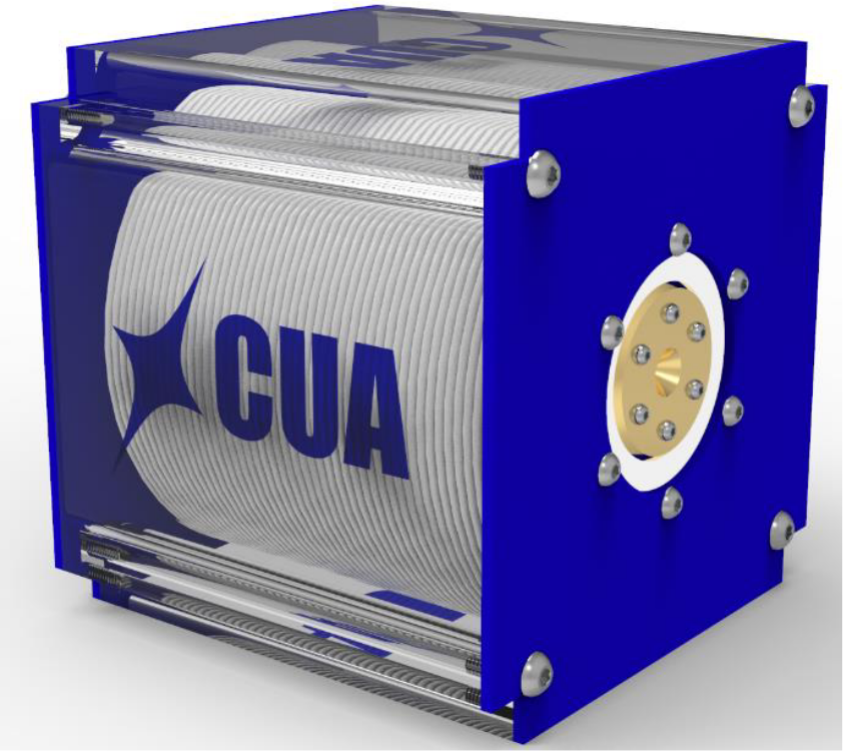

Figure 1.3: Monofilament Vaporization Thruster developed by CUAerospace [12]

The properties of these two systems are shown in Table 1.2. The capabilities and limitations of these two propulsion systems are bring investigated to quantify their applicability for future CubeSat Missions.

Product	**Thrust**	**Specific Impulse**
Fiber-Fed Pulsed Plasma Thruster (FPPT)	0.33 mN	3500 s
Monofilament Vaporization Thruster (MVP)	4.5 mN	66 s

Table 1.2: MVP and FPPT propulsion system properties

Both of these systems are different from their contemporary counterparts. Instead of relying on a spring mechanism to move a rigid propellant, FPPT utilizes a fiber feeding mechanism to manipulate its

flexible fiber propellant [10]. This allows for precise control and more effective storage of the propellant. The components of this system can be seen in Figure 1.4.

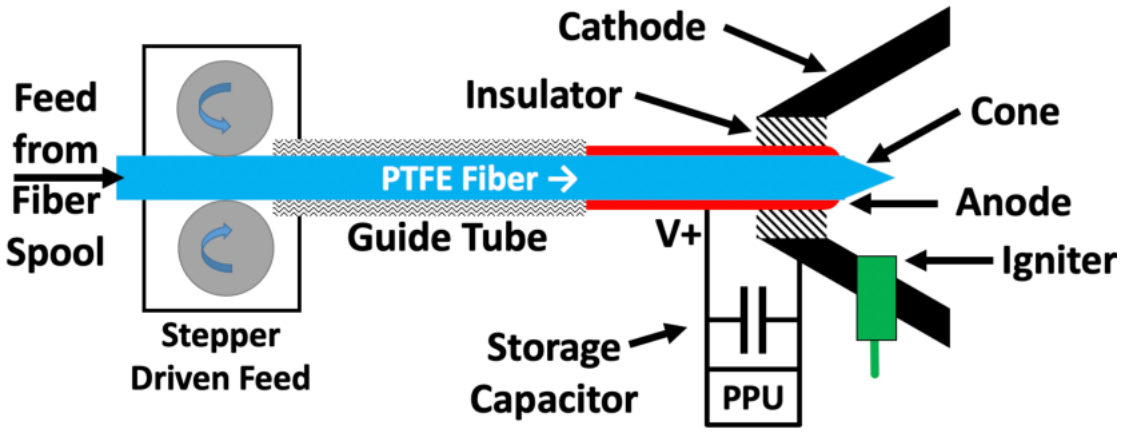

Figure 1.4: FPPT Design

In MVP's case, rather than using a pressurized gas like traditional resistojets, the propellant is solid and is stored as a flexible fiber instead [12]. This allows for safer and more efficient storage of the propellant. The components of this system can be seen in Figure 1.5.

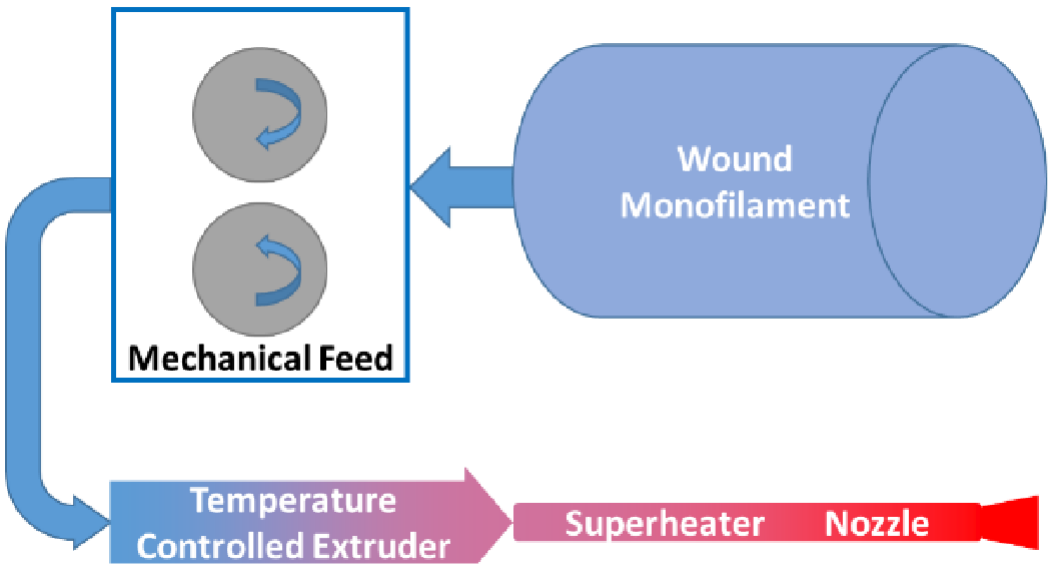

Figure 1.5: MVP Design

1.2 Background

1.2.1 CubeSats

The CubeSat platform concept was developed by a collaborative effort with researchers at the California Polytechnic State University and Stanford University under the guidance of Professor Jordi Puig-Sauri and Professor Bob Twiggs. The goal of this effort was to develop an architecture that could allow for a design standard to expedite, simplify, and lower the cost of developing a satellite system capable of carrying research payloads [13]. By standardizing the platform on which these small satellites could be built, it was believed that the costs would be dramatically reduced. This would also open opportunities for students to gain experience working on flight hardware.

Early CubeSats were adopted by universities for education and basic research, with a focus on the smaller 1U-3U sizes. The platform's flexible configurations and low cost were attractive to government entities and have led to the identification of multiple applications for CubeSats. This led to the development of larger form factors, with the inclusion of the 6U standard in 2014 and the 12U standard in 2016. These form factors can be seen in Figure 1.6.

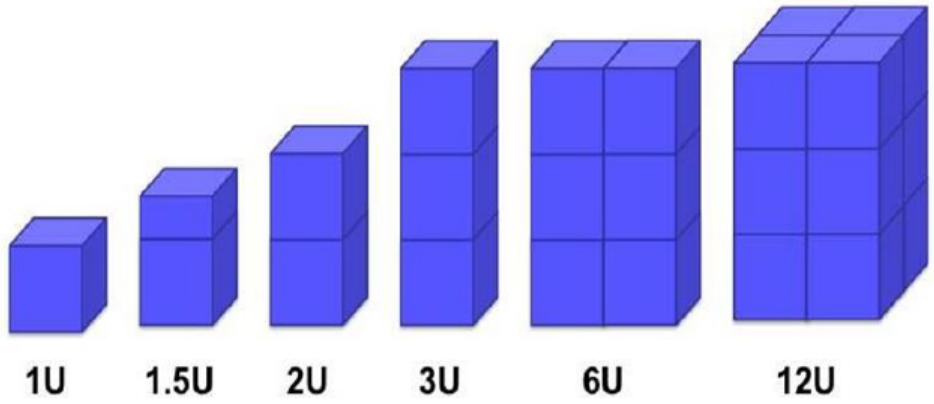

Figure 1.6: Arrangement of units for larger size CubeSats [14]

Since this expansion, there has been a significant increase in CubeSats launched. This is shown in Figure 1.7.

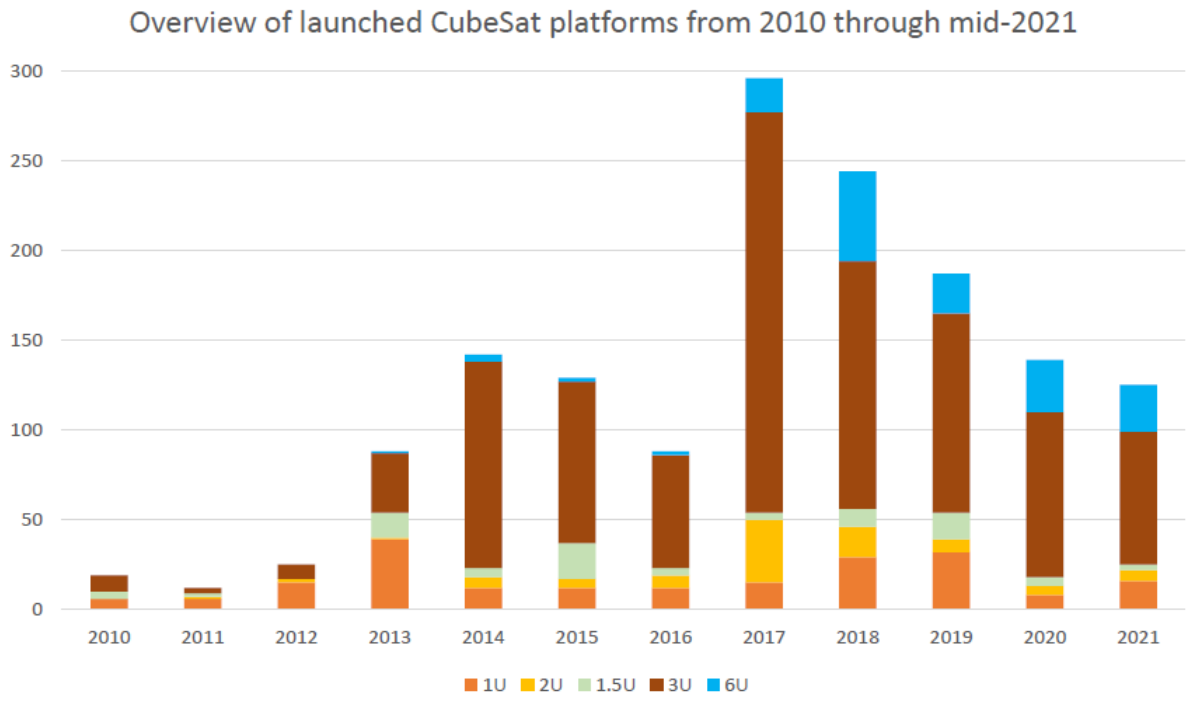

Figure 1.7: Graph of CubeSats launched per year [14]

CubeSat missions are also being launched for missions beyond LEO. Two Mars Cube One (MarCO) CubeSats were successfully flown by Mars, demonstrating the capability of the platform to operate beyond Earth orbit. MarCO used Compressed R236FA cold gas thrusters for orbital maneuvers [15]. This propulsion system is shown in Figure 1.8. This new class of interplanetary CubeSat missions would also benefit from the development of new, highly efficient propulsion systems.

Figure 1.8: VACCO JPL MarCO Micro Propulsion System [16]

1.2.2 Electric Propulsion Systems

In-space propulsion for CubeSats has been identified as a major enabling technology for future exploration missions [14]. Mission designs like GMS-T launched in 2021 have already demonstrated electrical propulsion systems on larger systems [17]. The demonstration of CUA's electrical propulsion systems on a smaller CubeSat will raise the Technology Readiness Level (TRL) [18], a measure of the maturity of a system, of these thrusters and enable more complex CubeSat missions.

Pulsed Plasma Thrusters

Pulsed plasma thrusters (PPT) produce thrust by accelerating ionized particles through a generated magnetic field and ejecting the high-velocity particles to impart momentum on the spacecraft. The propulsion system consists of a power supply, a power processing unit, and an energy storage medium. Generally, power supplied from solar panels is converted to high voltage by a power processing unit and stored in a capacitor bank. This can be seen in Figure 1.9.

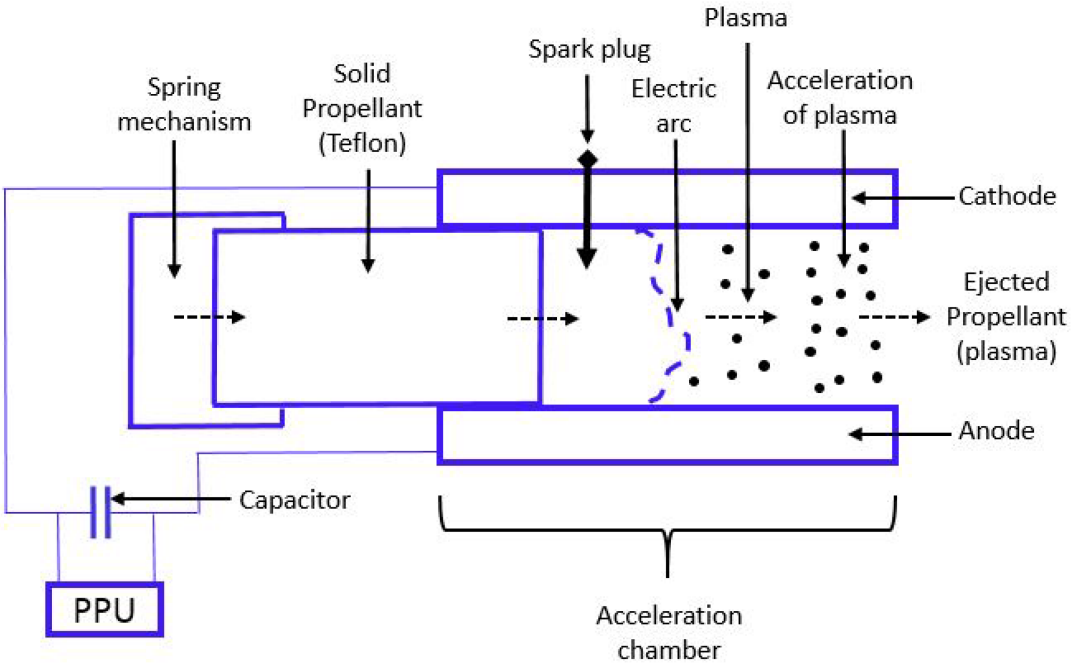

Figure 1.9: Components of a Pulsed Plasma Thruster [19]

This capacitor bank can discharge its energy when at maximum capacity, resulting in a short (10-20 μs) high current pulse [20] that can ablate a solid propellant and eject the resulting particulates at high velocities. Pulsed plasma thrusters have been successfully demonstrated dating back to the 1960s [21]. The thrust from a pulsed plasma thruster can be determined by equation 1.1. Pulsed plasma thrusters generally have low thrust and high efficiencies.

$$F_{Thrust} = m\frac{dv}{dt} = q(\vec{E} + V_{part} \times \vec{B}) + \sum F_{part} \tag{1.1}$$

Resistojets

Resistojets produce thrust by heating a propellant to a high temperature. Heating the propellant increases the pressure in the pressure chamber. The high pressure and high-temperature propellant is accelerated through a nozzle (typically a converging-diverging nozzle) and ejected into space to generate thrust and impart momentum into the spacecraft. The individual components of a resistojet can be seen in Figure 1.10.

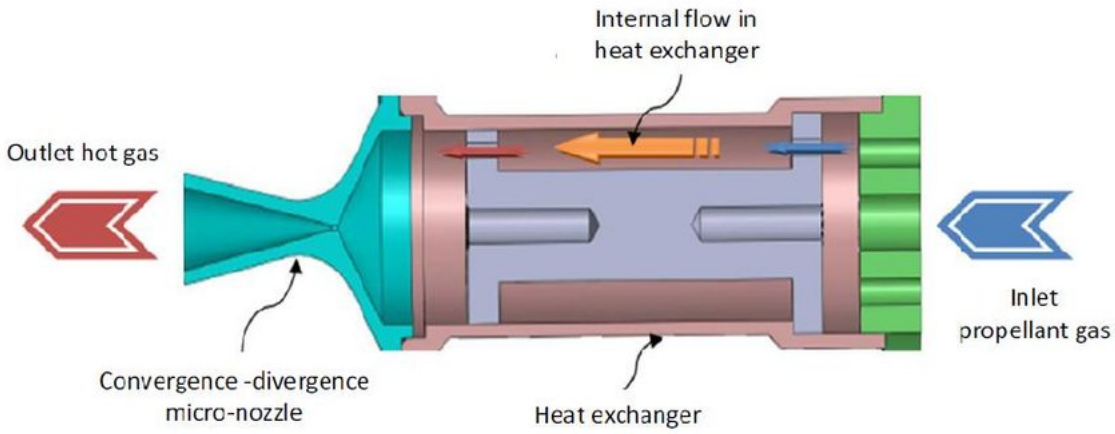

Figure 1.10: The components of a resistojet [22]

The equations describing the resultant thrust and specific impulse are shown in equations 1.2 and 1.3 respectively.

$$F_{Thrust} = m\frac{dv}{dt} = P_{exit}A_o(\frac{n_{dens}KT_0}{2}) \tag{1.2}$$

$$I_{sp} = \frac{1}{g_0} * \sqrt{\frac{\pi K T_0}{2m}} \tag{1.3}$$

Resistojets typically use pressure vessels to store pressurized propellant. This can lead to range safety complications. Propellants for resistojets tend to be toxic and can have a limited operating lifespan. If propellant could be stored in an inert, solid form, many of the hazards inherent in a pressurized propellant could be eliminated [23]. This idea forms the motivation for the concept investigation herein.

Chapter 2

Modeling Electric Propulsion Systems for CubeSats

The largest challenge with modeling low thrust maneuvers for CubeSats is that they must be modeled as a non-impulsive continuous burn. Analytic, closed-form solutions available for impulsive maneuvers do not provide accurate results and cannot be used for maneuver optimization. Another challenge with electric propulsion systems is dealing with non-uniform power availability, with large differences in available power between eclipse and peak power generation. These constraints have to be taken into consideration when determining the near-optimal steering and thrusting profiles required to complete desired orbital maneuvers.

2.1 Orbital Dynamics

2.1.1 Defining an Orbit

The dynamics of an object in orbit around a gravitational body can be defined in six degrees of freedom. These correspond to three spatial and three rotational dimensions that can encode an object's motion through space. Because of this, the dynamics that describe the motion of objects in orbit can be quantified using orbital elements. These orbital elements are parameters that can define a specific orbit. For a given reference frame and a specified point in time, six parameters can uniquely define any unperturbed orbit. The six traditional Keplerian orbital elements are one such set of elements. These elements are shown in Figure 2.1.

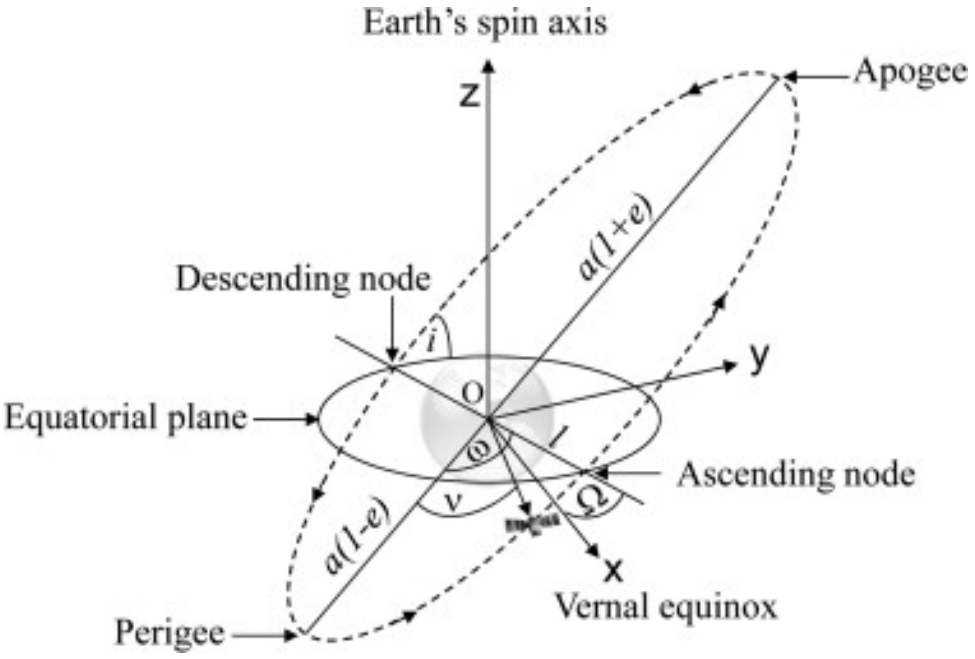

Figure 2.1: The six Keplerian orbital elements [24]

One subset of the traditional Keplerian elements defines the size and shape of the orbit. These include the **Semimajor axis (a)** and the **Eccentricity (e)**. The semimajor axis describes the size of the orbit, defined by measuring the distance between the orbit periapsis and orbit apoapsis. The eccentricity describes

the shape of the orbit, where an eccentricity of zero describes a circular orbit and the orbit gets more elliptical as the eccentricity increases.

Another subset of the traditional Keplerian elements defines the orientation of the orbital plane with respect to a reference frame. The **inclination (i)** measures the tilt of the orbit with respect to a reference plane. The **longitude of the ascending node (Ω)** measures the angle from a reference point where an orbit passes through a reference plane.

The last two traditional Keplerian orbital elements are the argument of periapse (ω) and the **true anomaly(ν)**. The argument of periapse describes the location of the orbit's periapse as measured from the ascending node. The true anomaly measures the location of the spacecraft itself on the orbit, as measured from the periapse.

These six elements describe an unperturbed and idealized orbit. The presence of orbital perturbations and thrust will induce a change in the original orbit, causing a change in the elements that define the orbit.

2.1.2 Changes to an orbit from perturbations

Gauss' variation of parameter equations defines the Keplerian orbital element rates-of-change from perturbations and forces on the satellite defined in terms of the Local Vertical Local Horizontal (LVLH) reference frame. The orientation of this frame is shown in Figure 2.2. The X direction is directed along the velocity vector in the direction of the local horizon. The Y component is radially normal to the velocity vector. The Z component is normal to the orbit plane and can be defined by the cross-product of X and Y.

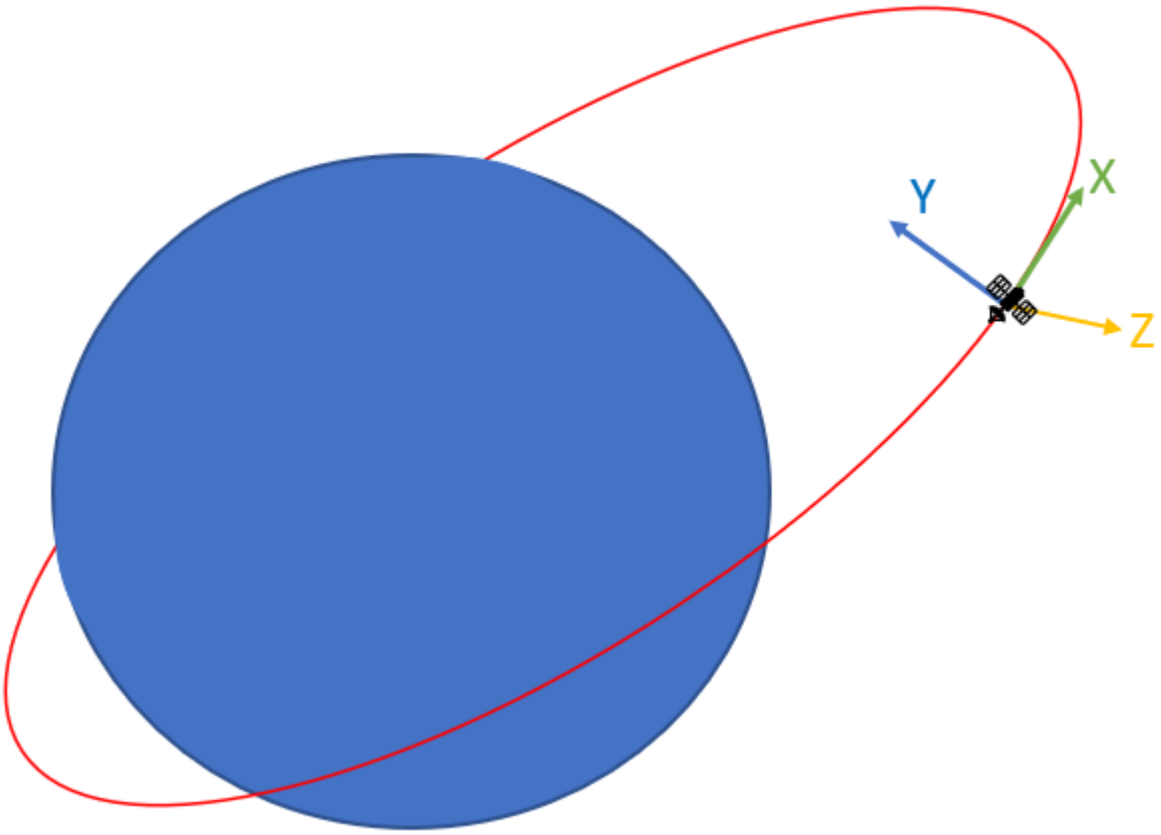

Figure 2.2: LVLH Frame for spacecraft in orbit, with X in the velocity direction, Y radially pointing in, and Z normal to the orbital plane

Using this reference frame, Gauss' variation of parameters equations are represented as shown in equations 2.1-2.7.

$$\frac{da}{dt} = \frac{2a^2}{h}\left(e\sin(v)\alpha_y + \frac{p}{r}\alpha_x\right) \tag{2.1}$$

$$\frac{de}{dt} = \frac{1}{h}\left[p\sin(v)\alpha_y + ((p+r)\cos(v) + re)\alpha_x\right] \tag{2.2}$$

$$\frac{di}{dt} = \frac{r\cos(\omega+\nu)}{h}\alpha_z \tag{2.3}$$

$$\frac{d\Omega}{dt} = \frac{r\sin(\omega+\nu)}{h\sin(i)}\alpha_z \tag{2.4}$$

$$\frac{d\omega}{dt} = \frac{1}{he}\left[-p\cos(v)\alpha_y + (p+r)\sin(v)\alpha_x\right] - \frac{r\cot(i)\sin(\omega+\nu)}{h}\alpha_z \tag{2.5}$$

$$\frac{dv}{dt} = \frac{h}{r^2} + \frac{1}{he}\left[p\cos(v)\alpha_y - (p+r)\sin(v)\alpha_x\right] \tag{2.6}$$

where,

$$r = \frac{a\left(1-e^2\right)}{1+e\cos(v)} \quad \text{and} \quad h = \sqrt{\mu a\left(1-e^2\right)} \tag{2.7}$$

These equations can be used to numerically model the effect that perturbing accelerations have on the orbital dynamics of a spacecraft. The three components of thrust can be directly modeled as perturbing accelerations to determine their impact on the orbit over time.

2.1.3 Perturbations from gravitational harmonics

One major source of orbital perturbations for satellites in LEO is Earth gravitational Harmonics. The largest portion of these perturbations comes from zonal harmonics. Zonal harmonics describe the effect of Earth's oblateness. The non-uniform distribution of mass with changing latitude creates variations in the gravitational force experienced by the spacecraft. The bands of mass leading to these perturbations can be seen in Figure 2.3.

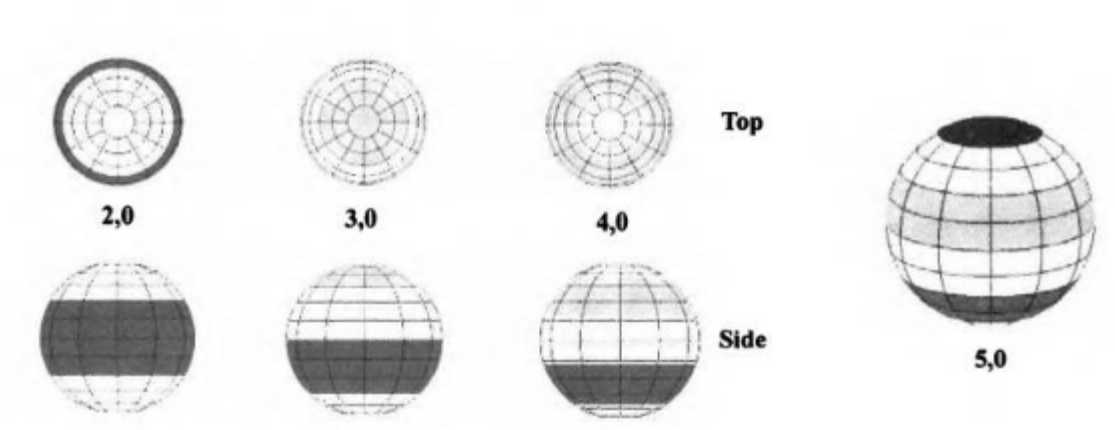

Figure 2.3: J2-J5 Zonal Harmonics[25]

Other sources of harmonics include sectoral and tesseral harmonics. Sectoral harmonics include the variation in Earth's mass distribution going across longitude. Tesseral harmonics model-specific regions of non-uniform mass distribution such as mountains. These higher-order harmonics are multiple orders of magnitude lower than J2 perturbations [25].

To determine the perturbation from these harmonics, the spherical harmonics potential function U is used. This equation is shown in equation 2.8.

$$U = \frac{\mu}{r}\left[1 - \sum_{t=2}^{\infty} J_t \left(\frac{R_\oplus}{r}\right)^t P_t\left[\sin\left(\phi_{gc_{sat}}\right)\right] + \sum_{t=2}^{\infty}\sum_{m=1}^{t}\left(\frac{R_\oplus}{r}\right)^t P_{t,m}\left[\sin\left(\phi_{gc_{sat}}\right)\right]\left\{C_{l,m}\cos\left(m\lambda_{sat}\right) + S_{l,m}\sin\left(m\lambda_{sat}\right)\right\}\right] \tag{2.8}$$

where $R_\oplus$ is the average radius of Earth, λ and ϕ are the longitude and latitude of the satellite, $P_{l,m}$ are the Legendre Polynomials with degree l and m,

The referenced spherical harmonics potential equation includes zonal, sectoral, and tesseral terms. The first term in the expression describes only the zonal potential, and the second term includes the tesseral and

sectoral harmonics. For brevity in this derivation, the tesseral and sectoral harmonics are ignored, and only the first term is considered. For Earth J2 Zonal harmonics $l = 2$ and $m = 0$. The resulting equation is shown in equation 2.9.

$$U = -\frac{\mu J_2}{r}\left(\frac{R_\oplus}{r}\right)^2 P_{2,0}\left[\sin\left(\phi_{gc_{sat}}\right)\right] \tag{2.9}$$

This form still includes the latitude of the spacecraft. To further simplify this expression, the expression can be written as shown in equation 2.10.

$$U = -\frac{\mu J_2}{r}\left(\frac{R_\oplus}{r}\right)^2 \frac{3}{2}\left(\sin^2(i)\sin^2(\omega + v) - \frac{1}{3}\right) \tag{2.10}$$

With this potential function, the perturbing accelerations can now be determined by taking the gradient in the direction of interest. The resulting vector gradients in the directions of interest are shown in

$$\alpha_x = -\frac{3\mu J_2 R_\oplus^2}{r^4}\left(\sin^2(i)\sin(\omega + \nu)\cos(\omega + \nu)\right) \tag{2.11}$$

$$\alpha_y = \frac{9\mu J_2 R_\oplus^2}{2r^4}\left(\sin^2(i)\sin^2(\omega + \nu) - \frac{1}{3}\right) \tag{2.12}$$

$$\alpha_z = -\frac{3\mu J_2 R_\oplus^2}{r^4}(\sin(\omega + \nu)\sin(i)\cos(i)) \tag{2.13}$$

These terms can now be directly input into Gauss' variation of parameter equations to model the effect of J2 perturbations on the orbit.

2.2 Modeling thrusting and steering

Traditional methods of modeling optimal low thrust maneuvers, while optimal, assume constant thrust and power availability. One such example is Edelbaum's solutions for optimal low thrust maneuvers for satellites [26]. Edelbaum's solutions assume continuous and uninterrupted thrusting. For pulsed electrical propulsion systems, the thrusting profile is not continuous. For small CubeSats, the limited power available to the propulsion system may lead to interruptions in thrusting as the satellite enters Earth's shadow. For an optimal thrusting profile for CubeSats, these constraints may need to be considered.

One solution to the continuous power and thrust problem is to use the Optimal Low-Thrust Maneuvers in Presence of Earth Shadow method determined by Colasardo and Casalino [27]. Their solution provides an optimal pointing and thrusting methodology using primer vector theory. This optimal steering profile takes into account the solar vector to achieve optimal quasi-circular low thrust orbital transfers considering power availability during shadowed regions. Using this steering profile, an optimal solution for a low-thrust orbital transfer maneuver can be determined for spacecraft without continuous power availability.

Chapter 3

Problem Setup and Simulation

There are various methods available to simulate the orbital dynamics and maneuvers in the presence of various perturbing bodies and forces. One option is to directly simulate the dynamics in a native environment using programming languages like C++, Python, or MATLAB. These direct simulation environments can be computationally intensive to run, may be subject to developmental errors, and implementations may have additional issues as the problem scales up. To avoid these issues, a commercial simulation environment was chosen that provides computationally efficient orbit modeling and allows efficient scaling of the problem.

3.1 Simulation Environment

For this work, a.i. Solutions FreeFlyer™ was used to model the orbital dynamics. FreeFlyer is a commercially available software package that allows engineers and scientists to develop and test complex dynamic simulations for space mission design. As a mission planning tool, FreeFlyer allows for orbit and environment modeling, including various sensors and maneuvers, as well as graphical visualization including

plotting and data tracking. The basis for FreeFlyer mission design lies in its native scripting language "FreeForm", a derivative of the Microsoft Visual Basic language. This scripting language allows for the design and implementation of complex algorithms and fully customizable mission profiles. Standard perturbations can also be easily modeled with the native inclusion of various astrodynamics models. Gravity models like JGM-2, EGM-96, and LP-165 allow for gravitational perturbations to be easily modeled. Atmospheric models like the Jacchia-Roberts, Harris-Priester and NRL-MSIS models allow for accurate modeling of aerodynamic effects. Magnetic effects can be modeled using the International Geomagnetic Reference Field magnetic field model[28]. This native functionality allows for high-fidelity simulations to be developed and tested in a rapid fashion. Various entities have used FreeFlyer to support space missions, including NASA, JSpOC, the US Air Force, and many others [29].

Individual design spaces in FreeFlyer are called Mission Plans. Each mission plan allows for the inclusion of any number of satellites, ground stations, and various other relevant objects. Constellations and other interdependent objects can be grouped for ease of control interfacing. The behavior of these systems can be modified with FreeForm script. FreeForm scripts are capable of creating and executing complex functions and can be used in the creation and tracking of complex mission-related objects. Scripts also allow for the implementation of real-world constraints.

3.1.1 Mission Plan

The DUPLEX FreeFlyer mission plan was created with six FreeForm scripts to handle the logic for the mission profile. These scripts handle the creation of various mission-related objects and logical functions. The **Setup** script sets up the mission environment and various UI elements for ease of processing information. The **Battery** script handles the creation of objects and functions related to the battery and power storage. The **Power Generation** script handles functions related to calculating the incoming power from various emitting sources and objects. The **Dynamics Procedure** script handles the creation of dynamics-related objects and sensors and functions controlling maneuvering logic and system dynamics. The **Procedures**

script handles the creation and execution of other system relevant functions and objects like reference frames and relevant external objects. Finally, the **Operate Maneuvers** script handles the logic and functions dictating the direct numerical simulation of the planned mission objectives and outputs and plots requested data. These scripts are shown in order of execution at runtime in Figure 3.1. The code for each of these scripts is provided in Appendix A.

#	Content
1	FreeForm: Setup
2	FreeForm: Battery
3	FreeForm: PowerGeneration
4	FreeForm: Dynamics Procedure
5	FreeForm: Procedures
6	FreeForm: Operate Maneuvers

Figure 3.1: FreeFlyer Mission Plan FreeForm Scripts in order of execution

Setup

The first script run on code execution is the setup script. This script controls the creation of various mission-related objects and functions. The script first handles the creation of various mission-relevant reference frames, with the creation of the Local Vertical Local Horizontal (LVLH) reference frame, Mean of J2000 Earth Equator Frame (MJ2000), and the Sun Centered (SC) frame. These frames can be seen in Figure 3.2.

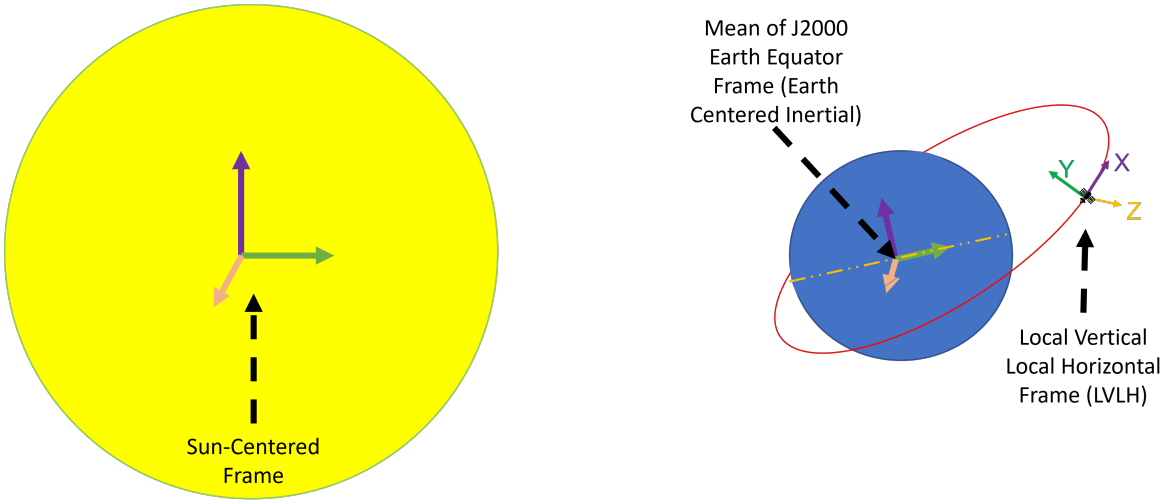

Figure 3.2: Examples of some mission-related reference frames

A spacecraft object is created with various spacecraft-related attributes. These attributes are organized into nine distinct categories. All of the spacecraft attribute categories are shown in Table 3.1.

Categories of Spacecraft Attributes
Orbit
Attitude
Physical Properties
Propagator
Force Model
Tanks
Thrusters
Sensors
Visualization
Proximity Zones

Table 3.1: Categories for defining Spacecraft Attributes

One such category is **Orbit**. This category includes the initialization of the spacecraft's initial position and defines various orbit-related attributes.Table 3.2 shows an example of the properties of the object as defined in the script.

Object Parameter	**Value**
Epoch Reference	Universal Coordinated Time (UTC)
Central Body	Earth
Orbital Element Type	Keplerian
Orbital Reference Frame	Mean of J2000 Earth Equator
Orbit Initial Semimajor Axis	6848 km
Orbit Initial Eccentricity	0 deg
Orbit Initial Inclination	51.8 deg
Orbit Initial RAAN	234.14 deg
Orbit Initial Argument of Periapse	0 deg
Orbit Initial True Anomaly	0 deg

Table 3.2: Spacecraft Orbit Attributes

A visualization of the orbit is shown in Figure 3.3.

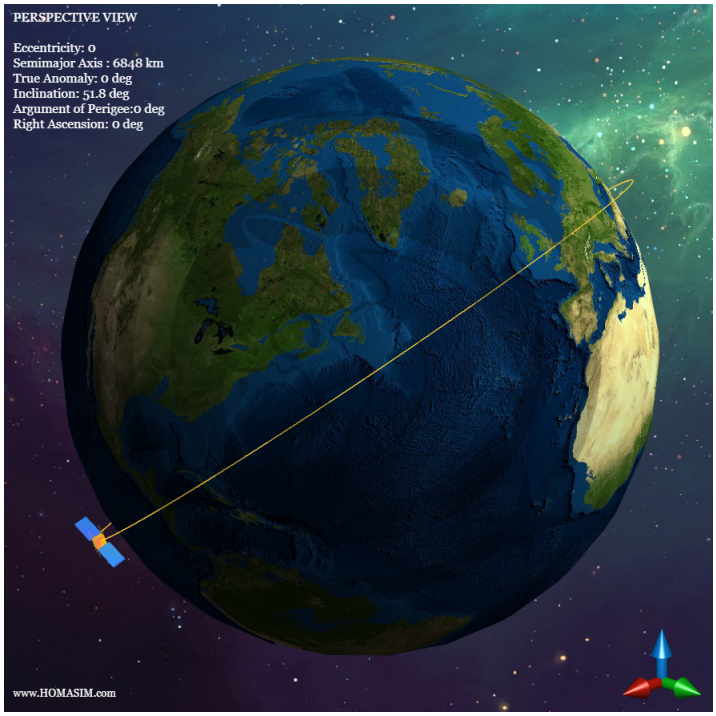

Figure 3.3: Orbit Visualization [30]

The corresponding ground track of this orbit can be seen in Figure 3.4.

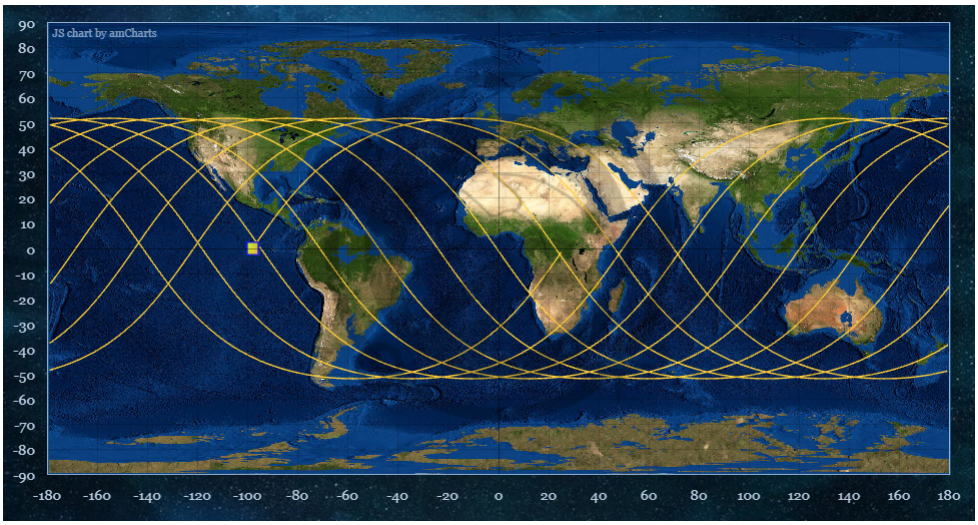

Figure 3.4: Orbit Ground Track [30]

The **Attitude** category holds all of the spacecraft's attitude-related attributes. The initial orientation and relative reference frame options are defined here. The **Physical Properties** holds the spacecraft's mass properties and other physical attributes. The **Propagator** holds attributes related to the direct numerical simulation of the various system dynamics and orbit propagation. In this scenario, the propagator of choice is an 8th-order Runge-Kutta method with 9th-order truncation error with a constant 15 second time step. This choice allows for high-fidelity system dynamics simulations to be computed. Equations 3.1 - 3.6, show the process for numerically integrating using a Runge-Kutta method.

$$\frac{dy}{dt} = f(y,t), y(t_0) = y_0; \tag{3.1}$$

$$y_{n+1} = y_n + \frac{t_{step}(k_1 + 2k_2 + 2k_3 + k_4}{6} \tag{3.2}$$

where,

$$k_1 = f(y_n, t_n) \tag{3.3}$$

$$k_2 = f(y_n + t_{step}k_1/2, t_n + t_{step}/2) \tag{3.4}$$

$$k_3 = f(y_n + t_{step}k_2/2, t_n + t_{step}/2) \tag{3.5}$$

$$k_4 = f(y_n + t_{step}k_3, t_n + t_{step}) \tag{3.6}$$

The **Force Model** category holds all of the perturbation-related attributes. This includes gravitationally relevant bodies, drag models, solar flux data, Earth zonal and tesseral perturbations, and other defined sources to be considered. In the case of this simulation, Earth, Moon, and Sun were considered gravitationally relevant, with major drag sources defined by the MSIS-2000 model and the F10.7 solar flux model.The **Tanks** category includes propellant related attributes. Propellant mass, density, and mass distribution are

defined here. The **Thrusters** category is where the spacecraft's thrusters are created and their thrusting properties are defined here. This includes specific impulse, propellant efficiencies, power efficiencies, and other thruster-related attributes. The **Sensors** category includes defined on-board sensors. This includes objects like attitude sensors with modeled noise and various other mission-relevant sensors. The **Visualization** and **Proximity Zones** handle the spacecraft's physical appearance and geometry. In the **Visualization** category, the appearance and reflective properties of each of the spacecraft external surfaces is defined. The **Proximity Zones** category defines the physical geometry and also includes attributes like object transparency and shadow generating objects. All together these categories in the **Setup** script setup and initialize the spacecraft and the environment in which it is operating.

Battery

The **Battery** script handles the creation of objects and functions related to the battery and power storage and consumption. This script includes functions that compute power storage and delivery efficiencies, power draw and efficiencies of various spacecraft components, and a state machine that defines various power-related attributes depending on the spacecraft's operating mode. The state machine differentiates between states such as active radio, active attitude control, thruster modes, and other mission-relevant operating modes under which power requirements could differ. This script effectively defines, controls, and tracks the spacecraft's available onboard power.

Power Generation

The **Power Generation** script handles the creation of objects and functions related to the power generation onboard the spacecraft. One major function in this script is that it computes the orientation between the different spacecraft solar panels and emitting bodies like the Sun, the Earth Albedo, and the Moon, and computes an efficiency and incident angle adjusted power input estimate for the solar panels.

The incoming light vectors are computed, compared to the solar panel normal vectors, and through various custom computations, an estimate of the input power can be determined. A simplified example of one such computation can be shown in Equation 3.7.

$$P_{in} = (\hat{r}_{sun} \cdot \hat{N}_{panel})\eta_{panel} A_{cell} \Phi_{sun} \tag{3.7}$$

Figure 3.5 shows the vectors that are used in this computation.

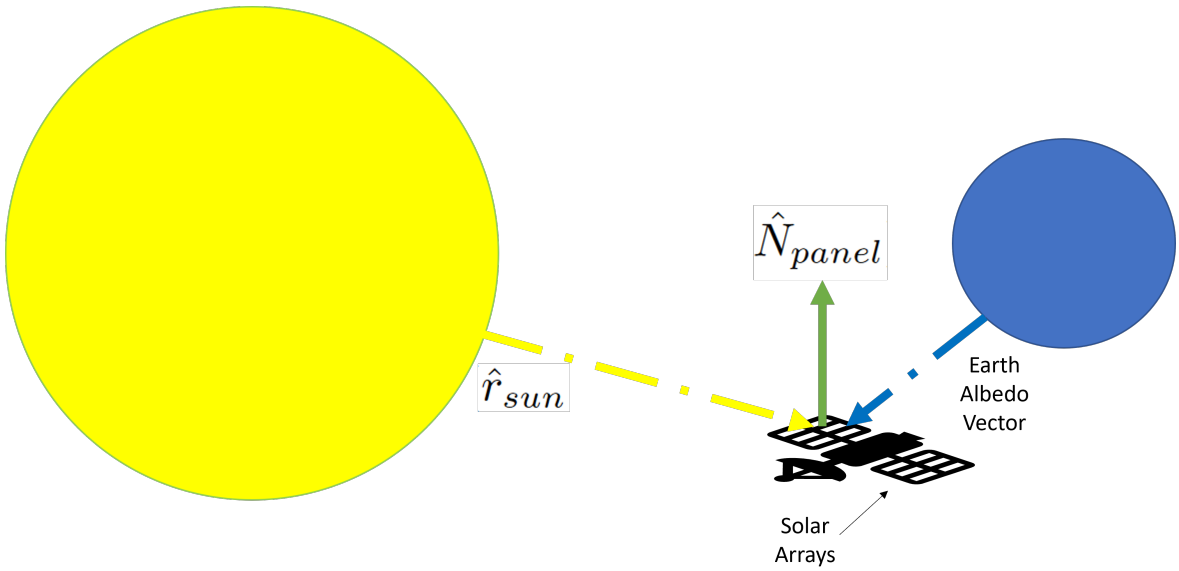

Figure 3.5: A simplified example of vectors used to estimate power

Dynamics Procedure

The **Dynamics Procedure** script handles all system dynamics and system pointing behavior. It features a set of functions defining pointing and thrusting schemes for each of the potential maneuvers. This function set includes maneuvers such as inclination change, eccentricity change, orbit raising, orbit lowering, orbit maintenance, and other relevant thruster operating modes. These functions schedule slewing maneuvers

and thruster burns based on various criteria, such as power availability.

Procedures

The **Procedure** script handles all minor procedures, like tracking vectors to relevant objects, sensor updates, and data management.

Operate Maneuvers

The **Operate Maneuvers** script handles all of the spacecraft's on-orbit behavior. This includes sequentially scheduling maneuvers, logic dictating when maneuvers can begin and end, tracking the spacecraft dynamics, calculating and plotting mission-relevant variables, and defining a finite state machine to iterate through based on a preset mission profile. This script handles all functions calls and maneuvers specific plotting and variable tracking. This is the script that loops and calls the other previously defined functions until the simulation ends. All of the generated plots, tables, and figures are produced by this script.

Chapter 4

Mission Analysis and Results

4.1 Mission Profile

CUA's DUPLEX CubeSat incorporates a Monofilament Vaporization Propulsion (MVP) thruster and a Fiber-fed Pulsed Plasma Thruster (FPPT). These thrusters have not yet flown in space and require flight testing to validate performance predictions, allowing these new polymer-based propellant systems to reach TRL 7 or higher. The mission profile starts with the deployment of DUPLEX from the ISS. DUPLEX will then execute several orbital maneuvers in low earth orbit (LEO), including orbit raising and lowering, orbit maintenance, inclination change, and deorbiting. The mission is divided into eight phases, where various maneuvers are performed using both propulsion systems independently to demonstrate the desired capabilities. The assumed start date of operations is November 15, 2022.

The phases of DUPLEX are shown in Figure 4.1 and indicate the location of the semi-major axis (SMA) relative to Earth when these phases are initiated

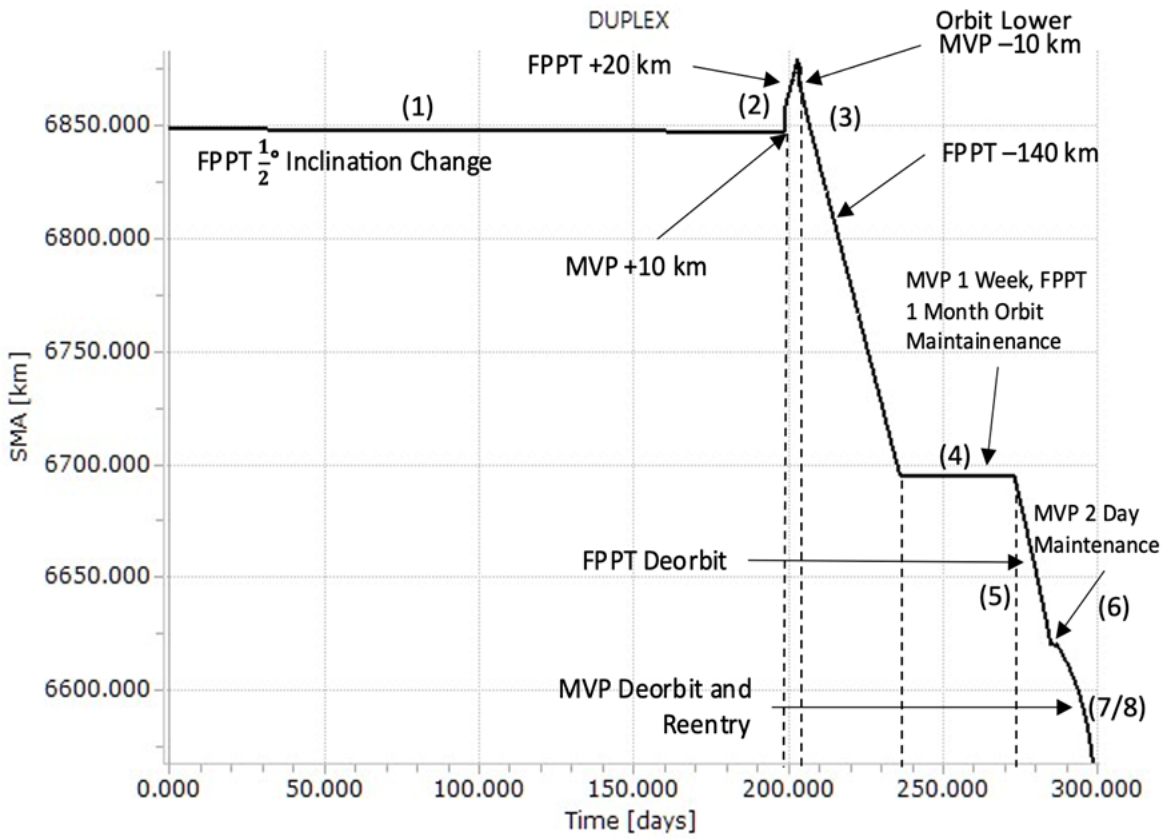

Figure 4.1: Mission Profile with 8 major stages highlighted

The phases are in order: (1) Inclination Change, (2) Orbit Raising, (3) Orbit Lowering, (4) High Altitude Orbit Maintain, (5) FPPT Deorbit, (6) Low Altitude Orbit Maintain, (7) MVP Deorbit, (8) Natural Decay and Reentry.

All maneuvers are performed in the LVLH reference frame. Figure 4.2 shows the orientation of the spacecraft relative to the LVLH frame during the orbit-raising phase.

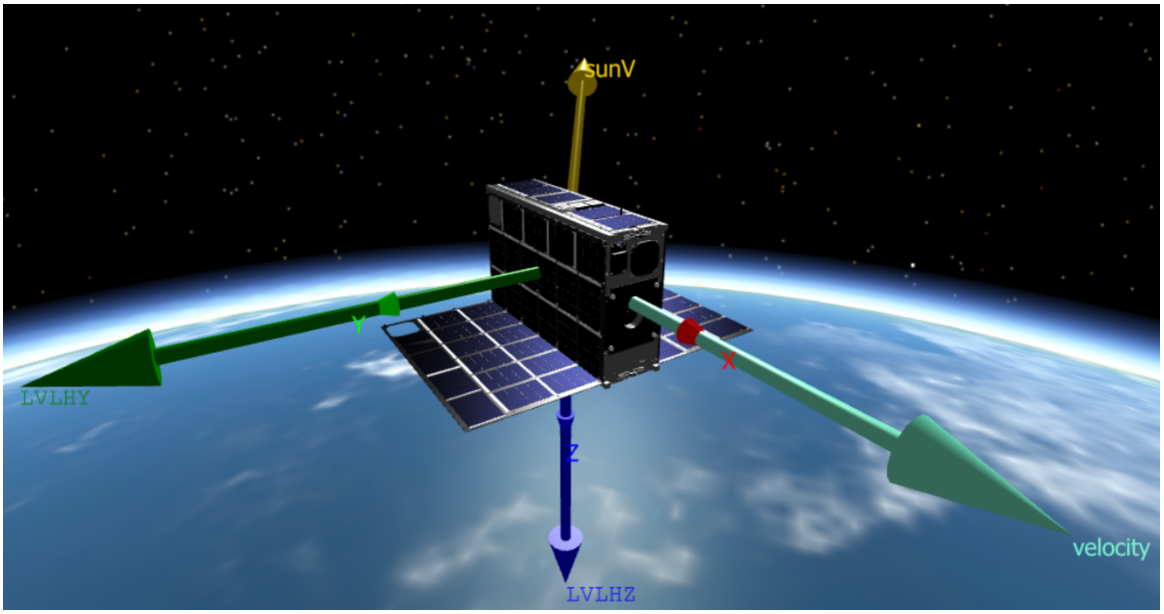

Figure 4.2: Orientation of the LVLH Frame relative to spacecraft

The orientation of the satellite is with respect to its orbit is defined by two angles, α, and β. This can be seen in Figure 4.3.

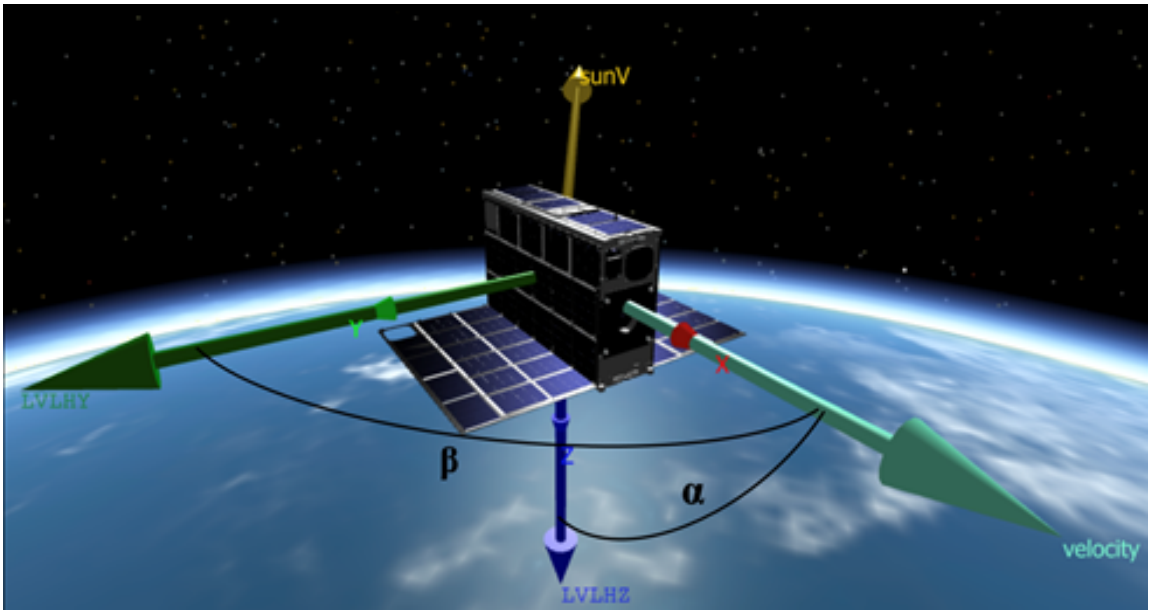

Figure 4.3: Orbit raising, lowering, maintenance, and deorbit maneuvers are performed with $\beta = 0°$, and inclination change maneuvers are performed at the descending node with $\beta = \pm 90°$ where the thrust and velocity vectors are perpendicular

4.1.1 Inclination Change

The first phase in the mission profile is the inclination change maneuver. This maneuver is accomplished by an FPPT burn to decrease the orbit inclination by 0.5°. The purpose of this maneuver is to change DUPLEX's inclination from that of the ISS to reduce the risk of conjunction and other related risks. The attitude during the mission profile is determined by Equations 4.1 and 4.2, where β represents the angle relative to the orbital plane.

$$\frac{di}{dt} = |\vec{f}|\frac{r}{h}\cos(\omega + \nu)\sin\beta \tag{4.1}$$

$$\beta = \frac{\pi}{2} \cos(\omega + \nu) \tag{4.2}$$

This β is found by finding the point at which the $\frac{di}{dt}$ is maximized. As such, these equations provide the maximum out-of-plane angle change for a low-thrust maneuver. This β angle is equal to 90 degrees at the descending node. To simplify slewing operations, the burns are commanded within ± 10 degrees of the descending node. This orientation can be seen in Figure 4.4.

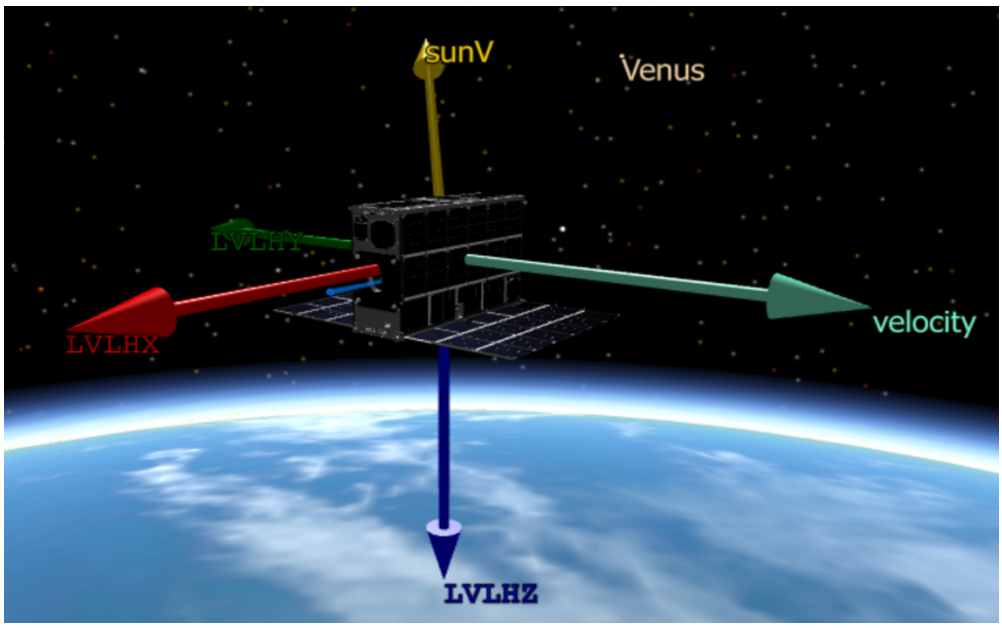

Figure 4.4: DUPLEX"s Orientation with respect to the LVLH frame at the descending node

The FPPT inclination change maneuver is commanded when battery power exceeds 65% and continues until battery power drops below 35%. This is further constrained by the thruster's heat rejection limitations, resulting in a duty cycle of 9.1% (2.1 hrs/day). Using estimated conditions and ignoring orbital perturbations, this phase is found to take 199 days. The inclination change maneuver uses 57 g of propellant. Table 4.1 lists the total impulse, propellant consumed, duty cycle, and time for this maneuver.

Propulsion System	Total Impulse [N-s]	Propellant Consumed [g]	Duty Cycle	Total Time Required [Days]	Propellant Remaining [g]
FPPT	1953	57	9.1%	199	793

Table 4.1: Characteristics of Inclination Change Maneuver

Figure 4.5 shows a day in the life of burn maneuvers and the inclination change in that time frame.

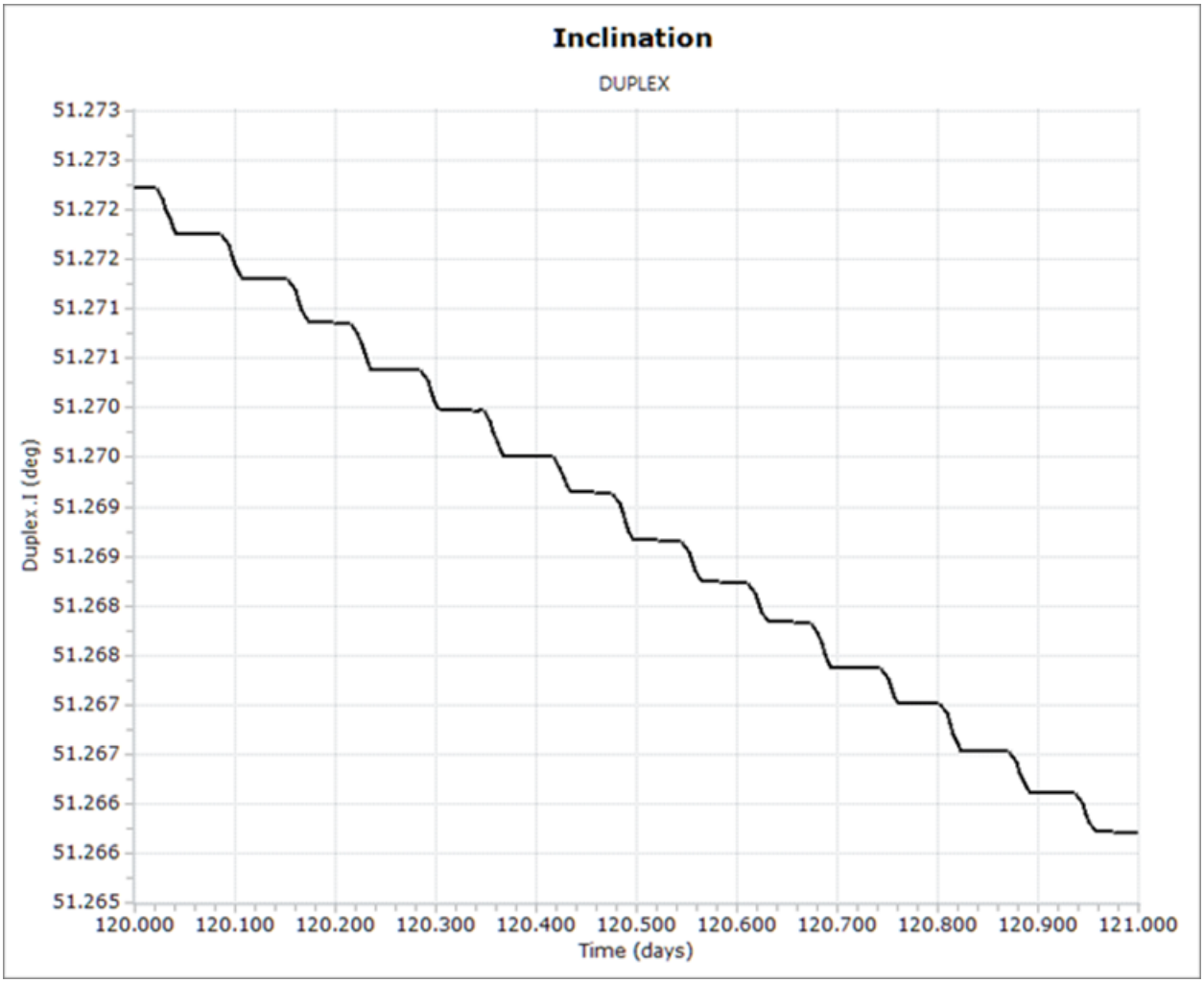

Figure 4.5: Inclination change over 1 day

4.1.2 Orbit Raising

The second phase in the mission profile is a 30 km orbit-raising maneuver. The purpose of this maneuver is to demonstrate both FPPT and MVP's orbit-raising capabilities. This maneuver is a 10 km orbit raise accomplished by an MVP burn, followed by an FPPT burn that raises the orbit by another 20 km. The burn direction is commanded to be in the direction of the velocity vector. Figure 4.6 shows the maneuver over its 4-day duration. With these low thrust maneuvers on a quasi-circular orbit, and with the burn direction in the ram direction, the orbit remains roughly circular. The MVP burn takes less than 1 day to complete, and the FPPT burn takes 4 days to complete.

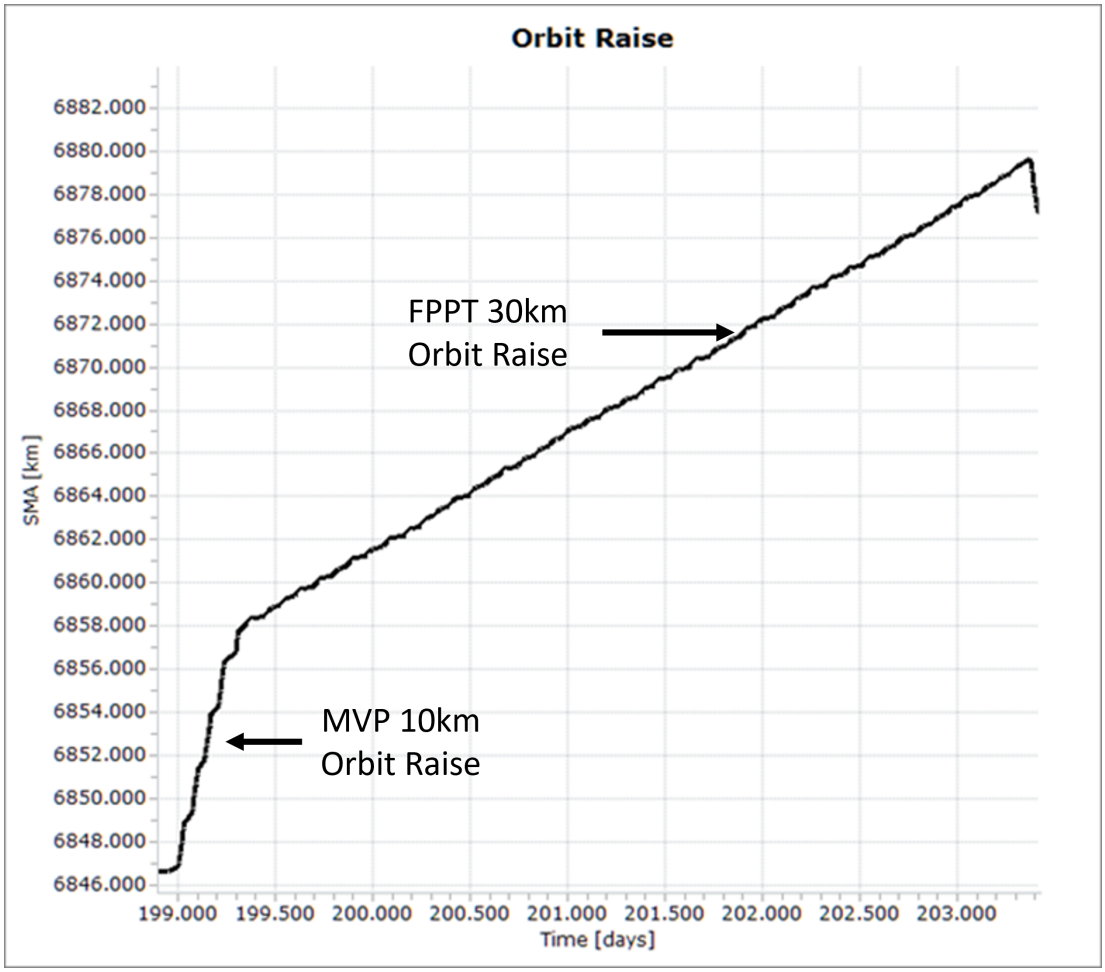

Figure 4.6: Semimajor Axis Profile for Orbit Raising Maneuver

Table 4.2 shows the total impulse, propellant consumed, and time per thruster for this maneuver to be completed. Similar to the inclination change maneuver, these burns occur when battery power exceeds 65% and continue until battery power falls to 32%. The 10 km MVP orbit raising burn uses 103 g of propellant, and the FPPT 30 km orbit raise uses 4 g of propellant.

Propulsion System	**Total Impulse [N-s]**	**Propellant Consumed [g]**	**Total Time Required [Days]**	**Propellant Remaining [g]**
MVP	54.7	103	0.44	330
FPPT	137.4	3.7	4.0	789.3

Table 4.2: Characteristics of Orbit Raising Maneuver

4.1.3 Orbit Lowering

The third phase in the mission profile is an orbit lowering maneuver to demonstrate MVP and FPPT's orbit lowering capabilities. The maneuver is accomplished with an MVP burn to lower the orbit altitude by 10 km, followed by an FPPT burn that lowers the orbit by another 140 km. The burn direction is set to be in the direction opposite that of the velocity vector. Figure 4.7 shows the maneuver over its 33-day duration.

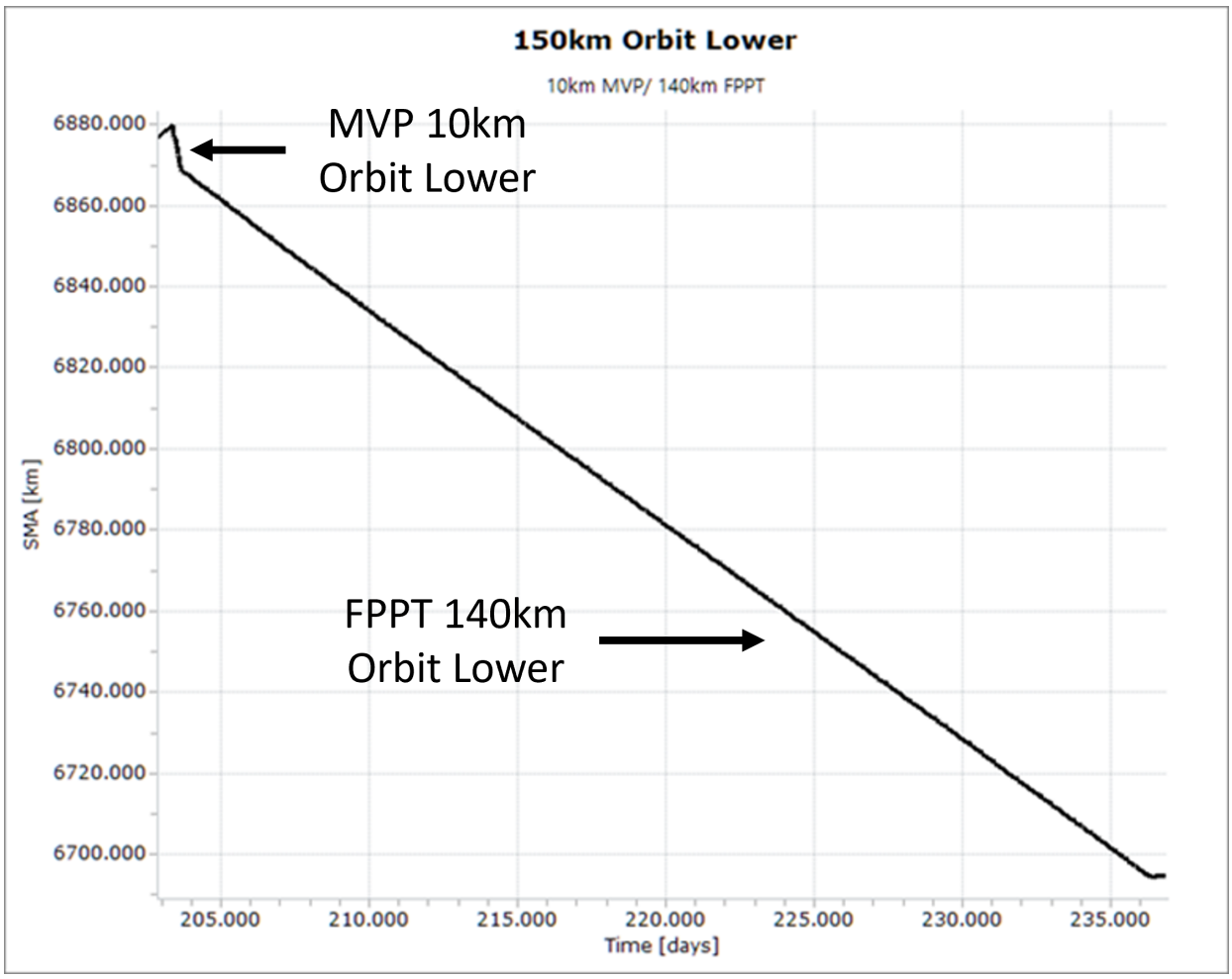

Figure 4.7: Semimajor Axis Profile for Orbit Lowering Maneuver

With these low thrust maneuvers on a quasi-circular orbit, and with the burn direction in the ram direction, the orbit remains roughly circular. The MVP burn takes less than 1 day to complete, and the FPPT burn takes 33 days to complete. Both thruster burns occur when available battery power exceeds 65% and continue until battery power falls to 32%. The 10 km MVP orbit lowering burn uses 102 g of propellant, and the 140 km FPPT orbit lowering burn uses 6.3 g of propellant. Table 4.3 shows the total impulse and time per thruster for this maneuver.

Propulsion System	Total Impulse [N-s]	Propellant Consumed [g]	Total Time Required [Days]	Propellant Remaining [g]
MVP	53.5	102	0.30	228
FPPT	236	6.3	33.0	783

Table 4.3: Characteristics for Orbit Lowering Maneuver

4.1.4 High Altitude Orbit Maintenance Maneuver

The fourth phase in the DUPLEX mission profile is a high-altitude orbit maintenance maneuver. The purpose of this maneuver is to demonstrate the thruster's capability to maintain a nominal altitude where remote sensing satellites could operate (325 km). The orbit's semimajor axis decreases due to drag from the atmosphere, and this maneuver offsets the atmospheric losses. This maneuver uses MVP for 7 days, followed by using FPPT for 30 days. The maintenance maneuver starts after the altitude has decreased by 200 m, after which the orbit is raised back up to the target altitude. Figure 4.8 shows a day in the life of the MVP high altitude orbit maintenance burn.

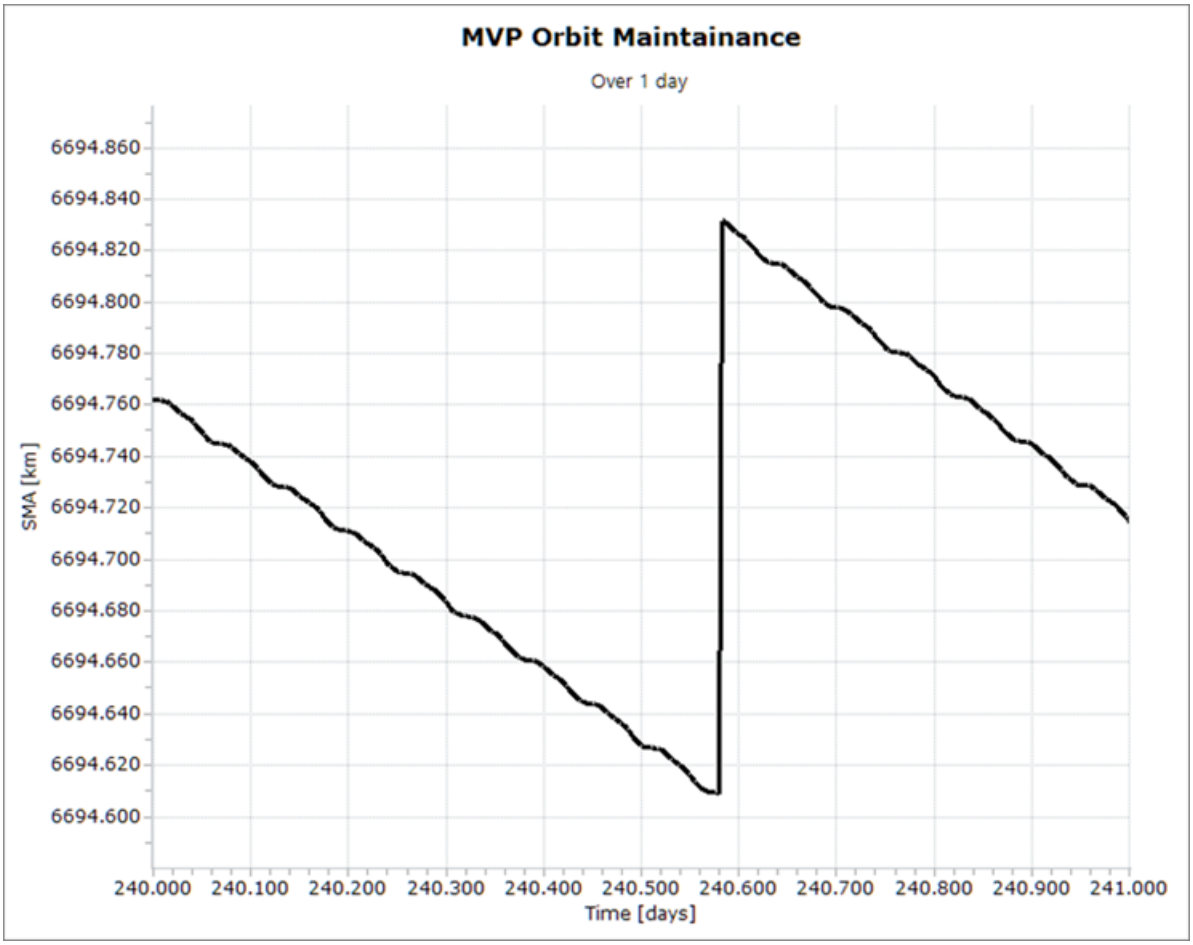

Figure 4.8: A close look at one day of MVP's High Altitude Orbit Maintenance

Figure 4.9 shows a day in the life of the FPPT high altitude orbit maintenance burn. The duty cycle is much lower here than in some of the other maneuvers, as not many burns need to be done to maintain altitude.

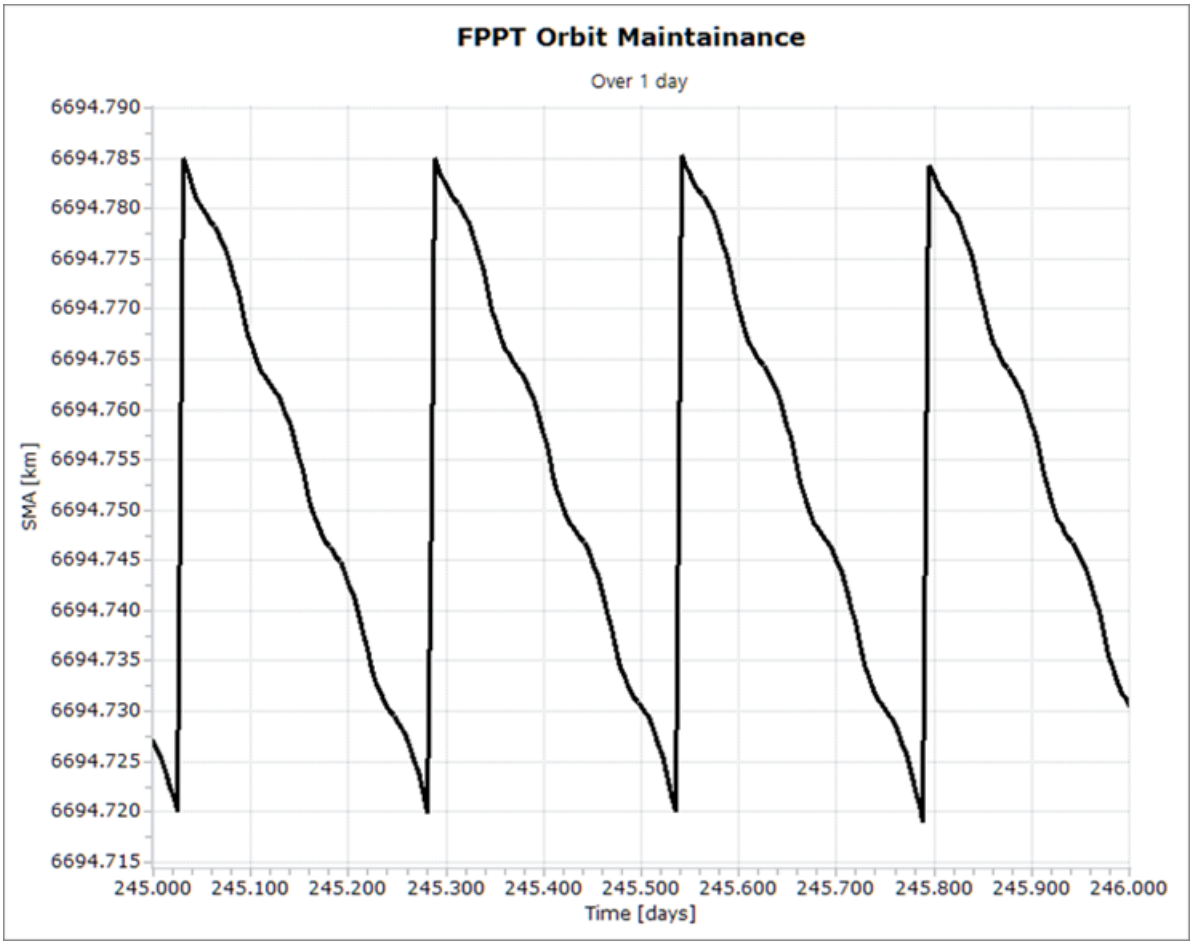

Figure 4.9: A close look at one day of FPPT's High Altitude Orbit Maintenance

The maneuver total impulse, propellant consumed, duty cycle, and total time for maneuver are shown in Table 4.4.

Propulsion System	**Total Impulse [N-s]**	**Propellant Consumed [g]**	**Duty Cycle**	**Total Time Required [Days]**	**Propellant Remaining [g]**
MVP	2.4	4.6	0.05%	7	223.4
FPPT	6.1	0.2	8.1%	30	782.8

Table 4.4: Characteristics for High Altitude Orbit Maintain Maneuver

4.1.5 FPPT Deorbit Maneuver

The fifth phase in the mission profile is an FPPT deorbit maneuver. This maneuver demonstrates FPPT's ability to deorbit a remote sensing satellite from a nominal operating altitude. FPPT burns as long as possible against the velocity vector, ending at an altitude of 240 km, and taking 14 days to complete. Figure 4.10 shows the deorbit maneuver over its 14-day span. The FPPT burn occurs when available battery power exceeds 65% and continues until battery power falls to 32%.

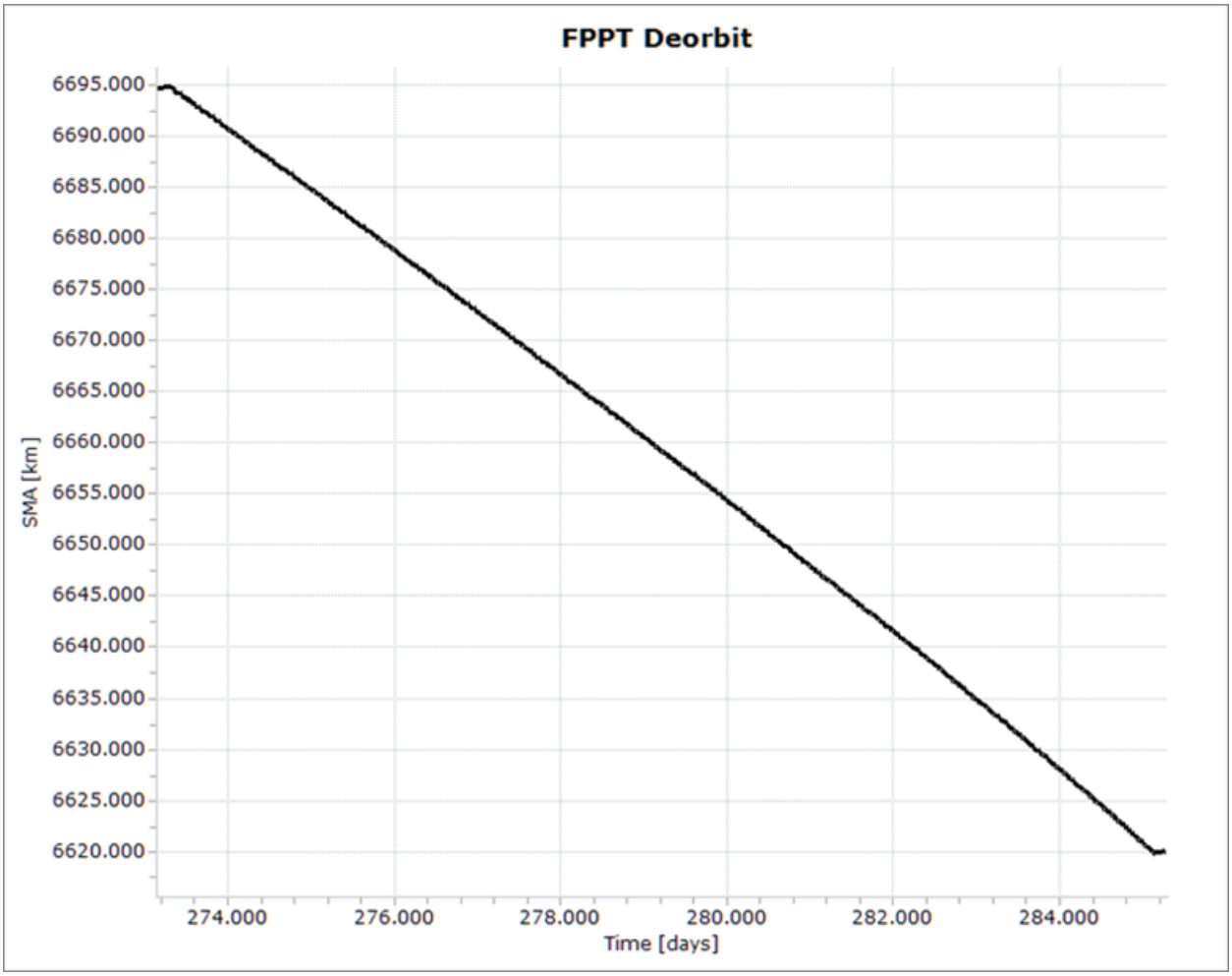

Figure 4.10: Semimajor Axis Profile for FPPT deorbit maneuver

Table 4.5 shows the total impulse, propellant consumed, duty cycle, and time for this maneuver.

Propulsion System	**Total Impulse [N-s]**	**Propellant Consumed [g]**	**Duty Cycle**	**Total Time Required [Days]**	**Propellant Remaining [g]**
FPPT	98	2.6	38.4%	13.8	780.2

Table 4.5: Characteristics for FPPT Deorbit Maneuver

4.1.6 MVP Low Altitude Orbit Maintenance Maneuver

The sixth phase in the mission profile is a low altitude orbit maintenance maneuver. This maneuver is intended to demonstrate the capability of MVP to maintain an orbit at a low altitude from which it would rapidly decay naturally due to atmospheric forces. In this maneuver, MVP maintains an orbit of 240 km for two days. The maintain maneuver waits until the altitude has decreased by 200 m after which the orbit is raised back up to the target altitude. Similar to the high-altitude maintenance maneuver, the thruster fires to make up losses from atmospheric drag. This results in a lower duty cycle than the altitude-changing maneuvers. Figure 4.11 shows a day in the life of the MVP Low Altitude Maintain burn.

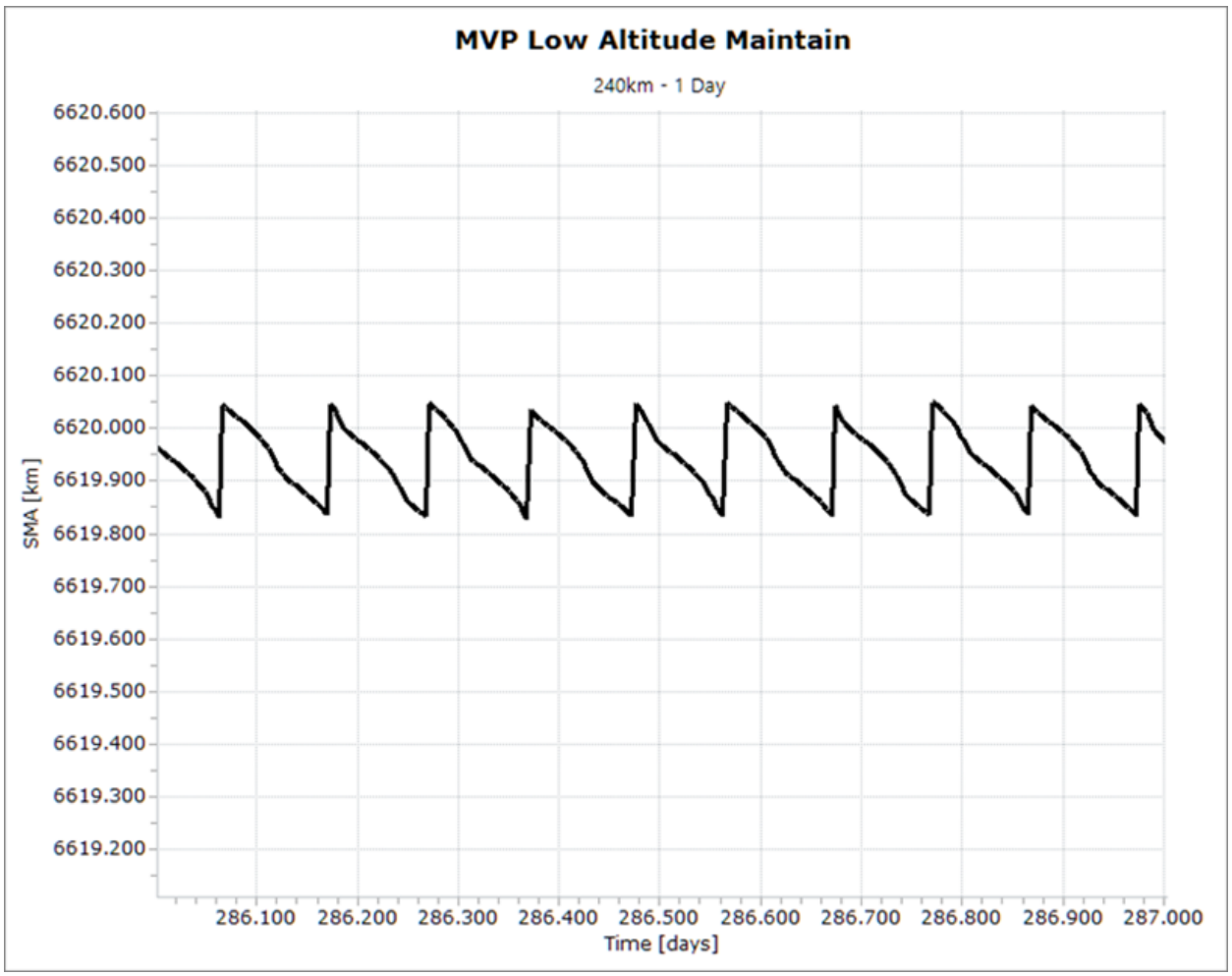

Figure 4.11: MVP low-altitude maintenance maneuver over 1 day at 240 km altitude

This maneuver uses 44 g of propellant. Table 4.6 shows the total impulse, propellant consumed, duty cycle, and time for this maneuver.

Propulsion System	**Total Impulse [N-s]**	**Propellant Consumed [g]**	**Duty Cycle**	**Total Time Required [Days]**	**Propellant Remaining [g]**
MVP	84	44	18%	2	179.4

Table 4.6: Characteristics for MVP low-altitude maintenance maneuver

Other simulations were performed to measure the lowest altitude at which MVP could effectively maintain an orbit, and results show that it could offset drag down to altitudes as low as 200km. Projecting the expected drag, it was shown that MVP could operate at even lower altitudes, but issues with effective system propagation at these elevated drag levels lead to inaccuracies in the numerical integrator results.

4.1.7 MVP Deorbit Maneuver and Natural Decay

The last two phases in the mission profile are an MVP deorbit maneuver followed by natural orbit decay. This maneuver is intended to demonstrate MVP's capability to deorbit a CubeSat after its mission has been completed. In this maneuver, MVP burns as much as possible against the velocity vector and ends when MVP runs out of propellant. DUPLEX's orbit naturally decays and the satellite re-enters the atmosphere. Figure 4.12 shows the deorbit maneuver.

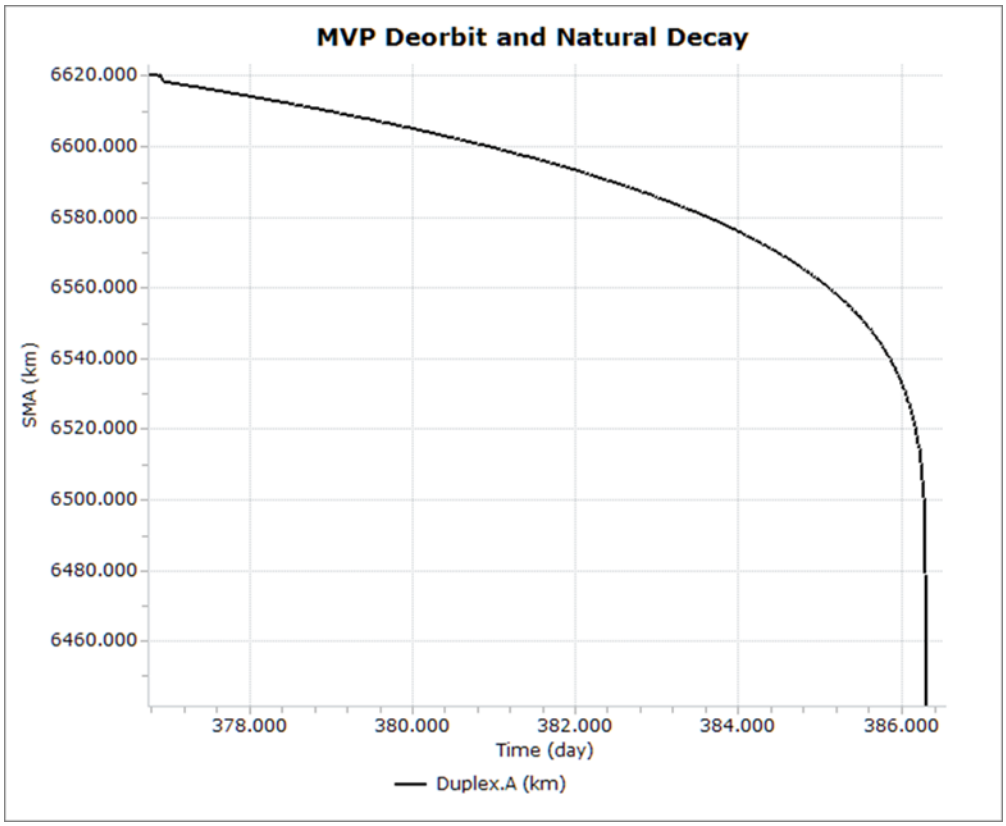

Figure 4.12: MVP final deorbit maneuver followed by natural decay and burn up in the atmosphere

The burn for MVP occurs when available battery power exceeds 65% and continues until battery power falls to 32%. Table 4.7 shows the total propellant remaining in both systems prior to this phase. MVP burns up all of its remaining propellant during this deorbit burn, but FPPT is left with the vast majority of its propellant remaining. This is due to FPPT's very high I_{sp} and total impulse. In other words, this demonstrates that the FPPT system is underutilized by this demonstration mission profile.

Propulsion System	**Propellant Remaining Before Final Deorbit [g]**	**Propellant Remaining [%]**
MVP	179.4	41.4
FPPT	780.2	91.7

Table 4.7: Propellant Remaining Prior to Final Deorbit Burn

In summary, these simulations show that MVP and FPPT are capable of operations in a number of different maneuvering modes, potentially enabling a variety of new CubeSat-based missions. The simulation results also show that these thrusters are capable of all of the possible maneuvers a CubeSat mission may need to meet mission objectives.

4.2 Simulation Sensitivity

The sensitivity of the simulation to numerical integrator variables such as the time-step is examined next. Longer time steps and simpler numerical integration methods could allow for faster computation if the resulting data is of high enough fidelity. Examining the inclination change segment as a reference, various test cases were defined to assess the impact of various simulation variables. The results of this sensitivity analysis are shown in Table 4.8

Numerical Method	**Time-Step(s)**	**Resulting Maneuver Duration (Days)**
RK45	300	130
RK89	300	133
RK45	100	161
RK89	100	162
RK45	50	183
RK89	50	183
RK45	15	199
RK89	15	199
RK45	10	199
RK89	10	199
RK45	5	199
RK89	5	199
RK45	1	199
RK89	1	199

Table 4.8: Simulation sensitivity to integration variables during inclination change maneuver

These results show that the simulations approach a convergent estimate of 199 days for the inclination change starting with a 15 second time step. This convergent behavior, which does not improve much at time steps shorter than 15 seconds, indicated that this time step is sufficient to realize high confidence in the presented simulation results.

4.3 Orbital Debris Assessment

An orbital debris assessment was performed to comply with NASA's spacecraft residual orbital debris requirements. The requirements comply with standards set by NASA-STD-8719.14 [31], which provides specific technical requirements for limiting orbital debris to comply with the NASA requirements. This assessment was completed with the use of the NASA Debris Assessment Software (DAS) v3.1. The compliance with orbital debris requirements assessed here are divided into five categories below.

4.3.1 Assessment of Spacecraft Debris during Normal Operations

The first assessment evaluates the risk of orbital debris during normal operation. The requirements evaluated here are 4.3-1 and 4.3-2. Requirement 4.3-1 is defined such that any orbital debris generated during normal operation of the spacecraft in LEO shall have an orbital life span of no more than 25 years. Since there is no debris generated by the deployment and normal operation of the spacecraft, this requirement is automatically satisfied. Requirement 4.3-2 relates to debris released in GEO, and since DUPLEX remains in LEO, this requirement is also automatically satisfied.

4.3.2 Assessment of Spacecraft Intentional Breakups and Potential of Explosions

This assessment evaluates the potential for explosions and intentional breakups. There are four requirements evaluated here: 4.4-1 through 4.4-4. These requirements along with their compliance statements are enumerated below.

Requirement 4.4-1: Limiting risk to other spacecraft from accidental explosions in deployment and mission operations while in orbit about Earth or the Moon:

The requirement states that for each spacecraft and launch vehicle orbital stage employed for a mission, the program or project shall demonstrate, via failure mode and effects analyses or equivalent analyses, that the integrated probability of explosion for all credible failure modes of each spacecraft and launch vehicle is less than 0.001. This requirement is satisfied because there are no pressure vessels onboard the spacecraft and the risk of battery explosion and failure is less than 0.001. The risk is further mitigated by the fact that even in the event of battery explosion, the cells are small with relatively low potential energy such that it's likely that most debris would be contained within the spacecraft in the small chance that an explosion occurs.

Requirement 4.4-2: Design for passivation after completion of mission operations while in orbit about Earth or the Moon

The requirement states that the design of all spacecraft and launch vehicle orbital stages shall include the ability to deplete all onboard sources of stored energy and disconnect all energy generation sources when they are no longer required for mission operations or post-mission disposal or control to a level which cannot cause an explosion or deflagration large enough to release orbital debris or break up the spacecraft. This requirement is satisfied by the fact that the spacecraft will be depleted of all energy prior to atmospheric entry.

Requirements 4.4-3/4: Limiting the long-term and short-term risk to other spacecraft from planned breakups

The requirements here check for the risks associated with planned breakups. Since there are no planned breakups, these requirements are automatically satisfied.

4.3.3 Assessment of Spacecraft Potential for On-Orbit Collisions

This section addresses requirements associated with potential collisions based on its orbit. The collision probability was determined using the DAS software.

Requirement 4.5-1: Limiting debris generated by collisions with large objects when operating in Earth Orbit

The requirement here states that "for each spacecraft and launch vehicle orbital stage in or passing through LEO, the program or project shall demonstrate that, during the orbital lifetime of each spacecraft and orbital stage, the probability of accidental collision with space objects larger than 10 cm in diameter is less than 0.001." This was evaluated with the DAS software with mission-relevant orbital parameters. DUPLEX meets the requirement at the following tested altitudes: 450, 425, 400, 375, 350, 325, 300, 275, 250. All cases reported a collision probability less than or equal to 0.00001.

Requirement 4.5-1: Limiting debris generated by collisions with large objects when operating in Earth Orbit

The requirement here states that "for each spacecraft, the program or project shall demonstrate that, during the mission of the spacecraft, the probability of accidental collision with orbital debris and meteoroids sufficient to prevent compliance with the applicable post-mission disposal requirements is less than 0.001." This was tested with the DAS software at various mission-relevant orbital parameters. DUPLEX meets the requirement at the following tested altitudes: 450, 425, 400, 375, 350, 325, 300, 275, 250. All cases report a collision probability less than or equal to .00001.

4.3.4 Assessment of Spacecraft Post-mission Disposal Plans and Procedures

The requirements in this section deal with post-mission disposal and deorbiting.

Requirement 4.6-1: Disposal for space structures passing through LEO

This requirement mandates that the spacecraft has a planned and approved disposal option. The approved disposal options are Atmospheric Re-entry, Storage Orbit, or Direct Retrieval. The DUPLEX spacecraft will reenter the atmosphere after its mission and will burn up on re-entry.

Requirements 4.6-2/3: Disposal for space structures near GEO/Between LEO and GEO.

These requirements deal with the disposal of any space structures deployed above LEO. Since DUPLEX is remaining in LEO for its mission, these requirements are automatically satisfied.

4.3.5 Assessment of Spacecraft Reentry Hazards

This section of the orbital debris requirements deals with risks associated with atmospheric reentry. The major requirement in this section is to determine the risk of human casualty with the planned mission disposal plan.

Requirement 4.7-1: Limit the risk of human casualty

The requirement stated that for uncontrolled reentry, the risk of human casualty from surviving debris shall not exceed 0.0001 (1:10,000). This potential for human casualty is assumed for any object with

impacting kinetic energy in excess of 15 Joules. To get an estimate for the casualty risk, the DAS software was used. According to the DAS software, the only component of interest for human casualty is the satellite's reaction wheels. The software estimates a risk of human casualty of 1:33500. The output from the software is tabulated in Table 4.9.

Surviving Component	**Original Mass (kg)**	**Impact Kinetic Energy (J)**	**Casualty Area (m^2)**	**Spacecraft Risk of Human Casualty**
Reaction Wheels	0.0904	186	1.16	
Fasteners	0.020	4.04	2.5	
Ceramic	0.0087	2.09	0.39	
Heater Block	0.005	0.60	0.38	
Bulkhead	0.0034	0.41	0.38	
Heater Cartridge	0.000397	0.040	0.37	
Sep Switches	0.002	0.28	1.5	
RTD HEL-707	0.00012	0.003	0.37	
Total Risk				1:33500

Table 4.9: Estimates for which components may survive atmospheric entry and estimated impact energy and risks for human casualty

4.4 Estimating Star Tracker Availability

Another study that was conducted was to estimate the availability of the DUPLEX star tracker. Availability is compromised if the image sensor in the star tracker faces direct sunlight and is overexposed and unable to determine the spacecraft's orientation. To get an estimate of the tracker time available per orbit, the geometry of the star tracker was simulated in FreeFlyer and the sensor availability was found by computing the period of time the Sun vector enters the star tracker baffle geometry. Figure 4.13 shows a simple diagram illustrating what the simulation considers obscuring the sensor.

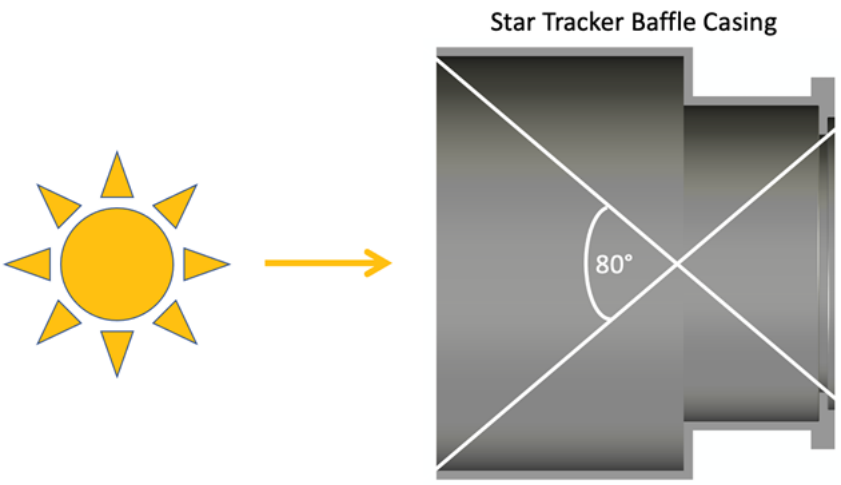

Figure 4.13: A simple illustration of the star tracker geometry blinded by Sun

From the simulation, a table of availability is generated and the availability of the star tracker during the various mission phases can be determined. This availability for all of the phases can be seen in Table 4.10.

Phase	**Availability (%)**	**Avg. Time Blinded Per Orbit (min)**
1 – Inclination Change	92.2	6
2/3 – Orbit Raising/Lowering	96.3	5
4 – Orbit Maintenance	92.4	5
5/6/7 – Deorbit	93.9	4

Table 4.10: Availability of the star tracker during the DUPLEX mission profile

4.5 Estimating Conjunction Avoidance Capability

Another study that was conducted determined the conjunction avoidance capability of the DUPLEX spacecraft. This study looks at the time before a conjunction that the spacecraft needs to command a divert maneuver to avoid conjunction. Preliminary simulations were performed to quantify the collision avoidance time threshold for the two thruster systems. This test quantified the average minimum warning time in each

mission phase to achieve a 200 m divert. With MVP, DUPLEX is able to divert 200 m and reduce collision probability with a warning of 60 minutes in all mission phases. With FPPT, DUPLEX is able to divert 200 m and reduce collision probability with a warning time of 250 minutes in all mission phases. A more detailed analysis by mission phase determining the average minimum time to divert in each phase is shown Table 4.11.

Phase	**MVP Avg. Avoidance Warning Time [min]**	**FPPT Avg. Avoidance Warning Time [min]**
1– Inclination Change	21	51
2/3– Orbit Raising/Lowering	23	55
4– Orbit Maintenance	31	67
5/6/7– Deorbit	56	241

Table 4.11: Required Advance Warning to Perform 200 m Avoidance Maneuver during DUPLEX Mission phases

Chapter 5

Conclusions

This thesis explored methods for simulating the orbit of the DUPLEX CubeSat incorporating models of its two novel propulsion systems. To perform this analysis, a theory for modeling low-thrust power restricted CubeSats was reviewed and the variables impacting the fidelity and optimality of the simulation were identified. A method for modeling environmental perturbations was also reviewed and the impact of these perturbations was determined using the FreeFlyer simulation tool.

In this investigation, a mission profile for the DUPLEX mission was conceived, a high-fidelity dynamics simulation was created, and the results were evaluated to determine the capabilities of the two onboard propulsion systems. The results show that DUPLEX spacecraft will be capable of meeting all of the stated mission requirements.

It is important to note that the DUPLEX mission is a feasibility demonstration for two novel electric propulsion systems for use in CubeSats, and these two systems have not yet been tested operationally in the space environment. Upon a successful demonstration, this model and simulation can be updated with real flight data to allow the thruster models to be re-used for future mission evaluations.

References

[1] W. Shiroma, L. Martin, J. Akagi, *et al.*, "Cubesats: A bright future for nanosatellites," *Open Engineering*, vol. 1, pp. 9–15, Mar. 2011. DOI: 10.2478/s13531-011-0007-8.

[2] S. Lee, A. Hutputanasin, and W. Lan, *Cubesat design specification - spaceq*. [Online]. Available: https://www.spaceq.ca/wp-content/uploads/2017/07/cds_rev13_final2.pdf.

[3] R. Nugent, R. Munakata, A. Chin, R. Coelho, and P. Puig-suari, "Cubesat: The pico-satellite standard for research and education," Sep. 2008. DOI: 10.2514/6.2008-7734.

[4] J. Straub, "Cubesats: A low-cost, very high-return space technology," 2012.

[5] B. Yost and S. Weston, *State-of-the-art small spacecraft technology*, Oct. 2021. [Online]. Available: https://ntrs.nasa.gov/citations/20210021263.

[6] D. M. Goebel and I. Katz, *Fundamentals of electric propulsion ion and hall thrusters*. Wiley, 2008.

[7] H. Burkhardt, M. Sippel, G. Krülle, *et al.*, "Evaluation of propulsion systems for satellite end-of-life de.orbiting," Jul. 2002, pp. 1–11. DOI: 10.2514/6.2002-4208.

[8] D. Gibbon, C. Underwood, M. Sweeting, and R. Amri, "Cost effective propulsion systems for small satellites using butane propellant," *Acta Astronautica*, vol. 51, pp. 145–152, Jul. 2002. DOI: 10.1016/S0094-5765(02)00074-7.

[9] D. Faber, A. Overlack, W. Welland, L. Vliet, W. Wieling, and F. Nardini, "Nanosatellite deorbit motor," Aug. 2013.

[10] I. 2019-A, D. M. King, R. L. Burton, and D. L. Carroll, "Fiber-fed pulsed plasma thruster (fppt) for small satellites," 2019.

[11] R. L. Burton, C. A. Woodruff, D. M. King, and D. L. Carroll, "Analysis of fiber-fed pulsed plasma thruster performance," *Journal of Propulsion and Power*, vol. 37, no. 1, pp. 176–178, 2021. DOI: 10.2514/1.B38114.

[12] C. Woodruff, D. L. Carroll, D. M. King, R. L. Burton, and N. Hejmanowski, "Monofilament vaporization propulsion (mvp) - cubesat propulsion system with inert polymer propellant," 2018.

[13] J. Puig-Suari, C. Turner, and W. Ahlgren, "Development of the standard cubesat deployer and a cubesat class picosatellite," in *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, vol. 1, 2001, 1/347–1/353 vol.1. DOI: 10.1109/AERO.2001.931726.

[14] C. Burkhard and S. Weston, *The evolution of cubesat spacecraft platforms - ntrs.nasa.gov*. [Online]. Available: https://ntrs.nasa.gov/api/citations/20210021348/downloads/Platforms_paper_final.pdf.

[15] [Online]. Available: https://mars.nasa.gov/internal_resources/344/.

[16] *Jpl marco micro cubesat propulsion system*, Sep. 2019. [Online]. Available: https://cubesat-propulsion.com/jpl-marco-micro-propulsion-system/.

[17] *Gms-t*. [Online]. Available: https://space.skyrocket.de/doc_sdat/gms-t.htm.

[18] B. Dunbar, *Technology readiness level*, May 2015. [Online]. Available: https://www.nasa.gov/directorates/heo/scan/engineering/technology/technology_readiness_level.

[19] A. Tummala and A. Dutta, "An overview of cube-satellite propulsion technologies and trends," *Aerospace*, vol. 4, Dec. 2017. DOI: 10.3390/aerospace4040058.

[20] *Plasma accelerators/magnetoplasmadynamic (mpd)thrusters*.

[21] C. Montag, H. Burghaus, G. Herdrich, and T. Schönherr, "Development of a new pulsed plasma thruster and a brief introduction of a planned test facility," Oct. 2016.

[22] A. Rezaeiha and F. Mankavi, "A study of the effect of input power on liquified-gas resistojet performance," Mar. 2012.

[23] C. Woodruff. [Online]. Available: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=4083&context=smallsat.

[24] L. Trichtchenko, L. Nikitina, A. Trishchenko, and L. Garand, "Highly elliptical orbits for arctic observations: Assessment of ionizing radiation," *Advances in Space Research*, vol. 54, Dec. 2014. DOI: 10.1016/j.asr.2014.09.012.

[25] D. A. Vallado and W. D. McClain, *Fundamentals of astrodynamics and applications*. Microcosm Press, 2013.

[26] T. N. EDELBAUM, "Propulsion requirements for controllable satellites," *ARS Journal*, vol. 31, no. 8, pp. 1079–1089, 1961. DOI: 10.2514/8.5723. eprint: https://doi.org/10.2514/8.5723. [Online]. Available: https://doi.org/10.2514/8.5723.

[27] G. Colasurdo and L. Casalino, "Optimal low-thrust maneuvers in presence of earth shadow," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. DOI: 10.2514/6.2004-5087. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2004-5087. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2004-5087.

[28] *Freeflyer® software*, Jan. 2021. [Online]. Available: https://ai-solutions.com/freeflyer-astrodynamic-software/.

[29] [Online]. Available: http://www.satnews.com/story.php?number=2117338379.

[30] *Homa*. [Online]. Available: http://en.homasim.com/.

[31] *Std-8719.14*. [Online]. Available: https://standards.nasa.gov/standard/nasa/nasa-std-871914.

Appendix A

FreeFlyer Code

```
// Console Setup
Console.BackColor = ColorTools.Black;
Console.CurrentTextColor = ColorTools.White;
Console.Dimension = 42;
Console.DockMode =3;
Console.Show();
Console.WordWrap =1;
Console.WindowTitle = "Duplex Con-Ops Analysis";



WindowOverlay wo;
Variable i;
// UI Data
OutputLayout.ShowNewWindowBehavior = 3;
OutputLayout.SetWindowPosition(ViewWindow1.ID,1,0,1,0);
```

```
OutputLayout.SetWindowSize(ViewWindow1.ID,1,1,1,1);
OutputLayout.SetWindowFrameVisibility(ViewWindow1.ID,0);

wo.RemoveAllShapes();
For i = 0 to 7;
    wo.AddShape();
    wo.Shapes[i].TextOptions.AlignmentHorizontal = 0;
    wo.Shapes[i].TextOptions.TextColor = ColorTools.Aqua;
    wo.Shapes[i].TextOptions.Font.Bold = 1;
End;

wo.Shapes[0].SetOrigin(0,0.01,0);
wo.Shapes[0].SetSize(0,0.16667,0.05);
wo.Shapes[1].SetOrigin(0,0.01,0.05);
wo.Shapes[1].SetSize(0,0.16667,0.05);
wo.Shapes[2].SetOrigin(0,0.01,0.1);
wo.Shapes[2].SetSize(0,0.16667,0.05);
wo.Shapes[3].SetOrigin(0,0.17667,0);
wo.Shapes[3].SetSize(0,0.18334,0.05);
wo.Shapes[4].SetOrigin(0,0.17667,0.05);
wo.Shapes[4].SetSize(0,0.18334,0.05);
wo.Shapes[5].SetOrigin(0,0.17667,0.1);
wo.Shapes[5].SetSize(0,0.18334,0.05);
wo.Shapes[0].SetOrigin(0,3.93*0.18,0.07);
wo.Shapes[0].SetSize(0,0.16667,0.03);
wo.Shapes[0].TextOptions.TextColor = ColorTools.Green;
```

```
wo.Shapes[1].SetOrigin(0,3.93*0.18,0.12);
wo.Shapes[1].SetSize(0,0.16667,0.03);
wo.Shapes[1].TextOptions.TextColor = ColorTools.Green;

wo.Shapes[2].SetOrigin(0,3.93*0.18,0.17);
wo.Shapes[2].SetSize(0,0.16667,0.03);
wo.Shapes[2].TextOptions.TextColor = ColorTools.Blue;

wo.Shapes[3].SetOrigin(0,4.85*0.17667,0.07);
wo.Shapes[3].SetSize(0,0.18334,0.03);
wo.Shapes[3].TextOptions.TextColor = ColorTools.Blue;

wo.Shapes[4].SetOrigin(0,4.85*0.17667,0.12);
wo.Shapes[4].SetSize(0,0.18334,0.03);
wo.Shapes[4].TextOptions.TextColor = ColorTools.Red;

wo.Shapes[5].SetOrigin(0,4.85*0.17667,0.17);
wo.Shapes[5].SetSize(0,0.18334,0.03);
wo.Shapes[5].TextOptions.TextColor = ColorTools.Red;
// Title Block
wo.Shapes[6].SetOrigin(0,3.75*0.202,0.02);
wo.Shapes[6].SetSize(0,0.18334,0.03);
wo.Shapes[6].TextOptions.TextColor = ColorTools.White;
// Orbit Average Power
wo.Shapes[7].SetOrigin(0,3.75*0.200,0.22);
wo.Shapes[7].SetSize(0,0.18334,0.03);
wo.Shapes[7].TextOptions.TextColor = ColorTools.White;
```

```
wo.AddShape();
wo.Shapes[8].Type = "Rectangle";
wo.Shapes[8].SetOrigin(1,970,4);
wo.Shapes[8].SetSize(0,0.32,0.26);
wo.Shapes[8].RectangleOptions.SetCornerRadius(0,0.15);
wo.Shapes[8].RectangleOptions.BorderWidth = 0;
wo.Shapes[8].RectangleOptions.FillColor = ColorTools.Black;
wo.Shapes[8].RectangleOptions.FillOpacity = 0.5;
ViewWindow1.AddObject(wo);
```

```
Define Procedure battery(Spacecraft Duplex, Variable Power, Variable Battery,
    Variable Thruster_State, TimeSpan TimeStep);
        If (Thruster_State == 0); // Charge Battery
                Battery = Battery + Power * TimeStep.ToSeconds/3600;
                Battery = Battery - 0.140 * TimeStep.ToSeconds/3600; // bus
                    systems [Wh]
                Battery = Battery - 1.5 * TimeStep.ToSeconds/3600; // MAI-401
                    ADCS [Wh]
                Battery = Battery - 0.420* TimeStep.ToSeconds/3600; // Rx [Wh]
                Battery = Battery - 0.14* TimeStep.ToSeconds/3600; // Black
                    Box [Wh]
```

```
        If (Duplex.ElapsedTime.ToSeconds%30 == 0);
                Battery = Battery - 1.106* TimeStep.ToSeconds/3600;
        End;
End;

If (Thruster_State == 1); // FPPT On
        Battery = Battery - 30* TimeStep.ToSeconds/3600; // FPPT [Wh]
        Battery = Battery - 0.140* TimeStep.ToSeconds/3600; // bus
            systems [Wh]
        Battery = Battery - 0.528* TimeStep.ToSeconds/3600; // MAI-401
             ADCS [Wh]
        Battery = Battery - 2.18* TimeStep.ToSeconds/3600; // Duplex [
            Wh]
        Battery = Battery - 0.14* TimeStep.ToSeconds/3600; // Black
            Box [Wh]
        If (Duplex.ElapsedTime.ToSeconds%30 == 0);
                Battery = Battery - 1.106* TimeStep.ToSeconds/3600;
        End;

End;

If (Thruster_State == 2); // MVP On
        Battery = Battery - 40* TimeStep.ToSeconds/3600; // MVP [Wh]
        Battery = Battery - 0.140* TimeStep.ToSeconds/3600; // bus
            systems [Wh]
        Battery = Battery - 0.528* TimeStep.ToSeconds/3600; // MAI-401
             ADCS [Wh]
```

```
            Battery = Battery - 2.18* TimeStep.ToSeconds/3600; // Duplex [
                Wh]
            Battery = Battery - 0.14* TimeStep.ToSeconds/3600; // Black
                Box [Wh]
            If (Duplex.ElapsedTime.ToSeconds%30 == 0);
                    Battery = Battery - 1.106* TimeStep.ToSeconds/3600;
            End;
        End;

        If (Thruster_State == 5); // FPPT_I On
            Battery = Battery - 120* TimeStep.ToSeconds/3600; // MVP [Wh]
            Battery = Battery - 0.140* TimeStep.ToSeconds/3600; // bus
                systems [Wh]
            Battery = Battery - 0.528* TimeStep.ToSeconds/3600; // MAI-401
                 ADCS [Wh]
            Battery = Battery - 2.18* TimeStep.ToSeconds/3600; // Duplex [
                Wh]
            Battery = Battery - 0.14* TimeStep.ToSeconds/3600; // Black
                Box [Wh]
            If (Duplex.ElapsedTime.ToSeconds%30 == 0);
                    Battery = Battery - 1.106* TimeStep.ToSeconds/3600;
            End;
        End;

EndProcedure;
```

```
Global Variable DOTPYS;
Global Variable DOTNYS;
Global Variable DOTPXS;
Global Variable DOTNXS;
Global Variable DOTPZS;
Global Variable DOTNZS;
Global Variable EdgeLossAngle = 85; // Degrees
Global Variable ExtensionAngle = 40; // Increased visibility of Deployable
    panels
Global Variable meanPower = 1366.7;
Global Variable meanSunDistance = 149530000; // 1 [AU]
Global Variable Efficiency = 0.18;
//Global Variable CellArea = 0.110*0.073;
Global Variable CellArea = 0.03*0.0665*3*2;
Global Variable EarthAlbedo = 0.10;



Define Procedure PowerGeneration(Spacecraft Duplex, Vector sunV, Vector LVLHX,
     Vector LVLHY, Vector LVLHZ, Vector antiLVLHX, Vector antiLVLHY,Vector
    antiLVLHZ, Variable Test,Variable PYpanel,Variable NYpanel,Variable
    NZpanel,Variable PZpanel,Variable PYDpanel, Variable NYDpanel, Variable
    Power, Array PowerGeneration, Variable OrbitAveragePower, Variable
    OrbitCount,WindowOverlay wo, Variable n,Variable Thruster_State,Variable
    Battery,Variable Total_I);
        Power = 0;
        DOTPXS = LVLHX.VertexAngle(Duplex.SunVector);
```

```
DOTNXS = antiLVLHX.VertexAngle(Duplex.SunVector);
DOTPYS = LVLHY.VertexAngle(Duplex.SunVector);
DOTNYS = antiLVLHY.VertexAngle(Duplex.SunVector);
DOTPZS = LVLHZ.VertexAngle(Duplex.SunVector);
DOTNZS = antiLVLHZ.VertexAngle(Duplex.SunVector);

// Sunlit Orbit
// Exposure
If (Thruster_State == 1);
If (Duplex.InShadow == 0);
// +Y Bus Face
        If (DOTPYS < EdgeLossAngle);
                PYpanel = abs(cos(rad(DOTPYS)))*Efficiency*meanPower*
                    CellArea*(8)*(Duplex.Range(Sun)/meanSunDistance);
                // Check +Y For Deployable Panel Solar Shielding
                If (sunV.Intersects(Duplex.ProximityZones[1]));
                        PYpanel = 0;
                End;
        End;

// -Y Bus Face
        If (DOTNYS < EdgeLossAngle);
                NYpanel = abs(cos(rad(DOTNYS)))*Efficiency*meanPower*
                    CellArea*(8)*(Duplex.Range(Sun)/meanSunDistance);
                // Check -Y For Deployable Panel Solar Shielding
                If (sunV.Intersects(Duplex.ProximityZones[2]));
                        NYpanel = 0;
```

```
            End;
        End;

// -Z Bus Face
        If (sunV.Intersects(Duplex.ProximityZones[3]));
            NZpanel = abs(cos(rad(DOTNZS)))*Efficiency*meanPower*
                CellArea*(2.5)*(Duplex.Range(Sun)/meanSunDistance)
                ;
        End;
// +Z Bus Face
        If (sunV.Intersects(Duplex.ProximityZones[4]));
            PZpanel = abs(cos(rad(DOTPZS)))*Efficiency*meanPower*
                CellArea*(19)*(Duplex.Range(Sun)/meanSunDistance)
                + EarthAlbedo*Efficiency*meanPower*CellArea*(38)*(
                Duplex.Range(Sun)/meanSunDistance);
        End;
// +Y Deployable Panel  Cross Over
        If (DOTPYS < (90 + ExtensionAngle) and DOTNZS < 90);
            PYDpanel = abs(cos(rad(DOTNZS)))*Efficiency*meanPower*
                CellArea*(8)*(Duplex.Range(Sun)/meanSunDistance);
        End;

        If (DOTPYS > (90) and sunV.Intersects(Duplex.ProximityZones
            [5]) == 0);
            PYDpanel = abs(sin(rad(DOTPYS+DOTNZS)))*Efficiency*
                meanPower*CellArea*(8)*(Duplex.Range(Sun)/
                meanSunDistance);
```

```
            End;

//      // -Y Deployable Panel

            If (DOTNYS < (90 + ExtensionAngle) and DOTNZS < 90);
                NYDpanel = abs(cos(rad(DOTNZS)))*Efficiency*meanPower*
                    CellArea*(8)*(Duplex.Range(Sun)/meanSunDistance);
            End;

            If (DOTNYS > (90) and sunV.Intersects(Duplex.ProximityZones
                [6]) == 0);
                NYDpanel = abs(sin(rad(DOTNYS+DOTNZS)))*Efficiency*
                    meanPower*CellArea*(8)*(Duplex.Range(Sun)/
                    meanSunDistance);
            End;


        // Non-Exposure
            If (DOTPYS > EdgeLossAngle);
                PYpanel = 0;
            End;
            If (DOTNYS > EdgeLossAngle);
                NYpanel = 0;
            End;

            If (sunV.Intersects(Duplex.ProximityZones[3]) == 0);
                NZpanel = 0;
```

```
        End;
        // Earth Reflection on Z+ Arrays
        If (sunV.Intersects(Duplex.ProximityZones[4]) == 0);
                PZpanel = ((100-Duplex.PercentShadow)/100)*EarthAlbedo
                    *Efficiency*meanPower*CellArea*(19)*(Duplex.Range(
                    Sun)/meanSunDistance);
        End;

        If (DOTPYS > 90 and sunV.Intersects(Duplex.ProximityZones[5])
            or DOTPZS < 90);
                PYDpanel = 0;
        End;

        If (DOTNYS > 90 and sunV.Intersects(Duplex.ProximityZones[6])
            or DOTPZS < 90);
                NYDpanel = 0;
        End;

End;

// Umbra
If (Duplex.InShadow == 1);
        PYpanel = 0;
        NYpanel = 0;
        NZpanel = 0;
        PYDpanel = 0;
        NYDpanel = 0;
```

```
        PZpanel = ((100-Duplex.PercentShadow)/100)*EarthAlbedo*
            Efficiency*meanPower*CellArea*(19)*(Duplex.Range(Sun)/
            meanSunDistance);
End;

Power = (PYpanel+NYpanel+PZpanel+NZpanel+PYDpanel+NYDpanel);


If (OrbitCount < n);
        PowerGeneration.PushBack(Power);
End;
If (OrbitCount == n);
        OrbitAveragePower = PowerGeneration.Mean();
        PowerGeneration.Clear();
        n = n + 1;
End;


// Display Solar Panel Power Readings
wo.Shapes[6].TextOptions.Text = StringConcat("FPPT Opperating",'','');
wo.Shapes[0].TextOptions.Text = StringConcat("FPPT: ",'On','');
wo.Shapes[1].TextOptions.Text = StringConcat("MVP: ",'Off','');
wo.Shapes[4].TextOptions.Text = StringConcat("I: ",Duplex.I.Format
    ("%0.3f"),'');
wo.Shapes[3].TextOptions.Text = StringConcat("SMA: ",Duplex.A.Format
    ("%0.2f"),'');
wo.Shapes[2].TextOptions.Text = StringConcat("FPPT I: ",Total_I.Format
```

```
        ("%0.04f"),'');
//      wo.Shapes[5].TextOptions.Text = StringConcat("MVP",'Off','');
        wo.Shapes[7].TextOptions.Text = StringConcat("Battery: ",Battery.
            Format("%0.2f"),'W');
        End;



        If (Thruster_State == 0); // FPPT not active


            If (Duplex.InShadow == 1);
                PZpanel = 0;
                PYpanel = 0;
                NYpanel = 0;
                NZpanel = 0;
                PYDpanel = 0;
                NYDpanel = 0;
            End;


            If (Duplex.InShadow == 0);
                PZpanel = abs(cos(rad(DOTPZS)))*Efficiency*meanPower*
                    CellArea*(19)*(Duplex.Range(Sun)/meanSunDistance);
                PYpanel = 0;
                NYpanel = 0;
                NZpanel = 0;
                PYDpanel = 0;
```

```
                NYDpanel = 0;
            End;


            wo.Shapes[6].TextOptions.Text = StringConcat("Charging",'','')
                ;
            wo.Shapes[0].TextOptions.Text = StringConcat("FPPT: ",'Off
                ','');
            wo.Shapes[1].TextOptions.Text = StringConcat("MVP: ",'Off','')
                ;
            wo.Shapes[4].TextOptions.Text = StringConcat("I: ",Duplex.I.
                Format("%0.3f"),'');
            wo.Shapes[3].TextOptions.Text = StringConcat("SMA: ",Duplex.A.
                Format("%0.2f"),'');
//          wo.Shapes[4].TextOptions.Text = StringConcat("FPPT",'On','');
//          wo.Shapes[5].TextOptions.Text = StringConcat("MVP",'Off','');
            wo.Shapes[7].TextOptions.Text = StringConcat("Battery: ",
                Battery.Format("%0.2f"),'W');
            Power = PZpanel;
        End;


            If (Thruster_State == 2); // MVP Active
            wo.Shapes[6].TextOptions.Text = StringConcat("MVP Opperating
                ","','');
            wo.Shapes[0].TextOptions.Text = StringConcat("FPPT: ",'Off
                ','');
            wo.Shapes[1].TextOptions.Text = StringConcat("MVP: ",'On','');
            wo.Shapes[4].TextOptions.Text = StringConcat("I: ",Duplex.I.
```

```
                    Format("%0.3f"),'');
                wo.Shapes[3].TextOptions.Text = StringConcat("SMA: ",Duplex.A.
                    Format("%0.2f"),'');
                wo.Shapes[2].TextOptions.Text = StringConcat("MVP I: ",Total_I
                    .Format("%0.04f"),'');
//              wo.Shapes[5].TextOptions.Text = StringConcat("MVP",'Off','');
                wo.Shapes[7].TextOptions.Text = StringConcat("Battery: ",
                    Battery.Format("%0.2f"),'W');
                End;
EndProcedure;



Global List<Vector> TestList[6];

Define Procedure dynamics(Spacecraft Duplex, Vector sunV, Vector velocity,
    Vector LVLHX, Vector LVLHY, Vector LVLHZ, Vector antiLVLHX, Vector
    antiLVLHY, Vector antiLVLHZ);

// Sun Position
sunV.BuildVector(9, Duplex, Sun);
sunV.DrawMethod = 1;
sunV.Color  = ColorTools.Gold;
sunV.MagnitudeScaleFactor = 1;

// Duplex Velocity Vector
velocity.BuildVector(7,Duplex);
```

```
velocity.DrawMethod = 1;
velocity.Color   = ColorTools.Aquamarine;
velocity.MagnitudeScaleFactor = 1;

// Duplex LVLHX
LVLHX.BuildVector(4,Duplex,1);
LVLHX.DrawMethod = 1;
LVLHX.Color   = ColorTools.Red;
LVLHX.MagnitudeScaleFactor = 1;
// Duplex anti LVLHY
antiLVLHX.BuildVector(10,LVLHX);
antiLVLHX.DrawMethod = 1;
antiLVLHX.Color   = ColorTools.Red;
antiLVLHX.MagnitudeScaleFactor = 1;

// Duplex LVLHY
LVLHY.BuildVector(4,Duplex,2);
LVLHY.DrawMethod = 1;
LVLHY.Color   = ColorTools.Green;
LVLHY.MagnitudeScaleFactor = 1;
// Duplex anti LVLHY
antiLVLHY.BuildVector(10,LVLHY);
antiLVLHY.DrawMethod = 1;
antiLVLHY.Color   = ColorTools.Green;
antiLVLHY.MagnitudeScaleFactor = 1;

// Duplex LVLHZ
```

```
LVLHZ.BuildVector(4,Duplex,3);
LVLHZ.DrawMethod = 1;
LVLHZ.Color  = ColorTools.Blue;
LVLHZ.MagnitudeScaleFactor = 1;

// Duplex anti LVLHZ
antiLVLHZ.BuildVector(10,LVLHZ);
antiLVLHZ.DrawMethod = 1;
antiLVLHZ.Color  = ColorTools.Blue;
antiLVLHZ.MagnitudeScaleFactor = 1;



EndProcedure;




Define Procedure InclinationSteering(Spacecraft Duplex,Variable Beta,
    FiniteBurn FiniteBurn1,Variable w,Variable f);
        CoordinateSystem LVLH_frame;
        Duplex.AttitudeRefFrame = "LVLH";
        Variable sgn;
        Beta = cos(w+f)*rad(90);
        If (deg(Beta) < 0);
                sgn = -1;
        End;
        If (deg(Beta) > 0);
```

```
            sgn = 1;
        End;
        //Beta = rad(90);
        FiniteBurn1.BurnDirection[1] = sgn;
        Duplex.Thrusters[0].Orientation[0] = cos(Beta);
        Duplex.Thrusters[0].Orientation[1] = sin(Beta);
        LVLH_frame.SetEulerAngles(0,0,deg(Beta),"1-2-3");
        Duplex.SetOrientation(LVLH_frame);
        Duplex.ProximityZones[0].Active = 1;
        Duplex.ProximityZones[1].Active = 1;
        Duplex.ProximityZones[2].Active = 1;
        Duplex.ProximityZones[3].Active = 1;
        Duplex.ProximityZones[4].Active = 1;
        Duplex.ProximityZones[5].Active = 1;
        Duplex.ProximityZones[6].Active = 1;
        Duplex.ProximityZones[7].Active = 1;
        Duplex.ProximityZones[0].Orientation[2] = deg(Beta);
        Duplex.ProximityZones[1].Orientation[0] = deg(Beta);
        Duplex.ProximityZones[2].Orientation[0] = deg(Beta);
        Duplex.ProximityZones[3].Orientation[0] = deg(Beta);
        Duplex.ProximityZones[4].Orientation[0] = deg(Beta);
        Duplex.ProximityZones[5].Orientation[0] = deg(Beta);
        Duplex.ProximityZones[6].Orientation[0] = deg(Beta);
        Duplex.ProximityZones[7].Orientation[0] = deg(Beta);
EndProcedure;

Define Procedure sunpointing(Spacecraft Duplex, Vector LVLHZ,Variable PZpanel,
```

```
WindowOverlay wo, Variable Power, Vector sunV);
    Duplex.ProximityZones[0].Active = 0;
    Duplex.ProximityZones[1].Active = 0;
    Duplex.ProximityZones[2].Active = 0;
    Duplex.ProximityZones[3].Active = 0;
    Duplex.ProximityZones[4].Active = 0;
    Duplex.ProximityZones[5].Active = 0;
    Duplex.ProximityZones[6].Active = 0;
    Duplex.ProximityZones[7].Active = 0;


    Duplex.AttitudeRefFrame = "MJ2000";
    Vector SpacecraftSun;
    Vector SpacecraftEarth;
    CoordinateSystem TailtoSun;


    SpacecraftEarth.BuildVector(9,Duplex,Earth);



    TailtoSun.BuildCoordinateSystem(3,sunV,2,SpacecraftEarth);
    TailtoSun.Epoch = Duplex.Epoch;


    Duplex.SetOrientation(TailtoSun);
```

```
EndProcedure;


Define Procedure SMASteering(Spacecraft Duplex,Variable Alpha,FiniteBurn
    FiniteBurn2,FiniteBurn FiniteBurn1,Variable Thruster_State, Variable
    direction);
        CoordinateSystem LVLH_frame;
        Duplex.AttitudeRefFrame = "LVLH";
        Variable sgn;
        Variable x;
        x = ((Duplex.E*sin(rad(Duplex.TA))) / (1 + Duplex.E*cos(rad(Duplex.TA)
            )));
        Alpha = atan(x);
        If (Thruster_State == 2);// MVP
                sgn = 1;
        End;
        If (direction == -1);// FPPT
                sgn = 0;
        End;
        If (Thruster_State == 1);// FPPT
        FiniteBurn1.BurnDirection[0] = direction;
        FiniteBurn1.BurnDirection[1] = 0;
        Duplex.Thrusters[0].Orientation[2] = sin(Alpha);
        Duplex.Thrusters[0].Orientation[0] = cos(Alpha);
        LVLH_frame.SetEulerAngles(0,deg(Alpha),180,"1-2-3");
        End;
        If (Thruster_State == 2);// MVP
        FiniteBurn2.BurnDirection[0] = direction;
```

```
        Duplex.Thrusters[1].Orientation[2] = sin(Alpha);
        Duplex.Thrusters[1].Orientation[0] = cos(Alpha);
        LVLH_frame.SetEulerAngles(0,deg(Alpha),0,"1-2-3");
        End;



        Duplex.SetOrientation(LVLH_frame);
        Duplex.ProximityZones[0].Active = 1;
        Duplex.ProximityZones[1].Active = 1;
        Duplex.ProximityZones[2].Active = 1;
        Duplex.ProximityZones[3].Active = 1;
        Duplex.ProximityZones[4].Active = 1;
        Duplex.ProximityZones[5].Active = 1;
        Duplex.ProximityZones[6].Active = 1;
        Duplex.ProximityZones[7].Active = 1;
        Duplex.ProximityZones[0].Orientation[1] = deg(Alpha);
        Duplex.ProximityZones[1].Orientation[2] = deg(Alpha);
        Duplex.ProximityZones[2].Orientation[2] = deg(Alpha);
        Duplex.ProximityZones[3].Orientation[2] = deg(Alpha);
        Duplex.ProximityZones[4].Orientation[2] = deg(Alpha);
        Duplex.ProximityZones[5].Orientation[2] = deg(Alpha);
        Duplex.ProximityZones[6].Orientation[2] = deg(Alpha);
        Duplex.ProximityZones[7].Orientation[2] = deg(Alpha);
EndProcedure;



Global Variable time;
```

```
MVP_I = 0;
FPPT_I = 0;
Charge_Time = 0;
prevSMA = Duplex.A;
MVP_runtime = 0;
FPPT_runtime = 0;
DeltaSMA = 0.00;
Duplex.Thrusters[0].ThrusterOn = 0; // On
Duplex.Thrusters[1].ThrusterOn = 0; // Off
TimeStep = Duplex.Propagator.StepSize;
//WhileManeuvering Duplex using FiniteBurn1;
//        If (Duplex.ElapsedTime.ToDays > 1);
//                Break;
//        End;
//        w = rad(Duplex.W);
//        f = rad(Duplex.TA);
//        Thruster_State = 1;
//        Duplex.Sensors[0].Active = 0;
//        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
//        Call InclinationSteering(Duplex,Beta,FiniteBurn1,w,f);
//        Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,Power,
    PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,Thruster_State,Battery,
    FPPT_I);
//        Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
//        Plot Duplex.ElapsedTime.ToHours, Duplex.I;
```

```
//        Update DataTableWindow1;
//        FPPT_I = FPPT_I + TimeStep.ToSeconds * Duplex.Thrust(Duplex.Thrusters
    [0]);
//        DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
//        prevSMA = Duplex.A;
//        FPPT_runtime = FPPT_runtime + 1;
//        Update ViewWindow1;
//
//End;

PlotWindow Drag1({Duplex.ElapsedTime, Duplex.Drag.Norm()});
PlotWindow Fuel1({Duplex.ElapsedTime, (Duplex.Tanks[0] AsType ElectricalTank).
    TankMass});
PlotWindow Fuel2({Duplex.ElapsedTime, (Duplex.Tanks[1] AsType ElectricalTank).
    TankMass});
PlotWindow Baffle({Duplex.ElapsedTime, antiLVLHX.VertexAngle(Duplex.SunVector)
    });

(PlotWindow1.Series[0] AsType PlotScatterSeries).EnableDataCulling = 0;
(PlotWindow2.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;
(PlotWindow3.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;

(PlotWindow4.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;

//(Drag1.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;
(Fuel1.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;
(Fuel2.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;
```

```
(Baffle.Series[0] AsType PlotScatterSeries).EnableDataCulling  = 0;



Update Drag1;

Update Fuel1;

Update Fuel2;

Update Baffle;


Battery = 65;

Update DataTableWindow1;

DataTableWindow DataTableWindow4({Duplex.ElapsedTime, Battery, Power,
    Thruster_State, Duplex.Thrusters[0].ThrusterOn, Duplex.Thrusters[1].
    ThrusterOn, Duplex.OrbitNumberCumulative, Duplex.Quaternion,Duplex.
    Thrusters[0].Orientation, Duplex.Thrusters[1].Orientation,Duplex.A, Duplex
    .I});

 //Maneuver Spacecraft: 1 degree inclination change with FPPT

While (Duplex.ElapsedTime < TIMESPAN(1000 days));

        Update DataTableWindow4;

        Thruster_State = 0;

        Duplex.Sensors[0].Active = 0;

        Duplex.Thrusters[2].ThrusterOn = 0;

        OrbitCount = Duplex.OrbitNumberCumulative;

        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
            antiLVLHY,antiLVLHZ);

        Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);

        Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
            ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,
```

```
    Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,FPPT_I);
Call battery(Duplex,Power,Battery,Thruster_State,TimeStep);
Charge_Time = Charge_Time + 1;
w = rad(Duplex.W);
f = rad(Duplex.TA);
If (Battery > 64);
        //If (w+f - Duplex.RAAN < 10);
        Duplex.Thrusters[2].ThrusterOn = 1; // On
        Duplex.Thrusters[1].ThrusterOn = 0; // Off
        Duplex.Thrusters[0].ThrusterOn = 0; // Off
                WhileManeuvering Duplex using FiniteBurn1;
                        If (Battery < 32);
                                Update DataTableWindow4;
                                Break;
                        End;
                        If (Duplex.I < 51.1);
                                Update DataTableWindow4;
                                Break;
                        End;
                        Thruster_State = 5;
                        Duplex.Sensors[0].Active = 1;
                        w = rad(Duplex.W);
                        f = rad(Duplex.TA);
                        Beta = cos(w+f)*rad(90);
                        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY
                            ,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
```

```
Call InclinationSteering(Duplex,Beta,
    FiniteBurn1,w,f);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,
    LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,
    PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,
    NYDpanel,Power,PowerGenerated,
    OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,FPPT_I);
Call battery(Duplex,Power,Battery,
    Thruster_State,TimeStep);
Update Baffle;
Update PlotWindow4;
Update PlotWindow1;
Update Drag1;
Update Fuel1;
Update Fuel2;
Update Baffle;
Update PlotWindow2;
Update PlotWindow3;
Update ViewWindow1;
Update PlotWindow6;
Update DataTableWindow3;
Update DataTableWindow4;
FPPT_I = FPPT_I + TimeStep.ToSeconds * Duplex.
    Thrust(Duplex.Thrusters[2]);
DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
prevSMA = Duplex.A;
```

```
                                FPPT_runtime = FPPT_runtime + 1;
                                End;
                        //End;
                End;

        If (Duplex.I < 51.1);
                Update DataTableWindow1;
                Update DataTableWindow4;
                Break;
        End;
        Update PlotWindow4;
        Update PlotWindow1;
        Update Drag1;
        Update Fuel1;
        Update Fuel2;
        Update Baffle;
        // Update Drag3;
        Update PlotWindow2;
        Update PlotWindow3;
        Update ViewWindow1;
        Update PlotWindow6;
        Update DataTableWindow3;
        Update DataTableWindow4;
        Step Duplex;

End;
```

```
////10km MVP Raise
//Update DataTableWindow1;
//prevSMA = Duplex.A;
//DeltaSMA = 0;
//DeltaSMADes = 10;
//While (Duplex.ElapsedTime < TIMESPAN(1000 days));
//
//        Thruster_State = 0;
//        Duplex.Sensors[1].Active = 0;
//        OrbitCount = Duplex.OrbitNumberCumulative;
//        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
//        Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
//        Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,Power,
    PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,Thruster_State,Battery,
    MVP_I);
//        Call battery(Duplex,Power,Battery,Thruster_State,TimeStep);
//        Charge_Time = Charge_Time + 1;
//        If (Battery > 64);
//                If (DeltaSMA < DeltaSMADes);
//                        Duplex.Thrusters[1].ThrusterOn = 1; // On
//                        Duplex.Thrusters[0].ThrusterOn = 0; // Off
//                                WhileManeuvering Duplex using FiniteBurn2;
//                                        If (Battery < 32);
//                                                Break;
```

```
//                                          End;
//                                          If (DeltaSMA > DeltaSMADes);
//                                                  Thruster_State = 4;
//                                                  Update DataTableWindow1;
//                                                  Update PlotWindow4;
//                                                  Break;
//                                          End;
//                                          Thruster_State = 2;
//                                          Duplex.Sensors[1].Active = 1;
//                                          Call dynamics(Duplex,sunV,velocity,
    LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
//                                          Call SMASteering(Duplex,Alpha,
    FiniteBurn2,FiniteBurn1,Thruster_State,direction);
//                                          Call PowerGeneration(Duplex,sunV,LVLHX
    ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,
    PZpanel,PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
    OrbitCount,wo,n,Thruster_State,Battery,MVP_I);
//                                          Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
//                                          MVP_I = MVP_I + TimeStep.ToSeconds *
    Duplex.Thrust(Duplex.Thrusters[1]);
//                                          //DeltaSMA = DeltaSMA + Duplex.A -
    prevSMA;
//                                          DeltaSMA = DeltaSMA + Duplex.A -
    prevSMA;
//                                          prevSMA = Duplex.A;
//                                          MVP_runtime = MVP_runtime + 1;
```

```
//                                          Update PlotWindow4;
//                                          Update PlotWindow1;
//                                          Update PlotWindow2;
//                                          Update PlotWindow3;
//
//                                  End;
//
//                          End;
//        End;
//        prevSMA = Duplex.A;
//        Step Duplex;
//        If (Thruster_State == 4);
//                Break;
//        End;
//
//Update PlotWindow4;
//Update PlotWindow1;
//Update PlotWindow2;
//Update PlotWindow3;
//
//End;
//
////20km FPPT Raise
//prevSMA = Duplex.A;
//DeltaSMA = 0;
//DeltaSMADes = 20;
//While (Duplex.ElapsedTime < TIMESPAN(1000 days));
```

```
//      If (Battery > 64);
//              If (DeltaSMA < DeltaSMADes);
//              Duplex.Thrusters[0].ThrusterOn = 1; // On
//              Duplex.Thrusters[1].ThrusterOn = 0; // Off
//                      WhileManeuvering Duplex using FiniteBurn1;
//                                      If (Battery < 32);
//                                              Break;
//                                      End;
//                                      If (DeltaSMA > DeltaSMADes);
//                                              Thruster_State = 4;
//                                              Update DataTableWindow1;
//                                              Break;
//                                      End;
//                                      Thruster_State = 1;
//                                      Duplex.Sensors[0].Active = 1;
//                                      direction = 1; // orbit decrease SMA
//                                      Call dynamics(Duplex,sunV,velocity,
   LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
//                                      Call SMASteering(Duplex,Alpha,
   FiniteBurn2,FiniteBurn1,Thruster_State,direction);
//                                      Call PowerGeneration(Duplex,sunV,LVLHX
   ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,
   PZpanel,PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
   OrbitCount,wo,n,Thruster_State,Battery,MVP_I);
//                                      Call battery(Duplex,Power,Battery,
   Thruster_State, TimeStep);
//                                      FPPT_I = FPPT_I + TimeStep.ToSeconds *
```

```
    Duplex.Thrust(Duplex.Thrusters[0]);
//                                          DeltaSMA = DeltaSMA + Duplex.A -
   prevSMA;
//                                          prevSMA = Duplex.A;
//                                          FPPT_runtime = FPPT_runtime + 1;
//                                          Update PlotWindow4;
//                                          Update PlotWindow1;
//                                          Update PlotWindow2;
//                                          Update PlotWindow3;
//
//                          End;
//
//                  End;
//        End;
//        prevSMA = Duplex.A;
//        Step Duplex;
//        If (Thruster_State == 4);
//                  Break;
//        End;
//End;
//
//
//
//


Update DataTableWindow1;
```

```
prevSMA = Duplex.A;
DeltaSMA = 0;
While (Duplex.ElapsedTime < TIMESPAN(500 days));
        Thruster_State = 0;
        Duplex.Sensors[1].Active = 0;
        Duplex.Thrusters[1].ThrusterOn = 0; // On
        Duplex.Thrusters[0].ThrusterOn = 0; // Off
        OrbitCount = Duplex.OrbitNumberCumulative;
        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
            antiLVLHY,antiLVLHZ);
        Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
        Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
            ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,
            Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,
            Thruster_State,Battery,MVP_I);
        Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
        Charge_Time = Charge_Time + 1;
        Update DataTableWindow4;
        If (Battery > 64);
                If (DeltaSMA < 10);
                        Duplex.Thrusters[1].ThrusterOn = 1; // On
                        Duplex.Thrusters[0].ThrusterOn = 0; // Off
                                WhileManeuvering Duplex using FiniteBurn2;
                                        If (Battery < 32);
                                                Update DataTableWindow4;
                                                Break;
                                        End;
```

```
If (DeltaSMA > 10);
        Thruster_State = 1;
        Update DataTableWindow1;
        Update DataTableWindow4;
        Break;
End;
Thruster_State = 2;
Duplex.Sensors[1].Active = 1;
direction = 1;
Call dynamics(Duplex,sunV,velocity,
    LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
Call SMASteering(Duplex,Alpha,
    FiniteBurn2,FiniteBurn1,
    Thruster_State,direction);
Call PowerGeneration(Duplex,sunV,LVLHX
    ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,
    antiLVLHZ,Test,PYpanel,NYpanel,
    NZpanel,PZpanel,PYDpanel,NYDpanel,
    Power,PowerGenerated,
    OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,MVP_I);
Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
MVP_I = MVP_I + TimeStep.ToSeconds *
    Duplex.Thrust(Duplex.Thrusters[1])
    ;
```

```
                                DeltaSMA = DeltaSMA + abs(Duplex.A -
                                    prevSMA);
                                prevSMA = Duplex.A;
                                MVP_runtime = MVP_runtime + 1;
                                Update PlotWindow4;
                                Update PlotWindow4;
                                Update DataTableWindow4;

                            End;

                    End;
End;

If (Battery > 64);
        Update DataTableWindow4;
        If (DeltaSMA >= 10);
        Duplex.Thrusters[0].ThrusterOn = 1; // On
        Duplex.Thrusters[1].ThrusterOn = 0; // Off
                WhileManeuvering Duplex using FiniteBurn1;
                                If (Battery < 32);
                                        Update DataTableWindow4;
                                        Break;
                                End;
                                If (DeltaSMA > 30);
                                        Thruster_State = 4;
                                        Update DataTableWindow1;
                                        Update DataTableWindow4;
```

```
            Break;
        End;
        Thruster_State = 1;
        Duplex.Sensors[0].Active = 1;
        direction = 1; // orbit decrease SMA
        Call dynamics(Duplex,sunV,velocity,
            LVLHX,LVLHY,LVLHZ,antiLVLHX,
            antiLVLHY,antiLVLHZ);
        Call SMASteering(Duplex,Alpha,
            FiniteBurn2,FiniteBurn1,
            Thruster_State,direction);
        Call PowerGeneration(Duplex,sunV,LVLHX
            ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,
            antiLVLHZ,Test,PYpanel,NYpanel,
            NZpanel,PZpanel,PYDpanel,NYDpanel,
            Power,PowerGenerated,
            OrbitAveragePower,OrbitCount,wo,n,
            Thruster_State,Battery,MVP_I);
        Call battery(Duplex,Power,Battery,
            Thruster_State, TimeStep);
        FPPT_I = FPPT_I + TimeStep.ToSeconds *
             Duplex.Thrust(Duplex.Thrusters
            [0]);
        DeltaSMA = DeltaSMA + abs(Duplex.A -
            prevSMA);
        prevSMA = Duplex.A;
        FPPT_runtime = FPPT_runtime + 1;
```

```
                                        Update PlotWindow4;
                                        Update Baffle;
                                        Update PlotWindow1;
                                        Update Drag1;
                                        Update Fuel1;
                                        Update Fuel2;
                                        Update Baffle;
                                        Update PlotWindow2;
                                        Update PlotWindow3;
                                        Update ViewWindow1;
                                        Update DataTableWindow4;

                                End;

                        End;
                End;

                prevSMA = Duplex.A;
                Step Duplex;
                If (Thruster_State == 4);
                        Update DataTableWindow4;
                        Break;
                End;
End;
```

```
//// 10 km Orbit Lower with MVP
//prevSMA = Duplex.A;
//DeltaSMA = 0.00;
//DeltaSMADes = -10;
//While (Duplex.ElapsedTime < TIMESPAN(1000 days));
//
//        Thruster_State = 0;
//        Duplex.Sensors[1].Active = 0;
//        OrbitCount = Duplex.OrbitNumberCumulative;
//        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
//        Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
//        Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,Power,
    PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,Thruster_State,Battery,
    MVP_I);
//        Call battery(Duplex,Power,Battery,Thruster_State,TimeStep);
//        If (Battery > 64);
//                If (DeltaSMA > DeltaSMADes);
//                        Duplex.Thrusters[1].ThrusterOn = 1; // On
//                        Duplex.Thrusters[0].ThrusterOn = 0; // Off
//                                WhileManeuvering Duplex using FiniteBurn1;
//                                        If (Battery < 32);
```

```
//                                                  Break;
//                                          End;
//                                          If (DeltaSMA <= DeltaSMADes);
//                                                  Thruster_State = 4;
//                                                  Update DataTableWindow1;
//                                                  Break;
//                                          End;
//                                          Thruster_State = 1;
//                                          Duplex.Sensors[1].Active = 1;
//                                          direction = -1; // orbit decrease SMA
//                                          Call dynamics(Duplex,sunV,velocity,
    LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
//                                          Call SMASteering(Duplex,Alpha,
    FiniteBurn2,FiniteBurn1,Thruster_State,direction);
//                                          Call PowerGeneration(Duplex,sunV,LVLHX
    ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,
    PZpanel,PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
    OrbitCount,wo,n,Thruster_State,Battery,MVP_I);
//                                          Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
//                                          MVP_I = MVP_I + TimeStep.ToSeconds *
    Duplex.Thrust(Duplex.Thrusters[1]);
//                                          DeltaSMA = DeltaSMA + Duplex.A -
    prevSMA;
//                                          prevSMA = Duplex.A;
//                                          MVP_runtime = MVP_runtime + 1;
//                                          Update PlotWindow4;
```

```
//                                          Update PlotWindow1;
//                                          Update PlotWindow2;
//                                          Update PlotWindow3;
//                                          Update Drag1;
//                                          Update Fuel1;
//                                          Update Fuel2;
//                                          Update Baffle;
//                                          // Update Drag3;
//                                  End;
//                          Update PlotWindow4;
//                          Update Drag1;
//                          Update Fuel1;
//                          Update Fuel2;
//                          Update Baffle;
//                          // Update Drag3;
//                          End;
//      End;
//
//      Charge_Time = Charge_Time + 1;
//      prevSMA = Duplex.A;
//      Step Duplex;
//
//      If (Thruster_State == 4);
//              Break;
//      End;
//
//End;
```

```
//
//// 140km Orbit Lower with FPPT
//prevSMA = Duplex.A;
//DeltaSMA = 0.00;
//DeltaSMADes = -140;
//While (Duplex.ElapsedTime < TIMESPAN(500 days));
//
//       Thruster_State = 0;
//       Duplex.Sensors[0].Active = 0;
//       OrbitCount = Duplex.OrbitNumberCumulative;
//       Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
//       Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
//       Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,Power,
    PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,Thruster_State,Battery,
    MVP_I);
//       Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
//       If (Battery > 64);
//              If (DeltaSMA > DeltaSMADes);
//                     Duplex.Thrusters[0].ThrusterOn = 1; // On
//                     Duplex.Thrusters[1].ThrusterOn = 0; // Off
//                            WhileManeuvering Duplex using FiniteBurn1;
//                                   If (Battery < 32);
//                                          Break;
//                                   End;
//                                   If (DeltaSMA <= DeltaSMADes);
```

```
//                                          Thruster_State = 4;
//                                          Update DataTableWindow1;
//                                          Break;
//                                      End;
//                                      Thruster_State = 1;
//                                      Duplex.Sensors[0].Active = 1;
//                                      direction = -1; // orbit decrease SMA
//                                      Call dynamics(Duplex,sunV,velocity,
    LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
//                                      Call SMASteering(Duplex,Alpha,
    FiniteBurn2,FiniteBurn1,Thruster_State,direction);
//                                      Call PowerGeneration(Duplex,sunV,LVLHX
    ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,
    PZpanel,PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
    OrbitCount,wo,n,Thruster_State,Battery,MVP_I);
//                                      Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
//                                      FPPT_I = FPPT_I + TimeStep.ToSeconds *
     Duplex.Thrust(Duplex.Thrusters[0]);
//                                      DeltaSMA = DeltaSMA + Duplex.A -
    prevSMA;
//                                      prevSMA = Duplex.A;
//                                      FPPT_runtime = FPPT_runtime + 1;
//                                      Update PlotWindow4;
//                                      Update PlotWindow1;
//                                      Update PlotWindow2;
//                                      Update PlotWindow3;
```

```
//                                          Update Drag1;
//                                          Update Fuel1;
//                                          Update Fuel2;
//                                          Update Baffle;
//                                          // Update Drag3;
//                                  End;
//                          Update PlotWindow4;
//                          End;
//       End;
//
//       Charge_Time = Charge_Time + 1;
//       prevSMA = Duplex.A;
//       Step Duplex;
//
//       If (Thruster_State == 4);
//               Break;
//       End;
//
//End;
Update DataTableWindow1;
Update DataTableWindow4;
prevSMA = Duplex.A;
DeltaSMA = 0;
While (Duplex.ElapsedTime < TIMESPAN(500 days));
        Thruster_State = 0;
        Duplex.Sensors[1].Active = 0;
        Duplex.Thrusters[1].ThrusterOn = 0; // Off
```

```
Duplex.Thrusters[0].ThrusterOn = 0; // Off
OrbitCount = Duplex.OrbitNumberCumulative;
Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,
    Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,MVP_I);
Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
Charge_Time = Charge_Time + 1;
Update DataTableWindow4;
If (Battery > 64);
        If (DeltaSMA < 10);
                    WhileManeuvering Duplex using FiniteBurn2;
                        Duplex.Thrusters[1].ThrusterOn = 1; //
                            On
                        Duplex.Thrusters[0].ThrusterOn = 0; //
                            Off
                        If (Battery < 32);
                            Update DataTableWindow4;
                            Duplex.Thrusters[1].ThrusterOn
                                = 0; // Off
                            Break;
                        End;
                        If (DeltaSMA > 10);
                            Thruster_State = 1;
```

```
            Update DataTableWindow1;
            Update DataTableWindow4;
            Break;
        End;
        Thruster_State = 2;
        Duplex.Sensors[1].Active = 1;
        direction = -1;
        Call dynamics(Duplex,sunV,velocity,
            LVLHX,LVLHY,LVLHZ,antiLVLHX,
            antiLVLHY,antiLVLHZ);
        Call SMASteering(Duplex,Alpha,
            FiniteBurn2,FiniteBurn1,
            Thruster_State,direction);
        Call PowerGeneration(Duplex,sunV,LVLHX
            ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,
            antiLVLHZ,Test,PYpanel,NYpanel,
            NZpanel,PZpanel,PYDpanel,NYDpanel,
            Power,PowerGenerated,
            OrbitAveragePower,OrbitCount,wo,n,
            Thruster_State,Battery,MVP_I);
        Call battery(Duplex,Power,Battery,
            Thruster_State, TimeStep);
        MVP_I = MVP_I + TimeStep.ToSeconds *
            Duplex.Thrust(Duplex.Thrusters[1])
            ;
        DeltaSMA = DeltaSMA + abs(Duplex.A -
            prevSMA);
```

```
                                    prevSMA = Duplex.A;
                                    MVP_runtime = MVP_runtime + 1;
                                    Update PlotWindow4;
                                    Update PlotWindow4;
                                    Update DataTableWindow4;

                            End;

                    End;
End;

If (Battery > 64);
        If (DeltaSMA >= 10);
                WhileManeuvering Duplex using FiniteBurn1;
                                    Duplex.Thrusters[0].ThrusterOn = 1; //
                                        On
                                    Duplex.Thrusters[1].ThrusterOn = 0; //
                                        Off
                                    If (Battery < 32);
                                            Update DataTableWindow4;
                                            Duplex.Thrusters[0].ThrusterOn
                                                = 0; // Off
                                            Break;
                                    End;
                                    If (DeltaSMA > 170);
                                            Thruster_State = 4;
                                            Update DataTableWindow1;
```

```
            Break;
        End;
        Thruster_State = 1;
        Duplex.Sensors[0].Active = 1;
        direction = -1; // orbit decrease SMA
        Call dynamics(Duplex,sunV,velocity,
            LVLHX,LVLHY,LVLHZ,antiLVLHX,
            antiLVLHY,antiLVLHZ);
        Call SMASteering(Duplex,Alpha,
            FiniteBurn2,FiniteBurn1,
            Thruster_State,direction);
        Call PowerGeneration(Duplex,sunV,LVLHX
            ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,
            antiLVLHZ,Test,PYpanel,NYpanel,
            NZpanel,PZpanel,PYDpanel,NYDpanel,
            Power,PowerGenerated,
            OrbitAveragePower,OrbitCount,wo,n,
            Thruster_State,Battery,MVP_I);
        Call battery(Duplex,Power,Battery,
            Thruster_State, TimeStep);
        FPPT_I = FPPT_I + TimeStep.ToSeconds *
             Duplex.Thrust(Duplex.Thrusters
            [0]);
        DeltaSMA = DeltaSMA + abs(Duplex.A -
            prevSMA);
        prevSMA = Duplex.A;
        FPPT_runtime = FPPT_runtime + 1;
```

```
                                        Update PlotWindow4;
                                        Update Baffle;
                                        Update PlotWindow1;
                                        Update Drag1;
                                        Update Fuel1;
                                        Update Fuel2;
                                        Update Baffle;
                                        Update PlotWindow2;
                                        Update PlotWindow3;
                                        Update ViewWindow1;
                                        Update DataTableWindow4;
                                End;

                        End;
                End;

                prevSMA = Duplex.A;
                Step Duplex;
                If (Thruster_State == 4);
                        Update DataTableWindow4;
                        Break;
                End;
End;


// 1 week orbit maintence MVP
Thruster_State = 2;
```

```
CurrSMA = Duplex.A;
While (Duplex.ElapsedTime < TIMESPAN(7 days));

        If (Thruster_State == 0);
                Duplex.Sensors[1].Active = 0;
                OrbitCount = Duplex.OrbitNumberCumulative;
                Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX
                    ,antiLVLHY,antiLVLHZ);
                Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
                Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,
                    antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,
                    PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
                    OrbitCount,wo,n,Thruster_State,Battery,FPPT_I);
                Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
                Charge_Time = Charge_Time + 1;
                Update DataTableWindow4;

        End;
        If (Battery > 64);
                Thruster_State = 1;
                Update PlotWindow4;
                Update Drag1;
                Update Fuel1;
                Update Fuel2;
                Update Baffle;
                Update DataTableWindow4;
                // Update Drag3;
```

```
End;



If (Battery > 32);
        If (Duplex.A < CurrSMA - .05);
                DeltaSMA = 0;
                prevSMA = Duplex.A;
                WhileManeuvering Duplex using FiniteBurn1;
                        Duplex.Thrusters[1].ThrusterOn = 1; // On
                        Duplex.Thrusters[0].ThrusterOn = 0; // Off
                        If (Battery < 32);
                                Thruster_State = 0;
                                Duplex.Thrusters[1].ThrusterOn = 0; //
                                    On
                                Update DataTableWindow4;
                                Break;
                        End;
                        If (Duplex.A > CurrSMA);
                                DeltaSMA = 0;
                                prevSMA = Duplex.A;
                                Update DataTableWindow4;
                                Break;
                        End;
                        Thruster_State = 1;
                        Duplex.Sensors[1].Active = 1;
                        direction = 1; // Orbit Increase
                        w = rad(Duplex.W);
```

```
f = rad(Duplex.TA);
Beta = cos(w+f)*rad(90);
Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY
    ,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
Call SMASteering(Duplex,Alpha,FiniteBurn2,
    FiniteBurn1,Thruster_State,direction);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,
    LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,
    PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,
    NYDpanel,Power,PowerGenerated,
    OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,FPPT_I);
Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
MVP_I = MVP_I + TimeStep.ToSeconds * Duplex.
    Thrust(Duplex.Thrusters[1]);
DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
prevSMA = Duplex.A;
MVP_runtime = MVP_runtime + 1;
Update PlotWindow4;
Update PlotWindow1;
Update PlotWindow2;
Update PlotWindow3;
Update Drag1;
Update Fuel1;
Update Fuel2;
Update Baffle;
```

```
                                Update DataTableWindow4;
                                // Update Drag3;

                                    End;
                            Update PlotWindow4;
                            Update Drag1;
                            Update Fuel1;
                            Update Fuel2;
                            Update DataTableWindow4;
                            // Update Drag3;
                    End;
                            If (Battery < 64);
                            Update DataTableWindow4;
                            Thruster_State = 0;
                            End;
End;


                    DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
                    prevSMA = Duplex.A;

                    Step Duplex;


End;

Update DataTableWindow1;
```

```
Update DataTableWindow1;
Update DataTableWindow4;

// 1 month orbit maintence FPPT
CurrSMA = Duplex.A;
Thruster_State = 1;
While (Duplex.ElapsedTime < TIMESPAN(30 days));

        If (Thruster_State == 0);
                Duplex.Sensors[0].Active = 0;
                OrbitCount = Duplex.OrbitNumberCumulative;
                Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX
                    ,antiLVLHY,antiLVLHZ);
                Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
                Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,
                    antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,
                    PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
                    OrbitCount,wo,n,Thruster_State,Battery,FPPT_I);
                Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
                Charge_Time = Charge_Time + 1;
                Update DataTableWindow4;

        End;
        If (Battery > 64);
                Thruster_State = 1;
                Update PlotWindow4;
                Update DataTableWindow4;
```

```
End;



If (Battery > 32);
        If (Duplex.A < CurrSMA - .05);
                DeltaSMA = 0;
                prevSMA = Duplex.A;
                WhileManeuvering Duplex using FiniteBurn1;
                        Duplex.Thrusters[0].ThrusterOn = 1; // On
                        Duplex.Thrusters[1].ThrusterOn = 0; // Off
                        If (Battery < 32);
                                Thruster_State = 0;
                                Duplex.Thrusters[0].ThrusterOn = 0; //
                                    Off
                                Update DataTableWindow4;
                                Break;
                        End;
                        If (Duplex.A > CurrSMA);
                                DeltaSMA = 0;
                                prevSMA = Duplex.A;
                                Update DataTableWindow4;
                                Break;
                        End;
                        Thruster_State = 1;
                        Duplex.Sensors[0].Active = 1;
                        direction = 1; // Orbit Increase
                        w = rad(Duplex.W);
```

```
f = rad(Duplex.TA);
Beta = cos(w+f)*rad(90);
Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY
    ,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
Call SMASteering(Duplex,Alpha,FiniteBurn2,
    FiniteBurn1,Thruster_State,direction);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,
    LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,
    PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,
    NYDpanel,Power,PowerGenerated,
    OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,FPPT_I);
Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
FPPT_I = FPPT_I + TimeStep.ToSeconds * Duplex.
    Thrust(Duplex.Thrusters[0]);
DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
prevSMA = Duplex.A;
FPPT_runtime = FPPT_runtime + 1;
Update PlotWindow4;
Update PlotWindow1;
Update PlotWindow2;
Update PlotWindow3;
Update Drag1;
Update Fuel1;
Update Fuel2;
Update Baffle;
```

```
                                    Update DataTableWindow4;
                                    // Update Drag3;
                                End;
                            Update PlotWindow4;
                    End;
                            If (Battery < 64);
                            Thruster_State = 0;
                            End;
End;


                    DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
                    prevSMA = Duplex.A;

                    Step Duplex;



End;
Update DataTableWindow1;
Update DataTableWindow4;
//// Deorbit with MVP
//prevSMA = Duplex.A;
//DeltaSMA = 0.00;
//While (Duplex.ElapsedTime < TIMESPAN(10 days));
//
//      Thruster_State = 0;
//      Duplex.Sensors[0].Active = 0;
```

```
//        OrbitCount = Duplex.OrbitNumberCumulative;
//        Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
//        Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
//        Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,Power,
    PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,Thruster_State,Battery,
    MVP_I);
//        Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
//
//        If ((Duplex.Tanks[1] AsType ElectricalTank).TankMass < .02 );
//                Break;
//        End;
//        If (Battery > 64);
//                //If (Duplex.A > 6578.137);
//                If (Duplex.A > 6600);
//                        Duplex.Thrusters[0].ThrusterOn = 0; // On
//                        Duplex.Thrusters[1].ThrusterOn = 1; // Off
//                                WhileManeuvering Duplex using FiniteBurn1;
//                                        If (Battery < 32);
//                                                Break;
//                                        End;
//                                        //If (Duplex.A < 6578.137);
//                                        If (Duplex.A < 6600);
//                                                Thruster_State = 4;
//                                                Update DataTableWindow1;
//                                                Break;
```

```
//                                          End;
//                                          Thruster_State = 1;
//                                          Duplex.Sensors[0].Active = 1;
//                                          direction = -1; // orbit decrease SMA
//                                          Call dynamics(Duplex,sunV,velocity,
    LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
//                                          Call SMASteering(Duplex,Alpha,
    FiniteBurn2,FiniteBurn1,Thruster_State,direction);
//                                          Call PowerGeneration(Duplex,sunV,LVLHX
    ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,
    PZpanel,PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
    OrbitCount,wo,n,Thruster_State,Battery,MVP_I);
//                                          Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
//                                          FPPT_I = FPPT_I + TimeStep.ToSeconds *
     Duplex.Thrust(Duplex.Thrusters[0]);
//                                          DeltaSMA = DeltaSMA + Duplex.A -
    prevSMA;
//                                          prevSMA = Duplex.A;
//                                          FPPT_runtime = FPPT_runtime + 1;
//                                          Update PlotWindow4;
//                                          Update PlotWindow1;
//                                          Update PlotWindow2;
//                                          Update PlotWindow3;
//                                          Update Drag1;
//                                          Update Fuel1;
//                                          Update Fuel2;
```

```
//                                          Update Baffle;
//                                          // Update Drag3;
//                                  End;
//                          Update PlotWindow4;
//                          Update Drag1;
//                          Update Fuel1;
//                          Update Fuel2;
//                          Update Baffle;
//                  // Update Drag3;
//                          End;
//      End;
//
//      Charge_Time = Charge_Time + 1;
//      prevSMA = Duplex.A;
//      Step Duplex;
//
//      If (Thruster_State == 4);
//              Break;
//      End;
//
//End;


// Deorbit with FPPT
prevSMA = Duplex.A;
DeltaSMA = 0.00;
While (Duplex.ElapsedTime < TIMESPAN(1000 days));
```

```
Thruster_State = 0;
Duplex.Sensors[0].Active = 0;
OrbitCount = Duplex.OrbitNumberCumulative;
Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,
    Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,MVP_I);
Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
If (Duplex.A < 6620);
        Break;
End;
If (Battery > 64);
        //If (Duplex.A > 6578.137);
        If (Duplex.A > 6600);
                    WhileManeuvering Duplex using FiniteBurn1;
                        Duplex.Thrusters[0].ThrusterOn = 1; //
                            On
                        Duplex.Thrusters[1].ThrusterOn = 0; //
                            Off
                        If (Battery < 32);
                            Update DataTableWindow4;
                            Duplex.Thrusters[0].ThrusterOn
                                = 0; // On
                            Break;
```

```
End;
//If (Duplex.A < 6578.137);
If (Duplex.A < 6600);
        Thruster_State = 4;
        Update DataTableWindow1;
        Update DataTableWindow4;
        Break;
End;
Thruster_State = 1;
Duplex.Sensors[0].Active = 1;
direction = -1; // orbit decrease SMA
Call dynamics(Duplex,sunV,velocity,
    LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
Call SMASteering(Duplex,Alpha,
    FiniteBurn2,FiniteBurn1,
    Thruster_State,direction);
Call PowerGeneration(Duplex,sunV,LVLHX
    ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,
    antiLVLHZ,Test,PYpanel,NYpanel,
    NZpanel,PZpanel,PYDpanel,NYDpanel,
    Power,PowerGenerated,
    OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,MVP_I);
Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
FPPT_I = FPPT_I + TimeStep.ToSeconds *
```

```
                                        Duplex.Thrust(Duplex.Thrusters
                                        [0]);
                                    DeltaSMA = DeltaSMA + Duplex.A -
                                        prevSMA;
                                    prevSMA = Duplex.A;
                                    FPPT_runtime = FPPT_runtime + 1;
                                    Update PlotWindow4;
                                    Update PlotWindow1;
                                    Update PlotWindow2;
                                    Update PlotWindow3;
                                    Update Drag1;
                                    Update Fuel1;
                                    Update Fuel2;
                                    Update Baffle;
                                    Update DataTableWindow4;
                                    // Update Drag3;
                            End;
                    Update PlotWindow4;
                    Update Drag1;
                    Update Fuel1;
                    Update Fuel2;
                    Update Baffle;
                    Update DataTableWindow4;
            // Update Drag3;
                    End;
End;
```

```
        Charge_Time = Charge_Time + 1;
        prevSMA = Duplex.A;
        Step Duplex;

        If (Thruster_State == 4);
                Update DataTableWindow4;
                Break;
        End;

End;

//// MVP maintain
//
//Thruster_State = 2;
//CurrSMA = Duplex.A;
//While (Duplex.ElapsedTime < TIMESPAN(7 days));
//
//      If (Thruster_State == 0);
//              Duplex.Sensors[1].Active = 0;
//              OrbitCount = Duplex.OrbitNumberCumulative;
//              Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX
    ,antiLVLHY,antiLVLHZ);
//              Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
//              Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel
    ,Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,Thruster_State,
    Battery,FPPT_I);
```

```
//              Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
//              Charge_Time = Charge_Time + 1;
//
//
//      End;
//      If (Battery > 64);
//              Thruster_State = 1;
//              Update PlotWindow4;
//              Update Drag1;
//              Update Fuel1;
//              Update Fuel2;
//              Update Baffle;
//              // Update Drag3;
//      End;
//
//
//      If (Battery > 32);
//              If (Duplex.A < CurrSMA - .05);
//                      DeltaSMA = 0;
//                      prevSMA = Duplex.A;
//                      Duplex.Thrusters[1].ThrusterOn = 1; // On
//                      Duplex.Thrusters[0].ThrusterOn = 0; // Off
//                      WhileManeuvering Duplex using FiniteBurn1;
//                              If (Battery < 32);
//                                      Thruster_State = 0;
//                                      Break;
//                              End;
```

```
//                                  If (Duplex.A > CurrSMA);
//                                          DeltaSMA = 0;
//                                          prevSMA = Duplex.A;
//                                          Break;
//                                  End;
//                                  Thruster_State = 1;
//                                  Duplex.Sensors[1].Active = 1;
//                                  direction = 1; // Orbit Increase
//                                  w = rad(Duplex.W);
//                                  f = rad(Duplex.TA);
//                                  Beta = cos(w+f)*rad(90);
//                                  Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY
    ,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
//                                  Call SMASteering(Duplex,Alpha,FiniteBurn2,
    FiniteBurn1,Thruster_State,direction);
//                                  Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,
    LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,
    PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,FPPT_I);
//                                  Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
//                                  MVP_I = MVP_I + TimeStep.ToSeconds * Duplex.
    Thrust(Duplex.Thrusters[1]);
//                                  DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
//                                  prevSMA = Duplex.A;
//                                  MVP_runtime = MVP_runtime + 1;
//                                  Update PlotWindow4;
```

```
//                                  Update PlotWindow1;
//                                  Update PlotWindow2;
//                                  Update PlotWindow3;
//                                  Update Drag1;
//                                  Update Fuel1;
//                                  Update Fuel2;
//                                  Update Baffle;
//                                  // Update Drag3;
//
//                          End;
//                  Update PlotWindow4;
//                  Update Drag1;
//                  Update Fuel1;
//                  Update Fuel2;
//                  // Update Drag3;
//          End;
//                  If (Battery < 64);
//                  Thruster_State = 0;
//                  End;
//End;
//
//
//          DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
//          prevSMA = Duplex.A;
//
//          Step Duplex;
//
```

```
//
//End;
//
//Update DataTableWindow1;
//Update DataTableWindow1;

// 1 month orbit maintence FPPT
CurrSMA = Duplex.A;
Thruster_State = 1;
While (Duplex.ElapsedTime < TIMESPAN(2 days));

        If (Thruster_State == 0);
                Duplex.Sensors[1].Active = 0;
                OrbitCount = Duplex.OrbitNumberCumulative;
                Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX
                    ,antiLVLHY,antiLVLHZ);
                Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
                Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,
                    antiLVLHY,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,
                    PYDpanel,NYDpanel,Power,PowerGenerated,OrbitAveragePower,
                    OrbitCount,wo,n,Thruster_State,Battery,FPPT_I);
                Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
                Charge_Time = Charge_Time + 1;
                Update DataTableWindow4;

        End;
        If (Battery > 64);
```

```
        Thruster_State = 1;
        Update PlotWindow4;
        Update DataTableWindow4;
End;



If (Battery > 32);
        If (Duplex.A < CurrSMA - .05);
                DeltaSMA = 0;
                prevSMA = Duplex.A;
                WhileManeuvering Duplex using FiniteBurn1;
                        Duplex.Thrusters[1].ThrusterOn = 1; // On
                        Duplex.Thrusters[0].ThrusterOn = 0; // Off
                        If (Battery < 32);
                                Thruster_State = 0;
                                Duplex.Thrusters[1].ThrusterOn = 0; //
                                    Off
                                Break;
                        End;
                        If (Duplex.A > CurrSMA);
                                DeltaSMA = 0;
                                prevSMA = Duplex.A;
                                Break;
                        End;
                        Thruster_State = 1;
                        Duplex.Sensors[1].Active = 1;
                        direction = 1; // Orbit Increase
```

```
w = rad(Duplex.W);
f = rad(Duplex.TA);
Beta = cos(w+f)*rad(90);
Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY
    ,LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ);
Call SMASteering(Duplex,Alpha,FiniteBurn2,
    FiniteBurn1,Thruster_State,direction);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,
    LVLHZ,antiLVLHX,antiLVLHY,antiLVLHZ,Test,
    PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,
    NYDpanel,Power,PowerGenerated,
    OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,FPPT_I);
Call battery(Duplex,Power,Battery,
    Thruster_State, TimeStep);
FPPT_I = FPPT_I + TimeStep.ToSeconds * Duplex.
    Thrust(Duplex.Thrusters[0]);
DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
prevSMA = Duplex.A;
FPPT_runtime = FPPT_runtime + 1;
Update PlotWindow4;
Update PlotWindow1;
Update PlotWindow2;
Update PlotWindow3;
Update Drag1;
Update Fuel1;
Update Fuel2;
```

```
                                    Update Baffle;
                                    Update DataTableWindow4;
                                    // Update Drag3;
                                End;
                        Update PlotWindow4;
                        Update DataTableWindow4;
                End;
                        If (Battery < 64);
                        Update DataTableWindow4;
                        Thruster_State = 0;
                        End;
End;


        DeltaSMA = DeltaSMA + Duplex.A - prevSMA;
        prevSMA = Duplex.A;

        Step Duplex;


End;
Update DataTableWindow1;
Update DataTableWindow4;
// MVP Deorbit
prevSMA = Duplex.A;
DeltaSMA = 0.00;
While (Duplex.ElapsedTime < TIMESPAN(1000 days));
```

```
Update Fuel1;
Update Fuel2;
Thruster_State = 0;
Duplex.Sensors[1].Active = 0;
OrbitCount = Duplex.OrbitNumberCumulative;
Call dynamics(Duplex,sunV,velocity,LVLHX,LVLHY,LVLHZ,antiLVLHX,
    antiLVLHY,antiLVLHZ);
Call sunpointing(Duplex,LVLHZ,PZpanel,wo,Power,sunV);
Call PowerGeneration(Duplex,sunV,LVLHX,LVLHY,LVLHZ,antiLVLHX,antiLVLHY
    ,antiLVLHZ,Test,PYpanel,NYpanel,NZpanel,PZpanel,PYDpanel,NYDpanel,
    Power,PowerGenerated,OrbitAveragePower,OrbitCount,wo,n,
    Thruster_State,Battery,MVP_I);
Call battery(Duplex,Power,Battery,Thruster_State, TimeStep);
If (Duplex.A < 6620);
        Update DataTableWindow4;
        Break;
End;
If (Battery > 64);
        If (Duplex.A > 6578);
                    WhileManeuvering Duplex using FiniteBurn1;
                        Duplex.Thrusters[0].ThrusterOn = 0; //
                            On
                        Duplex.Thrusters[1].ThrusterOn = 1; //
                            Off
                        If (Battery < 32);
                            Update DataTableWindow4;
                            Duplex.Thrusters[1].ThrusterOn
```

```
                = 0; // Off
            Break;
    End;
    //If (Duplex.A < 6578.137);
    If (Duplex.A < 6578);
            Thruster_State = 4;
            Update DataTableWindow1;
            Update DataTableWindow4;
            Break;
    End;
    Thruster_State = 1;
    Duplex.Sensors[0].Active = 1;
    direction = -1; // orbit decrease SMA
    Call dynamics(Duplex,sunV,velocity,
        LVLHX,LVLHY,LVLHZ,antiLVLHX,
        antiLVLHY,antiLVLHZ);
    Call SMASteering(Duplex,Alpha,
        FiniteBurn2,FiniteBurn1,
        Thruster_State,direction);
    Call PowerGeneration(Duplex,sunV,LVLHX
        ,LVLHY,LVLHZ,antiLVLHX,antiLVLHY,
        antiLVLHZ,Test,PYpanel,NYpanel,
        NZpanel,PZpanel,PYDpanel,NYDpanel,
        Power,PowerGenerated,
        OrbitAveragePower,OrbitCount,wo,n,
        Thruster_State,Battery,MVP_I);
    Call battery(Duplex,Power,Battery,
```

```
                    Thruster_State, TimeStep);
                FPPT_I = FPPT_I + TimeStep.ToSeconds *
                    Duplex.Thrust(Duplex.Thrusters
                    [0]);
                DeltaSMA = DeltaSMA + Duplex.A -
                    prevSMA;
                prevSMA = Duplex.A;
                FPPT_runtime = FPPT_runtime + 1;
                Update PlotWindow4;
                Update PlotWindow1;
                Update PlotWindow2;
                Update PlotWindow3;
                Update Drag1;
                Update Fuel1;
                Update Fuel2;
                Update Baffle;
                Update DataTableWindow4;
                // Update Drag3;
            End;
        Update PlotWindow4;
        Update Drag1;
        Update Fuel1;
        Update Fuel2;
        Update Baffle;
        Update DataTableWindow4;
// Update Drag3;
        End;
```

```
        End;


        Charge_Time = Charge_Time + 1;
        prevSMA = Duplex.A;
        Step Duplex;


        If (Thruster_State == 4);
                Update DataTableWindow4;
                Break;
        End;


End;


// Naturally Decay
While (Duplex.ElapsedTime < TIMESPAN(100 days));
        If (Duplex.A < 6378);
                Break;
        End;
        Update PlotWindow4;
        Step Duplex;
End;
```