9-18-2014

Optimal Autonomous Spacecraft Resiliency Maneuvers Using Metaheuristics

Daniel J. Showalter

OPTIMAL AUTONOMOUS SPACECRAFT RESILIENCY MANEUVERS USING METAHEURISTICS

DISSERTATION

Daniel J. Showalter, Captain, USAF

AFIT-ENY-DS-14-S-29

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENY-DS-14-S-29

OPTIMAL AUTONOMOUS SPACECRAFT RESILIENCY MANEUVERS USING METAHEURISTICS

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctoral of Science in Astronautical Engineering

Daniel J. Showalter, BS, MS

Captain, USAF

September 2014

AFIT-ENY-DS-14-S-29

OPTIMAL AUTONOMOUS SPACECRAFT RESILIENCY MANEUVERS USING METAHEURISTICS

DISSERTATION

Daniel J. Showalter, BS, MS
Captain, USAF

Approved:

/signed/	29 Jul 2014
Jonathan T. Black, PhD (Chairman)	Date
/signed/	29 Jul 2014
Darryl K. Ahner, PhD (Member)	Date
/signed/	29 Jul 2014
Raymond R. Hill, PhD (Member)	Date
/signed/	29 Jul 2014
William E. Wiesel, PhD (Member)	Date

Accepted:

/signed/	15 Aug 2014
A.B. Badiru Dean, Graduate School of Engineering and Management	Date

AFIT-ENY-DS-14-S-29

Abstract

The growing congestion in space has increased the need for spacecraft to develop resilience capabilities in response to natural and man-made hazards. Equipping satellites with increased maneuvering capability has the potential to enhance resilience by altering their arrival conditions as they enter potentially hazardous regions. The propellant expenditure corresponding to increased maneuverability requires these maneuvers be optimized to minimize fuel expenditure and to the extent which resiliency can be preserved. This research introduces maneuvers to enhance resiliency and investigates the viability of metaheuristics to enable their autonomous optimization. Techniques are developed to optimize impulsive and continuous-thrust resiliency maneuvers. The results demonstrate that impulsive and low-thrust resiliency maneuvers require only meters per second of delta-velocity. Additionally, bi-level evolutionary algorithms are explored in the optimization of resiliency maneuvers which require a maneuvering spacecraft to perform an inspection of one of several target satellites while en-route to geostationary orbit. The methods developed are shown to consistently produce optimal and near-optimal results for the problems investigated and can be applied to future classes of resiliency maneuvers yet to be defined. Results indicate that the inspection requires an increase of only five percent of the propellant needed to transfer from low Earth orbit to geostationary orbit. The maneuvers and optimization techniques developed throughout this dissertation demonstrate the viability of the autonomous optimization of spacecraft resiliency maneuvers and can be utilized to optimize future classes of resiliency maneuvers.

To Shannon

Acknowledgments

A special thanks to my advisor, Dr. Jonathan Black. His direction and feedback focused my effort and resulted in a much better product than I could have produced independently. I'd also like to thank my classmates, in particular the guys in the Outhouse. They made the whole PhD experience memorable and fun.

Daniel J. Showalter

Table of Contents

List of Figures

Figure		Page

List of Tables

List of Symbols

Symbol	Definition
a_e	semimajor axis of exclusion ellipse, km
A_T	low-thrust maneuver thrust acceleration, m/sec^2
A_T	low-thrust maneuver thrust acceleration, m/sec^2
$A_{T_{max}}$	maximum allowable low-thrust maneuver thrust acceleration, m/sec^2
$A_{T_{min}}$	minimum allowable low-thrust maneuver thrust acceleration, m/sec^2
b_e	semiminor axis of exclusion ellipse, km
c_1	swarm cognitive parameter
c_2	swarm social parameter
$\boldsymbol{g}_{best}$	global best position in the solution space
$\boldsymbol{g}_k$	unit vector perpendicular to $\boldsymbol{v}_k$ and $\boldsymbol{h}_k$ at k^{th} expected time of entry into exclusion zone
$\boldsymbol{h}_k$	expected angular momentum vector of satellite at k^{th} time of entry into exclusion zone, km^2/sec
$i^{m,c}$	inclination of the target, chaser orbit rad
J	cost of nonlinear function to be optimized
$J_{g_{best}}$	lowest cost associated with the swarm
$J_{l_{best}}$	lowest cost associated with the neighborhood
$J_{p_{best}}$	lowest cost associated with the particle
$J_p(s)$	cost associated with a particle at the s^{th} iteration
$\boldsymbol{l}_{best}$	neighborhood best position in the solution space
l_{GEO}	true longitude at epoch of the arrival location on the geostationary orbit, rad
m	number of particles in the swarm
n	number of design variables in the nonlinear function to be optimized

Symbol	Definition
N_{rev}	minimum number of orbital revolutions required for multiple revolution impulsive maneuver
P	period of the initial orbit, *sec*
$\boldsymbol{p}_{best}$	particle best position in the solution space
R_{a_k}	orbit apogee radius after the k^{th} maneuver, *km*
$\boldsymbol{r}^c_{CYL}$	position vector of the chaser in the cylinder coordinate frame, *km*
$\boldsymbol{r}^c_{IJK}$	position vector of the chaser in the inertial coordinate frame, *km*
R_e	distance from expected position of the spacecraft to the actual position of the spacecraft, *km*
$\boldsymbol{r}^g_{IJK}$	inertial position vector of the ground site, *km*
$\boldsymbol{r}_k$	expected position vector of satellite at k^{th} time of entry into exclusion zone, *km*
$\boldsymbol{r}^*_k$	actual position vector of spacecraft at k^{th} time of entry into exclusion zone, *km*
$\boldsymbol{r}_{k_t^-}$	position vector at the instant just before the k^{th} impulse, *km*
$\boldsymbol{r}^m_{IJK}$	inertial position vector of the *m*th target
R_{max}	maximum allowable orbital radius, *km*
R_{min}	minimum allowable orbital radius, *km*
R_{p_k}	orbit perigee radius after the k^{th} maneuver, *km*
$\boldsymbol{r}^c_{RSW}$	position vector of the chaser in the local vertical, local horizontal coordinate frame, *km*
$\boldsymbol{r}_0$	initial position vector, *km*
$R_\oplus$	radius of the earth, *km*
$\boldsymbol{S}$	Solution space encompassing all n design variables
t_k	expected k^{th} time of entry into exclusion zone, *sec*
t_f	final time of GTMEI scenario, *sec*
t^k_{enter}	entry time of the *m*th target's *k*th pass over the ground site, *sec*

Symbol	Definition
t^k_{exit}	exit time of the *m*th target's *k*th pass over the ground site, *sec*
T_k	time of flight of the k^{th} maneuver, *sec*
t_1	time of initial impulsive maneuver to cooperative inspection segment, *sec*
t_2	time of fight for maneuver from initial orbit to cooperative inspection segment, *sec*
t_3	coast time following cooperative inspection phase, *sec*
t_4	time of flight for maneuver to the final mission orbit, *sec*
$u^{m,c}$	argument of latitude of the target, chaser orbit *rad*
$\boldsymbol{v}^c_{IJK}$	velocity vector of the chaser in the inertial coordinate frame, *km*
$\boldsymbol{v}_k$, $\boldsymbol{v}^*_k$	expected and actual velocity vector of satellite at k^{th} time of entry into exclusion zone, km/sec
$\boldsymbol{v}_{k^-_i}$, $\boldsymbol{v}_{k^+_i}$	velocity vectors at the instant just before and just after the k^{th} impulse, km/sec
$\boldsymbol{V}_p(s)$	*n*-dimensional velocity vector of the p^{th} particle at the s^{th} iteration
v^i_{max}, v^i_{min}	upper and lower bounds on the velocity of the i^{th} design variable
$\boldsymbol{v}_0$	initial velocity vector, km/sec
$\boldsymbol{v}^c_{RSW}$	velocity vector of the chaser in the local vertical, local horizontal coordinate frame, *km*
$\boldsymbol{X}_p(s)$	*n*-dimensional position vector of the p^{th} particle at the s^{th} iteration
x^i_{max}	upper and lower bounds on the position of the i^{th} design variable
α	angle measured from the orbital plane to the cylinder in the local vertical, local horizontal frame, *rad*
β	angle measured from the primary axis to the cylinder in the local vertical, local horizontal frame, *rad*
$\Delta\boldsymbol{V}_k$	velocity vector of the k^{th} maneuver, km/sec

Symbol	Definition
ΔV_k	cost of the k^{th} maneuver, m/sec
γ_{k_f}	pre-maneuver flight path angle for the k^{th} maneuver, rad
γ_{k_t}	pre-maneuver flight path angle for the k^{th} maneuver, rad
ϵ^c	elevation angle of the chaser with respect to the ground site, rad
ϵ^c_{max}	maximum allowable elevation angle of the chaser with respect to the ground site, rad
ϵ^g_{min}	minimum elevation angle required by ground site for line-of-sight contact with the *m*th target, rad
ϵ^m	elevation angle of the target satellite, rad
η	thrust pointing angle, rad
θ_k	angle defining position of spacecraft on the k^{th} exclusion ellipse, rad
μ	Earth's gravitational parameter, km^3/sec^2
ν_{enter}	true anomaly of the spacecraft as it enters the latitude band of the exclusion zone
$\boldsymbol{\rho}_{IJK}$	vector from the ground site to the target in the inertial coordinate frame, km
$\boldsymbol{\rho}_{RSW}$	vector from the ground site to the target in the local vertical, local horizontal coordinate frame, km
$\boldsymbol{\rho}_{SEZ}$	vector from the ground site to the target in the topocentric horizon coordinate frame, km
ϕ, λ	geocentric latitude and longitude, rad
χ	swarm constriction factor
ψ_k	expected angle traveled by the spacecraft in the orbit plane during the k^{th} maneuver, rad
ψ_k^*	actual angle traveled by the spacecraft in the orbit plane during the k^{th} maneuver, rad
$\omega_{\oplus}$	rotation rate of the earth, rad

Symbol	Definition
$\Omega^{m,c}$	right ascension of the ascending node of the target, chaser orbit *rad*

List of Acronyms

Acronym	Definition
ACO	ant colony optimization
AIAA	American Institute of Aeronautics and Astronautics
B&B	branch and bound
COV	calculus of variations
CP	conditional penalty
CW	Clohessy-Wiltshire
CYL	cylinder coordinate frame
DE	differential evolution
DOC	direct orthogonal collocation
DoD	Department of Defense
DOE	design of experiments
DSB	Defense Science Board
DTRK	direct transcription with Runge-Kutta implicit integration
EA	evolutionary algorithm
GA	genetic algorithm
GBEST	global best particle swarm optimization variant
GMT	Greenwich Mean Time
GP	genetic algorithm outer-loop with inner-loop particle swarm
GPi	genetic algorithm outer-loop with inner-loop particle swarm employing infeasible cutoff
GTMEI	geostationary transfer maneuver with cooperative en-route inspection
HOC	hybrid optimal control
IJK	geocentric equatorial coordinate frame

Acronym	Definition
IPOPT	Interior Point Optimizer
LBEST	local best particle swarm optimization variant
LEO	low Earth orbit
LTRTM	low-thrust responsive theater maneuver
MBH	monotomic basin hopping
MEO	mid-Earth orbit
MGA	multi gravity assist
MGADSM	multi gravity assist with deep space maneuvers
NLP	nonlinear programming
NSP	National Space Policy
NSSS	National Security Space Strategy
PP	particle swarm outer-loop with inner-loop particle swarm
PPi	particle swarm outer-loop with inner-loop particle swarm employing infeasible cutoff
PQW	perifocal coordinate frame
PSO	particle swarm optimization
PSOG	global PSO
PSOL	local PSO
QDR	Quadrennial Defense Review
RSW	local vertical, local horizontal coordinate frame
RTM	responsive theater maneuver
SA	simulated annealing
SAM	specific angular momentum
SME	specific mechanical energy
SEZ	topocentric horizon coordinate frame
SSA	space situational awareness

Acronym	Definition
SUS	stochastic universal sampling

OPTIMAL AUTONOMOUS SPACECRAFT RESILIENCY MANEUVERS USING METAHEURISTICS

I. Introduction

1.1 Motivation

The United States has long enjoyed a competitive advantage over the rest of the world in the space domain. As a result, it has relied heavily on space capabilities to provide products and services to military and civilian users.

The United States' asymmetric advantage in space has decreased in recent years as more countries have invested in space capabilities. In addition, the space environment itself has changed from an uncontested one to an environment in which access to and the use of space can no longer be taken for granted. In light of this shifting paradigm, President Obama released an updated National Space Policy (NSP) in 2010 [1] which states "The United States will employ a variety of measures to help assure the use of space for all responsible parties, and, consistent with the inherent right of self-defense, deter others from interference and attack, defend our space systems and contribute to the defense of allied space systems, and, if deterrence fails, defeat efforts to attack them."

The Department of Defense (DoD) released its National Security Space Strategy (NSSS) in 2011 in response to the guidance specified in the NSP. One of the key tenets of this strategy is to deter attacks on U.S. systems by denying adversaries the benefits of attacks through "cost-effective" protection and resilience [2].

Similarly, the 2014 Quadrennial Defense Review (QDR) highlighted the need to prepare for adversary attempts to deny current U.S. advantages in space [3]. In response to this threat, the QDR states that the United States “will move toward less complex, more affordable, more resilient systems...to deter attacks on space systems.”

An NSSS supplemental document on resilience highlighted four basic principles which define resilience: avoidance, robustness, reconstitution, and recovery [4]. The NSSS supplement defines avoidance as “countermeasures against potential adversaries, proactive and reactive defensive measures taken to diminish the likelihood and consequence of hostile acts or adverse conditions” [4].

U.S reliance on space capabilities for military operations and intelligence [2] and the global nature of space systems make it impossible to avoid potentially hostile areas of the globe. As a result, resilience through avoidance in space must be achieved by preventing the occurrence of hostile action. One way to prevent hostile action is to introduce uncertainty into the arrival conditions of friendly space assets when they overfly potentially hazardous geographic regions on the Earth. This uncertainty can be achieved by equipping space assets with enhanced maneuvering capability which would allow them to modify their arrival conditions from those predicted by previous observations and orbit prediction algorithms.

Increased resiliency through satellite maneuverability comes at a price, however, specifically in terms of the amount of propellant required to achieve it. Increased maneuverability requires additional propellant for a given mission, which in turn leads to heavier satellites and larger launch costs. Currently, it costs nearly $10,000 per pound to place a satellite into Earth orbit [5]. As a result, avoidance maneuvers should be optimized to minimize the amount of propellant consumed during their execution to the extent which resiliency can be preserved.

Generating optimal spacecraft trajectories comes with its own cost with respect to the manpower required for design and analysis. One way to address the long-term manpower costs associated with maneuverability is to introduce autonomy into the maneuver optimization process. A recent DoD Defense Science Board (DSB) study on the role of autonomy states that increased use of autonomy in space systems “has the potential to enable manpower efficiencies and cost reductions” [6]. The study also states that increased spacecraft autonomy can make U.S. systems more adaptive to operational variations and anomalies, and therefore may be a key to resiliency.

The DSB study [6] also states “two promising space system application areas for autonomy are the increased use of autonomy to enable an independent acting system and automation as an augmentation of human operation. In such cases, autonomy’s fundamental benefits are to increase a system’s operational capability and provide cost savings via increased human labor efficiencies, reducing staffing requirements and increasing mission assurance.” The DSB study also highlights the need to develop automated planning to facilitate the decomposition of high level objectives into a series of actions to achieve them [6].

Accurate and timely space situational awareness (SSA) is critical to autonomous satellite resiliency. Specifically, the need for accurate tracking and characterization of orbiting objects is necessary to prevent unintended consequences, such as collisions, which could result from maneuvering. The NSSS highlights the importance of SSA to ensure safe space operations [2]. SSA is particularly relevant to autonomous maneuver generation While the DoD and other organizations track over $20,000$ objects, the are still “hundreds of thousands of additional objects that are too small to track” [2]. As a result, SSA is a top priority for the DoD space enterprise. Specifically, the NSSS highlights the need for SSA to be obtained in higher quantities and with better quality [2].

1.2 Background

The field of spacecraft trajectory optimization has been extensively researched. The development of modern tools such as evolutionary algorithms (EAs) and metaheuristics have made a significant impact on the field. The impact results from the fact that EAs and metaheuristics do not require initial guesses, something on which more traditional methods are dependent. Additionally, EAs are more likely to find a global minimum than more traditional methods. The use of EAs and metaheuristics in spacecraft trajectory optimization has seen a dramatic increase due to these benefits. The limitations of EAs, namely that problems must be parameterized into a relatively small set of variables, can be overcome by employing more traditional optimization techniques to refine results generated by EAs. In fact, the current state-of-the-art in trajectory optimization is to utilize an EA or metaheuristic independently or as a method to generate initial guesses for a direct transcription method [7].

Several researchers have employed these techniques to investigate interplanetary missions [8–25] or asteroid rendezvous and interception [26–31]. There is significantly less research in optimal trajectory design to achieve mission-focused ground effects. Existing research in this field has focused on orbit design for optimal coverage [32, 33] or low-thrust maneuvering to improve responsive coverage of designated ground sites [34, 35].

Currently, there is no trajectory optimization research focused on spacecraft resiliency. The purpose of this dissertation is to develop resiliency maneuvers and the tools which will enable their autonomous generation. This research utilizes modern optimization methods to demonstrate their utility in solving several spacecraft trajectory optimization problems, such as impulsive and continuous low-thrust resiliency maneuvers as well as hybrid optimal control (HOC) problems.

1.3 Research Objectives

The primary objective of this dissertation is to develop spacecraft resiliency maneuvers and the tools which enable their autonomous optimization. This objective is accomplished in three phases, which are covered in Chapters 3, 4, and 5. The first phase consists of the design and optimization of impulsive resiliency maneuvers. This phase is the jumping off point for this dissertation because impulsive maneuvers can be defined by a relatively small set of parameters, which allows for a performance evaluation of various optimization algorithms. The second phase of this research extends resiliency to continuous-thrust maneuvers, which require the definition of a large control history. The final phase of this research investigates maneuvers designed to increase SSA. The optimization of these SSA maneuvers are formulated as hybrid optimal control problems, which consist of a mixture of categorical and continuous variables. The results from all three phases demonstrate the potential for the autonomous optimization of spacecraft resiliency maneuvers in support of human operations.

1.4 Document Preview

This dissertation follows the scholarly article format, in which the research contributions in Chapters 3, 4, and 5 are presented as they appeared/were submitted to various journals. The document is structured according to the following outline.

Chapter 2 provides background on the coordinate frames and governing equations of motion employed in this dissertation. Additionally, it presents a literature review detailing current and past research relevant to autonomous trajectory optimization. The literature review is divided into three sections. The first provides information on enabling techniques in orbital mechanics which are foundational to the methods described in this dissertation. The second section details optimization techniques and the final section provides a description of relevant research in spacecraft trajectory optimization.

Chapter 3 develops an impulsive maneuvering strategy to enable satellite resiliency and evaluates several EAs in the optimization of these types of maneuvers. Example results are presented for single, double, and triple pass scenarios over a specified geographic region on the surface of the Earth. This work was accepted for published by the American Institute of Aeronautics and Astronautics (AIAA) Journal of Spacecraft and Rockets in July 2014.

Chapter 4 presents a continuous, low-thrust implementation of the maneuvers defined in Chapter 3. Feasible solutions to the low-thrust problems presented are generated using particle swarm optimization (PSO) algorithms, which are used to seed a direct optimization method to determine the true optimal trajectory and control history. Example results are presented for single, double, and triple pass scenarios. This work is under peer review for publication in the AIAA Journal of Spacecraft and Rockets.

Chapter 5 introduces an impulsive maneuvering strategy to deliver a spacecraft to its final mission orbit while providing an en-route inspection of an uncharacterized orbiting target in cooperation with a ground-based sensor. The performance of four different HOC algorithms are investigated in the optimization of a simple three target problem. The best performing algorithm is then utilized to optimize a fifteen target problem. This work is under peer review for publication in Acta Astronautica.

Chapter 6 summarizes the major contributions of this research and highlights potential areas for future work.

II. Background

As stated in Chapter 1, the goal of this research is to develop, optimize, and enable the autonomous generation of maneuvers that enhance spacecraft resiliency. The field of spacecraft trajectory optimization requires a fundamental understanding of both astrodynamics and optimization. The purpose of this section is to provide the necessary background in these areas to lay the foundation for the methods developed in Chapters 3, 4, and 5. This background is divided into four sections: coordinate frames, system dynamics, enabling techniques, and optimization techniques.

2.1 Coordinate Frames

The methods developed in subsequent chapters utilize a variety of coordinate frames, each of which is more convenient than others for various applications. This dissertation employs five different coordinate frames: the geocentric equatorial coordinate frame (IJK), the perifocal coordinate frame (PQW), the topocentric horizon coordinate frame (SEZ), the local vertical, local horizontal coordinate frame (RSW), and the cylinder coordinate frame (CYL). Definitions of the IJK, PQW, SEZ, and RSW frames are provided in [36, pp. 153-166] and presented here for completeness. The CYL frame was developed as part of this research and is defined completely in Chapter 5.

2.1.1 Geocentric Equatorial Coordinate Frame

The most common coordinate frame used throughout this dissertation is the IJK frame. Its origin is the center of the earth and the earth's equatorial plane is the fundamental plane of the frame. The principle axis $\hat{I}$ points toward the vernal equinox and is coincident with the intersection of the equatorial and ecliptic planes. The $\hat{K}$-axis is perpendicular to the equatorial plane and points towards the Earth's north pole. The $\hat{J}$-axis completes the right-handed coordinate system. Figure 2.1 depicts the IJK frame.

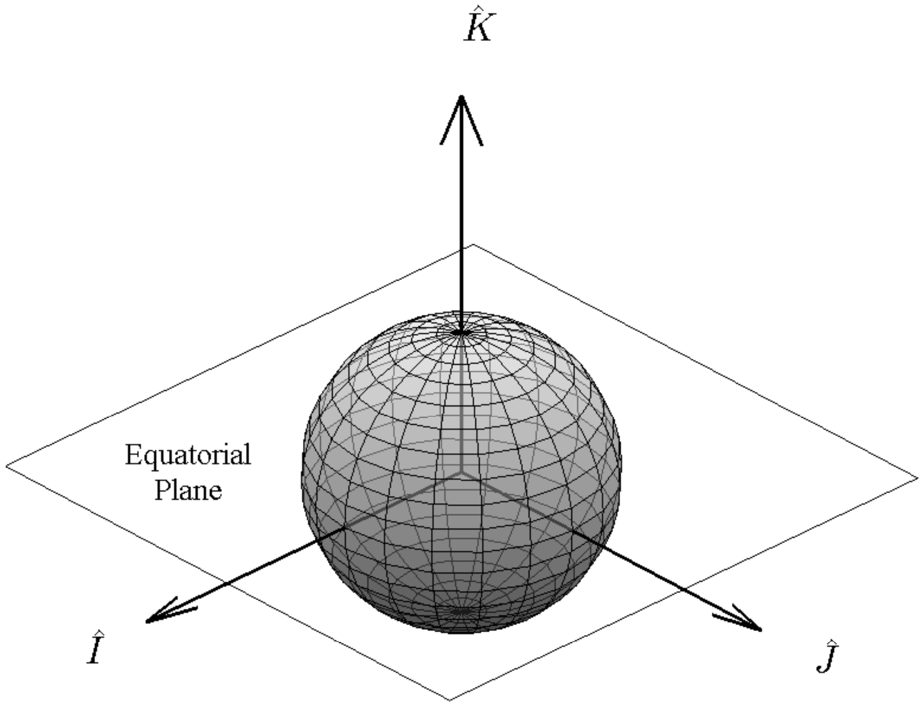


Figure 2.1: Geocentric equatorial coordinate frame

For the duration of this dissertation, the states of all spacecraft are defined in Cartesian coordinates whenever the IJK frame is used. As a result, the state of a spacecraft in the IJK frame is given by the position $\boldsymbol{r}$ and velocity $\boldsymbol{v}$ vectors shown in Equation 2.1.

$$\begin{aligned} \boldsymbol{r} &= x\hat{I} + y\hat{J} + z\hat{K} \\ \boldsymbol{v} &= v_x\hat{I} + v_y\hat{J} + v_z\hat{K} \end{aligned} \tag{2.1}$$

2.1.2 Perifocal Coordinate Frame

The PQW frame is convenient for describing the motion of a spacecraft in the orbital plane. The origin of the PQW frame is the center of the earth and its fundamental plane is coplanar with the satellite's orbital plane. The principal axis $\hat{P}$ is aligned with perigee of the satellite's orbit. The $\hat{Q}$-axis is in the fundamental plane and 90° from the $\hat{P}$-axis in the direction of motion. The $\hat{W}$-axis is normal to the orbital plane and completes the right-handed system. Figure 2.2 depicts the PQW frame.

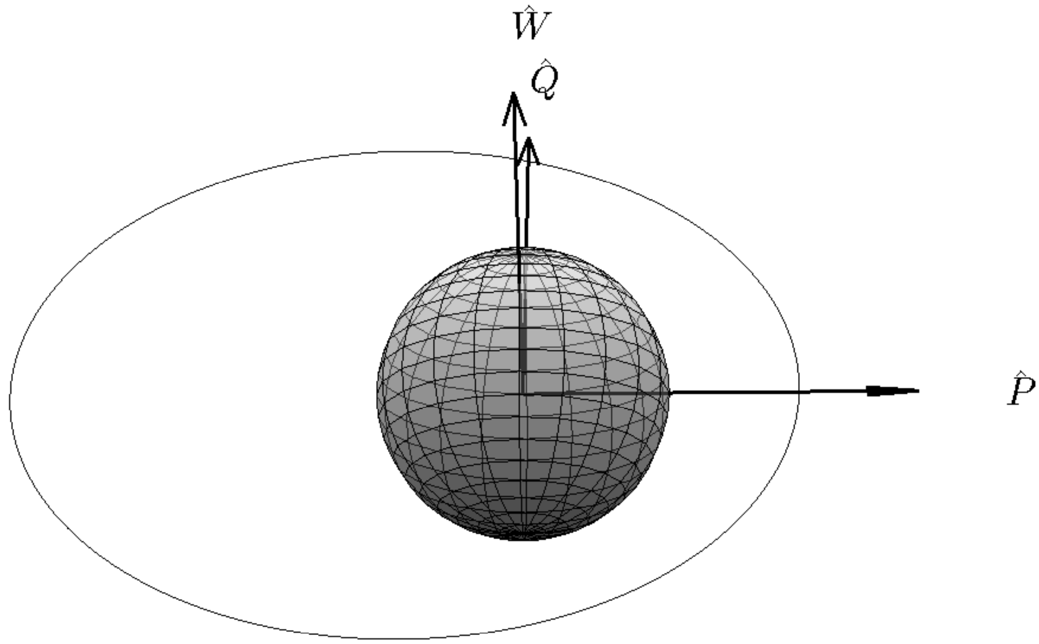


Figure 2.2: Perifocal coordinate frame

The rotation of a position vector $\boldsymbol{r}_{IJK}$ in the IJK frame to a corresponding vector $\boldsymbol{r}_{PQW}$ in the PQW frame is defined by Equation 2.2. The variables ω, inc, and Ω are the orbit's argument of perigee, inclination, and right ascension of the ascending node, respectively. $R1$ and $R3$ are rotation matrices about the first and third axes, respectively.

$$\boldsymbol{r}_{PQW} = R3\,(\omega)\,R1\,(inc)\,R3\,(\Omega)\,\boldsymbol{r}_{IJK} \tag{2.2}$$

It is important to note that the PQW frame is undefined for equatorial or circular orbits. For circular orbits, it is common to use the nodal coordinate frame in place of the PQW frame. In such cases, the $\hat{P}$-axis is defined to be coincident with the ascending node of the satellite's orbit. A vector in the nodal frame can be found according to Equation 2.2 where ω is replaced with zero.

Some of the techniques used throughout this dissertation define the states of spacecraft in the PQW and nodal frames using spherical coordinates. Figure 2.3 depicts the definitions of these spherical coordinates in the PQW frame. In such cases, r represents the magnitude of the position vector, ψ is the angle measured from the $\hat{P}$-axis to the spacecraft in the orbital plane, and ϕ (not-depicted) is the out-of-plane angle.

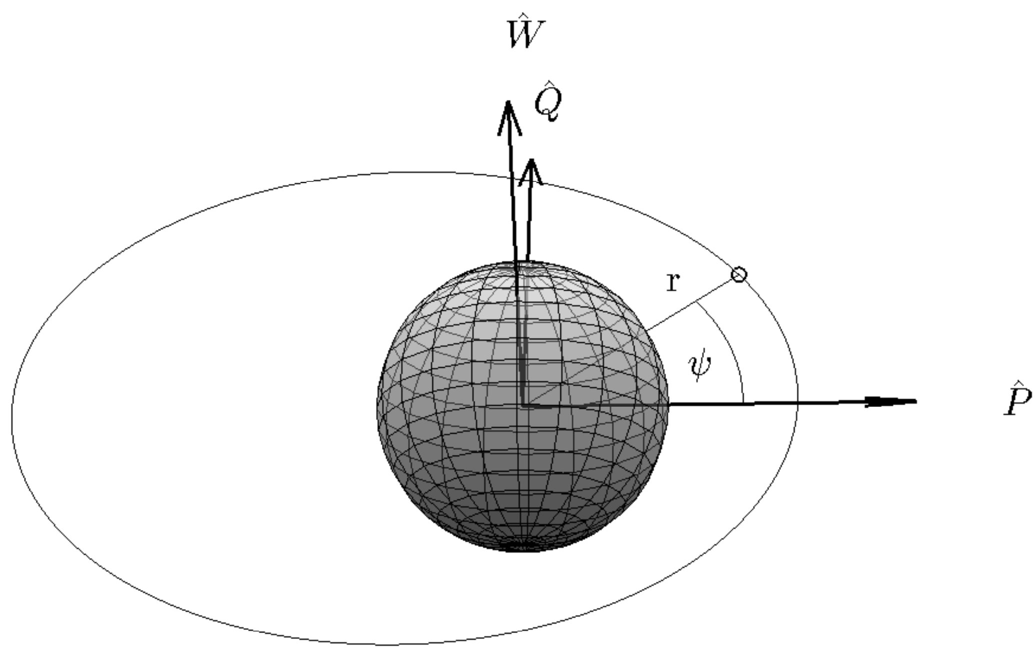


Figure 2.3: Spherical coordinate definitions in perifocal coordinate frame

2.1.3 Topocentric Horizon Coordinate Frame

The SEZ coordinate frame is an Earth-based reference system, the origin of which is located at a point on the earth's surface defined by its geocentric latitude Φ and longitude λ. The SEZ frame rotates with the earth and is oriented such that the $\hat{S}$ axis points south from the origin and the $\hat{E}$ axis points east. The $\hat{Z}$ axis is normal to the earth's surface. The rotation from the IJK frame into the SEZ frame is shown in Equation 2.3 where $\omega_{\oplus}$

is the rotation rate of the Earth and $t_{\hat{I}}$ is the current local sidereal time at the origin of the SEZ frame. $R2$ and $R3$ are rotation matrices about the second and third axes, respectively. Figure 2.4 depicts the SEZ coordinate frame.

$$\boldsymbol{r}_{SEZ} = R2\left(\pi/2 - \Phi\right) R3\left(\Lambda + \omega_{\oplus} t_{\hat{I}}\right) \boldsymbol{r}_{IJK} \tag{2.3}$$

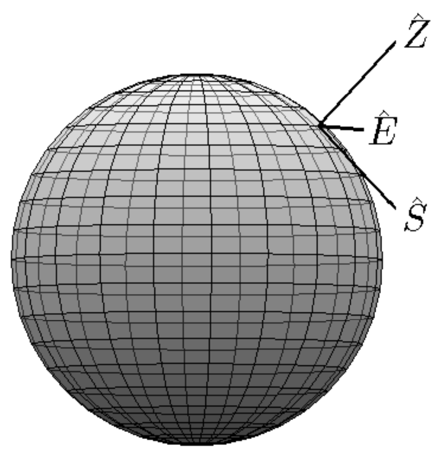


Figure 2.4: Topocentric horizon coordinate frame

The SEZ frame is employed in this dissertation to determine a satellite's line-of-sight contact with a ground site, which occurs when the $\hat{Z}$ component of a satellite's position vector in the SEZ frame is positive.

2.1.4 Local Vertical, Local Horizontal Coordinate Frame

The RSW frame is a satellite-based coordinate frame, the origin of which is the orbiting satellite. The principle $\hat{R}$-axis is aligned with the vector connecting the origin of the earth to the satellite. The $\hat{S}$-axis is perpendicular to $\hat{R}$ and points in the direction of the satellite's velocity vector while the $\hat{W}$-axis is perpendicular to the orbit plane. Equation 2.4 provides the transformation for a vector $\boldsymbol{r}_{PQW}$ in the PQW frame into a vector $\boldsymbol{r}_{RSW}$ in the RSW frame, where ν is the true anomaly. Figure 2.5 shows the RSW frame.

$$\boldsymbol{r}_{RSW} = R3\left(\nu\right)\boldsymbol{r}_{PQW} \tag{2.4}$$

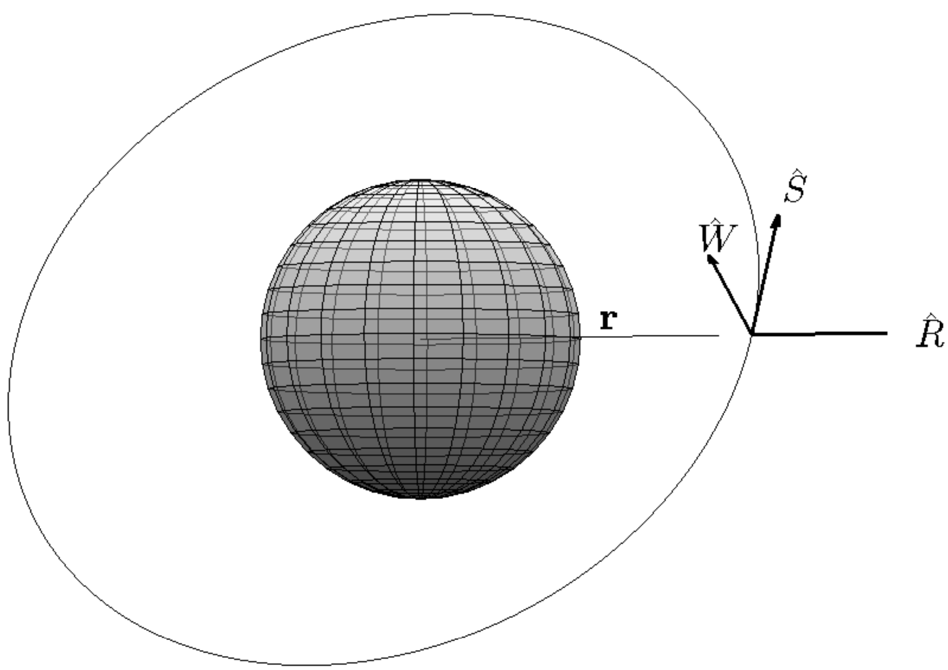


Figure 2.5: Local vertical, local horizontal coordinate frame

2.2 Orbital Mechanics

The entirety of this research focuses on Earth-orbiting satellites. Consequently, this section will focus on the dynamics of a satellite as it orbits the earth. For the purposes of this research, two sets of dynamical equations are presented here. The first set of equations are employed for satellite motion in the IJK frame while the second set are utilized in the PQW or nodal frames.

The underlying principles for the motion of the spacecraft about the earth result from Newton's second law and universal law of gravitation. Several resources [36, pp. 20-31], [37, pp. 1-40], and [38, pp. 130-138] present derivations of the equations of motion beginning with these underlying principles and several simplifying assumptions. These assumptions, known as the two-body assumptions, include:

1. The coordinate frame is inertial, meaning that it does not rotate or accelerate.
2. The earth and spacecraft are modeled by spheres of uniform density, allowing them to be treated as point masses.
3. The mass of the spacecraft is much less than that of the earth.
4. The only forces acting on the earth and spacecraft are the gravitational forces between them.

2.2.1 Equations of Motion in Geocentric Equatorial Frame

The two-body assumptions lead to the equations of motion governing spacecraft motion about the earth. The state of the spacecraft in Cartesian coordinates is defined by position and velocity vectors, $\boldsymbol{r}$ and $\boldsymbol{v}$, respectively. In the IJK frame, $\boldsymbol{r}$ and $\boldsymbol{v}$ take the form shown in Equation 2.5.

$$\begin{aligned} \boldsymbol{r} &= x\hat{I} + y\hat{J} + z\hat{K} \\ \boldsymbol{v} &= v_x\hat{I} + v_y\hat{J} + v_z\hat{K} \end{aligned} \tag{2.5}$$

The Cartesian form of the equations of motion are presented in Equation 2.6 where μ is the Earth's gravitational constant.

$$\begin{bmatrix} \dot{\boldsymbol{r}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ -\frac{\mu}{|\boldsymbol{r}|^3}\boldsymbol{r} \end{bmatrix} \tag{2.6}$$

All maneuvers in this dissertation defined in the IJK frame are impulsive. That is, they occur instantaneously. A maneuver is defined by a vector $\boldsymbol{\Delta V}$ with components in each axis of the IJK frame as shown in Equation 2.7. The cost of each maneuver is ΔV, the magnitude of $\boldsymbol{\Delta V}$, which is equal to the difference between the velocity vector at the instant after the maneuver, $\boldsymbol{v}^+$, and the velocity vector at the instant prior to the maneuver, $\boldsymbol{v}^-$.

$$\boldsymbol{\Delta V} = \boldsymbol{v}^+ - \boldsymbol{v}^- \tag{2.7}$$

2.2.2 Equations of Motion in Perifocal and Nodal Frames

The two-body assumptions also make it possible to derive two constants of orbital motion, specific angular momentum (SAM) and specific mechanical energy (SME). Original derivations for SAM and SME are presented in [36, pp. 23-27] and [37, pp. 14-18]. The SAM of an orbit, $\boldsymbol{h}$, can be found according to Equation 2.8.

$$\boldsymbol{h} = \boldsymbol{r} \times \boldsymbol{v} \tag{2.8}$$

The conservation of SAM implies that the motion of a non-maneuvering spacecraft is confined to its orbital plane. As a result, consider the motion of a spacecraft in the PQW or nodal frames. The conservation of SAM dictates that the motion of a non-maneuvering spacecraft is restricted to the $\hat{P}\hat{Q}$ plane. Consequently, only four states are necessary to completely describe the motion of a spacecraft in the PQW or nodal frames if motion is restricted to the orbital plane. Throughout this dissertation, spherical coordinates are used to represent the state of a spacecraft in the PQW or nodal frames. Further, all maneuvers in the PQW frame are coplanar and modeled as continuous using a thrust acceleration vector,

$\boldsymbol{A_T}$. The thrust acceleration vector is defined by its magnitude, A_T and the angle η measured from local horizontal to $\boldsymbol{A_T}$, as shown in Figure 2.6. The local horizontal is defined as a line perpendicular to the position vector in the orbital plane. .

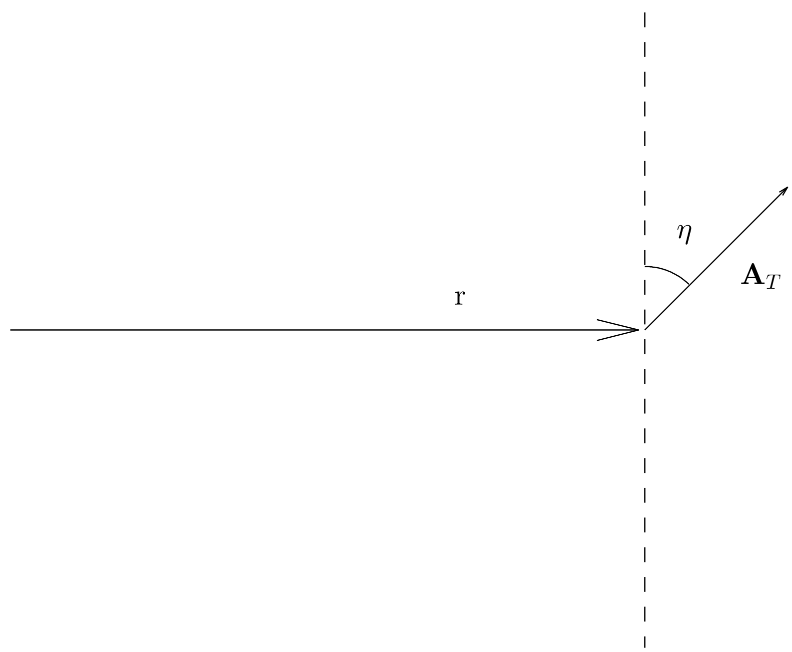


Figure 2.6: Thrust acceleration vector

The resulting equations of motion in the PQW and nodal frames are defined in Equation 2.9.

$$
\begin{aligned}
\dot{r} &= V_r \\
\dot{\psi} &= \frac{V_\psi}{r} \\
\dot{V}_r &= \frac{V_\psi^2}{r} - \frac{\mu}{r^2} + A_T \sin\eta \\
\dot{V}_\psi &= \frac{-V_\psi V_r}{r} + A_T \cos\eta
\end{aligned}
\tag{2.9}
$$

The ΔV corresponding to continuous-thrust maneuvers is found according to Equation 2.10, where t_{start} is the maneuver start time and t_{end} is the maneuver end time.

$$\Delta V = \int_{t_{start}}^{t_{end}} A_T \, \mathrm{d}t \tag{2.10}$$

2.3 Literature Review

The field of spacecraft trajectory optimization has been studied extensively since the 1960s when Lawden [39] applied the calculus of variations (COV) to determine the necessary conditions for optimal impulsive transfers between circular orbits. None of the previous research, however, has developed maneuvers designed to enhance spacecraft resiliency. This literature review provides a comprehensive overview of current and past research techniques that enable the optimization of resiliency maneuvers with varying levels of autonomy. The areas which provide the foundation of this research are divided into three categories: enabling techniques, numerical optimization techniques, and spacecraft trajectory optimization research.

2.3.1 Enabling Techniques

The methods developed to design and optimize resiliency maneuvers described in Chapters 3, 4, and 5 of this dissertation employ several techniques developed by other researchers. This section of the literature review is meant to provide brief descriptions of these enabling methods and references of their use in other research. The techniques are analogous to one another because all are used to determine the trajectory that will deliver a spacecraft from one position to another in a specified time. They are distinct, however, due to the coordinate frames they utilize or the types of trajectories they generate: either impulsive or continuous thrust.

2.3.1.1 Gauss' Problem

The first enabling method used is a solution to the classic Lambert's problem (originally proposed by Gauss), which is to determine the initial and final velocity vectors of an orbit segment which connects two position vectors in a specified time-of-flight. These velocity vectors can be used to determine the impulsive ΔV required to transfer a satellite from its current orbit to a specified position in a fixed amount of time. A Lambert's problem solver is particularly useful because it allows a trajectory to be defined by a small number of parameters.

Many techniques have been developed in the solution of Lambert's problem, most famously Gauss' solution, derivations of which are presented in [37, pp. 258-264], [36, pp. 472-475] and [40, pp. 325-342]. [41] provides a software algorithm to solve Lambert's problem which is used throughout this dissertation.

2.3.1.2 Shape-Based Low-Thrust Trajectory Approximation

Shape-based low-thrust trajectory approximation is employed to determine discrete approximations to low thrust-trajectories connecting two known positions in a specified time. The approximation was originally developed and presented in [42, 43]. The highlights are presented here for clarity. [42] developed a two-dimensional approximation which is appropriate for interception trajectories restricted to motion occurring in a fixed-plane. The approximation utilizes a spherical coordinate system and provides a sixth-degree inverse polynomial approximation of r as a function of ψ, shown in Equation 2.11.

$$r(\psi) = \frac{1}{a + b\psi + c\psi^2 + d\psi^3 + e\psi^4 + f\psi^5 + g\psi^6} \tag{2.11}$$

The values for a, b, and c are dependent on the initial boundary conditions: position magnitude r_i, velocity magnitude v_i, and flight path angle γ_i. Let the initial angle ψ_i equal zero and the final angle ψ_f equal the total angle to be traveled by the maneuvering spacecraft. Then a, b, and c take on the values shown in Equation 2.12, where μ is the gravitational parameter of the body about which the spacecraft is orbiting.

$$a = \tfrac{1}{r_i} \quad b = \tfrac{\tan\gamma_i}{r_i} \quad c = \tfrac{1}{2r_i}\left(\tfrac{\mu}{r_i^3\dot{\psi}_i^2} - 1\right) \tag{2.12}$$

where

$$\dot{\psi}_i = \frac{v_i \cos\gamma_i}{r_i} \tag{2.13}$$

The value for d is chosen to specify the transfer time and must be solved with a root finding function. The values for e, f, g are dependent on d and the final boundary conditions: transfer time t_f, position magnitude r_f, velocity magnitude v_f, and flight path angle γ_f.

$$\begin{bmatrix} e \\ f \\ g \end{bmatrix} = \frac{1}{2\psi_f^6} \begin{bmatrix} 30\psi_f^2 & -10\psi_f^3 & \psi_f^4 \\ -48\psi_f & 18\psi_f^2 & -2\psi_f^3 \\ 20 & -8\psi_f & \psi_f^2 \end{bmatrix} \begin{bmatrix} \frac{1}{r_f} - \left(a + b\psi_f + c\psi_f^2 + d\psi_f^3\right) \\ -\frac{\tan\gamma_f}{r_f} - \left(b + 2c\psi_f + 3d\psi_f^2\right) \\ \frac{\mu}{r_f^4 \dot{\psi}_f^2} - \left(\frac{1}{r_f} + 2c + 6d\psi_f\right) \end{bmatrix} \tag{2.14}$$

The shape based approximation is complete when a value of d satisfies the relationship in Equation 2.15.

$$\int_0^{t_f} \mathrm{d}t = \int_0^{\psi_f} \sqrt{\frac{r(\psi)^4}{\mu} \left[1/r(\psi) + 2c + 6d\psi + 12e\psi^2 + 20f\psi^3 + 30g\psi^4\right]} \,\mathrm{d}\psi \tag{2.15}$$

[42] notes that supplying an initial guess of $d = 0$ into the MATLAB root finding function *fzero* provides sufficient robustness to satisfy the relationship defined in Equation 2.15. The approximation for the thrust acceleration is found according to Equation 2.16.

$$A_T = -\frac{\mu}{2r^3 \cos\gamma} \frac{6d + 24e\psi + 60f\psi^2 + 120g\psi^3 - (\tan\gamma)/r}{\left(1/r + 2c + 6d\psi + 12e\psi^2 + 20f\psi^3 + 30g\psi^4\right)^2} \tag{2.16}$$

where

$$\tan\gamma = \frac{\dot{r}}{r\dot{\psi}} = -r\left(b + 2c\psi + 3d\psi^2 + 4e\psi^3 + 5f\psi^4 + 6g\psi^5\right) \tag{2.17}$$

The approximation is assumed to be a prograde trajectory, implying that the flight path angle, γ, must be between $-\pi/2$ and $\pi/2$ The corresponding ΔV can be found by integrating Equation 2.18 using quadrature and the trapezoidal rule.

$$\Delta V = \int_0^{\psi_f} \frac{A_T}{\dot{\psi}} \,\mathrm{d}\psi \tag{2.18}$$

where

$$\dot{\psi} = \frac{\mu}{r^4} \frac{1}{\left(1/r + 2c + 6d\psi + 12e\psi^2 + 20f\psi^3 + 30g\psi^4\right)} \tag{2.19}$$

2.3.1.3 Time-Fixed Maneuvers in Relative Orbits

The final enabling technique employed in this research is similar to a Lambert solver in that it is used to generate the initial and final velocities which connect two position vectors in a specified time. This method, however, is applied to a chaser satellite in a relative orbit with a target satellite. The motion of the chaser is described in the RSW frame using the linearized equations of motion originally proposed by Hill [44] and Clohessy and Wiltshire [45], shown in Appendix B.

The initial and final relative positions of the target in the RSW frame, $\boldsymbol{r}_i$ and $\boldsymbol{r}_f$, respectively, are defined in Equations 2.20 and 2.21. The time-of-flight is t_{if} seconds.

$$\boldsymbol{r}_i = x_i\hat{R} + y_i\hat{S} + z_i\hat{W} \tag{2.20}$$

$$\boldsymbol{r}_f = x_f\hat{R} + y_f\hat{S} + z_f\hat{W} \tag{2.21}$$

Irvin et al. [46] described a technique to determine the scaled initial and final velocities of the chaser, $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_f$, respectively, given $\boldsymbol{r}_i$, $\boldsymbol{r}_f$, and t_{if}. The scaling is such that the time is scaled by the orbital period of the target satellite. That is, the scaled time $\tilde{T} = (n/2\pi)\, t_{if}$, where n is the mean motion of the target satellite. The position vectors are unaffected by this scaling. That is, $\tilde{\boldsymbol{r}}_i = \boldsymbol{r}_i$ and $\tilde{\boldsymbol{r}}_f = \boldsymbol{r}_f$.

Equations 2.22 and 2.23 show $\tilde{\boldsymbol{v}}_i$ and $\tilde{\boldsymbol{v}}_f$, respectively, as functions of $\tilde{\boldsymbol{r}}_i$, $\tilde{\boldsymbol{r}}_f$, and $\tilde{T}$. Other values in the equations are functions of known quantities: $\tilde{S} = \sin 2\pi\tilde{T}$, $\tilde{C} = \cos 2\pi\tilde{T}$, and $\Delta\tilde{y} = \tilde{y}_f - \tilde{y}_i$.

$$\begin{bmatrix} \dot{\tilde{x}}_i \\ \dot{\tilde{y}}_i \\ \dot{\tilde{z}}_i \end{bmatrix} = 2\pi \begin{bmatrix} \frac{-4\tilde{S}+6\pi\tilde{T}\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{4\tilde{S}-6\pi\tilde{T}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{-2+2\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} \\ \frac{-14+12\pi\tilde{T}\tilde{S}+14\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{2-2\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{\tilde{S}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} \\ 0 & -\frac{\tilde{C}}{\tilde{S}} & 0 & \frac{1}{\tilde{S}} & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_i \\ \tilde{z}_i \\ \tilde{x}_f \\ \tilde{z}_f \\ \Delta\tilde{y} \end{bmatrix} \tag{2.22}$$

$$\begin{bmatrix} \dot{\tilde{x}}_f \\ \dot{\tilde{y}}_f \\ \dot{\tilde{z}}_f \end{bmatrix} = 2\pi \begin{bmatrix} \frac{-4\tilde{S}+6\pi\tilde{T}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{4\tilde{S}-6\pi\tilde{T}\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{2-2\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} \\ \frac{2-2\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{-14+12\pi\tilde{T}\tilde{S}+14\tilde{C}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} & 0 & \frac{\tilde{S}}{8-6\pi\tilde{T}\tilde{S}-8\tilde{C}} \\ 0 & -\frac{1}{\tilde{S}} & 0 & \frac{\tilde{C}}{\tilde{S}} & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}_i \\ \tilde{z}_i \\ \tilde{x}_f \\ \tilde{z}_f \\ \Delta\tilde{y} \end{bmatrix} \tag{2.23}$$

Thus, it is possible to determine the velocities needed to connect two position vectors in the RSW frame given the time of flight between them.

2.3.2 Optimization Techniques

The purpose of this section is provide relevant background information on optimization techniques utilized to generate optimal trajectories. Betts [47] and more recently, Conway [7] authored surveys on state-of-the art numerical optimization techniques. Both provide detailed descriptions of several methods employed in the solution of optimal control problems, which are generally classified into three categories: direct methods, indirect methods, and evolutionary algorithms (EAs) or metaheuristics.

This section is divided into two parts. The first details more traditional numerical optimization techniques, namely direct and indirect methods. The second section describes a separate class of numerical optimization techniques known as EAs and metaheuristics.

2.3.2.1 Direct and Indirect Techniques

Indirect methods utilize the analytical necessary conditions derived from the COV, employed as both constraints and states. Specifically, additional states representing the costates, also known as Lagrange multipliers, of each state must be added, automatically doubling the size of the problem. Additional constraints resulting from the analytical necessary conditions must also be added to the problem constraints.

Betts [47] highlighted three primary drawbacks to applying indirect methods to solve trajectory optimization problems. These include the requirement to derive analytic necessary conditions for complicated dynamical systems, potentially small convergence regions, and the requirement to guess sub-arcs for problems requiring discrete variables (such as a series of thrust-coast sequences). Conway [7] notes an additional drawback, which is that the costates have no physical significance. This makes it very challenging to determine the magnitude or even the sign of the initial costate values required for the initial guess.

These challenges have resulted in the use of direct methods to optimize the majority of spacecraft trajectory optimization problems [7]. One such method is direct transcription. The direct transcription method converts a continuous optimal control problem into a large parameter optimization problem by discretizing the states and controls. The states and controls are defined at nodes and the system dynamics are satisfied using explicit or implicit integration [7] at each node. The states and controls are approximated linearly in between each node. This discretization can then be solved with a nonlinear programming (NLP) problem solver. A similar method called direct collocation discretizes the states and controls in the same fashion, however, they are approximated by higher-order polynomials rather than linearly.

There are several common collocation methods in which the primary differences are seen in the implicit integration rules. Of these methods, those employing Gauss-Lobatto or pseudospectral methods, also known as direct orthogonal collocation, [48, 49, 49, 50] provide significant benefit with respect to accuracy [51].

[7] states that direct transcription/collocation methods provide distinct advantages over indirect methods. The first benefit is that there is no need to derive the analytical necessary conditions, which can be problematic for realistic problems [51]. They are also robust to poor initial guesses.

Despite these benefits, direct transcription/collocation methods have two significant limitations. The first is that they require an initial guess, which can be difficult to generate

[51]. Additionally, these methods are likely to converge in the neighborhood of the initial guess, which implies they are likely to generate locally optimal solutions.

2.3.2.2 Evolutionary Algorithms and Metaheuristics

Metaheuristics and EAs are numerical optimization methods that define an optimization problem in a finite number of parameters. These methods are similar to one another because they do not require initial guesses, but rather randomly initialize populations throughout the solution space. EAs employ methods to preserve the fittest (most optimal) member of a population to serve as parents for subsequent generations. Metaheuristics use stochastic methods over several iterations to generate optimal solutions [7].

Metaheuristics and EAs have two distinct advantages over direct transcription/collocation methods. The first of these is that they do not require an initial guess. The second is that they are more likely, although not guaranteed, to converge to a globally optimal solution [7, 51].

In fact, Conway [7] specifically states that the best solution method "in almost all cases is that the best approach is an evolutionary algorithm or metaheuristic alone or in combination with a direct transcription method."

There are several different EAs and metaheuristics, and each uses different principles to generate optimal solutions. Two popular variants of metaheuristic and EA are particle swarm optimization (PSO) and genetic algorithm (GA), respectively. Both algorithms are utilized throughout this dissertation.

2.3.2.2.1 Particle Swarm Optimization The PSO algorithm is a specific type of metaheuristic utilized in this dissertation. PSO was initially developed by Eberhart and Kennedy [52, 53]. The algorithm and relevant research related to its performance is presented here.

Consider an unconstrained, n-dimensional optimization problem. The search space $\boldsymbol{S}$ of the problem is defined by the bounds on each variable. For example, the i^{th} design variable x^i has lower and upper limits x^i_{min} and x^i_{max}, respectively. The PSO is initialized by assigning each particle a position and velocity vector in $\boldsymbol{S}$ according to a uniform random distribution. The p^{th} particle's position $\boldsymbol{X}_p$ and velocity $\boldsymbol{V}_p$ vectors in $\boldsymbol{S}$ take the forms shown in Equation 2.24.

$$\begin{aligned} \boldsymbol{X}_p &= \left[x_p^1, x_p^2, \ldots, x_p^n\right] \\ \boldsymbol{V}_p &= \left[v_p^1, v_p^2, \ldots, v_p^n\right] \end{aligned} \tag{2.24}$$

The bounds on each component in $\boldsymbol{X}_p$ match the bounds in $\boldsymbol{S}$ corresponding to that component. That is, the i^{th} dimension of each particle's position vector is bounded by x^i_{min} and x^i_{max}. Similarly, the i^{th} dimension of each particle's velocity vector, v^i_p, is subject to an upper bound $v^i_{max} = x^i_{max} - x^i_{min}$ and a lower bound $v^i_{min} = -v^i_{max}$.

$$\begin{aligned} x^i_{min} &\leq x^i_p \leq x^i_{max} \\ v^i_{min} &\leq v^i_p \leq v^i_{max} \end{aligned} \tag{2.25}$$

The cost associated with each particle's position J_p is calculated at each iteration. The velocity of each particle is updated based on the particle's relative position in $\boldsymbol{S}$ to the best position visited by swarm ($\boldsymbol{g}_{best}$) and the best position ever visited by that specific particle ($\boldsymbol{p}_{best}$). Each particle's position in $\boldsymbol{S}$ is then updated by adding its new velocity to its current position.

The original implementation of PSO [52, 53] used the velocity update shown in Equation 2.26, where s is iteration number. The parameters c_1 and c_2 are the cognitive and social parameters, respectively. The cognitive parameter influences the velocity of each particle towards ($\boldsymbol{p}_{best}$) while the social parameter influences particle velocity towards ($\boldsymbol{g}_{best}$). The variables z_1 and z_1 are stochastic parameters uniformly distributed between zero and one.

$$\boldsymbol{V}_p(s) = \boldsymbol{V}_p(s-1) + c_1 z_1 \left(\boldsymbol{p}_{best} - \boldsymbol{X}_p(s-1)\right) + c_2 z_2 \left(\boldsymbol{g}_{best} - \boldsymbol{X}_p(s-1)\right) \tag{2.26}$$

If the i^{th} component of the velocity is outside the bounds defined in Equation 2.25, it is reset to the closest boundary. The position of each particle at the s^{th} iteration is updated according to Equation 2.27, regardless of the PSO variant.

$$\boldsymbol{X}_p(s) = \boldsymbol{X}_p(s-1) + \boldsymbol{V}_p(s) \tag{2.27}$$

Similarly, if the i^{th} component of the position is outside the bounds defined in Equation 2.25, it is reset to the closet boundary. This process is repeated until a specified convergence criteria is achieved or until a maximum number of iterations is reached.

Eberhart and Kennedys' initial research showed that the PSO algorithm described above (known as the global best particle swarm optimization variant (GBEST)) had a tendency to become trapped in local extrema. They developed the local best particle swarm optimization variant (LBEST) in order to mitigate this problem.

The velocity update for LBEST varies slightly from that of GBEST because each particle only shares information with its q adjacent neighbors on either side, where $2q$ is the neighborhood size. At each iteration, $J_p(s)$ is compared to the lowest cost ever achieved by any particle in its neighborhood, $J_{l_{best}}$, over the previous s iterations. If $J_p(s) < J_{l_{best}}$, then $J_{l_{best}}$ is set equal to $J_p(s)$ and the best position ever visited by any particle in the neighborhood $\boldsymbol{l}_{best}$ is set equal to $\boldsymbol{X}^p(s)$. The velocity update for the local PSO variant used in this research is shown in Equation 2.28.

$$\boldsymbol{V}_p(s) = \boldsymbol{V}_p(s-1) + c_1 z_1 \left(\boldsymbol{p}_{best} - \boldsymbol{X}_p(s-1)\right) + c_2 z_2 \left(\boldsymbol{l}_{best} - \boldsymbol{X}_p(s-1)\right) \tag{2.28}$$

Eberhart and Shi demonstrated success by setting the number of neighbors to 15% of the swarm size [54]. They compared the performance of GBEST and LBEST on several benchmark functions and found that LBEST is less susceptible than GBEST to local minima. This improved converge performance generally requires more iterations to converge, and thus greater computational time.

Later research on PSO focused on modifications to the velocity update equation. Shi and Eberhart [55] introduced the concept of an inertia weight w, which is meant to balance the global vs. local search capability of the PSO. The inertia weight is a multiplier of each particle's current velocity. The resulting velocity update equation takes the form shown in Equation 2.29

$$\boldsymbol{V}_p(s) = w\boldsymbol{V}_p(s-1) + c_1 z_1 \left(\boldsymbol{p}_{best} - \boldsymbol{X}_p(s-1)\right) + c_2 z_2 \left(\boldsymbol{g}_{best} - \boldsymbol{X}_p(s-1)\right) \tag{2.29}$$

[55] found that linearly decreasing the inertia weight as a function of the iteration number provided better performance than static inertia weights. This linear reduction allows for exploration of $\boldsymbol{S}$ at early iterations and exploitation of promising neighborhoods in $\boldsymbol{S}$ at later iterations.

[56, 57] introduced an additional parameter, called the constriction factor, into the velocity update equation. The constriction factor, χ is designed to prevent explosion, which occurs when the particles in the swarm tend toward the variable boundaries in $\boldsymbol{S}$. The constriction factor is defined in Equation 2.30, where $\phi = c_1 + c_2$

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{2.30}$$

The corresponding velocity update equation is shown in Equation 2.31.

$$\boldsymbol{V}_p(s) = \chi\left[\boldsymbol{V}_p(s-1) + c_1 z_1 \left(\boldsymbol{p}_{best} - \boldsymbol{X}_p(s-1)\right) + c_2 z_2 \left(\boldsymbol{g}_{best} - \boldsymbol{X}_p(s-1)\right)\right] \tag{2.31}$$

Eberhart and Shi [58] compared the performance of a PSO employing an inertia weight to that of a PSO employing a constriction factor on five benchmark problems. They discovered that the best approach is to use the constriction factor while defining a maximum velocity for each variable equal its dynamic range in the solution space.

Trelea investigated the effect of swarm size on convergence success for several benchmark functions. He found that convergence success increased as the number of particles increased, but mentions the trade off between number of particles and speed [59]. A swarm employing a larger number of particles more completely covers the solution space and is more likely to converge to the globally optimal solution. As swarm size increases, the number of cost functions evaluations per iteration also increases, resulting in slower computational performance. Zhang, Yu, and Hu investigated the effect of the swarm parameters and determined ϕ should be between 4.1 and 4.2 for high dimensional problems and 4.05 and 4.3 for lower dimensional problems [60]. They do not provide, however, a definition of lower and higher dimensional problems.

2.3.2.2.2 Genetic Algorithms The GA is an example of an EA and is used in this dissertation. Holland [61] originally developed the GA to model natural adaptive processes and later applied it to optimization problems. The GA begins with an initial population uniform randomly distributed throughout the solution space $\boldsymbol{S}$. The population in subsequent generations results from some combination of members of the previous generation, called parents. This is accomplished using two primary methods: selection and reproduction.

Selection determines which members of the current population will be chosen as parents for the next generation. It is a probabilistic method in which more optimal members are more likely to be chosen as parents. Talbi [62] highlighted several methods of selection such as roulette wheel selection, stochastic universal sampling, tournament selection and rank-based selection.

Roulette wheel, or proportionate, selection is the most common selection method used in GAs [62–64]. In this method, each member p of the population is assigned a fitness value based on the objective function value corresponding to that individual. The probability of that individual being selected as a parent for the next generation is proportional to the fitness value. That is, more fit individuals are assigned larger sections of the roulette wheel.

Roulette wheel selection is performed by randomly selecting a position on the roulette wheel, which corresponds to an individual in the population. This process is repeated Γ times to choose Γ parents for the next generation. This form of selection makes it more likely that individuals with better fitness values will be selected as parents. [62] noted two specific drawbacks to roulette wheel selection. The first is that it introduces bias towards strong performing individuals early in the algorithm which can cause convergence to local optima. Additionally, roulette wheel selection does not perform as well when all members of the population have similar fitness values.

An alternate selection method called stochastic universal sampling (SUS) is designed to reduce roulette wheel bias. Each individual in the population is assigned space on a roulette wheel proportional to the fitness value. The SUS method, however, is designed to choose all Γ parents with one spin of the wheel, so an additional wheel with Γ equally spaced pointers is placed around the the original wheel. When the wheel is stops, all Γ positions are chosen at once.

Another alternative is the tournament selection method, in which individuals are randomly chosen from the population to compete in a tournament against one another. The winner of the tournament is the individual with the best fitness value. A tournament can include all members of the population, but the standard tournament size is two members [62]. This process is repeated Γ times to choose Γ parents for the next generation.

Reproduction is accomplished via two operations called mutation and crossover. In mutation, a small change is made to one of the individuals retained via the selection process. There are many methods to accomplish mutation, but Talbi [62] lists three key principles that each method must meet. The first is ergodicity, which means that the mutation must provide the ability to reach all solutions in the search space. The second key principle is validity, meaning the mutation must produce valid solutions. The final principle is locality, which means the mutation must produce a small change.

The crossover operation is the second method of reproduction and is meant to combine pieces of one or more parent solutions preserved from the selection phase. Talbi [62] lists two key factors that must be considered when applying a crossover operator. The first of these is heritability, which means that each new solution should inherit characteristics from each parent solution. The second factor is validity.

The set of new solutions generated via the selection, mutation, and crossover operations is called a generation. Selection and reproduction are performed on the new generation and the process is repeated until a defined stopping criteria has been achieved. Examples of stopping criteria include a limit on the number of generations or a limit on the number of consecutive generations in which the lowest cost solution has not changed.

2.3.2.2.3 Other Evolutionary Algorithms There are several additional EAs seen throughout the literature. Price and Storn developed differential evolution (DE), which uses differences between solution vectors of the population to generate new vectors to search the solution space. This strategy is similar to GA in that it employs mutation and crossover, but the crossover operator is based on the distance between randomly chosen vectors and the parent vector. It has demonstrated a great deal of success in the solution of continuous optimization problems [62]. Ant colony optimization was originally proposed by Dorigo [65–68] to solve difficult combinatorial problems. Multiple authors have noted that ant colony optimization (ACO) has demonstrated success in solving several different types of optimization problems such as combinatorial, scheduling, routing, and assignment [62, 69].

2.3.2.2.4 Constrained Optimization with Evolutionary Algorithms The methods described above do not address methods to handle problem constraints, which can be classified into two categories: equality and inequality constraints. The purpose of this section is to describe research in constraint handling techniques relevant to EAs and metaheuristics.

Previous research indicated that EAs have difficulty handling equality constraints [70]. One common way to address this difficulty is to convert equality constraints into two inequality constraints by introducing an acceptable tolerance [70, 71].

Michalewicz and Schoenauer provided a background on techniques for handling constraints when using EAs [72]. They divided constraint handling techniques into four primary categories: methods based on preserving feasibility of solutions, methods based on penalty functions, methods which make a clear distinction between feasible and infeasible solutions, and hybrid methods.

Penalty functions are the most commonly used method to handle constraints in EAs and metaheuristics [72] and work by assigning an additional cost to any particle that violates the problem constraints.

The simplest penalty function is the death penalty method, which assigns an infinite cost to any solution that violates a constraint. It has been proven to be effective for several engineering problems [73, 74].

Joines and Houck introduced a dynamic penalty function in which the penalty increases as the iteration number increases [75]. A shortfall of the dynamic penalty method is that the algorithm has a tendency to become trapped in local optima due to the rapid growth of the penalty strength as iterations are increased [76].

The adaptive penalty function was originally developed by Bean and Hadj-Alouane [77, 78] and modifies the penalty function based on how long the best solution has been in/out of the feasible subspace. The adaptive penalty increases the penalty function if the fittest/best member of the population has not been in the feasible subspace for a finite number of consecutive iterations. It decreases the penalty function if the fittest/best member of the population has been in the feasible subspace for a finite number of consecutive iterations.

Despite the extensive research in the realm of constraint handling, there is no single method that is guaranteed to provide the best performance for all problems. Many authors have stated that penalty functions must be tuned to obtain the best results for each problem considered [72, 79, 80]. Penalty functions that are too large can cause premature convergence while penalties that aren't large enough allow solutions that violate constraints.

2.3.3 Spacecraft Trajectory Optimization Research

The field of spacecraft trajectory optimization is extensive. The purpose of this section is to provide the reader with a survey of current research employing the techniques utilized in this dissertation. Specifically, this section is divided into two pieces. The first provides a survey of spacecraft trajectory optimization research which utilized an EA or metaheuristic alone or in conjunction with a direct transcription methods employing an NLP problem solver. The second provides background on spacecraft trajectory optimization research in hybrid optimal control (HOC) problem, which consist of a combination of categorical and continuous variables.

2.3.3.1 Evolutionary Algorithms in Trajectory Optimization

The use of metaheuristics and EAs to solve spacecraft trajectory optimization problems has increased dramatically in recent years. The vast majority of research in the field has focused on finding optimal solutions to a variety of interplanetary trajectories and missions [8–25]. Several authors have also implemented heuristics to solve rendezvous and docking trajectory problems. Luo et al. applied a hybrid GA to solve a minimum-impulsive minimum-time rendezvous with constraints in the RSW frame [26]. Stupik et al. used a PSO to solve a continuous thrust minimax pursuit/evasion problem in the RSW frame where a target spacecraft is trying to maximize the rendezvous time as a pursuer spacecraft is trying to minimize the rendezvous time with the target [27].

Additional researchers studied different types of trajectory optimization problems using PSO. These include optimal impulsive transfers between several different orbit types [25, 81, 82], impulsive and finite thrust rendezvous trajectories [83], Lyapunov orbits around the Lagrange points in the Earth-Moon system [25, 84], lunar periodic orbits [25, 84], and orbit transfers using electric propulsion and a solar sail [85].

There is comparatively less research in optimal trajectory design for spacecraft in low Earth orbits with the purpose to achieve some effect or effects on the Earth's surface.

Guelman and Kogan implemented a maneuvering strategy to determine optimal trajectories that overfly a specified number of ground sites in a given time using electric propulsion [34]. Co et al. investigated the effects of propulsion method, orbit type, and thrust time on maximizing distance between a maneuvering satellite and a non-maneuvering reference satellite [35]. Abdelkhalik and Mortari implemented a GA to determine an optimal orbit to visit multiple ground sites in a specified time frame [32]. Kim et al. used a GA to find the optimal orbit to minimize average revisit time over a specific ground target in a finite number of days [33].

2.3.3.2 Hybrid Optimal Control

HOC problems consist of combinations of categorical variables and continuous variables. HOC algorithms are particularly interesting because they enable high level autonomous decision making and can be applied to a variety of real world engineering problems, which result from a mixture of logical decisions and continuous dynamics [86].

Recent research on the use of HOC in spacecraft trajectory optimization [28–31, 87, 88] has focused on bi-level HOC algorithms with multiple uses for the categorical variables. One use for the categorical variables is to select a planet to fly-by or an asteroid to rendezvous with [28–31]. A second use for the categorical variables is to define the number and sequence of the maneuvers to be performed [30, 31]. Finally, recent research has focused on using the categorical variables to determine the type of maneuvers to be performed, in addition to their number and sequence [87, 88]. In all cases, the structure defined by the categorical variables completely defines the inner-loop optimization problem.

Conway et al. [28] formulated an HOC problem in the solution of a three asteroid interception mission. A maneuvering spacecraft with impulsive-only thrust capability was required to intercept three of a possible eight asteroids with minimum fuel. The authors compared a bi-level algorithm with an outer-loop GA and an inner-loop method applying direct transcription with Runge-Kutta implicit integration (DTRK) to a bi-level algorithm employing a branch and bound (B&B) outer-loop and a GA inner-loop. Complete enumeration was used to determine the optimal sequence and cost. The GA-DTRK found the optimal solution while requiring only a fraction of the number of cost function evaluations required for complete enumeration of the problem space. The B&B-GA located similar solutions to those found by the GA-DTRK algorithm with even fewer cost function evaluations.

Wall and Conway [29] examined the low-thrust version of the minimum fuel asteroid rendezvous problem defined in [28]. The authors used a shape-based approximation to generate feasible low-thrust trajectories with defined boundary conditions. They compared the performance of a bi-level HOC algorithm with a B&B outer-loop solver coupled with a GA inner-loop to that of a GA outer-loop coupled with an inner-loop GA. Once the outer-loop algorithms terminated, the best trajectories found by each hybrid algorithm were used as initial guesses for a DTRK method. [29] implemented a bi-level GA-GA algorithm to solve a larger asteroid rendezvous in which a spacecraft must rendezvous with one asteroid in each of four groups of asteroids. Once again, the best solutions generated by the GA-GA algorithm with shape-based approximation were used as initial guesses for a more accurate DTRK method. The solutions found with the GA-GA algorithm very nearly approximated the optimal solutions identified by the DTRK and required significantly less computational time to generate.

Englander et al. [30] used a bi-level HOC algorithm to optimize interplanetary transfers with unknown locations, numbers, and sequences of en-route flybys. The outer-loop utilized a GA to determine the number, location, and sequence of fly-bys, while the inner-loop employed a combination of PSO and DE to optimize the variables corresponding to the sequences generated by the outer-loop. The authors applied this algorithm to three problems: an impulsive multi gravity assist (MGA) transfer from Earth to Jupiter, an impulsive MGA transfer from Earth to Saturn, and an impulsive multi gravity assist with deep space maneuvers (MGADSM) transfer from Earth to Saturn.

Englander et al. [31] extended the work of [30] by adding a capability to model low-thrust trajectories. They utilized a bi-level algorithm consisting of an outer-loop GA coupled with an inner-loop monotomic basin hopping (MBH) algorithm. The result from the MBH algorithm was used as an initial guess in the solution of a Sims-Flanagan transcription algorithm used to generate low-thrust trajectories. The authors applied this algorithm to generate optimal trajectories for an Earth to Jupiter transfer employing nuclear electric propulsion, an early proposal for the BepiColombo mission to Mercury, and a solar-electric mission from Earth to Uranus.

Chilan and Conway [87] introduced a new use for HOC in spacecraft trajectory optimization by using the categorical variables to define the number, types, and sequence of maneuvers to be performed between defined boundary conditions. They implemented a bi-level HOC algorithm with a GA outer-loop solver combined with a NLP inner-loop solver. The inner-loop solver was seeded with an initial guess using feasible region analysis and a conditional penalty (CP) method. They demonstrated the effectiveness of the algorithm by solving a minimum-fuel, time-fixed rendezvous between circular orbits originally posed by Prussing and Chui [89]. The algorithm proposed in [87] generated the optimal solution found by Colasurdo and Pastrone [90].

In a subsequent work, Chilan and Conway [88] used a bi-level HOC employing a GA outer-loop solver coupled with an NLP inner-loop solver which was seeded by a GA employing the CP method. They applied the algorithm to the time-fixed rendezvous problem posed by [89] and found a low-thrust trajectory which had a lower cost than, but was analogous to the best impulsive solution found by [90]. [88] applied the same bi-level HOC to find an optimal minimum fuel, free final time trajectory from Earth to Mars.

Yu et al. [91] developed a bi-level HOC algorithm to determine optimal trajectories for several variants of a GEO debris removal problem. They compared the performance of a simulated annealing (SA) outer solver coupled with a GA to that of an exhaustive search coupled with a GA to solve the inner-loop problem. Additionally, the authors developed a so-called Rapid Method for the outer-loop solver and found that it generated similar solutions to that of the SA outer-loop solver, but required much less computational time.

2.4 Summary

This chapter provided background information on research relevant to this dissertation, specifically on research in the field of spacecraft trajectory optimization. While the field is quite extensive, there is no current research on maneuvers which enable or enhance satellite resiliency. The purpose of this dissertation is to develop these types of maneuvers and investigate methods that facilitate their autonomous optimization. In particular, this dissertation will develop resiliency maneuvers which can be optimized using the methods covered in this literature review. Specifically, EAs and metaheuristics will be utilized in conjunction with Lambert targeting algorithms, shape-based trajectory approximation, NLP problem solvers, and bi-level HOC to produce optimal and near-optimal resiliency maneuvers.

III. Responsive Theater Maneuvers via Particle Swarm Optimization

3.1 Abstract

This research investigates the performance of the particle swarm optimization algorithm in the solution of responsive theater maneuvers, introduced here for the first time. The responsive theater maneuver is designed to alter a spacecraft's arrival position as it overflies a hazardous geographic region while still meeting sensor range constraints. The maneuver places the satellite on an exclusion ellipse centered at the spacecraft's expected arrival position at the expected time of entry into the hazardous region. A global particle swarm optimization algorithm is shown to generate optimal solutions for the single pass responsive theater maneuver scenario in shorter time frames than local particle swarm variants, a genetic algorithm, and a parameter search. The global particle swarm algorithm is then shown to generate consistent performance in the solution of single, double, and triple pass responsive theater maneuver scenarios for various size exclusion ellipses.

3.2 Nomenclature

a_e	=	semimajor axis of exclusion ellipse, *km*
b_e	=	semiminor axis of exclusion ellipse, *km*
c_1	=	swarm cognitive parameter
c_2	=	swarm social parameter
$\boldsymbol{g}_{best}$	=	global best position in the solution space
$\boldsymbol{g}_k$	=	unit vector perpendicular to $\boldsymbol{v}_k$ and $\boldsymbol{h}_k$ at k^{th} expected time of entry into exclusion zone
$\boldsymbol{h}_k$	=	expected angular momentum vector of satellite at k^{th} time of entry into exclusion zone, km^2/sec
J	=	cost of nonlinear function to be optimized
$J_{g_{best}}$, $J_{l_{best}}$, $J_{p_{best}}$	=	lowest cost associated with the swarm, neighborhood, and particle
$J_p(s)$	=	cost associated with a particle at the s^{th} iteration
$\boldsymbol{l}_{best}$	=	neighborhood best position in the solution space
m	=	number of particles in the swarm
n	=	number of design variables in the nonlinear function to be optimized
P	=	period of the initial orbit, *sec*
$\boldsymbol{p}_{best}$	=	particle best position in the solution space
R_{a_k}	=	orbit apogee radius after the k^{th} maneuver, *km*
R_e	=	distance from expected position of the spacecraft to the actual position of the spacecraft, *km*
R_{p_k}	=	orbit perigee radius after the k^{th} maneuver, *km*
R_{max}, R_{min}	=	maximum and minimum allowable orbital radius, *km*
$\boldsymbol{r}_k$	=	expected position vector of satellite at k^{th} time of entry into exclusion zone, *km*
$\boldsymbol{r}_k^*$	=	actual position vector of spacecraft at k^{th} time of entry into exclusion zone, *km*

$\boldsymbol{r}_{k_t^-}$	=	position vector at the instant just before the k^{th} impulse, *km*
$\boldsymbol{r}_0$	=	initial position vector, *km*
$\boldsymbol{S}$	=	Solution space encompassing all n design variables
T_k	=	time of flight of the k^{th} maneuver, *sec*
t_k	=	expected k^{th} time of entry into exclusion zone, *sec*
t_0	=	initial time, *sec*
$\boldsymbol{V}_p(s)$	=	n-dimensional velocity vector of the p^{th} particle at the s^{th} iteration
v^i_{max}, v^i_{min}	=	upper and lower bounds on the velocity of the i^{th} design variable
$\boldsymbol{v}_k$, $\boldsymbol{v}^*_k$	=	expected and actual velocity vector of satellite at k^{th} time of entry into exclusion zone, km/sec
$\boldsymbol{v}_{k_t^-}$, $\boldsymbol{v}_{k_t^+}$	=	velocity vectors at the instant just before and just after the k^{th} impulse, km/sec
$\boldsymbol{v}_0$	=	initial velocity vector, km/sec
$\boldsymbol{X}_p(s)$	=	n-dimensional position vector of the p^{th} particle at the s^{th} iteration
x^i_{max}, x^i_{min}	=	upper and lower bounds on the position of the i^{th} design variable
χ	=	swarm constriction factor
ϕ, λ	=	geocentric latitude and longitude, $^\circ$
θ_k	=	angle defining position of spacecraft on the k^{th} exclusion ellipse, *rad*
ν_{enter}	=	true anomaly of the spacecraft as it enters the latitude band of the exclusion zone
μ	=	Earth's gravitational parameter, km^3/sec^2
$\Delta\boldsymbol{V}_k$	=	velocity vector of the k^{th} maneuver, km/sec
ΔV_k	=	cost of the k^{th} maneuver, m/sec

3.3 Introduction

In recent years, the space domain has moved from an uncontested to a contested environment in which access to and the use of space can no longer be taken for granted. In light of this shifting paradigm, the United States Department of Defense (DoD) released a National Security Space Strategy (NSSS) in 2011 which promotes "cost-effective" spacecraft protection and resilience [2]. The NSSS defines resilience as "the ability of an architecture to support functions necessary for mission success in spite of adverse conditions. An architecture is more resilient if it can provide these functions with higher probability, shorter periods of reduced capability, and across a wider range of scenarios and conditions" [4].

Increased satellite maneuverability enhances resilience by enabling operation in hazardous conditions. A new set of maneuvers, introduced here as responsive theater maneuvers (RTMs), are proposed to enhance resilience for friendly space assets by introducing uncertainty while still meeting sensor range to collection target requirements.

3.4 Background

The field of optimal spacecraft trajectories is extensive and well researched. Conway [51] authored a survey of known solution methods as well as an overview of the most recent developments in the field of spacecraft trajectory optimization. According to [51], the critical limitation of many commonly used optimization techniques is the need for a suitable initial guess. Even when a suitable initial guess is provided, these techniques converge to a local optimal solution in the neighborhood of the guess. Conway specifically mentions the advantages of evolutionary algorithms because they don't suffer from these limitations and are more likely, albeit not guaranteed, to find the global optimal solution [51].

One such evolutionary algorithm is the particle swarm optimization (PSO) algorithm, initially developed by Eberhart and Kennedy [52, 53]. The swarm is initialized by randomly assigning each particle a position and velocity vector in the solution space. The costs associated with the positions of each particle are used to update the best position visited by swarm $\boldsymbol{g}_{best}$ and the best position ever visited by that specific particle $\boldsymbol{p}_{best}$. These values are then used to update each particle's velocity and position vectors for the next iteration. The process is repeated until a defined convergence criteria is met or a maximum number of iterations is reached.

Eberhart and Kennedys' initial research showed that the PSO algorithm described above (known as GBEST) had a tendency to become trapped in local extrema and they developed a different version (known as LBEST) in which each particle only had access to the best positions visited by its nearest neighbors [52]. Eberhart and Shi found that LBEST is less likely to converge to local minima than GBEST, but generally takes more iterations to converge [54].

Shi and Eberhart [55] introduced the concept of an inertia weight, which is meant to balance the global vs local search capability of the PSO. Clerc [56] and Clerc and Kennedy [57] introduced a constriction factor, which is designed to ensure the swarm converges rather than allowing particles to tend towards the boundaries of the solution space. Eberhart and Shi [58] compared the performance of a PSO using an inertia weight to that of a PSO using a constriction factor on five benchmark problems and discovered that the best approach is to use the constriction factor while defining a maximum velocity for each variable equal to its dynamic range in the solution space. Zhang et al. [60] investigated the effect of the constriction factor on particle swarm performance. They noted that the sum of the cognitive and social parameters should be between 4.1 and 4.2 for high dimensional problems and 4.05 and 4.3 for lower dimensional problems [60].

Penalty functions, which assign an additional cost to any particle that violates the constraints, are the most commonly used constraint handling technique. Authors have researched the effectiveness of different types of penalties including: static penalty methods

[79], dynamic penalty methods [75, 92], adaptive penalty methods [77, 78], and the death penalty method [73, 74]. Previous research has shown that penalty functions must be tuned to obtain the best results for each specific problem and the relative magnitude of the penalty must be considered in each case [72, 79, 80].

The use of metaheuristics/evolutionary algorithms to solve spacecraft trajectory optimization problems has increased dramatically in recent years. The vast majority of research in the field has focused on finding optimal solutions to a variety of interplanetary trajectories and missions [8–25]. Several authors have also implemented heuristics to solve rendezvous and docking trajectory problems. Luo et al. applied a hybrid genetic algorithm to solve a minimum-impulsive minimum-time rendezvous with constraints in the Clohessy-Wiltshire (CW) frame [26]. Stupik et al. used a PSO to solve a continuous thrust minimax pursuit/evasion problem in the CW frame where a target spacecraft is trying to maximize the rendezvous time as a pursuer spacecraft is trying to minimize the rendezvous time with the target [27].

Additional researchers studied different types of trajectory optimization problems using PSO. These include optimal impulsive transfers between several different orbit types [25, 81, 82], impulsive and finite thrust rendezvous trajectories [83], Lyapunov orbits around the Lagrange points in the Earth-Moon system [25, 84], lunar periodic orbits [25, 84], and orbit transfers using electric propulsion and a solar sail [85].

There is comparatively less research in optimal trajectory design for spacecraft in low Earth orbits with the purpose to achieve some effect or effects on the Earth's surface. Guelman and Kogan implemented a maneuvering strategy to determine optimal trajectories that overfly a specified number of ground sites in a given time using electric propulsion [34]. Co et al. investigated the effects of propulsion method, orbit type, and thrust time on maximizing distance between a maneuvering satellite and a non-maneuvering reference satellite [35]. Abdelkhalik and Mortari implemented a genetic algorithm (GA) to determine an optimal orbit to visit multiple ground sites in a specified time frame [32]. Kim et al. used a GA to find the optimal orbit to minimize average revisit time over a specific ground target in a finite number of days [33].

The purpose of this research is to extend the field of spacecraft trajectory optimization problems delivering ground effects to include maneuvers which enhance resiliency for satellites operating over potentially hazardous regions. RTM are designed to enhance resiliency by altering a spacecraft's arrival position from its predicted position as it enters a specified geographic region.

3.5 Methodology

Each pass over the specified geographic region k of the RTM problem has two design variables corresponding to the optimal departure and arrival location of the maneuver resulting in a total of $n = 2k$ design variables. The acceptable bounds on each design variable define the solution space $\boldsymbol{S}$ and the total cost of the maneuver J is the sum of the cost of the maneuvers required for each pass.

The PSO developed below is based on the work of several previous authors [25, 55–57, 81, 84, 93]. It has a total of m particles and each particle's position $\boldsymbol{X}_p$ and velocity $\boldsymbol{V}_p$ in $\boldsymbol{S}$ are n-dimensional vectors where the i^{th} dimension of each vector corresponds to the i^{th} design variable:

$$\begin{aligned} \boldsymbol{X}_p &= \left[x_p^1, x_p^2, \ldots, x_p^n\right] \\ \boldsymbol{V}_p &= \left[v_p^1, v_p^2, \ldots, v_p^n\right] \end{aligned} \tag{3.1}$$

The i^{th} dimension of each particle's position vector x_p^i is bounded by the lower and upper limits of the i^{th} design variable x_{min}^i and x_{max}^i respectively. Similarly, the i^{th} dimension of each particle's velocity vector v_p^i is subject to an upper bound $v_{max}^i = x_{max}^i - x_{min}^i$ and a lower bound $v_{min}^i = -v_{max}^i$:

$$\begin{aligned} x_{min}^i &\leq x_p^i \leq x_{max}^i \\ v_{min}^i &\leq v_p^i \leq v_{max}^i \end{aligned} \tag{3.2}$$

The swarm is initialized such that each particle's position and velocity is uniformly randomized in the solution space defined by these bounds. The cost associated with the position of each particle $J_p(s)$ is evaluated at each iteration s along with the constraints. If any of the constraints are violated, then $J_p(s)$ is set equal to infinity. If $J_p(s)$ is less than the lowest cost associated with the particle over the previous $s-1$ iterations $\left(J_{p_{best}}\right)$, then $J_{p_{best}}$ is set equal to $J_p(s)$ and the best position ever visited by the particle $\boldsymbol{p}_{best}$ is updated to the current particle position $\boldsymbol{X}_p(s)$.

The velocity of each particle at the s^{th} iteration $\boldsymbol{V}_p(s)$ is a function of the position and velocity of that particle at the previous iteration, as well as $\boldsymbol{p}_{best}$. The velocity update for the global version of the PSO is also dependent on $\boldsymbol{g}_{best}$, which is the best position visited by the swarm so far. The velocity update equation for the global PSO algorithm used for the purposes of this research is shown in Equation 3.4, where c_1 is the cognitive parameter, c_2 is the social parameter, and

$$\chi = \frac{2}{|2-\phi-\sqrt{\phi^2-4\phi}|} \tag{3.3}$$

is the constriction factor with $\phi = c_1 + c_2$. Additionally, z_1 and z_2 are distinct uniformly distributed random numbers between zero and one:

$$\boldsymbol{V}_p(s) = \chi\left[\boldsymbol{V}_p(s-1) + c_1 z_1\left(\boldsymbol{p}_{best} - \boldsymbol{X}_p(s-1)\right) + c_2 z_2\left(\boldsymbol{g}_{best} - \boldsymbol{X}_p(s-1)\right)\right] \tag{3.4}$$

The velocity update for the local version of the PSO varies slightly from the global version because each particle only shares information with its q adjacent neighbors on either side, where $2q$ is the neighborhood size. At the s^{th} iteration, $J_p(s)$ is compared to the lowest cost ever achieved by any particle in its neighborhood $J_{l_{best}}$ over the previous iterations. If $J_p(s) < J_{l_{best}}$, then $J_{l_{best}}$ is set equal to $J_p(s)$ and the best position ever visited by any particle in the neighborhood $\boldsymbol{l}_{best}$ is set equal to $\boldsymbol{X}_p(s)$. The velocity update for the local PSO variant used in this research is shown in Equation 3.5:

$$\boldsymbol{V}_p(s) = \chi\left[\boldsymbol{V}_p(s-1) + c_1 r_1\left(\boldsymbol{p}_{best} - \boldsymbol{X}_p(s-1)\right) + c_2 r_2\left(\boldsymbol{l}_{best} - \boldsymbol{X}_p(s-1)\right)\right] \tag{3.5}$$

If the i^{th} component of the velocity is outside the bounds defined in Equation 3.2, it is reset to the closest boundary. The position of each particle at the s^{th} iteration is updated according to Equation 3.6, regardless of the PSO variant:

$$\boldsymbol{X}_p(s) = \boldsymbol{X}_p(s-1) + \boldsymbol{V}_p(s) \tag{3.6}$$

Similarly, any component of $\boldsymbol{X}_p$ outside the bounds defined in Equation 3.2 is reset to the nearest boundary. This process is repeated until a specified convergence criteria is achieved or until a maximum number of iterations is reached.

3.6 Responsive Theater Maneuvers

RTMs require a maneuver in order to increase the unpredictability of a spacecraft as it flies over a hazardous geographic region on the Earth, called the exclusion zone and defined by latitude (ϕ_{min}, ϕ_{max}) and longitude $(\lambda_{min}, \lambda_{max})$ bands. These maneuvers are constrained such that the spacecraft must arrive on an exclusion ellipse at its expected time of entry into the exclusion zone.

3.6.1 Single Pass Maneuvers

The satellite begins in an Earth orbit at initial time t_0 with Earth-centered, inertial position and velocity vectors, $\boldsymbol{r}_0$ and $\boldsymbol{v}_0$, respectively. Additionally, the Earth is assumed to be a perfect sphere and the spacecraft is subject only to two-body Keplerian forces. As a result, the geocentric longitude λ and latitude ϕ can be computed at any time t using the current position vector and the Greenwich Mean Time (GMT). For simplicity, GMT at t_0 is assumed zero.

The expected satellite entry state into the exclusion zone state consists of the time of entry t_1, the position vector at entry $\boldsymbol{r}_1$, and the velocity vector at entry $\boldsymbol{v}_1$. The two-body and spherical Earth assumptions make it possible to analytically determine the true anomaly of the spacecraft ν_{enter} as it enters the exclusion zone latitude band. Let ζ be the argument of latitude corresponding to the point at which the latitude band defined by

(ϕ_{min}, ϕ_{max}) is entered. If $0 < \phi_{min} < \phi_{max} < \xi < \pi/2$ (where ξ denotes the orbit inclination) and $0 < \zeta < \pi/2$, then

$$\sin\zeta = \frac{\sin\phi_{min}}{\sin\xi} \tag{3.7}$$

and the true anomaly ν_{enter} is given by

$$\nu_{enter} = \zeta - \omega \tag{3.8}$$

The true anomaly ν_{enter} corresponds to the inertial position and velocity vectors denoted with $\boldsymbol{r}_{enter}$ and $\boldsymbol{v}_{enter}$, respectively. Equations (5.10) and (3.8) are to be modified if the previously reported inequalities are not satisfied, if the exclusion latitude band is entered while the spacecraft is traveling toward the equatorial plane (i.e. when $\pi/2 < \zeta < \pi$), or in the presence of a retrograde orbit. Once ν_{enter} has been obtained, the first time at which the satellite enters the latitude band t_{enter} is found from the solution of Kepler's equation, under the assumption that the true anomaly at t_0 is known.

All subsequent entries into the latitude band occur one orbital period after the previous entry. Further, the longitude of the spacecraft at t_{enter} is found using the entry time and the inertial position vector corresponding to ν_{enter}. A similar process is used to determine the true anomaly ν_{exit}, time t_{exit}, and the longitude of the spacecraft when it exits the latitude band.

The spacecraft enters the exclusion zone between t_{enter} and t_{exit} in two instances. The first occurs if $\lambda_{min} \leq \lambda_{enter} \leq \lambda_{max}$ and implies that the true anomaly upon entry into the exclusion zone, ν_1, is equal to ν_{enter}. The second case occurs when $\lambda_{enter} < \lambda_{min}$ and $\lambda_{min} < \lambda_{exit}$. This scenario implies $\nu_{enter} < \nu_1 < \nu_{exit}$ and requires interpolation to determine ν_1. The satellite's expected entry state into the exclusion zone can be found from ν_1.

The expected specific angular momentum vector of the orbit $\boldsymbol{h}_1$ is defined by $\boldsymbol{r}_1$ and $\boldsymbol{v}_1$:

$$\boldsymbol{h}_1 = \boldsymbol{r}_1 \times \boldsymbol{v}_1 \tag{3.9}$$

Additionally, a unit vector perpendicular to $\boldsymbol{v}_1$ and $\boldsymbol{h}_1$ is defined as

$$\boldsymbol{g}_1 = \frac{\boldsymbol{v}_1 \times \boldsymbol{h}_1}{|\boldsymbol{v}_1||\boldsymbol{h}_1|} \tag{3.10}$$

The exclusion zone is defined by an ellipse with semimajor axis a_e and semiminor axis b_e. It is centered at $\boldsymbol{r}_1$ and oriented such that a_e is aligned with $\boldsymbol{v}_1$. The satellite must arrive at some point on the exclusion ellipse rather than $\boldsymbol{r}_1$ at time t_1. The first variable θ_1, is an angle which defines the satellite's location on the exclusion ellipse and is measured from $\boldsymbol{v}_1$ in the direction of $\boldsymbol{g}_1$. The distance from the ellipse center to any point on the ellipse is defined by a_e, b_e, and θ_1, as shown in Equation (4.3):

$$R_e = \frac{a_e b_e}{\sqrt{b_e^2 \cos^2 \theta_1 + a_e^2 \sin^2 \theta_1}} \tag{3.11}$$

The position where the intercept will take place on the ellipse is then defined in the inertial frame as shown in Equation (4.2):

$$\boldsymbol{r}_1^* = \boldsymbol{r}_1 + R_e \cos \theta_1 \frac{\boldsymbol{v}_1}{|\boldsymbol{v}_1|} + R_e \sin \theta_1 \boldsymbol{g}_1 \tag{3.12}$$

A second variable T_1 defines how many seconds in advance of t_1 the satellite will perform an impulsive maneuver that will deliver it to $\boldsymbol{r}_1^*$ at t_1. It is assumed that T_1 must be less than or equal to one orbital period of the initial orbit and greater than 1200 seconds to allow the spacecraft time to prepare for data collection as it passes over the exclusion zone. The position and velocity vectors at the instant before the maneuver are $\boldsymbol{r}_{1_t^-}$ and $\boldsymbol{v}_{1_t^-}$, respectively. The orbital geometry is depicted in Figure 3.1.

The velocity vector of the maneuver that will take the spacecraft from the state defined by $\boldsymbol{r}_{1_t^-}$ and $\boldsymbol{v}_{1_t^-}$ to $\boldsymbol{r}_1^*$ in T_1 s is $\Delta \boldsymbol{V}_1$, and are found by solving the well known Lambert's problem.

The new orbit must have an apogee radius R_{a_1} less than or equal to some maximum radius R_{max} as well as a perigee radius R_{p_1} greater than or equal to some minimum radius

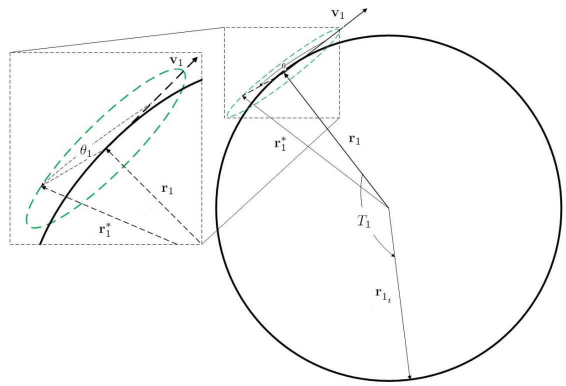


Figure 3.1: Single pass RTM intercept geometry

R_{min} in order for the spacecraft to perform adequate data collection for the duration of its mission.

3.6.2 Mulitple-Pass Maneuvers

The solution method for the single pass RTM problem can be extended to optimize an n-pass RTM problem over the exclusion zone by reinitializing the initial conditions after each maneuver, but the number of optimization variables increases by two for each pass over the exclusion zone.

Consider a double pass RTM problem. The algorithm begins with initial conditions $(\boldsymbol{r}_0, \boldsymbol{v}_0, t_0)$ and determines the arrival state into the exclusion zone defined by t_1, $\boldsymbol{r}_1$, and $\boldsymbol{v}_1$. The variables T_1 and θ_1 determine the cost of the first maneuver ΔV_1. They also define the post maneuver position and velocity vectors of the spacecraft at t_1, $\boldsymbol{r}_1^*$ and $\boldsymbol{v}_1^*$, respectively, which become the initial conditions for the second pass over the exclusion zone. The algorithm identifies the second time the spacecraft will fly over the exclusion zone t_2, as well as the expected position and velocity vectors upon arrival $\boldsymbol{r}_2$ and $\boldsymbol{v}_2$, respectively. The variable T_2 determines the time of flight needed to make the maneuver, and the variable θ_2 determines the spacecraft's intercept point on the exclusion ellipse. This information can then be used to determine the cost of the second maneuver ΔV_2. A double pass maneuver has four design variables: T_1, θ_1, T_2, and θ_2 with a cost $J = \Delta V_1 + \Delta V_2$. This process can be extended to n pass maneuvers as needed.

3.7 Numerical Results

3.7.1 Comparison of Optimization Tools for Single Pass RTM Problem

The first objective of this research was to identify the most efficient method to optimize RTM problems. The single pass RTM problem was used as a test function to determine the effectiveness and efficiency of PSO algorithms in comparison to a genetic algorithm and a simple parameter search. Additionally, this problem was used to identify a concept of operations for employing evolutionary algorithms to generate optimal solutions for RTM scenarios. Ten algorithms (four global global PSO (PSOG) of varying swarm size, four local PSO (PSOL) algorithms with varying swarm size, the genetic algorithm toolbox in MATLAB, and a simple parameter search) were used to solve the single pass RTM problem shown in Equation (4.7), where P is the period of the initial orbit. The parameter search was performed in increments of 0.5 s and 0.001 rad in order to generate results with the same fidelity as seen in the evolutionary algorithms:

$$\boxed{\begin{array}{ll} \text{minimize J} = \Delta V_1 \text{ m/s} & \\ \text{subject to:} & \\ \boldsymbol{r}_0 = [6800\ 0\ 0] \text{ km} & \boldsymbol{v}_0 = [0\ 5.41377\ 5.41377] \text{ km/sec} \\ (\phi_{min}, \phi_{max}) = (-10^\circ, 10^\circ) & (\lambda_{min}, \lambda_{max}) = (-50^\circ, -10^\circ) \\ a_e = 150 \text{ km},\ b_e = 0.1 a_e & \\ 1200 \text{ s} \leq T_1 \leq P & 0 \leq \theta_1 \leq 2\pi \\ R_{a_1} \leq 6850 \text{ km} & 6750 \text{ km} \leq R_{p_1} \end{array}} \tag{3.13}$$

The parameter search was used to identify the global optimal solution and to measure the convergence success of the evolutionary algorithms for the single pass RTM due to its two-dimensional nature. The global optimal solution for the single-pass RTM problem with $a_e = 150$ km and $b_e = 15$ km is as follows: $T_1 = 2877$ s, $\theta_1 = 5.906$ rad, and

J = 4.08255 m/s, and run time = 6934.03 s. The three-dimensional response surface is shown in Figure 3.2. Note that there are two distinct troughs in the response surface, one of which corresponds to the previously mentioned global minimum, and another which corresponds to a local minimum approximately 0.04 m/s greater than the global optimal solution. The shape of the response surface illustrates that a poor initial guess would make it impossible to determine the global optimal solution using analytical gradient-based methods.

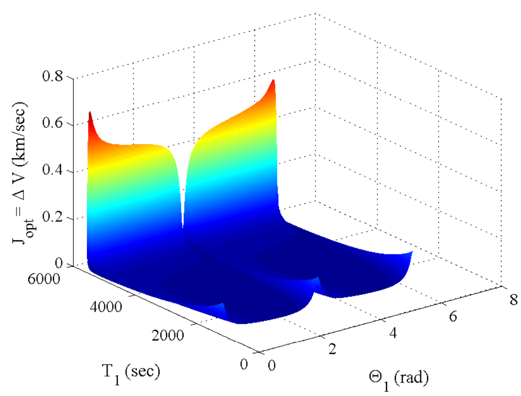


Figure 3.2: Response surface for single pass RTM with $a_e = 150\ km$ and $b_e = 15\ km$

All eight PSO algorithms were implemented with identical cognitive and social parameters ($c_1 = c_2 = 2.1$). The global versions of the PSO algorithm employed a stopping condition that terminated the algorithm when the best cost of each individual particle $J_{p_{best}}$ was within $1e^{-10}$ km/s of the lowest cost of the swarm $J_{g_{best}}$. The local versions of the PSO terminated in the same circumstances as the global versions, and also if 75% of the particles' costs were within $1e^{-10}$ km/s after 1000 iterations. The maximum number of iterations for all PSO variants was capped at 7000. The genetic algorithm used a population size of 50 and a crossover rate of 0.8. Selection was accomplished via stochastic uniform selection and five elite members of each generation were automatically selected for the next generation. Additionally, each member of the first generation was reinitialized until it satisfied the constraints. The maximum allowable generations parameter was set to 2000. We did not investigate the ideal parameter settings for the GA; it is only presented here to demonstrate that it produces similar results to the PSO variants. Each evolutionary algorithm was parallelized on a machine with a six core, 2.9 GHz processor and run 20 times. The following data were collected to measure performance: cost, iterations/generations [minimum (min); maximum(max); average (avg)] required for convergence, and the run time required for convergence. The performance of each algorithm is shown in Table 3.1.

Table 3.1: Comparison of optimization algorithms in single pass RTM problem

		Neighborhood	J (m/sec)		Iterations/Generations			Run Time (sec)			Convergence	
Method	Pop Size	Size	Min	Max	Min	Max	Avg	Min	Max	Avg	Global	Local
PSOG	30	–	4.08254	4.12261	84	979	204.05	4.93	56.54	12.48	30%	70%
PSOG	60	–	4.08254	4.12261	107	433	201.45	6.88	27.91	13.04	70%	30%
PSOG	100	–	4.08254	4.12261	102	722	266.70	7.00	49.17	18.20	80%	20%
PSOG	120	–	4.08254	4.12261	88	1649	319.05	6.43	119.45	23.25	90%	10%
PSOL	30	4	4.08254	4.12302	273	7000	3650.10	17.96	504.32	241.69	65%	15%
PSOL	60	8	4.08254	4.12261	235	7000	4181.70	20.34	690.56	379.58	75%	5%
PSOL	100	14	4.08254	4.08256	384	7000	4160.55	40.77	855.51	474.56	95%	–
PSOL	120	18	4.08254	4.08255	322	7000	2435.7	46.13	971.76	329.11	95%	–
GA	50	–	4.08254	4.12261	17	17	17	45.90	224.81	97.43	55%	15%

The PSOG with 30 particles had the fastest average convergence time, but also demonstrated the lowest global convergence rate. The PSOL variants with sufficient size provided the best global convergence rate and avoided the local minimum solution to which all other algorithms converged at least once. This success, however, came with a significant penalty in solution time relative to the PSOG variants of similar size. Both the PSOL variants and the GA were significantly slower than all PSOG in terms of average run time. The faster convergence for smaller swarm sizes, coupled with their convergence to the global minimum over the course of 20 runs, led to the conclusion that it is more efficient to run the PSOG several times than to run other algorithms that provide more consistent performance but take much longer to generate a solution. It is important to note that the GA was essentially an off-the-shelf model that was not tuned or studied to the extent of the PSO variants. Further investigation should indicate that the performance of the GA could be improved for this problem.

3.7.2 Single Pass Results

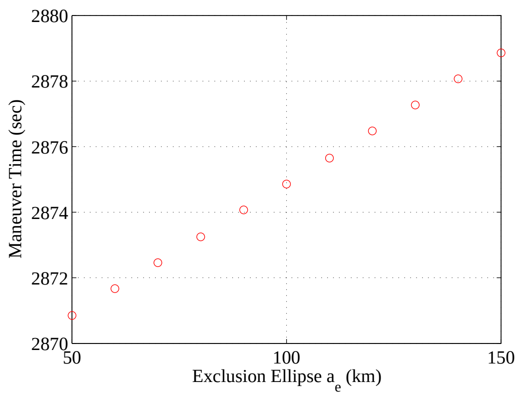
The PSOG variant with 30 particles was used to solve several cases of the single pass RTM problem using the same initial conditions described in Equation (4.7) as well as for a circular orbit with $\boldsymbol{r}_0 = [7300\ 0\ 0]$ km and $\boldsymbol{v}_0 = [0\ 5.22507\ 5.22507]$ km/s. The exclusion zone for all cases was defined as $(\phi_{min}, \phi_{max}) = (-10^\circ, 10^\circ)$ and $(\lambda_{min}, \lambda_{max}) = (-50^\circ, -10^\circ)$. The size of the exclusion ellipse semimajor axis ranged from 50 km to 150 km in increments of 10 km, with $b_e = 0.1a_e$. The constraints were defined such that $R_{max} = r_0 + 50$ km and $R_{min} = r_0 - 50$ km. Each case was run 20 times using the same workstation described in the previous section.

In all cases, the PSO converged to two distinct solutions. The difference between the lowest cost solutions and the local minimum solutions increased with increasing exclusion ellipse size, with a maximum of 0.040 m/s for $r_0 = 6800$ km and a maximum of 0.034 m/s for $r_0 = 7300$ km. These differences are negligible when considering the control capability of real world thrusters. The lowest costs found by the PSO are shown in Table 3.2 in units of m/s. Figures 3.3(a) and 3.3(b) show the maneuver times (T_1) and arrival locations (θ_1) of the lowest cost solutions as functions of exclusion ellipse size for the case with $r_0 = 6800$ km. These figures are representative of the results seen for $r_0 = 7300$ km.

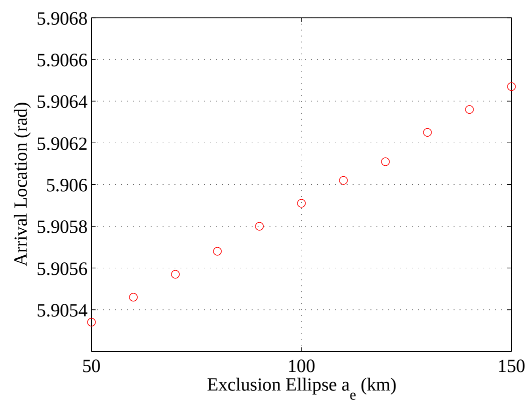
Figure 3.3(b) shows the optimal arrival location on the exclusion ellipse is always larger than π rad, which implies that the spacecraft arrival location over the exclusion zone is lower in altitude than the expected arrival location. The reduction in altitude results in

Table 3.2: Optimal cost of single pass RTM problem for varying exclusion ellipse sizes

		a_e/b_e (km)										
r_0, km	(m/sec)	50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
6800	ΔV	1.365	1.638	1.910	2.182	2.454	2.726	2.998	3.269	3.541	3.812	4.083
7300	ΔV	1.228	1.473	1.718	1.962	2.207	2.451	2.696	2.940	3.184	3.428	3.672



(a) T_1 as a function of exclusion ellipse size



(b) θ_1 as a function of exclusion ellipse size

Figure 3.3: Optimal design variables and constraints for single pass RTM as functions of exclusion ellipse size (r_0 = 6800 km)

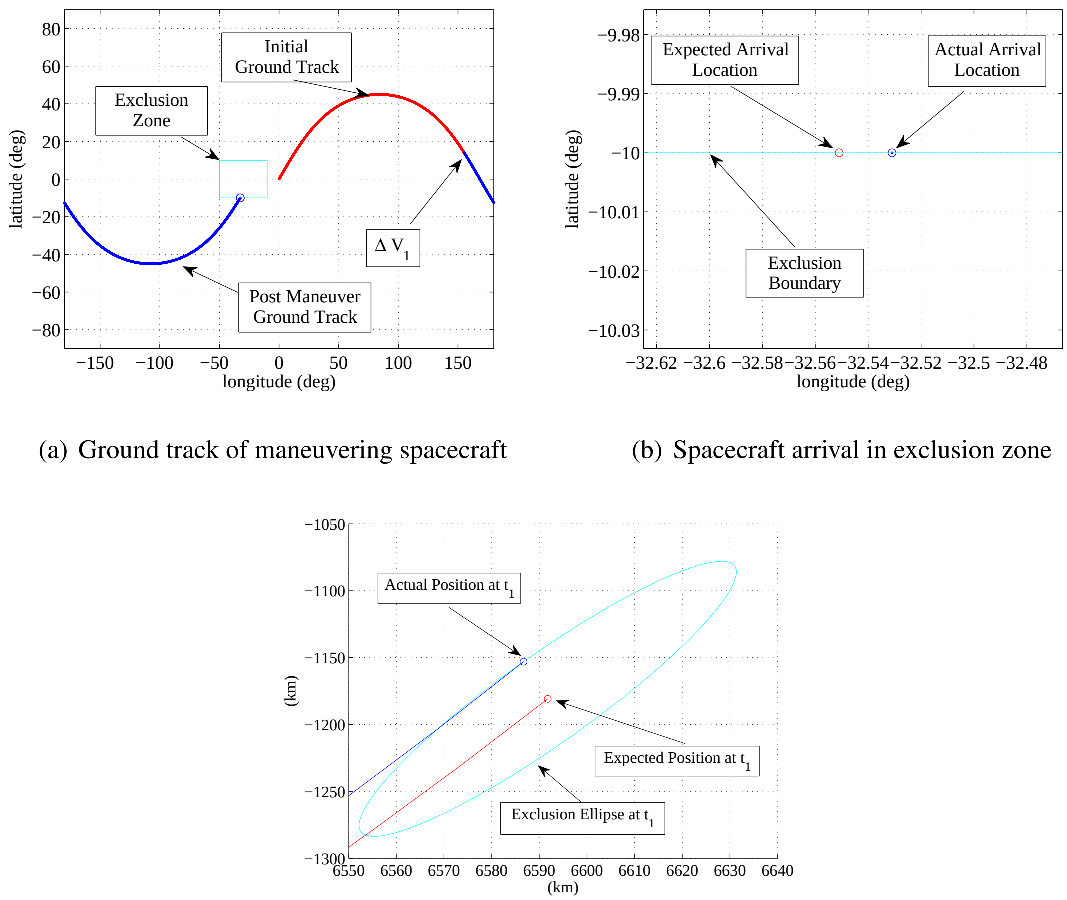
an earlier arrival over the exclusion zone because a decrease in altitude corresponds to an increased angular rate around the Earth.

The cost associated with the RTM increases with increasing exclusion ellipse size. Unexpectedly, the cost increases proportionally to the size of the exclusion ellipse. Equation 3.14 provides a method for estimating the cost associated with maneuvering to exclusion ellipse sizes not investigated in this paper. This relationship is accurate within 0.0088 m/s for r_0 = 6800 km and 0.0098 m/s for r_0 = 7300 km.

$$\frac{a_{e_1}}{a_{e_2}} \approx \frac{\Delta V_1}{\Delta V_2} \tag{3.14}$$

Another important result is the relative speed of the PSOG for the single pass RTM problem. The algorithm completed 20 runs in less than five min for all cases considered with an average time of completion of 201.43 s. Table 3.5 in the Appendix shows the run time required for all 20 runs of each case as well as the global convergence rate.

Figure 3.4 shows the ground track, predicted/actual entry locations, and the exclusion ellipse for the single pass RTM problem with $r_0 = 6800$ km and $a_e = 150$ km. These results are representative of those seen in the other single pass RTM with varying size exclusion ellipses.



(a) Ground track of maneuvering spacecraft

(b) Spacecraft arrival in exclusion zone

(c) Spacecraft arrival on exclusion ellipse

Figure 3.4: Optimal solution for single pass RTM maneuver with $a_e = 150$ km, $b_e = 15$ km

3.7.3 *n-Pass RTM*

3.7.3.1 *Double Pass RTM*

The PSOG with 30 particles was used to solve multiple double pass RTM problems using the same initial orbits investigated in the single pass RTM. The exclusion zone, exclusion ellipses, and apogee/perigee constraints also remained the same as those investigated in the single pass RTM problems. A summary of the double pass RTM is shown in Equation 3.15:

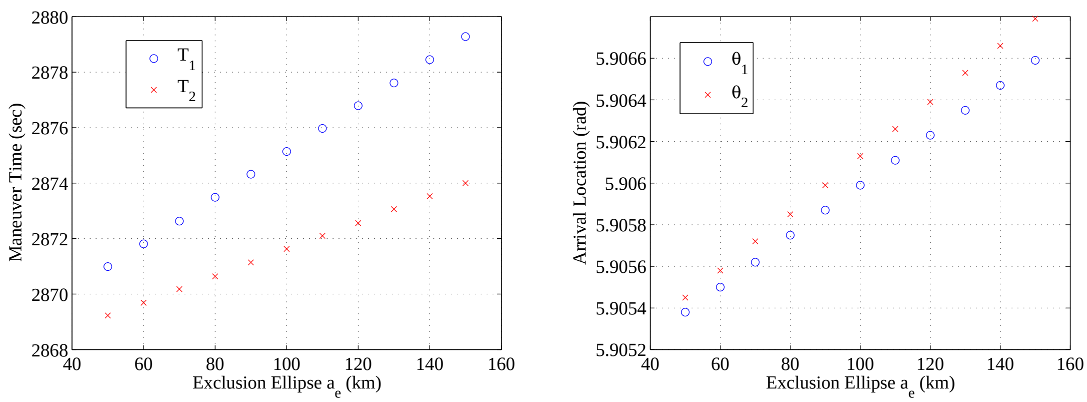
$$\boxed{\begin{array}{ll} \text{minimize } J = \Delta V_1 + \Delta V_2 \text{ m/s} & \\ \text{subject to:} & \\ (\phi_{min}, \phi_{max}) = (-10^\circ, 10^\circ) & (\lambda_{min}, \lambda_{max}) = (-50^\circ, -10^\circ) \\ 1200 \text{ s} \le T_1, T_2 \le P & 0 \le \theta_1, \theta_2 \le 2\pi \\ R_{a_1}, R_{a_2} \le r_0 + 50 \text{ km} & r_0 - 50 \text{ km} \le R_{p_1}, R_{p_2} \end{array}} \tag{3.15}$$

The lowest cost solution obtained by the PSO over the course of 20 runs is here referred to as the optimal solution (given that there is no analytical solution). Table 3.3 shows the lowest cost found by the PSO over the course of 20 runs as a function of varying exclusion ellipse size. The associated design variables for each case can be seen in Table 3.6 of the Appendix. Figures 3.5(a) and 3.5(b) show the optimal maneuver times and arrival locations on the exclusion ellipse.

The results seen for the double pass RTM problems are very similar to those seen for the single pass cases. The average time required to execute 20 runs was 235.70 s. The PSO converged to one of four solutions for each case considered. The maneuver times and arrival locations on the exclusion ellipses for each pass are nearly the same as those seen in the single pass cases. Once again, the difference between the global and local solutions increased with increasing exclusion ellipse size with a maximum of 0.059 m/s for $r_0 = 6800$ km and 0.050 m/s for $r_0 = 7300$ km. Similar to the single pass cases, the

Table 3.3: Optimal cost of double pass RTM problem for varying exclusion ellipse sizes

		a_e/b_e (km)										
r_0, km		50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
	ΔV_1	1.365	1.638	1.910	2.182	2.454	2.726	2.998	3.269	3.541	3.812	4.083
6800	ΔV_2	1.366	1.640	1.912	2.185	2.458	2.731	3.003	3.276	3.548	3.821	4.093
	J	2.732	3.278	3.823	4.368	4.912	5.457	6.001	6.545	7.089	7.633	8.176
	ΔV_1	1.228	1.473	1.718	1.962	2.207	2.451	2.696	2.940	3.184	3.428	3.672
7300	ΔV_2	1.229	1.474	1.720	1.965	2.210	2.455	2.701	2.946	3.191	3.435	3.680
	J	2.457	2.947	3.438	3.927	4.417	4.907	5.396	5.886	6.375	6.864	7.352



(a) T_1 and T_2 as functions of exclusion ellipse size (b) θ_1 and θ_2 as functions of exclusion ellipse size

Figure 3.5: Optimal design variables and constraints for double pass RTM as functions of exclusion ellipse size ($r_0 = 6800$ km)

lowest cost solutions required the spacecraft to arrive over the exclusion zone with a lower altitude and in advance of its expected arrival time for each pass, regardless of exclusion ellipse size.

In each case, the optimal first maneuver nearly (but not exactly) matches that seen in the single pass RTM for exclusion ellipses of the same size. Additionally, the second maneuver is very similar to the first in terms of the time of flight needed to complete the maneuver (T_1 and T_2) and the intercept location (θ_1 and θ_2) on the exclusion ellipse, but always requires more fuel to execute. Equation 3.14 is once again an accurate predictor of maneuver cost, with a maximum difference between the predicted and actual cost of 0.0072 m/s for r_0 = 6800 km and 0.0068 m/s for r_0 = 7300 km.

3.7.3.2 Triple Pass RTM

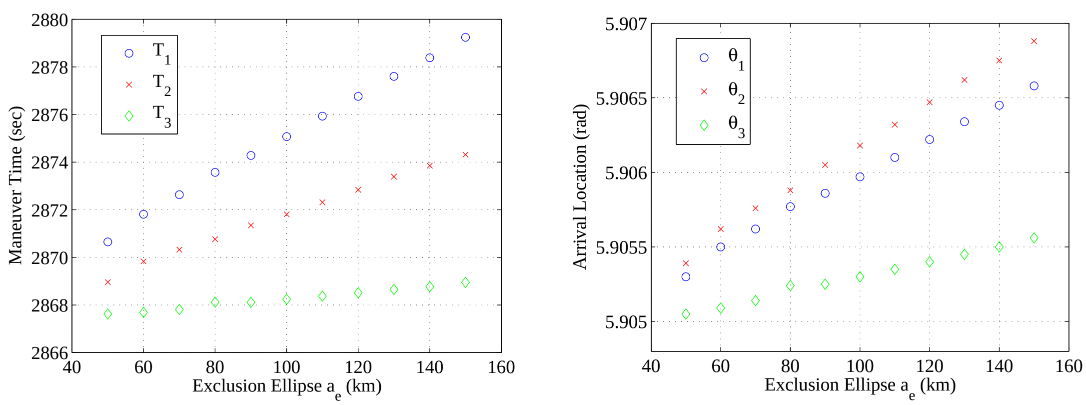
A PSOG with 60 particles was used to optimize triple pass RTM problems with the same conditions studied in the single and double pass cases. The increase in the number of particles was meant to account for the higher dimensionality of the solution space. Equation 3.16 summarizes the triple pass RTM problem.

$$
\boxed{\begin{array}{ll}
\text{minimize J} = \Delta V_1 + \Delta V_2 + \Delta V_3 \text{ m/s} & \\
\text{subject to:} & \\
(\phi_{min}, \phi_{max}) = (-10^\circ, 10^\circ) & (\lambda_{min}, \lambda_{max}) = (-50^\circ, -10^\circ) \\
1200 \text{ s} \leq T_1, T_2, T_3 \leq P & 0 \leq \theta_1, \theta_2, \theta_3 \leq 2\pi \\
R_{a_1}, R_{a_2}, R_{a_3} \leq r_0 + 50 \text{ km} & r_0 - 50 \text{ km} \leq R_{p_1}, R_{p_2}, R_{p_3}
\end{array}} \tag{3.16}
$$

The average time required to complete 20 runs was 706.68 s. This is a significant increase over the single and double pass cases, and is likely due to the larger search space as well as an increased swarm size. The lowest cost solutions are shown in Table 3.4 and the associated design variables can be seen in Table 3.7 of the Appendix. The PSO found several local optimal solutions in addition to those shown in Table 3.4. The largest difference between the local solutions and the best known solutions were 0.089 m/s for r_0 = 6800 km and 0.075 m/s for r_0 = 7300 km, and occurred when a_e = 150 km. Figures 3.6(b) and 3.6(a) show the optimal arrival locations and maneuver times for the triple pass RTM. Equation 3.14 is accurate to within 0.0073 m/s for r_0 = 6800 km and 0.0068 m/sec for r_0 = 7300 km.

Table 3.4: Optimal cost of triple pass RTM problem for varying exclusion ellipse sizes

		a_e/b_e km										
r_0, km		50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
6800	ΔV_1	1.365	1.638	1.910	2.182	2.454	2.726	2.998	3.269	3.541	3.812	4.083
	ΔV_2	1.366	1.640	1.912	2.185	2.458	2.731	3.003	3.276	3.548	3.821	4.093
	ΔV_3	1.366	1.639	1.911	2.184	2.456	2.728	3.000	3.272	3.544	3.816	4.088
	J	4.098	4.916	5.734	6.551	7.369	8.185	9.002	9.818	10.633	11.448	12.263
7300	ΔV_1	1.228	1.473	1.718	1.962	2.207	2.451	2.696	2.940	3.184	3.428	3.672
	ΔV_2	1.229	1.474	1.719	1.965	2.210	2.455	2.700	2.946	3.191	3.435	3.680
	ΔV_3	1.228	1.473	1.719	1.964	2.209	2.453	2.698	2.943	3.187	3.432	3.676
	J	3.684	4.420	5.156	5.891	6.626	7.360	8.094	8.828	9.562	10.295	11.028



(a) T_1, T_2 and T_3 as functions of exclusion ellipse size

(b) θ_1, θ_2, and θ_3 as functions of exclusion ellipse size

Figure 3.6: Optimal design variables and constraints for triple pass RTM as functions of exclusion ellipse size ($r_0 = 6800\ km$)

.

3.8 Conclusion

The particle swarm optimization algorithm proved to be an effective tool for solving single and multiple pass responsive theater maneuvers for a variety of exclusion ellipse sizes. The global and local solutions to the responsive theater maneuver problem, regardless of the number of passes considered, are very similar in terms of time of flight and the optimal intercept location on the exclusion ellipse. The small costs associated with these maneuvers make the responsive theater maneuver construct a viable alternative to increase satellite resiliency in a tactical scenario. Further, the methodology presented in this research could be applied to longer mission scenarios or extended to include maneuvers that take multiple orbits to intercept the exclusion ellipse, such as a case where a spacecraft has several orbits before it would overfly the exclusion zone.

3.9 Appendix

Tables 3.5, 3.6, and 3.7 show the optimal maneuvering solutions for the single, double, and triple pass RTM cases, respectively.

Table 3.5: Optimal single pass RTM for various exclusion ellipse sizes

a_e/b_e, km	T_1, s	θ_1, rad	J, m/s	Time, s	No. Optimal/Total, %
r_0 = 6800 km					
50/5	2870.85	5.90534	1.365	157.99	45
60/6	2871.67	5.90546	1.638	159.80	55
70/7	2872.46	5.90557	1.910	199.54	50
80/8	2873.25	5.90568	2.182	219.94	30
90/9	2874.07	5.90580	2.454	207.20	60
100/10	2874.86	5.90591	2.726	199.53	65
110/11	2875.65	5.90602	2.998	191.98	50
120/12	2876.48	5.90611	3.269	176.90	65
130/13	2877.27	5.90625	3.541	187.18	65
140/14	2878.07	5.90636	3.812	120.49	40
150/15	2878.86	5.90647	4.083	204.93	55
r_0 = 7300 km					
50/5	3192.97	5.90531	1.228	148.71	40
60/6	3193.78	5.90541	1.473	135.38	55
70/7	3194.62	5.90552	1.718	201.11	45
80/8	3195.43	5.90562	1.962	241.04	35
90/9	3196.27	5.90573	2.207	196.60	60
100/10	3197.08	5.90583	2.451	221.50	55
110/11	3197.93	5.90594	2.696	198.20	50
120/12	3198.74	5.90604	2.940	288.01	60
130/13	3199.59	5.90615	3.184	277.72	35
140/14	3200.40	5.90625	3.428	214.36	50
150/15	3201.25	5.90636	3.672	283.35	50

Table 3.6: Optimal double pass RTM for various exclusion ellipse sizes

a_e/b_e, km	T_1, s	θ_1, rad	T_2, s	θ_2, rad	Δ_{V_1}	Δ_{V_2}	J, m/s	Time, sec	No. Optimal/Total, %
				r_0 = 6800 km					
50/5	2870.99	5.90538	2869.23	5.90545	1.365	1.366	2.732	288.30	5
60/6	2871.81	5.90550	2869.69	5.90558	1.638	1.640	3.277	226.58	25
70/7	2872.63	5.90562	2870.18	5.90572	1.910	1.912	3.823	206.24	25
80/8	2873.49	5.90575	2870.64	5.90585	2.182	2.185	4.368	207.01	10
90/9	2874.32	5.90587	2871.14	5.90599	2.454	2.458	4.912	205.81	10
100/10	2875.14	5.90599	2871.63	5.90613	2.726	2.731	5.457	206.96	25
110/11	2875.97	5.90611	2872.10	5.90626	2.998	3.003	6.001	220.59	10
120/12	2876.79	5.90623	2872.56	5.90639	3.269	3.276	6.545	204.65	30
130/13	2877.61	5.90635	2873.06	5.90653	3.541	3.548	7.089	223.82	30
140/14	2878.45	5.90647	2873.53	5.90666	3.812	3.821	7.632	223.11	30
150/15	2879.28	5.90659	2874.00	5.90679	4.083	4.093	8.176	236.09	20
				r_0 = 7300 km					
50/5	3193.20	5.90537	3191.15	5.90538	1.228	1.229	2.456	228.61	20
60/6	3194.05	5.90548	3191.61	5.90550	1.473	1.474	2.947	234.71	35
70/7	3194.93	5.90560	3192.08	5.90562	1.718	1.720	3.437	229.87	25
80/8	3195.82	5.90572	3192.55	5.90574	1.962	1.965	3.927	236.91	25
90/9	3196.70	5.90584	3193.01	5.90586	2.207	2.210	4.417	223.01	30
100/10	3197.55	5.90595	3193.48	5.90598	2.451	2.455	4.907	339.88	15
110/11	3198.44	5.90607	3193.96	5.90610	2.696	2.700	5.396	240.80	25
120/12	3199.33	5.90619	3194.43	5.90622	2.940	2.946	5.885	249.52	15
130/13	3200.18	5.90630	3194.90	5.90634	3.184	3.191	6.375	261.34	15
140/14	3201.07	5.90642	3195.38	5.90646	3.428	3.435	6.863	240.76	15
150/15	3201.96	5.90654	3195.85	5.90658	3.672	3.680	7.352	250.89	20

Table 3.7: Optimal triple pass RTM for various exclusion ellipse sizes

a_e/b_e, km	T_1, s	θ_1, rad	T_2, s	θ_2, rad	T_3, s	θ_3, rad	J, m/s	Time, s	No. Optimal/Total, %
r_0 = 6800 km									
50/5	2870.65	5.90530	2868.96	5.90539	2867.62	5.90505	4.098	536.47	5
60/6	2871.81	5.90550	2869.83	5.90562	2867.69	5.90509	4.916	460.36	10
70/7	2872.63	5.90562	2870.32	5.90576	2867.81	5.90514	5.734	410.37	15
80/8	2873.57	5.90577	2870.76	5.90588	2868.12	5.90524	6.551	694.94	1.75
90/9	2874.28	5.90586	2871.34	5.90605	2868.11	5.90525	7.369	698.45	10
100/10	2875.07	5.90597	2871.81	5.90618	2868.24	5.90530	8.185	908.53	10
110/11	2875.93	5.90610	2872.31	5.90632	2868.37	5.90535	9.002	622.69	10
120/12	2876.76	5.90622	2872.84	5.90647	2868.51	5.90540	9.818	639.11	2.63
130/13	2877.60	5.90634	2873.39	5.90662	2868.65	5.90545	10.633	541.34	15
140/14	2878.38	5.90645	2873.85	5.90675	2868.77	5.90550	11.448	686.64	10
150/15	2879.24	5.90658	2874.31	5.90688	2868.95	5.90556	12.263	689.92	15
r_0 = 7300 km									
50/5	3193.16	5.90536	3191.34	5.90543	3189.43	5.90501	3.684	1276.04	10
60/6	3194.05	5.90548	3191.89	5.90557	3189.52	5.90505	4.420	728.71	10
70/7	3194.93	5.90560	3192.35	5.90569	3189.66	5.90510	5.156	513.70	15
80/8	3195.78	5.90571	3192.90	5.90583	3189.76	5.90514	5.891	763.20	15
90/9	3196.66	5.90583	3193.37	5.90595	3189.86	5.90518	6.626	884.74	10
100/10	3197.51	5.90594	3193.91	5.90609	3190.00	5.90523	7.360	794.95	5
110/11	3198.40	5.90606	3194.42	5.90622	3190.10	5.90527	8.094	870.69	3.45
120/12	3199.28	5.90618	3194.90	5.90634	3190.24	5.90532	8.828	820.93	4.35
130/13	3200.17	5.90630	3195.45	5.90648	3190.38	5.90537	9.562	482.37	15
140/14	3201.02	5.90641	3195.97	5.90661	3190.48	5.90541	10.295	724.63	5
150/15	3201.91	5.90653	3196.48	5.90674	3190.59	5.90545	11.028	789.73	10

IV. Low Thrust Responsive Theater Maneuvers Using Particle Swarm Optimization and Direct Collocation

$(\boldsymbol{p}_{best})$

4.1 Abstract

This research investigates a low-thrust implementation of the responsive theater maneuver, which is designed to alter a spacecraft's entry conditions as it overflies a specified geographic region, called the exclusion zone. A particle swarm optimization algorithm employing shape-based low-thrust trajectory approximation is used to seed a direct orthogonal collocation routine employing a nonlinear programming problem solver. This approach is used to generate optimal low-thrust responsive theater maneuver trajectories. The combination of particle swarm optimization, shape-based low-thrust trajectory approximation, and direct orthogonal collocation is shown to generate fuel-optimal trajectories for single, double, and triple pass cases of the responsive theater maneuver problem. Further, these low-thrust trajectories are shown to satisfy the analytical necessary conditions for an optimal control and require delta-velocities only slightly larger than those required for impulsive responsive theater maneuvers delivering the same effects. As low-thrust propulsion technology improves, the low-thrust responsive theater maneuvers can provide propellant mass expenditure savings over their impulsive counterparts despite requiring more delta-velocity.

4.2 Nomenclature

a_e	=	semimajor axis of exclusion ellipse, km
A_T	=	low-thrust maneuver thrust acceleration, m/sec^2
$A_{T_{max}}$	=	maximum allowable low-thrust maneuver thrust acceleration, m/sec^2
$A_{T_{min}}$	=	minimum allowable low-thrust maneuver thrust acceleration, m/sec^2
b_e	=	semiminor axis of exclusion ellipse, km
$\boldsymbol{h}_k$	=	expected angular momentum vector of satellite at k^{th} time of entry into exclusion zone, km^2/sec
N_{rev}	=	minimum number of orbital revolutions required for multiple revolution impulsive maneuver
P	=	period of the initial orbit, sec
R_{a_k}	=	orbit apogee radius after the k^{th} maneuver, km
R_e	=	distance from expected position of the spacecraft to the actual position of the spacecraft, km
$\boldsymbol{r}_k$	=	expected position vector of satellite at k^{th} time of entry into exclusion zone, km
$\boldsymbol{r}_k^*$	=	actual position vector of spacecraft at k^{th} time of entry into exclusion zone, km
$\boldsymbol{r}_{k_t^-}$	=	position vector of spacecraft at the instant just before the k^{th} impulse, km
R_{max}	=	maximum allowable orbital radius, km
R_{min}	=	minimum allowable orbital radius, km
R_{p_k}	=	orbit perigee radius after the k^{th} maneuver, km
$\boldsymbol{r}_0$	=	initial position vector, km
t_0	=	initial time, sec
t_k	=	expected k^{th} time of entry into exclusion zone, sec
$\boldsymbol{v}_0$	=	initial velocity vector, km/sec
$\boldsymbol{v}_k$	=	expected velocity vector of spacecraft at k^{th} time of entry into exclusion zone, km/sec

$\boldsymbol{v}_k^*$ = actual velocity vector of spacecraft at k^{th} time of entry into exclusion zone, km/sec

$\boldsymbol{v}_{k_t^-}$ = velocity vector of spacecraft at the instant just before the k^{th} impulse, km/sec

$\boldsymbol{v}_{k_t^+}$ = velocity vector of spacecraft at the instant just after the k^{th} impulse, km/sec

T_k = time of flight of the k^{th} maneuver, sec

γ_k = expected flight path angle at exclusion zone entry for the k^{th}

γ_k^* = post-maneuver flight path angle for the k^{th} maneuver, rad

γ_{k_t} = pre-maneuver flight path angle for the k^{th} maneuver, rad

η = thrust pointing angle, rad

θ_k = angle defining position of spacecraft on the k^{th} exclusion ellipse, rad

λ = geocentric longitude, deg

μ = Earth's gravitational parameter, km^3/sec^2

ϕ = geocentric latitude, deg

ψ_k = expected angle traveled by the spacecraft in the orbit plane during the k^{th} maneuver, rad

ψ_k^* = actual angle traveled by the spacecraft in the orbit plane during the k^{th} maneuver, rad

4.3 Introduction

The topic of system resiliency has become increasingly relevant in the space community. The 2010 United States National Space Policy [1], 2011 National Security Space Strategy [2], and the 2014 Quadrennial Defense Review [3] all highlight the importance of resiliency. Specifically, [1] states that one of the primary goals of U.S. space policy is to increase spacecraft resiliency against "denial, disruption, or degradation" from environmental and hostile causes. [4] highlighted four basic principles which define resilience: avoidance, robustness, reconstitution, and recovery. In particular, avoidance is defined as "countermeasures against potential adversaries, proactive and reactive defensive measures taken to diminish the likelihood and consequence of hostile acts or adverse conditions."

Recently, [94] embraced the concept of resiliency through avoidance and introduced impulsive responsive theater maneuvers (RTMs). These maneuvers enhance resiliency by introducing uncertainty into a spacecraft's arrival conditions upon entry into a specified geographic region, called the exclusion zone. The RTM requires the spacecraft to lie on an exclusion ellipse at the expected entry time into the exclusion zone. The ellipse is centered at the expected arrival position of the spacecraft into the exclusion zone. This research introduces a low-thrust version of the RTM, which takes advantage of the shape-based low-thrust trajectory approximation technique introduced in [42]. A particle swarm optimization (PSO) algorithm which employs the shape-based technique is used to generate feasible low-thrust RTM trajectories. These trajectories are then used as initial guesses to seed a direct orthogonal collocation method employing a nonlinear programming (NLP) problem solver. This approach is shown to generate optimal trajectories for single, double, and triple pass RTMs.

4.4 Background

Conway [51] provided a comprehensive survey on state-of-the-art techniques used to optimize spacecraft trajectory problems. In this work, he notes that methods employing NLP problem solvers are reliant on reasonable initial guesses from which to start. Dependence on initial guesses introduces two limitations of employing these methods alone. The first is that it is often extremely difficult to generate feasible initial guesses to these highly nonlinear problems. The second limitation is that even when a suitable initial guess is provided, NLP solvers typically converge in the neighborhood of the guess, making them likely to converge to local minima.

Population-based optimization routines such as evolutionary algorithms (EAs) do not suffer from these limitations. They do not require an initial guess, but rather randomly distribute their population uniformly in the solution space and the associated costs are evaluated. The population evolves or moves according to rules specific to the particular EA variant and the process is repeated. Additionally, EAs are designed as global search algorithms and are more likely to find a global optimal than direct methods employing NLP solvers [7]. In fact, [7] notes that EAs are capable of generating optimal solutions independently or can be used to generate initial guesses for more accurate methods if greater accuracy is required

There are several examples in the literature in which EAs have been employed independently to generate optimal solutions to a variety of spacecraft trajectory problems. Problems considered include interplanetary trajectories [8–19, 21, 22, 24, 25, 85], rendezvous and docking [27], or low Earth orbit trajectories to achieve some specific ground effects such as revisit time or coverage [32–34].

Other research has focused on the use of EAs to generate suitable initial guesses to seed more accurate optimization techniques [20, 23, 26, 87, 88, 95, 96]. In particular, [29, 95] used genetic algorithms employing the shape-based methods developed in [42, 43] to generate feasible low-thrust trajectories, which were used as initial guesses for more accurate methods employing NLP solvers. Specifically, [95] employed the technique to optimize an asteroid deflection mission. [29] optimized a low-thrust asteroid rendezvous trajectory in which three of eight asteroids must be visited as well as a problem in which a spacecraft must rendezvous with one asteroid from each of four groups.

Similarly, this research uses PSO algorithms employing the shape-based techniques from [42, 43] to generate initial guesses for low-thrust RTMs. The global version of the PSO, originally developed in [52, 53], consists of a collection of particles initialized by randomly assigning each particle a position and velocity vector in the solution space. The costs associated with the positions of each particle are used to update the best position visited by swarm, $\boldsymbol{g}_{best}$, and the best position ever visited by that specific particle, $\boldsymbol{p}_{best}$. These values are then used to update each particle's velocity vector, which in turn are used to update each particle's position vector for the next iteration. The process is repeated until a defined convergence criteria is met or a maximum number of iterations is reached.

[52] proposed a local variant of the PSO in which $\boldsymbol{g}_{best}$ is replaced by $\boldsymbol{l}_{best}$, the best position ever visited by a particle's pre-defined nearest neighbors. This modification was designed to prevent the algorithm from converging to local extrema. The local variant has been shown to be more successful in converging to global minima at the expense of computational speed [52]. The performance of the local PSO is highly dependent on the neighborhood size [52]. Hu et al. [97] noted that larger neighborhood sizes provide faster computational speed while smaller neighborhoods prevent premature convergence. [54] stated empirical evidence showed that neighborhood sizes equal to 15% of the swarm size provided good performance.

4.5 Methodology

4.5.1 Responsive Theater Maneuvers

The RTM was originally defined in [94] and is summarized below. The RTM is designed to alter the arrival conditions of a spacecraft as it overflies the exclusion zone, a potentially hazardous geographic region on the earth. The exclusion zone is defined by latitude (ϕ_{min}, ϕ_{max}) and longitude ($\lambda_{min}, \lambda_{max}$) bands.

The satellite state at the initial time t_0 is defined by Earth-centered, inertial position and velocity vectors, $\boldsymbol{r}_0$ and $\boldsymbol{v}_0$. The state of the satellite is subject only to two-body forces propagated forward using Kepler's equation. The earth is assumed spherical, which implies that the spacecraft's geocentric longitude λ and latitude ϕ can be computed at any time using the current position vector and the Greenwich Mean Time (GMT). For simplicity, GMT at t_0 is assumed zero.

[94] defines an analytical method to determine the expected time of entry t_1 into the exclusion zone as well as the expected position and velocity vectors, $\boldsymbol{r}_1$ and $\boldsymbol{v}_1$, respectively. These quantities define the specific angular momentum vector $\boldsymbol{h}_1$ and the $\boldsymbol{g}_1$ vector, which define the orientation of the exclusion ellipse centered at $\boldsymbol{r}_1$. The definition of $\boldsymbol{g}_1$ is shown in Equation 4.1

$$\boldsymbol{g}_1 = \frac{\boldsymbol{v}_1 \times \boldsymbol{h}_1}{|\boldsymbol{v}_1||\boldsymbol{h}_1|} \tag{4.1}$$

The RTM requires the spacecraft to maneuver such that its actual arrival position is on the exclusion ellipse at t_1. The ellipse is oriented such that the semimajor axis a_e is aligned with $\boldsymbol{v}_1$ and semiminor axis b_e is aligned with $\boldsymbol{g}_1$, resulting in an in-plane maneuver.

The spacecraft's actual position at t_1 is defined by Equation (4.2), where θ_1 is an angular variable measured from $\boldsymbol{v}_1$ in the direction of $\boldsymbol{g}_1$.

$$\boldsymbol{r}_1^* = \boldsymbol{r}_1 + R_e \cos\theta_1 \frac{\boldsymbol{v}_1}{|\boldsymbol{v}_1|} + R_e \sin\theta_1 \boldsymbol{g}_1 \tag{4.2}$$

where

$$R_e = \frac{a_e b_e}{\sqrt{b_e^2 \cos^2\theta_1 + a_e^2 \sin^2\theta_1}} \tag{4.3}$$

The variable T_1 defines the time in advance of t_1 at which the maneuver is initiated in addition to the position $\boldsymbol{r}_{1_t^-}$ and velocity $\boldsymbol{v}_{1_t^-}$ vectors just prior to maneuver initiation. That is, maneuver initiation occurs at $t_1 - T_1$ s. In the low-thrust version of the RTM, the variable T_1 also defines the duration of the maneuver.

The post-maneuver orbit is constrained such that its apogee R_{a_1} must be less than a maximum allowable apogee $R_{a_{max}}$ and its perigee R_{p_1} must be greater than a minimum allowable perigee $R_{p_{min}}$. These constraints are specified to ensure the spacecraft meets sensor range constraints required by the mission.

4.5.2 *Shape-Based Approximation Method Applied to Responsive Theater Maneuvers*

Wall and Conway [42] developed a two-dimensional shape-based method to approximate low-thrust interception trajectories. This method can be applied to RTMs because the maneuvers are restricted to the plane of the initial orbit. The specific details for the shape-based approximation are outside the scope of this paper, but can be found in [42]. Some details, such as system dynamics, are presented here for convenience. The notation is slightly modified from the original work to avoid confusion with notation used for the RTM.

The shape-based method defined four states in polar coordinates: the radius magnitude r, angle ψ, radial velocity V_r, and tangential velocity V_ψ, which are subject to the dynamics shown in Equation 4.4. The controls are the thrust acceleration A_T and the control angle η.

$$\begin{aligned} \dot{r} &= V_r \\ \dot{\psi} &= \frac{V_\psi}{r} \\ \dot{V}_r &= \frac{V_\psi^2}{r} - \frac{\mu}{r^2} + A_T \sin\eta \\ \dot{V}_\psi &= -\frac{V_\psi V_r}{r} + A_T \cos\eta \end{aligned} \tag{4.4}$$

The shape-based approximation generates a trajectory and the corresponding ΔV given the pre-maneuver position and velocity magnitudes $r_{1_t^-}$ and $v_{1_t^-}$, the pre-maneuver flight path angle γ_{1_t}, the final position magnitude r_1^*, the final velocity magnitude v_1^*, the final flight path angle γ_1^*, the total angle traveled ψ_1^*, and the maneuver time T_1. It is not necessary to convert from the Cartesian coordinates used to define the RTM to the polar coordinates; all inputs required for the shape-based approximation can be defined using the RTM variables or specified as optimization parameters. Specifically, the RTM variables θ_1 and T_1 define all of these quantities except for v_1^* and γ_1^*, which become optimization parameters.

Recall θ_1 defines the desired entry position of the spacecraft onto the exclusion ellipse at t_1 and thus the final position magnitude r_1^*. The maneuver time T_1 is used along with

Kepler's equation to define the position and velocity vectors at maneuver initiation, $\boldsymbol{r}_{1_t^-}$ and $\boldsymbol{v}_{1_t^-}$, respectively. As a result, $r_{1_t^-}$ and $v_{1_t^-}$ are simply the magnitudes of $\boldsymbol{r}_{1_t^-}$ and $\boldsymbol{v}_{1_t^-}$, respectively. The pre-maneuver state also defines γ_{1_t}.

Additionally, T_1 defines the expected angle ψ_1 the spacecraft will travel from maneuver initiation to exclusion zone entry. Consequently, ψ_1^* can be calculated according to Equation 4.5, where r_1 is the magnitude of $\boldsymbol{r}_1$.

$$\psi_1^* = \psi_1 + \delta \cos^{-1}\left(\frac{(R_e)^2 - (r_1)^2 - \left(r_1^*\right)^2}{-2 r_1 r_1^*}\right) \tag{4.5}$$

Equation 4.5 includes the variable δ, which takes on a value of either positive or negative one and is determined using θ_1 and the orientation of the exclusion ellipse with respect to $\boldsymbol{r}_1$. This orientation is defined by the expected flight path angle γ_1 of the spacecraft upon entry into the exclusion zone. Figure 4.1 shows this orientation and Equation 4.6 defines the value of δ as a function of γ_1.

$$\delta = \begin{cases} -1 & \text{if} \quad \frac{\pi}{2} - \gamma_1 < \theta_1 < \frac{3\pi}{2} - \gamma_1 \\ 1 & \quad\quad\quad \text{otherwise} \end{cases} \tag{4.6}$$

As a result, the variables θ_1 and T_1 define all required variables for the shape-based approximation except v_1^* and γ_1^*. These parameters become optimization variables and define the actual velocity vector $\boldsymbol{v}_1^*$ of the spacecraft as it arrives on the exclusion ellipse. The final flight path angle is restricted such that $\frac{-\pi}{2} < \gamma_1^* < \frac{\pi}{2}$ to ensure the final trajectory is prograde. It should be noted that all distances and times are scaled prior to input into the shape-based approximation. The scaling is such that distances are scaled by the semimajor axis of the initial orbit in km and all times are scaled such that 2π time units are equal to the original orbit's period in s.

A single pass low-thrust RTM can be extended to accommodate subsequent passes by reinitializing the parameters after each maneuver. That is, t_1, $\boldsymbol{r}_1^*$, and $\boldsymbol{v}_1^*$ become the initial conditions to determine the second exclusion zone entry.

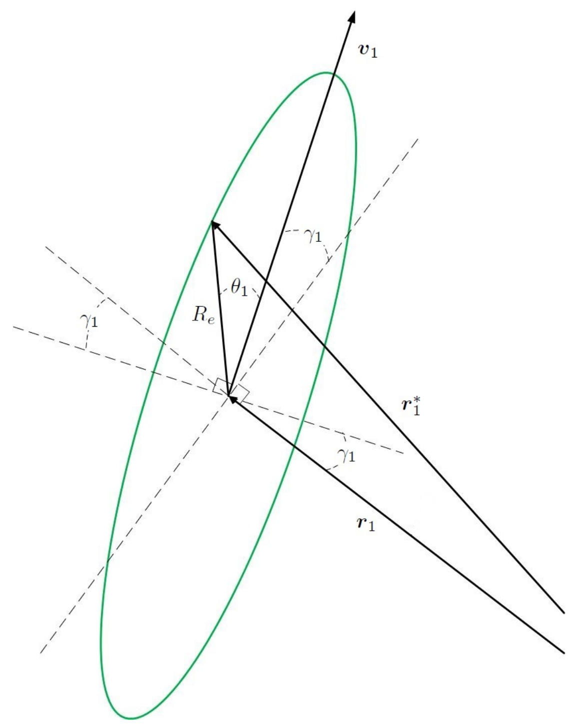


Figure 4.1: Exclusion ellipse orientation with respect to γ_1

4.5.3 Optimal Low-Thrust Responsive Theater Maneuvers

It is important to note that the shape-based approach defined in [42] generates feasible, albeit suboptimal, trajectories. The approach taken in this research was to implement a PSO algorithm combined with the shape-based approach to generate low-thrust trajectories to provide initial guesses into an optimization package [98] employing direct orthogonal collocation (DOC) [48–50, 99] to convert the problem into an NLP problem. The Interior Point Optimizer (IPOPT) [100] was employed as the NLP solver.

The PSO employed in this research is used to identify the minimum ΔV solution resulting from the shape-based method. The death penalty method was used to assign infinite cost to those trajectories not satisfying the apogee and perigee constraints. No restriction was placed on the maximum allowable thrust acceleration for trajectories generated by the PSO, but rather thrust acceleration restrictions were applied during the DOC portion of the optimization. The optimal control problem for the low-thrust RTM is subject to the dynamics in Equation 4.4 and has the form shown in Equation 4.7.

$$\begin{array}{|ll|}\hline \text{minimize J} = \Delta V_1 = \int_{t_0}^{t_1} A_T \, \mathrm{d}t & \\ \text{subject to:} & \\ \text{Exclusion zone:} & (\phi_{min}, \phi_{max}), (\lambda_{min}, \lambda_{max}) \\ R_{a_1} \leq R_{a_{max}} & R_{p_{min}} \leq R_{p_1} \\ A_{T_{min}} \leq A_T \leq A_{T_{max}} & 0 \leq \eta \leq 2\pi \\ \hline \end{array} \tag{4.7}$$

Thus, the system Hamiltonian can be written as shown in Equation 4.8. The variables λ_r, λ_ψ, λ_{V_r}, and λ_{V_ψ} are the Lagrange multipliers corresponding to r, ψ, V_r and V_ψ, respectively.

$$\mathscr{H} = \lambda_r V_r + \lambda_\psi \frac{V_\psi}{r} + \lambda_{V_r}\left(\frac{V_\psi^2}{r} - \frac{\mu}{r^2}\right) + \lambda_{V_\psi}\left(-\frac{V_\psi V_r}{r}\right) + A_T\left(1 + \lambda_{V_r}\sin\eta + \lambda_{V_\psi}\cos\eta\right) \tag{4.8}$$

According to Pontryagin's Minimum Principle, the optimal control can be found by minimizing the Hamiltonian at all times from t_0 to t_1. Thus, the optimal pointing angle is shown in Equation 4.9, where $\lambda_V = \sqrt{(\lambda_{V_r})^2 + \left(\lambda_{V_\psi}\right)^2}$.

$$\begin{aligned} \sin\eta &= -\frac{\lambda_{V_r}}{\lambda_V} \\ \cos\eta &= -\frac{\lambda_{V_\psi}}{\lambda_V} \end{aligned} \tag{4.9}$$

Similarly, the optimal thrust magnitude is shown in Equation 4.10, where $s = \left(1 + \lambda_{V_r}\sin\eta + \lambda_{V_\psi}\cos\eta\right)$ is the switching function.

$$A_T = \begin{cases} A_{T_{max}} & \text{if} \quad s < 0 \\ A_{T_{min}} & \quad \text{otherwise} \end{cases} \tag{4.10}$$

Equations 4.9 and 4.10 were employed to verify that trajectories converged upon by the DOC satisfied the analytical necessary conditions for an optimal control.

4.6 Analysis

4.6.1 Single Pass Responsive Theater Maneuvers

The impulsive single pass RTM scenarios investigated in [94] were analyzed using the low-thrust method described above. These scenarios included multiple exclusion ellipse sizes where a_e ranged from 50 km to 150 km in increments of 10 km and $b_e = 0.1a_e$. Additionally, two different orbits were evaluated. The first orbit was a circular, 45° inclined orbit with semimajor axis equal to 6800 km. The initial conditions were such that the spacecraft starts at the ascending node of the orbit. The second orbit was identical to the first except the semimajor axis was increased to 7300 km. The maximum allowable thrust acceleration $A_{T_{max}}$ was set equal to two meters per second squared while the minimum thrust acceleration $A_{T_{min}}$ was zero meters per second squared. The thrust angle η was unconstrained.

A PSO algorithm was used to generate the fuel-optimal shape-based trajectories with respect to three of the four variables required for the low-thrust RTM: θ_1, v_1^*, and γ_1^*. The variable T_1 was fixed for the purposes of this research. Fixing T_1 is justified because the goal of running the PSO was to generate feasible trajectories to use as initial guesses into the DOC.

In the single pass case $T_1 = t_1 - t_0$. The bounds for each variable are shown in Equation 4.11, while the PSO settings are shown in Table 5.2. The cost function tolerance was $1e-3$ m/s.

$$\begin{array}{ccccc} 0 & \leq & \theta_1 & \leq & 2\pi \\ \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{p_{min}}}{R_{a_{max}}}} & \leq & \boldsymbol{v}_1^* & \leq & \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{a_{max}}}{R_{p_{min}}}} \\ -\frac{\pi}{2} & < & \gamma_1^* & < & \frac{\pi}{2} \end{array} \tag{4.11}$$

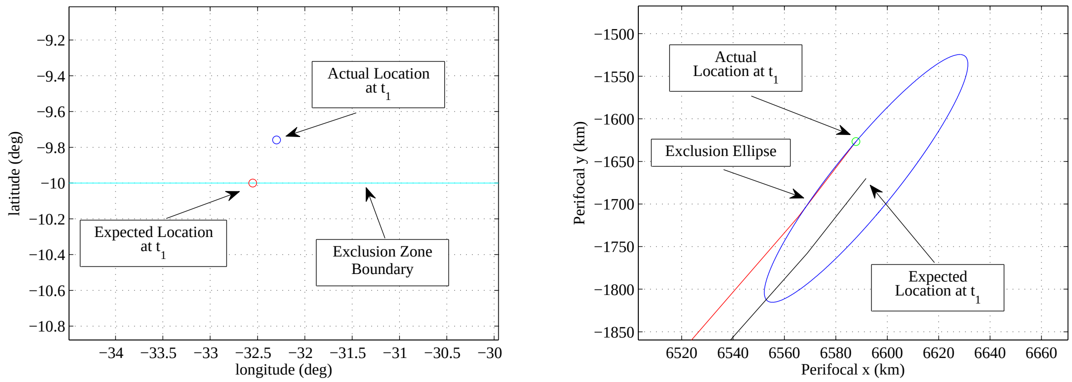
Table 4.1: PSO settings

Swarm Size	40
Max Iterations	1000
Cognitive Parameter	2.09
Social Parameter	2.09
Constriction Factor	0.656295

Research on impulsive RTMs indicated locally optimal solutions corresponding to increases and decreases in altitude [94]. As a result, the initial guesses used as inputs into the DOC were chosen such that one resulted from the lowest cost PSO solution corresponding to an increase in altitude and the other corresponded to the lowest cost PSO solution corresponding to a decrease in altitude. The PSO was used to solve each case twenty times and the lowest cost solutions corresponding to an increase and decrease in altitude were chosen as initial guesses for consecutive calls to the DOC. The first call discretized the continuous time problem into one phase consisting of 80 collocation points. The minimum allowable thrust $A_{T_{min}}$ was set such that $A_{T_{min}} = 0.1A_{T_{max}}$. The output from this call was used as the input to a second call to the DOC, which discretized the problem into a single phase consisting 160 collocation points. Additionally, $A_{T_{min}}$ was set equal to zero. This optimization scheme provided consistent convergence for all cases considered in this research.

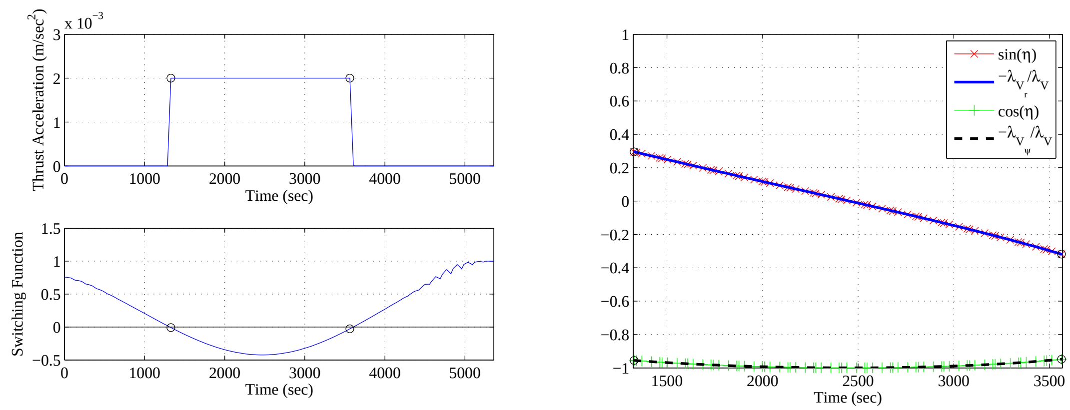
The lowest cost solution found for each case was considered to be the minimum. The combination of PSO and DOC converged to solutions meeting all constraints and satisfying the analytical necessary conditions shown in Equations 4.9 and 4.10 for each case. Figure 4.2 depicts the change in exclusion zone entry conditions while Figure 4.3 shows that the trajectory satisfies the optimal control conditions for the case with $r_0 = 6800\ km$ and $a_e = 150\ km$. The results are representative of those seen for all other cases.

Figures 4.2(a) and 4.2(b) show the entry conditions into the exclusion zone and the arrival conditions on the exclusion ellipse in the perifocal frame, respectively. Figure 4.3(a) shows the thrust magnitude history and the value of the switching function. Figure 4.3(b) shows the necessary condition for the thrust pointing angle while the thruster is on. These figures demonstrate that the trajectory satisfies the analytical necessary conditions for an optimal control defined in Equations 4.9 and 4.10.



(a) Actual latitude and longitude and expected zone entry (b) Arrival location on exclusion ellipse

Figure 4.2: Exclusion zone and exclusion ellipse arrival conditions for $r_0 = 6800$ km, $a_e = 150$ km



(a) A_T magnitude and switching function (b) Optimal thrust pointing condition

Figure 4.3: Optimal control necessary conditions for r_0 = 6800 km, a_e = 150 km

Table 4.2 shows the optimal cost for each single pass low-thrust RTM investigated. In all cases, the spacecraft performs a maneuver such that it arrives at a lower altitude than expected. These results are consistent with those reported for impulsive RTMs [94].

Table 4.2: Optimal cost in m/s of low-thrust single pass RTMs

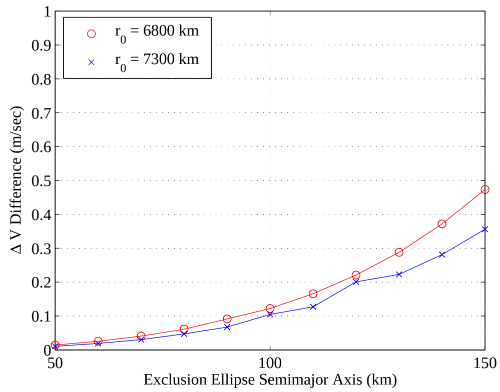
r_0		$a_e/b_e\ (km)$										
(km)	(m/sec)	50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
6800	ΔV	1.380	1.663	1.951	2.243	2.545	2.848	3.164	3.490	3.829	4.184	4.556
7300	ΔV	1.239	1.492	1.748	2.009	2.274	2.556	2.823	3.141	3.407	3.709	4.028

The costs associated with low-thrust RTMs are slightly higher than those seen for impulsive RTMs with similar exclusion ellipse sizes, which is expected. The cost difference between the low-thrust and impulsive cases are shown in Figure 4.4(a). The increased ΔV required for low-thrust RTMs in comparison to impulsive RTMs does not mean, however, that more propellant would be required.

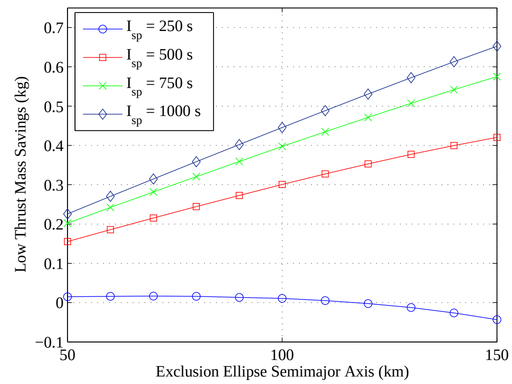
As an example, consider two 500 kg spacecraft, the first of which is designed to perform impulsive RTMs and is equipped with a currently available hydrazine propulsion system [101]. Such a propulsion system would provide a specific impulse I_{sp} of approximately 235 s. The second 500 kg spacecraft would require continuous one Newton thrust to generate the $A_{T_{max}}$ required for low-thrust RTMs.

Figures 4.4(b) and 4.4(c) depict the difference in propellant mass expenditure between low-thrust and impulsive RTMs as functions of exclusion ellipse size and I_{sp} given the proposed propulsion systems.

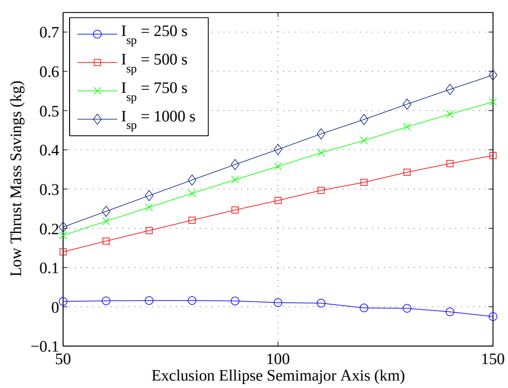
A current flight proven low-thrust propulsion system is capable of delivering the required one Newton thrust with I_{sp} = 250 s [102]. As a result, Figures 4.4(b) and 4.4(c) show that low-thrust RTMs enabled by the flight-proven low-thrust system [102] provide minimal benefit to impulsive RTMs in terms of the propellant mass required. In fact, low-thrust RTMs require more propellant than impulsive RTMs for $a_e > 130\ km$. The figures also show, however, that low-thrust RTMs will result in significant propellant savings as low-thrust propulsion efficiency increases.



(a) Difference in ΔV between impulsive and low-thrust RTMs



(b) Propellant mass savings for a 500 kg satellite (r_0 = 6800 km)



(c) Propellant mass savings for a 500 kg satellite (r_0 = 7300 km)

Figure 4.4: Comparison of impulsive and low-thrust single pass RTMs

4.6.2 Double Pass Responsive Theater Maneuvers

The previously described techniques were applied to solve double pass low-thrust RTMs employing the same initial conditions used in the single pass cases. The exclusion zone, exclusion ellipses, and apogee/perigee constraints also remained the same as those investigated in the single pass low-thrust RTM scenarios.

Two PSO algorithms were employed in series to obtain initial low-thrust guesses for the DOC. The first PSO generated feasible low-thrust trajectories dependent on θ_1, v_1^*, γ_1^*. Once again, $T_1 = t_1 - t_0$. The output from the first PSO run specified the entry conditions for the second pass over the exclusion zone, which occurred at t_2. The second PSO generated feasible low-thrust trajectories dependent on θ_2, v_2^*, γ_2^*. The variable T_2 was fixed such that $T_2 = t_2 - t_1$, where t_2 is the spacecraft's second entry time into the exclusion zone. The serial PSOs were run twenty times for each case studied with bounds on the design variables as shown in Equation 4.12.

$$\begin{array}{ccccc} 0 & \leq & \theta_1, \theta_2 & \leq & 2\pi \\ \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{p_{min}}}{R_{a_{max}}}} & \leq & v_1^*, v_2^* & \leq & \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{a_{max}}}{R_{p_{min}}}} \\ -\frac{\pi}{2} & < & \gamma_1^*, \gamma_2^* & < & \frac{\pi}{2} \end{array} \tag{4.12}$$

The results from the impulsive double pass RTM scenarios [94] described four locally optimal solutions. These locally optimal solutions corresponded to permutations of increasing and decreasing altitude for the first and second maneuvers. As a result, the lowest-cost solution corresponding to each permutation was chosen as an initial guess into the DOC. For all cases, any permutation of increasing/decreasing altitude not generated by the PSO algorithms was initially ignored.

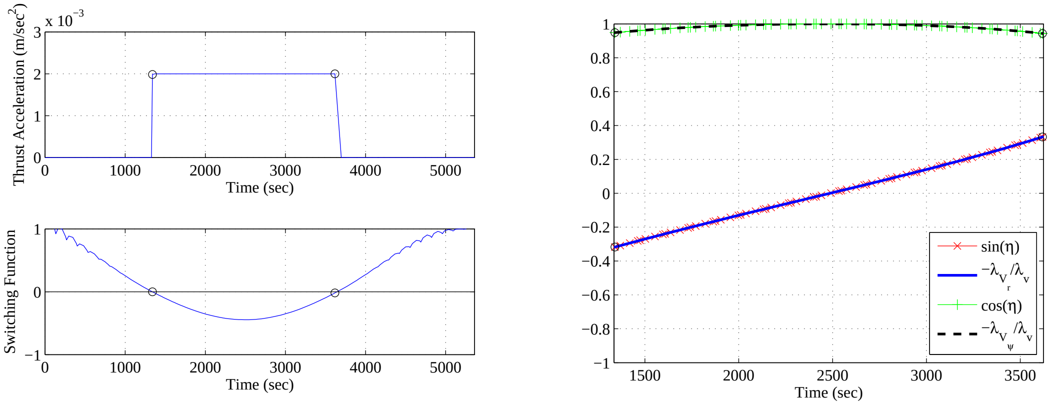
The lowest-cost output from the PSO algorithms corresponding to each possible maneuver permutation were used as initial guesses for a run of the DOC consisting of two phases, one for each pass. Each phase consisted of 80 collocation points. The output from this run was used as an input to a second run of the DOC structured identically to the first. The lower bound for thrust acceleration was set equal to $0.1A_{T_{max}}$ for the first run of the DOC and zero for the second. If the DOC did not yield optimal results for any case, the output from the unused serial PSO runs corresponding to these cases were used as additional initial guesses. This approach yielded optimal results for all cases. The costs corresponding to these solutions are shown in Table 4.3.

Table 4.3: Optimal cost in m/s of low-thrust double pass RTMs

r_0		$a_e/b_e\,(km)$										
(km)	(m/sec)	50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
	ΔV_1	1.384	1.663	1.961	2.255	2.549	2.850	3.192	3.490	3.829	4.241	4.629
6800	ΔV_2	1.378	1.671	1.949	2.239	2.570	2.878	3.156	3.540	3.885	4.170	4.541
	J	2.762	3.334	3.910	4.494	5.119	5.728	6.348	7.030	7.714	8.411	9.169
	ΔV_1	1.242	1.497	1.756	2.009	2.288	2.546	2.823	3.109	3.504	3.764	4.084
7300	ΔV_2	1.237	1.490	1.746	2.025	2.270	2.568	2.851	3.164	3.395	3.700	4.016
	J	2.479	2.987	3.503	4.035	4.559	5.114	5.674	6.273	6.899	7.454	8.100

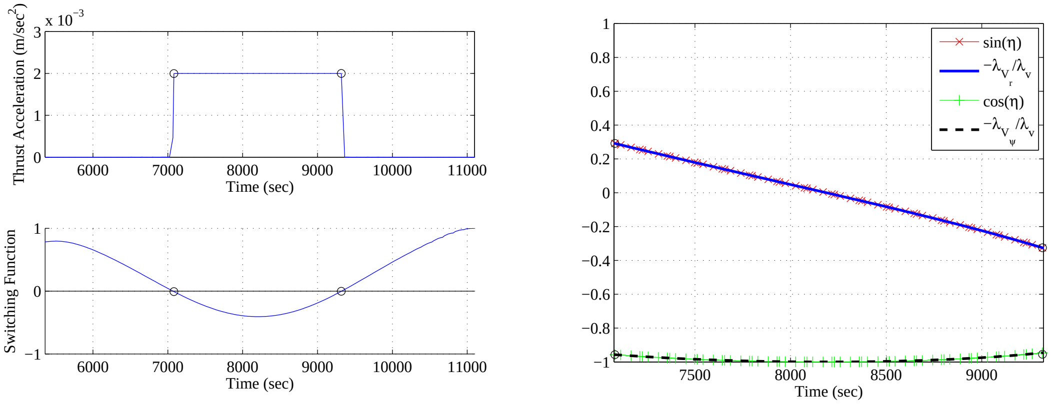
Figure 4.5 shows the thrust acceleration and switching condition along with the optimal pointing direction for each maneuver for the case with $r_0 = 6800\ km$ and $a_e = 150\ km$. The figures are representative of the other double-pass cases considered.

As expected, all low-thrust RTMs require more ΔV than impulsive RTMs for each scenario investigated. The difference in ΔV between the low-thrust and impulsive cases as functions of exclusion ellipse size are shown in Figure 4.6(a). The amount of propellant required for the impulsive and low-thrust RTMs were evaluated using the same propulsion systems described in the single pass case and the results are similar. Figures 4.6(b) and 4.6(c) show the difference in propellant mass expenditure between impulsive and low-thrust RTMs as functions of exclusion ellipse size and I_{sp}. The currently available low-thrust system $\left(I_{sp} = 250\ \text{s}\right)$ implies that low-thrust RTMs provide negligible benefit to impulsive RTMs. As in the single-pass cases, however, low-thrust RTMs will provide a significant benefit in comparison to impulsive RTMs as low-thrust propulsion efficiency increases.



(a) 1^{st} maneuver A_T magnitude and switching function

(b) 1^{st} maneuver optimal thrust pointing



(c) 2^{nd} maneuver A_T magnitude and switching function

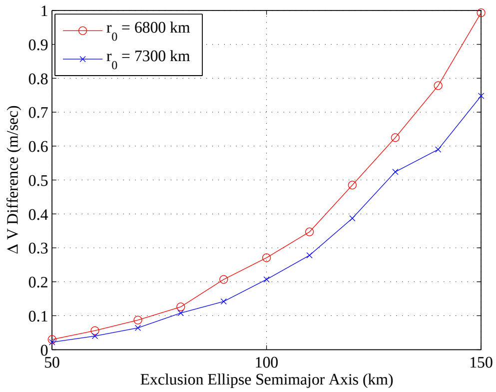
(d) 2^{nd} maneuver optimal thrust pointing

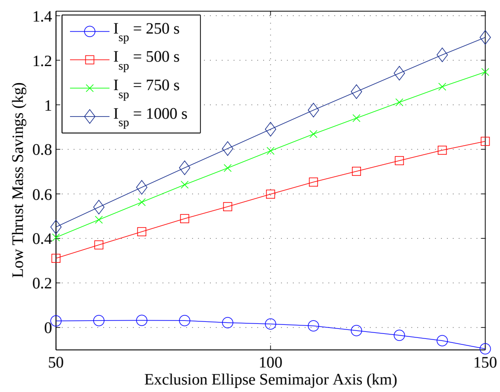
Figure 4.5: Optimal control necessary conditions for double-pass RTM $r_0 = 6800$ km, $a_e = 150$ km

4.6.3 Triple Pass Responsive Theater Maneuvers

4.6.3.1 Triple-Pass Low-Thrust RTMs

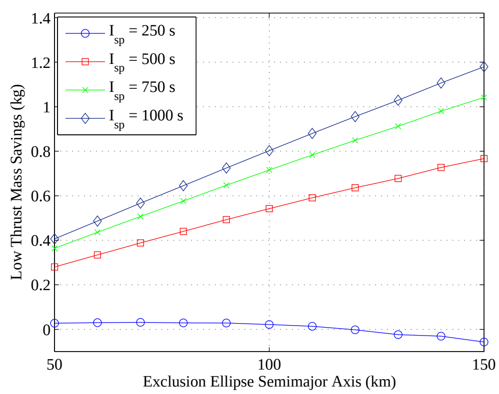
Triple-pass low-thrust RTM scenarios employing the same initial conditions used in the single pass cases were also investigated. The exclusion zone, exclusion ellipses, and apogee/perigee constraints also remained the same as those investigated in the single and double-pass scenarios.





(a) Difference in ΔV between impulsive and low-thrust RTMs

(b) Propellant mass savings for a 500 kg satellite (r_0 = 6800 km)



(c) Propellant mass savings for a 500 kg satellite (r_0 = 7300 km)

Figure 4.6: Comparison of impulsive and low-thrust double pass RTMs

Three PSO algorithms employed in series were used to generate feasible initial guesses to the DOC. The first two PSO algorithms employed restrictions on T_1 and T_2 identical to those described in the single and double pass cases. The time of flight for the third maneuver T_3 was fixed at one orbital period of the initial orbit. This restriction was employed because the scenarios investigated made several orbital revolution between the second and third passes over the exclusion zone. Each case was run twenty times using the three serial PSO algorithms. The serial PSOs were run twenty times for each case studied with bounds on the design variables as shown in Equation 4.13.

$$\begin{array}{ccccc} 0 & \leq & \theta_1, \theta_2, \theta_3 & \leq & 2\pi \\ \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{p_{min}}}{R_{a_{max}}}} & \leq & v_1^*, v_2^*, v_3^* & \leq & \sqrt{\left(\frac{2\mu}{R_{p_{min}}+R_{a_{max}}}\right)\frac{R_{a_{max}}}{R_{p_{min}}}} \\ -\frac{\pi}{2} & < & \gamma_1^*, \gamma_2^*, \gamma_3^* & < & \frac{\pi}{2} \end{array} \tag{4.13}$$

The initial guesses into the DOC for each case were generated by choosing the lowest cost solution found by the PSO algorithms which corresponded to each possible permutation of increasing and decreasing altitude. Any permutation not converged upon by the PSO algorithms was ignored. The trajectories resulting from the PSO runs seeded an initial run of the DOC consisting of four phases. The first two phases were for the first two passes, the third phase imposed a mandatory coast during the second pass through the zone, and the final phase represented the time from the second exit to the third entry into the exclusion zone. The first three phases were discretized into 80 collocation points while the fourth phase consisted of 320 collocation points. The increase in the number of collocation points for the fourth phase was meant to account for the relative length of the final phase in comparison to the first three. The output from the initial run of the DOC was used as an initial guess for a second run of the DOC structured identically to the first. The thrust lower bound was set equal to $0.1A_{T_{max}}$ for the first run and zero for the second. If the DOC did not yield optimal results for any case, the output from the unused serial PSO runs corresponding to these cases were used as additional initial guesses. The approach described above generated optimal results for all of the triple pass cases investigated in this research. The optimal costs for each case are shown in Table 4.4.

Figure 4.7 shows the thrust acceleration and switching condition along with the optimal pointing direction for each maneuver for the case with $r_0 = 6800$ km and $a_e = 150$ km. The figures are representative of those seen for the other cases.

Table 4.4: Optimal cost in m/s of low-thrust triple pass RTMs

r_0		$a_e/b_e\,(km)$										
(km)	(m/sec)	50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
6800	ΔV_1	1.384	1.667	1.964	2.260	2.563	2.875	3.196	3.530	3.878	4.244	4.559
	ΔV_2	1.390	1.676	1.952	2.244	2.542	2.847	3.162	3.486	3.824	4.175	4.649
	ΔV_3	0.483	0.573	0.663	0.752	0.842	0.932	1.022	1.111	1.201	1.290	1.379
	J	3.257	3.916	4.579	5.256	5.947	6.653	7.379	8.127	8.903	9.710	10.587
7300	ΔV_1	1.247	1.502	1.753	2.024	2.293	2.549	2.850	3.141	3.443	3.757	4.086
	ΔV_2	1.243	1.495	1.764	2.011	2.276	2.573	2.822	3.106	3.399	3.703	4.021
	ΔV_3	0.500	0.594	0.689	0.784	0.879	0.974	1.069	1.165	1.261	1.356	1.453
	J	2.989	3.591	4.205	4.819	5.448	6.096	6.742	7.412	8.104	8.816	9.560

The results for the triple pass cases are notable and provide additional insight into the RTM problem not seen in the single and double-pass results. This insight results from the spacecraft having several orbits to complete the final maneuver versus a single orbit to complete the first and second maneuvers. Consequently, the optimal third maneuver found for each case was a short burn immediately after the spacecraft exited the exclusion zone for the second time followed by a long coasting period. These relatively small maneuvers produce dramatic effects after multiple orbits due to the differences in mean motion between the nominal and post-maneuver trajectories. The initial conditions for the single and double-pass scenarios do not allow for these long drift times.

Additionally, the fact that the DOC converged to these specific solutions is significant due to the nature of the initial guess provided by the shape-based method. Recall that the shape-based trajectories generated by the PSO algorithms used a fixed maneuver time for each orbit. The maneuver time for the final orbit T_3 was fixed at one orbital period, which implies that the PSO generated solutions in which the maneuvers took place during the last T_3 s of the allowable maneuvering time. That is, the third maneuver was constrained such that it occurred on the last of several orbits between the second exit out of and third entry into the exclusion zone. The DOC, despite this initial guess, converged to optimal trajectories in which the maneuver occurred immediately after the spacecraft exited the exclusion zone for the second time. This demonstrates the robustness of the technique described in this research with respect to low-thrust RTMs.

4.6.3.2 Triple-Pass Multiple-Revolution Impulsive RTMs

It was desirable to compare the low-thrust triple pass RTM results shown in Table 4.4 to comparable impulsive maneuvers. The triple pass results presented in [94], however, restricted the impulsive maneuvers to take less than one orbital revolution. As a result, the third maneuver in the triple-pass sequences did not take advantage of the long drift time between the second and third passes over the exclusion zone.

In order to provide relevant comparisons to the low-thrust data in Table 4.4, new solutions were generated for the impulsive triple-pass RTMs which allowed for multiple revolution maneuvers. The initial conditions and constraints for the multiple revolution impulsive maneuvers were identical to those presented for the low-thrust triple-pass RTM problem. A Lambert targeting algorithm provided in [41] can generate impulsive maneuvers that complete greater than one revolution around the earth provided the desired number of revolutions N_{rev} are defined. For the purposes of this research, N_{rev} for a responsive theater maneuver is found by dividing T_3 by the period of the nominal orbit and rounding down to the nearest integer value. This algorithm was employed inside a PSO to produce optimal triple pass RTMs for the given problems.

Experimentation with several global and local PSO algorithms led to choosing a local PSO variant to optimize triple-pass multiple-revolution RTMs. The PSO employed a population of 200 particles, neighborhood size of 30 and a maximum of 5000 iterations. Additional stopping conditions were set such that the algorithm was considered to converge if 75% of the particles had the same cost or it the lowest cost found by the swarm had not changed in 1000 consecutive iterations. All other PSO parameters were identical to those shown in Table 5.2. The PSO was not tuned to optimize computational performance because the purpose for solving multiple-revolution impulsive RTMs was for comparison purposes only.

All but one combination of initial orbit size and exclusion ellipse size was optimized ten times. The case with $r_0 = 6800$ km and $a_e = 120$ km was optimized thirteen times to generate results consistent with the other cases. The lowest cost in each case is hereafter referred to as the minimum and each can be seen in Table 4.5. Notice that the ΔV required for the third maneuver is much smaller than that required for the first and second maneuvers. The smaller magnitude of the third maneuver results from the spacecraft having several orbits, and thus more time, to complete the final maneuver. As a result, a relatively small burn produces a change in the mean motion of the spacecraft. The small change in mean motion propagated over several orbits allows the spacecraft to arrive on the exclusion ellipse at the desired arrival time for significantly less ΔV than required for maneuvers occurring in fewer orbital revolutions.

Table 4.5: Optimal cost in m/s of impulsive triple pass RTMs

r_0		$a_e/b_e\,(km)$										
(km)	(m/sec)	50/5	60/6	70/7	80/8	90/9	100/10	110/11	120/12	130/13	140/14	150/15
6800	ΔV_1	1.365	1.645	1.921	2.195	2.449	2.748	3.020	3.276	3.549	3.847	4.130
	ΔV_2	1.367	1.636	1.910	2.181	2.451	2.723	2.993	3.318	3.586	3.803	4.073
	ΔV_3	3.1e-3	6.7e-5	1.9e-4	1.3e-4	8.7e-5	1e-4	1.7e-4	4e-4	9.9e-5	3.1e-4	9.9e-5
	J	2.735	3.281	3.831	4.376	4.920	5.471	6.013	6.594	7.135	7.650	8.203
7300	ΔV_1	1.234	1.480	1.729	1.973	2.223	2.472	2.718	2.962	3.211	3.520	3.697
	ΔV_2	1.227	1.472	1.735	1.960	2.216	2.428	2.704	2.934	3.266	3.486	3.732
	ΔV_3	6.8e-5	2.6e-5	2.9e-4	1.6e-4	5.8e-5	9.3e-5	9.7e-5	1.9e-4	8.7e-4	1e-4	2.8e-4
	J	2.461	2.952	3.464	3.933	4.439	4.920	5.423	5.896	6.477	6.936	7.429

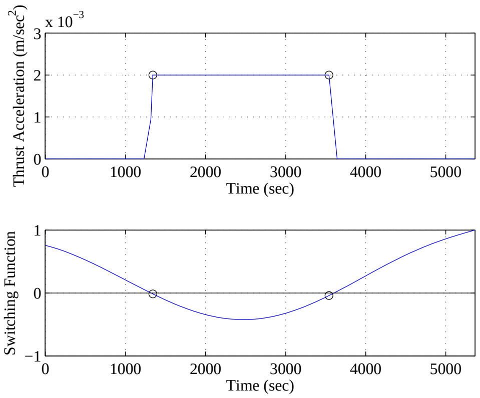
4.6.3.3 Comparison of Triple-Pass Low-Thrust and Impulsive RTMs

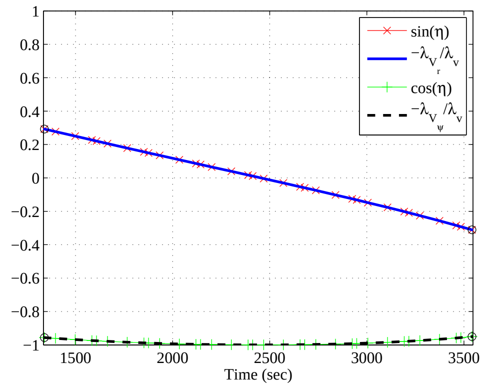
The ΔV required for the impulsive triple-pass RTM scenarios were less than that of their low-thrust counterparts, which is consistent with the single and double-pass results.

The triple-pass results are distinct, however, because the impulsive version is more efficient with respect to propellant consumption for all exclusion ellipse sizes given the current state of propulsion systems discussed in Section 4.6.1. Once again, low-thrust RTMs have the potential to provide significant propellant savings in comparison to impulsive RTMs given increases in low-thrust propulsion efficiency. Figure 4.8(a) shows the difference in ΔV between the impulsive and low-thrust results. Figures 4.8(b) and 4.8(c) show the potential propellant mass savings in kg for low-thrust RTMs in comparison to impulsive RTMs as functions of exclusion ellipse size and low-thrust I_{sp}.

4.7 Conclusion

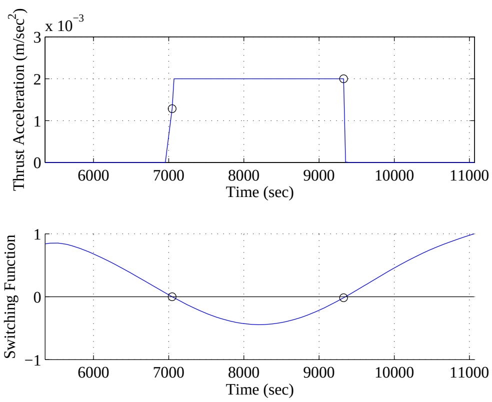
Implementing PSO algorithms to generate shape-based low-thrust trajectory approximations as initial guesses for a direct orthogonal collocation method employing a nonlinear programming problem solver was both effective and robust in generating low-thrust responsive theater maneuvers satisfying the analytical necessary conditions for an optimal control. The technique was able to generate low-thrust maneuvers for two distinct initial orbits and for exclusion ellipses of varying size for single, double and triple pass responsive theater maneuver scenarios. These low-thrust maneuvers required delta-velocities on the order of meters per second and are only slightly larger than those resulting from impulsive maneuvers designed to achieve the same resiliency effects. As low-thrust propulsion technology progresses, however, engine efficiency is expected to improve. While other factors such as power requirements and duty cycle must be considered, this improved efficiency will make low-thrust responsive theater maneuvers significantly more efficient than their impulsive counterparts. Further, these techniques can be extended to longer and more complex scenarios which require a spacecraft to perform several additional maneuvers. These results provide a methodology to develop and optimize maneuvers which increase the resiliency of spacecraft operating in hazardous environments.

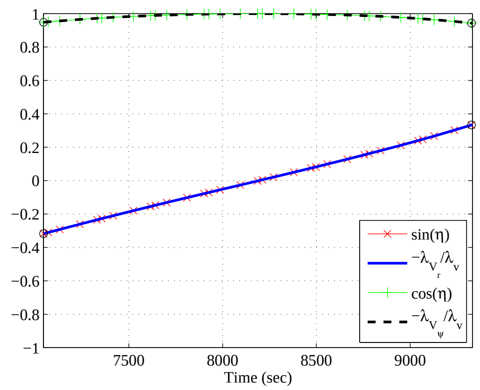




(a) 1^{st} maneuver A_T magnitude and switching function

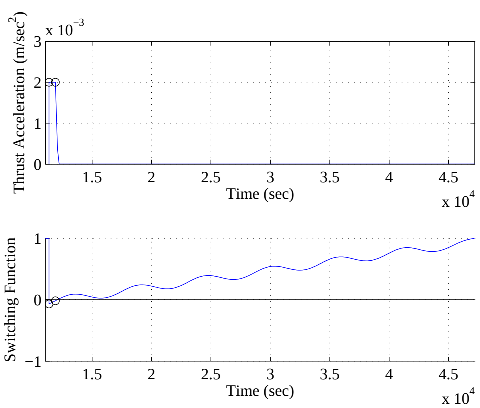
(b) 1^{st} maneuver optimal thrust pointing

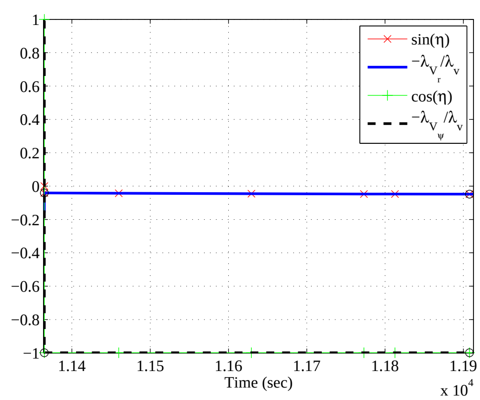




(c) 2^{nd} maneuver A_T magnitude and switching function

(d) 2^{nd} maneuver optimal thrust pointing

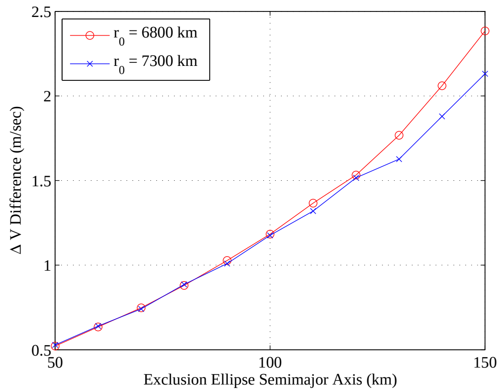


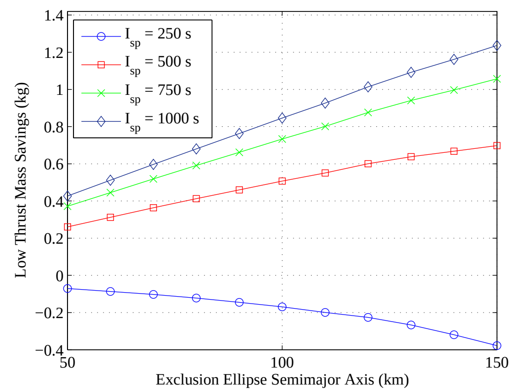


(e) 3^{rd} maneuver A_T magnitude and switching function

(f) 3^{rd} maneuver optimal thrust pointing

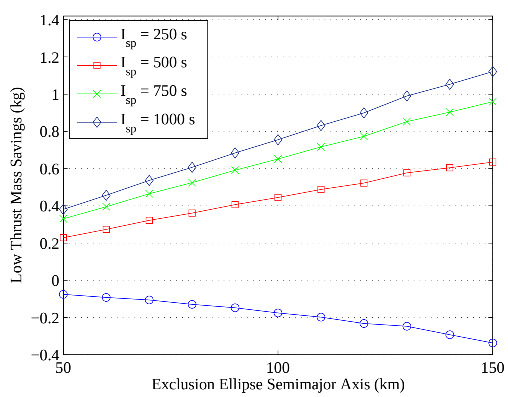
Figure 4.7: Optimal control necessary conditions for triple-pass RTM $r_0 = 6800$ km, $a_e = 150$ km





(a) Difference in ΔV between impulsive and low-thrust RTMs

(b) Propellant mass savings for a 500 kg satellite (r_0 = 6800 km)



(c) Propellant mass savings in for a 500 kg satellite (r_0 = 7300 km)

Figure 4.8: Comparison of impulsive and low-thrust triple pass RTMs

4.8 Appendix: Coordinate Transformation Matrices

$$R1(\tau) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\tau & \sin\tau \\ 0 & -\sin\tau & \cos\tau \end{bmatrix} \tag{4.14}$$

$$R2(\tau) = \begin{bmatrix} \cos\tau & 0 & -\sin\tau \\ 0 & 1 & 0 \\ \sin\tau & 0 & \cos\tau \end{bmatrix} \tag{4.15}$$

$$R3(\tau) = \begin{bmatrix} \cos\tau & \sin\tau & 0 \\ -\sin\tau & \cos\tau & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.16}$$

V. Optimal Geostationary Transfer Maneuvers with Cooperative En-route Inspection Using Hybrid Optimal Control

5.1 Abstract

This research investigates the performance of bi-level hybrid optimal control algorithms in the solution of minimum delta-velocity geostationary transfer maneuvers with cooperative en-route inspection. The maneuvers, introduced here for the first time, are designed to populate a geostationary constellation of space situational awareness satellites while providing additional characterization of objects in lower-altitude orbit regimes. The maneuvering satellite, called the chaser, performs a transfer from low Earth orbit to geostationary orbit. During the transfer, the chaser performs an inspection of one of several orbiting targets in conjunction with a ground site for the duration of the target's line-of-sight contact with the ground site. The chaser's orbit during the inspection is constrained such that it remains inside a cylindrical inspection volume relative to the target for the duration of the target's pass over the ground site. The long axis of the cylindrical volume is aligned with the vector connecting the ground site to the target for the duration of the inspection. The chaser is allowed to transfer to its final orbit upon completion of the cooperative inspection. A three target example is optimized to test the performance of multiple bi-level hybrid optimal control algorithms. Bi-level algorithms employing complete data repositories are shown to generate near-optimal solutions in significantly shorter computational time than complete enumeration of the problem space. A hybrid algorithm employing a data repository and two particle swarm optimization algorithms is then utilized to optimize a fifteen target geostationary transfer maneuver with cooperative en-route inspection. Results indicate that the bi-level algorithm is effective for larger dimensional problems and that these maneuvers can be accomplished for a fraction more delta-velocity than that which is required for a simple transfer to geostationary orbit given the same initial conditions.

5.2 Nomenclature

l_{GEO}	=	true longitude at epoch of the arrival location on the geostationary orbit, *rad*
$\boldsymbol{r}^c_{IJK}$, $\boldsymbol{r}^c_{RSW}$, $\boldsymbol{r}^c_{CYL}$	=	position vectors of the chaser in the inertial, local vertical, local horizontal, cylinder coordinate frames, *km*
$\boldsymbol{r}^g_{IJK}$	=	inertial position vector of the ground site, *km*
$\boldsymbol{r}^m_{IJK}$	=	inertial position vector of the *m*th target
$R_\oplus$	=	radius of the earth, *km*
t_f	=	maneuver completion time, *sec*
t^k_{enter}, t^k_{exit}	=	entry, exit times of the *m*th target's *k*th pass over the ground site, *sec*
t_{max}	=	latest time to initiate cooperative inspection segment, *sec*
t_0	=	initial time, *sec*
t_1	=	time of initial impulsive maneuver to cooperative inspection segment, *sec*
t_2	=	time of fight for maneuver from initial orbit to cooperative inspection segment, *sec*
t_3	=	coast time following cooperative inspection phase, *sec*
t_4	=	time of flight for maneuver to the final mission orbit, *sec*
$\boldsymbol{v}^c_{IJK}$, $\boldsymbol{v}^c_{RSW}$	=	velocity vectors of the chaser in the inertial, and local vertical, local horizontal coordinate frames, *km*
α	=	angle measured from the orbital plane to the cylinder in the local vertical, local horizontal frame, *rad*
β	=	angle measured from the primary axis to the cylinder in the local vertical, local horizontal frame, *rad*
ϵ^c	=	elevation angle of the chaser with respect to the ground site, *rad*

ϵ^c_{max}	=	maximum allowable elevation angle of the chaser with respect to the ground site, *rad*
ϵ^g_{min}	=	minimum elevation angle required by ground site for line-of-sight contact with the mth target, *rad*
ϵ^m	=	elevation angle of the target satellite, *rad*
ϕ, λ	=	geocentric latitude and longitude, *rad*
$\omega_\oplus$	=	rotation rate of the earth, *rad*
$i^{m,c}$, $\Omega^{m,c}$, $u^{m,c}$	=	inclination, right ascension of the ascending node, argument of latitude of the target, chaser *rad*
$\boldsymbol{\rho}_{IJK}$, $\boldsymbol{\rho}_{RSW}$, $\boldsymbol{\rho}_{SEZ}$	=	vector connecting ground site to the target in inertial, local vertical, local horizontal, and topocentric horizon coordinate frames, *km*
$\boldsymbol{\rho}^{\hat{R}}_{RSW}$, $\boldsymbol{\rho}^{\hat{S}}_{RSW}$, $\boldsymbol{\rho}^{\hat{W}}_{RSW}$	=	components of the vector connecting ground site to the target in the $\hat{R}$, $\hat{S}$, $\hat{W}$ directions of the local vertical, local horizontal coordinate frame, *km*
$\boldsymbol{\rho}^{\hat{Z}}_{SEZ}$	=	zenith component of the vector connecting ground site to the target in the topocentric horizon coordinate frame, *km*

5.3 Motivation

The United States Department of Defense (DoD) and Office of the Director of National Intelligence released the National Security Space Strategy (NSSS) in 2011. The document highlights the increasing number of man-made objects in space as well as the increasing number of nations owning or operating satellites [2]. As of 2010, there were over 1,500 active satellites orbiting the Earth. The DoD tracks these satellites along with nearly 20,000 other man-made objects in order to provide space situational awareness (SSA) to all nations using space. Despite these efforts, the DoD estimates that there are "hundreds of thousands of additional objects that are too small to track" [2]. The current congestion in all orbital regimes poses an increasing threat to the safety of active satellites, as highlighted by the 2009 collision of a Russian Cosmos satellite with an Iridium satellite [2]. The problem of congestion will only increase as more objects are launched into space. As a result, the NSSS lists SSA as its top priority, citing the need to improve both the quantity and quality of SSA information to better characterize natural disturbances as well as the capabilities and intentions of other space fairing nations [2].

Ziegler [103] noted that the vast majority of current SSA capability is provided by ground-based sensors, which are essentially limited to "counting and cataloging space objects." Tirpak highlighted current SSA capability gaps which include inadequate characterization of events occurring outside the view of sensors, weather dependent optical observations, and a lack of high quality data in the geosynchronous orbit regime [104]. Recent research has proposed space-based SSA platforms in the form of nanosatellite clusters positioned in various orbital regimes [103] and constellations of satellites operating in or near the geosynchronous belt [105] in order to augment current capabilities and provide characterization of objects.

This work proposes a new type of maneuver to enable higher fidelity, space-based SSA. This maneuver, called the geostationary transfer maneuver with cooperative en-route inspection (GTMEI), requires a maneuvering spacecraft to inspect of one of several orbiting targets before completing a transfer to a geostationary mission orbit. The inspection is performed in cooperation with a designated ground site and lasts for the duration of the target's line-of-sight contact with the site. The GTMEI has the added benefit that the maneuvering satellite could be used to populate a GEO-based SSA constellation such as those proposed in [105] while also providing characterization of targets in lower orbital regimes. This research employs hybrid optimal control (HOC) algorithms to generate minimum fuel GTMEI with target populations of varying size.

5.4 Background

HOC problems consist of combinations of categorical variables and continuous variables. HOC algorithms are particularly interesting because they enable high level autonomous decision making and can be applied to a variety of real world engineering problems. Recent research on the use of HOC in spacecraft trajectory optimization [28–31, 87, 88] has focused on bi-level HOC algorithms with multiple uses for the categorical variables. One use for the categorical variables is to select a planet to use for a gravity-assist or an asteroid with which to rendezvous [28–31]. A second use for the categorical variables is to define the number and sequence of the maneuvers to be performed [30, 31]. Finally, recent research has focused on using the categorical variables to determine the type of maneuvers to be performed, in addition to their number and sequence [87, 88]. In all cases, the structure defined by the categorical variables completely defines the inner-loop optimization problem.

Conway et al. [28] formulated an HOC problem in the solution of a three asteroid interception mission. A maneuvering spacecraft with impulsive-only thrust capability was required to intercept three of a possible eight asteroids with minimum fuel. The authors compared two bi-level algorithms. The first employed a genetic algorithm (GA) as the outer-loop solver and an inner-loop solver consisting of direct transcription with Runge-Kutta implicit integration (DTRK) parallel shooting. The second algorithm employed a branch and bound (B&B) outer-loop solver with a GA inner-loop solver. Complete enumeration was used to determine the optimal sequence and cost. The GA-DTRK found the optimal solution while requiring only a fraction of the number of cost function evaluations required for complete enumeration of the problem space. The B&B-GA located similar solutions to those found by the GA-DTRK algorithm with even fewer cost function evaluations.

Wall and Conway [29] examined the low-thrust version of the minimum fuel asteroid rendezvous problem defined in [28]. The authors used a shape-based approximation to generate feasible low-thrust trajectories with defined boundary conditions. They compared the performance of a bi-level HOC algorithm with a B&B outer-loop solver coupled with a GA inner-loop to that of a GA outer-loop coupled with an inner-loop GA. Once the outer-loop algorithms terminated, the best trajectories found by each hybrid algorithm were used as initial guesses for a DTRK method. [29] implemented a bi-level GA-GA algorithm to solve a larger asteroid rendezvous in which a spacecraft must rendezvous with one asteroid in each of four groups of asteroids. Once again, the best solutions generated by the GA-GA algorithm with shape-based approximation were used as initial guesses for a more accurate DTRK method. The solutions found with the GA-GA algorithm very nearly approximated the optimal solutions identified by the DTRK and required significantly less computational time to generate.

Englander et al. [30] used a bi-level HOC algorithm to optimize interplanetary transfers with unknown locations, numbers, and sequences of en-route flybys. The outer-loop utilized a GA to determine the number, location, and sequence of fly-bys, while the inner-loop employed a combination of particle swarm optimization (PSO) and differential evolution (DE) to optimize the variables corresponding to the sequences generated by the outer-loop. The authors applied this algorithm to three problems: an impulsive multi gravity assist (MGA) transfer from Earth to Jupiter, an impulsive MGA transfer from Earth to Saturn, and an impulsive MGA with deep space maneuver transfer from Earth to Saturn.

Englander et al. [31] extended the work in [30] by adding a capability to model low-thrust trajectories. They utilized a bi-level algorithm consisting of an outer-loop GA coupled with an inner-loop monotomic basin hopping (MBH) algorithm. The result from the MBH algorithm was used as an initial guess to a Sims-Flanagan transcription algorithm used to generate low-thrust trajectories. The authors applied this algorithm to generate optimal trajectories for an Earth to Jupiter transfer employing nuclear electric propulsion, an early proposal for the BepiColombo mission to Mercury, and a solar-electric mission from Earth to Uranus.

Chilan and Conway [87] introduced a new use for HOC in spacecraft trajectory optimization by using the categorical variables to define the number, types, and sequence of maneuvers to be performed between defined boundary conditions. They implemented a bi-level HOC algorithm with a GA outer-loop solver combined with a nonlinear programming (NLP) inner-loop solver. The inner-loop solver was seeded with an initial guess using feasible region analysis and the conditional penalty (CP) method. [87] also demonstrated the effectiveness of the algorithm by solving a minimum-fuel, time-fixed rendezvous between circular orbits originally posed by Prussing and Chui [89]. The algorithm proposed in [87] generated the optimal solution found by Colasurdo and Pastrone [90].

In a subsequent work, Chilan and Conway [88] used a bi-level HOC employing a GA outer-loop solver coupled with an NLP inner-loop solver which was seeded by a GA employing the CP method. They applied this algorithm to the time-fixed rendezvous problem posed in [89] and found a low-thrust trajectory which had a lower cost than, but was analogous to the best impulsive solution found in [90]. [88] applied the same bi-level HOC to find a minimum fuel, free final time trajectory from Earth to Mars.

Yu et al. [91] developed a bi-level HOC algorithm to determine optimal trajectories for several variants of a GEO debris removal problem. They compared the performance of a simulated annealing (SA) outer-loop solver coupled with a GA to that of an exhaustive search coupled with a GA to solve the inner-loop problem. Additionally, the authors developed a so-called rapid method for the outer-loop solver and found that it generated similar solutions to that of the SA outer-loop solver, but required much less computational time.

This research employs HOC algorithms to generate minimum fuel solutions to the GTMEI. The GTMEI is designed to deliver an SSA platform from low Earth orbit (LEO) to geostationary orbit while performing a cooperative inspection of one of a set of uncharacterized targets while en-route to the geostationary orbit belt, where it will serve as a space-based SSA platform. The categorical variables are used to designate a specific target and pass for the cooperative inspection. This inspection is defined such that the SSA platform is in a relative orbit with the designated object for the duration of the object's line-of-sight contact with a specified ground station.

The relative motion segment of the maneuver relies on the linearized equations of motion originally proposed by Hill [44] and Clohessy and Wiltshire [45]. Recent research in the field of relative spacecraft motion has focused on constraining the motion of the chaser inside a specified area or volume defined in relation to the target. Hope and Trask [106] proposed a pogo orbit that intersects itself in the local vertical, local horizontal coordinate frame (RSW), allowing the chaser to perform single impulsive burns at the intersection to maintain a "hover" relative to the target. [106] restricted the motion of the chaser such that it stayed in the orbital plane of the target. Irvin et al. [46] developed a more general framework in which the chaser's motion was constrained inside an elliptical cylinder fixed relative to the target. [46] also presented a method to determine the chaser's initial and final relative velocities given its initial and final relative positions and the time of flight between them.

This research extends the work in [46] by defining a volume that moves in the RSW frame with respect to the target satellite as it passes over the ground site. The GTMEI requires the chaser to remain inside the moving volume for the duration of the cooperative inspection segment, which lasts while the target is in view of a designated ground site. Once the cooperative inspection is complete, the maneuvering SSA platform can initiate a transfer to the final geostationary orbit.

5.5 Geostationary Transfer Maneuver with En-route Inspection

The GTMEI requires a chaser to transfer from a circular parking orbit to a final geostationary mission orbit. The chaser performs two impulsive maneuvers to place it in relative motion with the mth of M targets for the duration of the target's kth horizon-to-horizon contact with the ground site, which is defined by its geocentric latitude ϕ and longitude Λ. The motion of the chaser during the cooperative inspection is restricted to a cylindrical volume relative to the target, the axis of which is coincident with the vector connecting the ground site to the target. The chaser then performs two additional impulsive maneuvers upon completion of the cooperative inspection segment which deliver it to the final mission orbit. The selection of a specific target m and pass k determines the start and end times of the relative motion segment and the initial and final positions of the moving cylinder. The chaser completes the entire transfer from the initial to the final orbit in three segments: impulsive transfer to target, cooperative inspection, and impulsive transfer to geostationary orbit.

5.5.1 Cooperative Inspection Boundary Conditions

The problem begins at initial time t_0, assumed zero without loss of generality. The mth target begins in a circular orbit with a state defined by its semi-major axis a^m, inclination i^m, right ascension of the ascending node Ω^m, and argument of latitude $u^m(t_0)$ at t_0. The initial state of the ground site is defined by ϕ, Λ, and the site's Greenwich mean standard time at t_0, hereafter set equal to Λ for simplicity. The target's state is propagated forward using Kepler's equation from t_0 to a maximum time t_{max} and the state of the ground site is propagated according to a spherical Earth assumption in order to determine the number of target passes over the ground site. The inertial position vector of the ground site $\boldsymbol{r}^g_{IJK}$ at any time t is

$$\boldsymbol{r}^g_{IJK}(t) = \begin{bmatrix} R_\oplus \cos\phi \cos(\Lambda + \omega_\oplus t) \\ R_\oplus \cos\phi \cos(\Lambda + \omega_\oplus t) \\ R_\oplus \sin\phi \end{bmatrix} \tag{5.1}$$

Where, $R_\oplus$ and $\omega_\oplus$ are the radius and the rotation rate of the Earth, respectively. The position vector of the target at time t is $\boldsymbol{r}^m_{IJK}(t)$ and can be found using the target's initial orbital elements and Kepler's equation. The vector originating at the ground site and pointing to the target is

$$\boldsymbol{\rho}_{IJK}(t) = \boldsymbol{r}^m_{IJK}(t) - \boldsymbol{r}^g_{IJK}(t) \tag{5.2}$$

Equation 5.3 defines the rotation of a vector from the inertial frame to the topocentric horizon coordinate frame (SEZ) frame centered at the ground site [36, pp. 175].

$$\boldsymbol{\rho}_{sez}(t) = R2(\pi/2 - \phi)\, R3(\Lambda + \omega_\oplus t)\, \boldsymbol{\rho}_{IJK}(t) \tag{5.3}$$

Where $R1$, $R2$, and $R3$ are right handed rotation matrices about an angle τ and are defined in Equations 5.4, 5.5, and 5.6.

$$R1(\tau) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\tau & \sin\tau \\ 0 & -\sin\tau & \cos\tau \end{bmatrix} \tag{5.4}$$

$$R2(\tau) = \begin{bmatrix} \cos\tau & 0 & -\sin\tau \\ 0 & 1 & 0 \\ \sin\tau & 0 & \cos\tau \end{bmatrix} \tag{5.5}$$

$$R3(\tau) = \begin{bmatrix} \cos\tau & \sin\tau & 0 \\ -\sin\tau & \cos\tau & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

The times at which the target is in view of the ground station in the interval from t_0 to t_{max} can be determined at discrete time steps by evaluating the target's elevation angle ϵ^m

with respect to the ground site. If the relationship in Equation 5.7 holds true, the satellite is in view of the ground site. Note, $\boldsymbol{\rho}_{sez}^{\hat{z}}$ is the Zenith component of $\boldsymbol{\rho}_{sez}$ and ϵ_{min}^{g} is the minimum elevation angle required for line of sight contact from the ground site.

$$\epsilon^{m} = \sin^{-1}\left(\frac{\boldsymbol{\rho}_{sez}^{\hat{z}}(t)}{|\boldsymbol{\rho}_{sez}(t)|}\right) > \epsilon_{min}^{g} \tag{5.7}$$

Each pass k over the ground site between t_0 and t_{max} has an entry time t_{enter}^{k} and a corresponding exit time t_{exit}^{k} defining the line-of-sight contact of the target with the ground site. Given a specific choice of satellite m and pass k, the chaser is required to enter the cooperative inspection segment at t_{enter}^{k}. The cooperative inspection has a duration of $t_{enter}^{k} - t_{exit}^{k}$ seconds and the chaser is permitted to initiate its transfer to the final mission orbit sometime after t_{exit}^{k}.

5.5.2 Cooperative Inspection Segment

The RSW frame is centered at the target with the primary $\left(\hat{R}\right)$ axis aligned with $\boldsymbol{r}_{IJK}^{m}$. A second $\left(\hat{S}\right)$ axis is normal to $\hat{R}$ and points in the direction of the inertial velocity vector of the target. The $\hat{S}$-axis is coincident with the target's velocity vector if the target if the target's orbit is circular. The third $\left(\hat{Z}\right)$ axis points in the orbit normal direction. The coordinates of $\boldsymbol{\rho}_{IJK}(t)$ are converted into the RSW frame according to Equation 5.8. The rotation angles are the target's right ascension of the ascending node, (Ω^{m}), inclination, (i^{m}), and argument of latitude, $(u^{m}(t))$.

$$\boldsymbol{\rho}_{RSW}(t) = R3\left(u^{m}(t)\right) * R1\left(i^{m}\right) * R3\left(\Omega^{m}\right) * \boldsymbol{\rho}_{IJK}(t) \tag{5.8}$$

The angle $\alpha(t)$, measured from the fundamental (orbital) plane in the RSW frame to $\boldsymbol{\rho}_{RSW}(t)$ is found according to Equation 5.9, where $\hat{\boldsymbol{\rho}}_{RSW}(t)$ is the unit vector corresponding to $\boldsymbol{\rho}_{RSW}(t)$. Similarly, the angle $\beta(t)$, measured from the primary axis in the RSW frame to $\boldsymbol{\rho}_{RSW}(t)$ is found according to Equation 5.10. The superscripts, $\hat{R}$, $\hat{S}$, and $\hat{W}$ represent components on each axis in the RSW frame.

$$\sin\alpha(t) = \hat{\boldsymbol{\rho}}_{RSW}^{\hat{W}}(t) \tag{5.9}$$

$$\tan\beta(t) = \frac{\hat{\boldsymbol{\rho}}^{\hat{S}}_{RSW}(t)}{\hat{\boldsymbol{\rho}}^{\hat{R}}_{RSW}(t)} \tag{5.10}$$

The angles $\alpha(t)$ and $\beta(t)$ are used to define the cylinder frame, which is centered at the target and oriented such that its primary axis is aligned with $\boldsymbol{\rho}_{RSW}(t)$. Any vector in the cylinder frame, $\boldsymbol{r}_{CYL}(t)$, can be converted to a vector in the RSW frame, $\boldsymbol{r}_{RSW}(t)$, according to Equation 5.11.

$$\boldsymbol{r}_{RSW}(t) = R3(-\beta(t)) * R2(\alpha(t)) * \boldsymbol{r}_{CYL}(t) \tag{5.11}$$

The chaser's position during the cooperative inspection segment is constrained such that it must be on the primary axis of the cylinder coordinate frame (CYL) frame at t^k_{enter} and t^k_{exit} and inside the cylinder at all times in between. The cylinder is defined in the CYL frame such that one base is at x^{min}_{CYL} and the other is at x^{max}_{CYL} with a length equal to x^{max}_{CYL} minus x^{min}_{CYL}. The variables $x_{CYL}\left(t^k_{enter}\right)$ and $x_{CYL}\left(t^k_{exit}\right)$ define the chaser's position vectors in the CYL frame at the beginning and end of the cooperative inspection segment, respectively. The corresponding vectors in the CYL frame are shown in Equation 5.12.

$$\boldsymbol{r}^c_{CYL}\left(t^k_{enter}\right) = \begin{bmatrix} x_{CYL}\left(t^k_{enter}\right) \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{r}^c_{CYL}\left(t^k_{exit}\right) = \begin{bmatrix} x_{CYL}\left(t^k_{exit}\right) \\ 0 \\ 0 \end{bmatrix} \tag{5.12}$$

The vectors shown in Equation 5.12 are rotated into vectors in the RSW frame, $\boldsymbol{r}^c_{RSW}\left(t^k_{enter}\right)$ and $\boldsymbol{r}^c_{RSW}\left(t^k_{exit}\right)$, using Equation 5.11. [46] describes a method to find the entry and exit velocities in the RSW frame, $\boldsymbol{v}^c_{RSW}\left(t^k_{enter}\right)$ and $\boldsymbol{v}^c_{RSW}\left(t^k_{exit}\right)$, respectively, given two position vectors and the time of flight between them. In this case, the chaser's initial and final relative position vectors are $\boldsymbol{r}^c_{RSW}\left(t^k_{enter}\right)$ and $\boldsymbol{r}^c_{RSW}\left(t^k_{exit}\right)$, respectively. The time of flight is $t^k_{enter} - t^k_{exit}$ seconds. The relative position and velocity vectors at t^k_{enter} are converted to inertial coordinates using Equations 5.13 and 5.14. The inertial state of the chaser at t^k_{exit} is found in the same way. The inertial position and velocity vectors are used to determine the cost of the first and second impulsive maneuvers. For the duration of this paper, a minus superscript (−) denotes a state just prior to an impulsive maneuver while a plus superscript (+) denotes a state just after an impulsive maneuver. Note that all impulses are assumed instantaneous, which implies the position vectors just prior to and just after an impulse are the same.

$$\boldsymbol{r}^c_{IJK}\left(t^{k+}_{enter}\right) = R3\left(-\Omega^m\right) * R1\left(-i^m\right) * R3\left(-u^m\left(t^k_{enter}\right)\right) * \boldsymbol{r}^c_{RSW}\left(t^k_{enter}\right) \tag{5.13}$$

$$\boldsymbol{v}^c_{IJK}\left(t^{k+}_{enter}\right) = R3\left(-\Omega^m\right) * R1\left(-i^m\right) * R3\left(-u^m\left(t^k_{enter}\right)\right) * \boldsymbol{v}^c_{RSW}\left(t^k_{enter}\right) \tag{5.14}$$

5.5.3 Impulsive Transfer to Target

The chaser's initial circular orbit is defined by its semi-major axis a^c, inclination i^c, and right ascension of the ascending node Ω^c. The chaser initiates its first impulsive transfer at a specified argument of latitude $u^c\left(t_1\right)$ where t_1 is the time of maneuver initiation. The position and velocity vectors, $\boldsymbol{r}^c_{IJK}\left(t^-_1\right)$ and $\boldsymbol{v}^c_{IJK}\left(t^-_1\right)$, respectively, can be determined using the chaser's orbital elements at t^-_1.

The chaser must arrive in relative motion with the target at time t^k_{enter} with the inertial position specified by $\boldsymbol{r}^c_{IJK}\left(t^k_{enter}\right)$. The time of flight to complete the maneuver is t_2 seconds, which implies maneuver initiation occurs at $t_1 = t^k_{enter} - t_2$ seconds. Connecting two position vectors in a specified time is the well-known Lambert's problem, the solution of which yields the chaser's departure and arrival velocities on the transfer orbit $\boldsymbol{v}^c_{IJK}\left(t^+_1\right)$ and $\boldsymbol{v}^c_{IJK}\left(t^{k-}_{enter}\right)$, respectively. This research utilized a Lambert targeting algorithm provided by [41]. The first and second maneuvers have costs according to Equations 5.15 and 5.16.

$$\Delta V_1 = |\boldsymbol{v}^c_{IJK}\left(t^+_1\right) - \boldsymbol{v}^c_{IJK}\left(t^-_1\right)| \tag{5.15}$$

$$\Delta V_2 = |\boldsymbol{v}^c_{IJK}\left(t^{k+}_{enter}\right) - \boldsymbol{v}^c_{IJK}\left(t^{k-}_{enter}\right)| \tag{5.16}$$

The path of the chaser is constrained for $t_1 \leq t \leq t^k_{enter}$ according to Equation 5.17, where $\epsilon^c\left(t\right)$ is the elevation angle of the chaser with respect to the ground site at any time and ϵ^c_{max} is the maximum allowable elevation angle of the chaser with respect to the site.

$$\epsilon^c\left(t\right) < \epsilon^c_{max} \tag{5.17}$$

5.5.4 Impulsive Transfer to GEO Segment

The chaser coasts for t_3 seconds after t_{exit}^k, which defines the inertial state of the chaser at the instant prior to the third impulsive burn, $\boldsymbol{r}_{IJK}^c\left(t_{exit}^k + t_3^-\right)$ and $\boldsymbol{v}_{IJK}^c\left(t_{exit}^k + t_3^-\right)$. The coast is restricted such that the chaser may not come within 50 meters of the target. The desired final position of the chaser is defined by its true longitude at epoch in the geostationary orbit, $l_{GEO}\left(t_f\right)$, which corresponds to inertial position and velocity vectors, $\boldsymbol{r}_{IJK}^c\left(t_f^+\right)$ and $\boldsymbol{v}_{IJK}^c\left(t_f^+\right)$, respectively. The chaser travels from $\boldsymbol{r}_{IJK}^c\left(t_{exit}^k + t_3^-\right)$ to $\boldsymbol{r}_{IJK}^c\left(t_f\right)$ in t_4 seconds, which lends itself to a Lambert targeting solution. The initial and final velocities on the transfer orbit, $\boldsymbol{v}_{IJK}^c\left(t_{exit}^k + t_3^+\right)$ and $\boldsymbol{v}_{IJK}^c\left(t_f^-\right)$, respectively, provide the final elements needed to compute the cost corresponding to the third and fourth maneuvers, as shown in Equations 5.18 and 5.19.

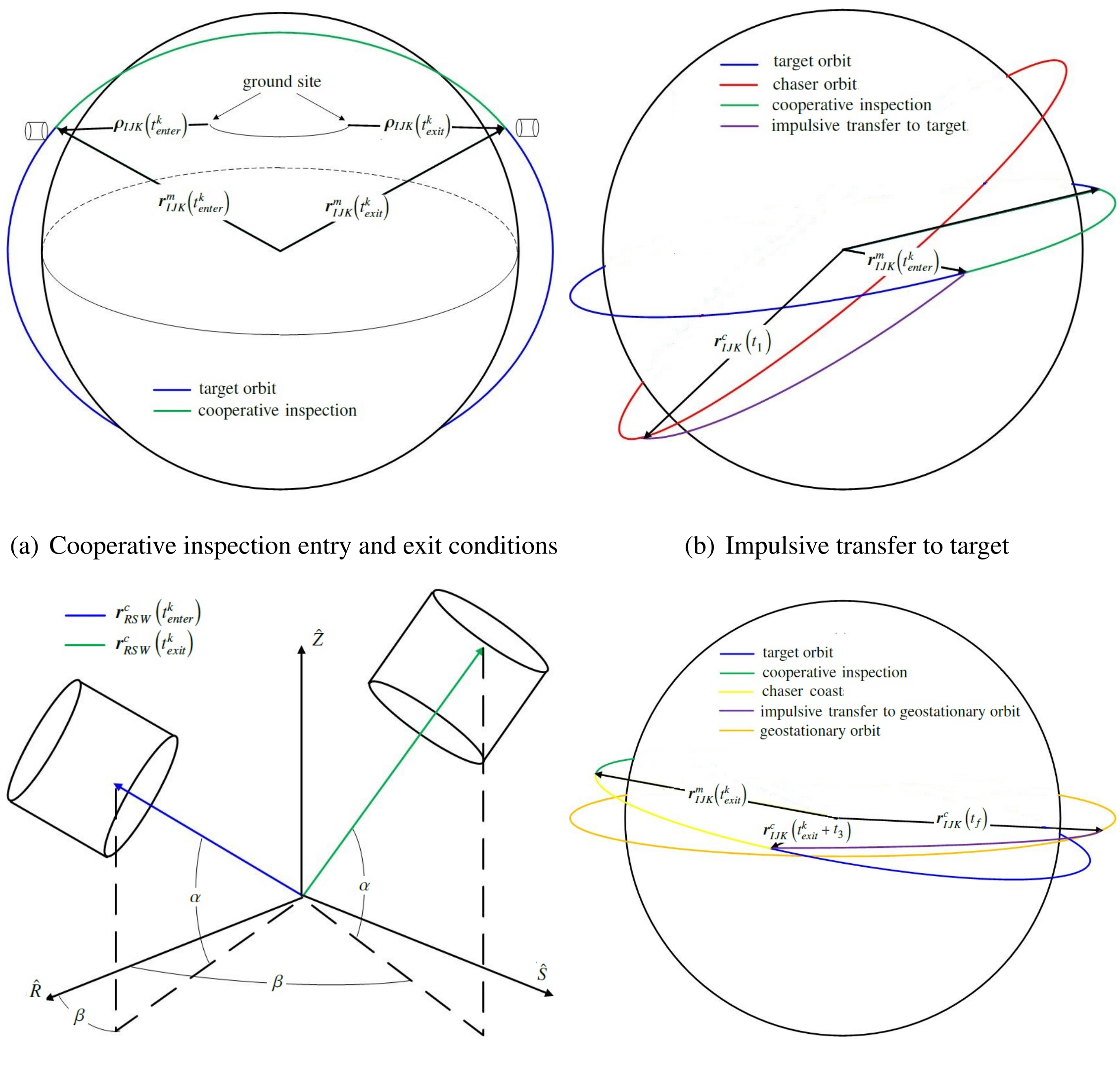
$$\Delta V_3 = |\boldsymbol{v}_{IJK}^c\left(t_{exit}^k + t_3^+\right) - \boldsymbol{v}_{IJK}^c\left(t_{exit}^k + t_3^-\right)| \tag{5.18}$$

$$\Delta V_4 = |\boldsymbol{v}_{IJK}^c\left(t_f^+\right) - \boldsymbol{v}_{IJK}^c\left(t_f^-\right)| \tag{5.19}$$

The path of the chaser is constrained for $t_{exit}^k \leq t \leq t_f$ according to Equation 5.17.

5.5.5 GTMEI as a Hybrid Optimal Control Problem

The GTMEI problem is an HOC problem with two categorical variables m and k. The choice of a specific target and pass combination defines the times of the chaser's cooperative inspection segment with the target. The definition of the target-pass combination specifies the bounds required to optimize the seven continuous variables: $u_c\left(t_1\right)$, t_2, $x_{CYL}\left(t_{enter}^k\right)$, $x_{CYL}\left(t_{enter}^k\right)$, t_3, $l_{GEO}\left(t_f\right)$, and t_4. Figure 5.1 depicts the four phases of the GTMEI problem and Equation 5.20 defines the optimization formulation. The target with the largest number of passes over the ground site from t_0 to t_{max} sets the upper bound K on the categorical pass variable. Any target m which has $L < K$ passes over the ground site for $t < t_{max}$ is assigned an infinite cost for $k > L$. Additionally, the value of k specifies the upper bounds on the inner-loop problem variables t_2 and t_3.



(a) Cooperative inspection entry and exit conditions

(b) Impulsive transfer to target

(c) Cooperative inspection in RSW frame

(d) Impulsive transfer to geostationary orbit

Figure 5.1: Segments of the GTMEI problem

$$
\begin{aligned}
&\text{minimize } J(\boldsymbol{x}) = \Delta V_1 + \Delta V_2 + \Delta V_3 + \Delta V_4 \text{ km/s} \\
&\text{where } \boldsymbol{x} = \left[m, k, u^c(t_1), t_2, x_{CYL}\left(t^k_{enter}\right), x_{CYL}\left(t^k_{exit}\right), t_3, l_{GEO}\left(t_f\right), t_4\right] \\
&\text{subject to:} \\
&1 < m < M \\
&1 < k < K \\
&0 \le u^c(t_1) < 2\pi \\
&t^k_{exit} < t_{max} \\
&1 < t_2 < t^k_{enter} \\
&x^{min}_{CYL} \le x_{CYL}\left(t^k_{enter}\right), x_{CYL}\left(t^k_{exit}\right) \le x^{max}_{CYL} \\
&0 \le t_3 < t^{k+1}_{enter} - t^k_{exit}
\end{aligned}
\tag{5.20}
$$

5.6 Analysis

5.6.1 Three Target Problem

The three target GTMEI problem required the chaser satellite to transfer from LEO to geostationary orbit while inspecting one of three coplanar targets. Two of these targets were in LEO while the third target was in mid-Earth orbit (MEO). The relatively small number of targets allowed for complete enumeration of the problem space and provided an opportunity to test the performance of different bi-level HOC algorithms with respect to cost, computational speed, and number of cost function evaluations required for convergence.

The chaser's cooperative inspection segment with the target must be in conjunction with a ground site defined by $\phi = 45^o$ and $\Lambda = 0^o$. Further, the cooperative inspection lasts for the duration of the target's horizon to horizon contact with the ground site. In other words, ϵ^g_{min} is set equal to zero. Finally, the chaser's elevation angle with respect to the ground site ϵ^c_{max} was defined to be equal to one degree. The cylinder bases x^{min}_{CYL} and x^{max}_{CYL} are set at one and three km, respectively. t_{max} and $t_{4_{max}}$ are set equal to 36 and 16 hours, respectively. The value of t_{max} determines the number of passes for each of the three potential targets. The initial conditions of the chaser and targets are shown in Table 5.1 along with the number of feasible passes over the ground site in the given scenario time. It should be noted that Target 1 is in line of sight with the ground site at t_0, making that pass an infeasible choice for the cooperative inspection.

The start times and duration of each targets' passes over the ground site can be seen in Figure 5.2. Note that all orbits are circular and share the same right ascension of the ascending node. Additionally, the chaser's initial orbit is defined, but its initial position on that orbit is a function of the optimization variable, $u^c(t_1)$.

For comparison purposes, consider a two-burn combined plane-change transfer from the chaser's initial orbit to geostationary orbit without the requirement of an en-route

Table 5.1: Initial conditions of the chaser and targets

	a (km)	i (^{o})	$u(t_0)$ (^{o})	# passes
Chaser	6578.14	55	–	–
Target 1	26,561.76	55	0	2
Target 2	7378.14	55	0	14
Target 3	6878.14	55	0	14

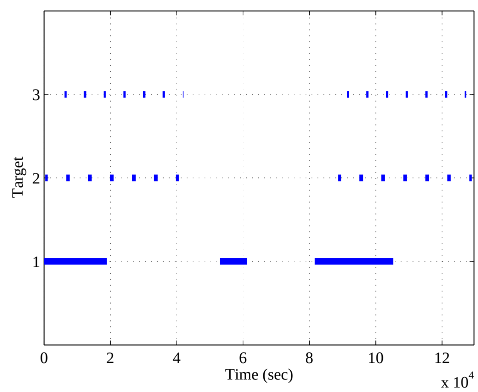


Figure 5.2: Target pass times for the three target GTMEI

inspection. The optimal solution for such a transfer can be found according to simple two body orbital mechanics. It requires a plane change of 2.86° at the first burn and the associated cost is 4.93944 km/s. For ease of comparison, all further costs are normalized by this value.

5.6.1.1 Three Target Enumeration

The relatively small number of targets were chosen because they allowed for complete enumeration of the categorical variable space and provided an opportunity to evaluate the performance of various bi-level algorithms. Complete enumeration was accomplished by using a PSO to optimize the continuous variables for each target-pass combination. The PSO defined in Table 5.2 is based on algorithms developed in [25, 55, 81, 84, 93]. The

PSO optimized each target-pass combination 20 times. There were 30 possible target-pass combinations, resulting in a total of 600 optimizations. The angular variables, $u^c(t_1)$ and l_{GEO}, were encoded to preserve accuracy to the nearest hundredth of a radian. Similarly, the relative position variables preserved accuracy to one meter. The time variables preserved accuracy to the nearest second in order to facilitate faster evaluation of the elevation constraint on the chaser spacecraft.

Table 5.2: Inner-loop PSO settings

Swarm Size	300
Max Iterations	500
Cognitive Parameter	2.09
Social Parameter	2.09
Constriction Factor	0.656295
Tolerance	1e-6 km/s

The ten lowest cost solutions found using complete enumeration of the solution space are shown in Table 5.3. Note that all ten require approximately 2% more ΔV than the optimal LEO-GEO transfer without en-route inspection. Additionally, all ten require the chaser to inspect during one of Target three's passes over the ground site. In fact, the top 137 solutions found during enumeration all required the chaser to inspect one of Target three's passes. The lowest cost solutions found for Targets one and two were $\bar{J} = 1.34875$ and $\bar{J} = 1.04267$, respectively. These ranked 203 and 138, respectively, of all solutions found during enumeration. Solving the inner loop problem required an average of 117,600 cost function evaluations for each target-pass combination. This implies that it would take approximately 3.52 million cost function evaluations to generate a single solution for each target-pass combination.

Table 5.3: Best three target costs found by enumeration

Rank	Satellite	Pass	$\bar{J}$
1	3	7	1.01730
2	3	7	1.01753
3	3	14	1.01754
4	3	14	1.01757
5	3	14	1.01773
6	3	7	1.01783
7	3	14	1.01785
8	3	7	1.01786
9	3	7	1.01790
10	3	11	1.01790

Enumeration of the categorical variable space provided further insight into the solution space of the three target GTMEI. First, it is important to note that several target-pass combinations yielded no feasible solutions after 20 PSO runs, while no target-pass combination yielded both feasible and infeasible solutions. Figure 5.3 depicts the topography of the categorical variable space where a normalized cost of 2 indicates an infeasible target-pass combination. Note there are only 13 feasible target-pass combinations for this example.

The performance of the PSO with respect to the feasible target-pass combinations is also insightful. Each feasible target-pass combination yielded several locally-optimal solutions, which is consistent with the stochastic nature of the PSO. The vast majority of feasible solutions yielded costs that were competitive with the best solution found. Specifically, half of the feasible solutions were within one percent of the best solution found, while three quarters of the feasible solutions were within ten percent of the lowest

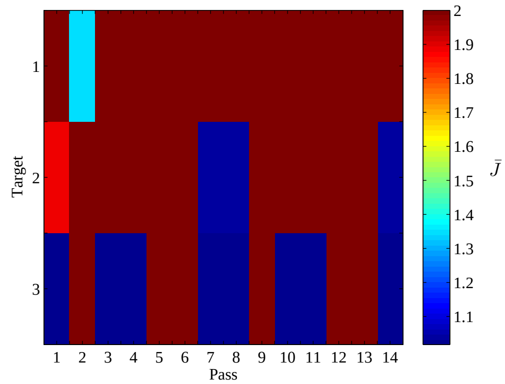


Figure 5.3: Characterization of three target categorical variable space

cost solution. Further, no feasible target-pass combination took more than 26 infeasible iterations to generate a feasible solution, implying infeasible target-pass combinations can be identified without requiring the maximum number of inner-loop iterations.

5.6.1.2 Three Target Hybrid Optimization

The results from complete enumeration of the three target problem led to the implementation of four bi-level HOC algorithms. Two of the bi-level algorithms employed an outer-loop PSO, while the other two employed an outer-loop GA. Both types of outer-loop optimizers are defined in Table 5.4. Each outer-loop optimizer employed a repository which prevents additional inner-loop optimization for a previously evaluated target-pass combination. Once the mth target's kth pass has been optimized by the inner-loop PSO, the inner-loop variables and cost are stored in the repository location corresponding to the specific combination of m and k. During subsequent outer-loop iterations, any previously-evaluated target-pass combination was assigned the appropriate inner-loop variables and cost stored in the repository. This approach was used previously in [88] and is appropriate to this problem because the locally optimal solutions identified through enumeration are competitive with the best cost found.

Table 5.4: Outer-loop optimization routines

PSO		GA	
Swarm Size	15	Population Size	15
Iterations	10	Generations	9
Cognitive Parameter	2.09	Selection Function	Binary tournament
Social Parameter	2.09	Crossover Function	Integer
Constriction Factor	0.656295	Elite members	1
		Crossover Rate	80%

Additionally, two types of inner-loop optimizers were employed as part of the bi-level algorithms. The first inner-loop optimizer was a PSO identical to the one defined in Table 5.2. The second inner-loop optimizer employed an identical PSO as the first, but assigned an infeasible cost to any target-pass combination which did not generate a feasible solution after the first 50 inner-loop iterations. This was designed to prevent superfluous inner-loop iterations for target-pass combinations that were likely to produce infeasible results.

Each outer-loop optimizer was paired with each inner-loop optimizer, resulting in four bi-level HOC algorithms identified as follows: genetic algorithm outer-loop with inner-loop particle swarm (GP), genetic algorithm outer-loop with inner-loop particle swarm employing infeasible cutoff (GPi), particle swarm outer-loop with inner-loop particle swarm (PP), particle swarm outer-loop with inner-loop particle swarm employing infeasible cutoff (PPi). Each bi-level routine was used to solve the three target problem 30 times. The inner-loop optimizations were parallelized on an Intel Xeon E5-2667 processor. The number of outer-loop iterations/generations were fixed to allow for more meaningful performance comparisons between the GA and PSO outer-loop solvers. The PPi algorithm converged to the lowest cost solution found by all algorithms. The associated cost was $\bar{J} = 1.01726$, and is hereafter referred to as the minimum for the three target problem.

The variable values of the minimum solution are shown in Table 5.5 along with the best solutions generated by the other bi-level algorithms, all of which were within two hundredths of one percent of the minimum.

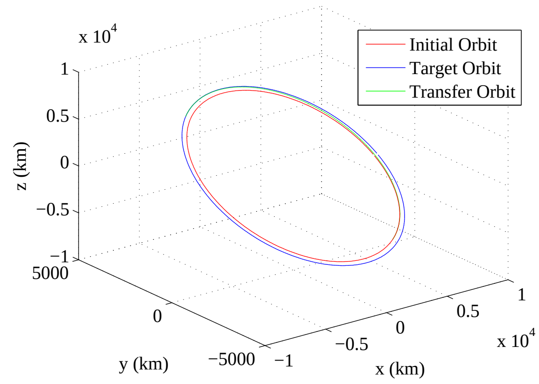
Table 5.5: Lowest cost solution for three target problem found by each bi-level algorithm

	m/k	$u^c(t_1)$	t_2	x_{CYL}^{enter}	x_{CYL}^{exit}	t_3	l_{GEO}	t_4	$\bar{J}$
PP	3/7	5.26	2893	2.247	1.000	46060	0	19073	1.01747
GP	3/7	5.29	2866	1.006	1.163	547	0	19153	1.01730
PPi	3/14	5.19	2845	1.155	1.276	45955	0	19151	1.01726
GPi	3/7	5.33	2831	1.000	1.297	546	0	19164	1.01730

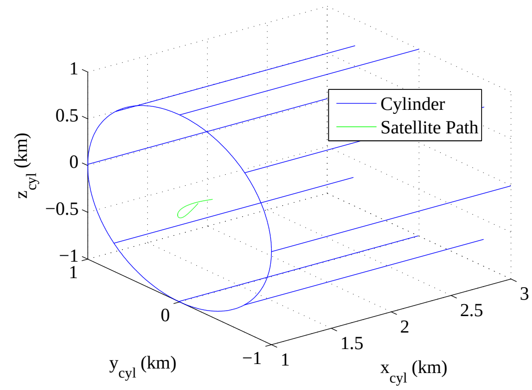
The chaser's path for the duration of the maneuver sequence corresponding to the minimum solution is shown in Figure 5.4. Figure 5.4(a) illustrates the chaser's maneuver from its initial orbit to the cooperative inspection segment, Figure 5.4(b) shows the chaser's path in the rotating cylinder frame during the cooperative inspection, and Figure 5.4(c) shows the chaser's path from the relative motion phase to GEO.

The performance of each algorithm with respect to the metrics are shown in Table 5.6. The PPi provided the most consistent cost performance and the greatest computational benefit to complete enumeration. Additionally, the PPi required an average of $607,000$ cost function evaluations, which are one fifth as many as would be required to enumerate the problem space. The worst solution found by any algorithm had a normalized cost of $\bar{J} = 1.01919$, which was within 0.2% of the minimum.

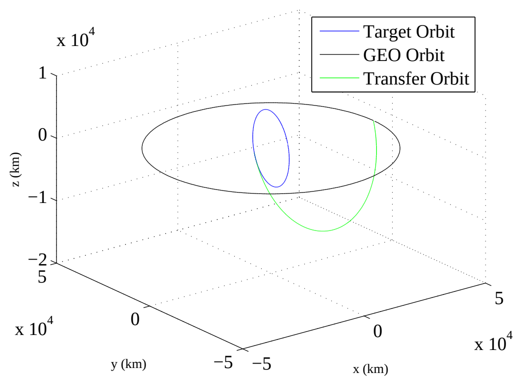
Figure 5.5 shows the performance of each bi-level algorithm with respect to cost and the number of cost function evaluations required for convergence, with the best results for each category highlighted in bold text. Figure 5.5(a) shows the performance of each bi-level algorithm with respect to the minimum cost found for the three target problem,



(a) Chaser transfer orbit to inspection in the inertial frame



(b) Chaser path in the cylinder frame during inspection



(c) Chaser transfer orbit to geostationary orbit after inspection in the inertial frame

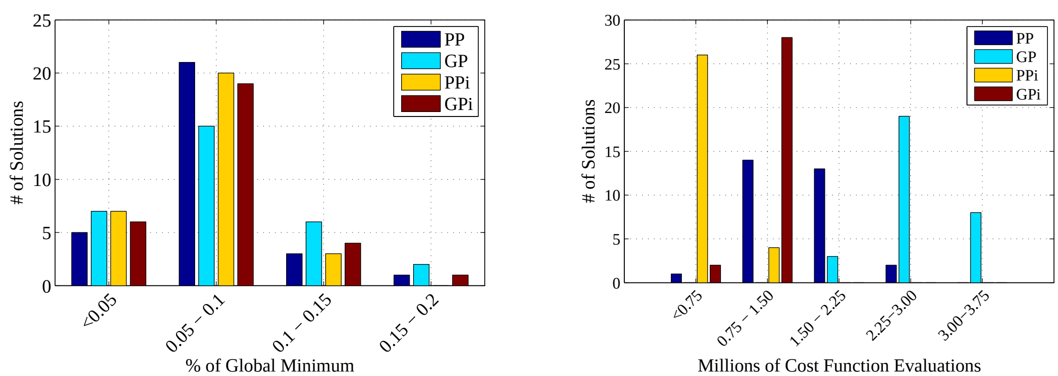
Figure 5.4: Path of chaser corresponding to the optimal three target GTMEI

$\bar{J}$ = 1.01726. Figure 5.5(b) shows the number of cost functions evaluations required for each hybrid algorithm to converge to a solution. Note that all bi-level algorithms require fewer cost function evaluations than what would be required for complete enumeration.

All bi-level algorithms provided similar cost performance with respect to the minimum solutions found. The PPi and GPi, however, generate these solutions with fewer cost function evaluations than were required using the other methods. Further, the

Table 5.6: Bi-level algorithm performance comparison

Metric		PP	GP	PPi	GPi
Cost	$\bar{J}_{min}$	1.01747	1.01730	**1.01726**	1.01730
	$\bar{J}_{max}$	1.01881	1.01903	**1.01838**	1.01919
	$\bar{J}_{mean}$	1.01797	1.01799	**1.01784**	1.01798
	$\sigma_{\bar{J}}$	**0.00031**	0.00043	**0.00031**	0.00038
Millions of Cost Function Evaluations	f_{min}	0.677	1.730	**0.277**	0.624
	f_{max}	2.329	3.351	**0.979**	1.308
	f_{mean}	1.512	2.744	**0.607**	0.942
	σ_f	0.459	0.358	0.150	**0.142**



(a) Cost performance as percentages of the minimum (b) Cost function evaluations required for convergence

Figure 5.5: Bi-level algorithm performance data for three target problem

computational benefit of the GPi and PPi are expected to increase as the number of target-pass combinations increase.

5.6.2 *Fifteen Target Problem*

The results of the three target problem led to implementing the PPi algorithm to optimize a larger, fifteen target problem. The outer-loop swarm size was increased to 20 particles to account for the larger categorical variable space. Additionally, the maximum number of iterations was increased to 50 and an additional stopping criteria was added such that the optimization terminated if the objective value didn't change for ten consecutive iterations. The inner-loop parameters remain identical to those shown in Table 5.2. The initial chaser orbit, ground site, elevation constraints and limits on all non-pass dependent variables were identical to those defined in the three target problem. Each target satellite began in a circular orbit with orbital elements uniformly randomized on intervals of [6878 7378] km for semi-major axis, [28.5° 55°] for inclination, [$-5°$ $-5°$] for right ascension of the ascending node, and [0° 360°] for initial argument of latitude. The targets' defining orbital elements are shown in Table 5.7 along with the number of passes over the ground site. Figure 5.6.2 shows the line of s contact times for each of the fifteen targets with the ground station for the time interval from t_0 to t_{max}. The PPi was used to solve the fifteen target problem 30 times on the same workstation utilized for the three target problem.

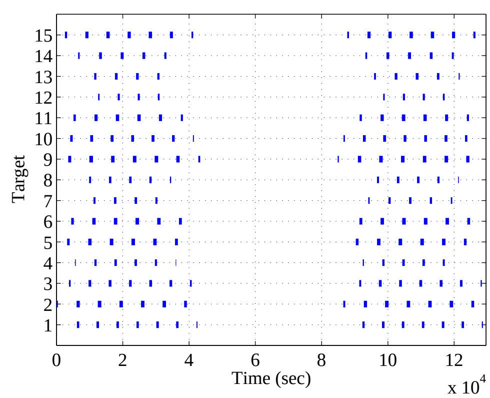


Figure 5.6: Target pass times for the fifteen target GTMEI

Table 5.7: Target satellites' initial conditions

	a (km)	i (°)	Ω (°)	u (°)	passes
Target 1	6931.33	53.99	2.75	1.67	14
Target 2	7286.65	51.52	359.00	30.40	14
Target 3	7007.94	49.70	4.11	155.31	14
Target 4	6968.92	35.49	356.36	52.39	11
Target 5	7312.65	43.86	356.45	197.95	12
Target 6	7304.52	44.98	0.13	126.34	12
Target 7	7078.90	30.51	356.23	86.37	9
Target 8	6969.95	34.86	355.50	150.22	10
Target 9	7329.36	53.54	359.89	176.71	14
Target 10	7046.86	52.35	356.11	132.93	14
Target 11	7268.13	38.83	359.04	87.01	12
Target 12	6926.23	32.00	4.56	339.14	8
Target 13	7165.60	30.08	358.53	84.52	9
Target 14	7288.60	28.91	356.69	15.49	10
Target 15	7202.56	47.89	359.51	233.19	14

The PPi algorithm converged to solutions for five different target-pass combinations of a possible 177, resulting in 22 distinct solutions in the course of the 30 runs. The best and worst solutions for each target-pass combination are shown in Table 5.8, along with their respective rank out of the 30 runs. Once again, the GTMEI can be achieved for only a fraction more ΔV than what is required to complete a transfer from the initial orbit to geostationary orbit. Additionally, Figure 5.6.2 shows the lowest normalized cost found during the course of this research for each target-pass combination. All infeasible combinations were assigned $\bar{J} = 2$.

Table 5.8: Best/worst solution for each target-pass combination converged upon by the PPi

Rank	m/k	$u^c(t_1)$	t_2	x_{CYL}^{enter}	x_{CYL}^{exit}	t_3	l_{GEO}	t_4	$\bar{J}$
1	9/1	3.10	3236	2.856	1.940	1885	6.28	19173	1.04825
30	9/1	4.07	2340	1.956	2.774	1840	0.00	19660	1.07353
10	9/9	3.10	3249	3.000	3.000	1819	6.28	19174	1.04892
15	9/9	3.10	3249	3.000	3.000	1774	0.00	19660	1.04901
16	9/8	3.14	3332	1.138	1.025	2293	6.28	19155	1.05417
19	9/8	3.14	3332	1.000	3.000	2290	6.28	19221	1.05467
18	1/14	5.00	3117	3.000	3.000	480	0.05	19551	1.05450
24	1/14	5.00	3117	3.000	3.000	32294	3.18	18097	1.06009
20	1/7	4.99	3126	1.385	3.000	26399	3.19	19306	1.05470
27	1/7	5.00	3119	2.094	1.841	37916	3.19	19214	1.05504

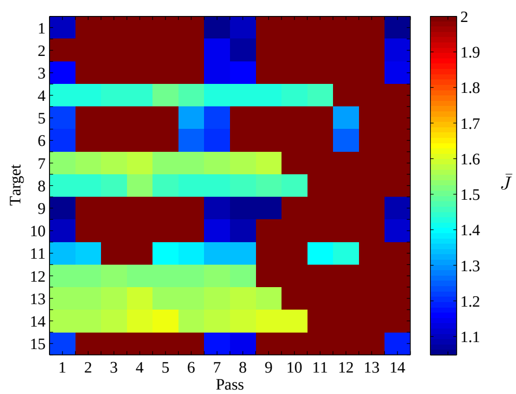
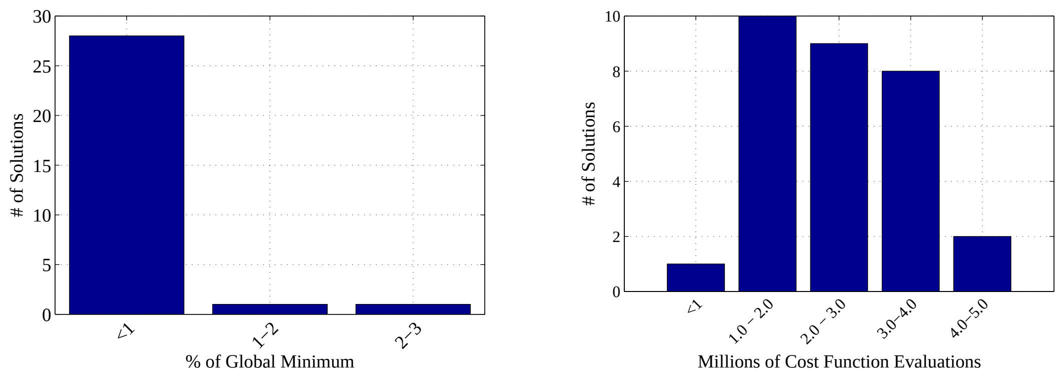


Figure 5.7: Characterization of fifteen target categorical variable space

As expected, the algorithm converged to multiple locally optimal solutions for each target-pass combination. The best and worst solutions found over the course of 30 runs occurred on the ninth target's first pass; the associated costs were $\bar{J} = 1.04825$ and $\bar{J} = 1.07353$, respectively, resulting in a difference of only 2.4%. Figure 5.8(a) shows the cost performance of the bi-level PPi, with respect to the minimum cost found. Similarly, Figure 5.8(b) shows the number of cost function evaluations required for convergence.



(a) Solutions as percentages of the minimum

(b) Cost function evaluations required for convergence

Figure 5.8: Fifteen target performance data

As expected, the bi-level PPi algorithm provides an even greater benefit with respect to cost function evaluations required for convergence. The PPi required an average of 2.41 million cost function evaluations to converge to a solution for the fifteen target problem. Recall that enumeration of the three target problem required 117,600 cost functions evaluations for each target pass combination. As a result, enumerating the fifteen target problem would require approximately 20.82 million cost function evaluations. This implies that the PPi can generate a solution nearly nine times faster than enumeration.

5.7 Conclusions

This work defined the geostationary transfer maneuver with en-route inspection problem. This problem is designed to optimize a transfer for a space situational awareness platform from low Earth orbit to geostationary orbit, during which the platform performs a close-proximity inspection with one of several uncharacterized objects in cooperation with a designated ground site. The cooperative inspection requires the maneuvering satellite to stay within a cylindrical volume defined by the target and ground site for the duration of the object's pass over the ground-based observer. The cylindrical volume is oriented such that the long axis of the cylinder is aligned with the vector connecting the ground site to the object.

The geostationary transfer maneuver with en-route inspection problem is formulated as a hybrid optimal control problem and solved using several bi-level algorithms. The outer-loop algorithm optimized the categorical variables: the target and pass combination to perform the en-route inspection. The inner-loop optimized the continuous variables associated with designated target-pass combinations. The bi-level algorithms employed either a genetic algorithm or a particle swarm optimization algorithms as the outer-loop solver and employed inner-loop particle swarm optimization algorithms. Two types of inner-loop algorithms were employed: the first was a particle swarm optimization algorithm while the second was a particle swarm optimization algorithm that assigned an infinite cost to any target-pass combination that yielded infeasible results after a finite number of inner-loop iterations. Each inner-loop optimizer was paired with each outer-loop algorithm, resulting in four bi-level optimizers. A three target geostationary transfer maneuver with en-route inspection problem was used to evaluate the performance of the bi-level variants in comparison to one another and complete enumeration for the categorical variable space. The results of the three target problem showed that all variants converged to near optimal solutions. The results further led to the implementation of a bi-level algorithm which employed an outer-loop particle swarm and inner-loop particle swarm with infeasible cutoff, which converged to near optimal solutions for a fifteen target problem. Results for the two example problems indicate that the bi-level algorithm particle swarm outer-loop paired with particle swarm inner-loop with infeasible cutoff provides additional computational efficiency as the size of the categorical space increases while still generating near optimal results. The three and fifteen target example problems showed that the en-route inspection can be accomplished with the addition of a fraction of the delta-velocity required for a transfer from low Earth orbit to geostationary orbit. As a result, the geostationary transfer maneuver with en-route inspection problem can be considered as a potential method to enhance space-based space situational awareness at low and geostationary orbits.

VI. Conclusions and Contributions

6.1 Impulsive Responsive Theater Maneuvers

The first contribution of this research was the design and optimization of impulsive responsive theater maneuvers (RTMs) that enable resiliency by altering a spacecraft's arrival conditions over a potentially hazardous geographic region. Several particle swarm optimization (PSO) algorithms and a genetic algorithm (GA) were shown to generate optimal solutions for a single pass RTM scenario. These results demonstrated the utility of evolutionary algorithms (EAs) in the optimization of impulsive resiliency maneuvers. Further, the performance of each algorithm was evaluated based on convergence percentage to the global minimum as well as computational speed. The performance characterization led to the development of an optimization strategy utilizing a global version of the PSO that consistently generated optimal solutions in only minutes of computational time.

This optimization strategy was applied to single, double, and triple pass RTMs with varying initial conditions and maneuver constraints and was shown to consistently produce optimal maneuvers for each. The robustness of the technique with respect to impulsive RTMs implies that EAs have the potential to enable the autonomous optimization of impulsive resiliency maneuvers. This potential results from the consistent convergence performance of the PSO and the fact that it does not require an initial guess to generate a solution. Further, the impulsive RTM definition and solution algorithm can be applied to more complex and longer scenarios.

6.2 Continuous Thrust Responsive Theater Maneuvers

The second major contribution of this research was the extension of the RTM to include continuous, low-thrust maneuvers, which was accomplished with the application of a two-stage optimization algorithm. The algorithm leveraged the strengths of a PSO and a direct orthogonal collocation (DOC) method with a nonlinear programming (NLP) problem solver; the PSO did not require an initial guess and provided a broad search capability, while DOC provided a method to accurately model a large number of control parameters impacting the system dynamics.

The two-stage optimization routine was applied to single, double, and triple pass RTM scenarios with varying initial conditions and maneuver constraints and shown to consistently generate solutions satisfying the analytical necessary conditions for an optimal control. The ability of the two-stage optimization algorithm to provide consistent convergence performance regardless of the initial conditions and maneuver constraints indicate its potential to aid in the autonomous generation of low-thrust resiliency maneuvers.

The low-thrust RTM research also demonstrated that resiliency maneuvers can be accomplished with a low-thrust engine in less than one orbit. Thus, mission planners have several propulsion options at their disposal when designing satellites to perform resiliency maneuvers. Additionally, as engine technology improves the low-thrust version of the RTM can provide provide significant propellant mass savings in comparison to the impulsive version. This savings could be used to extend mission life by adding additional fuel or to increase the payload capacity of a spacecraft designed for resiliency.

6.3 Geosynchronous Transfer Maneuvers with Cooperative En-Route Inspection

The final contribution of this research was the development of a technique to generate near-optimal trajectories for a new type of maneuver, called geostationary transfer maneuver with cooperative en-route inspection (GTMEI). GTMEIs are designed to improve space situational awareness (SSA) and require a maneuvering spacecraft to transfer from low Earth orbit (LEO) to geostationary orbit while performing an en-route inspection of one of several target satellites while the target is in line-of-sight contact with a designated ground location. They are a class of hybrid optimal control (HOC) problems, which consist of a combination of categorical and continuous variables.

Four separate bi-level HOC algorithms consisting of GA and PSO algorithms were shown to generate optimal and near optimal solutions to a simplified three target GTMEI. A bi-level HOC algorithm, particle swarm outer-loop with inner-loop particle swarm employing infeasible cutoff (PPi), was shown to provide significant computational savings over other explored bi-level algorithms. The PPi was then applied to a larger fifteen-target GTMEI problem and shown to provide significant computational benefit to complete enumeration of the solution space.

This research is significant because it shows that a relatively simple algorithm has the capability to generate near-optimal solutions to complex problems. The bi-level HOC algorithms developed in this research should provide even greater computational benefit for larger GTMEI scenarios. Further, the consistent performance of these algorithms in the solution of GTMEIs demonstrate their potential to enable the autonomous generation of to-be-developed resiliency maneuvers requiring HOC.

6.4 Overall Conclusion

This research defined a new set of maneuvers to enhance spacecraft resiliency through avoidance and provided several options for mission planners in their design. The maneuvers included both impulsive and continuous thrust options for altering a spacecraft's arrival conditions as they enter a potentially hostile geographic region on the earth. These maneuvers, each of which require only meters per second of ΔV, can be employed by mission planners to introduce uncertainty for ground-based tracking systems. As a result, these maneuvers provide a low-cost option for the enhancement of spacecraft resiliency. The methods presented in this dissertation lay the groundwork for future work in the autonomous design of resiliency maneuvers.

This research also demonstrated the effectiveness of a bi-level HOC algorithm in the optimization of the GTMEI problem, which enhances resiliency by introducing uncertainty to ground-based tracking algorithms. Additionally, the bi-level HOC algorithms developed herein generated near-optimal trajectories at much faster computational speeds than complete enumeration of the problem space. These savings are expected to increase as the complexity and size of the GTMEI scenarios increase.

The tools and techniques developed in this research demonstrated their effectiveness in producing optimal and near optimal RTMs and GTMEIs. The performance of these algorithms provide confidence that they can be applied to more complex RTM and GTMEI scenarios. More importantly, this research demonstrated the effectiveness of EAs and metaheuristics as enablers for autonomous resiliency maneuver generation for a variety of optimal trajectory problems including impulsive and continuous thrust trajectories as well as hybrid optimal control problems. As a result, these methods and algorithms can be applied to future resiliency maneuvers that have yet to be developed by mission planners.

6.5 Assumptions and Limitations

The algorithms developed in this research provide a foundation for the autonomous optimization of responsive resiliency maneuvers. There are, however, several simplifying assumptions that will limit their utility if not addressed. First, no consideration was given to additional spacecraft constraints such as power and duty cycle limitations on the propulsion system resulting from mission requirements. Such considerations add constraints to these problems and could limit the number, duration, or frequency of resiliency maneuvers.

Other critical assumptions made throughout this research were those leading to the two-body dynamics representing all spacecraft motion. Linearizing the equations of motion removes the need to perform computationally expensive numerical integration inside the EAs, which dramatically improves the speed of the algorithms. Higher fidelity models, which would be required to perform conjunction analysis, will increase the computational time of these algorithms to the point at which a spacecraft may no longer be able to maneuver every orbit.

Conjunction analysis presents a further limitation to autonomous maneuver generation. Specifically, conjunction analysis is historically controlled by a centralized location and requires significant computational resources. Any maneuver generated by an autonomous algorithm would require vetting by such an organization. A hypothetical scenario requiring resiliency maneuvers on every orbit would require significant resources on the part of the vetting organization, greatly reducing the autonomous nature of the maneuvers proposed in this dissertation.

6.6 Areas for Future Work

There are several areas in which this research can be continued which are listed below.

1. Quantify RTM effects on ground-based tracking performance.
    a. Determine how long it takes ground-based tracking systems to converge to an accurate post-maneuver orbit fit.
    b. Analyze the impact of maneuver size on tracking algorithm performance.
    c. Develop a maneuvering strategy to maximize the impact on tracking algorithm performance while minimizing ΔV.
2. Introduce additional complexity into the RTM problem.
    a. Quantify the impact of power system requirements and duty cycle on the RTM problem. Determine the implications of RTMs on satellite sub-system design.
    b. Develop a maneuvering strategy for multiple exclusion zone scenarios.
3. Apply hybrid optimal control algorithms to optimize RTM for a planned system with a dual impulsive and continuous-thrust propellant system.

4. Quantify GTMEI effects on ground-based tracking algorithms.

5. Develop and optimize RTMs and GTMEIs for multiple ground locations.

Appendix A: Derivation of Spherical Equations of Motion

Consider the spherical coordinate system in the perifocal frame shown in Figure 2.3, in which the gravitational force of the Earth is the only force acting on a spacecraft with mass m. The coordinates are specified as r and ψ, where r is the distance from the center of the coordinate frame and ψ is the angle measured from some reference axis.

The spacecraft has kinetic energy T as shown in Equation A.1, where $\boldsymbol{v}$ is the velocity vector of the spacecraft.

$$T = \frac{1}{2}m\left(\boldsymbol{v}\cdot\boldsymbol{v}\right) = \frac{1}{2}m\left(\dot{r}^2 + r^2\dot{\psi}^2\right) \tag{A.1}$$

Similarly, the spacecraft has potential energy V shown in Equation A.2, where μ is the gravitational parameter of the Earth.

$$V = -\frac{\mu}{r}m \tag{A.2}$$

As a result, the Lagrangian can be written as

$$\mathscr{L} = T - V = \frac{1}{2}m\left(\dot{r}^2 + r^2\dot{\psi}^2\right) + \frac{\mu}{r}m. \tag{A.3}$$

The resulting momenta are expressed as shown in Equations A.4.

$$\begin{aligned} p_r &= \tfrac{\partial\mathscr{L}}{\partial\dot{r}} = m\dot{r} \\ p_\phi &= \tfrac{\partial\mathscr{L}}{\partial\dot{\psi}} = mr^2\dot{\psi} \end{aligned} \tag{A.4}$$

Equation A.4 can be rearranged to provide expressions for $\dot{r}$ and $\dot{\psi}$.

$$\begin{aligned} \dot{r} &= \dot{q}_r = \tfrac{p_r}{m} \\ \dot{\psi} &= \dot{q}_\phi = \tfrac{p_\psi}{mr^2} \end{aligned} \tag{A.5}$$

The system Hamiltonian is defined as $\mathscr{H} = \sum p_i\dot{q}_i - \mathscr{L}$. After some arithmetic, this results in Equation A.6.

$$\mathscr{H} = \frac{1}{2}\frac{p_r^2}{m} + \frac{1}{2}\frac{p_\psi^2}{mr^2} - \frac{\mu}{r}m \tag{A.6}$$

The rate of change of the momenta can be expressed as shown in Equation A.7.

$$\begin{aligned} \dot{p}_r &= -\frac{\partial \mathscr{H}}{\partial r} = \frac{p_\psi^2}{mr^2} + \frac{\mu}{r^2}m \\ \dot{p}_\psi &= -\frac{\partial \mathscr{H}}{\partial \psi} = 0 \end{aligned} \tag{A.7}$$

Taking the time derivative of Equation A.4 and substituting the results into Equation A.7 provides expressions for $\ddot{r}$ and $\ddot{\psi}$.

$$\begin{aligned} \ddot{r} &= \frac{r^2\dot{\psi}^2}{r} - \frac{\mu}{r^2} \\ \ddot{\psi} &= -\frac{2\dot{r}\dot{\psi}}{r^2} \end{aligned} \tag{A.8}$$

Now choose four states, r, ψ, V_r, and V_ψ, where V_r, and V_ψ are defined by Equation A.9.

$$\begin{aligned} V_r &= \dot{r} \\ V_\psi &= r\dot{\psi} \end{aligned} \tag{A.9}$$

The time rates of change of the states are shown in Equation A.10.

$$\begin{aligned} \dot{r} &= V_r \\ \dot{\psi} &= \frac{V_\psi}{r} \\ \dot{V}_r &= \ddot{r} \\ \dot{V}_\psi &= \dot{r}\dot{\psi} + r\ddot{\psi} \end{aligned} \tag{A.10}$$

Substituting the expressions for $\ddot{r}$ and $\ddot{\psi}$ from Equation A.8 into Equation A.10 provides an alternative representation of the equations of motion.

$$\begin{aligned} \dot{r} &= V_r \\ \dot{\psi} &= \frac{V_\psi}{r} \\ \dot{V}_r &= \frac{V_\psi^2}{r} - \frac{\mu}{r^2} \\ \dot{V}_\psi &= -\frac{V_r V_\psi}{r} \end{aligned} \tag{A.11}$$

Appendix B: Equations of Motion in the Local Vertical, Local Horizontal Frame

The local vertical, local horizontal coordinate frame (RSW) frame is typically used as the frame of reference when analyzing the motion of a satellite, called the chaser, with respect to a second satellite, called the target. In such cases, the target serves as the origin of the RSW frame and the relative position and velocity vectors of the chaser, $\boldsymbol{r}_{RSW}$ and $\boldsymbol{v}_{RSW}$ respectively, are given by Equation B.1.

$$\begin{aligned} \boldsymbol{r}_{RSW} &= x\hat{R} + y\hat{S} + z\hat{W} \\ \boldsymbol{v}_{RSW} &= \dot{x}\hat{R} + \dot{y}\hat{S} + \dot{z}\hat{W} \end{aligned} \tag{B.1}$$

The motion of the chaser relative to the target can be found according to Newton's second law and the universal law of gravitation. It is possible to derive the equations of motion shown in Equation B.2 using the following simplifying assumptions

1. the target and chaser are in nearly circular orbits
2. the distance between the target and chaser is much smaller than the semimajor axis of the target orbit

These equations provide analytical expressions to determine the chaser's position and velocity relative to the target as functions of time. A subscript of zero designates the chaser's relative position or velocity at the initial time t_0 and the variable t represents the amount of time that has passed since t_0. The mean motion of the target is n. A complete derivation of these equations can be found in [36, 389-393].

$$
\begin{aligned}
x(t) &= \tfrac{\dot{x}_0}{n}\sin(nt) - \left(3x_0 + \tfrac{2\dot{y}_0}{n}\right)\cos(nt) + \left(4x_0 + \tfrac{2\dot{y}_0}{n}\right) \\
y(t) &= \left(6x_0 + \tfrac{4\dot{y}_0}{n}\right)\sin(nt) + \tfrac{2\dot{x}_0}{n}\cos(nt) - (6nx_0 + 3\dot{y}_0)\,t + \left(y_0 - \tfrac{2\dot{x}_0}{n}\right) \\
z(t) &= z_0\cos(nt) + \tfrac{\dot{z}_0}{n}\sin(nt) \\
\dot{x}(t) &= \dot{x}_0\cos(nt) + (3nx_0 + 2\dot{y}_0)\sin(nt) \\
\dot{y}(t) &= (6nx_0 + 4\dot{y}_0)\cos(nt) - 2\dot{x}_0\sin(nt) - (6nx_0 + 3\dot{y}_0) \\
\dot{z}(t) &= -z_0 n\sin(nt) + \dot{z}_0\cos(nt)
\end{aligned}
\tag{B.2}
$$

An equivalent but alternative formulation [46] can be found by scaling t by the orbital period of the target satellite. This results in a scaled time $\tilde{t} = \frac{n}{2\pi}t$. The relative position components $(\tilde{x}, \tilde{y}, \tilde{z})$ are identical to their counterparts in the unscaled frame. The relative velocity components $\left(\dot{\tilde{x}}, \dot{\tilde{y}}, \dot{\tilde{z}}\right)$, however, are all scaled by P_{tgt}. This transformation leads to the equations of motion shown in Equation B.3. A derivation can be found in [46].

$$
\begin{aligned}
\tilde{x}(t) &= \tfrac{1}{2\pi}\dot{\tilde{x}}_0\sin(2\pi\tilde{t}) - \left(3\tilde{x}_0 + \tfrac{1}{\pi}\right)\dot{\tilde{y}}_0\cos(2\pi\tilde{t}) + \left(4\tilde{x}_0 + \tfrac{1}{\pi}\dot{\tilde{y}}_0\right) \\
\tilde{y}(t) &= \left(6\tilde{x}_0 + \tfrac{2}{\pi}\dot{\tilde{y}}_0\right)\sin(2\pi\tilde{t}) + \tfrac{1}{\pi}\dot{\tilde{x}}_0\cos(2\pi\tilde{t}) - \left(12\pi\tilde{x}_0 + 3\dot{\tilde{y}}_0\right)\tilde{t} + \left(\tilde{y}_0 - \tfrac{1}{\pi}\dot{\tilde{y}}_0\right) \\
\tilde{z}(t) &= \tilde{z}_0\cos(2\pi\tilde{t}) + \tfrac{1}{2\pi}\dot{\tilde{z}}_0\sin(2\pi\tilde{t}) \\
\dot{\tilde{x}}(t) &= \dot{\tilde{x}}_0\cos(2\pi\tilde{t}) + \left(6\pi\tilde{x}_0 + 2\dot{\tilde{y}}_0\right)\sin(2\pi\tilde{t}) \\
\dot{\tilde{y}}(t) &= \left(12\pi\tilde{x}_0 + 4\dot{\tilde{y}}_0\right)\cos(2\pi\tilde{t}) - 2\dot{\tilde{x}}_0\sin(2\pi\tilde{t}) - \left(12\pi\tilde{x}_0 + 3\dot{\tilde{y}}_0\right) \\
\dot{\tilde{z}}(t) &= -2\pi\tilde{z}_0\sin(2\pi\tilde{t}) + \dot{\tilde{z}}_0\cos(2\pi\tilde{t})
\end{aligned}
\tag{B.3}
$$

Appendix C: Design of Experiments on Particle Swarm Optimization Parameters

The following are results from a design of experiments (DOE) approach to determine the ideal PSO parameters to optimize single pass impulsive RTM problems. The goal was to determine a set of PSO parameters that provided consistent convergence to the global minimum, eliminated all solutions not at least locally optimal, and provided fast computational speed, thus enabling autonomy.

The two variable single pass RTM defined in Equation 4.7 of Chapter 3 was used as the test case because the optimal results were found using a simple parameter search. Additionally, the problem is known to have a locally optimal solution only slightly larger than globally optimal cost: 4.122 m/sec compared to 4.083.

A two parameter DOE study investigated the effect of swarm size and $c = c_1 = c_2$ on the performance of the PSO in the solution of the single pass RTM defined in 4.7.

A PSO algorithm utilizing each set of bounds defined by [107] was run twenty times. Each design was evaluated according to the minimum, maximum, and average number of iterations required for convergence. Additionally, each design was evaluated according to cost function performance, which was measured in convergence percentage to the global minimum, local minimum, and other solutions.

The initial bounds on each variable were chosen based on the literature and are defined in Equation C.1. It should be noted that the PSO algorithm employed utilized a constriction factor, which requires $c_1 + c_2 > 4$.

$$\begin{aligned} 2 &< c \leq 3.5 \\ 20 &\leq s \leq 200 \end{aligned} \tag{C.1}$$

The design space and performance results according to each combination of parameters is seen in Table C.1. The top three performing algorithms with respect to percent convergence to the global minimum and average number of iterations required are identified by *, **, and ***, respectively. The worst three algorithms with respect to percent convergence to the global minimum and average number of iterations required are identified by *, **, and ***, respectively.

Table C.1: Performance data for initial set of DOE bounds

s	c	min	max	avg	global	local	other
200	2.47	114	1000	538.80	75***	25	0
65	2.09	111	1000	341.95	80**	20	0
99	2.19	68	902	232.95**	85*	15	0
133	2.28	78	1000	402.95	70	30	0
189	3.13	1000	1000	1000.00***	60	40	0
76	3.50	539	1000	971.65**	35	35	30
54	2.94	150	1000	580.90	50	40	10
178	2.84	275	1000	810.55	50	30	20
110	2.75	150	807	524.90	70	30	0
20	2.03	30	1000	339.30	15*	30	55
155	3.41	1000	1000	1000.00***	30***	65	05
121	3.31	1000	1000	1000.00***	50	45	05
88	3.22	464	1000	895.05**	25**	65	10
31	2.38	44	273	120.55*	65	35	0
166	2.56	133	1000	599.00	70	25	5
43	2.66	76	1000	247.95***	45	55	0

The results from the initial study led to a new set of bounds of the variables, defined in Equation C.2. The results are shown in Table C.2. Notice there are several combinations which lead to solutions that are not at least locally optimal.

$$
\begin{aligned}
2.05 &\leq c \leq 3 \\
30 &\leq s \leq 150
\end{aligned}
\tag{C.2}
$$

Table C.2: Performance data for second set of DOE bounds on two parameter study

		Iterations			Convergence Percentage		
s	c	min	max	avg	global	local	other
150	2.35	143	1000	488.80	75**	25	0
60	2.11	81	1000	298.15	70***	30	0
83	2.17	62	739	182.85*	50	50	0
105	2.23	96	655	242.60	70***	30	0
143	2.76	204	1000	623.90***	30*	65	5
68	3.00	270	1000	563.45	35**	65	0
53	2.64	123	704	319.10	45	55	0
135	2.58	113	1000	567.55	55	35	10
90	2.53	137	1000	319.15	55	45	0
30	2.70	58	1000	335.65	45	50	5
120	2.94	158	1000	880.70*	65	35	0
98	2.88	251	1000	651.10**	45	50	5
75	2.82	163	1000	487.70	55	45	0
38	2.29	44	683	183.10**	55	45	0
113	2.05	160	702	311.70	80*	20	0
128	2.41	95	908	313.80	65	35	0
45	2.47	62	784	214.20***	40***	60	0

The results led to a new set of bounds, defined in Equation C.3. The results are shown in Table C.3. Notice there are still several combinations which lead to solutions that are not at least locally optimal.

$$\begin{aligned} 2.05 \quad \leq \quad c \quad \leq \quad 2.5 \\ 30 \quad \leq \quad s \quad \leq \quad 120 \end{aligned} \tag{C.3}$$

Table C.3: Performance data for third set of DOE bounds on two parameter study

s	c	min	max	avg	global	local	other
120	2.19	70	1000	398.65^{**}	65	35	0
53	2.08	94	668	230.15	$85^{\star}$	15	0
69	2.11	80	773	219.85	65	35	0
86	2.13	79	553	223.30	$70^{\star\star\star}$	30	0
114	2.29	78	1000	262.75	45^{***}	50	0
58	2.50	63	1000	241.80	45^{***}	55	0
47	2.33	46	639	$194.00^{\star\star\star}$	45	55	0
109	2.30	73	629	215.70	65	35	0
75	2.28	102	1000	247.35	$70^{\star\star\star}$	30	0
30	2.36	45	839	$161.05^{\star}$	55	45	0
98	2.47	79	1000	352.40^{***}	50	40	10
81	2.44	97	1000	245.95	35^{**}	60	5
64	2.42	62	1000	271.65	45^{***}	50	5
36	2.16	57	1000	208.55	45^{***}	55	0
92	2.05	154	999	436.80^{*}	$80^{\star\star}$	20	0
103	2.22	60	682	211.50	$70^{\star\star\star}$	30	0
41	2.25	51	549	$162.60^{\star\star}$	30^{*}	70	0

The results led to a new set of bounds, defined in Equation C.4. The results are shown in Table C.4. Notice all combinations of parameters yield at least locally optimal results.

$$\begin{aligned} 2.05 \quad \leq \quad c \quad \leq \quad 2.3 \\ 30 \quad \leq \quad s \quad \leq \quad 90 \end{aligned} \tag{C.4}$$

Table C.4: Performance data for fourth set of DOE bounds on two parameter study

s	c	min	max	avg	global	local	other
56	2.08	77	761	217.85	$70^{\star\star\star}$	30	0
45	2.07	108	1000	280.15^{***}	$80^{\star}$	20	0
56	2.08	92	580	185.95	50^{***}	50	0
68	2.10	90	566	207.30	55	45	0
86	2.24	60	979	251.30	$75^{\star\star}$	25	0
49	2.30	46	694	153.55	40^{*}	60	0
41	2.21	55	220	$108.40^{\star}$	65	35	0
83	2.19	65	1000	307.35^{**}	$70^{\star\star\star}$	30	0
60	2.18	66	908	212.60	60	40	0
30	2.20	46	728	$178.85^{\star\star\star}$	45^{**}	55	0
75	2.28	51	694	259.75	$70^{\star\star\star}$	30	0
64	2.27	47	1000	248.55	50^{***}	50	0
53	2.25	54	942	199.15	$75^{\star\star}$	25	0
34	2.11	70	996	226.45	65	35	0
71	2.05	124	1000	354.30^{*}	60	40	0
79	2.14	72	1000	232.30	$75^{\star\star}$	25	0
38	2.16	61	826	$161.00^{\star\star}$	40^{*}	60	0

The results led to a new set of bounds, defined in Equation C.5. The results are shown in Table C.5. Notice all combinations of parameters yield at least locally optimal results.

$$\begin{aligned} 2.07 \;\leq\; &c \;\leq\; 2.25 \\ 30 \;\leq\; &s \;\leq\; 80 \end{aligned} \tag{C.5}$$

Table C.5: Performance data for fifth set of DOE bounds on two parameter study

s	c	min	max	avg	global	local	other
80	2.13	74	880	269.35^{***}	$70^{\star\star\star}$	30	0
43	2.08	96	966	298.55^{**}	65	35	0
52	2.09	108	1000	327.60^{*}	$75^{\star\star}$	25	0
61	2.10	77	546	$177.65^{\star\star\star}$	$80^{\star}$	20	0
77	2.21	76	1000	238.65	45^{*}	55	0
46	2.25	44	835	194.80	45^{*}	55	0
39	2.18	53	1000	$174.35^{\star\star}$	55^{***}	45	0
74	2.17	70	722	191.25	60	40	0
55	2.16	73	538	185.90	50^{**}	50	0
30	2.19	53	256	$116.60^{\star}$	55^{***}	45	0
68	2.24	61	646	216.25	50^{**}	50	0
58	2.23	53	892	227.55	60	40	0
49	2.22	51	1000	237.50	60	40	0
33	2.12	69	1000	196.40	60	40	0
64	2.07	114	617	223.10	65	35	0
71	2.14	76	720	199.35	65	35	0
36	2.15	68	1000	192.40	55^{***}	45	0

The results from this study led to the conclusion that to the following bounds on bounds on c and s. It is expected that these bounds provide the best balance between convergence and computational speed for the single pass RTM problems.

$$\begin{aligned} 2.09 &\leq c \leq 2.13 \\ 30 &\leq s \leq 60 \end{aligned} \tag{C.6}$$

Appendix D: Code for Impulsive Responsive Theater Maneuvers

D.1 Single Pass RTMs

D.1.1 Single Pass RTM Data Script

```
t0 = 0;
GMST0 = 0;
latlim = [-10 10]*pi/180;
longlim = [-50 -10]*pi/180;

wgs84data
global MU
r0vec = [6800 7300;0 0;0 0];
v0vec = [0 0;5.41376581448788 sqrt(MU/7300)/sqrt(2);5.41376581448788
    sqrt(MU/7300)/sqrt(2)];
swarm = 30;
iter = 1000;
aevec = [50 60 70 80 90 100 110 120 130 140 150];
bevec = [5 6 7 8 9 10 11 12 13 14 15];
Rmaxvec = [6850 7350];
Rminvec = [6750 7250];
prec = [2;5;16];

for k = 1:1

    r0 = r0vec(:,k);
    v0 = v0vec(:,k);
    Rmax = Rmaxvec(k);
    Rmin = Rminvec(k);
    [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0,v0);
    period = 2*pi*sqrt(a^3/MU);
```

```
state0=[r0 v0];

fprintf(fid,'\n\n\n\r %s %3i\r\n','r0=',norm(r0));

for aa = 11:11

    ae = aevec(aa);
    be = bevec(aa);

    fprintf(fid,'\n\n\n\r %s %3i\r\n','swarm=',swarm);
    fprintf(fid,'%s %3i\r\n','ae=',ae);
    fprintf(fid,'%s %3i\r\n','be=',be);
    fprintf(fid,'%s %3i\r\n','maxiter=',iter);
    fprintf(fid,'%2s %10s %8s %8s %8s %8s\r\n','run #','T1','theta1'
        ,'J','iterations','Run Time');

    itn = zeros(20,1);
    rt = zeros(20,1);
    tot_time = 0;

    for h = 20:20

        clear JG Jpbest gbest manDV

        tstart = tic;

        [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,
            lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,
            latlim,longlim);

```

```
        [JG,Jpbest,gbest,x,iter_needed,preburn_state1,initial_target
            ] = PSO_RTM_analytical_prec(2,[1200 period;0 2*pi],prec,
            iter,swarm,rf1,vf1,ae,be,Rmax,Rmin,latlim,longlim,tf1);

        tend = toc(tstart)

        DV1 = norm(preburn_state1(8:10)*1000);
        manDV = round(JG*1000*10^5)/10^5;
        itn(h) = iter_needed;
        rt(h) = tend;

        if h == 1
            minDV = manDV;
            mincount = 1;
        elseif manDV < minDV
            minDV = manDV;
            mincount = 1;
        elseif manDV == minDV
            mincount = mincount + 1;
        end

        fprintf(fid,'%2i %10.2f %8.5f %10.5f %4i %10.4f\r\n',h,gbest
            (1),gbest(2),manDV,itn(h),rt(h));
    end

    gpercent = mincount/h*100;
    tot_time = tot_time + sum(rt);
    mintime = min(rt);
    maxtime = max(rt);
    meantime = mean(rt);
    miniter = min(itn);
    maxiter = max(itn);
```

```
        meaniter = mean(itn);
        fprintf(fid,'%s %8.5f\r\n','min time=',mintime);
        fprintf(fid,'%s %8.5f\r\n','max time=',maxtime);
        fprintf(fid,'%s %8.5f\r\n','avg time=',meantime);
        fprintf(fid,'%s %8.5f\r\n','min iter=',miniter);
        fprintf(fid,'%s %8.5f\r\n','max iter=',maxiter);
        fprintf(fid,'%s %8.5f\r\n','avg iter=',meaniter);
        fprintf(fid,'%s %i\r\n','global conv=',gpercent);
    end
    fprintf(fid,'\n\n\n\r %s','--------------------------------------------------------------------------------');
end
```

D.1.1.1 Constants and Parameters

```
function wgs84data
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                function wgs84data
%% This script provides global conversion factors and WGS 84 constants
%% that may be referenced by subsequent MatLab script files and functions.
%% Note these variables are case-specific and must be referenced as such.
%%
%% The function must be called once in either the MatLab workspace or from a
%% main program script or function. Any function requiring all or some of the
%% variables defined must be listed in a global statement as follows,
%%
```

```
%%  global Deg Rad MU RE OmegaEarth SidePerSol RadPerDay SecDay Flat EEsqrd ...
%%          EEarth J2 J3 J4 GMM GMS AU HalfPI TwoPI Zero_IE Small Undefined
%%
%% in part or in its entirety. Order is not relevent. Case is.
%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%        Originally written by Capt Dave Vallado
%%        Modified and Extended for Ada by Dr Ron Lisowski
%%        Extended from DFASMath.adb by Thomas L. Yoder, LtCol, Spring 00
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global Deg Rad MU RE OmegaEarth SidePerSol RadPerDay SecDay Flat EEsqrd ...
       EEarth J2 J3 J4 GMM GMS AU HalfPI TwoPI Zero_IE Small Undefined g0

%%  Degrees and Radians
      Deg=180.0/pi;                                         %% deg/rad
      Rad= pi/180.0;                                        %% rad/deg

%%  Earth Characteristics from WGS 84
      MU=398600.5;                                                  %% km^3/sec^2
      RE=6378.137;                                                  %% km
      OmegaEarth=0.000072921151467;      %% rad/sec
      SidePerSol=1.00273790935;              %% Sidereal Days/Solar Day
      RadPerDay=6.30038809866574;        %% rad/day
```

```
        SecDay=86400.0;                                  %% sec/day
        Flat=1.0/298.257223563;                  %%
        EEsqrd=(2.0-Flat)*Flat;
        EEarth=sqrt(EEsqrd);
        J2= 0.00108263;
        J3=-0.00000254;
        J4=-0.00000161;
        g0=9.81;

%%  Moon & Sun Characteristics from WGS 84
        GMM= 4902.774191985;                             %% km^3/sec^2
        GMS= 1.32712438E11;                              %% km^3/sec^2
        AU=  149597870.0;                                %% km

%%  HALFPI,PI2           PI/2, & 2PI in various names
        HalfPI= pi/2.0;
        TwoPI= 2.0*pi;

        Zero_IE  = 0.015;                  %% Small number for incl & ecc purposes
        Small    = 1.0E-6;                 %% Small number used for tolerance purposes
        Undefined= 999999.1;
```

D.1.1.2 Determine Classical Orbital Elements for Position and Velocity Vectors

```
function [a,ecc,inc,RAAN,w,nu] = RV2COE(r,v)

%Author: Dan Showalter 18 Oct 2012

%Purpose: Compute classical orbital elements for a position and velocity
%vector.  Based on algorithm in Bate/Mueller/White Fundamentals of
```

```
%Astrodynamics

%% Algorithm
global MU

khat = [0;0;1];

% calculate angular momentum vector
h = cross(r,v);

% calculate nodal vector
n = cross(khat,h);

%calculate eccentricity vector
evec = 1/MU*((norm(v)^2 - MU/norm(r))*r - dot(r,v)*v);

% eccentricity
ecc = norm(evec);

% compute specific mechanical energy

SME = norm(v)^2/2 - MU/norm(r);

% compute semimajor axis
a = -MU/(2*SME);

%compute inclination
inc = acos(h(3)/norm(h));

% compute RAAN
RAAN = acos(n(1)/norm(n));

```

```
if n(2) < 0
    RAAN = 2*pi-RAAN;
end

if ecc <= 0.00001
    ecc = 0;
    w = 0;
    nu = acos(dot(n,r)/(norm(n)*norm(r)));


        if imag(nu) ~= 0
           temp = dot(n,r)/(norm(n)*norm(r));
           if abs(temp) > 1
               temp = sign(temp)*1;
               nu = acos(temp);
           end
        end

        if r(3) < 0
           nu = 2*pi - nu;
        end
else
    w = acos(dot(n,evec)/(norm(n)*norm(evec)));

    if evec(3) < 0
        w = 2*pi -w ;
    end
    nu = acos(dot(evec,r)/(norm(evec)*norm(r)));
    if imag(nu) ~= 0
        temp = dot(evec,r)/(norm(evec)*norm(r));
        if abs(temp) > 1
            temp = sign(temp)*1;
```

```
            nu = acos(temp);
        end
    end
    if dot(r,v) < 0
        nu = 2*pi - nu;
    end
end
```

D.1.1.3 Determine Spacecraft Entry into Exclusion Zone

```
function [R_enter,V_enter,t_enter,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
%UNTITLED2 This function takes a spacecraft's initial position/velocity
%vectors, initial time, initial greenwich mean time and latitude and
%longitude limits and produces the spacecraft's first entry and exit
%conditions into the exclusion zone

%INPUTS
%   r0 = inertial initial position vector (km)
%   v0 = inertial initial velocity vector (km)
%   GMST0 = initial greenwhich mean standard time
%   t0 = initial time (sec)

%OUTPUTS
%   R_enter = inertial entry position into exclusion zone (km)
%   V_enter = inertial velocity vector into exclusion zone (km)
%   t_enter = entry time into exclusion zone
%   lat_enter = latitude of spacecraft when it enters exclusion zone (rad)
%   long_enter = longitude of spacecraft when it enters exclusion zone (rad)
%   R_exit = inertial exit position out of exclusion zone (km)
```

```
%   V_exit = inertial velocity vector out of exclusion zone (km)
%   t_exit = exit time into exclusion zone
%   lat_exit = latitude of spacecraft when it exits exclusion zone (rad)
%   long_exit = longitude of spacecraft when it exits exclusion zone (
    rad)
%%
wgs84data
global MU

longlim_temp = longlim;
if longlim(2) < 0
    longlim_temp(2) = 2*pi + longlim(2);
end
if longlim(1) < 0
    longlim_temp(1) = 2*pi + longlim(1);
end

if longlim_temp(1) > longlim_temp(2)
    longlim_temp(1) = longlim_temp(1) - 2*pi;
    weird_flag = 1;
else
    weird_flag = 0;
end

zone_long_diff = longlim_temp(2) - longlim_temp(1);


[a,ecc,inc,RAAN,w,nu0] = RV2COE(r0,v0);

period = 2*pi*sqrt(a^3/MU);
%% Find spacecraft entry and exit points into exclusion zone
%determine exclusion zone entry/exit times underneath orbit plane
```

```
[nu_enter_AN,nu_exit_AN,nu_enter_DN,nu_exit_DN] = exclusion_nu_intercept
    (latlim,inc,w);

%===========Ascending node opportunity
    =====================================
%Determine inertial position vector to nu_enter_AN and nu_exit_AN
[R_enter_AN,V_enter_AN] = COE2RV(a,ecc,inc,RAAN,w,nu_enter_AN);
[R_exit_AN,V_exit_AN] = COE2RV(a,ecc,inc,RAAN,w,nu_exit_AN);

%determine time of flight from nu0 to nu_enter_AN and nu_exit_AN
[TOF_enter_AN] = TOF_from_nu(a,ecc,nu0,nu_enter_AN,0);
[TOF_exit_AN] = TOF_from_nu(a,ecc,nu0,nu_exit_AN,0);

if TOF_enter_AN < 20 && TOF_enter_AN > 0
    TOF_enter_AN = TOF_enter_AN + period;
    TOF_exit_AN = TOF_exit_AN + period;
end

if TOF_exit_AN < TOF_enter_AN
    if TOF_enter_AN > 0
        TOF_exit_AN = TOF_exit_AN + period;
    else
        TOF_enter_AN = TOF_enter_AN + 2*period;
        TOF_exit_AN = TOF_exit_AN + period;
    end
end

%============Descending node opportunity
    ===================================
%Determine inertial position vector to nu_enter_DN and nu_exit_DN
[R_enter_DN,V_enter_DN] = COE2RV(a,ecc,inc,RAAN,w,nu_enter_DN);
[R_exit_DN,V_exit_DN] = COE2RV(a,ecc,inc,RAAN,w,nu_exit_DN);
```

```
%determine time of flight from nu0 to nu_enter_DN and nu_exit_DN
[TOF_enter_DN] = TOF_from_nu(a,ecc,nu0,nu_enter_DN,0);
[TOF_exit_DN] = TOF_from_nu(a,ecc,nu0,nu_exit_DN,0);

if TOF_enter_DN < 20 && TOF_enter_DN > 0
    TOF_enter_DN = TOF_enter_DN + period;
    TOF_exit_DN = TOF_exit_DN + period;
end

if TOF_exit_DN < TOF_enter_DN
    if TOF_enter_DN > 0
        TOF_exit_DN = TOF_exit_DN + period;
    else
        TOF_enter_DN = TOF_enter_DN + 2*period;
        TOF_exit_DN = TOF_exit_DN + period;
    end
end

flag = 0;
count = 0;

% determine is satellite is/is not in correct longitude range when it is
    in correct
% latitude range. If not, find the next time it will be in the correct
    longitude
% range
while flag == 0;
    %Determine lattitude and longitude of spacecraft at nu_enter_AN and
        nu_exit_AN
```

```
[lat_enter_AN,long_enter_AN,GMST_enter_AN] = IJK_to_LATLONG(
    R_enter_AN(1),R_enter_AN(2),R_enter_AN(3),GMST0,t0+TOF_enter_AN)
    ;
[lat_exit_AN,long_exit_AN,GMST_exit_AN] = IJK_to_LATLONG(R_exit_AN
    (1),R_exit_AN(2),R_exit_AN(3),GMST0,t0+TOF_exit_AN);

if long_enter_AN < 0
    if weird_flag == 0
        long_enter_AN_temp = 2*pi + long_enter_AN;
    else
        long_enter_AN_temp = long_enter_AN;
    end
else
    long_enter_AN_temp = long_enter_AN;
end

if long_exit_AN < 0
    if weird_flag == 0
        long_exit_AN_temp = 2*pi + long_exit_AN;
    else
        long_exit_AN_temp = long_exit_AN;
    end
else
    long_exit_AN_temp = long_exit_AN;
end


%Determine lattitude and longitude of spacecraft at nu_enter_AN and
    nu_exit_AN
[lat_enter_DN,long_enter_DN,GMST_enter_DN] = IJK_to_LATLONG(
    R_enter_DN(1),R_enter_DN(2),R_enter_DN(3),GMST0,t0+TOF_enter_DN)
    ;
```

```
[lat_exit_DN,long_exit_DN,GMST_exit_DN] = IJK_to_LATLONG(R_exit_DN
    (1),R_exit_DN(2),R_exit_DN(3),GMST0,t0+TOF_exit_DN);
flag;

if long_enter_DN < 0
    if weird_flag == 0
        long_enter_DN_temp = 2*pi + long_enter_DN;
    else
        long_enter_DN_temp = long_enter_DN;
    end
else
    long_enter_DN_temp = long_enter_DN;
end

if long_exit_DN < 0
    if weird_flag == 0
        long_exit_DN_temp = 2*pi + long_exit_DN;
    else
        long_exit_DN_temp = long_exit_DN;
    end
else
    long_exit_DN_temp = long_exit_DN;
end



if (longlim_temp(1) <= long_enter_AN_temp && long_enter_AN_temp <=
    longlim_temp(2)) || (longlim_temp(1) <= long_exit_AN_temp &&
    long_exit_AN_temp < longlim_temp(2))
    flag = 1;
    AN = 1;

```

```
        if (longlim_temp(1) <= long_enter_AN_temp && long_enter_AN_temp <= longlim_temp(2)) && (longlim_temp(1) <= long_exit_AN_temp && long_exit_AN_temp < longlim_temp(2))
            nu_enter = nu_enter_AN;
            t_enter = t0 + TOF_enter_AN;
            nu_exit = nu_exit_AN;
            t_exit = t0 + TOF_exit_AN;
        elseif (longlim_temp(1) <= long_enter_AN_temp && long_enter_AN_temp <= longlim_temp(2)) %Exact entry location known, but exact exit unknown
            nu_enter = nu_enter_AN;
            t_enter = t0 + TOF_enter_AN;
            t_exit_guess = t0 + TOF_exit_AN;
            [nu_exit,t_exit] = exclusion_exit_condition_dual2(a,ecc,inc,RAAN,w,nu0,longlim,t_exit_guess,t_enter,GMST0);
        else %Exact entry unknown, but exact exit location known
            nu_exit = nu_exit_AN;
            t_exit = t0 + TOF_exit_AN;
            t_enter_guess = t0 + TOF_enter_AN;
            [nu_enter,t_enter] = exclusion_entry_condition_dual2(a,ecc,inc,RAAN,w,nu0,longlim,t_exit,t_enter_guess,GMST0);
        end
    elseif (longlim_temp(1) <= long_enter_DN_temp && long_enter_DN_temp <= longlim_temp(2)) || (longlim_temp(1) <= long_exit_DN_temp && long_exit_DN_temp < longlim_temp(2))
        flag = 1;
        AN = 2;
        if (longlim_temp(1) <= long_enter_DN_temp && long_enter_DN_temp <= longlim_temp(2)) && (longlim_temp(1) <= long_exit_DN_temp && long_exit_DN_temp < longlim_temp(2))
            nu_enter = nu_enter_DN;
            t_enter = t0 + TOF_enter_DN;
```

```
            nu_exit = nu_exit_DN;
            t_exit = t0 + TOF_exit_DN;
        elseif (longlim_temp(1) <= long_enter_DN_temp &&
            long_enter_DN_temp <= longlim_temp(2)) %Exact entry location
            known, but exact exit unknown
            nu_enter = nu_enter_DN;
            t_enter = t0 + TOF_enter_DN;
            t_exit_guess = t0+ TOF_exit_DN;
            [nu_exit,t_exit] = exclusion_exit_condition_dual2(a,ecc,inc,
                RAAN,w,nu0,longlim,t_exit_guess,t_enter,GMST0);
        else %Exact entry unknown, but exact exit location known
            nu_exit = nu_exit_DN;
            t_exit = t0 + TOF_exit_DN;
            t_enter_guess = t0 + TOF_enter_DN;
            [nu_enter,t_enter] = exclusion_entry_condition_dual2(a,ecc,
                inc,RAAN,w,nu0,longlim,t_exit,t_enter_guess,GMST0);
        end
    elseif flag ~= 1
        long_diff_AN_temp = long_exit_AN - long_enter_AN;
        if long_diff_AN_temp < 0
            long_diff_AN_temp = long_diff_AN_temp + 2*pi;
        end
        long_diff_DN_temp = long_exit_DN_temp - long_enter_AN_temp;
        if long_exit_DN_temp < long_enter_DN_temp
            long_diff_DN_temp = long_diff_DN_temp + 2*pi;
        end
        if long_diff_AN_temp > zone_long_diff && long_enter_AN < longlim
            (1) && long_exit_AN > longlim(2)
            flag = 1;
            nu_exit_guess = nu_exit_AN;
            t_exit_guess = t0 + TOF_exit_AN;
            t_enter_guess = t0 + TOF_enter_AN;
```

```
            [nu_enter,t_enter,phi_enter,lam_enter] =
                exclusion_entry_condition_dual2(a,ecc,inc,RAAN,w,nu0,
                longlim,t_exit_guess,t_enter_guess,GMST0);
            [nu_exit,t_exit,phi_exit,lam_exit] =
                exclusion_exit_condition_dual2(a,ecc,inc,RAAN,w,nu0,
                longlim,t_exit_guess,t_enter,GMST0);
            %not a valid entry if before t0 or if the longitude does not
                 match
            %the limits
            if t_enter < 0 || abs(lam_enter - longlim(1)) >  0.001 ||
                abs(lam_exit - longlim(2)) > 0.001
                 flag = 0;
            end
        end
        if long_diff_DN_temp > zone_long_diff && long_enter_DN < longlim
            (1) && long_exit_DN > longlim(2)
            flag = 1;
            nu_exit_guess = nu_exit_DN;
            t_exit_guess = t0 + TOF_exit_DN;
            t_enter_guess = t0 + TOF_enter_DN;
            [nu_enter,t_enter,phi_enter,lam_enter] =
                exclusion_entry_condition_dual2(a,ecc,inc,RAAN,w,nu0,
                longlim,t_exit_guess,t_enter_guess,GMST0);
            [nu_exit,t_exit,phi_exit,lam_exit] =
                exclusion_exit_condition_dual2(a,ecc,inc,RAAN,w,nu0,
                longlim,t_exit_guess,t_enter,GMST0);
            %not a valid entry if before t0 or if the longitude does not
                 match
            %the limits
            if t_enter < 0 || abs(lam_enter - longlim(1)) >  0.001 ||
                abs(lam_exit - longlim(2)) > 0.001
                 flag = 0;
```

```
            end
        end

    end
    %Exact and exit unknown but spacecraft passes through the exclusion zone



    if flag ~= 0
        if t_enter < t0
            flag = 0;
        end
    end

    if flag == 0
        TOF_enter_AN = TOF_enter_AN + period;
        TOF_exit_AN = TOF_exit_AN + period;
        TOF_enter_DN = TOF_enter_DN + period;
        TOF_exit_DN = TOF_exit_DN + period;

        count = count + 1;

        if count == 100
            error
        end
    end


end
[R_enter,V_enter] = COE2RV(a,ecc,inc,RAAN,w,nu_enter);
```

```
[lat_enter,long_enter,GMST_enter] = IJK_to_LATLONG(R_enter(1),R_enter(2)
    ,R_enter(3),GMST0,t_enter);

[R_exit,V_exit] = COE2RV(a,ecc,inc,RAAN,w,nu_exit);
[lat_exit,long_exit,GMST_enter] = IJK_to_LATLONG(R_exit(1),R_exit(2),
    R_exit(3),GMST0,t_exit);


end
```

D.1.1.4 Determine True Anomaly of Spacecraft at Exclusion Zone Entry

```
function [nu_enter_AN,nu_exit_AN,nu_enter_DN,nu_exit_DN] =
    exclusion_nu_intercept(latlim,incl,omega)
%exclusion_zone_orbit_intercept determines
%   1) the true anomalies of the orbit when it intersects the minimum
    and
%   maximum latitudes of the exclusion zone for botht he ascending node
%   (AN) and descending node (DN) passes.


%INPUTS:
%   latlim = [phi_min phi_max]
%       phi_min = the minimum latitude bound (rad)
%       phi_max = the maximum latitude bound (rad)
%   incl = orbit inclination (rad)
%   omega = orbit argument of perigee


%OUTPUTS
%   nu_enter_AN = limit of true anomaly of spacecraft at entry into
%   exclusion zone on AN pass
```

```
%   nu_exit_AN = limit of true anomaly of spacecraft at exit exclusion zone
%   on AN pass
%   nu_enter_DN = limit of true anomaly of spacecraft at entry into
%   exclusion zone on DN pass
%   nu_exit_DN = limit of true anomaly of spacecraft at exit exclusion zone
%   on DN pass


%% ============================================================================

phi_min = latlim(1);
phi_max = latlim(2);

%%
if incl ~= pi/2
%% **************************PROGRADE ORBITS *******************************
if incl < pi/2
    alpha = incl;
else
    alpha = pi - incl;
end

    %=========================AN passes ===================================
    % 1) Exclusion zone 1st point beneath orbit plane (phi_min, lambda_max)
    delta_nu_enter_AN = asin(sin(norm(phi_min))/sin(alpha));

```

```
if phi_min > 0
    nu_enter_AN = delta_nu_enter_AN - omega;
elseif phi_min < 0
    nu_enter_AN = 2*pi - omega - delta_nu_enter_AN;
else
    nu_enter_AN = 2*pi - omega;
end

% 2) Exclusion zone last point out from under orbit plane (phi_max,
    lambda_min)
delta_nu_exit_AN = asin(sin(norm(phi_max))/sin(alpha));

if phi_max > 0
    nu_exit_AN = delta_nu_exit_AN - omega;
elseif phi_max < 0
    nu_exit_AN = 2*pi - omega - delta_nu_exit_AN;
else
    nu_exit_AN = 2*pi - omega;
end

%==========================DN passes
    ===================================
%Exclusion zone 1st point beneath orbit plane (phi_max, lambda_max)
delta_nu_enter_DN = asin(sin(norm(phi_max))/sin(alpha));

if phi_max > 0
    nu_enter_DN = pi - omega - delta_nu_enter_DN;
elseif phi_max < 0
    nu_enter_DN = pi - omega + delta_nu_enter_DN;
else
    nu_enter_DN = pi - omega;
end
```

```


    %Exclusion zone last point out from under orbit plane (phi_min,
        lambda_min)
    delta_nu_exit_DN = asin(sin(norm(phi_min))/sin(alpha));

    if phi_min > 0
        nu_exit_DN = pi - omega - delta_nu_exit_DN;
    elseif phi_min < 0
        nu_exit_DN = pi - omega + delta_nu_exit_DN;
    else
        nu_exit_DN = pi - omega;
    end
elseif incl == pi/2
%% ************************POLAR ORBITS
    **********************************
    %========================AN PASS
        ====================================
    %Exclusion zone 1st point in lambda_max
    if phi_min > 0
        nu_enter_AN = norm(phi_min) - omega;
    elseif phi_min < 0
        nu_enter_AN = 2*pi - norm(phi_min) - omega;
    else
        nu_enter_AN = 2*pi - omega;
    end

    if phi_max > 0
        nu_exit_AN = norm(phi_max) - omega;
    elseif phi_max < 0
        nu_exit_AN = 2*pi - norm(phi_max) - omega;
    else
```

```
        nu_exit_AN = 2*pi - omega;
    end

    %=========================DN PASS ======================================
    if phi_max > 0
        nu_enter_DN = pi - norm(phi_max) - omega;
    elseif phi_max < 0
        nu_enter_DN = pi + norm(phi_max) - omega;
    else
        nu_enter_DN = pi - omega;
    end

    if phi_min > 0
        nu_exit_DN = pi - norm(phi_min) - omega;
    elseif phi_min < 0
        nu_exit_DN = pi + norm(phi_min) - omega;
    else
        nu_exit_DN = pi - omega;
    end
end

if incl > pi || incl < 0
    disp('Error in exclusion_zone_orbit_intercept: inclination not feasible')
    clear nu_enter_AN
end

if nu_enter_AN < 0
    nu_enter_AN = 2*pi + nu_enter_AN;
elseif nu_enter_AN >= 2*pi
    nu_enter_AN = 2*pi - nu_enter_AN;
```

```
end

if nu_exit_AN < 0
    nu_exit_AN = 2*pi + nu_exit_AN;
elseif nu_exit_AN >= 2*pi
    nu_exit_AN = 2*pi - nu_exit_AN;
end

if nu_enter_DN < 0
    nu_enter_DN = 2*pi + nu_enter_DN;
elseif nu_enter_DN >= 2*pi
    nu_enter_DN = 2*pi - nu_enter_DN;
end

if nu_exit_DN < 0
    nu_exit_DN = 2*pi + nu_exit_DN;
elseif nu_exit_DN >= 2*pi
    nu_exit_DN = 2*pi - nu_exit_DN;
end
```

D.1.1.5 Interpolate to Find Exclusion Zone Entry

```
function [nu_enter,t_enter,lat_enter,long_enter] =
    exclusion_entry_condition_dual2(a,ecc,inc,RAAN,omega,nu0,longlim,
    t_exit,t_enter,GMST0)
%This function computes the the entry states of the spacecraft
%into a rectangular exclusion zone (direct orbits only)

%INPUTS
%   a = orbit semimajor axis (km)
%   ecc = orbit eccentricity
%   inc = orbit inclination (rad)
%   nu_exit = true anomaly of spacecraft upon exit from exclusion zone
```

```
%             (rad)
%   lambda_exit = longitude of spacecraft upon exclusion zone exit (rad)
%   latlim = [phi_min phi_max]
%       phi_min = the minimum latitude bound (rad)
%       phi_max = the maximum latitude bound (rad)
%   longlim = [lambda_min lambda_max]
%       lambda_min = the minimum longitude bound (rad)
%       lambda_max = the maximum longitude bound (rad)

%OUTPUTS
%   nu_enter_ex = true anomaly of spacecraft upon exclusion zone entry (
    rad)
%   phi_enter = latitude of spacecraft upon entry into exclusion zone (
    rad)
%   lambda_enter = longitude of spacecraft upon entry in exclusion zone
    (rad)
%%
wgs84data
global OmegaEarth

if inc < pi/2
    alpha = inc;
elseif inc > pi/2
    alpha = pi - inc;
else
    disp('ERROR:Inclination must be valid')
    clear alpha
end
longlim_temp = longlim(1);
if longlim(1) < 0
    longlim_temp = longlim(1) + 2*pi;
end
```

```
[nu_guess] = nuf_from_TOF(nu0,t_enter,a,ecc);


[R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess);

[lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1),R_guess(2),R_guess(3)
    ,GMST0,t_enter);

if longlim(1) < 0
    long_guess_temp = long_guess;
    if long_guess < 0
        long_guess_temp = 2*pi + long_guess;
    end
%      plot((long_guess)*180/pi,lat_guess*180/pi,'bO')
    del_lambda = longlim_temp - long_guess_temp;
else
    del_lambda = longlim(1) - long_guess;
end

gamma = acos(sin(alpha)*cos(del_lambda));
del_nu = acos(cot(gamma)*cot(alpha));
nu_guess2 = nu_guess + del_nu;
if nu_guess2 > 2*pi
    nu_guess2 = nu_guess2 - 2*pi;
end
delt = TOF_from_nu(a,ecc,nu_guess,nu_guess2,0);
t_guess = t_enter + delt;
```

```
[R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess2);
[lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1),R_guess(2),R_guess(3)
    ,GMST0,t_guess);
if longlim(1) < 0
    long_guess_temp = long_guess;
    if long_guess < 0
        long_guess_temp = 2*pi + long_guess;
        diff = longlim_temp - long_guess_temp;
    else
        diff = longlim(1) - long_guess_temp;
    end
%
%     plot((long_guess_temp)*180/pi,lat_guess*180/pi,'bO')
else
%     plot(long_guess*180/pi,lat_guess*180/pi,'kX')
    diff = longlim(1) - long_guess;
end


count = 0;

while abs(diff) > 1e-6
    del_lambda = del_lambda + diff;
    gamma = acos(sin(alpha)*cos(del_lambda));
    del_nu = acos(cot(gamma)*cot(alpha));
    nu_guess2 = nu_guess + del_nu;
    if nu_guess2 > 2*pi
        nu_guess2 = nu_guess2 - 2*pi;
    end
    delt = TOF_from_nu(a,ecc,nu_guess,nu_guess2,0);
    t_guess = t_enter + delt;
    [R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess2);
```

```
    [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1),R_guess(2),
        R_guess(3),GMST0,t_guess);
    if longlim(1) < 0
        long_guess_temp = long_guess;
        if long_guess < 0
            long_guess_temp = 2*pi + long_guess;
        end
        diff = longlim_temp - long_guess_temp;
%          plot((long_guess_temp)*180/pi,lat_guess*180/pi,'bO')
    else
        diff = longlim(1) - long_guess;
%          plot(long_guess*180/pi,lat_guess*180/pi,'kX')
    end

end


t_enter = t_guess;

nu_enter = nu_guess2;

R_enter = R_guess;
V_enter = V_guess;
lat_enter = lat_guess;
long_enter = long_guess;

% hold on
% plot(long_guess(:)*180/pi,lat_guess(:)*180/pi,'b.')
% plot(long_enter*180/pi,lat_enter*180/pi,'gD',lambda_exit*180/pi,
    phi_exit*180/pi,'gO')
```

D.1.1.6 Interpolate to Find Exclusion Zone Exit

```
function [nu_exit,t_exit,lat_exit,long_exit] = exclusion_exit_condition_dual2(a,ecc,inc,RAAN,omega,nu0,longlim,t_exit,t_enter,GMST0)
%This function computes the the entry states of the spacecraft
%into a rectangular exclusion zone (direct orbits only)

%INPUTS
%   a = orbit semimajor axis (km)
%   ecc = orbit eccentricity
%   inc = orbit inclination (rad)
%   nu_exit = true anomaly of spacecraft upon exit from exclusion zone
%             (rad)
%   lambda_exit = longitude of spacecraft upon exclusion zone exit (rad)
%   latlim = [phi_min phi_max]
%       phi_min = the minimum latitude bound (rad)
%       phi_max = the maximum latitude bound (rad)
%   longlim = [lambda_min lambda_max]
%       lambda_min = the minimum longitude bound (rad)
%       lambda_max = the maximum longitude bound (rad)

%OUTPUTS
%   nu_enter_ex = true anomaly of spacecraft upon exclusion zone entry (rad)
%   phi_enter = latitude of spacecraft upon entry into exclusion zone (rad)
%   lambda_enter = longitude of spacecraft upon entry in exclusion zone (rad)
%%

if inc < pi/2
    alpha = inc;
elseif inc > pi/2
```

```
    alpha = pi - inc;
else
    disp('ERROR:Inclination must be valid')
    clear alpha
end

longlim_temp = longlim(2);
if longlim(2) < 0
    longlim_temp = longlim(2) + 2*pi;
end

lambda_max = longlim(2);

[nu_guess] = nuf_from_TOF(nu0,t_exit,a,ecc);


[R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess);

[lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1),R_guess(2),R_guess(3)
    ,GMST0,t_exit);

if longlim(2) < 0
    long_guess_temp = long_guess;
    if long_guess < 0
        long_guess_temp = 2*pi + long_guess;
    end
%     plot((long_guess)*180/pi,lat_guess*180/pi,'bO')
    del_lambda = long_guess_temp - longlim_temp;
else
    del_lambda = long_guess -longlim(2);
%     plot(long_guess*180/pi,lat_guess*180/pi,'rO')
end
```

```
gamma = acos(sin(alpha)*cos(del_lambda));
del_nu = acos(cot(gamma)*cot(alpha));
nu_guess2 = nu_guess - del_nu;
if nu_guess2 < 0
   nu_guess2 =  nu_guess2 + 2*pi;
end
delt = TOF_from_nu(a,ecc,nu_guess2,nu_guess,0);
t_guess = t_exit - delt;
[R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess2);
[lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1),R_guess(2),R_guess(3)
    ,GMST0,t_guess);
if longlim(2) < 0
    long_guess_temp = long_guess;
    if long_guess < 0
        long_guess_temp = 2*pi + long_guess;
    end
    diff = long_guess_temp - longlim_temp;
%      plot((long_guess)*180/pi,lat_guess*180/pi,'bO')
else
%      plot(long_guess*180/pi,lat_guess*180/pi,'kX')
    diff = long_guess -longlim(2);
end


count = 0;

while abs(diff) > 1e-6
    del_lambda = del_lambda + diff;
    gamma = acos(sin(alpha)*cos(del_lambda));
    del_nu = acos(cot(gamma)*cot(alpha));
    nu_guess2 = nu_guess - del_nu;
```

```
    if nu_guess2 < 0
        nu_guess2 =  nu_guess2 + 2*pi;
    end
    delt = TOF_from_nu(a,ecc,nu_guess2,nu_guess,0);
    t_guess = t_exit - delt;
    [R_guess,V_guess] = COE2RV(a,ecc,inc,RAAN,omega,nu_guess2);
    [lat_guess,long_guess] = IJK_to_LATLONG(R_guess(1),R_guess(2),
        R_guess(3),GMST0,t_guess);
    if longlim(2) < 0
        long_guess_temp = long_guess;
        if long_guess < 0
            long_guess_temp = 2*pi + long_guess;
        end
        diff = long_guess_temp -longlim_temp;
%           plot((long_guess)*180/pi,lat_guess*180/pi,'bO')
    else
        diff = long_guess -longlim(2);
%           plot(long_guess*180/pi,lat_guess*180/pi,'kX')
    end

end


t_exit = t_guess;

nu_exit = nu_guess2;

R_exit = R_guess;
V_exit = V_guess;
lat_exit = lat_guess;
long_exit = long_guess;
```

D.1.1.7 Convert Inertial State into Latitude and Longitude

```
function [lat,long,GMST] = IJK_to_LATLONG(x,y,z,GMST0,t)

global OmegaEarth

r = sqrt(x^2 + y^2 + z^2);

alpha = atan2(y,x);

GMST = GMST0 + OmegaEarth*t;

if GMST >= 2*pi
    GMST = GMST-2*pi;
end

long = alpha - GMST;

if long <= -pi
    long = 2*pi+long;
elseif long >= pi
    long = -2*pi+long;
end

lat = asin(z/r);
```

D.1.2 Single Pass RTM PSO Algorithm

```
function [JGmin,Jpbest,gbest,x,k,preburn_state1,initial_target] =
    PSO_RTM_analytical_prec(n,limits,prec,iter,swarm,rf1,vf1,ae,be,Rmax,
    Rmin,latlim,longlim,tf1)

%Author: Dan Showalter 18 Oct 2012

```

```
%Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem

%generic PSO variable
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row n being
%       upper bound for element n
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO variables
%   n = 4
%       n1 = TOF1 = TOF of first maneuver
%       n2 = theta1 = location on exclusion ellipse where spacecraft will
%       arrive upon completion of maneuver 1
%       n3 = TOF2 = TOF of 2nd maneuver
%       n4 = theta2 = location on exclusion ellipse where spacecraft will
%       arrive upon completion of maneuver 2


%Specific Problem Variables
%   rf1: expected position vector when spacecraft enters exclusion zone
%   vf1: expected velocity vector when spacecraft enters exclusion zone
%   ae: semimajor axis of exclusion ellipse
%   be: semiminor axis of exclusion ellipse
%   Rmax: maximum allowable distance from Earth (constraint on maneuvers)
```

```
%   Rmin: minimum alowable distance from Earth (constraint on maneuvers)
%   latlim: vector defining latitude bounds on exclusion zone
%   longlim: vector defining longitude bounds on exclusion zone
%   end time of maneuver sequence


%%

[N,M] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end

gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);

count = 0;
IND = 0;

CoreNum = 6;
if (matlabpool('size'))<=0
```

```
    matlabpool('open','local',CoreNum);
else
    disp('Parallel Computing Enabled')
end

%loop until maximum iteration have been met
for k = 1:iter

    %create particles dictated by swarm size input


    % if this is the first iteration
    if k == 1
        for h = 1:swarm
            x(:,h) = random('unif',llim,ulim,[n,1]);
            v(:,h) = random('unif',-d,d,[n,1]);
        end

        %if this is after the first iteration, update velocity and
           position
        %of each particle in the swarm
    else
        parfor h = 1:swarm

            %set random weighting for each component
            c1 = 2.1;
            c2 = 2.1;
            phi = c1+c2;
            ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
            %             ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
            cc = c1*random('unif',0,1);
            cs = c2*random('unif',0,1);
```

```
            vdum = v(:,h);
            %
            %              if h ~= IND
            %                  vdum = v(:,h);
            %              else
            %                  vdum = 0;
            %              end
            %update velocity
            vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
                (:,h)));


            %check to make sure velocity doesn't exceed max velocity for
                each
            %variable
            for w = 1:n

                %if the variable velocity is less than the min, set it
                    to the min
                if vdum(w) < -d(w)
                    vdum(w) = -d(w);
                    %if the variable velocity is more than the max, set
                        it to the max
                elseif vdum(w) > d(w);
                    vdum(w) = d(w);
                end
            end

            v(:,h) = vdum;

```

```
        %update position
        xdum = x(:,h) + v(:,h);

        for r = 1:n

            %if particle has passed lower limit
            if xdum(r) < llim(r)
                xdum(r) = llim(r);

            elseif xdum(r) > ulim(r)
                xdum(r) = ulim(r);
            end

            x(:,h) = xdum;

        end

    end

end

% round variables to get finite precision
for aa = 1:n
    x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
    v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
end

%% ***********************Cost Function ************************************
parfor m = 1:swarm
    % ****************Cost function evaluation here ****************************
```

```
    [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] =
        Single_Burn_Maneuver(rf1,vf1,x(1,m),x(2,m),ae,be);

    [Ra1,Rp1] = Ra_Rp_from_RV(r01,v01+DV1vec');

    if Ra1 > Rmax
        J(m) = Inf;
    elseif Rp1 < Rmin
        J(m) = Inf;
    else
        J(m) = manDV1;
    end

end

%% **************************Constraint Equations
    **************************
%%
    **************************************************************************

%%

%round cost to nearest precision required
J = round(J*10^prec(n+1))/10^prec(n+1);

if k == 1

    Jpbest(1:swarm) = J(1:swarm);
    pbest(:,1:swarm) = x(:,1:swarm);

    [Jgbest,IND] = min(Jpbest(:));
```

```

        gbest(:) = x(:,IND);

    else

        for h=1:swarm
            if J(h) < Jpbest(h)
                Jpbest(h) = J(h);
                pbest(:,h) = x(:,h);
                if Jpbest(h) < Jgbest

                    Jgbest = Jpbest(h);
                    gbest(:) = x(:,h);
                    IND = h;

                end
            end
        end
    end

    count = 0;

    for y = 1:swarm

        diff = Jgbest - Jpbest(y);

        if abs(diff)<1e-10
            count = count+1;
        end

    end

```

```
    JG(k) = Jgbest;
    manDV = Jgbest;
    JGmin = Jgbest;

    if count == swarm
        break
    end

        figure(1)
        plot(x(1,:),x(2,:),'x',gbest(1),gbest(2),'rO')
        axis([llim(1) ulim(1) llim(2) ulim(2)])

end

TOF1 = gbest(1);
theta1 = gbest(2);


[r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = Single_Burn_Maneuver(rf1,vf1,gbest(1),gbest(2),ae,be);


initial_target = [rtijk1;vtijk1;rmiss1];
preburn_state1 = [r01;v01;tf1-TOF1;DV1vec'];


% figure
% plot(1:iter,JG)
% title('Cost vs. Iteration #')
% xlabel('# iterations')
% ylabel('cost')
% grid
```

```
% axis square
```

D.1.3 Single Burn Maneuver

```
function [r0,v0,rtijk,vtijk,manDV,DV1vec,rmiss] = Single_Burn_Maneuver(
    rf,vf,TOF,theta,ae,be)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
wgs84data;

global Small MU

%% determine orbit elements at spacecraft entrance into exclusion zone
[a,ecc,inc,RAAN,w,nu] = RV2COE(rf,vf);

%determine position vector of new arrival location
h = cross(rf,vf);

hunit = h/norm(h);

vunit = vf/norm(vf);

gunit = cross(vunit,hunit);

re = ae*be/sqrt((be*cos(theta))^2 + (ae*sin(theta))^2);

rtijk = rf + re*cos(theta)*vunit + re*sin(theta)*gunit;

rmiss = norm(rtijk - rf);


%% determine orbital elements/position vector of departure location
[nu0] = nuf_from_TOF(nu,-TOF,a,ecc);
```

```
[r0,v0] = COE2RV(a,ecc,inc,RAAN,w,nu0);

%% solve lambert's problem both ways
[V1s, V2s, extremal_distances_s, exitflag_s] = lambert2(r0',rtijk',TOF
    /(3600*24),0,MU);

[V1l, V2l, extremal_distances_l, exitflag_l] = lambert2(r0',rtijk',-TOF
    /(3600*24),0,MU);

DVS = norm(V1s - v0');
DVL = norm(V1l - v0');

if DVL < DVS

    manDV = DVL;
    DV1vec = V1l - v0';
    vtijk = V2l';

else

    manDV = DVS;
    DV1vec = V1s - v0';
    vtijk = V2s';

end
```

D.1.4 Lambert Targeting Algorithm

Code provided by [41].

D.1.5 Position and Velocity Vectors from Classical Orbital Elements

```
function [Rijk,Vijk] = COE2RV(a,ecc,inc,RAAN,w,nu)
```

```
%Author: Dan Showalter 18 Oct 2012

%Purpose: find inertial position and velocity vector gievn classical
%orbital elements

%% Algorithm

MU = 398600.5;

%find magnitude of position vector
p = a*(1-ecc^2);

r = p/(1+ecc*cos(nu));

Rpqw = r*[cos(nu);sin(nu);0];
Vpqw = sqrt(MU/p)*[-sin(nu);(ecc+cos(nu));0];

ROT = [cos(RAAN)*cos(w)-sin(RAAN)*sin(w)*cos(inc),-cos(RAAN)*sin(w)-sin(
    RAAN)*cos(w)*cos(inc),sin(RAAN)*sin(inc);...
            sin(RAAN)*cos(w)+cos(RAAN)*sin(w)*cos(inc),-sin(RAAN)*sin(w)+
                cos(RAAN)*cos(w)*cos(inc),-cos(RAAN)*sin(inc);...
            sin(w)*sin(inc),cos(w)*sin(inc),cos(inc)];



Rijk = ROT*Rpqw;
Vijk = ROT*Vpqw;
```

```
end
```

D.1.6 Kepler's Equation

```
function [nuf] = nuf_from_TOF(nu0,TOF,a,e)
%This function computes the final true anomaly based on the initial trua
%anomaly and the time of flight

%INPUTS
% a      = semi-major axis (km)
% nu0    = initial true anomaly (rad)
% TOF    = Time of flight (sec)
% e      = eccentricity (unitless)

%OUTPUTS
% nuf    = final true anomaly (rad)

%GLOBALS
% MU = Earth's gravitational parameter (km^3/sec^2)

%INTERNALS
% n      = mean motion (rad/sec)
% E0     = initial eccentric anomaly (rad)
% M0     = initial mean anomaly (rad)
% Mf     = final mean anomaly (rad)
% Ef     = final eccentric anomaly (rad)
% Eg     = guess for final eccentric anomaly (rad)

global MU
%% 1) compute orbital mean motion
n = sqrt(MU/a^3);


```

```
%% 2) convert initial true anomaly to initial mean anomaly

if nu0 == 0;
    M0 = 0;
elseif nu0 == pi
    M0 = pi;
else
    E0 = acos((e+cos(nu0))/(1+e*cos(nu0)));
    if (nu0 > pi)
        E0 = 2*pi - E0;
    end
    M0 = E0 - e*sin(E0);
end

%% 3) compute final mean anomaly
Mf = M0 + n*TOF;
while Mf > 2*pi
   Mf = Mf - 2*pi;
end

if Mf < 0
    Mf = 2*pi + Mf;
end

Eg = Mf;
Ef = Eg + (Mf - Eg + e*sin(Eg))/(1 - e*cos(Eg));

    while (abs(Ef-Eg) > 1e-12)
        Eg = Ef;
        Ef = Eg + (Mf - Eg + e*sin(Eg))/(1 - e*cos(Eg));
    end

```

```
nuf = acos((cos(Ef)-e)/(1-e*cos(Ef)));
if Ef > pi
    nuf = 2*pi - nuf;
end
```

D.1.7 Determine Perigee and Apogee Radii from Position and Velocity Vectors

```
function [Ra,Rp] = Ra_Rp_from_RV(rvec,vvec)

%rvec = position vector km
%vvec = velocity vector km/s

%Ra = radius of apogee
%Rp = radius of perigee

global MU

%magnitudes of r and v
R = norm(rvec);
V = norm(vvec);

%specific mechanical energy
E = V^2/2 - MU/R;

%semimajor axis from specirfic mechanical energy
a = -MU/(2*E);

%specific angular momentum vector from rvec and vvec
h = cross(rvec,vvec);

%magnitude of specific angular momentum vector
H = norm(h);

```

```
%ecccectricity
e = sqrt(1 + 2*E*H^2/MU^2);

Ra = a*(1+e);
Rp = a*(1-e);


end
```

D.2 Double Pass RTMs

D.2.1 Double Pass RTM Data Script

```
t0 = 0;
GMST0 = 0;
latlim = [-10 10]*pi/180;
longlim = [-50 -10]*pi/180;

wgs84data
global MU
r0vec = [6800 7300;0 0;0 0];
v0vec = [0 0;5.41376581448788 sqrt(MU/7300)/sqrt(2);5.41376581448788
    sqrt(MU/7300)/sqrt(2)];
r0 = 6800*[1 0 0];
v0 = sqrt(MU/6800)/sqrt(2)*[0 1 1];

swarm = 30;
iter = 1000;
aevec = [50 60 70 80 90 100 110 120 130 140 150];
bevec = [5 6 7 8 9 10 11 12 13 14 15];
Rmaxvec = [6850 7350];
Rminvec = [6750 7250];
prec = [2;5;2;5;16];

```

```
for k = 1:2

    r0 = r0vec(:,k);
    v0 = v0vec(:,k);
    Rmax = Rmaxvec(k);
    Rmin = Rminvec(k);
    [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0,v0);
    period = 2*pi*sqrt(a^3/MU);

    state0=[r0 v0];

    fprintf(fid,'\n\n\n\r %s %3i\r\n','r0=',norm(r0));

    for aa = 1:11

        ae = aevec(aa);
        be = bevec(aa);

        fprintf(fid,'\n\n\n\r %s %3i\r\n','swarm=',swarm);
        fprintf(fid,'%s %3i\r\n','ae=',ae);
        fprintf(fid,'%s %3i\r\n','be=',be);
        fprintf(fid,'%s %3i\r\n','maxiter=',iter);
        fprintf(fid,'%2s %10s %8s %8s %8s %8s %8s %8s %8s %8s\r\n','run
           #','T1','theta1','T2','theta2','DV1','DV2','J','iterations',
           'Run Time');

        itn = zeros(20,1);
        rt = zeros(20,1);
        tot_time = 0;

        for h = 1:20

```

```
            clear JG Jpbest gbest manDV
            tstart = tic;

            [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,
               lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,
               latlim,longlim);

            [JG,Jpbest,gbest,x,iter_needed,preburn_state1,
               initial_target1,preburn_state2,initial_target2] =
               PSO_2pass_RTM_analytical_prec(4,[1200 period;0 2*pi;1200
                period;0 2*pi],prec,iter,swarm,rf1,vf1,ae,be,Rmax,Rmin,
               latlim,longlim,tf1);

            tend = toc(tstart)

            DV1 = norm(preburn_state1(8:10)*1000);
            DV2 = norm(preburn_state2(8:10)*1000);
            manDV = round(JG*1000*10^5)/10^5;
            itn(h) = iter_needed;
            rt(h) = tend;

            if h == 1
                minDV = manDV;
                mincount = 1;
            elseif manDV < minDV
                minDV = manDV;
                mincount = 1;
            elseif manDV == minDV
                mincount = mincount + 1;
            end

```

```
            fprintf(fid,'%2i %10.2f %8.5f %10.2f %8.5f %10.5f %10.5f %10.5f %4i %10.4f\r\n',h,gbest(1),gbest(2),gbest(3),gbest(4),DV1,DV2,manDV,itn(h),rt(h));
        end

        gpercent = mincount/h*100;


        tot_time = tot_time + sum(rt);
        mintime = min(rt);
        maxtime = max(rt);
        meantime = mean(rt);

        miniter = min(itn);
        maxiter = max(itn);
        meaniter = mean(itn);


        fprintf(fid,'%s %8.5f\r\n','min time=',mintime);
        fprintf(fid,'%s %8.5f\r\n','max time=',maxtime);
        fprintf(fid,'%s %8.5f\r\n','avg time=',meantime);
        fprintf(fid,'%s %8.5f\r\n','min iter=',miniter);
        fprintf(fid,'%s %8.5f\r\n','max iter=',maxiter);
        fprintf(fid,'%s %8.5f\r\n','avg iter=',meaniter);
        fprintf(fid,'%s %i\r\n','global conv=',gpercent);
    end
    fprintf(fid,'\n\n\n\r %s','--------------------------------------------------------------------------------');
end
```

D.2.2 Double Pass RTM PSO Algorithm

```
function [JGmin,Jpbest,gbest,x,k,preburn_state1,initial_target1,
    preburn_state2,initial_target2] = PSO_2pass_RTM_analytical_prec(n,
    limits,prec,iter,swarm,rf1,vf1,ae,be,Rmax,Rmin,latlim,longlim,tf1)


[N,M] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end

gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);

count = 0;
IND = 0;

CoreNum = 12;
if (matlabpool('size'))<=0
    matlabpool('open','local',CoreNum);
else
    disp('Parallel Computing Enabled')
```

```
end

%loop until maximum iteration have been met
for k = 1:iter

    %create particles dictated by swarm size input
    parfor h = 1:swarm

        % if this is the first iteration
        if k == 1
            x(:,h) = random('unif',llim,ulim,[n,1]);
            v(:,h) = random('unif',-d,d,[n,1]);


            %if this is after the first iteration, update velocity and
                position
            %of each particle in the swarm
        else

            %set random weighting for each component
            c1 = 2.1;
            c2 = 2.1;
            phi = c1+c2;
            ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
            %              ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
            cc = c1*random('unif',0,1);
            cs = c2*random('unif',0,1);


            vdum = v(:,h);
            %
            %              if h ~= IND
```

```
            %                 vdum = v(:,h);
            %             else
            %                 vdum = 0;
            %             end
            %update velocity
            vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
                (:,h)));


            %check to make sure velocity doesn't exceed max velocity for
                 each
            %variable
            for w = 1:n

                %if the variable velocity is less than the min, set it
                    to the min
                if vdum(w) < -d(w)
                    vdum(w) = -d(w);
                    %if the variable velocity is more than the max, set
                        it to the max
                elseif vdum(w) > d(w);
                    vdum(w) = d(w);
                end
            end

            v(:,h) = vdum;

            %update position
            xdum = x(:,h) + v(:,h);

            for r = 1:n

```

```
            %if particle has passed lower limit
            if xdum(r) < llim(r)
                xdum(r) = llim(r);

            elseif xdum(r) > ulim(r)
                xdum(r) = ulim(r);
            end

            x(:,h) = xdum;

        end

    end

end

% round variables to get finite precision
for aa = 1:n
    x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
    v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
end

%% ***********************Cost Function
    ************************************
parfor m = 1:swarm
    % ****************Cost function evaluation here
        ****************************
    OmegaEarth=0.000072921151467;
    [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] =
        Single_Burn_Maneuver(rf1,vf1,x(1,m),x(2,m),ae,be);

    [Ra1,Rp1] = Ra_Rp_from_RV(r01,v01+DV1vec');
```

```
    if Ra1 > Rmax
        J(m) = Inf;
    elseif Rp1 < Rmin
        J(m) = Inf;
    else

        [rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,lat_exit2,long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+OmegaEarth*tf1,0,latlim,longlim);

        [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,rmiss2] = Single_Burn_Maneuver(rf2,vf2,x(3,m),x(4,m),ae,be);

        [Ra2,Rp2] = Ra_Rp_from_RV(r02,v02+DV2vec');

        if Ra2 > Rmax
            J(m) = Inf;
        elseif Rp2 < Rmin
            J(m) = Inf;
        else

            J(m) = manDV1 + manDV2;
        end
    end

end

%% *************************Constraint Equations *************************
```

```
%%
    ************************************************************************

%%

%round cost to nearest precision required
J = round(J*10^prec(n+1))/10^prec(n+1);

if k == 1

    Jpbest(1:swarm) = J(1:swarm);
    pbest(:,1:swarm) = x(:,1:swarm);

    [Jgbest,IND] = min(Jpbest(:));

    gbest(:) = x(:,IND);

else

    for h=1:swarm
        if J(h) < Jpbest(h)
            Jpbest(h) = J(h);
            pbest(:,h) = x(:,h);
            if Jpbest(h) < Jgbest

                Jgbest = Jpbest(h);
                gbest(:) = x(:,h);
                IND = h;

            end
        end
    end
```

```
    end

    count = 0;

    for y = 1:swarm

        diff = Jgbest - Jpbest(y);

        if abs(diff)<1e-10
            count = count+1;
        end

    end

    JG(k) = Jgbest;
    manDV = Jgbest;
    JGmin = Jgbest;

    if count == swarm
        break
    end
end
OmegaEarth=0.000072921151467;
TOF1 = gbest(1);
theta1 = gbest(2);
TOF2 = gbest(3);
theta2 = gbest(4);


[r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = Single_Burn_Maneuver(rf1,
    vf1,gbest(1),gbest(2),ae,be);

```

```
[rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,lat_exit2,
    long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+OmegaEarth*tf1,0,
    latlim,longlim);

[r02,v02,rtijk2,vtijk2,manDV2,DV2vec,rmiss2] = Single_Burn_Maneuver(rf2,
    vf2,gbest(3),gbest(4),ae,be);


initial_target1 = [rtijk1;vtijk1;rmiss1];
preburn_state1 = [r01;v01;tf1-TOF1;DV1vec'];
initial_target2 = [rtijk2;vtijk2;rmiss2];
preburn_state2 = [r02;v02;tf2-TOF2;DV2vec'];


% figure
% plot(1:iter,JG)
% title('Cost vs. Iteration #')
% xlabel('# iterations')
% ylabel('cost')
% grid
% axis square
```

D.3 Triple Pass RTMs

D.3.1 Triple Pass RTM Data Script

```
t0 = 0;
GMST0 = 0;
latlim = [-10 10]*pi/180;
longlim = [-50 -10]*pi/180;

wgs84data
global MU
r0vec = [6800 7300;0 0;0 0];
```

```
v0vec = [0 0;5.41376581448788 sqrt(MU/7300)/sqrt(2);5.41376581448788
    sqrt(MU/7300)/sqrt(2)];


r0 = 6800*[1 0 0];
v0 = sqrt(MU/6800)/sqrt(2)*[0 1 1];

swarm = 60;
iter = 1000;
aevec = [50 60 70 80 90 100 110 120 130 140 150];
bevec = [5 6 7 8 9 10 11 12 13 14 15];
Rmaxvec = [6850 7350];
Rminvec = [6750 7250];
prec = [2;5;2;5;2;5;16];

mincase(1) = 4;
mincase(2) = 1;

for k = 2:2
    r0 = r0vec(:,k);
    v0 = v0vec(:,k);
    Rmax = Rmaxvec(k);
    Rmin = Rminvec(k);
    [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0,v0);
    period = 2*pi*sqrt(a^3/MU);

    state0=[r0 v0];

    fprintf(fid,'\n\n\n\r %s %3i\r\n','r0=',norm(r0));

    for aa = 8:8

```

```
        ae = aevec(aa);
        be = bevec(aa);

        fprintf(fid,'\n\n\n\r %s %3i\r\n','swarm=',swarm);
        fprintf(fid,'%s %3i\r\n','ae=',ae);
        fprintf(fid,'%s %3i\r\n','be=',be);
        fprintf(fid,'%s %3i\r\n','maxiter=',iter);
        fprintf(fid,'%2s %10s %8s %8s %8s %8s %8s %8s %8s %8s %8s %8s %8
            s\r\n','run #','T1','theta1','T2','theta2','T3','theta3','
            DV1','DV2','DV3','J','iterations','Run Time');

        itn = zeros(20,1);
        rt = zeros(20,1);
        tot_time = 0;

        for h = 41:60

            clear JG Jpbest gbest manDV

            tstart = tic;

            [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,
                lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,
                latlim,longlim);

            [JG,Jpbest,gbest,x,iter_needed,preburn_state1,
                initial_target1,preburn_state2,initial_target2,
                preburn_state3,initial_target3] = ...
                 PSO_3pass_RTM_analytical_prec(6,[1200 period;0 2*pi;1200
                      period;0 2*pi;1200 period;0 2*pi],prec,iter,swarm,
                    rf1,vf1,ae,be,Rmax,Rmin,latlim,longlim,tf1);

```

```
        tend = toc(tstart)

        DV1 = norm(preburn_state1(8:10)*1000);
        DV2 = norm(preburn_state2(8:10)*1000);
        DV3 = norm(preburn_state3(8:10)*1000);
        manDV = round(JG*1000*10^5)/10^5;
        itn(h) = iter_needed;
        rt(h) = tend;

        if h == 1 || h == 21 || h == 41
            minDV = manDV;
            mincount = 1;
        elseif manDV < minDV
            minDV = manDV;
            mincount = 1;
        elseif manDV == minDV
            mincount = mincount + 1;
        end

        fprintf(fid,'%2i %10.2f %8.5f %10.2f %8.5f %10.2f %8.5f
            %10.5f %10.5f %10.5f %10.5f %4i %10.4f\r\n',h,gbest(1),
            gbest(2),gbest(3),gbest(4),gbest(5),gbest(6),DV1,DV2,DV3
            ,manDV,itn(h),rt(h));
    end

    gpercent = mincount/h*100;
    tot_time = tot_time + sum(rt);
    mintime = min(rt);
    maxtime = max(rt);
    meantime = mean(rt);
    miniter = min(itn);
    maxiter = max(itn);
```

```
        meaniter = mean(itn);


        fprintf(fid,'%s %8.5f\r\n','min time=',mintime);
        fprintf(fid,'%s %8.5f\r\n','max time=',maxtime);
        fprintf(fid,'%s %8.5f\r\n','avg time=',meantime);
        fprintf(fid,'%s %8.5f\r\n','min iter=',miniter);
        fprintf(fid,'%s %8.5f\r\n','max iter=',maxiter);
        fprintf(fid,'%s %8.5f\r\n','avg iter=',meaniter);
        fprintf(fid,'%s %i\r\n','global conv=',gpercent);
    end
    fprintf(fid,'\n\n\n\r %s','--------------------------------------------------------------------------------');
end
```

D.3.2 Triple Pass RTM PSO Algorithm

```
function [JGmin,Jpbest,gbest,x,k,preburn_state1,initial_target1,preburn_state2,initial_target2,preburn_state3,initial_target3] = PSO_3pass_RTM_analytical_prec(n,limits,prec,iter,swarm,rf1,vf1,ae,be,Rmax,Rmin,latlim,longlim,tf1)

%Author: Dan Showalter 18 Oct 2012

%Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem

%generic PSO variable
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row n being
%       upper bound for element n
```

```
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO variables
%   n = 4
%       n1 = TOF1 = TOF of first maneuver
%       n2 = theta1 = location on exclusion ellipse where spacecraft will
%       arrive upon completion of maneuver 1
%       n3 = TOF2 = TOF of 2nd maneuver
%       n4 = theta2 = location on exclusion ellipse where spacecraft will
%       arrive upon completion of maneuver 2


%Specific Problem Variables
%   rf1: expected position vector when spacecraft enters exclusion zone
%   vf1: expected velocity vector when spacecraft enters exclusion zone
%   ae: semimajor axis of exclusion ellipse
%   be: semiminor axis of exclusion ellipse
%   Rmax: maximum allowable distance from Earth (constraint on maneuvers)
%   Rmin: minimum alowable distance from Earth (constraint on maneuvers)
%   latlim: vector defining latitude bounds on exclusion zone
%   longlim: vector defining longitude bounds on exclusion zone
%   end time of maneuver sequence


%%

```

```
[N,M] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end

gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);

count = 0;
IND = 0;

CoreNum = 12;
if (matlabpool('size'))<=0
    matlabpool('open','local',CoreNum);
else
    disp('Parallel Computing Enabled')
end

%loop until maximum iteration have been met
for k = 1:iter

```

```
%create particles dictated by swarm size input


% if this is the first iteration
if k == 1
    for h = 1:swarm
        x(:,h) = random('unif',llim,ulim,[n,1]);
        v(:,h) = random('unif',-d,d,[n,1]);
    end

    %if this is after the first iteration, update velocity and
        position
    %of each particle in the swarm
else
    parfor h = 1:swarm

        %set random weighting for each component
        c1 = 2.1;
        c2 = 2.1;
        phi = c1+c2;
        ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
        %              ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
        cc = c1*random('unif',0,1);
        cs = c2*random('unif',0,1);


        vdum = v(:,h);
        %
        %              if h ~= IND
        %                  vdum = v(:,h);
        %              else
        %                  vdum = 0;
```

```
        %               end
        %update velocity
        vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
            (:,h)));


        %check to make sure velocity doesn't exceed max velocity for
             each
        %variable
        for w = 1:n

            %if the variable velocity is less than the min, set it
                to the min
            if vdum(w) < -d(w)
                vdum(w) = -d(w);
                %if the variable velocity is more than the max, set
                    it to the max
            elseif vdum(w) > d(w);
                vdum(w) = d(w);
            end
        end

        v(:,h) = vdum;

        %update position
        xdum = x(:,h) + v(:,h);

        for r = 1:n

            %if particle has passed lower limit
            if xdum(r) < llim(r)
                xdum(r) = llim(r);
```

```
                elseif xdum(r) > ulim(r)
                    xdum(r) = ulim(r);
                end

                x(:,h) = xdum;

            end

        end

    end

    % round variables to get finite precision
    for aa = 1:n
        x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
        v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
    end

    %% ***********************Cost Function
        ************************************
    parfor m = 1:swarm
        % ****************Cost function evaluation here
            ****************************
        OmegaEarth=0.000072921151467;
        [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,~] = Single_Burn_Maneuver(
            rf1,vf1,x(1,m),x(2,m),ae,be);

        [Ra1,Rp1] = Ra_Rp_from_RV(r01,v01+DV1vec');

        if Ra1 > Rmax
            J(m) = Inf;
```

```
        elseif Rp1 < Rmin
            J(m) = Inf;
        else

            [rf2,vf2,tf2,~,~,~,~,~,~,~] = zone_entry_exit2(rtijk1,vtijk1
                ,0+OmegaEarth*tf1,0,latlim,longlim);

            [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,~] =
                Single_Burn_Maneuver(rf2,vf2,x(3,m),x(4,m),ae,be);

            [Ra2,Rp2] = Ra_Rp_from_RV(r02,v02+DV2vec');

            if Ra2 > Rmax
                J(m) = Inf;
            elseif Rp2 < Rmin
                J(m) = Inf;
            else

                [rf3,vf3,tf3,~,~,~,~,~,~,~] = zone_entry_exit2(rtijk2,
                    vtijk2,0+OmegaEarth*(tf1+tf2),0,latlim,longlim);

                [r03,v03,rtijk3,vtijk3,manDV3,DV3vec,~] =
                    Single_Burn_Maneuver(rf3,vf3,x(5,m),x(6,m),ae,be);

                [Ra3,Rp3] = Ra_Rp_from_RV(r03,v03+DV3vec');

%                 figure
%                 % plot(long1(:)*180/pi,lat1(:)*180/pi,'r.')
%
%
%                 longmin_plot(1:2) = longlim(1);
%                 longmax_plot(1:2) = longlim(2);
```

```
%                 latmax_plot(1:2) = latlim(1);
%                 latmin_plot(1:2) = latlim(2);
%
%                 plot(longmin_plot(:)*180/pi,latlim(:)*180/pi,'c-')
%                 xlabel('longitude (deg)')
%                 ylabel('latitude (deg)')
%                 axis([-180 180 -90 90])
%                 grid
%                 hold on
%                 plot(longmax_plot(:)*180/pi,latlim(:)*180/pi,'c-')
%                 if longlim(2) > longlim(1)
%                     plot(longlim(:)*180/pi,latmax_plot(:)*180/pi,'c-')
%                     plot(longlim(:)*180/pi,latmin_plot(:)*180/pi,'c-')
%                 else
%                     longlim_plot = [longlim(1) pi -pi longlim(2)];
%                     plot(longlim_plot(1:2)*180/pi,latmax_plot(:)*180/pi,'c-')
%                     plot(longlim_plot(3:4)*180/pi,latmax_plot(:)*180/pi,'c-')
%                     plot(longlim_plot(1:2)*180/pi,latmin_plot(:)*180/pi,'c-')
%                     plot(longlim_plot(3:4)*180/pi,latmin_plot(:)*180/pi,'c-')
%                 end
%                 plot(long_enter2*180/pi,lat_enter2*180/pi,'rO',long_exit2*180/pi,lat_exit2*180/pi,'bO')
%                 plot(long_enter3*180/pi,lat_enter3*180/pi,'rO',long_exit3*180/pi,lat_exit3*180/pi,'bO')
%
%                 close all

                if Ra3 > Rmax
```

```
                J(m) = Inf;
            elseif Rp3 < Rmin
                J(m) = Inf;
            else

                J(m) = manDV1 + manDV2 + manDV3;
            end
        end
    end

end

%% *************************Constraint Equations
    *************************
%%
    ***********************************************************************


%%

%round cost to nearest precision required
J = round(J*10^prec(n+1))/10^prec(n+1);

if k == 1

    Jpbest(1:swarm) = J(1:swarm);
    pbest(:,1:swarm) = x(:,1:swarm);

    [Jgbest,IND] = min(Jpbest(:));

    gbest(:) = x(:,IND);

else
```

```
    for h=1:swarm
        if J(h) < Jpbest(h)
            Jpbest(h) = J(h);
            pbest(:,h) = x(:,h);
            if Jpbest(h) < Jgbest

                Jgbest = Jpbest(h);
                gbest(:) = x(:,h);
                IND = h;

            end
        end
    end
end

count = 0;

for y = 1:swarm

    diff = Jgbest - Jpbest(y);

    if abs(diff)<1e-10
        count = count+1;
    end

end

JG(k) = Jgbest;
manDV = Jgbest;
JGmin = Jgbest;

```

```
    if count == swarm
        break
    end
end
OmegaEarth=0.000072921151467;
TOF1 = gbest(1);
TOF2 = gbest(3);
TOF3 = gbest(5);

[r01,v01,rtijk1,vtijk1,manDV1,DV1vec,rmiss1] = Single_Burn_Maneuver(rf1,
    vf1,gbest(1),gbest(2),ae,be);

[rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,lat_exit2,
    long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+OmegaEarth*tf1,0,
    latlim,longlim);

[r02,v02,rtijk2,vtijk2,manDV2,DV2vec,rmiss2] = Single_Burn_Maneuver(rf2,
    vf2,gbest(3),gbest(4),ae,be);

[rf3,vf3,tf3,lat_enter3,long_enter3,R_exit3,V_exit3,t_exit3,lat_exit3,
    long_exit3] = zone_entry_exit2(rtijk2,vtijk2,0+OmegaEarth*(tf1+tf2)
    ,0,latlim,longlim);

[r03,v03,rtijk3,vtijk3,manDV3,DV3vec,rmiss3] = Single_Burn_Maneuver(rf3,
    vf3,gbest(5),gbest(6),ae,be);


initial_target1 = [rtijk1;vtijk1;rmiss1];
preburn_state1 = [r01;v01;tf1-TOF1;DV1vec'];
initial_target2 = [rtijk2;vtijk2;rmiss2];
preburn_state2 = [r02;v02;tf2-TOF2;DV2vec'];
initial_target3 = [rtijk3;vtijk3;rmiss3];
```

```
preburn_state3 = [r03;v03;tf3-TOF3;DV3vec'];


% figure
% plot(1:iter,JG)
% title('Cost vs. Iteration #')
% xlabel('# iterations')
% ylabel('cost')
% grid
% axis square
```

D.3.3 Triple Pass Multiple Revolution RTM PSO Algorithm

```
function [JGmin,Jpbest,gbest,x,k,ex_flag,Jsubout] =
    PSO_3pass_RTM_local_nrev(n,limits,prec,iter,swarm,nhood,rf1,vf1,ae,
    be,Rmax,Rmin,latlim,longlim,tf1)

%Author: Dan Showalter 18 Oct 2012

%Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem

%generic PSO variable
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row
    n being
%       upper bound for element n
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO variables
```

```
%   n = 4
%       n1 = TOF1 = TOF of first maneuver
%       n2 = theta1 = location on exclusion ellipse where spacecraft
    will
%       arrive upon completion of maneuver 1
%       n3 = TOF2 = TOF of 2nd maneuver
%       n4 = theta2 = location on exclusion ellipse where spacecraft
    will
%       arrive upon completion of maneuver 2


%Specific Problem Variables
%   rf1: expected position vector when spacecraft enters exclusion zone
%   vf1: expected velocity vector when spacecraft enters exclusion zone
%   ae: semimajor axis of exclusion ellipse
%   be: semiminor axis of exclusion ellipse
%   Rmax: maximum allowable distance from Earth (constraint on maneuvers
    )
%   Rmin: minimum alowable distance from Earth (constraint on maneuvers)
%   latlim: vector defining latitude bounds on exclusion zone
%   longlim: vector defining longitude bounds on exclusion zone
%   end time of maneuver sequence


%%

[N,M] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
```

```
    fprintf('Error! limits size does not match number of variables')
    stop
end

lbest = zeros(n,swarm);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);
Jsubs = zeros(3,swarm);
Jsubp = zeros(3,swarm);
Jsubout = zeros(1,3);

count = 0;
IND = 0;

CoreNum = 12;
if (matlabpool('size'))<=0
    matlabpool('open','local',CoreNum);
else
    disp('Parallel Computing Enabled')
end

%loop until maximum iteration have been met
for k = 1:iter

    %create particles dictated by swarm size input


```

```
    % if this is the first iteration
    if k == 1
        for h = 1:swarm
            x(:,h) = random('unif',llim,ulim,[n,1]);
            v(:,h) = random('unif',-d,d,[n,1]);
        end

        %if this is after the first iteration, update velocity and
           position
        %of each particle in the swarm
    else
        parfor h = 1:swarm


            %set random weighting for each component
            c1 = 2.09;
            c2 = 2.09;
            phi = c1+c2;
            ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
            %              ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
            cc = c1*random('unif',0,1);
            cs = c2*random('unif',0,1);


            vdum = v(:,h);
            %
            %              if h ~= IND
            %                  vdum = v(:,h);
            %              else
            %                  vdum = 0;
            %              end
            %update velocity
```

```
        vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(lbest(:,h) - x(:,h)));


        %check to make sure velocity doesn't exceed max velocity for each
        %variable
        for w = 1:n

            %if the variable velocity is less than the min, set it to the min
            if vdum(w) < -d(w)
                vdum(w) = -d(w);
                %if the variable velocity is more than the max, set it to the max
            elseif vdum(w) > d(w);
                vdum(w) = d(w);
            end
        end

        v(:,h) = vdum;

        %update position
        xdum = x(:,h) + v(:,h);

        for r = 1:n

            %if particle has passed lower limit
            if xdum(r) < llim(r)
                xdum(r) = llim(r);

            elseif xdum(r) > ulim(r)
```

```
                    xdum(r) = ulim(r);
                end

                x(:,h) = xdum;

            end

        end

    end

    % round variables to get finite precision
    for aa = 1:n
        x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
        v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
    end

    %% ***********************Cost Function
        ************************************
    parfor m = 1:swarm
        % ****************Cost function evaluation here
            ****************************
        OmegaEarth=0.000072921151467;
        [r01,v01,rtijk1,vtijk1,manDV1,DV1vec,~] = Single_Burn_Maneuver(
            rf1,vf1,x(1,m),x(2,m),ae,be);

        [Ra1,Rp1] = Ra_Rp_from_RV(r01,v01+DV1vec');

        if Ra1 > Rmax
            J(m) = Inf;
            Jsubs(:,m) = [Inf;Inf;Inf];
        elseif Rp1 < Rmin
```

```
        J(m) = Inf;
        Jsubs(:,m) = [Inf;Inf;Inf];
    else

        [rf2,vf2,tf2,lat_enter2,long_enter2,~,~,t2_exit,lat_exit2,
            long_exit2] = zone_entry_exit2(rtijk1,vtijk1,0+
            OmegaEarth*tf1,0,latlim,longlim);

        [r02,v02,rtijk2,vtijk2,manDV2,DV2vec,~] =
            Single_Burn_Maneuver(rf2,vf2,x(3,m),x(4,m),ae,be);

        [Ra2,Rp2] = Ra_Rp_from_RV(r02,v02+DV2vec');

        if Ra2 > Rmax
            J(m) = Inf;
            Jsubs(:,m) = [manDV1;Inf;Inf];
        elseif Rp2 < Rmin
            J(m) = Inf;
            Jsubs(:,m) = [manDV1;Inf;Inf];
        else

            [rf3,vf3,tf3,lat_enter3,long_enter3,~,~,~,lat_exit3,
                long_exit3] = zone_entry_exit2(rtijk2,vtijk2,0+
                OmegaEarth*(tf1+tf2),0,latlim,longlim);

            if x(5,m) > (tf2+tf3-(t2_exit))
                J(m) = Inf;
                Jsubs(:,m) = [manDV1;manDV2;Inf];
            else

```

```
                    [r03,v03,rtijk3,vtijk3,manDV3,DV3vec,~] = Single_Burn_Maneuver_nrev(rf3,vf3,x(5,m),x(6,m),ae,be);

                    [Ra3,Rp3] = Ra_Rp_from_RV(r03,v03+DV3vec');

                    if Ra3 > Rmax
                        J(m) = Inf;
                        Jsubs(:,m) = [manDV1;manDV2;Inf];
                    elseif Rp3 < Rmin
                        J(m) = Inf;
                        Jsubs(:,m) = [manDV1;manDV2;Inf];
                    else

                        J(m) = manDV1 + manDV2 + manDV3;
                        Jsubs(:,m) = [manDV1;manDV2;manDV3];
                    end
                    %                         J(m) = manDV1 + manDV2 + manDV3;
                end
            end
        end

end

%% **************************Constraint Equations **************************
%% *************************************************************************

%%

```

```
%round cost to nearest precision required
J = round(J*10^prec(n+1))/10^prec(n+1);


if k == 1
    Jpbest = J;
    pbest = x;
    Jsubp = Jsubs;
    parfor aa = 1:swarm
        Jtemp = J;
        nup = aa+nhood/2;
        ndown = aa-nhood/2;

        indl = (ndown:1:nup);
        inddown = find(indl < 1);
        indl(inddown) = swarm+indl(inddown);
        indup = find(indl > swarm);
        indl(indup) = indl(indup)-swarm;

        [Jlbest(aa),indmin] = min(Jtemp(indl));
        lbest(:,aa) = x(:,indl(indmin));

    end
else
    parfor aa = 1:swarm
        Jtemp = J;
        nup = aa+nhood/2;
        ndown = aa-nhood/2;

        indl = (ndown:1:nup);
        inddown = find(indl < 1);
        indl(inddown) = swarm+indl(inddown);
```

```
            indup = find(indl > swarm);
            indl(indup) = indl(indup)-swarm;

            [Jmintemp,indmin] = min(Jtemp(indl));
            if Jmintemp < Jlbest(aa)
                Jlbest(aa) = Jmintemp;
                lbest(:,aa) = x(:,indl(indmin));
            end

            if Jtemp(aa) < Jpbest(aa)
                Jpbest(aa) = Jtemp(aa);
                pbest(:,aa) = x(:,aa);
                Jsubp(:,aa) = Jsubs(:,aa);
            end
        end

    end

    [Jgbest,indgbest] = min(Jpbest);
    gbest = pbest(:,indgbest);
    Jsubout = Jsubp(:,indgbest);

    diff = zeros(swarm,1);
    parfor y = 1:swarm
        diff(y) = Jgbest - Jpbest(y);
    end

    indcount = find(abs(diff)<10^(-prec(n+1)));

    JG(k) = Jgbest;
    manDV = Jgbest;
    JGmin = Jgbest;
```

```

    if k > 1
        if JG(k) == JG(k-1)
            count = count + 1;
        else
%             MinCost = Jgbest*1000
%             k
            count = 0;
        end
        %         if length(indcount) > previndcount
        %            length(indcount)
        %         end

        %         if length(indcount) > 100
        %            keyboard
        %         end

    end

    if length(indcount) > 0.75*swarm
        ex_flag = 0;
        break
    end

%         figure(1)
%         plot(x(1,:),x(2,:),'x',lbest(1,:),lbest(2,:),'kO',pbest(1,:), pbest(2,:),'m.',gbest(1),gbest(2),'rO')
%         axis([llim(1) ulim(1) llim(2) ulim(2)])
%
%         figure(2)
%         plot(x(3,:),x(4,:),'x',lbest(3,:),lbest(4,:),'kO',pbest(3,:), pbest(4,:),'m.',gbest(3),gbest(4),'rO')
```

```
%           axis([llim(3) ulim(3) llim(4) ulim(4)])
%
%           figure(3)
%           plot(x(5,:),x(6,:),'x',lbest(5,:),lbest(6,:),'kO',pbest(5,:),
    pbest(6,:),'m.',gbest(5),gbest(6),'rO')
%           axis([llim(5) ulim(5) llim(6) ulim(6)])


    if count > 1000
        ex_flag = 1;
        break
    end
end

if k == iter
    ex_flag = 2;
end

% figure
% plot(1:iter,JG)
% title('Cost vs. Iteration #')
% xlabel('# iterations')
% ylabel('cost')
% grid
% axis square
```

D.3.4 Single Burn Maneuver with Multiple Revolutions

```
function [r0,v0,rtijk,vtijk,manDV,DV1vec,rmiss] =
    Single_Burn_Maneuver_nrev(rf,vf,TOF,theta,ae,be)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
wgs84data;
```

```
global MU

%% determine orbit elements at spacecraft entrance into exclusion zone
[a,ecc,inc,RAAN,w,nu] = RV2COE(rf,vf);

%determine position vector of new arrival location
h = cross(rf,vf);

hunit = h/norm(h);

vunit = vf/norm(vf);

gunit = cross(vunit,hunit);

re = ae*be/sqrt((be*cos(theta))^2 + (ae*sin(theta))^2);

rtijk = rf + re*cos(theta)*vunit + re*sin(theta)*gunit;

rmiss = norm(rtijk - rf);


%% determine orbital elements/position vector of departure location
[nu0] = nuf_from_TOF(nu,-TOF,a,ecc);

[r0,v0] = COE2RV(a,ecc,inc,RAAN,w,nu0);

P0 = 2*pi*sqrt(a^3/MU);

rat = TOF/P0;
m = floor(rat);

```

```
%% solve lambert's problem both ways
[V1s, V2s, extremal_distances_s, exitflag_s] = lambert2(r0',rtijk',TOF/(3600*24),m,MU);

[V1l, V2l, extremal_distances_l, exitflag_l] = lambert2(r0',rtijk',-TOF/(3600*24),m,MU);

if isnan(V1s(1)) == 1
    [V1s, V2s, extremal_distances_s, exitflag_s] = lambert2(r0',rtijk',TOF/(3600*24),0,MU);
elseif isnan(V1l(1)) == 1
    [V1l, V2l, extremal_distances_l, exitflag_l] = lambert2(r0',rtijk',-TOF/(3600*24),0,MU);
end

DVS = norm(V1s - v0');
DVL = norm(V1l - v0');

if DVL < DVS

    manDV = DVL;
    DV1vec = V1l - v0';
    vtijk = V2l';

else

    manDV = DVS;
    DV1vec = V1s - v0';
    vtijk = V2s';

end
```

Appendix E: Code for Low Thrust Responsive Theater Maneuvers

E.1 Single Pass low-thrust responsive theater maneuvers (LTRTMs)

E.1.1 Particle Swarm Algorithms

E.1.1.1 Single Pass LTRTM PSO Driver

```
t0 = 0;
GMST0 = 0;
latlim = [-10 10]*pi/180;
longlim = [-50 -10]*pi/180;

wgs84data
global MU MU2
r0vec = [7300;0;0];
v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];

[a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
period = 2*pi*sqrt(a^3/MU);

aevec = [150 140 130 120 110 100 90 80 70 60 50];
bevec = [15 14 13 12 11 10 9 8 7 6 5];
Rmaxvec = norm(r0vec)+50;
Rminvec = norm(r0vec)-50;

DU = norm(r0vec);
TU = period/(2*pi);
MU2 = MU*TU^2/DU^3;

m0 = 1000;
r0 = r0vec;
v0 = v0vec;
```

```
Rmax = Rmaxvec;
Rmin = Rminvec;

%Energy of most elliptical orbit
ab = (Rmax + Rmin)/2; %semi-major axis of orbit
Eb = -MU/(2*ab);   %energy of orbit
Vmax = sqrt(2*(MU/Rmin + Eb));
Vmin = sqrt(2*(MU/Rmax + Eb));

state0=[r0 v0];
Tmax = 2e-3;
dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\Single Pass\Data\';

fid = fopen([dir 'PSOSinglePassData_Final_5312014.txt'],'a');

swarm = 40;
iter = 1000;
prec = [3;6;3;6];

for bb = 2:11

    if bb == 1
                fprintf(fid,'%s %i\r\n','r0 (km) =',r0vec(1));
                fprintf(fid,'%s %i\r\n','swarm =',swarm);
                fprintf(fid,'%s\t %s\t %s\t %s\t %s\t %s\t %s\t \r\n','TOF','Phi','Vf','fpa','DV','iter','time');
                fprintf(fid,'%s\r\n','-------------------------------------');
        endval = 20;
    else endval = 20;
    end
```

```
    ae = aevec(bb);
    be = bevec(bb);


    for aa = 1:endval

        tstart = tic;

        [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
            long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
            ;
        [JGmin,Jpbest,gbest,x,k] = LT_RTM_PSO_TFIXED(3,[0 2*pi;Vmin Vmax
            ;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf1,vf1,tf1,
            ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);
        Cost1 = JGmin*DU/TU*1000;
        tend = toc(tstart);

        fprintf(fid,'%i \t\t %10.5f\t %4.3f\t %7.6f\t %4.3f\t %7.6f\t %i
            \t %4.1f\r\n',ae,tf1,gbest(1),gbest(2),gbest(3),Cost1,k,tend
            );

    end

end
```

E.1.1.2 Single Pass LTRTM PSO Algorithm

```
function [JGmin,Jpbest,gbest,x,k] = LT_RTM_PSO_TFIXED(n,limits,iter,
    swarm,prec,rfvec,vfvec,tf,ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0)

%Author: Dan Showalter 18 Oct 2012

```

```
%Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem

%generic PSO variable
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%   in row n being lower bound for element n and 2nd element in row n
    being
%   upper bound for element n
%   iter: number of iterations
%   swarm: swarm size

%Problem specific PSO variables
%   n = 4
%       n1 = TOF1 = TOF of first maneuver
%       n2 = theta1 = location on exclusion ellipse where spacecraft
    will
%       arrive upon completion of maneuver 1
%       n3 = TOF2 = TOF of 2nd maneuver
%       n4 = theta2 = location on exclusion ellipse where spacecraft
    will
%       arrive upon completion of maneuver 2


%Specific Problem Variables
%   rf1: expected position vector when spacecraft enters exclusion zone
%   vf1: expected velocity vector when spacecraft enters exclusion zone
%   ae: semimajor axis of exclusion ellipse
%   be: semiminor axis of exclusion ellipse
%   Rmax: maximum allowable distance from Earth (constraint on maneuvers
    )
%   Rmin: minimum alowable distance from Earth (constraint on maneuvers)
%   latlim: vector defining latitude bounds on exclusion zone
```

```
%   longlim: vector defining longitude bounds on exclusion zone
%   end time of maneuver sequence


%%

[N,M] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end

gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(iter,swarm);

count = 0;
IND = 0;

CoreNum = 12;
if (matlabpool('size'))<=0
    matlabpool('open','local',CoreNum);
else
```

```
    disp('Parallel Computing Enabled')
end

%loop until maximum iteration have been met
for k = 1:iter

    %create particles dictated by swarm size input
    parfor h = 1:swarm

        % if this is the first iteration
        if k == 1
            x(:,h) = random('unif',llim,ulim,[n,1]);
            v(:,h) = random('unif',-d,d,[n,1]);

            %if this is after the first iteration, update velocity and position
            %of each particle in the swarm
        else
            %set random weighting for each component
            ci = 2/abs(2-2*2.09 - sqrt(4.18^2 - 4*4.18));
            %              ci = 0.7/(n-1)*k + (1.2 - 0.7/(n-1));
            cc = 2.09*random('unif',0,1);
            cs = 2.09*random('unif',0,1);



            vdum = v(:,h);
            %update velocity
            vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x(:,h)));


```

```
        %check to make sure velocity doesn't exceed max velocity for each
        %variable
        for w = 1:n

            %if the variable velocity is less than the min, set it to the min
            if vdum(w) < -d(w)
                vdum(w) = -d(w);
                %if the variable velocity is more than the max, set it to the max
            elseif vdum(w) > d(w);
                vdum(w) = d(w);
            end
        end

        v(:,h) = vdum;

        %update position
        xdum = x(:,h) + v(:,h);

        for r = 1:n

            %if particle has passed lower limit
            if xdum(r) < llim(r)
                xdum(r) = llim(r);

            elseif xdum(r) > ulim(r)
                xdum(r) = ulim(r);
            end

            x(:,h) = xdum;
```

```

            end

        end

    end
    %% ***********************Cost Function ************************************
    parfor m = 1:swarm
        % ****************Cost function evaluation here ****************************
        MU2 = MU*TU^2/DU^3;

        phi = x(1,m);
        Vt_mag = x(2,m);
        fpa_t = x(3,m);

        [DV,~,~,~,~,~,~,~,~,~,~,rt_ijk,vt_ijk,~,~] = Single_LT_Maneuver(rfvec,vfvec,tf,phi,ae,be,Vt_mag,fpa_t,DU,TU,MU2);

        %          maxT = maxT*DU/TU^2;

        [a,ecc,inc,RAAN,w,nu] = RV2COE(rt_ijk,vt_ijk);

        Ra = a*(1+ecc);
        Rp = a*(1-ecc);

        %          if maxT > Tmax/m0
        %              J(m) = Inf;
        %          else
        if Ra > Rmax || Rp < Rmin
            J(m) = Inf;
```

```
    else

        J(m) = DV;
    end

end

%% *************************Constraint Equations
    *************************
%%
    ************************************************************************

%%

%round cost to nearest precision required
J = round(J*10^prec(n+1))/10^prec(n+1);

if k == 1
    count = 0;
    Jpbest(1:swarm) = J(1:swarm);
    pbest(:,1:swarm) = x(:,1:swarm);

    [Jgbest,IND] = min(Jpbest(:));
    gbest(:) = x(:,IND);

else

    Jtemp = J;
    parfor h=1:swarm
        if Jtemp(h) < Jpbest(h)
            Jpbest(h) = J(h);
            pbest(:,h) = x(:,h);
```

```
            end
        end

        [Jgbest,indgbest] = min(Jpbest);
        gbest = pbest(:,indgbest);

    end



    diff = zeros(swarm,1);
    parfor y = 1:swarm

        diff(y) = Jgbest - Jpbest(y);
    end

    indcount = find(abs(diff)<10^(-prec(n+1)));




    JG(k) = Jgbest;
    JGmin = Jgbest;


    if length(indcount) == swarm
        break
    end

    if k > 1
        if JG(k) == JG(k-1)
            count = count + 1;
```

```
        else
            count = 0;
        end

    end


end
```

E.1.1.3 Single Low Thrust Maneuver

```
function [LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,Tvec,TOF_calc,rt_ijk,vt_ijk,rt_pqw,vt_pqw,r0_pqw,v0_pqw,rf_pqw,vf_pqw,rmiss] = Single_LT_Maneuver(rf,vf,TOF,phi,ae,be,Vt_mag,fpa_t,DU,TU,MU2)
%Single_LT_Maneuver computes a feasible low thrust maneuver to intercept rf
%at a specified time
wgs84data;

%INPUTS
% rf = inertial position vector at expected arrival location (DU)
% vf = inertial velocity vector at expected arrival location (DU/TU)
% TOF = time of flight (TU)
% phi = angle of exclusion ellipse (rad)
% ae = exclusion ellipse semi-major axis (DU)
% be = exclusion ellipse semi-minor axis (DU)
% Vt_mag = velocity magnitude at new arrival location (DU/TU)
% fpa_t = flight path angle at new arrival location (rad)

%OUTPUTS
%LT_DV = total delta V required for shape-based maneuver (DU/TU)
%maxT = maximum thrust acceleration allowed (DY/TU^2)
```

```
%r = vector of radius values (DU) in perifocal frame
%T_a = thrust acceleration profile (DU/TU^2)
%thetaf_int = vector of theta values (rad)
%theta_dot = vector of time rate of change of thetaf_int (rad/TU)
%theta_ddot = vector of time rate of change of theta_dot (rad/TU^2)
%rdot = vector of rate time rate of change of r (DU/TU)
%Tvec = vector of time values (TU)
%TOF_calc = calculated time of flight (TU) - should match TOF
%rt_ijk = inertial position vector ,vt_ijk

%
MU = 398600.5;
%% determine inertial position vectors of maneuver initiaion and
    completion
[a,ecc,inc,RAAN,w,nu] = RV2COE(rf,vf);

period = 2*pi*sqrt(a^3/MU);

%determine position vector of new arrival location
h = cross(rf,vf);

hunit = h/norm(h);

vunit = vf/norm(vf);

gunit = cross(vunit,hunit);

re = ae*be/sqrt((be*cos(phi))^2 + (ae*sin(phi))^2);

%inertial position vector of new arrival position
rt_ijk = rf + re*cos(phi)*vunit + re*sin(phi)*gunit;

```

```
%inertial velocity vector at arrival
%maneuver is coplanar so expected angular momentum is in same direction as
%actual angular momentum at arrival

%unit vector used to help determine actual velocity vector
funit = cross(hunit,rt_ijk)/norm(rt_ijk);

vt_ijk = Vt_mag*sin(fpa_t)*rt_ijk/norm(rt_ijk) + Vt_mag*cos(fpa_t)*funit;

rmiss = norm(rt_ijk - rf);


% determine orbital elements/position vector of departure location
[nu0] = nuf_from_TOF(nu,-TOF,a,ecc);

[r0_ijk,v0_ijk] = COE2RV(a,ecc,inc,RAAN,w,nu0);


% convert inertial coordinates to perifocal frame
[r0_pqw,v0_pqw] = IJK_to_PQW(r0_ijk,v0_ijk,inc,RAAN,w);
[rt_pqw,vt_pqw] = IJK_to_PQW(rt_ijk,vt_ijk,inc,RAAN,w);
[rf_pqw,vf_pqw] = IJK_to_PQW(rf,vf,inc,RAAN,w);

%determine total transfer angle
cos_psi = (re^2 - norm(rf_pqw)^2 - norm(rt_pqw)^2)/(-2*norm(rf_pqw)*norm(rt_pqw));
psi = acos(cos_psi);

%expected flight path angle
[fpa_1] = fpa_calc(ecc,nu);
```

```
if phi > pi/2-fpa_1 && phi < 3*pi/2-fpa_1
    psi = -psi;
end

%total transfer angle
revs = TOF/period;

nrevs = floor(revs);

if nu > nu0
    ang1 = nu-nu0;
else
    ang1 = 2*pi + nu-nu0;
end

ang = ang1+2*pi*nrevs;

thetaf = ang + psi;

%flight path angle of satellite at maneuver initiation
[gamma0] = fpa_calc(ecc,nu0);

%% scale vectors
r0_pqw = r0_pqw/DU;
v0_pqw = v0_pqw/DU*TU;
rt_pqw = rt_pqw/DU;
vt_pqw = vt_pqw/DU*TU;
rf_pqw = rf_pqw/DU;
vf_pqw = vf_pqw/DU*TU;
TOF = TOF/TU;
```

```
[LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,Tvec,TOF_calc] = LT_TF_FIXED_F0(r0_pqw,v0_pqw,rt_pqw,vt_pqw,thetaf,gamma0,fpa_t,TOF,MU2);
```

E.1.1.4 Calculate Flight Path Angle

```
function [fpa] = fpa_calc(e,nu)
%Generates flight path angle as a function of orbit eccentricity and flight
%path angle

%INPUTS
% e = orbit eccentricity (unitless)
% nu = orbit true anomaly (rad)

%OUTPUT
% fpa = flight path angle (rad)

%sin of flight path angle
sin_fpa = (e*sin(nu))/sqrt(1+2*e*cos(nu)+e^2);

%cos of flight path angle
cos_fpa = (1+e*cos(nu))/sqrt(1+2*e*cos(nu)+e^2);

fpa = atan2(sin_fpa,cos_fpa);


end
```

E.1.1.5 Shape-Based Low Thrust Trajectory Optimization

```
function [LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,Tvec,TOF_calc] = LT_TF_FIXED_F0(r1vec,v1vec,rfvec,vfvec,thetaf,gamma1,gammaf,TOF,MU)
```

```
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

%INPUTS
%r1vec = position vector (3x1) of initial orbit at theta0 (DU)
%v1vec = velocity vector (3x1) of initial orbit at theta0 (DU/TU)
%rfvec = position vector (3x1) of final orbit at thetaf (DU)
%vfvec = velocity vector (3x1) of initial orbit at theta0 (DU/TU)
%gamma1 = flight path angle of initial orbit at theta1 (rad)
%gamma2 = flight path angle of final orbit at thetaf (rad)
%
    ==========================================================================

h1vec = cross(r1vec,v1vec); %specific angular momentum of body 1
h1 = norm(h1vec); %magnitude of specific angular momentum
r1 = norm(r1vec); %magnitude of position vector

hfvec = cross(rfvec,vfvec); %specific angular momentum of body2 at
    thetaf
hf = norm(hfvec); %magnitude of specific angular momentum
rf = norm(rfvec); %magnitude of position vector
vf = norm(vfvec);

a = 1/r1; %parameter a
b = -tan(gamma1)/r1; %parameter b

thetadot1 = h1/(r1^2); %rate of change of theta1
thetadotf = hf/(rf^2);

c = 1/(2*r1)*(MU/(r1^3*thetadot1^2)-1); %parameter c

```

```
flag = 0;
guess = 0;
n = 0;
step = .1;
total = 20;
while flag == 0
    options = optimset('Display','off');
    [d,FVAL,ex_flag] = fzero(@(x) TF_PARAM_d_RTM_f0(x,rf,TOF,thetaf,
       gammaf,a,b,c,thetadotf,MU,n,step),guess,options);
    if d == guess && FVAL == 0
        if n < total && n >= 0
            guess = guess + step;
            n = n + 1;
        elseif n == total
            guess = -step;
            n = -1;
        else
            guess = guess - step;
            n = n - 1;
            if n == -step*total;
                flag = 1;
            end
        end
    else
        flag = 1;
    end

end

if ex_flag ~=1
%     disp('Fzero did not converge to a solution')
    LT_DV = Inf;
```

```
    maxT = Inf;
    r = 0;
    gamma = 0;
    T_a = 0;
    thetaf_int = 0;
    theta_dot = 0;
    theta_ddot = 0;
    rdot = 0;
    Tvec = 0;
    TOF_calc = 0;
%      gammaf
%      vf

else


    mat1 = [30*thetaf^2 -10*thetaf^3 thetaf^4;...
        -48*thetaf 18*thetaf^2 -2*thetaf^3;...
        20 -8*thetaf thetaf^2];

    mat2 = [1/rf-(a+b*thetaf+c*thetaf^2+d*thetaf^3);...
        -tan(gammaf)/rf-(b+2*c*thetaf+3*d*thetaf^2);...
        MU/(rf^4*thetadotf^2)-(1/rf+2*c+6*d*thetaf)];

    soln_vec = 1/(2*thetaf^6)*mat1*mat2;

    e = soln_vec(1); %parameter d
    f = soln_vec(2); %parameter e
    g = soln_vec(3); %parameter f

    thetaf_int = linspace(0,thetaf,100); %set up

```

```
theta = thetaf_int; %theta values

r = 1./(a + b*thetaf_int + c*thetaf_int.^2 + d*thetaf_int.^3 + e*
    thetaf_int.^4 + f*thetaf_int.^5 + g*thetaf_int.^6); %r values
    based on parametric representation as a function of theta

tan_gamma = -r.*(b + 2*c.*thetaf_int + 3*d.*thetaf_int.^2 + 4*e.*
    thetaf_int.^3 + 5*f.*thetaf_int.^4 + 6*g*thetaf_int.^5); %
    tangent of flight path angle (thrust assumed along fpa)

gamma = atan(tan_gamma); %actual flight path angle

denom = (1./r + 2*c + 6*d.*theta + 12*e.*theta.^2 + 20*f.*theta.^3 +
     30*g.*theta.^4); %denominator of terms used to compute angular
    velocity (theta_dot) acceleration (theta_ddot) and thrust
    acceleration (T_a)

term1 = 4.*tan_gamma./denom; %term used for angular acceleration (
    theta_ddot)
term2 = (6*d + 24*e.*theta + 60*f.*theta.^2 + 120*g.*theta.^3 -
    tan_gamma./r)./denom.^2; %term used for angular acceleration (
    theta_ddot)

theta_ddot = -MU./(2.*r.^4).*(term1 + term2); %angular acceleration
theta_dot = sqrt(MU./(r.^4).*(1./denom)); %angular velocity
T_a = -MU./(2.*(r.^3).*cos(gamma)).*term2; %thrust acceleration

rdot = -r.^2.*(b + 2*c.*theta + 3*d.*theta.^2 + 4*e.*theta.^3 + 5*f
    .*theta.^4 + 6*g.*theta.^5).*theta_dot;

maxT = max(abs(T_a));

```

```
    time_func = sqrt((r.^4/MU.*denom)); %function values used for
        quadrature integration of time of flight

    dT = zeros(length(theta),1);
    Tvec = zeros(length(theta),1);

    for aa = 2:length(theta)
        fa = time_func(aa-1);
        fb = time_func(aa);

        dT(aa) = (theta(aa) - theta(aa-1))*(fa + fb)/2;
        Tvec(aa) = Tvec(aa-1) + dT(aa);
    end

    TOF_calc = sum(dT);

    for bb = 2:length(theta)
        % Delta V
        fa_DV = abs(T_a(bb-1))/theta_dot(bb-1);
        fb_DV = abs(T_a(bb))/theta_dot(bb);
        DV_vec(bb) = (theta(bb) - theta(bb-1))*(fa_DV + fb_DV)/2;
    end

    LT_DV = sum(DV_vec);


end
```

E.1.1.6 Root Finding Equation

```
function [func] = TF_PARAM_d_RTM_f0(x,rf,TOF,thetaf,gammaf,a,b,c,
    thetadotf,MU2,n,step)
d = x;
```

```

mat1 = [30*thetaf^2 -10*thetaf^3 thetaf^4;...
        -48*thetaf 18*thetaf^2 -2*thetaf^3;...
        20 -8*thetaf thetaf^2];

mat2 = [1/rf-(a+b*thetaf+c*thetaf^2+d*thetaf^3);...
        -tan(gammaf)/rf-(b+2*c*thetaf+3*d*thetaf^2);...
        MU2/(rf^4*thetadotf^2)-(1/rf+2*c+6*d*thetaf)];

soln_vec = 1/(2*thetaf^6)*mat1*mat2;

e = soln_vec(1); %parameter d
f = soln_vec(2); %parameter e
g = soln_vec(3); %parameter f

thetaf_int = linspace(0,thetaf,100); %set up

theta = thetaf_int; %theta values

r = 1./(a + b*thetaf_int + c*(thetaf_int.^2) + d*(thetaf_int.^3) + e*(
    thetaf_int.^4) + f*(thetaf_int.^5) + g*(thetaf_int.^6)); %r values
    based on parametric representation as a function of theta

denom = (1./r + 2*c + 6*d*theta + 12*e*(theta.^2) + 20*f*(theta.^3) +
    30*g*(theta.^4)); %denominator of terms used to compute angular
    velocity (theta_dot) acceleration (theta_ddot) and thrust
    acceleration (T_a)

ind = find(denom < 0);

time_func = sqrt(((r.^4)/MU2).*denom); %function values used for
    quadrature integration of time of flight
```

```
for aa = 2:length(theta)
    fa = time_func(aa-1);
    fb = time_func(aa);

    dT(aa) = (theta(aa) - theta(aa-1))*(fa + fb)/2;
end

TOF_calc = sum(dT);

func = TOF_calc - TOF;

    if norm(x - n*step) < 1e-6 && isreal(TOF_calc) == 0
        func = TOF - sqrt(real(TOF_calc)^2 + imag(TOF_calc)^2);
    end
end
```

E.1.2 Direct Collocation Algorithms

E.1.2.1 Single Pass LTRTM Driver

```
for zz = 2:2

    if zz == 1
        load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Single Pass\Data\data6800_LT_1RTMsort.mat')
        PSO_data = data6800_LT_1RTMsort;
        rmag = 6800;
    elseif zz == 2
        load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Single Pass\Data\data7300_LT_1RTMsort.mat')
        PSO_data = data7300_LT_1RTMsort;
        rmag = 7300;
    end
```

```
for cc = 1:22
    fid2 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust
        RTM\Single Pass\Data\PSO2GPOPSSinglePassData.txt','a');
    clear guess setup limits output

    clc
    tstart = tic;

    fid = fopen('PSO_to_GPOPS.txt','a');
    t0 = 0;
    GMST0 = 0;
    latlim = [-10 10]*pi/180;
    longlim = [-50 -10]*pi/180;

    wgs84data
    global MU
    r0vec = [rmag;0;0];
    v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];

    [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
    period = 2*pi*sqrt(a^3/MU);

    swarm = 30;
    iter = 1000;
    Rmaxvec = norm(r0vec)+50;
    Rminvec = norm(r0vec)-50;
    prec = [2;5;16];

    r0 = r0vec;
    v0 = v0vec;
    Rmax = Rmaxvec;
```

```
        Rmin = Rminvec;

        ae = PSO_data(cc,1);
        be = ae/10;

        TOF = PSO_data(cc,2);
        phi = PSO_data(cc,3);
        Vt_mag = PSO_data(cc,4);
        fpa_t = PSO_data(cc,5);

        tstart = tic;

        [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
           long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
           ;

        DU = norm(rf1);
        TU = period/(2*pi);

        ae1 = ae/DU;
        be1 = be/DU;
        r01 = norm(r0)/DU;
        MU2 = MU*TU^2/DU^3;

        t0min = 0; % minimum initial time
        t0max = 0; % maximum initial time
        tfmin = period/TU; % minimum final time
        tfmax = period/TU;
        n0 = sqrt(MU2/(norm(r0)/DU)^3);

        [LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,
           Tvec,TOF_calc,rt_ijk,vt_ijk,rt_pqw,vt_pqw,r0_pqw,v0_pqw] =
```

```
        Single_LT_Maneuver(rf1,vf1,TOF,phi,ae,be,Vt_mag,fpa_t,DU,TU,
        MU2);

    delt = (tf1 - TOF)/TU;
    time_mod = Tvec + delt;

    [rf_pqw,vf_pqw] = IJK_to_PQW(rf1,vf1,inc,RAAN,w);
    rf_pqw = rf_pqw/DU;
    vf_pqw = vf_pqw/DU*TU;



    vunit = vf_pqw/norm(vf_pqw);
    hfp = cross(rf_pqw,vf_pqw);
    hunit = hfp/norm(hfp);

    gunit = cross(vunit,hunit);

    ang = (0:0.001:2*pi);
    re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);



    theta_rf = atan2(rf_pqw(2),rf_pqw(1));
    if theta_rf < 0
        theta_rf = 2*pi + theta_rf;
    end
    [rtest] = IJK_to_PQW(r0,v0,inc,RAAN,w);
    theta0 = atan2(rtest(2),rtest(1));

    theta_mod = thetaf_int + atan2(r0_pqw(2),r0_pqw(1));

    coast_length = 1;

```

```
time_guess = zeros(coast_length+length(time_mod),1);
time_guess(1:coast_length) = 0;
time_guess(coast_length+1:end) = time_mod;
theta_guess = zeros(coast_length+length(theta_mod),1);
theta_guess(coast_length+1:end) = theta_mod;
theta_guess(1:coast_length) = theta0;
r_guess = zeros(coast_length+length(theta_mod),1);
r_guess(1:coast_length) = norm(r0)/DU;
r_guess(coast_length+1:end) = r;
vr_guess = zeros(coast_length+length(theta_mod),1);
vr_guess(1:coast_length) = 0;
vr_guess(coast_length+1:end) = rdot;
vtheta_guess = zeros(coast_length+length(theta_mod),1);
vtheta_guess(1:coast_length) = sqrt(MU2/(norm(r0)/DU));
vtheta_guess(coast_length+1:end) = r.*theta_dot;
T_guess = zeros(coast_length+length(time_mod),1);
T_guess(1:coast_length) = 0;
T_guess(coast_length+1:end) = T_a;
B_guess = zeros(coast_length+length(time_mod),1);
B_guess(1:coast_length) = 0;
B_guess(coast_length+1:end) = gamma;

ind = find(T_guess ~= 0);

%inertial position vector of new arrival position
for aa = 1:length(ang)
    r_ell(:,aa) = rf_pqw + re(aa)*cos(ang(aa))*vunit + re(aa)*
        sin(ang(aa))*gunit;
end

for dd = 1:length(r_guess)
```

```
        rg_pqw = DU*[r_guess(dd)*cos(theta_guess(dd));r_guess(dd)*
            sin(theta_guess(dd));0];
        [rgi(dd,:)] = PQW_to_IJK(rg_pqw,[],inc,RAAN,w);
    end

    for ee = 1:length(ang)
        rell_pqw = [r_ell(1,ee)*DU;r_ell(2,ee)*DU;0];
        rnom_pqw = norm(r0)*[cos(ang(ee));sin(ang(ee));0];
        [rell_ijk(:,ee)] = PQW_to_IJK(rell_pqw,[],inc,RAAN,w);
        [rnom_ijk(ee,:)] = PQW_to_IJK(rnom_pqw,[],inc,RAAN,w);
    end
    %% GPOPS RUN
    % variables from PSo phase
    r1 = 1;
    rf = norm(rt_pqw);
    rmax = r1 + be/DU;
    rmin = r1 - be/DU;
    thetaf_min = theta_rf - atan(ae/norm(r0));
    thetaf_max = theta_rf + atan(ae/norm(r0));

    %colocation points and fraction
    colnum = 4;
    colp = 20;

    % Control and time boundaries
    if phi > pi
        umin =  0; % minimum control angle
        umax =  2*pi; % maximum control angle
    else
        umin =  -pi; % minimum control angle
        umax =  pi; % maximum control angle
    end
```

```
Tmax = 2*0.0001160;
Tmin = Tmax/1000;

% GPOPS Setup
% Phase 1 Information
iphase = 1;
limits(iphase).intervals = 1;
limits(iphase).nodesperint = 100;
bounds.phase(iphase).initialtime.lower = t0min;
bounds.phase(iphase).initialtime.upper = t0max;
bounds.phase(iphase).finaltime.lower = tf1/TU;
bounds.phase(iphase).finaltime.upper = tf1/TU;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2
    0];
bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2
    1.1];
bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU
    -0.2 0];
bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.1];
bounds.phase(iphase).control.lower = [Tmin umin];
bounds.phase(iphase).control.upper = [Tmax umax];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.parameter.lower = 0;
bounds.parameter.upper = 2*pi;
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
```

```
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [0 0 Rmin/DU Rmin/DU]; % None
        bounds.eventgroup(iphase).upper = [0 0 Rmax/DU Rmax/DU]; % None
        % GUESS SOLUTION
        guess.phase(iphase).time = time_guess;
        guess.phase(iphase).state(:,1) = r_guess;
        guess.phase(iphase).state(:,2) = theta_guess;
        guess.phase(iphase).state(:,3) = vr_guess;
        guess.phase(iphase).state(:,4) = vtheta_guess;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_guess;
        guess.phase(iphase).control(:,2) = B_guess;
        guess.parameter = phi;
        guess.phase(iphase).integral = LT_DV;

        %auxiliary data
        auxdata.MU = MU2;
        auxdata.ae = ae1;
        auxdata.be = be1;
        auxdata.rf_pqw = rf_pqw;
        auxdata.vunit = vunit;
        auxdata.gunit = gunit;

        % NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae" required
        setup.name  = ['TIME_FIXED_INTERCEPT' ];

        setup.functions.continuous = @LT_RTM_Continuous;
        setup.functions.endpoint = @LT_RTM_Endpoint;
        setup.nlp.solver = 'ipopt';
        setup.mesh.maxiteration = 10;
        setup.mesh.tolerance = 1e-12;
```

```
        setup.mesh.colpointsmin = 40;
        setup.mesh.colpointsmax = 200;
        setup.mesh.phase(iphase).colpoints = colnum*ones(1,colp);
        setup.mesh.phase(iphase).fraction = (1/colp)*ones(1,colp);
        setup.bounds = bounds;
        setup.guess = guess;
        setup.auxdata = auxdata;
        setup.mesh.method = 'RPMintegration';
        setup.derivatives.supplier = 'sparseFD';
        setup.derivativelevel ='second';
        setup.dependencies = 'sparseNaN';
        setup.scales = 'none';

        output = gpops2(setup);
        solution = output.result.solution;

        r_GPOPS = solution.phase.state(:,1);
        theta_GPOPS = solution.phase.state(:,2);
        Vr_GPOPS = solution.phase.state(:,3);
        Vt_GPOPS = solution.phase.state(:,4);
        lambda_r = solution.phase.costate(:,1);
        lambda_theta = solution.phase.costate(:,2);
        lambda_Vr = solution.phase.costate(:,3);
        lambda_Vt = solution.phase.costate(:,4);
        tvec = solution.phase.time;

        thetadot_GPOPS = Vt_GPOPS./r_GPOPS;
        T_GPOPS = solution.phase.control(:,1);
        Beta_GPOPS = solution.phase.control(:,2);
        phi_GPOPS = solution.parameter;
        re_GPOPS = ae1*be1/sqrt((be1*cos(phi_GPOPS))^2 + (ae1*sin(
            phi_GPOPS))^2);
```

```
Cost = solution.phase.integral*DU/TU*1000

rt = rf_pqw*DU + re_GPOPS*DU*cos(phi_GPOPS)*vunit + re_GPOPS*DU*
    sin(phi_GPOPS)*gunit;


%%
clear setup guess bounds

colnum = 4;
colp = 40;
% GPOPS Setup
% Phase 1 Information
iphase = 1;
limits(iphase).intervals = 1;
limits(iphase).nodesperint = 100;
bounds.phase(iphase).initialtime.lower = t0min;
bounds.phase(iphase).initialtime.upper = t0max;
bounds.phase(iphase).finaltime.lower = tf1/TU;
bounds.phase(iphase).finaltime.upper = tf1/TU;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2
    0];
bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2
    1.1];
bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU
    -0.2 0];
```

```
bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.1];
bounds.phase(iphase).control.lower = [0 umin];
bounds.phase(iphase).control.upper = [Tmax umax];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.parameter.lower = 0;
bounds.parameter.upper = 2*pi;
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1;
bounds.eventgroup(iphase).lower = [0 0 Rmin/DU Rmin/DU]; % None
bounds.eventgroup(iphase).upper = [0 0 Rmax/DU Rmax/DU]; % None
% bounds.eventgroup(iphase).lower = [0 0]; % None
% bounds.eventgroup(iphase).upper = [0 0]; % None
% GUESS SOLUTION
guess.phase(iphase).time = tvec;
guess.phase(iphase).state(:,1) = r_GPOPS;
guess.phase(iphase).state(:,2) = theta_GPOPS;
guess.phase(iphase).state(:,3) = Vr_GPOPS;
guess.phase(iphase).state(:,4) = Vt_GPOPS;
% Control guess :
guess.phase(iphase).control(:,1) = T_GPOPS;
guess.phase(iphase).control(:,2) = Beta_GPOPS;
guess.parameter = phi_GPOPS;
guess.phase(iphase).integral = LT_DV;

%auxiliary data
auxdata.MU = MU2;
auxdata.ae = ae1;
auxdata.be = be1;
auxdata.rf_pqw = rf_pqw;
auxdata.vunit = vunit;
```

```
        auxdata.gunit = gunit;

        % NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
            required
        r0string = num2str(norm(r0vec));
        aestr = num2str(ae);
        itstr = num2str(PSO_data(cc,end));
        tempstr = [aestr itstr];
        aestring = num2str(tempstr);
        setup.name  = ['SinglePass' r0string aestring];

        setup.functions.continuous = @LT_RTM_Continuous;
        setup.functions.endpoint = @LT_RTM_Endpoint;
        setup.nlp.solver = 'ipopt';
        setup.mesh.maxiteration = 50;
        setup.mesh.tolerance = 1e-12;
        setup.mesh.colpointsmin = 40;
        setup.mesh.colpointsmax = 200;
        setup.mesh.phase(iphase).colpoints = colnum*ones(1,colp);
        setup.mesh.phase(iphase).fraction = (1/colp)*ones(1,colp);
        setup.bounds = bounds;
        setup.guess = guess;
        setup.auxdata = auxdata;
        setup.mesh.method = 'RPMintegration';
        setup.derivatives.supplier = 'sparseFD';
        setup.derivativelevel ='second';
        setup.dependencies = 'sparseNaN';
        setup.scales = 'none';

        output = gpops2(setup);
        solution2 = output.result.solution;

```

```
r_GPOPS2 = solution2.phase.state(:,1);
theta_GPOPS2 = solution2.phase.state(:,2);
Vr_GPOPS2 = solution2.phase.state(:,3);
Vt_GPOPS2 = solution2.phase.state(:,4);
lambda_r2 = solution2.phase.costate(:,1);
lambda_theta2 = solution2.phase.costate(:,2);
lambda_Vr2 = solution2.phase.costate(:,3);
lambda_Vt2 = solution2.phase.costate(:,4);
tvec2 = solution2.phase.time;

thetadot_GPOPS2 = Vt_GPOPS2./r_GPOPS2;
T_GPOPS2 = solution2.phase.control(:,1);
Beta_GPOPS2 = solution2.phase.control(:,2);
phi_GPOPS2 = solution2.parameter;
re_GPOPS2 = ae1*be1/sqrt((be1*cos(phi_GPOPS2))^2 + (ae1*sin(phi_GPOPS2))^2);

Cost2 = solution2.phase.integral*DU/TU*1000



fprintf(fid,'%i\t %10.5f\t %4.3f\t %4.3f\t %4.3f\t %6.5f\t %6.5f\t %4.3f\r\n',ae,TOF,phi,Vt_mag,fpa_t,PSO_data(cc,6)*DU/TU,Cost2*DU/TU*1000,phi_GPOPS2);



optans.ics = struct('r0',r0vec,'v0',v0vec,'t0',t0,'ae',ae,'be',be,'Rmax',Rmax,'Rmin',Rmin,'rf1',rf1,'vf1',vf1,'tf1',tf1,'latlim',latlim,'longlim',longlim,'GMST0',GMST0,...
     'inc',inc,'RAAN',RAAN,'w',w,'ang',ang);
optans.scale = struct('TU',TU,'DU',DU,'MU',MU2);
```

```
        optans.entry = struct('lat_enter',lat_enter,'long_enter',
            long_enter,'lat_exit',lat_exit,'long_exit',long_exit,'r_ell'
            ,r_ell,'rtijk',rt_ijk,'vtijk',vt_ijk);
        optans.phase = struct('state',solution2.phase(1).state,'costate'
            ,solution2.phase(1).costate,'control',solution2.phase(1).
            control,'time',tvec2);
        optans.parameter = solution2.parameter;

        dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Single Pass\Images\';
        dir2 = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Single Pass\Data\';

        tend = toc(tstart);

        exflag = output.result.nlpinfo;



        if cc == 1
        fprintf(fid2,'%s %i %s %i %s\r\n','colpoint = ',colnum,'(1,',
            colp,')');
        end

        if exflag == 0
        fprintf(fid2,'%i\t %i  \t %4.3f\t %4.3f\t %6.5f\t %6.5f\t %6.5f
            %5.2f\t %i\t\r\n',...
             norm(r0),ae,phi,phi_GPOPS2,PSO_data(cc,6),Cost,Cost2,tend,
                exflag);

        [optfin] = LT_SINGLE_PASS_PLOTS(optans,r0string,aestring,dir);

```

```
        dataname = ['SinglePass' r0string aestring];

        save(strcat(dir2,[dataname]),'output');
        end

        close all
        clear optfin

    end
end
```

E.1.2.2 Single Pass LTRTM Equations of Motion and Cost Function

```
function phaseout = LT_RTM_Continuous(input)

s = input.phase.state;
u = input.phase.control;

%% Equations of Motion
%
    --------------------------------------------------------------------------

r = s(:,1);
vr = s(:,3);
vtheta = s(:,4);

T = u(:,1);
B = u(:,2);

MU2 = input.auxdata.MU;

r_dot = vr;
theta_dot = vtheta./r;
```

```
vr_dot = (vtheta.^2)./r - MU2./(r.^2) + T.*sin(B);
vtheta_dot = -vtheta.*vr./r + T.*cos(B);

% Form matrix output
daeout = [r_dot theta_dot vr_dot vtheta_dot];

phaseout.dynamics = daeout;
% ----------------------------------------------------------------------------

%% Cost Function
phaseout.integrand = T;
```

E.1.2.3 Single Pass LTRTM Constraints

```
function output = LT_RTM_Endpoint(input)


%% Cost Function Evaluation
% ----------------------------------------------------------------------------

J = input.phase(1).integral;
output.objective  = J;
% ----------------------------------------------------------------------------

%% Event Constraints

t0 = input.phase(1).initialtime;
tf = input.phase(1).finaltime;
x0 = input.phase(1).initialstate;
xf = input.phase(1).finalstate;
```

```
rf = xf(1);
thetaf = xf(2);
Vrf = xf(3);
Vtf = xf(4);

p = input.parameter;
phi = p(1);

ae1 = input.auxdata.ae;
be1 = input.auxdata.be;
MU2 = input.auxdata.MU;
rf_pqw = input.auxdata.rf_pqw;
vunit = input.auxdata.vunit;
gunit = input.auxdata.gunit;

term1 = (be1*cos(phi))^2 + (ae1*sin(phi))^2;

re = ae1*be1/sqrt(term1);

rt = rf_pqw + re*cos(phi)*vunit + re*sin(phi)*gunit;

%final position constraints
event1 = rf*cos(thetaf) - rt(1);
event2 = rf*sin(thetaf) - rt(2);

% output.eventgroup(1).event = [event1 event2];

%apogee and perigee constraints
Vf_mag = sqrt(Vrf^2 + Vtf^2);
fpa = atan(Vrf/Vtf);

```

```
vt = Vf_mag*[-sin(thetaf-fpa);cos(thetaf-fpa);0];

[a,ecc,~,~,~,~] = RV2COE_MU(rt,vt,MU2);
Ra = a*(1+ecc);
Rp = a*(1-ecc);

event3 = Ra;
event4 = Rp;

output.eventgroup(1).event = [event1 event2 event3 event4];
```

E.2 Double Pass LTRTMs

E.2.1 Particle Swarm Algorithms

E.2.1.1 Double Pass LTRTM PSO Driver

```
wgs84data
global MU
OmegaEarth = 0.000072921151467;

for bb = 10:10

    t0 = 0;
    GMST0 = 0;
    latlim = [-10 10]*pi/180;
    longlim = [-50 -10]*pi/180;

    r0vec = [7300;0;0];
    v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];

    [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
    period = 2*pi*sqrt(a^3/MU);

    aevec = [150 140 130 120 110 100 90 80 70 60 50];
```

```
bevec = [15 14 13 12 11 10 9 8 7 6 5];
Rmaxvec = norm(r0vec)+50;
Rminvec = norm(r0vec) - 50;

DU = norm(r0vec);
TU = period/(2*pi);
MU2 = MU*TU^2/DU^3;

m0 = 1000;
r0 = r0vec;
v0 = v0vec;
Rmax = Rmaxvec;
Rmin = Rminvec;

%Energy of most elliptical orbit
ab = (Rmax + Rmin)/2; %semi-major axis of orbit
Eb = -MU/(2*ab);   %energy of orbit
Vmax = sqrt(2*(MU/Rmin + Eb));
Vmin = sqrt(2*(MU/Rmax + Eb));

fid = fopen([dir 'PSODoublePassDataFinal_06012014.txt'],'a');
state0=[r0 v0];

Tmax = 2e-3;
swarm = 40;
iter = 1000;
prec = [5;5;5;9];

if bb == 1

    fprintf(fid,'%s %i\r\n','r0 (km) =',r0vec(1));
    fprintf(fid,'%s %i\r\n','swarm =',swarm);
```

```
        fprintf(fid,'%s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t\r\n','TOF','Phi','Vf','fpa','TOF2','Phi2','Vf2','fpa2','DV','DV2','DVTOT','iter','iter2','iterTOT','time','time2','timeTOT');
        fprintf(fid,'%s\r\n','----------------------------------------------------------------------------');
    end

    if bb == 10
        endval = 1;
    else
        endval = 20;
    end

    ae = aevec(bb);
    be = bevec(bb);

    [rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim);

    for aa = 1:endval

        tstart = tic;

        [JGmin,Jpbest,gbest,x,k] = LT_RTM_PSO_TFIXED(3,[0 2*pi;Vmin Vmax;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf1,vf1,tf1,ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);

        Cost1 = JGmin*DU/TU*1000

```

```
        tend = toc(tstart)

        tstart2 = tic;

        [~,~,~,~,~,~,~,~,~,~,~,rt_ijk,vt_ijk] = Single_LT_Maneuver(rf1,vf1,tf1,gbest(1),ae,be,gbest(2),gbest(3),DU,TU,MU2);

        [rf2,vf2,tf2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*tf1,0,latlim,longlim);

        [JGmin2,Jpbest2,gbest2,x2,k2] = LT_RTM_PSO_TFIXED(3,[0 2*pi;Vmin Vmax;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf2,vf2,tf2,ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);

        Cost2 = JGmin2*DU/TU*1000

        CostTOT = Cost1 + Cost2

        tend2 = toc(tstart2)



        fprintf(fid,'%i\t %i \t %10.5f\t %6.5f\t %7.6f\t %6.5f\t %10.5f\t %6.5f\t %7.6f\t %6.5f\t %7.6f\t %7.6f\t %7.6f\t %i\t %i\t %i\t %4.1f\t %4.1f\t %4.1f\r\n',...
         norm(r0),ae,tf1,gbest(1),gbest(2),gbest(3),tf2,gbest2(1),gbest2(2),gbest2(3),Cost1,Cost2,CostTOT,k,k2,k+k2,tend,tend2,tend+tend2);
        tend + tend2

```

```
        clear tstart JGmin Jpbest gbest x k Cost1 tend tstart2 rt_ijk vt_ijk rf2 vf2 tf2 JGmin2 Jpbest2 gbest2 x2 k2  Cost2 CostTOT tend2
    end

end
```

E.2.2 Direct Collocation Algorithms

E.2.2.1 Double Pass LTRTM Driver

```
for zz = 1:1
    clear guess setup limits output
    close all
    clc

    if zz == 1
%          load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\Double Pass\Journal Data\data6800_LT_2RTMsort.mat')
%          [ind0] = find(data6800_LT_2RTMsort(:,1) ~= 0);
%          PSO_data = data6800_LT_2RTMsort(ind0,:);
        load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\Double Pass\Journal Data\data6800_LT_2RTM_2ndTier.mat')
        [ind0] = find(data6800_LT_2RTM_2ndTier(:,1) ~= 0);
        PSO_data = data6800_LT_2RTM_2ndTier(ind0,:);
        cmax = length(PSO_data);
        cmin = 1;
        rmag = 6800;
    elseif zz == 2
%          load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\Double Pass\Journal Data\data7300_LT_2RTMsort.mat')
%          [ind0] = find(data7300_LT_2RTMsort(:,1) ~= 0);
%          PSO_data = data7300_LT_2RTMsort(ind0,:);
```

```
        load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
           Double Pass\Journal Data\data7300_LT_2RTM_2ndTier.mat')
        [ind0] = find(data7300_LT_2RTM_2ndTier(:,1) ~= 0);
        PSO_data = data7300_LT_2RTM_2ndTier(ind0,:);
        cmax = length(PSO_data);
        cmin = 1;
        rmag = 7300;
    end

    tstart = tic;

    for cc = cmin:cmax
        fid = fopen('PSO_to_GPOPS_2RTM.txt','a');
        clear guess setup limits output
        close all
        clc

        t0 = 0;
        GMST0 = 0;
        latlim = [-10 10]*pi/180;
        longlim = [-50 -10]*pi/180;

        wgs84data
        global MU2 MU
        OmegaEarth = 0.000072921151467;
        r0vec = [rmag;0;0];
        v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];

        [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
        period = 2*pi*sqrt(a^3/MU);

        swarm = 30;
```

```
iter = 1000;
Rmaxvec = rmag + 50;
Rminvec = rmag - 50;
prec = [2;5;16];

r0 = r0vec;
v0 = v0vec;
Rmax = Rmaxvec;
Rmin = Rminvec;

ae = PSO_data(cc,2);
be = ae/10;

TOF = PSO_data(cc,3);
phi = PSO_data(cc,4);
Vt_mag = PSO_data(cc,5);
fpa_t = PSO_data(cc,6);

tstart = tic;

[rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
    long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
    ;

DU = norm(rf1);
TU = period/(2*pi);

ae1 = ae/DU;
be1 = be/DU;
r01 = norm(r0)/DU;
MU2 = MU*TU^2/DU^3;
t0min = 0; % minimum initial time
```

```
    t0max = 0; % maximum initial time
    tfmin = tf1; % minimum final time
    tfmax = tf1;
    n0 = sqrt(MU2/(norm(r0)/DU)^3);

    %% First Maneuver
    [LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,
        Tvec,TOF_calc,rt_ijk,vt_ijk,rt_pqw,vt_pqw,r0_pqw,v0_pqw] =
        Single_LT_Maneuver(rf1,vf1,TOF,phi,ae,be,Vt_mag,fpa_t,DU,TU,
        MU2);

    delt = (tf1 - TOF)/TU;
    time_mod = Tvec + delt;

    [rf_pqw,vf_pqw] = IJK_to_PQW(rf1,vf1,inc,RAAN,w);
    rf_pqw = rf_pqw/DU;
    vf_pqw = vf_pqw/DU*TU;



    vunit = vf_pqw/norm(vf_pqw);
    hfp = cross(rf_pqw,vf_pqw);
    hunit = hfp/norm(hfp);

    gunit = cross(vunit,hunit);

    ang = (0:0.001:2*pi);
    re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);



    theta_rf = atan2(rf_pqw(2),rf_pqw(1));
    if theta_rf < 0
        theta_rf = 2*pi + theta_rf;
```

```
        end
        [rtest] = IJK_to_PQW(r0,v0,inc,RAAN,w);
        theta0 = atan2(rtest(2),rtest(1));

        theta_mod = thetaf_int + atan2(r0_pqw(2),r0_pqw(1));

        coast_length = 1;

        time_guess = zeros(coast_length+length(time_mod),1);
        time_guess(1:coast_length) = 0;
        time_guess(coast_length+1:end) = time_mod;
        theta_guess = zeros(coast_length+length(theta_mod),1);
        theta_guess(coast_length+1:end) = theta_mod;
        theta_guess(1:coast_length) = theta0;
        r_guess = zeros(coast_length+length(theta_mod),1);
        r_guess(1:coast_length) = norm(r0)/DU;
        r_guess(coast_length+1:end) = r;
        vr_guess = zeros(coast_length+length(theta_mod),1);
        vr_guess(1:coast_length) = 0;
        vr_guess(coast_length+1:end) = rdot;
        vtheta_guess = zeros(coast_length+length(theta_mod),1);
        vtheta_guess(1:coast_length) = sqrt(MU2/(norm(r0)/DU));
        vtheta_guess(coast_length+1:end) = r.*theta_dot;
        T_guess = zeros(coast_length+length(time_mod),1);
        T_guess(1:coast_length) = 0;
        T_guess(coast_length+1:end) = T_a;
        B_guess = zeros(coast_length+length(time_mod),1);
        B_guess(1:coast_length) = 0;
        B_guess(coast_length+1:end) = gamma;

        ind = find(T_guess ~= 0);

```

```
%inertial position vector of new arrival position
for aa = 1:length(ang)
    r_ell(:,aa) = rf_pqw + re(aa)*cos(ang(aa))*vunit + re(aa)*sin(ang(aa))*gunit;
end

for dd = 1:length(r_guess)
    rg_pqw = DU*[r_guess(dd)*cos(theta_guess(dd));r_guess(dd)*sin(theta_guess(dd));0];
    [rgi(dd,:)] = PQW_to_IJK(rg_pqw,[],inc,RAAN,w);
end

for ee = 1:length(ang)
    rell_pqw = [r_ell(1,ee)*DU;r_ell(2,ee)*DU;0];
    rnom_pqw = norm(r0)*[cos(ang(ee));sin(ang(ee));0];
    [rell_ijk(:,ee)] = PQW_to_IJK(rell_pqw,[],inc,RAAN,w);
    [rnom_ijk(ee,:)] = PQW_to_IJK(rnom_pqw,[],inc,RAAN,w);
end


%% determine limits on subsequent passes into exclusion zone
% assume upper limit based on circular orbit with phi = pi/2
% assume lower limit based on circular orbit with phi = 2pi/2
phi_low = 3*pi/2;
phi_upp = pi/2;

[rf_upp,vf_upp] = Single_LT_Limits(rf1,vf1,phi_upp,ae,be,DU,TU);

[rf2_upp,vf2_upp,tf2_upp] = zone_entry_exit2(rf_upp,vf_upp,GMST0+OmegaEarth*tf1,0,latlim,longlim);

ang_upp = sqrt(MU/norm(rf_upp)^3)*tf2_upp;
```

```
        thetaf2_max = theta_guess(end) + ang_upp;
        tf2_max = (tf1 + tf2_upp)/TU;


        [rf_low,vf_low] = Single_LT_Limits(rf1,vf1,phi_low,ae,be,DU,TU);

        [rf2_low,vf2_low,tf2_low] = zone_entry_exit2(rf_low,vf_low,GMST0
            +OmegaEarth*tf1,0,latlim,longlim);

        ang_low = sqrt(MU/norm(rf_low)^3)*tf2_low;

        thetaf2_min = theta_guess(end) + ang_low;
        tf2_min = (tf1 + tf2_low)/TU;



        %% Second Maneuver
        [rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,
            lat_exit2,long_exit2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0
            +OmegaEarth*tf1,0,latlim,longlim);


        TOF2 = PSO_data(cc,7);
        phi2 = PSO_data(cc,8);
        Vt_mag2 = PSO_data(cc,9);
        fpa_t2 = PSO_data(cc,10);

        [LT_DV2,maxT2,r2,gamma2,T_a2,thetaf_int2,theta_dot2,theta_ddot2,
            rdot2,Tvec2,TOF_calc2,rt_ijk2,vt_ijk2] = Single_LT_Maneuver(
            rf2,vf2,TOF2,phi2,ae,be,Vt_mag2,fpa_t2,DU,TU,MU2);

```

```
        theta02 = theta_guess(end);

        delt2 = (tf2 - TOF2)/TU;
        time_mod2 = Tvec2 + delt2 + time_guess(end);

        [rf_pqw2,vf_pqw2] = IJK_to_PQW(rf2,vf2,inc,RAAN,w);
        rf_pqw2 = rf_pqw2/DU;
        vf_pqw2 = vf_pqw2/DU*TU;

        vunit2 = vf_pqw2/norm(vf_pqw2);
        hfp2 = cross(rf_pqw2,vf_pqw2);
        hunit2 = hfp2/norm(hfp2);

        gunit2 = cross(vunit2,hunit2);



        %inertial position vector of new arrival position
        for bb = 1:length(ang)
            r_ell2(:,bb) = rf_pqw2 + re(bb)*cos(ang(bb))*vunit2 + re(bb)
                *sin(ang(bb))*gunit2;
        end

        [rt_pqw2,vt_pqw2] = IJK_to_PQW(rt_ijk2,vt_ijk2,inc,RAAN,w);
        ang_mod2 = atan2(rt_pqw2(2),rt_pqw2(1));
        if ang_mod2 < 0
            ang_mod2 = ang_mod2 + 2*pi;
        end
        ang_mod1 = atan2(rt_pqw(2),rt_pqw(1));

        diff = ang_mod2 - ang_mod1;
        diff2 = thetaf_int2(end) - thetaf_int2(1);
        theta_diff = diff - diff2;
```

```
        theta_mod2 = thetaf_int2 + theta_diff + theta_guess(end);

        coast_length = 1;

        time_guess2 = zeros(coast_length+length(time_mod2),1);
        time_guess2(1:coast_length) = time_guess(end);
        time_guess2(coast_length+1:end) = time_mod2;
        theta_guess2 = zeros(coast_length+length(theta_mod2),1);
        theta_guess2(coast_length+1:end) = theta_mod2;
        theta_guess2(1:coast_length) = theta02;
        r_guess2 = zeros(coast_length+length(theta_mod2),1);
        r_guess2(1:coast_length) = r_guess(end);
        r_guess2(coast_length+1:end) = r2;
        vr_guess2 = zeros(coast_length+length(theta_mod2),1);
        vr_guess2(1:coast_length) = vr_guess(end);
        vr_guess2(coast_length+1:end) = rdot2;
        vtheta_guess2 = zeros(coast_length+length(theta_mod2),1);
        vtheta_guess2(1:coast_length) = vtheta_guess(end);
        vtheta_guess2(coast_length+1:end) = r2.*theta_dot2;
        T_guess2 = zeros(coast_length+length(time_mod2),1);
        T_guess2(1:coast_length) = 0;
        T_guess2(coast_length+1:end) = T_a2;
        B_guess2 = zeros(coast_length+length(time_mod2),1);
        B_guess2(1:coast_length) = B_guess(end);
        B_guess2(coast_length+1:end) = gamma2;

        ind2 = find(T_guess2 ~= 0);

        nom_orb2_time = [(0:1:tf2) tf2];

        [at,et,it,Ot,ot,nut]= RV2COE(rt_ijk,vt_ijk);
```

```
        for ee = 1:length(nom_orb2_time)
            [nutf] = nuf_from_TOF(nut,nom_orb2_time(ee),at,et);
            [Rdum(:,ee),Vdum] = COE2RV(at,et,it,Ot,ot,nutf);
            [nom_orb2_R] = IJK_to_PQW(Rdum(:,ee),Vdum,inc,RAAN,w);

            ROrb2_PQW(ee,:) = nom_orb2_R;

        end
        if nutf < nut
            nutf = nutf + 2*pi;
        end

        %Angle of expected 2nd pass entry location into exclusion zone
        thetaf2 = (nutf - nut) + 0;



        %% GPOPS RUN (1st Run Through assigns a non-zero minimum thrust
           to help GPOPS-II converge)
        % variables from PSo phase
        r1 = 1;
        rf = norm(rt_pqw);
        rmax = r1 + be/DU;
        rmin = r1 - be/DU;
        thetaf_min = theta_rf - atan(ae/norm(r0));
        thetaf_max = theta_rf + atan(ae/norm(r0));

        % Control and time boundaries
        umin =  -pi; % minimum control angle
        umax =  pi; % maximum control angle
        Tmax = 2*0.0001160;
        Tmin = Tmax/1000;
```

```
umin1 = -0.5;
umin2 = -0.5;
umax1 = 2*pi+0.5;
umax2 = 2*pi+0.5;

%colocation points and fraction
colnum = 4;
colp = 40;

% GPOPS Setup
% Phase 1 Information
iphase = 1;
limits(iphase).intervals = 1;
limits(iphase).nodesperint = 100;
bounds.phase(iphase).initialtime.lower = t0min;
bounds.phase(iphase).initialtime.upper = t0max;
bounds.phase(iphase).finaltime.lower = tf1/TU;
bounds.phase(iphase).finaltime.upper = tf1/TU;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2
    0];
bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2
    1.2];
bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU
    -0.2 0];
bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.2];
bounds.phase(iphase).control.lower = [Tmin umin1];
```

```
        bounds.phase(iphase).control.upper = [Tmax umax1];
        % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
        bounds.phase(iphase).path.lower = []; % None
        bounds.phase(iphase).path.upper = []; % None
        bounds.phase(iphase).integral.lower = 0;
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/
            DU]; % None
        bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/
            DU]; % None
        % GUESS SOLUTION
        guess.phase(iphase).time = time_guess;
        guess.phase(iphase).state(:,1) = r_guess;
        guess.phase(iphase).state(:,2) = theta_guess;
        guess.phase(iphase).state(:,3) = vr_guess;
        guess.phase(iphase).state(:,4) = vtheta_guess;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_guess;
        guess.phase(iphase).control(:,2) = B_guess;
        guess.phase(iphase).integral = LT_DV;

        % Phase 2 Information (second Maneuver
        iphase = 2;
        limits(iphase).intervals = 1;
        limits(iphase).nodesperint = 100;
        bounds.phase(iphase).initialtime.lower = tf1/TU;
        bounds.phase(iphase).initialtime.upper = tf1/TU;
        bounds.phase(iphase).finaltime.lower = tf2_min-1;
        bounds.phase(iphase).finaltime.upper = tf2_max+1;
        % LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
        bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.2
            0];
```

```
        bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.2 1.2];
        bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.2 0];
        bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.2 1.2];
        bounds.phase(iphase).state.lower = [r1-0.1 thetaf_min -0.2 0];
        bounds.phase(iphase).state.upper = [r1+0.1 thetaf2_max+1 0.2 1.2];
        bounds.phase(iphase).control.lower = [Tmin umin2];
        bounds.phase(iphase).control.upper = [Tmax umax2];
        bounds.parameter.lower = [0 0];
        bounds.parameter.upper = [2*pi 2*pi];
        bounds.phase(iphase).path.lower = []; % None
        bounds.phase(iphase).path.upper = []; % None
        bounds.phase(iphase).integral.lower = 0;
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; % None
        bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; % None
        % GUESS SOLUTION
        guess.phase(iphase).time = time_guess2;
        guess.phase(iphase).state(:,1) = r_guess2;
        guess.phase(iphase).state(:,2) = theta_guess2;
        guess.phase(iphase).state(:,3) = vr_guess2;
        guess.phase(iphase).state(:,4) = vtheta_guess2;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_guess2;
        guess.phase(iphase).control(:,2) = B_guess2;
        guess.parameter = [phi phi2];
        % guess.parameter = [phi];
```

```
guess.phase(iphase).integral = LT_DV2;

%auxiliary data
auxdata.MU = MU2;
auxdata.ae = ae1;
auxdata.be = be1;
auxdata.rf_pqw = rf_pqw;
auxdata.vunit = vunit;
auxdata.gunit = gunit;
auxdata.inc = inc;
auxdata.RAAN = RAAN;
auxdata.w = w;
auxdata.latlim = latlim;
auxdata.longlim = longlim;
auxdata.GMST0 = GMST0;
auxdata.OmegaEarth = OmegaEarth;
auxdata.DU = DU;
auxdata.TU = TU;

% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
    required
r0string = num2str(norm(r0vec));
aestr = num2str(ae);
itstr = num2str(PSO_data(cc,end));
tempstr = [aestr itstr];
aestring = num2str(tempstr);
setup.name  = ['DoublePass' r0string aestring];

setup.functions.continuous = @LT_2RTM_Continuous;
setup.functions.endpoint = @LT_2RTM_Endpoint;
setup.nlp.solver = 'ipopt';
setup.mesh.maxiteration = 10;
```

```
        setup.mesh.tolerance = 1e-10;
        setup.mesh.colpointsmin = 40;
        setup.mesh.colpointsmax = 400;
        for ival = 1:2
            setup.mesh.phase(ival).colpoints = colnum*ones(1,colp);
            setup.mesh.phase(ival).fraction = (1/colp)*ones(1,colp);
        end
        setup.bounds = bounds;
        setup.guess = guess;
        setup.auxdata = auxdata;
        setup.mesh.method = 'RPMintegration';
        setup.derivatives.supplier = 'sparseFD';
        setup.derivativelevel ='second';
        setup.dependencies = 'sparseNaN';
        setup.scales = 'none';

        output = gpops2(setup);
        solution = output.result.solution;
        %%
        %States and costates from phase 1 (first maneuver)
        r_GPOPS_P1 = solution.phase(1).state(:,1);
        theta_GPOPS_P1 = solution.phase(1).state(:,2);
        Vr_GPOPS_P1 = solution.phase(1).state(:,3);
        Vt_GPOPS_P1 = solution.phase(1).state(:,4);
        lambda_r_P1 = solution.phase(1).costate(:,1);
        lambda_theta_P1 = solution.phase(1).costate(:,2);
        lambda_Vr_P1 = solution.phase(1).costate(:,3);
        lambda_Vt_P1 = solution.phase(1).costate(:,4);
        tvec_P1 = solution.phase(1).time;

        thetadot_GPOPS_P1 = Vt_GPOPS_P1./r_GPOPS_P1;
        T_GPOPS_P1 = solution.phase(1).control(:,1);
```

```
        Beta_GPOPS_P1 = solution.phase(1).control(:,2);
        phi_GPOPS_P1 = solution.parameter(1);
        re_GPOPS_P1 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P1))^2 + (ae1*sin(phi_GPOPS_P1))^2);

        % --------------------------------------------------------------------------

        %States and Costates from pahse 2 (second maneuver)
        r_GPOPS_P2 = solution.phase(2).state(:,1);
        theta_GPOPS_P2 = solution.phase(2).state(:,2);
        Vr_GPOPS_P2 = solution.phase(2).state(:,3);
        Vt_GPOPS_P2 = solution.phase(2).state(:,4);
        lambda_r_P2 = solution.phase(2).costate(:,1);
        lambda_theta_P2 = solution.phase(2).costate(:,2);
        lambda_Vr_P2 = solution.phase(2).costate(:,3);
        lambda_Vt_P2 = solution.phase(2).costate(:,4);
        tvec_P2 = solution.phase(2).time;

        thetadot_GPOPS_P2 = Vt_GPOPS_P2./r_GPOPS_P2;
        T_GPOPS_P2 = solution.phase(2).control(:,1);
        Beta_GPOPS_P2 = solution.phase(2).control(:,2);
        phi_GPOPS_P2 = solution.parameter(2);
        re_GPOPS_P2 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P2))^2 + (ae1*sin(phi_GPOPS_P2))^2);
        % --------------------------------------------------------------------------

        Cost = (solution.phase(1).integral + solution.phase(2).integral)*DU/TU*1000;

```

```
%% GPOPS Run two (Minimum thrust is set to zero in run 2 to
    generate true optimal solution
clear guess setup bound limits

%colocation points and fraction
colnum = 4;
colp = 40;

% Phase 1 Information
iphase = 1;
limits(iphase).intervals = 1;
limits(iphase).nodesperint = 100;
bounds.phase(iphase).initialtime.lower = t0min;
bounds.phase(iphase).initialtime.upper = t0max;
bounds.phase(iphase).finaltime.lower = tf1/TU;
bounds.phase(iphase).finaltime.upper = tf1/TU;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.2
    0];
bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.2
    1.1];
bounds.phase(iphase).state.lower = [r1-0.1 theta_rf-n0*tf1/TU
    -0.2 0];
bounds.phase(iphase).state.upper = [r1+0.1 thetaf_max 0.2 1.1];
bounds.phase(iphase).control.lower = [0 umin1];
bounds.phase(iphase).control.upper = [Tmax umax1];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.phase(iphase).path.lower = []; % None
```

```
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1;
bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/DU]; % None
bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/DU]; % None
% GUESS SOLUTION
guess.phase(iphase).time = tvec_P1;
guess.phase(iphase).state(:,1) = r_GPOPS_P1;
guess.phase(iphase).state(:,2) = theta_GPOPS_P1;
guess.phase(iphase).state(:,3) = Vr_GPOPS_P1;
guess.phase(iphase).state(:,4) = Vt_GPOPS_P1;
% Control guess :
guess.phase(iphase).control(:,1) = T_GPOPS_P1;
guess.phase(iphase).control(:,2) = Beta_GPOPS_P1;
guess.phase(iphase).integral = LT_DV;

% Phase 2 Information (second Maneuver
iphase = 2;
limits(iphase).intervals = 1;
limits(iphase).nodesperint = 100;
bounds.phase(iphase).initialtime.lower = tf1/TU;
bounds.phase(iphase).initialtime.upper = tf1/TU;
bounds.phase(iphase).finaltime.lower = tf2_min-1;
bounds.phase(iphase).finaltime.upper = tf2_max+1;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.2 0];
bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.2 1.1];
```

```
        bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.2
             0];
        bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.2
            1.1];
        bounds.phase(iphase).state.lower = [r1-0.1 thetaf_min -0.2 0];
        bounds.phase(iphase).state.upper = [r1+0.1 thetaf2_max+1 0.2
            1.1];
        bounds.phase(iphase).control.lower = [0 umin2];
        bounds.phase(iphase).control.upper = [Tmax umax2];
        % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
        bounds.parameter.lower = [0 0];
        bounds.parameter.upper = [2*pi 2*pi];
        bounds.phase(iphase).path.lower = []; % None
        bounds.phase(iphase).path.upper = []; % None
        bounds.phase(iphase).integral.lower = 0;
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; %
            None
        bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; %
            None
        % GUESS SOLUTION
        guess.phase(iphase).time = tvec_P2;
        guess.phase(iphase).state(:,1) = r_GPOPS_P2;
        guess.phase(iphase).state(:,2) = theta_GPOPS_P2;
        guess.phase(iphase).state(:,3) = Vr_GPOPS_P2;
        guess.phase(iphase).state(:,4) = Vt_GPOPS_P2;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_GPOPS_P2;
        guess.phase(iphase).control(:,2) = Beta_GPOPS_P2;
        guess.parameter = [phi_GPOPS_P1 phi_GPOPS_P2];
        % guess.parameter = [phi];
        guess.phase(iphase).integral = Cost;
```

```
%auxiliary data
auxdata.MU = MU2;
auxdata.ae = ae1;
auxdata.be = be1;
auxdata.rf_pqw = rf_pqw;
auxdata.vunit = vunit;
auxdata.gunit = gunit;
auxdata.inc = inc;
auxdata.RAAN = RAAN;
auxdata.w = w;
auxdata.latlim = latlim;
auxdata.longlim = longlim;
auxdata.GMST0 = GMST0;
auxdata.OmegaEarth = OmegaEarth;
auxdata.DU = DU;
auxdata.TU = TU;

% NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
    required
r0string = num2str(norm(r0vec));
aestr = num2str(ae);
itstr = num2str(PSO_data(cc,end));
tempstr = [aestr itstr];
aestring = num2str(tempstr);
setup.name  = ['DoublePass' r0string aestring];

setup.functions.continuous = @LT_2RTM_Continuous;
setup.functions.endpoint = @LT_2RTM_Endpoint;
setup.nlp.solver = 'ipopt';
setup.mesh.maxiteration = 10;
setup.mesh.tolerance = 1e-10;
```

```
        setup.mesh.colpointsmin = 40;
        setup.mesh.colpointsmax = 400;
        for ival = 1:2
            setup.mesh.phase(ival).colpoints = colnum*ones(1,colp);
            setup.mesh.phase(ival).fraction = (1/colp)*ones(1,colp);
        end
        setup.bounds = bounds;
        setup.guess = guess;
        setup.auxdata = auxdata;
        setup.mesh.method = 'RPMintegration';
        setup.derivatives.supplier = 'sparseFD';
        setup.derivativelevel ='second';
        setup.dependencies = 'sparseNaN';
        setup.scales = 'none';

        output = gpops2(setup);
        solution2 = output.result.solution;
        %%
        %States and costates from phase 1 (first maneuver)
        r_GPOPS_P12 = solution2.phase(1).state(:,1);
        theta_GPOPS_P12 = solution2.phase(1).state(:,2);
        Vr_GPOPS_P12 = solution2.phase(1).state(:,3);
        Vt_GPOPS_P12 = solution2.phase(1).state(:,4);
        lambda_r_P12 = solution2.phase(1).costate(:,1);
        lambda_theta_P12 = solution2.phase(1).costate(:,2);
        lambda_Vr_P12 = solution2.phase(1).costate(:,3);
        lambda_Vt_P12 = solution2.phase(1).costate(:,4);
        tvec_P12 = solution2.phase(1).time;

        thetadot_GPOPS_P12 = Vt_GPOPS_P12./r_GPOPS_P12;
        T_GPOPS_P12 = solution2.phase(1).control(:,1);
        Beta_GPOPS_P12 = solution2.phase(1).control(:,2);
```

```
        ind1_large = find(Beta_GPOPS_P12 > 2*pi);
        ind1_small = find(Beta_GPOPS_P12 < 0);

        while isempty(ind1_large) == 0
            Beta_GPOPS_P12(ind1_large) = Beta_GPOPS_P12(ind1_large) - 2*
                pi;
            ind1_large = find(Beta_GPOPS_P12 > 2*pi);
        end

        while isempty(ind1_small) == 0
            Beta_GPOPS_P12(ind1_small) = Beta_GPOPS_P12(ind1_small) + 2*
                pi;
            ind1_small = find(Beta_GPOPS_P12 < 0);
        end

        phi_GPOPS_P12 = solution2.parameter(1);
        re_GPOPS_P12 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P12))^2 + (ae1*
            sin(phi_GPOPS_P12))^2);

        %States and Costates from pahse 2 (second maneuver)
        r_GPOPS_P22 = solution2.phase(2).state(:,1);
        theta_GPOPS_P22 = solution2.phase(2).state(:,2);
        Vr_GPOPS_P22 = solution2.phase(2).state(:,3);
        Vt_GPOPS_P22 = solution2.phase(2).state(:,4);
        lambda_r_P22 = solution2.phase(2).costate(:,1);
        lambda_theta_P22 = solution2.phase(2).costate(:,2);
        lambda_Vr_P22 = solution2.phase(2).costate(:,3);
        lambda_Vt_P22 = solution2.phase(2).costate(:,4);
        tvec_P22 = solution2.phase(2).time;

        thetadot_GPOPS_P22 = Vt_GPOPS_P22./r_GPOPS_P22;
        T_GPOPS_P22 = solution2.phase(2).control(:,1);
```

```
        Beta_GPOPS_P22 = solution2.phase(2).control(:,2);

        ind2_large = find(Beta_GPOPS_P22 > 2*pi);
        ind2_small = find(Beta_GPOPS_P22 < 0);

        while isempty(ind2_large) == 0
            Beta_GPOPS_P22(ind2_large) = Beta_GPOPS_P22(ind2_large) - 2*
                pi;
            ind2_large = find(Beta_GPOPS_P22 > 2*pi);
        end

        while isempty(ind2_small) == 0
            Beta_GPOPS_P22(ind2_small) = Beta_GPOPS_P22(ind2_small) + 2*
                pi;
            ind2_small = find(Beta_GPOPS_P22 < 0);
        end
        phi_GPOPS_P22 = solution2.parameter(2);
        re_GPOPS_P22 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P22))^2 + (ae1*
            sin(phi_GPOPS_P22))^2);

        Cost2 = (solution2.phase(1).integral + solution2.phase(2).
            integral)*DU/TU*1000;



        %%
        %
            ---------------------------------------------------------------------------

        % Determine entry condition for second maneuver
        rt = [r_GPOPS_P12(end)*cos(theta_GPOPS_P12(end));r_GPOPS_P12(end
            )*sin(theta_GPOPS_P12(end));0];

```

```
%apogee and perigee constraints
Vf_mag = sqrt(Vr_GPOPS_P12(end)^2 + Vt_GPOPS_P12(end)^2);
fpa = atan(Vr_GPOPS_P12(end)/Vt_GPOPS_P12(end));

%perifocal velocity
vt = Vf_mag*[-sin(theta_GPOPS_P12(end)-fpa);cos(theta_GPOPS_P12(
    end)-fpa);0];

[rt_ijk_P12,vt_ijk_P12] = PQW_to_IJK(rt,vt,inc,RAAN,w);
rt_ijk_P12 = rt_ijk_P12*DU;
vt_ijk_P12 = vt_ijk_P12*DU/TU;

[r2,v2,t2] = zone_entry_exit2(rt_ijk_P12,vt_ijk_P12,GMST0+
    OmegaEarth*tvec_P12(end)*TU,0,latlim,longlim);

[rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);

rf_pqw2 = rf_pqw2/DU;
vf_pqw2 = vf_pqw2/DU*TU;

vunit2 = vf_pqw2/norm(vf_pqw2);
hfp2 = cross(rf_pqw2,vf_pqw2);
hunit2 = hfp2/norm(hfp2);

gunit2 = cross(vunit2,hunit2);
%
    ----------------------------------------------------------------------



% for plotting purposes in PQW frame
```

```
    % --------------------------------------------------------------------------
    term12 = (be1*cos(phi_GPOPS_P22))^2 + (ae1*sin(phi_GPOPS_P22))^2;
    re2 = ae1*be1/sqrt(term12);

    rt2 = rf_pqw2 + re2*cos(phi_GPOPS_P22)*vunit2 + re2*sin(phi_GPOPS_P22)*gunit2;

    for aa = 1:length(ang)
        r_ell2(:,aa) = rf_pqw2 + re(aa)*cos(ang(aa))*vunit2 + re(aa)*sin(ang(aa))*gunit2;
    end


    %First maneuver inertial position and velocity
    for dd = 1:length(r_GPOPS_P12)
        %perifocal position vector
        rg_pqw = DU*[r_GPOPS_P12(dd)*cos(theta_GPOPS_P12(dd));r_GPOPS_P12(dd)*sin(theta_GPOPS_P12(dd));0];
        %velocity magnitude
        Vf_mag = sqrt(Vr_GPOPS_P12(dd)^2 + Vt_GPOPS_P12(dd)^2);
        fpa = atan(Vr_GPOPS_P12(dd)/Vt_GPOPS_P12(dd));

        %perifocal velocity
        vg_pqw = DU/TU*Vf_mag*[-sin(theta_GPOPS_P12(dd)-fpa);cos(theta_GPOPS_P12(dd)-fpa);0];

        [rgi(dd,:),vgi(dd,:)] = PQW_to_IJK(rg_pqw,vg_pqw,inc,RAAN,w);
    end
```

```
% actual arrival in exclusion zone location at tf1
[lat_act_enter,long_act_enter] = IJK_to_LATLONG(rgi(end,1),rgi(
    end,2),rgi(end,3),GMST0+OmegaEarth*tf1,0);
figure(1)
plot(long_act_enter*180/pi,lat_act_enter*180/pi,'bO')

%expected arrival condition in exclusion zone at tf2
[rf2exp,vf2exp,tf2exp,lat_enter2exp,long_enter2exp] =
    zone_entry_exit2(rgi(end,:),vgi(end,:),GMST0+OmegaEarth*(tf1
    ),t0,latlim,longlim);

plot(long_enter2exp*180/pi,lat_enter2exp*180/pi,'rO')

%Second maneuver inertial position and velocity2
for dd = 1:length(r_GPOPS_P22)
    %perifocal position vector
    rg_pqw2 = DU*[r_GPOPS_P22(dd)*cos(theta_GPOPS_P22(dd));
        r_GPOPS_P22(dd)*sin(theta_GPOPS_P22(dd));0];

    %velocity magnitude and flight path angle
    Vf_mag = sqrt(Vr_GPOPS_P22(dd)^2 + Vt_GPOPS_P22(dd)^2);
    fpa = atan(Vr_GPOPS_P22(dd)/Vt_GPOPS_P22(dd));

    %perifocal velocity
    vg_pqw2 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P22(dd)-fpa);cos(
        theta_GPOPS_P22(dd)-fpa);0];

    [rgi2(dd,:),vgi2(dd,:)] = PQW_to_IJK(rg_pqw2,vg_pqw2,inc,
        RAAN,w);

end
```

```
% actual arrival in exclusion zone location at tf2
[lat_act_enter2,long_act_enter2] = IJK_to_LATLONG(rgi2(end,1),
    rgi2(end,2),rgi2(end,3),GMST0+OmegaEarth*(tf1+tf2exp),0);
figure(1)
plot(long_act_enter2*180/pi,lat_act_enter2*180/pi,'bO')
%
% [a2,e2,i2,O2,o2,nu2]= RV2COE(rgi2(end,:),vgi2(end,:));



%save optimal path in structure
optans2.ics = struct('r0',r0vec,'v0',v0vec,'t0',t0,'ae',ae,'be',
    be,'Rmax',Rmax,'Rmin',Rmin,'latlim',latlim,'longlim',longlim
    ,'GMST0',GMST0,...
     'inc',inc,'RAAN',RAAN,'w',w,'ang',ang);
optans2.scale = struct('TU',TU,'DU',DU,'MU',MU2);
optans2.entry(1) = struct('lat_enter',lat_enter,'long_enter',
    long_enter,'r_ell',r_ell,'rtijk',rgi(end,:),'vtijk',vgi(end
    ,:),'rt_pqw',rg_pqw,'rf_pqw',rf_pqw,...
     'lat_act_enter',lat_act_enter,'long_act_enter',
        long_act_enter,'rf1',rf1,'vf1',vf1,'tf1',tf1);
optans2.entry(2) = struct('lat_enter',lat_enter2exp,'long_enter'
    ,long_enter2exp,'r_ell',r_ell2,'rtijk',rgi2(end,:),'vtijk',
    vgi2(end,:),'rt_pqw',rg_pqw2,'rf_pqw',rf_pqw2,...
     'lat_act_enter',lat_act_enter2,'long_act_enter',
        long_act_enter2,'rf1',rf2exp,'vf1',vf2exp,'tf1',tf2exp);
optans2.phase(1) = struct('state',solution2.phase(1).state,'
    costate',solution2.phase(1).costate,'control',solution2.
    phase(1).control,'time',tvec_P12,'rgi',rgi);
```

```
optans2.phase(2) = struct('state',solution2.phase(2).state,'
    costate',solution2.phase(2).costate,'control',solution2.
    phase(2).control,'time',tvec_P22,'rgi',rgi2);
optans2.parameter = solution2.parameter;

r0string = num2str(norm(r0vec));
aestr = num2str(ae);
itstr = num2str(PSO_data(cc,end));
tempstr = [aestr itstr];
aestring = num2str(tempstr);

dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
    Double Pass\Images\';
dir2 = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
    Double Pass\Data\';

tend = toc(tstart);

exflag = output.result.nlpinfo;

fid2 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust
     RTM\Double Pass\Data\PSO2GPOPSDoublePassDataB.txt','a');

fprintf(fid2,'%i\t %i\t %4.3f\t %4.3f\t %4.3f\t %4.3f\t%6.5f\t
    %6.5f\t %6.5f\t %6.2f\t %i\r\n',...
     norm(r0),ae,phi,phi_GPOPS_P12,phi2,phi_GPOPS_P22,PSO_data(cc
         ,13),Cost,Cost2,tend,exflag);

fid3 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust
     RTM\Double Pass\Data\DoublePassCostB.txt','a');
fprintf(fid3,'%i\t %i  \t %4.3f\t %4.3f\t %4.3f\t %i\t\r\n',...
```

```
            norm(r0),ae,solution2.phase(1).integral*DU/TU*1000,solution2
                .phase(2).integral*DU/TU*1000,Cost2,exflag);



        if exflag == 0
        %plot optimal results
        [optout] = LT_DOUBLE_PASS_PLOTS(optans2,r0string,aestring,dir);

        dataname = ['DoublePass' r0string aestring];

        %save data
        save(strcat(dir2,[dataname]),'output');
        end

        clear optans
        close all
    end
end
```

E.2.2.2 Double Pass LTRTM Equations of Motion and Cost Function

```
function phaseout = LT_2RTM_Continuous(input)

%% Phase 1

s1 = input.phase(1).state;
u1 = input.phase(1).control;

% Equations of Motion
%
    ----------------------------------------------------------------------------
```

```
r1 = s1(:,1);
vr1 = s1(:,3);
vtheta1 = s1(:,4);

T1 = u1(:,1);
B1 = u1(:,2);

MU2 = input.auxdata.MU;

r_dot1 = vr1;
theta_dot1 = vtheta1./r1;
vr_dot1 = (vtheta1.^2)./r1 - MU2./(r1.^2) + T1.*sin(B1);
vtheta_dot1 = -vtheta1.*vr1./r1 + T1.*cos(B1);

% Form matrix output
daeout1 = [r_dot1 theta_dot1 vr_dot1 vtheta_dot1];

phaseout(1).dynamics = daeout1;
%
    ---------------------------------------------------------------------------

% Cost Function
phaseout(1).integrand = T1;

%% Phase 2

s2 = input.phase(2).state;
u2 = input.phase(2).control;

% Equations of Motion
```

```
%
    -------------------------------------------------------------------------

r2 = s2(:,1);
vr2 = s2(:,3);
vtheta2 = s2(:,4);

T2 = u2(:,1);
B2 = u2(:,2);

r_dot2 = vr2;
theta_dot2 = vtheta2./r2;
vr_dot2 = (vtheta2.^2)./r2 - MU2./(r2.^2) + T2.*sin(B2);
vtheta_dot2 = -vtheta2.*vr2./r2 + T2.*cos(B2);

% Form matrix output
daeout2 = [r_dot2 theta_dot2 vr_dot2 vtheta_dot2];

phaseout(2).dynamics = daeout2;
%
    -------------------------------------------------------------------------

% Cost Function
phaseout(2).integrand = T2;
```

E.2.2.3 Double Pass LTRTM Constraints

```
function output = LT_2RTM_Endpoint(input)


%% Cost Function Evaluation
```

```
%
    ---------------------------------------------------------------------------
J = input.phase(1).integral + input.phase(2).integral;
output.objective  = J;
%
    ---------------------------------------------------------------------------
%% Event Constraints

%% Phase 1 (First Maneuver)
%phase 2 variables
tf1 = input.phase(1).finaltime;
xf1 = input.phase(1).finalstate;
p = input.parameter;
phi = p(1);

%phase 2 variables
t02 = input.phase(2).initialtime;
tf2 = input.phase(2).finaltime;
x02 = input.phase(2).initialstate;
xf2 = input.phase(2).finalstate;
phi2 = p(2);

rf = xf1(1);
thetaf = xf1(2);
Vrf = xf1(3);
Vtf = xf1(4);

ae1 = input.auxdata.ae; %semimajor axis of exclusion ellipse
be1 = input.auxdata.be; % semiminor axis of exclusion ellipse
MU2 = input.auxdata.MU; %gravitational parameter scaled by DU and TU
```

```
rf_pqw = input.auxdata.rf_pqw; % perifocal position vector of initial
    corssing into exclusion zone
vunit = input.auxdata.vunit; %perifocal unit velocity vector of initial
    crossing into exclusion zone
gunit = input.auxdata.gunit; %perifocal unit vetcor of initial crossing
    into exclusion zone

term1 = (be1*cos(phi))^2 + (ae1*sin(phi))^2;

re = ae1*be1/sqrt(term1);

rt = rf_pqw + re*cos(phi)*vunit + re*sin(phi)*gunit;

%final position constraints
event1 = rf*cos(thetaf) - rt(1);
event2 = rf*sin(thetaf) - rt(2);

%velocity magnitude and flight path angle
Vf_mag = sqrt(Vrf^2 + Vtf^2);
fpa = atan(Vrf/Vtf);

%perifocal velocity
vt = Vf_mag*[-sin(thetaf-fpa);cos(thetaf-fpa);0];

[a,ecc,~,~,~,~] = RV2COE_MU(rt,vt,MU2);
Ra = a*(1+ecc);
Rp = a*(1-ecc);

event3 = Ra;
event4 = Rp;

% Linkage Constraints
```

```
event1_link_state = x02 - xf1;
event1_link_time = t02 - tf1;

output.eventgroup(1).event = [event1_link_state event1_link_time event1
    event2 event3 event4];

%% Phase 2 (Second Maneuver)

% constant variables
inc = input.auxdata.inc; %inclination of initial orbit (used to convert
    everything into perifocal frame of initial orbit)
RAAN = input.auxdata.RAAN; %RAAN of initial orbit (used to convert
    everything into perifocal frame of initial orbit)
w = input.auxdata.w; %argument of perigee of initial orbit (used to
    convert everything into perifocal frame of initial orbit)
latlim = input.auxdata.latlim;
longlim = input.auxdata.longlim;
GMST0 = input.auxdata.GMST0;
OmegaEarth = input.auxdata.OmegaEarth;
DU = input.auxdata.DU;
TU = input.auxdata.TU;


rf2 = xf2(1);
thetaf2 = xf2(2);
Vrf2 = xf2(3);
Vtf2 = xf2(4);

%position and velocity of initial intercept in perifocal frame of
    initial
%orbit
```

```
if isnan(rf) == 1 || isnan(thetaf) == 1 || isnan(Vf_mag) == 1 || isnan(
    tf1) == 1 || isnan(phi) == 1
        event21 = NaN;
        event22 = NaN;
        event25 = NaN;
        event23 = NaN;
        event24 = NaN;
else
[rt_ijk,vt_ijk] = PQW_to_IJK(rt,vt,inc,RAAN,w);
rt_ijk = rt_ijk*DU;
vt_ijk = vt_ijk*DU/TU;

[r2,v2,t2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*tf1*TU,0,
    latlim,longlim);

[rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);

rf_pqw2 = rf_pqw2/DU;
vf_pqw2 = vf_pqw2/DU*TU;

vunit2 = vf_pqw2/norm(vf_pqw2);
hfp2 = cross(rf_pqw2,vf_pqw2);
hunit2 = hfp2/norm(hfp2);

gunit2 = cross(vunit2,hunit2);

term12 = (be1*cos(phi2))^2 + (ae1*sin(phi2))^2;
re2 = ae1*be1/sqrt(term12);

rt2 = rf_pqw2 + re2*cos(phi2)*vunit2 + re2*sin(phi2)*gunit2;

%final position constraints
```

```
event21 = rf2*cos(thetaf2) - rt2(1);
event22 = rf2*sin(thetaf2) - rt2(2);

%apogee and perigee constraints
Vf_mag2 = sqrt(Vrf2^2 + Vtf2^2);
fpa2 = atan(Vrf2/Vtf2);

%perifocal velocity
vt2 = Vf_mag2*[-sin(thetaf2-fpa2);cos(thetaf2-fpa2);0];

[a2,ecc2,~,~,~,~] = RV2COE_MU(rt2,vt2,MU2);
Ra2 = a2*(1+ecc2);
Rp2 = a2*(1-ecc2);

event23 = Ra2;
event24 = Rp2;

event25 = tf2 - (tf1 + t2/TU);
end

output.eventgroup(2).event = [event21 event22 event25 event23 event24];
```

E.3 Triple Pass LTRTMs

E.3.1 Particle Swarm Algorithms

E.3.1.1 Triple Pass LTRTM PSO Driver

```
wgs84data
global MU
OmegaEarth = 0.000072921151467;

for bb = 10:10

    t0 = 0;
```

```
GMST0 = 0;
latlim = [-10 10]*pi/180;
longlim = [-50 -10]*pi/180;

r0vec = [7300;0;0];
v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];

[a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
period = 2*pi*sqrt(a^3/MU);

aevec = [150 140 130 120 110 100 90 80 70 60 50];
bevec = [15 14 13 12 11 10 9 8 7 6 5];
Rmaxvec = norm(r0vec)+50;
Rminvec = norm(r0vec) - 50;

DU = norm(r0vec);
TU = period/(2*pi);
MU2 = MU*TU^2/DU^3;

m0 = 1000;
r0 = r0vec;
v0 = v0vec;
Rmax = Rmaxvec;
Rmin = Rminvec;

%Energy of most elliptical orbit
ab = (Rmax + Rmin)/2; %semi-major axis of orbit
Eb = -MU/(2*ab);   %energy of orbit
Vmax = sqrt(2*(MU/Rmin + Eb));
Vmin = sqrt(2*(MU/Rmax + Eb));

fid = fopen([dir 'PSODoublePassDataFinal_06012014.txt'],'a');
```

```
state0=[r0 v0];

Tmax = 2e-3;
swarm = 40;
iter = 1000;
prec = [5;5;5;9];

if bb == 1

    fprintf(fid,'%s %i\r\n','r0 (km) =',r0vec(1));
    fprintf(fid,'%s %i\r\n','swarm =',swarm);

    fprintf(fid,'%s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t %s\t\r\n','TOF','Phi','Vf','fpa','TOF2','Phi2','Vf2','fpa2','DV','DV2','DVTOT','iter','iter2','iterTOT','time','time2','timeTOT');
    fprintf(fid,'%s\r\n','----------------------------------------------------------------------------');
end

if bb == 10
    endval = 1;
else
    endval = 20;
end

ae = aevec(bb);
be = bevec(bb);

[rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim);
```

```
for aa = 1:endval

    tstart = tic;

    [JGmin,Jpbest,gbest,x,k] = LT_RTM_PSO_TFIXED(3,[0 2*pi;Vmin Vmax
        ;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf1,vf1,tf1,
        ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);

    Cost1 = JGmin*DU/TU*1000

    tend = toc(tstart)

    tstart2 = tic;

    [~,~,~,~,~,~,~,~,~,~,~,rt_ijk,vt_ijk] = Single_LT_Maneuver(rf1,
        vf1,tf1,gbest(1),ae,be,gbest(2),gbest(3),DU,TU,MU2);

    [rf2,vf2,tf2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*
        tf1,0,latlim,longlim);

    [JGmin2,Jpbest2,gbest2,x2,k2] = LT_RTM_PSO_TFIXED(3,[0 2*pi;Vmin
         Vmax;-pi/2+0.000001 pi/2-0.000001],iter,swarm,prec,rf2,vf2,
        tf2,ae,be,DU,TU,MU,Rmax,Rmin,Tmax,m0);

    Cost2 = JGmin2*DU/TU*1000

    CostTOT = Cost1 + Cost2

    tend2 = toc(tstart2)
```

```

        fprintf(fid,'%i\t %i \t %10.5f\t %6.5f\t %7.6f\t %6.5f\t %10.5f\
            t %6.5f\t %7.6f\t %6.5f\t %7.6f\t %7.6f\t %7.6f\t %i\t %i\t
            %i\t %4.1f\t %4.1f\t %4.1f\r\n',...
             norm(r0),ae,tf1,gbest(1),gbest(2),gbest(3),tf2,gbest2(1),
                gbest2(2),gbest2(3),Cost1,Cost2,CostTOT,k,k2,k+k2,tend,
                tend2,tend+tend2);
        tend + tend2

        clear tstart JGmin Jpbest gbest x k Cost1 tend tstart2 rt_ijk
            vt_ijk rf2 vf2 tf2 JGmin2 Jpbest2 gbest2 x2 k2  Cost2
            CostTOT tend2
    end

end
```

E.3.2 Direct Collocation Algorithms

E.3.2.1 Triple Pass LTRTM Driver

```
for zz = 2:2
    clear guess setup limits output
    close all
    clc

    if zz == 1
        load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Triple Pass\Journal Data\data6800_LT_3RTMsort.mat')
        [ind0] = find(data6800_LT_3RTMsort(:,1) ~= 0);
        PSO_data = data6800_LT_3RTMsort(ind0,:);
        cmax = 67;
        cmin = 67;
        rmag = 6800;
    elseif zz == 2
```

```
        load('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
           Triple Pass\Journal Data\data7300_LT_3RTMsort.mat')
        [ind0] = find(data7300_LT_3RTMsort(:,1) ~= 0);
        PSO_data = data7300_LT_3RTMsort(ind0,:);
        cmax = 27;
        cmin = 22;
        rmag = 7300;
    end

    for cc = cmin:cmax
        fid = fopen('PSO_to_GPOPS_3RTM4.txt','a');
        clear guess setup limits output
        close all
        clc
        t0 = 0;
        GMST0 = 0;
        latlim = [-10 10]*pi/180;
        longlim = [-50 -10]*pi/180;

        wgs84data
        global MU2 MU
        OmegaEarth = 0.000072921151467;
        r0vec = [rmag;0;0];
        v0vec = sqrt(MU/norm(r0vec))*[0;1/sqrt(2);1/sqrt(2)];

        [a,ecc,inc,RAAN,w,nu0] = RV2COE(r0vec,v0vec);
        period = 2*pi*sqrt(a^3/MU);

        swarm = 30;
        iter = 1000;
        Rmaxvec = rmag + 50;
        Rminvec = rmag - 50;
```

```
r0 = r0vec;
v0 = v0vec;
Rmax = Rmaxvec;
Rmin = Rminvec;

ae = PSO_data(cc,2);
be = ae/10;

TOF = PSO_data(cc,3);
phi = PSO_data(cc,4);
Vt_mag = PSO_data(cc,5);
fpa_t = PSO_data(cc,6);

tstart = tic;

[rf1,vf1,tf1,lat_enter,long_enter,R_exit,V_exit,t_exit,lat_exit,
    long_exit] = zone_entry_exit2(r0,v0,GMST0,t0,latlim,longlim)
    ;

lat_exp_enter = lat_enter;
long_exp_enter = long_enter;

DU = norm(rf1);
TU = period/(2*pi);

ae1 = ae/DU;
be1 = be/DU;
r01 = norm(r0)/DU;
MU2 = MU*TU^2/DU^3;
t0min = 0; % minimum initial time
t0max = 0; % maximum initial time
```

```
        tfmin = tf1; % minimum final time
        tfmax = tf1;
        n0 = sqrt(MU2/(norm(r0)/DU)^3);

        %% First Maneuver
        [LT_DV,maxT,r,gamma,T_a,thetaf_int,theta_dot,theta_ddot,rdot,
            Tvec,TOF_calc,rt_ijk,vt_ijk,rt_pqw,vt_pqw,r0_pqw,v0_pqw] =
            Single_LT_Maneuver(rf1,vf1,tf1,phi,ae,be,Vt_mag,fpa_t,DU,TU,
            MU2);

        time_mod = Tvec;

        [rf_pqw,vf_pqw] = IJK_to_PQW(rf1,vf1,inc,RAAN,w);
        rf_pqw = rf_pqw/DU;
        vf_pqw = vf_pqw/DU*TU;



        vunit = vf_pqw/norm(vf_pqw);
        hfp = cross(rf_pqw,vf_pqw);
        hunit = hfp/norm(hfp);

        gunit = cross(vunit,hunit);

        ang = (0:0.001:2*pi);
        re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);



        theta_rf = atan2(rf_pqw(2),rf_pqw(1));
        if theta_rf < 0
            theta_rf = 2*pi + theta_rf;
        end
        [rtest] = IJK_to_PQW(r0,v0,inc,RAAN,w);
```

```
        theta0 = atan2(rtest(2),rtest(1));

        theta_mod = thetaf_int + atan2(r0_pqw(2),r0_pqw(1));

        time_guess = time_mod;
        theta_guess = theta_mod;
        r_guess = r;
        vr_guess = rdot;
        vtheta_guess = r.*theta_dot;
        T_guess = T_a;
        B_guess = gamma;

        ind = find(T_guess ~= 0);

        %inertial position vector of new arrival position
        for aa = 1:length(ang)
            r_ell(:,aa) = rf_pqw + re(aa)*cos(ang(aa))*vunit + re(aa)*
                sin(ang(aa))*gunit;
        end

        rgi = zeros(length(r_guess),3);
        for dd = 1:length(r_guess)
            rg_pqw = DU*[r_guess(dd)*cos(theta_guess(dd));r_guess(dd)*
                sin(theta_guess(dd));0];
            [rgi(dd,:)] = PQW_to_IJK(rg_pqw,[],inc,RAAN,w);
        end

        for ee = 1:length(ang)
            rell_pqw = [r_ell(1,ee)*DU;r_ell(2,ee)*DU;0];
            rnom_pqw = norm(r0)*[cos(ang(ee));sin(ang(ee));0];
            [rell_ijk(:,ee)] = PQW_to_IJK(rell_pqw,[],inc,RAAN,w);
            [rnom_ijk(ee,:)] = PQW_to_IJK(rnom_pqw,[],inc,RAAN,w);
```

```
        end


        %% determine limits on subsequent passes into exclusion zone
        % assume upper limit based on circular orbit with phi = pi/2
        % assume lower limit based on circular orbit with phi = 2pi/2
        phi_low = 3*pi/2;
        phi_upp = pi/2;

        [rf_upp,vf_upp] = Single_LT_Limits(rf1,vf1,phi_upp,ae,be,DU,TU);

        [rf2_upp,vf2_upp,tf2_upp] = zone_entry_exit2(rf_upp,vf_upp,GMST0
            +OmegaEarth*tf1,0,latlim,longlim);

        ang_upp = sqrt(MU/norm(rf_upp)^3)*tf2_upp;

        thetaf2_max = theta_guess(end) + ang_upp;
        tf2_max = (tf1 + tf2_upp)/TU;



        [rf_low,vf_low] = Single_LT_Limits(rf1,vf1,phi_low,ae,be,DU,TU);

        [rf2_low,vf2_low,tf2_low] = zone_entry_exit2(rf_low,vf_low,GMST0
            +OmegaEarth*tf1,0,latlim,longlim);

        ang_low = sqrt(MU/norm(rf_low)^3)*tf2_low;

        thetaf2_min = theta_guess(end) + ang_low;
        tf2_min = (tf1 + tf2_low)/TU;
```

```
%% Second Maneuver
[rf2,vf2,tf2,lat_enter2,long_enter2,R_exit2,V_exit2,t_exit2,
    lat_exit2,long_exit2] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0
    +OmegaEarth*tf1,0,latlim,longlim);


TOF2 = PSO_data(cc,7);
phi2 = PSO_data(cc,8);
Vt_mag2 = PSO_data(cc,9);
fpa_t2 = PSO_data(cc,10);

[LT_DV2,maxT2,r2,gamma2,T_a2,thetaf_int2,theta_dot2,theta_ddot2,
    rdot2,Tvec2,TOF_calc2,rt_ijk2,vt_ijk2] = Single_LT_Maneuver(
    rf2,vf2,tf2,phi2,ae,be,Vt_mag2,fpa_t2,DU,TU,MU2);

theta02 = theta_guess(end);

delt2 = (tf2 - TOF2)/TU;
time_mod2 = Tvec2 + time_guess(end) + delt2;

[rf_pqw2,vf_pqw2] = IJK_to_PQW(rf2,vf2,inc,RAAN,w);
rf_pqw2 = rf_pqw2/DU;
vf_pqw2 = vf_pqw2/DU*TU;

vunit2 = vf_pqw2/norm(vf_pqw2);
hfp2 = cross(rf_pqw2,vf_pqw2);
hunit2 = hfp2/norm(hfp2);

gunit2 = cross(vunit2,hunit2);

[rt_pqw2,vt_pqw2] = IJK_to_PQW(rt_ijk2,vt_ijk2,inc,RAAN,w);
```

```
        ang_mod2 = atan2(rt_pqw2(2),rt_pqw2(1));
        while ang_mod2 < theta_guess(end)
            ang_mod2 = ang_mod2 + 2*pi;
        end
        ang_mod1 = atan2(rt_pqw(2),rt_pqw(1));

        diff = ang_mod2 - ang_mod1;
        diff2 = thetaf_int2(end) - thetaf_int2(1);
        theta_diff = diff - diff2;


        %inertial position vector of new arrival position
        for bb = 1:length(ang)
            r_ell2(:,bb) = rf_pqw2 + re(bb)*cos(ang(bb))*vunit2 + re(bb)
                *sin(ang(bb))*gunit2;
        end

        theta_mod2 = thetaf_int2 + theta_diff + theta_guess(end);

        coast_length = 1;

        time_guess2 = zeros(coast_length+length(time_mod2),1);
        time_guess2(1:coast_length) = time_guess(end);
        time_guess2(coast_length+1:end) = time_mod2;
        theta_guess2 = zeros(coast_length+length(theta_mod2),1);
        theta_guess2(coast_length+1:end) = theta_mod2;
        theta_guess2(1:coast_length) = theta02;
        r_guess2 = zeros(coast_length+length(theta_mod2),1);
        r_guess2(1:coast_length) = r_guess(end);
        r_guess2(coast_length+1:end) = r2;
        vr_guess2 = zeros(coast_length+length(theta_mod2),1);
        vr_guess2(1:coast_length) = vr_guess(end);
```

```
        vr_guess2(coast_length+1:end) = rdot2;
        vtheta_guess2 = zeros(coast_length+length(theta_mod2),1);
        vtheta_guess2(1:coast_length) = vtheta_guess(end);
        vtheta_guess2(coast_length+1:end) = r2.*theta_dot2;
        T_guess2 = zeros(coast_length+length(time_mod2),1);
        T_guess2(1:coast_length) = 0;
        T_guess2(coast_length+1:end) = T_a2;
        B_guess2 = zeros(coast_length+length(time_mod2),1);
        B_guess2(1:coast_length) = B_guess(end);
        B_guess2(coast_length+1:end) = gamma2;

        ind2 = find(T_guess2 ~= 0);

        nom_orb2_time = [(0:1:tf2) tf2];

        [at,et,it,Ot,ot,nut]= RV2COE(rt_ijk,vt_ijk);

        for ee = 1:length(nom_orb2_time)
            [nutf] = nuf_from_TOF(nut,nom_orb2_time(ee),at,et);
            [Rdum(:,ee),Vdum] = COE2RV(at,et,it,Ot,ot,nutf);
            [nom_orb2_R] = IJK_to_PQW(Rdum(:,ee),Vdum,inc,RAAN,w);

            ROrb2_PQW(ee,:) = nom_orb2_R;

        end
        while nutf < nut
            nutf = nutf + 2*pi;
        end

        %Angle of expected 2nd pass entry location into exclusion zone
        thetaf2 = (nutf - nut) + 0;

```

```
%% Coasting phase
%modify angle to match scenario angle
%1) determine coes of post maneuver 2 orbit at t = time_guess2(
    end)
[at2,et2,it2,Ot2,ot2,nut2]= RV2COE(rt_ijk2,vt_ijk2);

tcoast3 = [(time_guess2(end)+.001:.1:time_guess2(end)+(t_exit2-
    tf2)/TU)';time_guess2(end)+(t_exit2-tf2)/TU];
coast_length = length(tcoast3);
time_guess3 = tcoast3;
r_guess3 = zeros(coast_length,1);
theta_guess3 = zeros(coast_length,1);
vr_guess3 = zeros(coast_length,1);
vtheta_guess3 = zeros(coast_length,1);
T_guess3 = zeros(coast_length,1);
Beta_guess3 = zeros(coast_length,1);

for yy = 1:coast_length
    if yy == 1
        tprev = time_guess2(end);
        angprev = theta_guess2(end);
        nu_prev = nut2;
    end
    %2) determine length of time step in seconds
    tstep = (tcoast3(yy)-tprev)*TU;
    %3) current time becomes previous time
    tprev = tcoast3(yy);
    %4) determine angle traveled during tstep
    angnew = nuf_from_TOF(nu_prev,tstep,at2,et2);

    if angnew < nu_prev
        angtemp = angnew+2*pi;
```

```
        else
            angtemp = angnew;
        end
        ang_diff = angtemp - nu_prev;
        theta_guess3(yy) = angprev+ang_diff;
        angprev = theta_guess3(yy);
        nu_prev = angnew;
        %5) determine position and velocity in IJK
        [r3ijk,v3ijk]=COE2RV(at2,et2,it2,Ot2,ot2,angnew);
        %6) convert position and velocity to perifocal frame of
           initial
        %orbit
        [r3pqw,v3pqw] = IJK_to_PQW(r3ijk,v3ijk,inc,RAAN,w);

        r_guess3(yy) = norm(r3pqw)/DU;
        % 7) Vr and Vtheta

        vr_guess3(yy) = (MU/at2*(1-et2^2))*et2*sin(angnew)/DU*TU;
        vtheta_guess3(yy) = sqrt(MU/at2*(1-et2^2))*(1+et2*cos(angnew
           ))/DU*TU;
        T_guess(yy) = 0;
        Beta_guess(yy) = 0;

        if yy == coast_length
            tend3 = tcoast3(yy)*TU;
            rend3 = r3ijk;
            vend3 = v3ijk;
            theta3end = theta_guess3(yy);
        end

    end

```

```
%% Third Maneuver
[rf4,vf4,tf4,lat_enter3,long_enter3,R_exit3,V_exit3,t_exit3,
    lat_exit3,long_exit3] = zone_entry_exit2(rend3,vend3,GMST0+
    OmegaEarth*(tf1+t_exit2),0,latlim,longlim);
if tf4 < period+500
    TOF3 = PSO_data(cc,11);
    phi3 = PSO_data(cc,12);
    Vt_mag3 = PSO_data(cc,13);
    fpa_t3 = PSO_data(cc,14);

    [LT_DV3,maxT3,r3,gamma3,T_a3,thetaf_int3,theta_dot3,
        theta_ddot3,rdot3,Tvec3,TOF_calc3,rt_ijk3,vt_ijk3] =
        Single_LT_Maneuver(rf4,vf4,tf4,phi3,ae,be,Vt_mag3,fpa_t3
        ,DU,TU,MU2);

    theta04 = theta_guess3(end);

    delt4 = (tf4 - TOF3)/TU;
    time_mod4 = Tvec3 + time_guess3(end) + delt4;

    [rf_pqw4,vf_pqw4] = IJK_to_PQW(rf4,vf4,inc,RAAN,w);
    rf_pqw4 = rf_pqw4/DU;
    vf_pqw4 = vf_pqw4/DU*TU;

    vunit4 = vf_pqw4/norm(vf_pqw4);
    hfp4 = cross(rf_pqw4,vf_pqw4);
    hunit4 = hfp4/norm(hfp4);

    gunit4 = cross(vunit4,hunit4);

    [rt_pqw4,vt_pqw4] = IJK_to_PQW(rt_ijk3,vt_ijk3,inc,RAAN,w);
    ang_mod4 = atan2(rt_pqw4(2),rt_pqw4(1));
```

```
while ang_mod4 < 0
    ang_mod4 = ang_mod4 + 2*pi;
end


diff31 = ang_mod4 - ang_mod2;
diff32 = thetaf_int3(end) - thetaf_int3(1);
theta_diff3 = diff31 - diff32;

%inertial position vector of new arrival position
for zz = 1:length(ang)
    r_ell4(:,zz) = rf_pqw4 + re(zz)*cos(ang(zz))*vunit4 + re(zz)*sin(ang(zz))*gunit4;
end

theta_mod4 = thetaf_int3 + theta_diff3 + theta_guess2(end);
while theta_mod4(1) < theta_guess3(end)
    theta_mod4 = theta_mod4 + 2*pi;
end

if theta_mod4(2) - theta04 > 0.1
    theta_mod4 = theta_mod4 - 2*pi;
end

coast_length = 1;

time_guess4 = zeros(coast_length+length(time_mod4),1);
time_guess4(1:coast_length) = time_guess3(end);
time_guess4(coast_length+1:end) = time_mod4;
theta_guess4 = zeros(coast_length+length(theta_mod4),1);
theta_guess4(coast_length+1:end) = theta_mod4;
theta_guess4(1:coast_length) = theta04;
```

```
            r_guess4 = zeros(coast_length+length(theta_mod4),1);
            r_guess4(1:coast_length) = r_guess3(end);
            r_guess4(coast_length+1:end) = r3;
            vr_guess4 = zeros(coast_length+length(theta_mod4),1);
            vr_guess4(1:coast_length) = vr_guess3(end);
            vr_guess4(coast_length+1:end) = rdot3;
            vtheta_guess4 = zeros(coast_length+length(theta_mod4),1);
            vtheta_guess4(1:coast_length) = vtheta_guess3(end);
            vtheta_guess4(coast_length+1:end) = r3.*theta_dot3;
            T_guess4 = zeros(coast_length+length(time_mod4),1);
            T_guess4(1:coast_length) = 0;
            T_guess4(coast_length+1:end) = T_a3;
            B_guess4 = zeros(coast_length+length(time_mod4),1);
            B_guess4(1:coast_length) = 0;
            B_guess4(coast_length+1:end) = gamma3;

            nom_orb3_time = [(0:1:tf4) tf4];

            [at2,et2,it2,Ot2,ot2,nut2]= RV2COE(rt_ijk2,vt_ijk2);

            for yy = 1:length(nom_orb3_time)
                [nutf2] = nuf_from_TOF(nut2,nom_orb3_time(yy),at2,et2);
                [Rdum2(:,yy),Vdum2] = COE2RV(at2,et2,it2,Ot2,ot2,nutf2);
                [nom_orb3_R] = IJK_to_PQW(Rdum2(:,yy),Vdum2,inc,RAAN,w);

                ROrb3_PQW(ee,:) = nom_orb3_R;

            end
        else
            TOF3 = period;
            phi3 = PSO_data(cc,12);
            Vt_mag3 = PSO_data(cc,13);
```

```
fpa_t3 = PSO_data(cc,14);

[LT_DV3,maxT3,r3,gamma3,T_a3,thetaf_int3,theta_dot3,
    theta_ddot3,rdot3,Tvec3,TOF_calc3,rt_ijk3,vt_ijk3] =
    Single_LT_Maneuver(rf4,vf4,TOF3,phi3,ae,be,Vt_mag3,
    fpa_t3,DU,TU,MU2);



tcoast4 = (time_guess3(end)+.001:.1:time_guess3(end)+(tf4-
    period)/TU);
time_mod4 = time_guess3(end)+(tf4-period)/TU + Tvec3;
coast_length4 = length(tcoast4);

%modify time to match scenario time
time_guess4 = zeros(coast_length4+length(time_mod4),1);
time_guess4(1:coast_length4) = tcoast4;
time_guess4(coast_length4+1:end) = time_mod4;

r_guess4 = zeros(length(time_guess4),1);
theta_guess4 = zeros(length(time_guess4),1);
vr_guess4 = zeros(length(time_guess4),1);
vtheta_guess4 = zeros(length(time_guess4),1);
T_guess4 = zeros(length(time_guess4),1);
Beta_guess4 = zeros(length(time_guess4),1);

%modify angle to match scenario angle
%1) determine coes of post maneuver 2 orbit at t =
    time_guess2(end)
[at2,et2,it2,Ot2,ot2,nut2]= RV2COE(rend3,vend3);

for yy = 1:coast_length4
```

```
if yy == 1
    tprev = time_guess3(end);
    angprev = theta_guess3(end);
    nu_prev = nut2;
end
%2) determine length of time step in seconds
tstep = (tcoast4(yy)-tprev)*TU;
%3) current time becomes previous time
tprev = tcoast4(yy);
%4) determine angle traveled during tstep
angnew = nuf_from_TOF(nu_prev,tstep,at2,et2);

if angnew < nu_prev
    angtemp = angnew+2*pi;
else
    angtemp = angnew;
end
ang_diff = angtemp - nu_prev;
theta_guess4(yy) = angprev+ang_diff;
angprev = theta_guess4(yy);
nu_prev = angnew;
%5) determine position and velocity in IJK
[r3ijk,v3ijk]=COE2RV(at2,et2,it2,Ot2,ot2,angnew);
%6) convert position and velocity to perifocal frame of
   initial
%orbit
[r3pqw,v3pqw] = IJK_to_PQW(r3ijk,v3ijk,inc,RAAN,w);

r_guess4(yy) = norm(r3pqw)/DU;
% 7) Vr and Vtheta

```

```
            vr_guess4(yy) = (MU/at2*(1-et2^2))*et2*sin(angnew)/DU*TU
                ;
            vtheta_guess4(yy) = sqrt(MU/at2*(1-et2^2))*(1+et2*cos(
                angnew))/DU*TU;


        end

        [rf_pqw4,vf_pqw4] = IJK_to_PQW(rf4,vf4,inc,RAAN,w);
        rf_pqw4 = rf_pqw4/DU;
        vf_pqw4 = vf_pqw4/DU*TU;

        vunit4 = vf_pqw4/norm(vf_pqw4);
        hfp4 = cross(rf_pqw4,vf_pqw4);
        hunit4 = hfp4/norm(hfp4);

        gunit4 = cross(vunit4,hunit4);
        [rt_pqw4,vt_pqw4] = IJK_to_PQW(rt_ijk3,vt_ijk3,inc,RAAN,w);



        ang_mod4 = atan2(rt_pqw4(2),rt_pqw4(1));
        while ang_mod4 < theta_guess4(yy)
            ang_mod4 = ang_mod4 + 2*pi;
        end

        diff31 = ang_mod4 - theta_guess4(yy);
        diff32 = thetaf_int3(end) - thetaf_int3(1);
        theta_diff3 = diff31 - diff32;




        %inertial position vector of new arrival position
```

```
            for zz = 1:length(ang)
                r_ell4(:,zz) = rf_pqw4 + re(zz)*cos(ang(zz))*vunit4 + re
                    (zz)*sin(ang(zz))*gunit4;
            end

            theta_mod4 = theta_guess4(yy) + thetaf_int3 + theta_diff3;

            while theta_mod4(1) < theta_guess4(yy)
                theta_mod4 = theta_mod4 + 2*pi;
            end


            time_guess4(coast_length4+1:end) = time_mod4;
            theta_guess4(coast_length4+1:end) = theta_mod4;
            r_guess4(coast_length4+1:end) = r3;
            vr_guess4(coast_length4+1:end) = rdot3;
            vtheta_guess4(coast_length4+1:end) = r3.*theta_dot3;
            T_guess4(1:coast_length4) = 0;
            T_guess4(coast_length4+1:end) = T_a3;
            Beta_guess4(1:coast_length4) = 0;
            Beta_guess4(coast_length4+1:end) = gamma3;

        end



        %% GPOPS RUN (1st Run Through assigns a non-zero minimum thrust
            to help GPOPS-II converge)
        % variables from PSo phase
        r1 = 1;
        rf = norm(rt_pqw);
        rmax = r1 + be/DU;
```

```
        rmin = r1 - be/DU;
        thetaf_min = theta_rf - atan(ae/norm(r0));
        thetaf_max = theta_rf + atan(ae/norm(r0));

        % Control and time boundaries
        umin =  -0.5; % minimum control angle
        umax =  2*pi+0.5; % maximum control angle
        Tmax = 2*0.0001160;
        Tmin = Tmax/1000;
        % ------------------------------------------------------------------------

        % GPOPS Setup
        % Phase 1 Information
        iphase = 1;
        bounds.phase(iphase).initialtime.lower = t0min;
        bounds.phase(iphase).initialtime.upper = t0max;
        bounds.phase(iphase).finaltime.lower = tf1/TU;
        bounds.phase(iphase).finaltime.upper = tf1/TU;
        % LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
        bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU 0 sqrt(MU2/r1)];
        bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU 0 sqrt(MU2/r1)];
        bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.1 0];
        bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.1 1.1];
        bounds.phase(iphase).state.lower = [rmin theta_rf-n0*tf1/TU -0.1 0];
        bounds.phase(iphase).state.upper = [rmax thetaf_max 0.1 1.1];
        bounds.phase(iphase).control.lower = [Tmin umin];
```

```
        bounds.phase(iphase).control.upper = [Tmax umax];
        % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
        bounds.phase(iphase).path.lower = []; % None
        bounds.phase(iphase).path.upper = []; % None
        bounds.phase(iphase).integral.lower = 0;
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/DU]; % None
        bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/DU]; % None
        % GUESS SOLUTION
        guess.phase(iphase).time = time_guess;
        guess.phase(iphase).state(:,1) = r_guess;
        guess.phase(iphase).state(:,2) = theta_guess;
        guess.phase(iphase).state(:,3) = vr_guess;
        guess.phase(iphase).state(:,4) = vtheta_guess;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_guess;
        guess.phase(iphase).control(:,2) = B_guess;
        guess.phase(iphase).integral = LT_DV;
        % -----------------------------------------------------------------------

        % Phase 2 Information (second Maneuver
        iphase = 2;
        bounds.phase(iphase).initialtime.lower = tf1/TU;
        bounds.phase(iphase).initialtime.upper = tf1/TU;
        bounds.phase(iphase).finaltime.lower = tf2_min-1;
        bounds.phase(iphase).finaltime.upper = tf2_max+1;
        % LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
        bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.1 0];
```

```
        bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.1
            1.1];
        bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.1
             0];
        bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.1
            1.1];
        bounds.phase(iphase).state.lower = [rmin thetaf_min -0.1 0];
        bounds.phase(iphase).state.upper = [rmax thetaf2_max+1 0.1 1.1];
        bounds.phase(iphase).control.lower = [Tmin umin];
        bounds.phase(iphase).control.upper = [Tmax umax];
        % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
        bounds.phase(iphase).path.lower = []; % None
        bounds.phase(iphase).path.upper = []; % None
        bounds.phase(iphase).integral.lower = 0;
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 0 Rmin/DU Rmin
            /DU]; % None
        bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 0 Rmax/DU Rmax
            /DU]; % None
        % GUESS SOLUTION
        guess.phase(iphase).time = time_guess2;
        guess.phase(iphase).state(:,1) = r_guess2;
        guess.phase(iphase).state(:,2) = theta_guess2;
        guess.phase(iphase).state(:,3) = vr_guess2;
        guess.phase(iphase).state(:,4) = vtheta_guess2;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_guess2;
        guess.phase(iphase).control(:,2) = B_guess2;
        guess.phase(iphase).integral = LT_DV2;
        %
            ----------------------------------------------------------------------------
```

```
% Phase 3 (Coast)
iphase = 3;
bounds.phase(iphase).initialtime.lower = tf2_min-1;
bounds.phase(iphase).initialtime.upper = tf2_max+1;
bounds.phase(iphase).finaltime.lower = tcoast3(end)-1;
bounds.phase(iphase).finaltime.upper = tcoast3(end)+1;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [rmin thetaf2_min-1
    -0.1 0];
bounds.phase(iphase).initialstate.upper = [rmax thetaf2_max+1
    0.1 1.1];
bounds.phase(iphase).finalstate.lower = [rmin theta3end-1 -0.1
    0];
bounds.phase(iphase).finalstate.upper = [rmax theta3end+1 0.1
    1.1];
bounds.phase(iphase).state.lower = [rmin thetaf2_min-1 -0.1 0];
bounds.phase(iphase).state.upper = [rmax theta3end+1 0.1 1.1];
bounds.phase(iphase).control.lower = [0 0];
bounds.phase(iphase).control.upper = [0 0];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1;
bounds.eventgroup(iphase).lower = [zeros(1,5) 0]; % None
bounds.eventgroup(iphase).upper = [zeros(1,5) 0]; % None
% GUESS SOLUTION
guess.phase(iphase).time = time_guess3;
guess.phase(iphase).state(:,1) = r_guess3;
guess.phase(iphase).state(:,2) = theta_guess3;
guess.phase(iphase).state(:,3) = vr_guess3;
guess.phase(iphase).state(:,4) = vtheta_guess3;
```

```
% Control guess :
guess.phase(iphase).control(:,1) = T_guess3;
guess.phase(iphase).control(:,2) = Beta_guess3;
guess.phase(iphase).integral = 0;

% Phase 4 Information (third Maneuver)
iphase = 4;
bounds.phase(iphase).initialtime.lower = tcoast3(end)-1;
bounds.phase(iphase).initialtime.upper = tcoast3(end)+1;
bounds.phase(iphase).finaltime.lower = (tf1+tf2+tf4)/TU-1;
bounds.phase(iphase).finaltime.upper = (tf1+tf2+tf4)/TU+1;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [rmin theta3end-1 -0.1 0];
bounds.phase(iphase).initialstate.upper = [rmax theta3end+1 0.1 1.1];
bounds.phase(iphase).finalstate.lower = [rmin theta_guess4(end)-1 -0.1 0];
bounds.phase(iphase).finalstate.upper = [rmax theta_guess4(end)+1 0.1 1.1];
bounds.phase(iphase).state.lower = [rmin theta3end-1 -0.1 0];
bounds.phase(iphase).state.upper = [rmax theta_guess4(end)+1 0.1 1.1];
bounds.phase(iphase).control.lower = [Tmin umin];
bounds.phase(iphase).control.upper = [Tmax umax];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.parameter.lower = [0 0 0];
bounds.parameter.upper = [2*pi 2*pi 2*pi];
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1;
```

```
        bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; %
            None
        bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; %
            None
        % GUESS SOLUTION
        guess.phase(iphase).time = time_guess4;
        guess.phase(iphase).state(:,1) = r_guess4;
        guess.phase(iphase).state(:,2) = theta_guess4;
        guess.phase(iphase).state(:,3) = vr_guess4;
        guess.phase(iphase).state(:,4) = vtheta_guess4;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_guess4;
        guess.phase(iphase).control(:,2) = Beta_guess4;
        guess.parameter = [phi phi2 phi3];
        guess.phase(iphase).integral = LT_DV3;
        %
            --------------------------------------------------------------------------

        %auxiliary data
        auxdata.MU = MU2;
        auxdata.ae = ae1;
        auxdata.be = be1;
        auxdata.rf_pqw = rf_pqw;
        auxdata.vunit = vunit;
        auxdata.gunit = gunit;
        auxdata.inc = inc;
        auxdata.RAAN = RAAN;
        auxdata.w = w;
        auxdata.latlim = latlim;
        auxdata.longlim = longlim;
        auxdata.GMST0 = GMST0;
        auxdata.OmegaEarth = OmegaEarth;
```

```
        auxdata.DU = DU;
        auxdata.TU = TU;

        % NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
            required
        setup.name  = 'TIME_FIXED_INTERCEPT';


        colp = 4;
        colnum = 20;
        colp2 = 4;
        colnum2 = 80;

        setup.functions.continuous = @LT_3RTM_Continuous_4Phase;
        setup.functions.endpoint = @LT_3RTM_Endpoint_4Phase;
        setup.nlp.solver = 'ipopt';
        setup.mesh.maxiteration = 10;
        setup.mesh.tolerance = 1e-10;
        setup.mesh.colpointsmin = 40;
        setup.mesh.colpointsmax = 1000;
        for i = 1:4
            if i < 4
                setup.mesh.phase(i).colpoints = colp*ones(1,colnum);
                setup.mesh.phase(i).fraction = 1/colnum*ones(1,colnum);
            elseif i == 3
                setup.mesh.phase(i).colpoints = 1*ones(1,5);
                setup.mesh.phase(i).fraction = 1/5*ones(1,5);
            else
                setup.mesh.phase(i).colpoints = colp2*ones(1,colnum2);
                setup.mesh.phase(i).fraction = 1/colnum2*ones(1,colnum2)
                    ;
            end
```

```
        end
        setup.bounds = bounds;
        setup.guess = guess;
        setup.auxdata = auxdata;
        setup.mesh.method = 'RPMintegration';
        setup.derivatives.supplier = 'sparseFD';
        setup.derivativelevel ='second';
        setup.dependencies = 'sparseNaN';
        setup.scales = 'none';

        output = gpops2(setup);
        solution = output.result.solution;
        %%
        %States and costates from phase 1 (first maneuver)
        r_GPOPS_P1 = solution.phase(1).state(:,1);
        theta_GPOPS_P1 = solution.phase(1).state(:,2);
        Vr_GPOPS_P1 = solution.phase(1).state(:,3);
        Vt_GPOPS_P1 = solution.phase(1).state(:,4);
        lambda_r_P1 = solution.phase(1).costate(:,1);
        lambda_theta_P1 = solution.phase(1).costate(:,2);
        lambda_Vr_P1 = solution.phase(1).costate(:,3);
        lambda_Vt_P1 = solution.phase(1).costate(:,4);
        tvec_P1 = solution.phase(1).time;

        thetadot_GPOPS_P1 = Vt_GPOPS_P1./r_GPOPS_P1;
        T_GPOPS_P1 = solution.phase(1).control(:,1);
        Beta_GPOPS_P1 = solution.phase(1).control(:,2);
        phi_GPOPS_P1 = solution.parameter(1);
        re_GPOPS_P1 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P1))^2 + (ae1*sin(
            phi_GPOPS_P1))^2);

```

```
        switch_func1 = lambda_Vr_P1.*sin(Beta_GPOPS_P1) + lambda_Vt_P1.*
            cos(Beta_GPOPS_P1) + 1;

        %
            -----------------------------------------------------------------------

        %States and Costates from phase 2 (second maneuver)
        r_GPOPS_P2 = solution.phase(2).state(:,1);
        theta_GPOPS_P2 = solution.phase(2).state(:,2);
        Vr_GPOPS_P2 = solution.phase(2).state(:,3);
        Vt_GPOPS_P2 = solution.phase(2).state(:,4);
        lambda_r_P2 = solution.phase(2).costate(:,1);
        lambda_theta_P2 = solution.phase(2).costate(:,2);
        lambda_Vr_P2 = solution.phase(2).costate(:,3);
        lambda_Vt_P2 = solution.phase(2).costate(:,4);
        tvec_P2 = solution.phase(2).time;

        thetadot_GPOPS_P2 = Vt_GPOPS_P2./r_GPOPS_P2;
        T_GPOPS_P2 = solution.phase(2).control(:,1);
        Beta_GPOPS_P2 = solution.phase(2).control(:,2);
        phi_GPOPS_P2 = solution.parameter(2);
        re_GPOPS_P2 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P2))^2 + (ae1*sin(
            phi_GPOPS_P2))^2);

        switch_func2 = lambda_Vr_P2.*sin(Beta_GPOPS_P2) + lambda_Vt_P2.*
            cos(Beta_GPOPS_P2) + 1;
        %
            -----------------------------------------------------------------------

        %States and Costates from phase 3 (third maneuver)
```

```
        r_GPOPS_P3 = solution.phase(3).state(:,1);
        theta_GPOPS_P3 = solution.phase(3).state(:,2);
        Vr_GPOPS_P3 = solution.phase(3).state(:,3);
        Vt_GPOPS_P3 = solution.phase(3).state(:,4);
        lambda_r_P3 = solution.phase(3).costate(:,1);
        lambda_theta_P3 = solution.phase(3).costate(:,2);
        lambda_Vr_P3 = solution.phase(3).costate(:,3);
        lambda_Vt_P3 = solution.phase(3).costate(:,4);
        tvec_P3 = solution.phase(3).time;

        thetadot_GPOPS_P3 = Vt_GPOPS_P3./r_GPOPS_P3;
        T_GPOPS_P3 = solution.phase(3).control(:,1);
        Beta_GPOPS_P3 = solution.phase(3).control(:,2);
        phi_GPOPS_P3 = solution.parameter(3);
        re_GPOPS_P3 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P3))^2 + (ae1*sin(
            phi_GPOPS_P3))^2);

        switch_func3 = lambda_Vr_P3.*sin(Beta_GPOPS_P3) + lambda_Vt_P3.*
            cos(Beta_GPOPS_P3) + 1;

        %States and Costates from phase 3 (third maneuver)
        r_GPOPS_P4 = solution.phase(4).state(:,1);
        theta_GPOPS_P4 = solution.phase(4).state(:,2);
        Vr_GPOPS_P4 = solution.phase(4).state(:,3);
        Vt_GPOPS_P4 = solution.phase(4).state(:,4);
        lambda_r_P4 = solution.phase(4).costate(:,1);
        lambda_theta_P4 = solution.phase(4).costate(:,2);
        lambda_Vr_P4 = solution.phase(4).costate(:,3);
        lambda_Vt_P4 = solution.phase(4).costate(:,4);
        tvec_P4 = solution.phase(4).time;

        thetadot_GPOPS_P4 = Vt_GPOPS_P4./r_GPOPS_P4;
```

```
        T_GPOPS_P4 = solution.phase(4).control(:,1);
        Beta_GPOPS_P4 = solution.phase(4).control(:,2);
        re_GPOPS_P4 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P3))^2 + (ae1*sin(
            phi_GPOPS_P3))^2);

        switch_func4 = lambda_Vr_P4.*sin(Beta_GPOPS_P4) + lambda_Vt_P4.*
            cos(Beta_GPOPS_P4) + 1;

        %
            ---------------------------------------------------------------------------

        Cost = (solution.phase(1).integral + solution.phase(2).integral
            + solution.phase(3).integral + solution.phase(4).integral)*
            DU/TU*1000
        %% GPOPS Run two (Minimum thrust is set to zero in run 2 to
            generate true optimal solution
        clear guess setup bound limits

        % variables from PSo phase
        r1 = 1;
        rmax = r1 + be/DU;
        rmin = r1 - be/DU;
        thetaf_min = theta_rf - atan(ae/norm(r0));
        thetaf_max = theta_rf + atan(ae/norm(r0));

        % Control and time boundaries
        umin =  -0.5; % minimum control angle
        umax =  2*pi+0.5; % maximum control angle
        Tmax = 2*0.0001160;
        %     Tmin = 0;

        % Phase 1 Information
```

```
iphase = 1;
bounds.phase(iphase).initialtime.lower = t0min;
bounds.phase(iphase).initialtime.upper = t0max;
bounds.phase(iphase).finaltime.lower = tf1/TU;
bounds.phase(iphase).finaltime.upper = tf1/TU;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).initialstate.upper = [r1 theta_rf-n0*tf1/TU
     0 sqrt(MU2/r1)];
bounds.phase(iphase).finalstate.lower = [rmin thetaf_min -0.1
    0];
bounds.phase(iphase).finalstate.upper = [rmax thetaf_max 0.1
    1.1];
bounds.phase(iphase).state.lower = [rmin theta_rf-n0*tf1/TU -0.1
     0];
bounds.phase(iphase).state.upper = [rmax thetaf_max 0.1 1.1];
bounds.phase(iphase).control.lower = [Tmin umin];
bounds.phase(iphase).control.upper = [Tmax umax];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1;
bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 Rmin/DU Rmin/
    DU]; % None
bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 Rmax/DU Rmax/
    DU]; % None
% GUESS SOLUTION
guess.phase(iphase).time = tvec_P1;
guess.phase(iphase).state(:,1) = r_GPOPS_P1;
guess.phase(iphase).state(:,2) = theta_GPOPS_P1;
```

```
guess.phase(iphase).state(:,3) = Vr_GPOPS_P1;
guess.phase(iphase).state(:,4) = Vt_GPOPS_P1;
% Control guess :
guess.phase(iphase).control(:,1) = T_GPOPS_P1;
guess.phase(iphase).control(:,2) = Beta_GPOPS_P1;
guess.phase(iphase).integral = solution.phase(1).integral;
%
    ----------------------------------------------------------------------------

% Phase 2 Information (second Maneuver
iphase = 2;
bounds.phase(iphase).initialtime.lower = tf1/TU;
bounds.phase(iphase).initialtime.upper = tf1/TU;
bounds.phase(iphase).finaltime.lower = tf2_min-1;
bounds.phase(iphase).finaltime.upper = tf2_max+1;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [rmin thetaf_min -0.1
    0];
bounds.phase(iphase).initialstate.upper = [rmax thetaf_max 0.1
    1.1];
bounds.phase(iphase).finalstate.lower = [rmin thetaf2_min-1 -0.1
     0];
bounds.phase(iphase).finalstate.upper = [rmax thetaf2_max+1 0.1
    1.1];
bounds.phase(iphase).state.lower = [rmin thetaf_min -0.1 0];
bounds.phase(iphase).state.upper = [rmax thetaf2_max+1 0.1 1.1];
bounds.phase(iphase).control.lower = [Tmin umin];
bounds.phase(iphase).control.upper = [Tmax umax];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
```

```
bounds.phase(iphase).integral.upper = 1;
bounds.eventgroup(iphase).lower = [zeros(1,5) 0 0 0 Rmin/DU Rmin/DU]; % None
bounds.eventgroup(iphase).upper = [zeros(1,5) 0 0 0 Rmax/DU Rmax/DU]; % None
% GUESS SOLUTION
guess.phase(iphase).time = tvec_P2;
guess.phase(iphase).state(:,1) = r_GPOPS_P2;
guess.phase(iphase).state(:,2) = theta_GPOPS_P2;
guess.phase(iphase).state(:,3) = Vr_GPOPS_P2;
guess.phase(iphase).state(:,4) = Vt_GPOPS_P2;
% Control guess :
guess.phase(iphase).control(:,1) = T_GPOPS_P2;
guess.phase(iphase).control(:,2) = Beta_GPOPS_P2;
guess.phase(iphase).integral = solution.phase(2).integral;
% -------------------------------------------------------------------------

% Phase 3 (Coast)
iphase = 3;
bounds.phase(iphase).initialtime.lower = tf2_min-1;
bounds.phase(iphase).initialtime.upper = tf2_max+1;
bounds.phase(iphase).finaltime.lower = tcoast3(end)-1;
bounds.phase(iphase).finaltime.upper = tcoast3(end)+1;
% LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
bounds.phase(iphase).initialstate.lower = [rmin thetaf2_min-1 -0.1 0];
bounds.phase(iphase).initialstate.upper = [rmax thetaf2_max+1 0.1 1.1];
bounds.phase(iphase).finalstate.lower = [rmin theta3end-1 -0.1 0];
```

```
        bounds.phase(iphase).finalstate.upper = [rmax theta3end+1 0.1
            1.1];
        bounds.phase(iphase).state.lower = [rmin thetaf2_min-1 -0.1 0];
        bounds.phase(iphase).state.upper = [rmax theta3end+1 0.1 1.1];
        bounds.phase(iphase).control.lower = [0 0];
        bounds.phase(iphase).control.upper = [0 0];
        % LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
        bounds.phase(iphase).path.lower = []; % None
        bounds.phase(iphase).path.upper = []; % None
        bounds.phase(iphase).integral.lower = 0;
        bounds.phase(iphase).integral.upper = 1;
        bounds.eventgroup(iphase).lower = [zeros(1,5) 0]; % None
        bounds.eventgroup(iphase).upper = [zeros(1,5) 0]; % None
        % GUESS SOLUTION
        guess.phase(iphase).time = tvec_P3;
        guess.phase(iphase).state(:,1) = r_GPOPS_P3;
        guess.phase(iphase).state(:,2) = theta_GPOPS_P3;
        guess.phase(iphase).state(:,3) = Vr_GPOPS_P3;
        guess.phase(iphase).state(:,4) = Vt_GPOPS_P3;
        % Control guess :
        guess.phase(iphase).control(:,1) = T_GPOPS_P3;
        guess.phase(iphase).control(:,2) = Beta_GPOPS_P3;
        guess.phase(iphase).integral = solution.phase(3).integral;

        % Phase 4 Information (third Maneuver)
        iphase = 4;
        bounds.phase(iphase).initialtime.lower = tcoast3(end)-1;
        bounds.phase(iphase).initialtime.upper = tcoast3(end)+1;
        bounds.phase(iphase).finaltime.lower = (tf1+tf2+tf4)/TU-1;
        bounds.phase(iphase).finaltime.upper = (tf1+tf2+tf4)/TU+1;
        % LIMITS ON STATE AND CONTROL VALUES THROUGHOUT TRAJECTORY
```

```
bounds.phase(iphase).initialstate.lower = [rmin theta3end-1 -0.1 0];
bounds.phase(iphase).initialstate.upper = [rmax theta3end+1 0.1 1.1];
bounds.phase(iphase).finalstate.lower = [rmin theta_guess4(end) -1 -0.1 0];
bounds.phase(iphase).finalstate.upper = [rmax theta_guess4(end) +1 0.1 1.1];
bounds.phase(iphase).state.lower = [rmin theta3end-1 -0.1 0];
bounds.phase(iphase).state.upper = [rmax theta_guess4(end)+1 0.1 1.1];
bounds.phase(iphase).control.lower = [Tmin umin];
bounds.phase(iphase).control.upper = [Tmax umax];
% LIMITS ON PARAMETERS, PATH, AND EVENT CONSTRAINTS
bounds.parameter.lower = [0 0 0];
bounds.parameter.upper = [2*pi 2*pi 2*pi];
bounds.phase(iphase).path.lower = []; % None
bounds.phase(iphase).path.upper = []; % None
bounds.phase(iphase).integral.lower = 0;
bounds.phase(iphase).integral.upper = 1;
bounds.eventgroup(iphase).lower = [0 0 0 Rmin/DU Rmin/DU]; % None
bounds.eventgroup(iphase).upper = [0 0 0 Rmax/DU Rmax/DU]; % None
% GUESS SOLUTION
guess.phase(iphase).time = tvec_P4;
guess.phase(iphase).state(:,1) = r_GPOPS_P4;
guess.phase(iphase).state(:,2) = theta_GPOPS_P4;
guess.phase(iphase).state(:,3) = Vr_GPOPS_P4;
guess.phase(iphase).state(:,4) = Vt_GPOPS_P4;
% Control guess :
guess.phase(iphase).control(:,1) = T_GPOPS_P4;
```

```
        guess.phase(iphase).control(:,2) = Beta_GPOPS_P4;
        guess.parameter = [phi_GPOPS_P1 phi_GPOPS_P2 phi_GPOPS_P3];
        guess.phase(iphase).integral = solution.phase(4).integral;

        %auxiliary data
        auxdata.MU = MU2;
        auxdata.ae = ae1;
        auxdata.be = be1;
        auxdata.rf_pqw = rf_pqw;
        auxdata.vunit = vunit;
        auxdata.gunit = gunit;
        auxdata.inc = inc;
        auxdata.RAAN = RAAN;
        auxdata.w = w;
        auxdata.latlim = latlim;
        auxdata.longlim = longlim;
        auxdata.GMST0 = GMST0;
        auxdata.OmegaEarth = OmegaEarth;
        auxdata.DU = DU;
        auxdata.TU = TU;

        % NOTE: Functions "phasingmaneuverCost" and "phasingmaneuverDae"
            required
        setup.name  = 'TIME_FIXED_INTERCEPT';


        colp = 4;
        colnum = 20;
        colp2 = 4;
        colnum2 = 80;

        setup.functions.continuous = @LT_3RTM_Continuous_4Phase;
```

```
setup.functions.endpoint = @LT_3RTM_Endpoint_4Phase;
setup.nlp.solver = 'ipopt';
setup.mesh.maxiteration = 10;
setup.mesh.tolerance = 1e-10;
setup.mesh.colpointsmin = 40;
setup.mesh.colpointsmax = 1000;
for i = 1:4
    if i < 4
        setup.mesh.phase(i).colpoints = colp*ones(1,colnum);
        setup.mesh.phase(i).fraction = 1/colnum*ones(1,colnum);
    elseif i == 3
        setup.mesh.phase(i).colpoints = 1*ones(1,5);
        setup.mesh.phase(i).fraction = 1/5*ones(1,5);
    else
        setup.mesh.phase(i).colpoints = colp2*ones(1,colnum2);
        setup.mesh.phase(i).fraction = 1/colnum2*ones(1,colnum2);
    end
end
setup.bounds = bounds;
setup.guess = guess;
setup.auxdata = auxdata;
setup.mesh.method = 'RPMintegration';
setup.derivatives.supplier = 'sparseFD';
setup.derivativelevel ='second';
setup.dependencies = 'sparseNaN';
setup.scales = 'none';

output = gpops2(setup);
solution2 = output.result.solution;

%%
```

```
%States and costates from phase 1 (first maneuver)
r_GPOPS_P12 = solution2.phase(1).state(:,1);
theta_GPOPS_P12 = solution2.phase(1).state(:,2);
Vr_GPOPS_P12 = solution2.phase(1).state(:,3);
Vt_GPOPS_P12 = solution2.phase(1).state(:,4);
lambda_r_P12 = solution2.phase(1).costate(:,1);
lambda_theta_P12 = solution2.phase(1).costate(:,2);
lambda_Vr_P12 = solution2.phase(1).costate(:,3);
lambda_Vt_P12 = solution2.phase(1).costate(:,4);
tvec_P12 = solution2.phase(1).time;

thetadot_GPOPS_P12 = Vt_GPOPS_P12./r_GPOPS_P12;
T_GPOPS_P12 = solution2.phase(1).control(:,1);
Beta_GPOPS_P12 = solution2.phase(1).control(:,2);
phi_GPOPS_P12 = solution2.parameter(1);
re_GPOPS_P12 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P12))^2 + (ae1*
    sin(phi_GPOPS_P12))^2);

%States and Costates from pahse 2 (second maneuver)
r_GPOPS_P22 = solution2.phase(2).state(:,1);
theta_GPOPS_P22 = solution2.phase(2).state(:,2);
Vr_GPOPS_P22 = solution2.phase(2).state(:,3);
Vt_GPOPS_P22 = solution2.phase(2).state(:,4);
lambda_r_P22 = solution2.phase(2).costate(:,1);
lambda_theta_P22 = solution2.phase(2).costate(:,2);
lambda_Vr_P22 = solution2.phase(2).costate(:,3);
lambda_Vt_P22 = solution2.phase(2).costate(:,4);
tvec_P22 = solution2.phase(2).time;

thetadot_GPOPS_P22 = Vt_GPOPS_P22./r_GPOPS_P22;
T_GPOPS_P22 = solution2.phase(2).control(:,1);
Beta_GPOPS_P22 = solution2.phase(2).control(:,2);
```

```
phi_GPOPS_P22 = solution2.parameter(2);
re_GPOPS_P22 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P22))^2 + (ae1*
    sin(phi_GPOPS_P22))^2);

%States and Costates from pahse 2 (second maneuver)
r_GPOPS_P32 = solution2.phase(3).state(:,1);
theta_GPOPS_P32 = solution2.phase(3).state(:,2);
Vr_GPOPS_P32 = solution2.phase(3).state(:,3);
Vt_GPOPS_P32 = solution2.phase(3).state(:,4);
lambda_r_P32 = solution2.phase(3).costate(:,1);
lambda_theta_P32 = solution2.phase(3).costate(:,2);
lambda_Vr_P32 = solution2.phase(3).costate(:,3);
lambda_Vt_P32 = solution2.phase(3).costate(:,4);
tvec_P32 = solution2.phase(3).time;

thetadot_GPOPS_P32 = Vt_GPOPS_P32./r_GPOPS_P32;
T_GPOPS_P32 = solution2.phase(3).control(:,1);
Beta_GPOPS_P32 = solution2.phase(3).control(:,2);
phi_GPOPS_P32 = solution2.parameter(3);
re_GPOPS_P32 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P32))^2 + (ae1*
    sin(phi_GPOPS_P32))^2);

%States and Costates from pahse 2 (second maneuver)
r_GPOPS_P42 = solution2.phase(4).state(:,1);
theta_GPOPS_P42 = solution2.phase(4).state(:,2);
Vr_GPOPS_P42 = solution2.phase(4).state(:,3);
Vt_GPOPS_P42 = solution2.phase(4).state(:,4);
lambda_r_P42 = solution2.phase(4).costate(:,1);
lambda_theta_P42 = solution2.phase(4).costate(:,2);
lambda_Vr_P42 = solution2.phase(4).costate(:,3);
lambda_Vt_P42 = solution2.phase(4).costate(:,4);
tvec_P42 = solution2.phase(4).time;
```

```
thetadot_GPOPS_P42 = Vt_GPOPS_P42./r_GPOPS_P42;
T_GPOPS_P42 = solution2.phase(4).control(:,1);
Beta_GPOPS_P42 = solution2.phase(4).control(:,2);
re_GPOPS_P42 = ae1*be1/sqrt((be1*cos(phi_GPOPS_P32))^2 + (ae1*
    sin(phi_GPOPS_P32))^2);

Cost2 = (solution2.phase(1).integral + solution2.phase(2).
    integral + solution2.phase(3).integral + solution2.phase(4).
    integral)*DU/TU*1000


%%
clear rgi rgi2

ang = (0:0.001:2*pi);
re = (ae1*be1)./sqrt((be1*cos(ang)).^2 + (ae1*sin(ang)).^2);

%
    ------------------------------------------------------------------------

% Determine entry condition for second maneuver
rt = [r_GPOPS_P12(end)*cos(theta_GPOPS_P12(end));r_GPOPS_P12(end
    )*sin(theta_GPOPS_P12(end));0];

%apogee and perigee constraints
Vf_mag = sqrt(Vr_GPOPS_P12(end)^2 + Vt_GPOPS_P12(end)^2);
fpa = atan(Vr_GPOPS_P12(end)/Vt_GPOPS_P12(end));

%perifocal velocity
vt = Vf_mag*[-sin(theta_GPOPS_P12(end)-fpa);cos(theta_GPOPS_P12(
    end)-fpa);0];
```

```
[rt_ijk_P12,vt_ijk_P12] = PQW_to_IJK(rt,vt,inc,RAAN,w);
rt_ijk_P12 = rt_ijk_P12*DU;
vt_ijk_P12 = vt_ijk_P12*DU/TU;

[lat_act_enter,long_act_enter] = IJK_to_LATLONG(rt_ijk_P12(1),
    rt_ijk_P12(2),rt_ijk_P12(3),GMST0,tvec_P12(end)*TU);

[r2,v2,t2] = zone_entry_exit2(rt_ijk_P12,vt_ijk_P12,GMST0+
    OmegaEarth*tvec_P12(end)*TU,0,latlim,longlim);

rf2exp = r2;
vf2exp = v2;
tf2exp = t2;

[lat_enter2exp,long_enter2exp] = IJK_to_LATLONG(r2(1),r2(2),r2
    (3),GMST0,tvec_P22(end)*TU);

[rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);

rf_pqw2 = rf_pqw2/DU;
vf_pqw2 = vf_pqw2/DU*TU;

vunit2 = vf_pqw2/norm(vf_pqw2);
hfp2 = cross(rf_pqw2,vf_pqw2);
hunit2 = hfp2/norm(hfp2);

gunit2 = cross(vunit2,hunit2);

%
    ----------------------------------------------------------------------
```

```
% for plotting purposes in PQW frame
% -----------------------------------------------------------------------------

term12 = (be1*cos(phi_GPOPS_P22))^2 + (ae1*sin(phi_GPOPS_P22))
    ^2;
re2 = ae1*be1/sqrt(term12);

rt2 = rf_pqw2 + re2*cos(phi_GPOPS_P22)*vunit2 + re2*sin(
    phi_GPOPS_P22)*gunit2;

%apogee and perigee constraints
Vf_mag2 = sqrt(Vr_GPOPS_P22(end)^2 + Vt_GPOPS_P22(end)^2);
fpa2 = atan(Vr_GPOPS_P22(end)/Vt_GPOPS_P22(end));

%perifocal velocity
vt2 = Vf_mag2*[-sin(theta_GPOPS_P22(end)-fpa2);cos(
    theta_GPOPS_P22(end)-fpa2);0];

[rt_ijk_P22,vt_ijk_P22] = PQW_to_IJK(rt2,vt2,inc,RAAN,w);
rt_ijk_P22 = rt_ijk_P22*DU;
vt_ijk_P22 = vt_ijk_P22*DU/TU;
rt2 = rt2*DU;
[lat_act_enter2,long_act_enter2] = IJK_to_LATLONG(rt_ijk_P22(1),
    rt_ijk_P22(2),rt_ijk_P22(3),GMST0,tvec_P22(end)*TU);

for aa = 1:length(ang)
     r_ell2(:,aa) = rf_pqw2 + re(aa)*cos(ang(aa))*vunit2 + re(aa)
         *sin(ang(aa))*gunit2;
end

```

```
[r4,v4,t4] = zone_entry_exit2(rt_ijk_P22,vt_ijk_P22,GMST0+
    OmegaEarth*tvec_P22(end)*TU,0,latlim,longlim);

rf4exp = r4;
vf4exp = v4;
tf4exp = t4;

[lat_enter4exp,long_enter4exp] = IJK_to_LATLONG(r4(1),r4(2),r4
    (3),GMST0,tvec_P42(end)*TU);

[rf_pqw4,vf_pqw4] = IJK_to_PQW(r4,v4,inc,RAAN,w);

rf_pqw4 = rf_pqw4/DU;
vf_pqw4 = vf_pqw4/DU*TU;

vunit4 = vf_pqw4/norm(vf_pqw4);
hfp4 = cross(rf_pqw4,vf_pqw4);
hunit4 = hfp2/norm(hfp4);

gunit4 = cross(vunit4,hunit4);

for aa = 1:length(ang)
    r_ell4(:,aa) = rf_pqw4 + re(aa)*cos(ang(aa))*vunit4 + re(aa)
        *sin(ang(aa))*gunit4;
end

rt4 = [r_GPOPS_P42(end)*cos(theta_GPOPS_P42(end));r_GPOPS_P42(
    end)*sin(theta_GPOPS_P42(end));0];

%apogee and perigee constraints
Vf_mag4 = sqrt(Vr_GPOPS_P42(end)^2 + Vt_GPOPS_P42(end)^2);
```

```
        fpa4 = atan(Vr_GPOPS_P42(end)/Vt_GPOPS_P42(end));

        %perifocal velocity
        vt4 = Vf_mag4*[-sin(theta_GPOPS_P42(end)-fpa4);cos(
            theta_GPOPS_P42(end)-fpa4);0];

        [rt_ijk_P42,vt_ijk_P42] = PQW_to_IJK(rt4,vt4,inc,RAAN,w);
        rt_ijk_P42 = rt_ijk_P42*DU;
        vt_ijk_P42 = vt_ijk_P42*DU/TU;
        rt4 = rt4*DU;

        [lat_act_enter4,long_act_enter4] = IJK_to_LATLONG(rt_ijk_P42(1),
            rt_ijk_P42(2),rt_ijk_P42(3),GMST0,tvec_P42(end)*TU);

        rgi = zeros(length(r_GPOPS_P12),3);
        vgi = zeros(length(r_GPOPS_P12),3);
        %First maneuver inertial position and velocity
        for dd = 1:length(r_GPOPS_P12)
            %perifocal position vector
            rg_pqw = DU*[r_GPOPS_P12(dd)*cos(theta_GPOPS_P12(dd));
                r_GPOPS_P12(dd)*sin(theta_GPOPS_P12(dd));0];
            %velocity magnitude
            Vf_mag = sqrt(Vr_GPOPS_P12(dd)^2 + Vt_GPOPS_P12(dd)^2);
            fpa = atan(Vr_GPOPS_P12(dd)/Vt_GPOPS_P12(dd));

            %perifocal velocity
            vg_pqw = DU/TU*Vf_mag*[-sin(theta_GPOPS_P12(dd)-fpa);cos(
                theta_GPOPS_P12(dd)-fpa);0];

            [rgi(dd,:),vgi(dd,:)] = PQW_to_IJK(rg_pqw,vg_pqw,inc,RAAN,w)
                ;
        end
```

```
        rgi2 = zeros(length(r_GPOPS_P22),3);
        vgi2 = zeros(length(r_GPOPS_P22),3);
        %First maneuver inertial position and velocity
        for dd = 1:length(r_GPOPS_P22)
            %perifocal position vector
            rg_pqw2 = DU*[r_GPOPS_P22(dd)*cos(theta_GPOPS_P22(dd));
                r_GPOPS_P22(dd)*sin(theta_GPOPS_P22(dd));0];
            %velocity magnitude
            Vf_mag = sqrt(Vr_GPOPS_P22(dd)^2 + Vt_GPOPS_P22(dd)^2);
            fpa = atan(Vr_GPOPS_P22(dd)/Vt_GPOPS_P22(dd));

            %perifocal velocity
            vg_pqw2 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P22(dd)-fpa);cos(
                theta_GPOPS_P22(dd)-fpa);0];

            [rgi2(dd,:),vgi2(dd,:)] = PQW_to_IJK(rg_pqw2,vg_pqw2,inc,
                RAAN,w);
        end

        rgi3 = zeros(length(r_GPOPS_P32),3);
        vgi3 = zeros(length(r_GPOPS_P32),3);
        %First maneuver inertial position and velocity
        for dd = 1:length(r_GPOPS_P32)
            %perifocal position vector
            rg_pqw3 = DU*[r_GPOPS_P32(dd)*cos(theta_GPOPS_P32(dd));
                r_GPOPS_P32(dd)*sin(theta_GPOPS_P32(dd));0];
            %velocity magnitude
            Vf_mag = sqrt(Vr_GPOPS_P32(dd)^2 + Vt_GPOPS_P32(dd)^2);
            fpa = atan(Vr_GPOPS_P32(dd)/Vt_GPOPS_P32(dd));

            %perifocal velocity
```

```
        vg_pqw3 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P32(dd)-fpa);cos(theta_GPOPS_P32(dd)-fpa);0];

        [rgi3(dd,:),vgi3(dd,:)] = PQW_to_IJK(rg_pqw3,vg_pqw3,inc,RAAN,w);
    end

    rgi4 = zeros(length(r_GPOPS_P42),3);
    vgi4 = zeros(length(r_GPOPS_P42),3);
    %First maneuver inertial position and velocity
    for dd = 1:length(r_GPOPS_P42)
        %perifocal position vector
        rg_pqw4 = DU*[r_GPOPS_P42(dd)*cos(theta_GPOPS_P42(dd));r_GPOPS_P42(dd)*sin(theta_GPOPS_P42(dd));0];
        %velocity magnitude
        Vf_mag = sqrt(Vr_GPOPS_P42(dd)^2 + Vt_GPOPS_P42(dd)^2);
        fpa = atan(Vr_GPOPS_P42(dd)/Vt_GPOPS_P42(dd));

        %perifocal velocity
        vg_pqw4 = DU/TU*Vf_mag*[-sin(theta_GPOPS_P42(dd)-fpa);cos(theta_GPOPS_P42(dd)-fpa);0];

        [rgi4(dd,:),vgi4(dd,:)] = PQW_to_IJK(rg_pqw4,vg_pqw4,inc,RAAN,w);
    end

    %%

    %save optimal path in structure
    optans2.ics = struct('r0',r0vec,'v0',v0vec,'t0',t0,'ae',ae,'be',be,'Rmax',Rmax,'Rmin',Rmin,'latlim',latlim,'longlim',longlim,'GMST0',GMST0,...
```

```
        'inc',inc,'RAAN',RAAN,'w',w,'ang',ang);
    optans2.scale = struct('TU',TU,'DU',DU,'MU',MU2);
    optans2.entry(1) = struct('lat_enter',lat_enter,'long_enter',
        long_enter,'r_ell',r_ell,'rtijk',rgi(end,:),'vtijk',vgi(end
        ,:),'rt_pqw',rg_pqw,'rf_pqw',rf_pqw,...
        'lat_act_enter',lat_act_enter,'long_act_enter',
            long_act_enter,'rf1',rf1,'vf1',vf1,'tf1',tf1);
    optans2.entry(2) = struct('lat_enter',lat_enter2exp,'long_enter'
        ,long_enter2exp,'r_ell',r_ell2,'rtijk',rgi2(end,:),'vtijk',
        vgi2(end,:),'rt_pqw',rt2,'rf_pqw',rf_pqw2,...
        'lat_act_enter',lat_act_enter2,'long_act_enter',
            long_act_enter2,'rf1',rf2exp,'vf1',vf2exp,'tf1',tf2exp);
    optans2.entry(4) = struct('lat_enter',lat_enter4exp,'long_enter'
        ,long_enter4exp,'r_ell',r_ell4,'rtijk',rgi4(end,:),'vtijk',
        vgi4(end,:),'rt_pqw',rt4,'rf_pqw',rf_pqw4,...
        'lat_act_enter',lat_act_enter4,'long_act_enter',
            long_act_enter4,'rf1',rf4exp,'vf1',vf4exp,'tf1',tf4exp);
    optans2.phase(1) = struct('state',solution2.phase(1).state,'
        costate',solution2.phase(1).costate,'control',solution2.
        phase(1).control,'time',tvec_P12,'rgi',rgi);
    optans2.phase(2) = struct('state',solution2.phase(2).state,'
        costate',solution2.phase(2).costate,'control',solution2.
        phase(2).control,'time',tvec_P22,'rgi',rgi2);
    optans2.phase(3) = struct('state',solution2.phase(3).state,'
        costate',solution2.phase(3).costate,'control',solution2.
        phase(3).control,'time',tvec_P32,'rgi',rgi3);
    optans2.phase(4) = struct('state',solution2.phase(4).state,'
        costate',solution2.phase(4).costate,'control',solution2.
        phase(4).control,'time',tvec_P42,'rgi',rgi4);
    optans2.parameter = solution2.parameter;

    r0string = num2str(norm(r0vec));
```

```
        aestr = num2str(ae);
        itstr = num2str(PSO_data(cc,end));
        tempstr = [aestr itstr];
        aestring = num2str(tempstr);

        dir = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Triple Pass\Images\';
        dir2 = 'C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust RTM\
            Triple Pass\Data\';

        tend = toc(tstart);

        exflag = output.result.nlpinfo;

        fid2 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust
             RTM\Triple Pass\Data\PSO2GPOPSTriplePassData120.txt','a');

        fprintf(fid2,'%i\t %i\t %4.3f\t %4.3f\t %4.3f\t %6.5f\t %6.5f\t
            %6.5f\t %6.2f\t %i\r\n',...
            norm(r0),ae,phi_GPOPS_P12,phi_GPOPS_P22,phi_GPOPS_P32,
                PSO_data(cc,18),Cost,Cost2,tend,exflag);

        fid3 = fopen('C:\Users\Dan Showalter\Documents\MATLAB\Low Thrust
             RTM\Triple Pass\Data\TriplePassCost120.txt','a');
        fprintf(fid3,'%i\t %i  \t %4.3f\t %4.3f\t %4.3f\t %4.3f\t %i\t\r
            \n',...
            norm(r0),ae,solution2.phase(1).integral*DU/TU*1000,solution2
                .phase(2).integral*DU/TU*1000,solution2.phase(4).
                integral*DU/TU*1000,Cost2,exflag);

        if exflag == 0
            %plot optimal results
```

```
            [optout] = LT_TRIPLE_PASS_PLOTS(optans2,r0string,aestring,dir);

            dataname = ['TriplePass' r0string aestring];

            %save data
            save(strcat(dir2,[dataname]),'output');
        end

        clear optans
        close all

    end
end
```

E.3.2.2 Triple Pass LTRTM Equations of Motion and Cost Function

```
function phaseout = LT_3RTM_Continuous_4Phase(input)

%% Phase 1

s1 = input.phase(1).state;
u1 = input.phase(1).control;

% Equations of Motion
% ---------------------------------------------------------------------------

r1 = s1(:,1);
vr1 = s1(:,3);
vtheta1 = s1(:,4);

T1 = u1(:,1);
```

```
B1 = u1(:,2);

MU2 = input.auxdata.MU;

r_dot1 = vr1;
theta_dot1 = vtheta1./r1;
vr_dot1 = (vtheta1.^2)./r1 - MU2./(r1.^2) + T1.*sin(B1);
vtheta_dot1 = -vtheta1.*vr1./r1 + T1.*cos(B1);

% Form matrix output
daeout1 = [r_dot1 theta_dot1 vr_dot1 vtheta_dot1];

phaseout(1).dynamics = daeout1;
%
    ---------------------------------------------------------------------------

% Cost Function
phaseout(1).integrand = T1;

%% Phase 2

s2 = input.phase(2).state;
u2 = input.phase(2).control;

% Equations of Motion
%
    ---------------------------------------------------------------------------

r2 = s2(:,1);
vr2 = s2(:,3);
vtheta2 = s2(:,4);

```

```
T2 = u2(:,1);
B2 = u2(:,2);

r_dot2 = vr2;
theta_dot2 = vtheta2./r2;
vr_dot2 = (vtheta2.^2)./r2 - MU2./(r2.^2) + T2.*sin(B2);
vtheta_dot2 = -vtheta2.*vr2./r2 + T2.*cos(B2);

% Form matrix output
daeout2 = [r_dot2 theta_dot2 vr_dot2 vtheta_dot2];

phaseout(2).dynamics = daeout2;
% ----------------------------------------------------------------------------
% Cost Function
phaseout(2).integrand = T2;

%% Phase 3

s3 = input.phase(3).state;
u3 = input.phase(3).control;

% Equations of Motion
% ----------------------------------------------------------------------------
r3 = s3(:,1);
vr3 = s3(:,3);
vtheta3 = s3(:,4);

T3 = u3(:,1);
```

```
B3 = u3(:,2);

r_dot3 = vr3;
theta_dot3 = vtheta3./r3;
vr_dot3 = (vtheta3.^2)./r3 - MU2./(r3.^2) + T3.*sin(B3);
vtheta_dot3 = -vtheta3.*vr3./r3 + + T3.*cos(B3);

% Form matrix output
daeout3 = [r_dot3 theta_dot3 vr_dot3 vtheta_dot3];

phaseout(3).dynamics = daeout3;
% ---------------------------------------------------------------------------

% Cost Function
phaseout(3).integrand = zeros(length(r3),1);

%% Phase 4

s4 = input.phase(4).state;
u4 = input.phase(4).control;

% Equations of Motion
% ---------------------------------------------------------------------------

r4 = s4(:,1);
vr4 = s4(:,3);
vtheta4 = s4(:,4);

T4 = u4(:,1);
B4 = u4(:,2);
```

```

r_dot4 = vr4;
theta_dot4 = vtheta4./r4;
vr_dot4 = (vtheta4.^2)./r4 - MU2./(r4.^2) + T4.*sin(B4);
vtheta_dot4 = -vtheta4.*vr4./r4 + T4.*cos(B4);

% Form matrix output
daeout4 = [r_dot4 theta_dot4 vr_dot4 vtheta_dot4];

phaseout(4).dynamics = daeout4;
% --------------------------------------------------------------------------
% Cost Function
phaseout(4).integrand = T4;
```

E.3.2.3 Triple Pass LTRTM Constraints

```
function output = LT_3RTM_Endpoint_4Phase(input)


%% Cost Function Evaluation
% --------------------------------------------------------------------------
J = input.phase(1).integral + input.phase(2).integral + input.phase(3).integral + input.phase(4).integral;
output.objective  = J;
% --------------------------------------------------------------------------
%% Event Constraints

```

```
%% Phase 1 (First Maneuver)
%phase 2 variables
tf1 = input.phase(1).finaltime;
xf1 = input.phase(1).finalstate;
p = input.parameter;
phi = p(1);

%phase 2 variables
t02 = input.phase(2).initialtime;
tf2 = input.phase(2).finaltime;
x02 = input.phase(2).initialstate;
xf2 = input.phase(2).finalstate;
phi2 = p(2);

%phase 3 variables
t03 = input.phase(3).initialtime;
tf3 = input.phase(3).finaltime;
x03 = input.phase(3).initialstate;
xf3 = input.phase(3).finalstate;


%phase 3 variables
t04 = input.phase(4).initialtime;
tf4 = input.phase(4).finaltime;
x04 = input.phase(4).initialstate;
xf4 = input.phase(4).finalstate;
phi3 = p(3);

rf = xf1(1);
thetaf = xf1(2);
Vrf = xf1(3);
Vtf = xf1(4);
```

```
ae1 = input.auxdata.ae; %semimajor axis of exclusion ellipse
be1 = input.auxdata.be; % semiminor axis of exclusion ellipse
MU2 = input.auxdata.MU; %gravitational parameter scaled by DU and TU
rf_pqw = input.auxdata.rf_pqw; % perifocal position vector of initial
    corssing into exclusion zone
vunit = input.auxdata.vunit; %perifocal unit velocity vector of initial
    crossing into exclusion zone
gunit = input.auxdata.gunit; %perifocal unit vetcor of initial crossing
    into exclusion zone

term1 = (be1*cos(phi))^2 + (ae1*sin(phi))^2;

re = ae1*be1/sqrt(term1);

rt = rf_pqw + re*cos(phi)*vunit + re*sin(phi)*gunit;

%final position constraints
event1 = rf*cos(thetaf) - rt(1);
event2 = rf*sin(thetaf) - rt(2);

%velocity magnitude and flight path angle
Vf_mag = sqrt(Vrf^2 + Vtf^2);
fpa = atan(Vrf/Vtf);

%perifocal velocity
vt = Vf_mag*[-sin(thetaf-fpa);cos(thetaf-fpa);0];

[a,ecc,~,~,~,~] = RV2COE_MU(rt,vt,MU2);
Ra = a*(1+ecc);
Rp = a*(1-ecc);

```

```
event3 = Ra;
event4 = Rp;



% Linkage Constraints
event1_link_state = x02 - xf1;
event1_link_time = t02 - tf1;

output.eventgroup(1).event = [event1_link_state event1_link_time event1
    event2 event3 event4];

%% Phase 2 (Second Maneuver)

% constant variables
inc = input.auxdata.inc; %inclination of initial orbit (used to convert
    everything into perifocal frame of initial orbit)
RAAN = input.auxdata.RAAN; %RAAN of initial orbit (used to convert
    everything into perifocal frame of initial orbit)
w = input.auxdata.w; %argument of perigee of initial orbit (used to
    convert everything into perifocal frame of initial orbit)
latlim = input.auxdata.latlim;
longlim = input.auxdata.longlim;
GMST0 = input.auxdata.GMST0;
OmegaEarth = input.auxdata.OmegaEarth;
DU = input.auxdata.DU;
TU = input.auxdata.TU;


rf2 = xf2(1);
thetaf2 = xf2(2);
Vrf2 = xf2(3);
```

```
Vtf2 = xf2(4);

%position and velocity of initial intercept in perifocal frame of initial
%orbit
if isnan(rf) == 1 || isnan(thetaf) == 1 || isnan(Vf_mag) == 1 || isnan(tf1) == 1 || isnan(phi) == 1 ...
     || isnan(rf2) == 1 || isnan(thetaf2) == 1  || isnan(tf2) == 1 || isnan(phi2) == 1
        event21 = NaN;
        event22 = NaN;
        event25 = NaN;
        event23 = NaN;
        event24 = NaN;
        Vf_mag2 = NaN;
else
[rt_ijk,vt_ijk] = PQW_to_IJK(rt,vt,inc,RAAN,w);
rt_ijk = rt_ijk*DU;
vt_ijk = vt_ijk*DU/TU;

[r2,v2,t2,~,~,~,~,t2_exit] = zone_entry_exit2(rt_ijk,vt_ijk,GMST0+OmegaEarth*tf1*TU,0,latlim,longlim);

[rf_pqw2,vf_pqw2] = IJK_to_PQW(r2,v2,inc,RAAN,w);

rf_pqw2 = rf_pqw2/DU;
vf_pqw2 = vf_pqw2/DU*TU;

vunit2 = vf_pqw2/norm(vf_pqw2);
hfp2 = cross(rf_pqw2,vf_pqw2);
hunit2 = hfp2/norm(hfp2);

```

```
gunit2 = cross(vunit2,hunit2);

term12 = (be1*cos(phi2))^2 + (ae1*sin(phi2))^2;
re2 = ae1*be1/sqrt(term12);

rt2 = rf_pqw2 + re2*cos(phi2)*vunit2 + re2*sin(phi2)*gunit2;

%final position constraints
event21 = rf2*cos(thetaf2) - rt2(1);
event22 = rf2*sin(thetaf2) - rt2(2);

%apogee and perigee constraints
Vf_mag2 = sqrt(Vrf2^2 + Vtf2^2);
fpa2 = atan(Vrf2/Vtf2);

%perifocal velocity
vt2 = Vf_mag2*[-sin(thetaf2-fpa2);cos(thetaf2-fpa2);0];

[a2,ecc2,~,~,~,~] = RV2COE_MU(rt2,vt2,MU2);
Ra2 = a2*(1+ecc2);
Rp2 = a2*(1-ecc2);

event23 = Ra2;
event24 = Rp2;

event25 = tf2 - (tf1 + t2/TU);
end

% Linkage Constraints
event2_link_state = x03 - xf2;
event2_link_time = t03 - tf2;

```

```
output.eventgroup(2).event = [event2_link_state event2_link_time event21
     event22 event25 event23 event24];

%% Phase 3 (Coast Phase)

rf3 = xf3(1);
thetaf3 = xf3(2);
Vrf3 = xf3(3);
Vtf3 = xf3(4);


if isnan(rf) == 1 || isnan(thetaf) == 1 || isnan(Vf_mag) == 1 || isnan(
    tf1) == 1 || isnan(phi) == 1 ...
        || isnan(rf2) == 1 || isnan(thetaf2) == 1 || isnan(Vf_mag2) == 1
             || isnan(tf2) == 1 || isnan(phi2) == 1 ...
        || isnan(t2_exit)
    event31 = NaN;
else
    event31 = tf3 - (tf1+t2_exit/TU);

end
event3_link_state = x04 - xf3;
event3_link_time = t04 - tf3;


output.eventgroup(3).event = [event3_link_state event3_link_time event31
    ];

%% Phase 4 (Third Maneuver)


rf4 = xf4(1);
```

```
thetaf4 = xf4(2);
Vrf4 = xf4(3);
Vtf4 = xf4(4);

%position and velocity of initial intercept in perifocal frame of initial
%orbit
if isnan(rf) == 1 || isnan(thetaf) == 1 || isnan(Vf_mag) == 1 || isnan(tf1) == 1 || isnan(phi) == 1 ...
        || isnan(rf2) == 1 || isnan(thetaf2) == 1 || isnan(Vf_mag2) == 1 || isnan(tf2) == 1 || isnan(phi2) == 1 ...
        || isnan(tf3) == 1 || isnan(Vrf3) == 1 || isnan(Vtf3) == 1 || isnan(thetaf3) == 1 || isnan(rf3) == 1
    event41 = NaN;
    event42 = NaN;
    event45 = NaN;
    event43 = NaN;
    event44 = NaN;
else

    %apogee and perigee constraints at terminus of third phase
    Vf_mag3 = sqrt(Vrf3^2 + Vtf3^2);
    fpa3 = atan(Vrf3/Vtf3);

    %perifocal velocity at terminus of thrid phase
    vt3 = Vf_mag3*[-sin(thetaf3-fpa3);cos(thetaf3-fpa3);0];

    rt3 = [rf3*cos(thetaf3);rf3*sin(thetaf3);0];

    [rt_ijk3,vt_ijk3] = PQW_to_IJK(rt3,vt3,inc,RAAN,w);
    rt_ijk3 = rt_ijk3*DU;
    vt_ijk3 = vt_ijk3*DU/TU;
```

```
[r4,v4,t4] = zone_entry_exit2(rt_ijk3,vt_ijk3,GMST0+OmegaEarth*(tf3)
    *TU,0,latlim,longlim);

[rf_pqw4,vf_pqw4] = IJK_to_PQW(r4,v4,inc,RAAN,w);

rf_pqw4 = rf_pqw4/DU;
vf_pqw4 = vf_pqw4/DU*TU;

vunit4 = vf_pqw4/norm(vf_pqw4);
hfp4 = cross(rf_pqw4,vf_pqw4);
hunit4 = hfp4/norm(hfp4);

gunit4 = cross(vunit4,hunit4);

term14 = (be1*cos(phi3))^2 + (ae1*sin(phi3))^2;
re4 = ae1*be1/sqrt(term14);

rt4 = rf_pqw4 + re4*cos(phi3)*vunit4 + re4*sin(phi3)*gunit4;

%final position constraints
event41 = rf4*cos(thetaf4) - rt4(1);
event42 = rf4*sin(thetaf4) - rt4(2);

%apogee and perigee constraints
Vf_mag4 = sqrt(Vrf4^2 + Vtf4^2);
fpa4 = atan(Vrf4/Vtf4);

%perifocal velocity
vt4 = Vf_mag4*[-sin(thetaf4-fpa4);cos(thetaf4-fpa4);0];

[a4,ecc4,~,~,~,~] = RV2COE_MU(rt4,vt4,MU2);
```

```
    Ra4 = a4*(1+ecc4);
    Rp4 = a4*(1-ecc4);

    event43 = Ra4;
    event44 = Rp4;

    event45 = tf4 - (tf3 + t4/TU);
end


output.eventgroup(4).event = [event41 event42 event45 event43 event44];
```

Appendix F: Code for Geostationary Transfer Maneuvers

F.1 Three Target GTMEI

F.1.1 Enumeration Script

```
%%

%maximum number of entries into exclusion zone before maneuver is
%required
close all
% clear all
clc

el_val_pass = 0;
el_val_shadow = 1;
GMST0 = 0;
lat_site = pi/4;
long_site = 0;
t0 = 0;
tf_max = 36*3600;
tstep = 1;
r_cyl = 1;
xmin = 1;
xmax = 3;
%design variables
%entry and exit locations on relative lobe
% psi0 = 0;
% psif = pi;
% x_cyl_in = 5;
% x_cyl_out = xmin;
%
```

```
% coast0 = 0;
% coastf = 0;


wgs84data
global MU OmegaEarth RE

%% Determine Chief Satellite Entry/Exit over Exclusion Zone

%Initial COEs of chief satellite
a_chief_vec = [26581.76 7378 6878];
e_chief = 0;
i_chief = 55*pi/180;
O_chief = 0;
o_chief = 0;
% nu_chief_vec = 0;
nu_chief = 0;

chief_params = [e_chief;i_chief;O_chief;o_chief;nu_chief];
%Initial COEs of deputy satellite
a_dep = 6578;
e_dep = 0;
i_dep = 55*pi/180;
O_dep = 0;
o_dep = 0;
nu_dep0 = 0;

dep_params = [a_dep;e_dep;i_dep;O_dep;o_dep];

a_GEO = 42164.14;
e_GEO = 0;
i_GEO = 0;
```

```
O_GEO = 0;
o_GEO = 0;

GEO_params = [a_GEO;e_GEO;i_GEO;O_GEO;o_GEO];


for aa = 1:length(a_chief_vec)
    [C_times_c,C_ind_c,Rijk_c,Vijk_c,~,~,rho_vec_cw_c,Tvec_c,~,~] =
        contact_times(a_chief_vec(aa),i_chief,e_chief,O_chief,o_chief,
        nu_chief,long_site,GMST0,tf_max+16*3600,tstep,lat_site,
        el_val_pass);
    val_ind = find(C_times_c(:,1) < tf_max);
    max_coastf = C_times_c(max(val_ind)+1,1)- C_times_c(max(val_ind),2);
    C_times_c = C_times_c(val_ind,:);
    [max_ind,~] = size(C_times_c);
    ThreePassEnumData(aa).times = C_times_c;
    ThreePassEnumData(aa).ind = C_ind_c;
    ThreePassEnumData(aa).Rc = Rijk_c;
    ThreePassEnumData(aa).Vc = Vijk_c;
    ThreePassEnumData(aa).rho_c = rho_vec_cw_c;
    ThreePassEnumData(aa).Tc = Tvec_c;
    ThreePassEnumData(aa).max_ind = max_ind;
    ThreePassEnumData(aa).max_coastf = max_coastf;
    clear C_times_c C_ind_c Rijk_c Vijk_c rho_vec_cw_c Tvec_c max_ind
        max_coastf
end

save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
    Article_Data\Rev2\ThreePassEnumData.mat','ThreePassEnumData')

for bb = 3:3

```

```
    C_times_c = ThreePassEnumData(bb).times;
    C_ind_c = ThreePassEnumData(bb).ind;
    Rijk_c = ThreePassEnumData(bb).Rc;
    Vijk_c = ThreePassEnumData(bb).Vc;
    rho_vec_cw_c = ThreePassEnumData(bb).rho_c;
    Tvec_c = ThreePassEnumData(bb).Tc;
    max_ind = ThreePassEnumData(bb).max_ind;


    a_chief = a_chief_vec(bb);
    if C_times_c(1,1) == 0
        min_start = 2;
    else
        min_start = 1;
    end

    for cc = 14:max_ind
        if cc == 14
            startval = 19;
        else
            startval = 1;
        end

        for dd = startval:20

            tstart = tic;
            %Period of Chief satellite's orbit
            Pc = 2*pi*sqrt(a_chief^3/MU);


            t_enter = C_times_c(cc,1);
            t_exit = C_times_c(cc,2);
```

```
        t_zone = t_exit - t_enter;

        %determine indices of minimum duration contact
        C_ind_contact = C_ind_c(cc,:);

        %find unit vector pointing towards the deputy that puts
            chief between
        %ground site and deputy
        rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):C_ind_contact
            (2));

        %determine alpha and beta angles during contact times
        [alphavec,betavec] = alphabeta(rho_unit_cw);

        %Vector of times for propogation
        T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) - Tvec_c(
            C_ind_contact(1))*ones(length(Tvec_c(C_ind_contact(1):
            C_ind_contact(2))),1);

        %Determine position/velocity vectors of chief satellite upon
             intial/final
        %contact
        chief_pos0 = Rijk_c(:,C_ind_c(cc,1));
        chief_vel0 = Vijk_c(:,C_ind_c(cc,1));
        chief_posf = Rijk_c(:,C_ind_c(cc,2));
        chief_velf = Vijk_c(:,C_ind_c(cc,2));

        if cc < max_ind
            max_coastf = C_times_c(cc+1,1) - C_times_c(cc,2);
        else
            max_coastf = ThreePassEnumData(bb).max_coastf;
        end
```

```matlab

            %time variables have precision to 1 second.  Others have
            %precision to 0.001 units (km,rad)
            prec2 = [2;0;3;3;0;2;0;6];


            [Jmin,~,gbest,~,k,JG] = PSO_REL_SHADOW_DV4(7,[0 2*pi;1 C_times_c(cc,1);xmin xmax;xmin xmax;1 max_coastf;0 2*pi;1 16*3600],prec2,500,300,chief_pos0,chief_vel0,chief_posf,...
                chief_velf,dep_params,GEO_params,alphavec,betavec,t_zone,Pc,t_enter,t_exit,r_cyl,T_prop,GMST0,lat_site,long_site,tstep,el_val_shadow);

            tend = toc(tstart);

            infvec = find(JG == Inf);
            inftot = length(infvec);

            fid = fopen('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\Article_Data\Rev2\ThreePassEnumerationSat3.txt','a');
            fprintf(fid,'%2i \t\t %2i \t\t %2i \t\t %3.2f \t\t %6i \t\t %4.3f \t\t %4.3f \t\t %6i \t %3.2f \t\t %6i \t\t %6.5f \t\t %5i \t\t %5i \t\t %6.2f\r\n',dd,bb,cc,gbest(1),gbest(2),gbest(3),gbest(4),gbest(5),gbest(6),gbest(7),Jmin,k,inftot,tend);

        end
    end
end
```

F.1.1.1 Determine Target Contact Times with Ground Site

```
function [C_times,C_ind,rijk,vijk,rgs,rho_sez,rho_RIC,tvec,uvec,long_site,nu_vec] = contact_times(a,inc,ecc,Omega,omega,nu0,lambda0,GMST0,tmax,tstep,lat_site,el_val)

%INPUTS
% a = satellite semimajor axis (km)
% inc = satellite inclination (rad)
% ecc = satellite eccentricity
% Omega = satellite RAAN (rad)
% omega = satellite argument of perigee (rad)
% nu0 = initial true anomaly (rad)
% lambda0 = initial GMST of ground site
% tmax = maximum scenario time (sec)
% tstep = time step (sec)

%OUTPUTS
% C_times = times satellite is in contact with the ground site
% C_ind = indices of satellite contat times
% rijk = position vectors of satellite at discretized times
% vijk = velocity vectors of satellite at discretized times
% rgs = position vectors of the ground site at discretized times
% rho_sez = vector from ground site to satellite in SEZ coordinates
% rho_RIC = vector from ground site to satellite in RIC coordinates
% ===========================================================================

tvec = (0:tstep:tmax)';

% determine true anomaly of spacecraft at each time step
[nu_vec] = nuf_from_TOF_vec(nu0,tvec,a,ecc);

```

```
%determine inertial position and velocity vectors at each tiem step
[rijk,vijk,r] = COE2RV_vec(a,ecc,inc,Omega,omega,nu_vec);

r = r';

wgs84data

global OmegaEarth RE

long_site = lambda0 + GMST0 + OmegaEarth*tvec;

Rsite = zeros(3,length(tvec));
Rsite(1,:) = RE*cos(lat_site).*cos(long_site);
Rsite(2,:) = RE*cos(lat_site).*sin(long_site);
Rsite(3,:) = RE*sin(lat_site);

rho_ijk = rijk - Rsite;

temp = zeros(3,length(tvec));
temp(1,:) = cos(long_site').*rho_ijk(1,:) + sin(long_site').*rho_ijk
    (2,:);
temp(2,:) = -sin(long_site').*rho_ijk(1,:) + cos(long_site').*rho_ijk
    (2,:);
temp(3,:) = rho_ijk(3,:);

rho_sez = zeros(3,length(tvec));

rho_sez(1,:) = cos(pi/2 - lat_site)*temp(1,:) - sin(pi/2 - lat_site)*
    temp(3,:);
rho_sez(2,:) = temp(2,:);
rho_sez(3,:) = sin(pi/2 - lat_site)*temp(1,:) + cos(pi/2 - lat_site)*
    temp(3,:);
```

```
ind_l = 1;
while ind_l == 1

    ind7 = find(long_site > 2*pi);
    if isempty(ind7)
        ind_l = 0;
    else
    long_site(ind7) = long_site(ind7) - 2*pi;
    end

end

uvec = omega + nu_vec;

zone_ind = zeros(length(tvec),1);
num_in = 0;

%inertial coordinates of ground site
rgs = zeros(3,length(tvec));
rgs(1,:) = RE*cos(lat_site)*cos(long_site');
rgs(2,:) = RE*cos(lat_site)*sin(long_site');
rgs(3,:) = RE*sin(lat_site);

%vector from ground site to satellite
rho_ijk = rijk - rgs;
```

```
%transform into sez coordinates
temp = zeros(3,length(tvec));
temp(1,:) = cos(long_site').*rho_ijk(1,:) + sin(long_site').*rho_ijk
    (2,:);
temp(2,:) = -sin(long_site').*rho_ijk(1,:) + cos(long_site').*rho_ijk
    (2,:);
temp(3,:) = rho_ijk(3,:);

rho_sez = zeros(3,length(tvec));

rho_sez(1,:) = cos(pi/2 - lat_site)*temp(1,:) - sin(pi/2 - lat_site)*
    temp(3,:);
rho_sez(2,:) = temp(2,:);
rho_sez(3,:) = sin(pi/2 - lat_site)*temp(1,:) + cos(pi/2 - lat_site)*
    temp(3,:);

rho_mag = sqrt(rho_sez(1,:).^2 + rho_sez(2,:).^2 + rho_sez(3,:).^2);

el_vec = asind(rho_sez(3,:)./rho_mag);


for aa = 1:length(tvec)
    if el_vec(aa) > el_val
        zone_ind(aa) = 1;
        if aa == 1
            num_in = num_in + 1;
            C_times(num_in,1) = tvec(aa);
            C_ind(num_in,1) = aa;
        else
            if zone_ind(aa - 1) == 0
                num_in = num_in + 1;
```

```
                C_times(num_in,1) = tvec(aa);
                C_ind(num_in,1) = aa;
            end
        end
    else
        zone_ind(aa) = 0;
        if aa ~= 1
            if zone_ind(aa - 1) == 1
                C_times(num_in,2) = tvec(aa-1);
                C_ind(num_in,2) = aa - 1;
            end
        end
    end
    if aa == length(tvec) && zone_ind(aa -1) == 1
        C_times(num_in,2) = tvec(aa);
        C_ind(num_in,2) = aa;
    end
end

%convert ijk to RIC
temp2 = zeros(3,length(tvec));
temp2(1,:) = cos(Omega)*rho_ijk(1,:) + sin(Omega)*rho_ijk(2,:);
temp2(2,:) = cos(Omega)*rho_ijk(2,:) - sin(Omega)*rho_ijk(1,:);
temp2(3,:) = rho_ijk(3,:);

temp3 = zeros(3,length(tvec));
temp3(1,:) = temp2(1,:);
temp3(2,:) = cos(inc)*temp2(2,:) + sin(inc)*temp2(3,:);
temp3(3,:) = cos(inc)*temp2(3,:) - sin(inc)*temp2(2,:);

rho_RIC = zeros(3,length(tvec));

```

```
rho_RIC(1,:) = cos(uvec').*temp3(1,:) + sin(uvec').*temp3(2,:);
rho_RIC(2,:) = cos(uvec').*temp3(2,:) - sin(uvec').*temp3(1,:);
rho_RIC(3,:) = temp3(3,:);

norm_vec = sqrt(rho_RIC(1,:).^2 + rho_RIC(2,:).^2 + rho_RIC(3,:).^2);

rho_RIC(1,:) = rho_RIC(1,:)./norm_vec;
rho_RIC(2,:) = rho_RIC(2,:)./norm_vec;
rho_RIC(3,:) = rho_RIC(3,:)./norm_vec;

if num_in == 0
    C_times = 0;
    C_ind = 0;
end
```

F.1.1.2 Determine Final True Anomaly Given Time of Flight

```
function [nuf] = nuf_from_TOF_vec(nu0,TOF_vec,a,e)

wgs84data
global MU

nuf = zeros(length(TOF_vec),1);
Eg = zeros(length(TOF_vec),1);



%% 1) compute orbital mean motion
n = sqrt(MU/abs(a)^3);



%% 2) convert initial true anomaly to initial mean anomaly
if e < 1
    if nu0 == 0;
```

```
        M0 = 0;

    elseif nu0 == pi

        M0 = pi;

    else

        E0 = acos((e+cos(nu0))/(1+e*cos(nu0)));


        if (nu0 > pi)

            E0 = 2*pi - E0;

        end

        M0 = E0 - e*sin(E0);

    end

    M0 = M0*ones(length(TOF_vec),1);



    %% 3) compute final mean anomaly
    Mold = M0 + n*TOF_vec;
    N = Mold/(2*pi);
    Mf = Mold - floor(N)*2*pi;



    Mflag = 1;

```

```
    while Mflag == 1
        ind_Mf = find(Mf > 2*pi);

        if isempty(ind_Mf) == 0

            Mf(ind_Mf) = Mf(ind_Mf) - 2*pi;
        else
            Mflag = 0;
        end

    end


    ind_Eg1 = find(Mf > pi);
    ind_Eg2 = find(Mf <= pi);

    Eg(ind_Eg1) = Mf(ind_Eg1) - e;
    Eg(ind_Eg2) = Mf(ind_Eg2) + e;

    Ef = Eg + (Mf - Eg + e*sin(Eg))./(1 - e*cos(Eg));

    Eflag = 1;

    while Eflag == 1
        diff = abs(Ef - Eg);

        ind_Ef = find(diff > 1e-8);

        if isempty(ind_Ef) == 0
            Eg = Ef;
            Ef = Eg + (Mf - Eg + e*sin(Eg))./(1 - e*cos(Eg));
        else
```

```
                Eflag = 0;
            end
        end

        nuf = acos((cos(Ef)-e)./(1-e*cos(Ef)));

        ind_quad = find(Ef > pi);

        nuf(ind_quad) = 2*pi - nuf(ind_quad);



    elseif e > 1
    %% Hyperbolic orbits
        sinh_H0 = sin(nu0)*sqrt(e^2 - 1)/(1+e*cos(nu0));

        H = zeros(length(TOF_vec),1);
        M = zeros(length(TOF_vec),1);
        M0 = e*sinh_H0 - asinh(sinh_H0);

        Mold = M0 + n*TOF_vec;
        N = Mold/(2*pi);
        M = Mold - floor(N)*2*pi;

        if e < 1.6
            H = M + e;
            ind1 = find(-pi < M & M < 0);
            ind4 = find(M > pi);
            H(ind1) = M(ind1) - e;
            H(ind4) = M(ind4) - e;
        else
            H = M/(e - 1);
```

```
        if e < 3.6
            ind2 = find(abs(M) > pi);
            if isempty(ind2) == 0
                H(ind2) = M(ind2) - sign(M(ind2))*e;
            end
        end
    end

    Hflag = 1;
    Hg = H;

    while Hflag == 1
       Hnew = Hg + (M - e*sinh(Hg) + Hg)./(e*cosh(Hg) - 1);

       diff = abs(Hnew - Hg);

       ind_H = find(diff > 1e-8);
       if isempty(ind_H) == 0
           Hg = Hnew;
       else
           Hflag = 0;
       end
    end

    sin_nu = (-sinh(Hnew)*sqrt(e^2 - 1))./(1 - e*cosh(Hnew));

    cos_nu = (cosh(Hnew) - e)./(1-e*cosh(Hnew));

    nuf = atan2(sin_nu,cos_nu);


end
```

```
if isreal(nuf) == 0
    a
    e
    nu0
    save nuf
end
% if e == 1
%     keyboard
% end
% zer_val = find(nuf == 0);
% if zer_val == length(nuf)
%     keyboard
% end
```

F.1.1.3 Determine State Given COEs

```
function [R_ijk,V_ijk,r] = COE2RV_vec(a,ecc,inc,RAAN,w,nu_vec)

%Author: Dan Showalter 18 Oct 2012

%Purpose: find inertial position and velocity vector gievn classical
%orbital elements

%% Algorithm
MU = 398600.5;

dim = length(nu_vec);

p = a*(1-ecc^2);

r = p./(1+ecc*cos(nu_vec));

```

```
R_pqw = zeros(3,dim);
V_pqw = zeros(3,dim);
R_ijk = zeros(3,dim);
V_ijk = zeros(3,dim);



R_pqw(1,:) = (r.*cos(nu_vec))';
R_pqw(2,:) = (r.*sin(nu_vec))';
V_pqw(1,:) = sqrt(MU/p)*(-sin(nu_vec)');
V_pqw(2,:) = sqrt(MU/p)*(ecc+cos(nu_vec))';

%first rotation about vertical axis by -w
R_temp1(1,:) = cos(-w)*R_pqw(1,:) + sin(-w)*R_pqw(2,:);
R_temp1(2,:) = cos(-w)*R_pqw(2,:) - sin(-w)*R_pqw(1,:);
R_temp1(3,:) = R_pqw(3,:);

V_temp1(1,:) = cos(-w)*V_pqw(1,:) + sin(-w)*V_pqw(2,:);
V_temp1(2,:) = cos(-w)*V_pqw(2,:) - sin(-w)*V_pqw(1,:);
V_temp1(3,:) = V_pqw(3,:);

%2nd rotation about primary axis by -inc
R_temp2(1,:) = R_temp1(1,:);
R_temp2(2,:) = cos(-inc)*R_temp1(2,:) + sin(-inc)*R_temp1(3,:);
R_temp2(3,:) = cos(-inc)*R_temp1(3,:) - sin(-inc)*R_temp1(2,:);

V_temp2(1,:) = V_temp1(1,:);
V_temp2(2,:) = cos(-inc)*V_temp1(2,:) + sin(-inc)*V_temp1(3,:);
V_temp2(3,:) = cos(-inc)*V_temp1(3,:) - sin(-inc)*V_temp1(2,:);

%3rd rotation about vertical axis by -RAAN
```

```
R_ijk(1,:) = cos(-RAAN)*R_temp2(1,:) + sin(-RAAN)*R_temp2(2,:);
R_ijk(2,:) = cos(-RAAN)*R_temp2(2,:) - sin(-RAAN)*R_temp2(1,:);
R_ijk(3,:) = R_temp2(3,:);

V_ijk(1,:) = cos(-RAAN)*V_temp2(1,:) + sin(-RAAN)*V_temp2(2,:);
V_ijk(2,:) = cos(-RAAN)*V_temp2(2,:) - sin(-RAAN)*V_temp2(1,:);
V_ijk(3,:) = V_temp2(3,:);
```

F.1.1.4 Inner Loop PSO Algorithm

```
function [JGmin,Jpbest,gbest,x,k,JG,ex_flag] = PSO_REL_SHADOW_DV4(n,
    limits,prec,iter,swarm,chief_pos0,chief_vel0,chief_posf,chief_velf,
    dep_params,GEO_params,alphavec,betavec,t_zone,Pc,T_enter,T_exit,...
                                                                   r_cyl,T_prop
                                                                      ,GMST0,
                                                                      lat_site
                                                                      ,
                                                                      long_site
                                                                      ,t_step,
                                                                      el_val)



%Author: Dan Showalter 18 Oct 2012

%Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem

%generic PSO variable
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row
    n being
%       upper bound for element n
```

```
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO variables



%Specific Problem Variables



%%

[N,~] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end



gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
```

```
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);

llim2 = ones(n,swarm);
ulim2 = ones(n,swarm);

for aa = 1:n
    llim2(aa,:) = llim(aa)*llim2(aa,:);
    ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
end

d2 = ulim2 - llim2;

CoreNum = 12;
if (matlabpool('size'))<=0
    matlabpool('open','local',CoreNum);
else
    disp('Parallel Computing Enabled')
end
tstart = tic;
%loop until maximum iteration have been met

for k = 1:iter

    %create particles dictated by swarm size input


    % if this is the first iteration
    if k == 1
        rng('shuffle');
        x = random('unif',llim2,ulim2,[n,swarm]);
```

```
        v = random('unif',-d2,d2,[n,swarm]);

        %if this is after the first iteration, update velocity and
            position
        %of each particle in the swarm
    else
        parfor h = 1:swarm
            c1 = 2.09;
            c2 = 2.09;
            phi = c1+c2;
            ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
            cc = c1*random('unif',0,1);
            cs = c2*random('unif',0,1);


            vdum = v(:,h);
            %update velocity
            vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
                (:,h)));


            %check to make sure velocity doesn't exceed max velocity for
                 each
            %variable
            for w = 1:n

                %if the variable velocity is less than the min, set it
                    to the min
                if vdum(w) < -d(w)
                    vdum(w) = -d(w);
                    %if the variable velocity is more than the max, set
                        it to the max
```

```
                elseif vdum(w) > d(w);
                    vdum(w) = d(w);
                end
            end

            v(:,h) = vdum;

            %update position
            xdum = x(:,h) + v(:,h);

            for r = 1:n

                %if particle has passed lower limit
                if xdum(r) < llim(r)
                    xdum(r) = llim(r);

                elseif xdum(r) > ulim(r)
                    xdum(r) = ulim(r);
                end

                x(:,h) = xdum;

            end

        end

    end

    % round variables to get finite precision
    parfor aa = 1:n
        x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
        v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
```

```
    end

    %% ***********************Cost Function
        ************************************

    xmin = limits(2,1);
    %     rel_pos = zeros(3,length(T_prop));
    %     rel_pos_box = zeros(3,length(T_prop));
    %     temp1 = zeros(1,length(T_prop));
    %     temp2 = zeros(1,length(T_prop));
    %     temp3 = zeros(1,length(T_prop));
%           fclose('all');
%           fid=fopen('K-M.txt','w');
%           fprintf(fid,'\r\n\r\n\r\n\r\n%s %i','K',k);
%           fid2 = fopen('optvals.txt','w');
%           fprintf(fid2,'\r\n\r\n\r\n%s %i','K',k);
    parfor m = 1:swarm
        % ***************Cost function evaluation here
            ****************************
        opt_vars = x(:,m);
%           fid2=fopen('optvals.txt','a');
%           fprintf(fid2,'\r\n%i\t%6.2f %6.3f %6.3f %6.2f\t %6.3f %6.2f\t
    ',m,opt_vars(1),opt_vars(2),opt_vars(3),opt_vars(4),opt_vars(5),
    opt_vars(6));
%           fclose(fid2);
        [J(m)] = rel_shadow_cost_function2(opt_vars,chief_pos0,
            chief_vel0,chief_posf,chief_velf,alphavec,betavec,t_zone,Pc,
            T_prop,xmin,r_cyl,dep_params,...
                                                        GEO_params,T_enter,
                                                            T_exit,GMST0,
                                                            long_site,
```

```
                                                                  lat_site,t_step,
                                                                  el_val);
%
%
%             fid=fopen('K-M.txt','a')
%             fprintf(fid,'\r\n%s %i %8.5f','M complete',m,J(m));
%             fclose(fid);

    end

    %% ************************Constraint Equations
       ************************
    %%
       ***********************************************************************

    %%


    %round cost to nearest precision required
    J = round(J*10^prec(n+1))/10^prec(n+1);

    if k == 1
        count = 0;
        Jpbest(1:swarm) = J(1:swarm);
        pbest(:,1:swarm) = x(:,1:swarm);

        [Jgbest,IND] = min(Jpbest(:));

        gbest(:) = x(:,IND);

    else

```

```
        for h=1:swarm
            if J(h) < Jpbest(h)
                Jpbest(h) = J(h);
                pbest(:,h) = x(:,h);
                if Jpbest(h) < Jgbest

                    Jgbest = Jpbest(h);
                    gbest(:) = x(:,h);

                end
            end
        end
    end



    diff = zeros(swarm,1);
    parfor y = 1:swarm

        diff(y) = Jgbest - Jpbest(y);
    end

    indcount = find(abs(diff)<10^(-prec(n+1)));




    JG(k) = Jgbest;
    JGmin = Jgbest;

%     kinf = 50;
%     if k > kinf
```

```
%         if Jgbest == Inf
%             break
%         end
%     end

    if length(indcount) == swarm
        ex_flag = 0;
        break
    end

    if k > 1
        if JG(k) == JG(k-1)
            count = count + 1;
        else
%               MinCost = Jgbest*1000
%               k
            count = 0;
        end
    end

    if count > 1000
        ex_flag = 1;
        break
    end
end

if k == iter
    ex_flag = 2;
end
```

F.1.1.5 Cost Function Script

```matlab
function [J] = rel_shadow_cost_function2(opt_vars,chief_pos0,chief_vel0,chief_posf,chief_velf,alphavec,betavec,t_zone,Pc,T_prop,xmin,r_cyl,dep_params,...
                                          GEO_params,T_enter,T_exit,GMST0,long_site,lat_site,t_step,el_val)

OmegaEarth=0.000072921151467;
RE=6378.137;
%          variable definitions
%          nu0 = opt_vars(1);
%          TOF1 = opt_vars(2);
%          x_in = opt_vars(3);
%          x_out = opt_vars(4);
%          coast3 = opt_vars(5);
%          nu_GEO = opt_vars(6);
%          Tend = opt_vars(7);

alpha0 = alphavec(1);
beta0 = betavec(1);
alphaf = alphavec(end);
betaf = betavec(end);

a_dep = dep_params(1);
e_dep = dep_params(2);
i_dep = dep_params(3);
```

```
O_dep = dep_params(4);
o_dep = dep_params(5);


a_GEO = GEO_params(1);
e_GEO = GEO_params(2);
i_GEO = GEO_params(3);
O_GEO = GEO_params(4);
o_GEO = GEO_params(5);


%determine entry and exit conditions of deputy in cylinder frame
box_vec0 = zeros(3,1);
box_vec0(1) = opt_vars(3);
box_vec0(2) = 0;
box_vec0(3) = 0;

[deputy_pos0,rel_pos0] = box2cw(chief_pos0,chief_vel0,box_vec0,alpha0,
    beta0);

box_vecf = zeros(3,1);
box_vecf(1) = opt_vars(4);
box_vecf(2) = 0;
box_vecf(3) = 0;

[deputy_posf,rel_posf] = box2cw(chief_posf,chief_velf,box_vecf,alphaf,
    betaf);

%determine required entry/exit velocities corresponding to entry/exit
    conditions
[v0_tilde,vf_tilde] = relative_velocity(t_zone,Pc,rel_pos0,rel_posf);

```

```
%propogate (discretely) relative motion for time chief in contact with
%ground site
[rel_pos] = CW_Motion3(rel_pos0,v0_tilde,T_prop',Pc);

%convert relative position from cw to cylinder frame
temp1 = cos(betavec).*rel_pos(1,:) + sin(betavec).*rel_pos(2,:);
temp2 = cos(betavec).*rel_pos(2,:) - sin(betavec).*rel_pos(1,:);
temp3 = rel_pos(3,:);

rel_pos_box = zeros(3,length(rel_pos));

rel_pos_box(1,:) = cos(-alphavec).*temp1 - sin(-alphavec).*temp3;
rel_pos_box(2,:) = temp2;
rel_pos_box(3,:) = cos(-alphavec).*temp3 + sin(-alphavec).*temp1;

[T_out] = out_of_cylinder(rel_pos_box,T_prop',xmin,r_cyl);

%propogate (discretely) relative motion for post inspection motion to
%ensure chaser doesn't intercept chief
T_post_ci = [(0:opt_vars(7)) opt_vars(7)];

[rel_pos2] = CW_Motion3(rel_posf,vf_tilde,T_post_ci,Pc);
rel_min_vec = sqrt(rel_pos2(1,:).*rel_pos2(1,:) + rel_pos2(2,:).*
    rel_pos2(2,:) + rel_pos2(3,:).*rel_pos2(3,:));

%closest approach must be more than 50 meters away
min_approach = min(rel_min_vec);

if T_out > 0 || min_approach < 0.05
    J = Inf;
else

```

```
v0_rel = v0_tilde/Pc;
[~,~,ic1,Oc1,~,nuc01] = RV2COE(chief_pos0,chief_vel0);
v0_arrive = chief_vel0 + rot3mat(-Oc1)*rot1mat(-ic1)*rot3mat(-nuc01)
    *v0_rel;

%              %determine departure location of maneuvering satellite
nu_dep = opt_vars(1);
[r0_d,v0_d] = COE2RV(a_dep,e_dep,i_dep,O_dep,o_dep,nu_dep);

%% solve lambert's problem both ways to get from satellite to lobe
    entry condition
[V1S, V2S] = lambert2(r0_d',deputy_pos0',( opt_vars(2))/(3600*24)
    ,0,398600.5);

[V1L, V2L] = lambert2(r0_d',deputy_pos0',-(opt_vars(2))/(3600*24)
    ,0,398600.5);

%Depature DV
DV1S = V1S - v0_d';
DV1L = V1L - v0_d';

%arrival DV
DV2S = v0_arrive' - V2S;
DV2L = v0_arrive' - V2L;

DV_shadeS = norm(DV1S) + norm(DV2S);
DV_shadeL = norm(DV1L) + norm(DV2L);

if DV_shadeS < DV_shadeL
    DV = DV_shadeS;
    V12_d = V1S';
    DV_depart1 = DV1S';
```

```
        DV_arrive1 = DV2S';
    else
        DV = DV_shadeL;
        V12_d = V1L';
        DV_depart1 = DV1L';
        DV_arrive1 = DV2L';
    end

    %determine ground site inertial position vectors for duration of
       second maneuver
    %(coast0 to Tenter)
    Tvec12 = (T_enter-opt_vars(2):t_step:T_enter)';
    GMST12 = GMST0*ones(length(Tvec12),1) + OmegaEarth.*Tvec12;
    longvec12 = long_site*ones(length(Tvec12),1) + GMST12;



    %inertial coordinates of the ground site
    Rsite12 = zeros(3,length(Tvec12));
    Rsite12(1,:) = RE*cos(lat_site).*cos(longvec12);
    Rsite12(2,:) = RE*cos(lat_site).*sin(longvec12);
    Rsite12(3,:) = RE*sin(lat_site);



    %determine maneuvering spacecraft inertial position vectors for
       duration od
    %second maneuver
    Tvec12m = Tvec12 - Tvec12(1);


    [am12,em12,im12,Om12,om12,num012] = RV2COE(r0_d,V12_d);
```

```
    if imag(num012) < 1e-6
        num012 = real(num012);
    else
        fid = fopen('error_data.txt','a');
        [~,ind_imag] = max(imag(num012));
        fprintf(fid,'%s %s','Real','Imag');
        fprintf(fid,'\n\r%10.8f %10.8f',real(num012(ind_imag)),imag(num012(ind_imag)));
    end



    [val] = site_contact_vec(am12,im12,em12,Om12,om12,num012,long_site,GMST12(1),Tvec12m(end),t_step,lat_site,Rsite12,el_val);

%      T_out2 = length(val)/length(Tvec12m)*Tvec12m(end);

    if isempty(val) == 0
        J = Inf;
    else

        %%  3rd and 4th Maneuver

        vf_rel = vf_tilde/Pc;
        [~,~,ic2,Oc2,~,nuc02] = RV2COE(chief_posf,chief_velf);
        vf_depart = chief_velf + rot3mat(-Oc2)*rot1mat(-ic2)*rot3mat(-nuc02)*vf_rel;

        %determine orbital parameters of satellite upon zone exit
        [at,et,it,Ot,ot,nut0] = RV2COE(deputy_posf,vf_depart);
```

```
        nu_tL = nuf_from_TOF(nut0,opt_vars(5),at,et);
        [r_tL,V_tL] = COE2RV(at,et,it,Ot,ot,nu_tL);


        %determine arrival location of maneuvering satellite
        [r_GEO,V_GEO] = COE2RV(a_GEO,e_GEO,i_GEO,O_GEO,o_GEO,opt_vars(6)
            );

        %% solve lambert's problem both ways to get from lobe exit
            condition to GEO
        [V3S, V4S] = lambert2(r_tL',r_GEO',(opt_vars(7))/(3600*24)
            ,0,398600.5);

        [V3L, V4L] = lambert2(r_tL',r_GEO',-(opt_vars(7))/(3600*24)
            ,0,398600.5);

        %Depature DV
        DV3S = V3S - V_tL';
        DV3L = V3L - V_tL';

        %arrival DV
        DV4S = V_GEO' - V4S;
        DV4L = V_GEO' - V4L;

        DV_GEOS = norm(DV3S) + norm(DV4S);
        DV_GEOL = norm(DV3L) + norm(DV4L);

        if DV_GEOS < DV_GEOL
            V_mL = V3S';
            DV2 = DV_GEOS;
            DV_depart2 = DV3S';
            DV_arrive2 = DV4S';
```

```
        else
            V_mL = V3L';
            DV2 = DV_GEOL;
            DV_depart2 = DV3L';
            DV_arrive2 = DV4L';
        end

        %determine ground site inertial position vectors for duration of
             second maneuver
        %(Texit + coastf) to Tend
        Tvec2 = (T_exit+opt_vars(5):t_step:T_exit+opt_vars(5)+opt_vars
            (7))';
        GMST = GMST0*ones(length(Tvec2),1) + OmegaEarth.*Tvec2;
        longvec = long_site*ones(length(Tvec2),1) + GMST;



        %inertial coordinates of the ground site
        Rsite = zeros(3,length(Tvec2));
        Rsite(1,:) = RE*cos(lat_site).*cos(longvec);
        Rsite(2,:) = RE*cos(lat_site).*sin(longvec);
        Rsite(3,:) = RE*sin(lat_site);

        %determine maneuvering spacecraft inertial position vectors for
            duration od
        %second maneuver (Texit + coastf) to Tend
        Tvec3 = Tvec2 - T_exit - opt_vars(5);

        [am,em,im,Om,om,num0] = RV2COE(r_tL,V_mL);

        if imag(num0) < 1e-6
            num0 = real(num0);
```

```
        else
            fid = fopen('error_data.txt','a');
            [~,ind_imag0] = max(imag(num0));
            fprintf(fid,'%s %s','Real','Imag');
            fprintf(fid,'\n\r%10.8f %10.8f',real(num0(ind_imag0)),imag(num0(ind_imag0)));
        end


        [val2] = site_contact_vec(am,im,em,Om,om,num0,long_site,GMST(1),Tvec3(end),t_step,lat_site,Rsite,el_val);


        if isempty(val2) == 0
            J = Inf;
        else
            J = DV + DV2;
        end
    end
end
```

F.1.1.6 Convert RSW Coordinates to Cylinder Frame

```
function [deputy_pos,rel_pos] = box2cw(chief_pos,chief_vel,box_vec,alpha,beta)
%this function converts from safe zone coordinate frame to the cw
%coordinate frame

%INPUTS
% box_vec - (3x1) vector defining a coordinate in the box frame (km)
% alpha - rotation angle between fundamental plane in box frame and
% fundamental plane in cw frame (rad)
% beta - rotation angle between principal axis in cw frame and box frame
```

```
rel_pos = rot3mat(-beta)*rot2mat(alpha)*box_vec;

xhat = chief_pos/norm(chief_pos);
yhat = chief_vel/norm(chief_vel);
hvec = cross(chief_pos,chief_vel);
zhat = hvec/norm(hvec);

[~,~,ic,Oc,~,nuc0] = RV2COE(chief_pos,chief_vel);

deputy_pos = chief_pos +  rot3mat(-Oc)*rot1mat(-ic)*rot3mat(-nuc0)*
    rel_pos;
```

F.1.1.7 Determine Initial and Final Velocities for Inspection Segment

```
function [v0_tilde,vf_tilde] = relative_velocity(T,P,pos0,posf)
%relative velocity returns the required initial and final relative
%velocities to get the deputy satellite from the relative position pos0
%to the relative position posf (relative to the chief satellite) in T/P
    time units

%INPUTS
% T = actual time of trajectory (sec)
% P = period of the chief satellite (sec)
% pos0 = relative position vector (3x1) of lobe entry point (km)
% posf = relative position vector (3x1) of lobe exit point (km)

%OUTPUTS
% v0_tilde = time scaled relative velocity vector (3x1) at pos0
% vf_tilde = time scaled relative velocity vector (3x1) at posf

%%

```

```
x0 = pos0(1);
y0 = pos0(2);
z0 = pos0(3);
xf = posf(1);
yf = posf(2);
zf = posf(3);

T_tilde = T/P;
S_tilde = sin(2*pi*T_tilde);
C_tilde = cos(2*pi*T_tilde);
delta_y = yf - y0;


%Initialize A Matrices to determine relative velocities at entry and
%arrival locations
A0 = zeros(3,5);
Af = zeros(3,5);

%A Matrix at lobe entry
A0(1,1) = (6*pi*T_tilde*C_tilde - 4*S_tilde)/(8 - 6*pi*T_tilde*S_tilde -
     8*C_tilde);
A0(1,3) = (4*S_tilde - 6*pi*T_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*
    C_tilde);
A0(1,5) = (2*C_tilde - 2)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
A0(2,1) = (-14 + 12*pi*T_tilde*S_tilde + 14*C_tilde)/(8 - 6*pi*T_tilde*
    S_tilde - 8*C_tilde);
A0(2,3) = (2 - 2*C_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
A0(2,5) = (S_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
A0(3,2) = -C_tilde/S_tilde;
A0(3,4) = 1/S_tilde;

A0 = 2*pi*A0;
```

```
%A Matrix at lobe exit
Af(1,1) = (-4*S_tilde + 6*pi*T_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*
    C_tilde);
Af(1,3) = (4*S_tilde - 6*pi*T_tilde*C_tilde)/(8 - 6*pi*T_tilde*S_tilde -
     8*C_tilde);
Af(1,5) = (2-2*C_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
Af(2,1) = (2 - 2*C_tilde)/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
Af(2,3) = (-14 + 12*pi*T_tilde*S_tilde + 14*C_tilde)/(8 - 6*pi*T_tilde*
    S_tilde - 8*C_tilde);
Af(2,5) = S_tilde/(8 - 6*pi*T_tilde*S_tilde - 8*C_tilde);
Af(3,2) = -1/S_tilde;
Af(3,4) = C_tilde/S_tilde;

Af = 2*pi*Af;

state_vec = [x0;z0;xf;zf;delta_y];

v0_tilde = A0*state_vec;
vf_tilde = Af*state_vec;
```

F.1.1.8 Propagate Motion of Chaser For Relative Inspection Phase

```
function [rel_pos] = CW_Motion3(deputy_rel0,v0_tilde,Tvec,P)
%CW Motion determines the position of a deputy satellite in a relative
%frame cenetred on a chief satellite given an initial relative position,
%velocity, the actual time of the motion and period of the chief
    satellite

%INPUTS
% deputy_rel0 = position vector (3x1) of deputy satellite (km)
% v0_tilde = velocity vector (3x1) of deputy satellite
% T = actual time of motion (sec)
```

```
% P = period of chief satellite (sec)



%OUTPUTS
% rel_pos = relative position vector (3xlength(Tvec)) of deputy (km)
% T_out = Amount of time spent outside of ellipse (sec/P)
%%


Tvec = Tvec/P;
Tmat1(1,:) = sin(2*pi*Tvec);
Tmat1(2,:) = cos(2*pi*Tvec);
Tmat1(3,:) = 1;

Tmat2(1,:) = sin(2*pi*Tvec);
Tmat2(2,:) = cos(2*pi*Tvec);
Tmat2(3,:) = (- 1/pi*v0_tilde(1) + deputy_rel0(2)).*Tmat1(3,:);
Tmat2(3,:) = Tmat2(3,:) - (3*v0_tilde(2) + 12*pi*deputy_rel0(1))*Tvec;

xvals = [1/(2*pi)*v0_tilde(1),-(1/pi*v0_tilde(2) + 3*deputy_rel0(1)),1/
    pi*v0_tilde(2) + 4*deputy_rel0(1)];
yvals = [(2/pi*v0_tilde(2) + 6*deputy_rel0(1)),1/pi*v0_tilde(1),1];
zvals = [1/(2*pi)*v0_tilde(3),deputy_rel0(3),0];

xpos = xvals*Tmat1;
ypos = yvals*Tmat2;
zpos = zvals*Tmat1;

rel_pos(1,:) = xpos;
rel_pos(2,:) = ypos;
rel_pos(3,:) = zpos;
```

F.1.1.9 Determine if Chaser Exits Cylinder

```
function [T_out,T_in,pos_out,pos_in,time_out] = out_of_cylinder(rel_pos,
    Tvec,xmin,r_cyl)

%% Determine if satellite leaves safe zone
time_out = zeros(length(Tvec),1);

r_vec = sqrt(rel_pos(2,:).^2 + rel_pos(3,:).^2);

%set of indices where deputy is less than xmin
ind_ex_xmin = find(rel_pos(1,:) < xmin);
time_out(ind_ex_xmin(:)) = 1;

%set of indices where deputy is greater than ymax
ind_ex_cyl = find(r_vec(:) > r_cyl);
time_out(ind_ex_cyl(:)) = 1;

ind_out = find(time_out > 0);
ind_in = find(time_out == 0);

pos_out = rel_pos(:,ind_out(:));
pos_in = rel_pos(:,ind_in(:));

T_out = length(ind_out)/length(time_out)*Tvec(end);
T_in = length(ind_in)/length(time_out)*Tvec(end);
```

F.1.1.10 Determine Maneuver Path is in Sight of Ground Site

```
function [val,rho_sez] = site_contact_vec(a,inc,ecc,Omega,omega,nu0,
    lambda0,GMST0,tmax,tstep,lat_site,rgs,el_val)

%INPUTS
% a = satellite semimajor axis (km)
```

```
% inc = satellite inclination (rad)
% ecc = satellite eccentricity
% Omega = satellite RAAN (rad)
% omega = satellite argument of perigee (rad)
% nu0 = initial true anomaly (rad)
% lambda0 = initial GMST of ground site
% tmax = maximum scenario time (sec)
% tstep = time step (sec)

%OUTPUTS
% C_times = times satellite is in contact with the ground site
% C_ind = indices of satellite contat times
% rijk = position vectors of satellite at discretized times
% vijk = velocity vectors of satellite at discretized times
% rgs = position vectors of the ground site at discretized times
% rho_sez = vector from ground site to satellite in SEZ coordinates
% rho_RIC = vector from ground site to satellite in RIC coordinates
%
    =========================================================================

tvec = (0:tstep:tmax)';

% determine true anomaly of spacecraft at each time step
[nu_vec] = nuf_from_TOF_vec(nu0,tvec,a,ecc);

if ecc == 1
    keyboard
end

%determine inertial position and velocity vectors at each tiem step
[rijk] = COE2RV_vec(a,ecc,inc,Omega,omega,nu_vec);

```

```
if size(rijk) ~= size(rgs)
    keyboard
end
wgs84data

global OmegaEarth

long_site = lambda0 + GMST0 + OmegaEarth*tvec;

%vector from ground site to satellite
rho_ijk = rijk - rgs;

%transform into sez coordinates
temp = zeros(3,length(tvec));
temp(1,:) = cos(long_site').*rho_ijk(1,:) + sin(long_site').*rho_ijk
    (2,:);
temp(2,:) = -sin(long_site').*rho_ijk(1,:) + cos(long_site').*rho_ijk
    (2,:);
temp(3,:) = rho_ijk(3,:);

rho_sez = zeros(3,length(tvec));
rho_mag = zeros(1,length(tvec));

rho_sez(1,:) = cos(pi/2 - lat_site)*temp(1,:) - sin(pi/2 - lat_site)*
    temp(3,:);
rho_sez(2,:) = temp(2,:);
rho_sez(3,:) = sin(pi/2 - lat_site)*temp(1,:) + cos(pi/2 - lat_site)*
    temp(3,:);

rho_mag = sqrt(rho_sez(1,:).^2 + rho_sez(2,:).^2 + rho_sez(3,:).^2);

```

```
val = find(asind(rho_sez(3,:)./rho_mag) > el_val);
```

F.1.2 PSO Driver Script

```
%%

%maximum number of entries into exclusion zone before maneuver is
%required
close all
% clear all
clc

el_val_pass = 0;
el_val_shadow = 1;
GMST0 = 0;
lat_site = pi/4;
long_site = 0;
t0 = 0;
tf_max = 36*3600;
tstep = 1;
r_cyl = 1;
xmin = 1;
xmax = 3;

%% Determine Chief Satellite Entry/Exit over Exclusion Zone

%Initial COEs of chief satellite
a_chief_vec = [26581.76 7378 6878];
e_chief = 0;
i_chief = 55*pi/180;
O_chief = 0;
o_chief = 0;
% nu_chief_vec = 0;
```

```
nu_chief = 0;

chief_params = [e_chief;i_chief;O_chief;o_chief;nu_chief];
%Initial COEs of deputy satellite
a_dep = 6578;
e_dep = 0;
i_dep = 55*pi/180;
O_dep = 0;
o_dep = 0;
nu_dep0 = 0;

dep_params = [a_dep;e_dep;i_dep;O_dep;o_dep];

a_GEO = 42164.14;
e_GEO = 0;
i_GEO = 0;
O_GEO = 0;
o_GEO = 0;

GEO_params = [a_GEO;e_GEO;i_GEO;O_GEO;o_GEO];

swarm = 15;
iter = 10;
prec = [0;0;6];
%if kinf ~= 0, inner loop PSO assigned inifinite cost to categorical
%variables if inner loop PSO has infinite cost after kinf iterations
kinf = 50;

load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
    Article_Data\Rev2\ThreePassEnumData');

for aa = 1:3
```

```
    C_times_c = ThreePassEnumData(aa).times;
    [max_ind(aa),~] = size(C_times_c);

end
maxP = max(max_ind);

for bb = 16:30

    if bb == 1 || bb == 0
        total_repPSOinf = zeros(length(a_chief_vec),maxP);
        save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\Article_Data\Rev2\total_repPSOinf.mat','total_repPSOinf');
    end

    tstart = tic;


    [JGmin,Jpbest,gbest_tot,x,k,k_tot,JG,rep_mat,pop_mat] = PSO_MULTISAT_COOP_WRAPPER(2,[1 length(a_chief_vec);1 maxP],prec,iter,swarm,GMST0,lat_site,long_site,tstep,a_chief_vec,dep_params,GEO_params,xmin,xmax,r_cyl,...
        el_val_shadow,max_ind,ThreePassEnumData,kinf);



    tend = toc(tstart);

    load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\Article_Data\Rev2\total_repPSOinf');
    fid = fopen('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\Article_Data\Rev2\ThreePassHybridPSODataInf.txt','a');
```

```
    fprintf(fid,'%i\t\t%2i\t\t%5i\t\t%3.2f\t\t%5i\t\t%4.3f\t\t%4.3f\t\t
       %5i\t%3.2f\t\t%5i\t\t%6.5f\t\t%2i\t\t%6.2f\r\n',...
        bb,gbest_tot(1),gbest_tot(2),gbest_tot(3),gbest_tot(4),gbest_tot
            (5),gbest_tot(6),gbest_tot(7),gbest_tot(8),gbest_tot(9),
            JGmin,k_tot,tend);

    for ee = 1:length(a_chief_vec)
        for ff = 1:maxP
            Jrep = rep_mat(ee,ff);
            Jtot = total_repPSOinf(ee,ff);
            if Jrep < Jtot || Jtot == 0
                if Jtot ~= Inf
                total_repPSOinf(ee,ff) = Jrep;
                end
            end
        end
    end



    save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
       Article_Data\Rev2\total_repPSOinf.mat','total_repPSOinf');
    clear total_repPSOinf
end
```

F.1.2.1 Outer Loop PSO

```
function [JGmin,Jpbest,gbest_tot,x,k,k_tot,JG,rep_mat,pop_mat] =
    PSO_MULTISAT_COOP_WRAPPER(n,limits,prec,iter,swarm,GMST0,lat_site,
    long_site,t_step,a_chief_vec,dep_params,GEO_params,xmin,xmax,r_cyl
    ,...
    el_val_shadow,max_ind,DataStruct,kinf)

```

```
%Author: Dan Showalter 23 Sep 2013

%Purpose: PSO inside of a PSO

%generic PSO inputs
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row
    n being
%       upper bound for element n
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO inputs
% GMST0 = initial Greenwich mean standard time (rad)
% lat_site = ground site latitude
% long_site = ground site longitude
% chief_params = vector (1x5) of fixed orbital elements of chief
    satellite
% nu_chief_vec = vector of potential initial true anomalies for chief
% dep_params = vector (1x6) of initial orbital elements of deputy
    satellite
% GEO_params = vector of (1x5) of fixed orbital elements of GEO
    satellite
% Coast_time_d = matrix (2xm) of allowed maneuver windows
%                (1,m) = start time of mth window
%                (2,m) = end time of mth window
% tf_max = maximum scenario time (sec)
% tstep = discrete time step (sec)
% xmin = minimum x distance from deputy to satellite in CW frame (km)
```

```
% xmax = maximum x distance from deputy to satellite in CW frame (km)
% Pc = period of chief satellite (sec)
% r_cyl = cylinder radius (km)
%
    =========================================================================================

%%

[N,~] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end

gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
x_inside = zeros(7,swarm);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);
rep_mat = zeros(ulim(1),ulim(2));
pop_mat = struct('pop',zeros(n,swarm),'J',zeros(swarm,1));

llim2 = ones(n,swarm);
```

```
ulim2 = ones(n,swarm);
% CoreNum = 12;
% if (matlabpool('size'))<=0
%     matlabpool('open','local',CoreNum);
% else
%     disp('Parallel Computing Enabled')
% end

parfor aa = 1:n
    llim2(aa,:) = llim(aa)*llim2(aa,:);
    ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
end

d2 = ulim2 - llim2;


xrep(ulim(1),max_ind) = struct('xinsidevals',zeros(1,7));
%loop until maximum iteration have been met
for k = 1:iter
    t_inside = tic;
    %create particles dictated by swarm size input


    % if this is the first iteration
    if k == 1
        x = unidrnd(ulim2);
        v = random('unif',-d2,d2,[n,swarm]);

        %if this is after the first iteration, update velocity and
            position
        %of each particle in the swarm
    else
```

```
        for h = 1:swarm

            c1 = 2.09;
            c2 = 2.09;
            phi = c1+c2;
            ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));

            cc = c1*random('unif',0,1);
            cs = c2*random('unif',0,1);


            vdum = v(:,h);

            %update velocity
            %              vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) +
                cs*(gbest - x(:,h)));

            vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x
                (:,h)));
            %check to make sure velocity doesn't exceed max velocity for
                 each
            %variable
            for w = 1:n

                %if the variable velocity is less than the min, set it
                    to the min
                if vdum(w) < -d(w)
                    vdum(w) = -d(w);
                    %if the variable velocity is more than the max, set
                        it to the max
                elseif vdum(w) > d(w);
```

```
                vdum(w) = d(w);
            end
        end

        v(:,h) = vdum;

        %update position
        xdum = x(:,h) + v(:,h);

        for r = 1:n

            %if particle has passed lower limit
            if xdum(r) < llim(r)
                xdum(r) = llim(r);

            elseif xdum(r) > ulim(r)
                xdum(r) = ulim(r);
            end

            x(:,h) = xdum;

        end

    end

end

% round variables to get finite precision
for aa = 1:n
    x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
    v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
end
```

```
pop_mat(k).pop = x;


%% ***********************Cost Function
    ************************************

for m = 1:swarm
    MU = 398600.5;

    % ****************Cost function evaluation here
        ****************************
    opt_vars = x(:,m);
    %          variable definitions
    satellite = opt_vars(1);
    min_ind = opt_vars(2);

    C_times_c = DataStruct(satellite).times;
    C_ind_c = DataStruct(satellite).ind;
    Rijk_c = DataStruct(satellite).Rc;
    Vijk_c = DataStruct(satellite).Vc;
    rho_vec_cw_c = DataStruct(satellite).rho_c;
    Tvec_c = DataStruct(satellite).Tc;
    max_ind = DataStruct(satellite).max_ind;

    if min_ind > max_ind
        J(m) = Inf;
    else
%             if rep_mat(satellite,min_ind) == Inf
%                 J(m) = Inf;
        if rep_mat(satellite,min_ind) ~= 0
            J(m) = rep_mat(satellite,min_ind);
```

```
            x_inside(:,m) = xrep(satellite,min_ind).xinsidevals;
        else



            %Period of Chief satellite's orbit
            Pc = 2*pi*sqrt(a_chief_vec(satellite)^3/MU);


            t_enter = C_times_c(min_ind,1);
            t_exit = C_times_c(min_ind,2);
            t_zone = t_exit - t_enter;

            %determine indices of minimum duration contact
            C_ind_contact = C_ind_c(min_ind,:);

            %find unit vector pointing towards the deputy that puts
                chief between
            %ground site and deputy
            rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):
                C_ind_contact(2));

            %determine alpha and beta angles during contact times
            [alphavec,betavec] = alphabeta(rho_unit_cw);

            %Vector of times for propogation
            T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) -
                Tvec_c(C_ind_contact(1))*ones(length(Tvec_c(
                C_ind_contact(1):C_ind_contact(2))),1);

            %Determine position/velocity vectors of chief satellite
                upon intial/final
```

```
%contact
chief_pos0 = Rijk_c(:,C_ind_c(min_ind,1));
chief_vel0 = Vijk_c(:,C_ind_c(min_ind,1));
chief_posf = Rijk_c(:,C_ind_c(min_ind,2));
chief_velf = Vijk_c(:,C_ind_c(min_ind,2));

if min_ind < max_ind
    max_coastf = C_times_c(min_ind+1,1) - C_times_c(
        min_ind,2);
else
    max_coastf = DataStruct(satellite).max_coastf;
end

%time variables have precision to .1 second.  Others
    have
%precision to 0.001 units (km,rad)
prec2 = [2;0;3;3;0;2;0;6];

x(:,m);

[J(m),~,x_inside_dum,~,k_inside,~] =
    PSO_REL_SHADOW_DV4inf(7,[0 2*pi;1 C_times_c(min_ind
    ,1);xmin xmax;xmin xmax;1 max_coastf;0 2*pi;1
    16*3600],prec2,500,300,chief_pos0,chief_vel0,
    chief_posf,...
    chief_velf,dep_params,GEO_params,alphavec,betavec,
        t_zone,Pc,t_enter,t_exit,r_cyl,T_prop,GMST0,
        lat_site,long_site,t_step,el_val_shadow,kinf);
if k == 1 || rep_mat(satellite,min_ind) == 0
    rep_mat(satellite,min_ind) = J(m);
    xrep(satellite,min_ind).xinsidevals = x_inside_dum;
else
```

```
                    if J(m) < rep_mat(satellite,min_ind)
                        rep_mat(satellite,min_ind) = J(m);
                    end
                end
                J(m);
                x_inside(:,m) = x_inside_dum;
                out_loop = m;
                if k == 1
                    k_tot = k_inside;
                else
                    k_tot = k_inside + k_tot;
                end
            end
        end
    end



    [minJ,ind_minJ] = min(J);
    x_inside(:,ind_minJ)
    %% **************************Constraint Equations **************************
    %% *************************************************************************

    %%


    if k == 1

        Jpbest(1:swarm) = J(1:swarm);
        pbest(:,1:swarm) = x(:,1:swarm);
```

```
        [Jgbest,IND] = min(Jpbest(:));

        gbest(:) = x(:,IND);
        g_inside_best = x_inside(:,IND);

    else
        parfor h=1:swarm
            if J(h) < Jpbest(h)
                Jpbest(h) = J(h);
                pbest(:,h) = x(:,h);
            end
        end

        [Jit_min,min_ind] = min(Jpbest);
        if Jit_min < Jgbest

            Jgbest = Jpbest(min_ind);
            gbest(:) = x(:,min_ind);
            g_inside_best = x_inside(:,min_ind);

        end
    end



    %round cost to nearest precision required
    J = round(J*10^prec(n+1))/10^prec(n+1);
    pop_mat(k).J = J;

    JG(k) = Jgbest;
    JGmin = Jgbest;
```

```
    iter_complete = k
    iter_time = toc(t_inside)
    format long g
    gbest
    g_inside_best
    JGmin
end

gbest_tot(1:n) = gbest;
gbest_tot(n+1:n+length(g_inside_best)) = g_inside_best;
```

F.1.2.2 Inner Loop PSO Algorithm with Infeasible Cutoff

```
function [JGmin,Jpbest,gbest,x,k,JG,ex_flag] = PSO_REL_SHADOW_DV4inf(n,
    limits,prec,iter,swarm,chief_pos0,chief_vel0,chief_posf,chief_velf,
    dep_params,GEO_params,alphavec,betavec,t_zone,Pc,T_enter,T_exit,...
                                                            r_cyl,T_prop
                                                            ,GMST0,
                                                            lat_site
                                                            ,
                                                            long_site
                                                            ,t_step,
                                                            el_val,
                                                            kinf)



%Author: Dan Showalter 18 Oct 2012

%Purpose: Utilize PSO to solve multi-orbit sinegle burn maneuver problem

%generic PSO variable
```

```
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row
    n being
%       upper bound for element n
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO variables



%Specific Problem Variables



%%

[N,~] = size(limits);

llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end



```

```
gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);

llim2 = ones(n,swarm);
ulim2 = ones(n,swarm);

for aa = 1:n
    llim2(aa,:) = llim(aa)*llim2(aa,:);
    ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
end

d2 = ulim2 - llim2;

CoreNum = 12;
if (matlabpool('size'))<=0
    matlabpool('open','local',CoreNum);
else
    disp('Parallel Computing Enabled')
end
tstart = tic;
%loop until maximum iteration have been met

for k = 1:iter

    %create particles dictated by swarm size input

```

```
    % if this is the first iteration
    if k == 1
        rng('shuffle');
        x = random('unif',llim2,ulim2,[n,swarm]);
        v = random('unif',-d2,d2,[n,swarm]);

        %if this is after the first iteration, update velocity and position
        %of each particle in the swarm
    else
        parfor h = 1:swarm
            c1 = 2.09;
            c2 = 2.09;
            phi = c1+c2;
            ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));
            cc = c1*random('unif',0,1);
            cs = c2*random('unif',0,1);


            vdum = v(:,h);
            %update velocity
            vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x(:,h)));


            %check to make sure velocity doesn't exceed max velocity for each
            %variable
            for w = 1:n

```

```
            %if the variable velocity is less than the min, set it to the min
            if vdum(w) < -d(w)
                vdum(w) = -d(w);
                %if the variable velocity is more than the max, set it to the max
            elseif vdum(w) > d(w);
                vdum(w) = d(w);
            end
        end

        v(:,h) = vdum;

        %update position
        xdum = x(:,h) + v(:,h);

        for r = 1:n

                %if particle has passed lower limit
                if xdum(r) < llim(r)
                    xdum(r) = llim(r);

                elseif xdum(r) > ulim(r)
                    xdum(r) = ulim(r);
                end

                x(:,h) = xdum;

        end

    end

```

```
    end

    % round variables to get finite precision
    parfor aa = 1:n
        x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
        v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
    end

    %% ***********************Cost Function ************************************

    xmin = limits(2,1);

    parfor m = 1:swarm
        % ****************Cost function evaluation here ****************************
        opt_vars = x(:,m);

        [J(m)] = rel_shadow_cost_function2(opt_vars,chief_pos0,chief_vel0,chief_posf,chief_velf,alphavec,betavec,t_zone,Pc,T_prop,xmin,r_cyl,dep_params,...
                                                        GEO_params,T_enter,T_exit,GMST0,long_site,lat_site,t_step,el_val);

    end

    %% **************************Constraint Equations **************************
```

```
%%
    ************************************************************************

%%


%round cost to nearest precision required
J = round(J*10^prec(n+1))/10^prec(n+1);

if k == 1
    count = 0;
    Jpbest(1:swarm) = J(1:swarm);
    pbest(:,1:swarm) = x(:,1:swarm);

    [Jgbest,IND] = min(Jpbest(:));

    gbest(:) = x(:,IND);

else

    for h=1:swarm
        if J(h) < Jpbest(h)
            Jpbest(h) = J(h);
            pbest(:,h) = x(:,h);
            if Jpbest(h) < Jgbest

                Jgbest = Jpbest(h);
                gbest(:) = x(:,h);

            end
        end
    end
```

```
    end



    diff = zeros(swarm,1);
    parfor y = 1:swarm

        diff(y) = Jgbest - Jpbest(y);
    end

    indcount = find(abs(diff)<10^(-prec(n+1)));




    JG(k) = Jgbest;
    JGmin = Jgbest;

    if kinf ~= 0;
        if k > kinf
            if Jgbest == Inf
                break
            end
        end
    end

    if length(indcount) == swarm
        ex_flag = 0;
        break
    end

    if k > 1
```

```
        if JG(k) == JG(k-1)
            count = count + 1;
        else
            count = 0;
        end
    end

    if count > 1000
        ex_flag = 1;
        break
    end
end

if k == iter
    ex_flag = 2;
end
```

F.1.3 GA Driver Script

```
clc
close all

for h =1:10


    el_val_pass = 0;
    el_val_shadow = 1;
    GMST0 = 0;
    lat_site = pi/4;
    long_site = 0;
    t0 = 0;
    tf_max = 36*3600;
    tstep = 1;
```

```
r_cyl = 1;
xmin = 1;
xmax = 3;
%% Determine Chief Satellite Entry/Exit over Exclusion Zone

%Initial COEs of chief satellite
a_chief = [26581.76 7378 6878];
e_chief = 0;
i_chief = 55*pi/180;
O_chief = 0;
o_chief = 0;
% nu_chief_vec = 0;
nu_chief_vec = [0 90 180 270]*pi/180;

chief_params = [e_chief;i_chief;O_chief;o_chief;nu_chief_vec(1)];
%Initial COEs of deputy satellite
a_dep = 6578;
e_dep = 0;
i_dep = 55*pi/180;
O_dep = 0;
o_dep = 0;
nu_dep0 = 0;

dep_params = [a_dep;e_dep;i_dep;O_dep;o_dep];

a_GEO = 42164.14;
e_GEO = 0;
i_GEO = 0;
O_GEO = 0;
o_GEO = 0;

GEO_params = [a_GEO;e_GEO;i_GEO;O_GEO;o_GEO];
```

```
    load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
        Article_Data\Rev3\ThreeTarget\ThreePassEnumData');

    kinf = 50;

for aa = 1:3
    C_times_c = ThreePassEnumData(aa).times;
    [max_ind(aa),~] = size(C_times_c);
    clear C_times_c
end
    maxP = max(max_ind);

    if h == 1
        total_repGAinf = zeros(3,maxP);
        save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative
            Motion\Article_Data\Rev3\ThreeTarget\total_repGAinf.mat','
            total_repGAinf');
    end

    llim = [1 1];
    ulim = [length(a_chief) maxP];

    PopSize = 15;
    ulim2 = zeros(PopSize,2);
    ulim2(:,1) = ulim(1);
    ulim2(:,2) = ulim(2);
    EliteSize = 1;

    rep_mat = zeros(length(a_chief),maxP);
    fid3 = fopen('GA_Jmin.txt','w');
    fprintf(fid3,'%f',10000);
```

```
    fclose(fid3);

    fid4 = fopen('GA_iters.txt','w');
    fprintf(fid4,'%i',0);
    fclose(fid4);

    fid5 = fopen('repository.txt','w');
    fprintf(fid5,'%7.5f %7.5f %7.5f',rep_mat);
    fclose(fid5);

    tstart = tic;


    rng('shuffle');
    PopInit = unidrnd(ulim2);

    options = gaoptimset('InitialPopulation',PopInit,'PopulationSize',PopSize,'UseParallel','never','CrossoverFraction',0.8,...
        'StallGenLimit',9,'Generation',9,'TolFun',1e-6,'EliteCount',EliteSize,'Display','diagnose','Vectorized','off');

    [gbest,J,exflag,output] = ga(@(x)GA_Hybrid_Cost_082014(x,GMST0,lat_site,long_site,tstep,a_chief,dep_params,GEO_params,xmin,xmax,r_cyl,...
        el_val_shadow,ThreePassEnumData,rep_mat,max_ind,kinf),2,[],[],[],[],llim,ulim,[],[1,2],options);

    fid = fopen('GA_intermediate_vals.txt');
    x_inside = fscanf(fid,'%f',7);
    J_inside = fscanf(fid,'%d',1);
    k_tot = fscanf(fid,'%d',1);
    fclose(fid);
```

```matlab
fid_int = fopen('GA_opt_int.txt');
min_sat = fscanf(fid_int,'%i',1);
min_pass = fscanf(fid_int,'%i',1);
fclose(fid_int);

fid_iters = fopen('GA_iters.txt');
iters = fscanf(fid_iters,'%d');

J
J_inside
tend = toc(tstart)

load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
    Article_Data\Rev3\ThreeTarget\total_repGA');
load('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
    Article_Data\Rev3\ThreeTarget\rep_mat_out');

for ee = 1:length(a_chief)
    for ff = 1:maxP
        Jrep = rep_mat_out(ee,ff);
        Jtot = total_repGAinf(ee,ff);
        if Jrep < Jtot || Jtot == 0
            if Jtot ~= Inf
                total_repGAinf(ee,ff) = Jrep;
            end
        end
    end
end

save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\
    Article_Data\Rev3\ThreeTarget\total_repGA.mat','total_repGA');
```

```
    fid_fin = fopen('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\Article_Data\Rev3\ThreeTarget\ThreePassHybridGAData.txt','a');
    fprintf(fid_fin,'\r\n%2i\t %2i\t %2i\t %4.3f\t %6i\t %4.3f\t %4.3f\t %6i\t %4.3f\t %6i\t %7.5f\t %i\t %i\t %8.2f',h,min_sat,min_pass,x_inside(1),x_inside(2),x_inside(3),x_inside(4),x_inside(5),...
        x_inside(6),x_inside(7),J,output.generations,iters,tend);

    clear all

end
```

F.1.3.1 GA Cost Function

```
function [J,x_inside_dum,k_inside,rep_mat_out] = GA_Hybrid_Cost_062014(x,GMST0,lat_site,long_site,t_step,a_chief_vec,dep_params,GEO_params,xmin,xmax,r_cyl,...
    el_val_shadow,DataStruct,rep_mat,max_ind,kinf)


%===========================================================================
% This function evaluates the cost for the MATLAB genetic algorithm routine
%Inputs:
%   x: 2x1 vector of design variables
%       x(1) defines the satellite that will be shadowed
%       x(2) is the pass of x(1) or the specified ground site to accomplish
%       the shadow
%Outputs:
```

```
% Global Variables


%
    ==============================================================================

MU = 398600.5;

satellite = x(1);
min_ind = x(2);

[rows,cols] = size(rep_mat);

fid_rep = fopen('repository.txt');
rep_mat = fscanf(fid_rep,'%g', [rows cols]);
fclose(fid_rep);

if min_ind > max_ind(satellite)
    J = Inf;
    rep_mat(satellite,min_ind) = J;
    fid_rep = fopen('repository.txt','w');
    %number of elements must equal number of satellites
    fprintf(fid_rep,'%g %g %g ',rep_mat);
    fclose(fid_rep);
else
    %     if rep_mat(satellite,min_ind) == Inf
    %         J = Inf;
    %     else

    if rep_mat(satellite,min_ind) ~= 0
        J = rep_mat(satellite,min_ind);
    else
```

```
        C_times_c = DataStruct(satellite).times;
        C_ind_c = DataStruct(satellite).ind;
        Rijk_c = DataStruct(satellite).Rc;
        Vijk_c = DataStruct(satellite).Vc;
        rho_vec_cw_c = DataStruct(satellite).rho_c;
        Tvec_c = DataStruct(satellite).Tc;
        max_ind = DataStruct(satellite).max_ind;

        %Period of Chief satellite's orbit
        Pc = 2*pi*sqrt(a_chief_vec(satellite)^3/MU);



        t_enter = C_times_c(min_ind,1);
        t_exit = C_times_c(min_ind,2);
        t_zone = t_exit - t_enter;

        %determine indices of minimum duration contact
        C_ind_contact = C_ind_c(min_ind,:);

        %find unit vector pointing towards the deputy that puts chief
           between
        %ground site and deputy
        rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):C_ind_contact(2));

        %determine alpha and beta angles during contact times
        [alphavec,betavec] = alphabeta(rho_unit_cw);

        %Vector of times for propogation
        T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) - Tvec_c(
           C_ind_contact(1))*ones(length(Tvec_c(C_ind_contact(1):
           C_ind_contact(2))),1);
```

```matlab

        %Determine position/velocity vectors of chief satellite upon intial/final
        %contact
        chief_pos0 = Rijk_c(:,C_ind_c(min_ind,1));
        chief_vel0 = Vijk_c(:,C_ind_c(min_ind,1));
        chief_posf = Rijk_c(:,C_ind_c(min_ind,2));
        chief_velf = Vijk_c(:,C_ind_c(min_ind,2));

        if min_ind < max_ind
            max_coastf = C_times_c(min_ind+1,1) - C_times_c(min_ind,2);
        else
            max_coastf = DataStruct(satellite).max_coastf;
        end

        %time variables have precision to .1 second.  Others have
        %precision to 0.001 units (km,rad)
        prec2 = [2;0;3;3;0;2;0;6];

        [J,~,x_inside_dum,~,k_inside,~] = PSO_REL_SHADOW_DV4inf(7,[0 2*pi;1 C_times_c(min_ind,1);xmin xmax;xmin xmax;1 max_coastf;0 2*pi;1 16*3600],prec2,500,300,chief_pos0,chief_vel0,chief_posf,...
            chief_velf,dep_params,GEO_params,alphavec,betavec,t_zone,Pc,t_enter,t_exit,r_cyl,T_prop,GMST0,lat_site,long_site,t_step,el_val_shadow,kinf);

        %determine lowest cost so far
        fid1 = fopen('GA_Jmin.txt');
        Jmin = fscanf(fid1,'%f');
        fclose(fid1);

```

```
        %Update inside loop iterations
        fid4 = fopen('GA_iters.txt');
        iters = fscanf(fid4,'%d');
        iters = iters + k_inside;
        fclose(fid4);
        fid5 = fopen('GA_iters.txt','w');
        fprintf(fid5,'%i',iters);
        fclose(fid5);


        Jrep = rep_mat(satellite,min_ind);
        if Jrep == 0 || J < Jrep;
            rep_mat(satellite,min_ind) = J;
        end
        %update repository
        fid_rep = fopen('repository.txt','w');
        %number of elements must equal number of satellites
        fprintf(fid_rep,'%g %g %g ',rep_mat);
        fclose(fid_rep);

        %If current cost is better than lowest cost so far, update inner loop
        %variables
        if J < Jmin
            fid3 = fopen('GA_Jmin.txt','w');
            fprintf(fid3,'%g',J);
            fclose(fid3);

            fid6 = fopen('GA_opt_int.txt','w');
            fprintf(fid6,'%i %i',satellite,min_ind);
            fclose(fid6);

```

```
            fid2=fopen('GA_intermediate_vals.txt','w');
            fprintf(fid2,'%3.2f\t %i\t %4.3f\t %4.3f\t %i\t %3.2f\t %i\t %7.5f',x_inside_dum(1),x_inside_dum(2),x_inside_dum(3),x_inside_dum(4),x_inside_dum(5),x_inside_dum(6),x_inside_dum(7),J);
            fclose(fid2);

        end
    end
end


fid7 = fopen('GA_opt_int.txt');
min_sat = fscanf(fid7,'%i',1);
min_pass = fscanf(fid7,'%i',1);
fclose(fid7);

rep_mat_out = rep_mat;
save('C:\Users\Dan Showalter\Documents\MATLAB\PSO\Relative Motion\Article_Data\Rev2\ThreeTarget\rep_mat_out.mat','rep_mat_out');
```

F.2 Fifteen Target GTMEI

F.2.0.2 Large Outer Loop PSO

```
function [JGmin,Jpbest,gbest_tot,x,k,k_tot,JG,rep_mat,pop_mat] = PSO_LARGE_MULTISAT_COOP_WRAPPER(n,limits,prec,iter,swarm,GMST0,lat_site,long_site,t_step,a_chief_vec,dep_params,GEO_params,xmin,xmax,r_cyl,...
    el_val_shadow,max_ind,stall_lim,DataStruct,kinf)

%Author: Dan Showalter 23 Sep 2013

%Purpose: PSO inside of a PSO
```

```
%generic PSO inputs
%   n: # of design variables
%   limits: bounds on design variables (n x 2 vector) with first element
%       in row n being lower bound for element n and 2nd element in row n being
%       upper bound for element n
%   iter: number of iterations
%   swarm: swarm size
%   prec: defines the number of decimal places to keep for each design
%       variable and the cost function evalution size: (n+1,1)

%Problem specific PSO inputs
% GMST0 = initial Greenwich mean standard time (rad)
% lat_site = ground site latitude
% long_site = ground site longitude
% chief_params = vector (1x5) of fixed orbital elements of chief satellite
% nu_chief_vec = vector of potential initial true anomalies for chief
% dep_params = vector (1x6) of initial orbital elements of deputy satellite
% GEO_params = vector of (1x5) of fixed orbital elements of GEO satellite
% Coast_time_d = matrix (2xm) of allowed maneuver windows
%               (1,m) = start time of mth window
%               (2,m) = end time of mth window
% tf_max = maximum scenario time (sec)
% tstep = discrete time step (sec)
% xmin = minimum x distance from deputy to satellite in CW frame (km)
% xmax = maximum x distance from deputy to satellite in CW frame (km)
% Pc = period of chief satellite (sec)
% r_cyl = cylinder radius (km)
```

```
% ====================================================================================

%%

[N,~] = size(limits);
llim = limits(:,1);
ulim = limits(:,2);

if N~=n
    fprintf('Error! limits size does not match number of variables')
    stop
end

gbest = zeros(n,1);
x = zeros(n,swarm);
v = zeros(n,swarm);
pbest = zeros(n,swarm);
Jpbest = zeros(swarm,1);
x_inside = zeros(7,swarm);
d = (ulim - llim);
JG = zeros(iter,1);
J = zeros(swarm,1);
rep_mat = zeros(ulim(1),ulim(2));
pop_mat = struct('pop',zeros(n,swarm),'J',zeros(swarm,1),'gbest',zeros(n,1));

llim2 = ones(n,swarm);
ulim2 = ones(n,swarm);
% CoreNum = 12;
% if (matlabpool('size'))<=0
```

```
%     matlabpool('open','local',CoreNum);
% else
%     disp('Parallel Computing Enabled')
% end

parfor aa = 1:n
    llim2(aa,:) = llim(aa)*llim2(aa,:);
    ulim2(aa,:) = ulim(aa)*ulim2(aa,:);
end

d2 = ulim2 - llim2;

tstart = tic;

xrep(ulim(1),max_ind) = struct('xinsidevals',zeros(1,7));
%loop until maximum iteration have been met
for k = 1:iter
    t_inside = tic;
    %create particles dictated by swarm size input


    % if this is the first iteration
    if k == 1
        x = unidrnd(ulim2);
        v = random('unif',-d2,d2,[n,swarm]);

        %if this is after the first iteration, update velocity and
            position
        %of each particle in the swarm
    else

        for h = 1:swarm
```

```
c1 = 2.09;
c2 = 2.09;
phi = c1+c2;
ci = 2/abs(2-phi - sqrt(phi^2 - 4*phi));

cc = c1*random('unif',0,1);
cs = c2*random('unif',0,1);


vdum = v(:,h);

%update velocity
%             vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x(:,h)));

vdum = ci*(vdum + cc*(pbest(:,h) - x(:,h)) + cs*(gbest - x(:,h)));
%check to make sure velocity doesn't exceed max velocity for each
%variable
for w = 1:n

    %if the variable velocity is less than the min, set it to the min
    if vdum(w) < -d(w)
        vdum(w) = -d(w);
        %if the variable velocity is more than the max, set it to the max
    elseif vdum(w) > d(w);
        vdum(w) = d(w);
    end
```

```
            end

            v(:,h) = vdum;

            %update position
            xdum = x(:,h) + v(:,h);

            for r = 1:n

                %if particle has passed lower limit
                if xdum(r) < llim(r)
                    xdum(r) = llim(r);

                elseif xdum(r) > ulim(r)
                    xdum(r) = ulim(r);
                end

                x(:,h) = xdum;

            end

        end

    end

    % round variables to get finite precision
    for aa = 1:n
        x(aa,:) =  round(x(aa,:)*10^(prec(aa)))/10^prec(aa);
        v(aa,:) =  round(v(aa,:)*10^(prec(aa)))/10^prec(aa);
    end
    pop_mat(k).pop = x;
```

```
%% ***********************Cost Function
    ************************************

for m = 1:swarm
    MU = 398600.5;

    % ****************Cost function evaluation here
        ****************************
    opt_vars = x(:,m);
    %          variable definitions
    satellite = opt_vars(1);
    min_ind = opt_vars(2);

    C_times_c = DataStruct(satellite).times;
    C_ind_c = DataStruct(satellite).ind;
    Rijk_c = DataStruct(satellite).Rc;
    Vijk_c = DataStruct(satellite).Vc;
    rho_vec_cw_c = DataStruct(satellite).rho_c;
    Tvec_c = DataStruct(satellite).Tc;
    max_ind = DataStruct(satellite).max_ind;

    if min_ind > max_ind
        J(m) = Inf;
    else

        if rep_mat(satellite,min_ind) ~= 0
            J(m) = rep_mat(satellite,min_ind);
            x_inside(:,m) = xrep(satellite,min_ind).xinsidevals;
        else



```

```
        %Period of Chief satellite's orbit
        Pc = 2*pi*sqrt(a_chief_vec(satellite)^3/MU);


        t_enter = C_times_c(min_ind,1);
        t_exit = C_times_c(min_ind,2);
        t_zone = t_exit - t_enter;

        %determine indices of minimum duration contact
        C_ind_contact = C_ind_c(min_ind,:);

        %find unit vector pointing towards the deputy that puts
            chief between
        %ground site and deputy
        rho_unit_cw = rho_vec_cw_c(:,C_ind_contact(1):
            C_ind_contact(2));

        %determine alpha and beta angles during contact times
        [alphavec,betavec] = alphabeta(rho_unit_cw);

        %Vector of times for propogation
        T_prop = Tvec_c(C_ind_contact(1):C_ind_contact(2)) -
            Tvec_c(C_ind_contact(1))*ones(length(Tvec_c(
            C_ind_contact(1):C_ind_contact(2))),1);

        %Determine position/velocity vectors of chief satellite
            upon intial/final
        %contact
        chief_pos0 = Rijk_c(:,C_ind_c(min_ind,1));
        chief_vel0 = Vijk_c(:,C_ind_c(min_ind,1));
        chief_posf = Rijk_c(:,C_ind_c(min_ind,2));
        chief_velf = Vijk_c(:,C_ind_c(min_ind,2));
```

```
if min_ind < max_ind
    max_coastf = C_times_c(min_ind+1,1) - C_times_c(
        min_ind,2);
else
    max_coastf = DataStruct(satellite).max_coastf;
end

%time variables have precision to .1 second.  Others
   have
%precision to 0.001 units (km,rad)
prec2 = [2;0;3;3;0;2;0;6];

x(:,m);

[J(m),~,x_inside_dum,~,k_inside,~] =
   PSO_REL_SHADOW_DV4inf(7,[0 2*pi;1 C_times_c(min_ind
   ,1);xmin xmax;xmin xmax;1 max_coastf;0 2*pi;1
   16*3600],prec2,500,300,chief_pos0,chief_vel0,
   chief_posf,...
    chief_velf,dep_params,GEO_params,alphavec,betavec,
        t_zone,Pc,t_enter,t_exit,r_cyl,T_prop,GMST0,
        lat_site,long_site,t_step,el_val_shadow,kinf);
if k == 1 || rep_mat(satellite,min_ind) == 0
    rep_mat(satellite,min_ind) = J(m);
    xrep(satellite,min_ind).xinsidevals = x_inside_dum;
else
    if J(m) < rep_mat(satellite,min_ind)
        rep_mat(satellite,min_ind) = J(m);
    end
end
J(m);
```

```
                x_inside(:,m) = x_inside_dum;
                out_loop = m;
                if k == 1
                    k_tot = k_inside;
                else
                    k_tot = k_inside + k_tot;
                end
            end
        end
    end

    [minJ,ind_minJ] = min(J);
    x_inside(:,ind_minJ);
    %% *************************Constraint Equations *************************
    %% **********************************************************************

    %%


    if k == 1

        Jpbest(1:swarm) = J(1:swarm);
        pbest(:,1:swarm) = x(:,1:swarm);

        [Jgbest,IND] = min(Jpbest(:));

        gbest(:) = x(:,IND);
        g_inside_best = x_inside(:,IND);
        stall = 0;

```

```
    else

        for h=1:swarm
            if J(h) < Jpbest(h)
                Jpbest(h) = J(h);
                pbest(:,h) = x(:,h);
                if Jpbest(h) < Jgbest

                    Jgbest = Jpbest(h);
                    gbest(:) = x(:,h);
                    g_inside_best = x_inside(:,h);

                end
            end
        end


    end

    count = 0;

    for y = 1:swarm

        diff = Jgbest - Jpbest(y);

        if abs(diff)<10^(-prec(n+1)+1)
            count = count+1;
        end

    end

    %round cost to nearest precision required
```

```
    J = round(J*10^prec(n+1))/10^prec(n+1);
    pop_mat(k).J = J;
    pop_mat(k).gbest = gbest;
    JG(k) = Jgbest;
    JGmin = Jgbest;

    if k > 1
        if (JG(k) - JG(k-1)) == 0
            stall = stall + 1;
        else
            stall = 0;
        end
    end

    if count == swarm
        break
    end

    if stall == stall_lim
        break
    end
    tend = toc(tstart);

    iter_complete = k
    iter_time = toc(t_inside)
    format long g
    gbest
    g_inside_best
    JGmin
end

gbest_tot(1:n) = gbest;
```

```
gbest_tot(n+1:n+length(g_inside_best)) = g_inside_best;
```

Bibliography

[1] Office of the President of the United States, *National Space Policy of the United States of America*, June 2010, http://www.space.commerce.gov/general/nationalspacepolicy/, Accessed 8 February, 2013.

[2] Department of Defense & Office of the Director of National intelligence, *National Security Space Strategy*, January 2011, http://www.defense.gov/home/features/2011/0111_nsss/docs/NationalSecuritySpaceStrategyUnclassifiedSummary_Jan2011.pdf, Accessed 8 February, 2013.

[3] Department of Defense, *Quadrennial Defense Review*, March 2014, http://www.defense.gov/pubs/2014_Quadrennial_Defense_Review.pdf, Accessed 21 March, 2014.

[4] Department of Defense & Office of the Director of National intelligence, *Fact Sheet: Resilience of Space Capabilities*, January 2011, http://www.defense.gov/home/features/2011/0111_nsss/docs/DoD$%$20Fact$%$20Sheet$%$20-$%$20Resilience.pdf, Accessed 8 February, 2013.

[5] National Aeronautics & Space Administration, *Advanced Space Transportation Program: Paving the Highway to Space*, 2014, http://www.nasa.gov/centers/marshall/news/background/facts/astp.html_prt.htm, Accessed 10 June, 2014.

[6] Department of Defense Defense Science Board, *Task Force Report: The Role Autonomy of in DoD Systems*, July 2012, http://www.dtic.mil/get-tr-doc/pdf?AD=ADA566864, (Accessed 8 February 2013).

[7] Conway, B. A., "A Survey of Methods Available for the Numerical Optimization of Continuous Dynamic Systems," *Journal of Optimization Theory and Applications*, Vol. 152, pp. 271–306.

[8] Abdelkhalik, O. and Gad, A., "Dynamic-Size Multiple Populations Genetic Algorithm for Multigravity-Assist Trajectories Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 2, March-April 2012, pp. 520–529.

[9] Gad, A. and Abdelkhalik, O., "Hidden Genes Genetic Algorithm for Multi-Gravity-Assist Trajectories Optimization," *Journal of Spacecraft and Rockets*, Vol. 48, No. 4, July-August 2011, pp. 629–641.

[10] Abdelkhalik, O., "Hidden Genes Genetic Optimization for Variable-Size Design Space Problems," *Journal of Optimization Theory and Applications*, Vol. 156, No. 2, August 2012, pp. 450–468.

[11] Wagner, S., Kaplinger, B., and Wie, B., "GPU Accelerated Genetic Algorithm for Multiple Gravity-Assist and Impulsive ΔV Maneuvers," *AIAA/AAS Astrodynamics Specialist Conference*, AIAA, Reston, VA, 2012.

[12] Vasile, M., "A Behavioral-Based Meta-Heuristic for Robust Global Trajectory Optimization," *Proceedings of the IEEE World Congress on Evolutionary Computation*, IEEE, Piscataway, NJ, 2007, pp. 2056–2063.

[13] Vasile, M. and De Pascale, P., "On the Preliminary Design of Multiple Gravity-Assist Trajectories," *Journal of Spacecraft and Rockets*, Vol. 43, No. 4, July - Aug 2006, pp. 794–805.

[14] Ceriotti, M. and Vasile, M., "Automated Multigravity Assist Trajectory Planning with a Modified Ant Colony Algorithm," *Journal of Aerospace Computing, Information, and Communication*, Vol. 7, No. 9, September 2010, pp. 261–293.

[15] Luo, Z., Dai, G., and Peng, L., "A Novel Model for the Optimization of Interplanetary Trajectory Using Evolutionary Algorithm," *Journal of Computers*, Vol. 6, No. 10, 2011, pp. 2243–2248.

[16] Vinkó, T., Izzo, D., and Bombardelli, C., "Benchmarking Different Global Optimisation Techniques for Preliminary Trajectory Design," *58th International Astronautical Congress*, Vol. 6, International Astronautical Federation, Paris, France, 2007, pp. 4181–4190.

[17] Bessette, C. and Spencer, D., "Identifying Optimal Interplanetary Trajectories Through a Genetic Approach," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, AIAA, Reston, VA, 2006.

[18] Olds, A., K. C. A. and Cupples, M., "Interplanetary Mission Design Using Differential Evolution," *Journal of Spacecraft and Rockets*, Vol. 44, No. 5, Sep - Oct 2006, pp. 1060–1070.

[19] Zhu, K., Li, J., and Baoyin, H., "Multi-Swingby Optimization of Mission to Saturn Using Global Optimization Algorithms," *Acta Mechanica Sinica*, Vol. 25, No. 6, 2009, pp. 839–845.

[20] Vavrina, M. and Howell, K., "Global Low-Thrust Trajectory Optimization Through Hybridization of a Genetic Algorithm and a Direct Method," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, AIAA, Reston, VA, 2008.

[21] Sentinella, M. R., "Comparison and Integrated Use of Differential Evolution and Genetic Algorithms for Space Trajectory Optimisation," *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Publications, Piscataway, NJ, 2007, pp. 973–978.

[22] Sentinella, M. R. and Casalino, L., "Hybrid Evolutionary Algorithm for the Optimization of Interplanetary Trajectories," *Journal of Spacecraft and Rockets*, Vol. 46, No. 2, March-April 2009, pp. 365–372.

[23] Subbarao, K. and Shippey, B. M., "Hybrid Genetic Algorithm Collocation Method for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 4, July-August 2009, pp. 1396–1403.

[24] Wuerl, A., Crain, T., and Braden, E., "Genetic Algorithm and Calculus of Variations-Based Trajectory Optimization Technique," *Journal of Spacecraft and Rockets*, Vol. 40, No. 6, November-December 2003, pp. 882–888.

[25] Pontani, M. and Conway, B. A., "Particle Swarm Optimization Applied to Space Trajectories," *Journal of Guidance, Control and Dynamics*, Vol. 33, No. 5, Sep - Oct 2010, pp. 1429 – 1441.

[26] Luo, Y.-Z., Tang, G.-J., and Li, H.-y., "Optimization of Multiple-Impulse Minimum-Time Rendezvous with Impulse Constraints Using a Hybrid Genetic Algorithm," *Aerospace Science and Technology*, Vol. 10, No. 6, 2006, pp. 534–540.

[27] Stupik, J., Pontani, M., and Conway, B., "Optimal Pursuit/Evasion Spacecraft Trajectories in the Hill Reference Frame," *AIAA/AAS Astrodynamics Specialist Conference*, AIAA, Reston, VA, 2012.

[28] Conway, B., Chilan, C., and Wall, B., "Evolutionary Principles Applied to Mission Planning Problems," *Celestial Mechanics and Dynamical Astronomy*, Vol. 97, No. 2, 2007, pp. 73–86.

[29] Wall, B. and Conway, B., "Genetic Algorithms Applied to the Solution of Hybrid Optimal Control Problems in Astrodynamics," *Journal of Global Optimization*, Vol. 44, No. 4, 2009, pp. 493–508.

[30] Englander, J. A., Conway, B. A., and Williams, T., "Automated Mission Planning via Evolutionary Algorithms," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 6, 2012, pp. 1878–1887.

[31] Englander, J. A., Conway, B. A., and Williams, T., "Automated Interplanetary Trajectory Planning," *2012 AIAA/AAS Astrodynamics Specialist Conference*, AIAA, Reston, VA, 2012.

[32] Abdelkhalik, O. and Mortari, D., "Orbit Design for Ground Surveillance Using Genetic Algorithms," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 5, September-October 2006, pp. 1231–1235.

[33] Kim, H.-D., Jung, O.-C., and Bang, H., "A Computational Approach to Reduce the Revisit Time Using a Genetic Algorithm," *International Conference on Control, Automation and Systems*, IEEE Publications, Piscataway, 2007, pp. 184–189.

[34] Guelman, M. and Kogan, A., "Electric Propulsion for Remote Sensing from Low Orbits," *Journal of Guidance, Control and Dynamics*, Vol. 22, No. 2, March-April 1999, pp. 313–321.

[35] Co, T. C., Zagaris, C., and Black, J., "Responsive Satellites Through Ground Track Manipulation Using Existing Technology," *Journal of Spacecraft and Rockets*, Vol. 50, No. 1, January-February 2013, pp. 206–216.

[36] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, Microcosm Press and Springer, Hawthorne, CA, 3rd ed., 2007, Co-published by Springer.

[37] Bate, R. R., Mueller, D. D., and White, J. E., *Fundamentals of Astrodynamics*, Dover Publications, New York, 1971.

[38] Sellers, J. J., Kirkpatrick, D., and Shute, A., *Understanding Space: An Introduction to Astronautics, Third Edition*, "McGraw Hill Companies", New York, 2005.

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

[40] Battin, R. H., *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*, American Institute of Aeronautics and Astronautics, 1999.

[41] Oldenhuis, R., "Robust Solver for Lambert's Orbital Boundary Value Problem," Jan. 2010, http://www.mathworks.com/matlabcentral/fileexchange/26348-robust-solver-for-lambert-s-orbital-boundary-value-problem, Accessed October 19, 2012.

[42] Wall, B. and Conway, B., "Shape-Based Approach to Low-Thrust Rendezvous Trajectory Design," *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 1, 2009, pp. 95–101.

[43] Wall, B., "Shape-Based Approximation Method for Low-Thrust Trajectory Optimization," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, AIAA, Reston, VA, 2008, pp. 441–448.

[44] Hill, G. W., "Researches in Lunar Theory," *American Journal of Mathematics*, Vol. 1, 1878, pp. 5–26.

[45] Clohessy, W. and Wiltshire, R., "Terminal Guidance System for Satellite Rendezvous," *Journal of Aerospace Sciences*, Vol. 27, No. 9, 1960, pp. 653–658.

[46] Irvin, D. J., Cobb, R. G., and Lovell, T. A., "Fuel-Optimal Maneuvers for Constrained Relative Satellite Orbits," *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 3, 2009, pp. 960–972.

[47] Betts, J. T., "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, March - April 1998, pp. 193 – 207.

[48] Garg, D., Patterson, M., Hager, W., Rao, A., Benson, D., and Huntington, G., "A Unified Framework for the Numerical Solution of Optimal Control Problems Using Pseudospectral Methods." *Automatica*, Vol. 46, No. 11, 2010, pp. 1843–1851.

[49] Garg, D., Hager, W. W., and Rao, A. V., "Pseudospectral Methods for Solving Infinite-Horizon Optimal Control Problems," *Automatica*, Vol. 47, No. 4, 2011, pp. 829–837.

[50] Patterson, M. A. and Rao, A., "Exploiting Sparsity in Direct Collocation Pseudospectral Methods for Solving Optimal Control Problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, 2012, pp. 364–377.

[51] Conway, B. A., "The Problem of Spacecraft Trajectory Optimization," *Spacecraft Trajectory Optimization*, Cambridge University Press, Cambridge, England, 2010, pp. 1–15.

[52] Eberhart, R. and Kennedy, J., "A New Optimizer Using Particle Swarm Theory," *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, IEEE Publications, Piscataway, NJ, 1995, pp. 39–43.

[53] Kennedy, J. and Eberhart, R., "Particle Swarm Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 4, IEEE Publications, Piscataway, NJ, 1995, pp. 1942–1948.

[54] Eberhart, R. and Shi, Y., "Particle Swarm Optimization: Developments, Applications and Resources," *Proceedings of the IEEE Congress on Evolutionary Computation*, Piscataway, NJ, 2001, pp. 81–86.

[55] Shi, Y. and Eberhart, R., "A Modified Particle Swarm Optimizer," *Proceedings of the IEEE International Conference on Evolutionary Computation*, IEEE Publications, Piscataway, NJ, 1998, pp. 69–73.

[56] Clerc, M., "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," *Proceedings of the IEEE Congress on Evolutionary Computation*, Vol. 3, IEEE Publications, Piscataway, NJ, 1999, pp. 1951–1957.

[57] Clerc, M. and Kennedy, J., "The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 1, February 2002, pp. 58–73.

[58] Eberhart, R. and Shi, Y., "Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization," *Proceedings of the IEEE Congress on Evolutionary Computation*, Vol. 1, IEEE Publications, Piscataway, NJ, 2000, pp. 84–88.

[59] Trelea, I. C., "The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection," *Information Processing Letters*, Vol. 85, 2003, pp. 317–325.

[60] ZHANG, L.-p., YU, H.-j., and HU, S.-x., "Optimal Choice of Parameters for Particle Swarm Optimization," *Journal of Zhejiang University SCIENCE*, Vol. 6A, No. 6, 2005, pp. 528–534.

[61] Holland, J. H., *Adaption in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, "University of Michigan Press", 1975.

[62] Talbi, E.-G., *Metaheuristics: From Design to Implementation*, John Wiley and Sons, 2009.

[63] Zhong, J., Hu, X., and Zhang, J., "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms," *International Conference for Modeling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, Vol. 2, Vienna, Austria, 2005, pp. 1115–1121.

[64] Thierens, D. and Goldberg, D., "Convergence Models of Genetic Algorithm Selection Schemes," *Parallel Problem Solving from Nature*, Vol. 866, 1994, pp. 119–129.

[65] Dorigo, M., *Optimization, Learning and Natural Algorithms (in Italian)*, Ph.d. thesis, Dipartimento di Ellettronica, Politecico di Milano, Italy.

[66] Dorigo, M., M. V. and Colorni, A., "Positive Feedback as a Search Strategy," Tech. Rep. 91-016, Dipartimento di Ellettronica, Politecico di Milano, Italy, Milan, Italy, 1991.

[67] Dorigo, M., M. V. and Colorni, A., "Ant System: Optimization by a Colony of Cooperating Agents," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 26, No. 1, pp. 29–41.

[68] Dorigo, M., M. V. and Colorni, A., "Ant Colony Optimization Theory: A Survey," *Theoretical Computer Science*, Vol. 344, pp. 243–278.

[69] Dorigo, M. and Stutzle, T., "The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances." *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2002.

[70] Parsopoulos, K. E. and Vrahatis, Michael, N., "Particle Swarm Optimization Method for Constrained Optimization Problems," *In Proceedings of the Euro-International Symposium on Computational Intelligence*, IOS Press, Amsterdam, Netherlands, 2002, pp. 214–220.

[71] Ray, T. and Liew, K., "A Swarm with an Effective Information Sharing Mechanism for Unconstrained and Constrained Single Objective Optimization Problems," *IEEE Congress on Evolutionary Computation*, Vol. 1, IEEE Publications, Piscataway, NJ, 2001, pp. 75–80.

[72] Michalewicz, Z. and Schoenauer, M., "Evolutionary Algorithms for Constrained Parameter Optimization Problems," *Evolutionary Computation*, Vol. 4, No. 1, 1996, pp. 1–32.

[73] Schoenauer, M. and Michalewicz, Z., "Evolutionary Computation at the Edge of Feasibility," *Proceedings of the 4th Conference on Parallel Problems Solving from Nature*, Vol. 1141, Springer-Berlin Heidleberg, Berlin, Germany, 1996, pp. 245–254.

[74] Michalewicz, Z., Dasguota, D., Le Riche, R. G., and Schoenauer, M., "Evolutionary Algorithms for Constrained Engineering Problems," *Computers and Industrial Engineering*, Vol. 30, No. 4, 1996, pp. 851–870.

[75] Joines, J. A. and Houck, C. R., "On the use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with Genetic Algorithms," *Proceedings of the IEEE World Conference on Computational Intelligence*, IEEE Publications, Piscataway, NJ, 1994, pp. 579–584.

[76] Michalewicz, Z., "Genetic Algorithms, Numerical Optimization, and Constraints," *Proceedings of Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Francisco, CA, 1995, pp. 151–158.

[77] Bean, J. C. and Hadj-Alouane, A. B., "A Dual Genetic Algorithm for Bounded Integer Programs," Tech. Rep. 92-53, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1992.

[78] Hadj-Alouane, A. B. and Bean, J. C., "A Genetic Algorithm for the Multiple-Choice Integer Program," Tech. Rep. 92-50, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1992.

[79] Homaifar, A., Qi, C. X., and Lai, S. H., "Constrained Optimization Via Genetic Algorithms," *Simulation*, Vol. 62, No. 4, 1994, pp. 242–251.

[80] Coello Coello, C. A., "Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art," *Computer Methods in Applied Mechanics and Engineering*, Vol. 191, No. 11-12, January 2002, pp. 1245 – 1287.

[81] Pontani, M. and Conway, B. A., "Particle Swarm Optimization Applied to Impulsive Orbital Transfers," *Acta Astronautica*, Vol. 74, May - Jun 2012, pp. 141 – 155.

[82] Pontani, M., "Simple Method to Determine Globally Optimal Orbital Transfers," *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 3, May-June 2009, pp. 899–914.

[83] Pontani, M., Ghosh, P., and Conway, B., "Particle Swarm Optimization of Multiple-Burn Rendezvous Trajectories," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 4, 2012, pp. 1192–1207.

[84] Pontani, M. and Conway, B. A., "Swarming Theory Applied to Space Trajectory Optimization," *Spacecraft Trajectory Optimization*, Cambridge University Press, Cambridge, England, 2010, pp. 263–293.

[85] Ghosh, P. and Conway, B., "Numerical Trajectory Optimization with Swarm Intelligence and Dynamic Assignment of Solution Structure," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 4, 2012, pp. 1178–1191.

[86] Branicky, M. and Mitter, S., "Algorithms for Optimal Hybrid Control," *Proceedings of the 34th IEEE Conference on Decision and Control*, Vol. 3, IEEE Publications, Piscataway, NJ, Dec 1995, pp. 2661–2666.

[87] Chilan, C. and Conway, B., "Using Genetic Algorithms for the Construction of a Space Mission Automaton," *IEEE Congress on Evolutionary Computation*, 2009, pp. 2316–2323.

[88] Chilan, C. M. and Conway, B. A., "Automated Design of Multiphase Space Missions Using Hybrid Optimal Control," *Journal of Guidance, Control, and Dynamics*, 2013, pp. 1–15.

[89] Prussing, J. E. and Chiu, J.-H., "Optimal Multiple-Impulse Time-Fixed Rendezvous Between Circular Orbits," *Journal of Guidance, Control, and Dynamics*, Vol. 9, No. 1, 1986, pp. 17–22.

[90] Colasurdo, G. and Pastrone, D. G., "Indirect Optimization Method for Impulsive Transfers," *AIAA/AAS Astrodynamics Conference*, 1994, pp. 441–448.

[91] Yu, J., Chen, X.-q., Chen, L.-h., and Hao, D., "Optimal Scheduling of GEO Debris Removing Based on Hybrid Optimal Control Theory," *Acta Astronautica*, Vol. 93, 2014, pp. 400 – 409.

[92] Kazarlis, S. and Petridis, V., "Varying Fitness Functions in Genetic Algorithms: Studying the Rate of Increase of the Dynamic Penalty Terms," *Proceedings of the* 5^{th} *International Conference on Parallel Problem Solving From Nature*, Springer Berlin Heidelberg, Berlin, Germany, 1998, pp. 211–220.

[93] Hu, X., Eberhart, R., and Shi, Y., "Engineering Optimization with Particle Swarm," *Proceedings of the IEEE Swarm Intelligence Symposium*, IEEE Publishing, Piscataway, NJ, 2003, pp. 53–57.

[94] Showalter, D. and Black, J. T., "Responsive Theater Maneuvers via Particle Swarm Optimization," *Journal of Spacecraft and Rockets*, 2014, accessed 20 June, 2014. doi: http://arc.aiaa.org/doi/abs/10.2514/1.A32989.

[95] Englander, J., Conway, B., and Wall, B., "Optimal Strategies Found Using Genetic Algorithms for Deflecting Hazardous Near-Earth Objects," *IEEE Congress on Evolutionary Computation*, May 2009, pp. 2309–2315.

[96] Li, S., Zhu, Y., and Wang, Y., "Rapid Design and Optimization of Low-Thrust Rendezvous/Interception Trajectory for Asteroid Deflection Missions," *Advances in Space Research*, Vol. 53, No. 4, 2014, pp. 696 – 707.

[97] Hu, X., Shi, Y., and Eberhart, R., "Recent Advances in Particle Swarm," *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE Publications, Piscataway, NJ, 2004, pp. 90–97.

[98] Patterson, M. A. and Rao, A. V., "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming," *ACM Transactions on Mathematical Software*, Vol. 39, No. 3, 2013.

[99] Garg, D., Patterson, M. A., Francolin, C., Darby, C. L., Huntington, G. T., Hager, W. W., and Rao, A. V., "Direct Trajectory Optimization and Costate Estimation of Finite-Horizon and Infinite-Horizon Optimal Control Problems Using a Radau pseudospectral Method," *Computational Optimization and Applications*, Vol. 49, No. 2, 2011, pp. 335–358.

[100] Wächter, A. and Biegler, L., "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, Vol. 106, No. 1, 2006, pp. 25–57.

[101] Young, M. and Van Allen, R., "Propulsion Subsystems I - Propulsion," *Space Mission Engineering: The New SMAD*, edited by R. Wertz, James, D. F. Everett, and J. J. Puschell, chap. 18, Microcosm Press, 2011, p. 535.

[102] Anflo, K. and Möllerberg, R., "Flight Demonstration of New Thruster and Green Propellant Technology on the PRISMA Satellite," *Acta Astronautica*, Vol. 65, No. 9, 2009, pp. 1238–1249.

[103] Ziegler, D. P., *Persistent Space Situational Awareness: Distributed Real-Time Awareness Global Network in Space (DRAGNETS)*, Center for Strategy and Technology, Air War College, Maxwell AFB, AL, 2007.

[104] Tirpak, J. A., "Securing the Space Arena," *Air Force Magazine*, Vol. 87, No. 4, 2004.

[105] Yates, J., Spanbauer, B., and Black, J., "Geostationary Orbit Development and Evaluation for Space Situational Awareness," *Acta Astronautica*, Vol. 81, No. 1, 2012, pp. 256–272.

[106] Hope, A. and Trask, A., "Pulsed Thrust Method for Hover Formation Flying," *AIAA/AAS Astrodynamics Specialist Conference Meeting*, AIAA, Reston, VA, 2003.

[107] Sanchez, S., "NOLHdesigns spreadsheet," 2011, Accessed 18 April, 2013.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
15–09–2014	Doctoral Dissertation	Oct 2011–Sep 2014

4. TITLE AND SUBTITLE
Optimal Autonomous Spacecraft Resiliency Maneuvers Using Metaheuristics

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Showalter, Daniel J., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENY-DS-14-S-29

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES
This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
The growing congestion in space has increased the need for spacecraft to develop resilience capabilities in response to natural and man-made hazards. Equipping satellites with increased maneuvering capability has the potential to enhance resilience by altering their arrival conditions as they enter potentially hazardous regions. The propellant expenditure corresponding to increased maneuverability requires these maneuvers be optimized to minimize fuel expenditure and to the extent which resiliency can be preserved. This research introduces maneuvers to enhance resiliency and investigates the viability of metaheuristics to enable their autonomous optimization. Techniques are developed to optimize impulsive and continuous-thrust resiliency maneuvers. The results demonstrate that impulsive and low-thrust resiliency maneuvers require only meters per second of delta-velocity. Additionally, bi-level evolutionary algorithms are explored in the optimization of resiliency maneuvers which require a maneuvering spacecraft to perform an inspection of one of several target satellites while en-route to geostationary orbit. The methods developed are shown to consistently produce optimal and near-optimal results for the problems investigated and can be applied to future classes of resiliency maneuvers yet to be defined. Results indicate that the inspection requires an increase of only five percent of the propellant needed to transfer from low Earth orbit to geostationary orbit. The maneuvers and optimization techniques developed throughout this dissertation demonstrate the viability of the autonomous optimization of spacecraft resiliency maneuvers and can be utilized to optimize future classes of resiliency maneuvers.

15. SUBJECT TERMS
Optimization, Metaheuristics, Astrodynamics

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON / 19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	486	Dr. Jonathan T. Black (ENY) (937) 255-3636 x4578 jonathan.black@afit.edu